NTNU
Norwegian University of
Science and Technology

Master's thesis
2019
Daniel Strømme Solberg

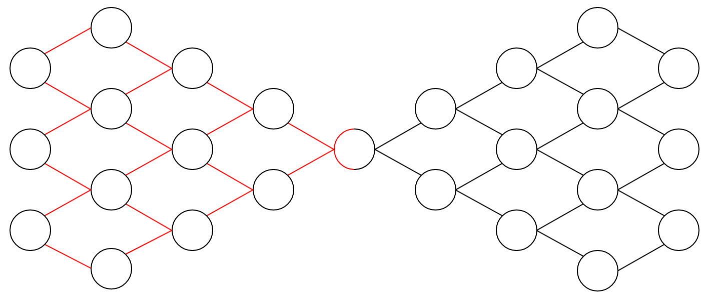
Master's thesis

NTNU
Norwegian University of
Science and Technology
Faculty of Information Technology and Electrical
Engineering
Department of Computer Science

Daniel Strømme Solberg

Neural Machine Translation using the RNNPB model with PB binding

July 2019



NTNU
Norwegian University of
Science and Technology

NTNU
Norwegian University of
Science and Technology

Neural Machine Translation using the RNNPB model with PB binding

Daniel Strømme Solberg

Computer Science
Submission date: July 2019
Supervisor: Keith L. Downing

Norwegian University of Science and Technology
Department of Computer Science

Abstract

This thesis considers the Recurrent Neural Network with Parametric Biases (RNNPB) model and applies it to the task of machine translation. The RNNPB model allows encoding multiple sequences in a single RNN by associating each sequence with a particular activation of the parametric bias nodes, which are placed in the input layer. Essentially, the parametric biases can be considered as sequence embeddings. Thus, a bidirectional mapping is achieved where parametric biases map to the complete sequences, as well as the inverse mapping where parametric biases are computed from a provided sequence.

One of the applications of this model has been with binding together the parametric biases of two separate models. By coupling the sequence representations of the two models, sequence to sequence mapping can be achieved. The concept is similar to the Encoder-Decoder model which has enjoyed much success in the Neural Machine Translation field. In order to make a translation, first a representation of the source sentence is computed in the encoder, and secondly, the decoder decodes this representation into the target sequence.

Traditionally, the RNNPB model has been used in robotic applications to represent motor programs, and the application in machine translation is therefore new. One of the interesting properties of this work, which essentially entails binding two language models together, is that the two models can theoretically benefit from monolingual data, which is a desirable trait in the NMT field. The present work takes a simple model, traditionally used in small-scale experiments, and refurnishes it with modern machine learning advances, such as LSTM layers and regularization, and applies it to large-scale tasks. Through experiments with autoencoding in English, the present work shows that the RNNPB model can model language and learn to autoencode on a large scale. However, for experiments with translation from English to German, the present work is not able to achieve high translation performance.

Preface

This thesis was written during the spring semester of the author's fifth year of a computer science master's degree. The degree includes two introductory years and a three-year specialization in artificial intelligence. The thesis was completed at the Gløshaugen campus of Norway's University of Science and Technology (NTNU), with the guidance of Professor Keith L. Downing.

Daniel Strømme Solberg
Trondheim, July 17, 2019

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of Neural Machine Translation (NMT) has enjoyed much progress the last few years. State-of-the-art translation systems have gradually moved from typical statistical approaches to machine learning solutions. Machine translation is particularly fruitful as a machine learning field due to several important qualities. There exists an abundance of both parallel, bilingual data and monolingual data to train the models on. The results can be evaluated easily by humans, as well as objectively using common performance metrics. Furthermore, the task is a member of a much broader category of sequence-to-sequence learning tasks, and as such, solutions often lend themselves easily to other tasks in this category. It is therefore unsurprising that important machine learning advances have arisen in this field, such as the Encoder-Decoder architecture and attention mechanisms.

This thesis examines the effectiveness of a particular model for the task of machine translation: the Recurrent Neural Network with Parametric Biases (RNNPB). The next section expands on this application. In the remainder of this first chapter, research goals and questions are stated, the research method utilized is described, contributions are summarized and an outline is given for the thesis.

1.1 Background and Motivation

Recurrent Neural Networks (RNNs) are neural networks used to process sequential data, such as time series. Contrary to feed forward networks, recurrent networks can take advantage of state information. The state arises from feedback loops in the network architecture and is simply defined as the activation of these feedback connections from the preceding round of inputs. This allows information to be carried forward from one round to the next. Typically, the network starts with some initial state and then processes a sequence of inputs, each of which augments the state. As such, the output at any given step depends not only on the current input, but also all preceding inputs in the sequence. If a new sequence should be processed, the state can simply be reset. RNNs have recently been used in state-of-the-art systems, in areas such as machine translation, handwriting recognition and natural language processing [Lipton, 2015].

A common use case for the RNN is *generative tasks*, where the network takes some input vector $x^{(t)}$ and outputs its prediction of the vector at the next time step, $x^{(t+1)}$. These architectures enable learning a sequence and then making predictions for future values. If the network is allowed to operate in a *closed-loop* mode, predictions far into the future can be made. In this mode, the output prediction from the network at one time step is fed back in as the input for the next time step. By repeating this process for several time steps, a look-ahead prediction of an arbitrary number of steps can be generated.

One such generative task of importance is language modeling. A language model often estimates the probability distribution of the words in a vocabulary, conditioned on a partial sentence. It can for instance predict the continuation of a partial sentence by repeatedly appending the current most likely word. This method of text generation has been used for novel tasks, such as generating text in the style of Shakespeare [Karpathy, 2015], but also closely relates to how neural machine translation is usually achieved.

In the approach suggested in this thesis, two language models are used to achieve translation. The models target two different languages and together they allow for bidirectional translation. This is achieved by linking the models together, so that the representations of corresponding sentence-pairs are bound. Both of the language models are implemented as RNNPB models, which provide the particular mechanism used for binding the representations.

The suggested approach has several advantages, the most important of which is that monolingual data can be utilized inherently. While a parallel corpus is required to facilitate the binding of representations, monolingual data can be used to train the language accuracy of each model individually. This mitigates a problem which is well-recognized within the field of neural machine translation: while models typically excel with language pairs for which a vast parallel corpus exists, most language pairs do not have such a vast parallel corpus. Practically every language, however, has a vast amount of available monolingual data.

The RNNPB model is an RNN with multiple parametric bias (PB) nodes in the input layer, proposed in Tani [2002]. Here, parametric means that the values of these bias nodes are exchangeable and will be given as an extra input to the model, as opposed to simply having fixed values after training. The model learns to associate each sequence with a particular set of PB values through training. This facilitates two main operations: generation and recognition. Generation denotes the forward operation where a sequence is generated from a vector of PB values, i.e.:

$$p \rightarrow x_1, x_2, ..., x_N \tag{1.1}$$

Here, p is a real-valued vector with a fixed dimensionality corresponding to the number of PB nodes. Recognition denotes the inverse operation, where PB values are computed from a given sequence, i.e.:

$$x_1, x_2, ..., x_N \rightarrow p \tag{1.2}$$

By binding the PB nodes of two models, generation and recognition can be used in conjunction to achieve translation. To translate from a source language to a target language, recognition can be used for the source model to compute the PB values, p, which can then be used in the target model to generate a corresponding sentence. This can be characterized as an Encoder-Decoder approach, as proposed by Sutskever et al. [2014]. In this framework, the encoder computes a fixed-size representation of the source sentence, which the decoder decompresses into the predicted translation. The architecture is however very different from the one suggested in Sutskever et al. [2014], as reflected by the ability of the two RNNPB models to function interchangeably as both encoder and decoder, enabling bilingual translation, as well as the ability to utilize monolingual corpora.

Sugita and Tani [2005] introduce the idea of binding two RNNPB models together, using the PB Binding Method. They use the approach to perform sequence-to-sequence learning, in the same manner as the current work, although the two domains which are bound are different. In the work of Sugita and Tani, the domains are language and behavior. As such, they are able to provide a sentence to a physical robot and have it generate the corresponding behavior. The authors show that this is possible even when a sentence is not learned; the model is able to generalize to understand unlearned mappings. Effectively, they're able to command the robot using natural language.

The work done by Sugita and Tani shows that the suggested approach is indeed able to model language and do sequence-to-sequence translation, but the scale of the stated results is rather small. The language model that they employ uses a vocabulary of nine words and sentences consisting of only two words. It is therefore of interest to ascertain the effectiveness of the model in a larger-scale task such as machine translation, which typically involves tens of thousands of words and millions of sentences. Should the model prove to scale well, it could offer a valuable contribution to machine translation for low-resource language pairs. This summarizes the goal of the current thesis. The experiments done here permit themselves to take advantage of recent advances in deep learning and language modeling and thus aim to apply the RNNPB model to a task of much larger scale.

1.2 Goals and Research Questions

The rest of this thesis is written with the following goals and research questions in mind. The main goal of this research can be stated as follows:

Goal Apply the RNNPB model to the task of machine translation on a large scale.

In trying to reach this goal, this thesis will carry out experiments in order to determine whether this application of the RNNPB is feasible. Along the way, whether the feasibility of the model for this task is affirmed or refuted, this thesis aims to contribute to advancing the work with the RNNPB model by developing the model further. Ideally, this work also contributes to the field of machine translation, but this goal is secondary as the choice of model is made ahead of time and is not based on the maximum likelihood of advancing machine translation. The work can therefore be said to be model-centric rather than task-centric.

Subsequent chapters will attempt to answer the following research questions:

Research question 1 Can the RNNPB model be used to learn a large-scale language model?

Since the suggested translation scheme involves binding two language models, one for each of the two languages, successful translation is contingent on the language models being successful separately. Here, a successful language model means that the model can *represent* sentences in its respective language. Specifically, this means that the model must be able to compute PB values which corresponds to a provided sentence after training, and particularly, it must be able to do this for sentences that do not occur in the training set. Existing work has already proved that this is achievable with small-scale language tasks, but for a translation task the capability must be asserted on a larger scale.

Research question 2 Can the proposed method achieve accurate machine translation?

This research question examines whether PB binding of two separately working language models yields an accurate translation model. In order to evaluate the accuracy, the BLEU metric will be calculated and presented, and compared to baseline models. The BLEU metric is well-established within the field of machine translation, allowing for accuracy comparisons across different works. Still, a baseline model is often used to ensure fair comparisons under the exact same conditions.[1]

[1]Some factors that may impact results are training time and hardware, preprocessing steps and choice of data.

Research question 3 Can the Bound RNNPB model additionally benefit from training on monolingual data, for the task of machine translation?

With this question, the current work seeks to answer whether the RNNPB model indeed possesses this desired trait. From a technical perspective, it is apparent that the RNNPB models can be trained additionally on monolingual data. However, since these sentences are not bound between the models, in contrast with sentence pairs from the parallel corpus, it is not known whether using these additional sentences are detrimental or beneficial for translation. Experiments are carried out to answer this question.

1.3 Research Method

The research method applied in this thesis is the empirical method. Experiments are designed and carried out in order to yield quantitative results, which will provide answers to the research questions. This choice is sensible given the suggested approach; the model itself is already tried and tested, and here, it is merely applied to a different well-defined task, at a larger scale. The task of machine translation is also one where the results can be relatively easily evaluated and judged, which further encourages an empirical method. The research can be said to be technique-driven rather than problem-driven; the primary focus of the research is applying the RNNPB model to a suitable task, not necessarily finding the best solution to the problem of machine translation.

1.4 Contributions

The main contributions of this thesis are mainly regarding applications with the RNNPB model. The key contribution is that the experiments here show that the RNNPB can model language well, and that it can model autoencoding as a sequence-to-sequence task. Further contributions can be stated as:

1. The original RNNPB model, which used a simple Jordan RNN architecture, is updated with LSTM recurrence, and the learning capability of this model is ascertained through experiments.

2. The experiments show that the proposed model is able to represent language very well. That is, the model is able to compute PB vectors corresponding to both seen (trained) and unseen sentences, and achieves a high score when the sentences are regenerated in comparison with the original sentences.

3. The concept of hard binding is proposed as an alternative to the soft binding originally used by Sugita and Tani, and the experiments show hard binding gives superior results for this task.
4. The model's capability to generalize is increased significantly by using the regularization techniques dropout and intermittent PB reset, where the latter is a technique that is proposed here.
5. The bound RNNPB model is shown to be capable of functioning as an encoder-decoder system and achieves a decent score for autoencoding.

1.5 Thesis Structure

The remainder of this thesis proceeds as follows. Chapter 2 first introduces the relevant background theory in its first section. This chapter mainly concerns itself with recurrent neural networks and language modeling. Chapter 3 puts the current work in context with regard to related work. The chapter first presents a Structured Literature Review (SLR) protocol, which defines the scope of the related work. The related work is then presented in section 3.2 and 3.3, where the former reviews machine translation literature and the latter reviews literature relating to the RNNPB model. In chapter 4, more details are given regarding the RNNPB model and the specific architecture implemented here. Chapter 5 applies the proposed model with four experiments, where the first two focus on autoencoding and the latter two on translation. Lastly, chapter 6 offers an evaluation of the results and a discussion of the work.

Chapter 2

Background Theory

The RNNPB model, as well as many state-of-the-art Neural Machine Translation (NMT) models, are based on recurrent neural networks. This chapter therefore starts by introducing these thoroughly. Some familiarity with neural networks and machine learning techniques is presumed, but no such presumptions are made regarding recurrent neural networks. In section 2.2, a few important RNN architectures are described. The relevant background on language models is presented in subsection 2.3, before this chapter concludes with a definition of two evaluation metrics for machine translation, BLEU and perplexity.

2.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) describe an alternative to conventional feed-forward neural networks, with the addition of feedback loops in the network architecture. Such feedback loops are not permitted in feed-forward networks, which must satisfy the property that all nodes can be arranged into layers, so that no nodes are dependent on nodes in the same or lower layers. For RNNs, the feedback loops are used to provide nodes in the network with information about previous inputs, thus making the network stateful and endowing it with the ability to remember its history. It is this feature that makes recurrent nets suitable for processing *sequential* data, i.e. sequences of points that are not independent [Lipton, 2015].

For the forward propagation, the computation differs very little from that of feed-forward networks. One input is presented to the network at a time, and computation proceeds in a feed-forward manner. The only difference from feed-forward networks is that some neurons will also depend on the current network state. The neurons which have inbound recurrent arcs receive the relevant activation from the previous round of computation, and typically multiplies it with some weight. As a result, the neuron's output will depend on the state of the network. It is the activation of the neurons with *outbound* recurrent arcs that is referred to as the state of the recurrent network. The state of a given recurrent network has fixed dimensionality and thus cannot grow to accommodate a larger history even if needed.

A simple recurrent network is shown in figure 2.1. The forward propagation of this simple network can be computed according to the following equations:

$$h_t = a_h(w_1 x_t + w_2 h_{t-1}) \tag{2.1}$$

$$o_t = a_o(w_3 h_t) \tag{2.2}$$

Here, there's a total of three weights, w_i, one for each of the three network arcs. a_h and a_o are activation functions for the hidden unit and output unit, respectively. A common choice for a_h is the tanh activation function, while a_o can for instance be chosen as the softmax function for classification tasks. x_t describes the input at time t; for instance the $t'th$ word in a sentence. The initial state, h_0, must be decided upon; often it is simply 0. The equations generalize easily to a network with multiple nodes in each layer, as depicted in figure 2.2. Here, the scalar weights, w_i, can simply be replaced with weight matrices, U, V and W:

$$h_t = a_h(U x_t + V h_{t-1}) \tag{2.3}$$

$$o_t = a_o(W h_t) \tag{2.4}$$

The backpropagation step for RNNs differs considerably more compared to for feed-forward networks. The increased complexity occurs because not only must the network learn how to take the state into account, which standard backpropagation achieves inherently, it must also learn how the state should be augmented for each iteration. Intuitively, the network must learn to compress the most important parts of the history into the state. The nodes with outbound recurrent arcs are responsible for this task, and thus, their activations affect not only the current output, but also future outputs. Therefore, given that the training loss is computed depending on these outputs, the gradient calculation must typically consider network activations from all earlier time steps.

An RNN can be described as a dynamical system,

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}; \theta) \tag{2.5}$$

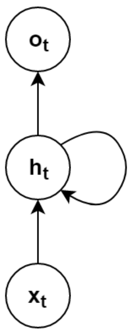


Figure 2.1: A simple RNN with one input unit, one recurrent hidden unit and one output unit. Since RNNs are used for sequences, the units are subscripted with t to denote time.

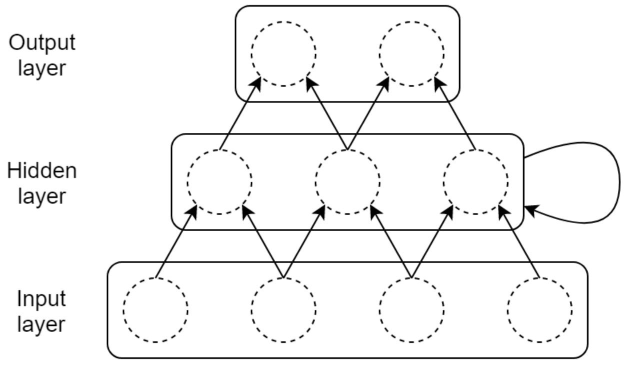


Figure 2.2: An RNN with multiple units in each layer. Here, the recurrence is shown from the hidden layer to itself and denotes that the recurrent connections are fully-connected within the hidden layer. The different layers are also typically fully-connected.

where $s^{(t)}$ denotes the state of the system at time t, $x^{(t)}$ the input to the system at time t and θ the parameters associated with the system. The literature often describes the properties of RNNs in terms of classic dynamical systems theory. In this framework, training the network can be regarded as learning to follow the intended trajectory through state space [Jordan, 1997]. Typically, these trajectories are *attractors*; in the presence of noise, the trajectories attract nearby states, ensuring that the system still transitions from state to state in the intended order, as given by the learned sequences.

Importantly, the dynamical system *recurrence* in equation 2.5 can be removed through recurrent expansion of the equation, so that $s^{(t)}$ can be computed from some initial state $s^{(0)}$ and inputs $x^{(1)}, x^{(2)}, ..., x^{(t)}$. This expansion is equivalent to *unrolling* a recurrent neural network, which is the procedure that enables backpropagation. It is used to expand the network through time so that backpropagation can be carried out the same way as for standard feed-forward networks. This procedure is referred to as Backpropagation Through Time and is explained more thoroughly in the next subsection.

2.1.1 Backpropagation Through Time

Figure 2.3 shows the process of unrolling a computational graph that's used in the Backpropagation Through Time (BPTT) algorithm. The figure shows how feedback loops are eliminated from the graph, so that both forward propagation and backpropagation can be carried out in the conventional manner, utilizing some provided initial hidden state, h_0. Rumelhart et al. [1986] provide details regarding how the weight changes can be calculated in a straight-forward manner for all steps in time. Namely, given an error signal, the effects of the weights, W, on the error at each time step are calculated, which give rise to one weight change, ΔW, per time step. Since the recurrent weights are shared over time steps in this unrolled graph, all the weight changes are summed before applying the total weight change at completion.

The BPTT algorithm computes the gradients by first doing a forward-pass through the unrolled graph, followed by a backwards-pass, exactly as the standard backpropagation algorithm. A notable difference is, however, the effect of time. For BPTT, the size of the computational graph grows linearly in terms of the length of the sequence. The activation values for each time step of all nodes must be saved in the forward pass. Thus, the BPTT algorithm has both time and space complexity of $O(T)$ for a sequence of length T [Goodfellow et al., 2016]. It is also noteworthy that this computation cannot be computed in parallel, due to the dependencies on prior hidden states in the forward-pass.

Furthermore, under these conditions, some of the gradients will be computed as a long chain of products, which make them vulnerable to the *vanishing gra-*

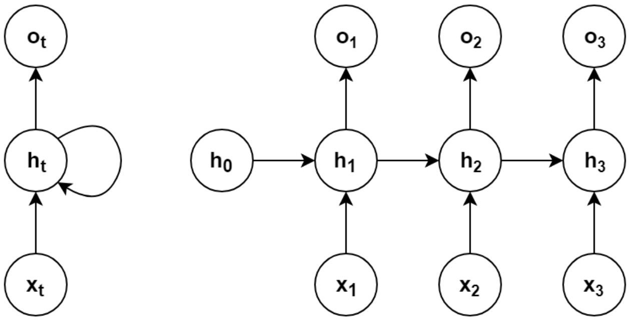


Figure 2.3: Unrolling a computation graph, here with a sequence of three inputs, x_1, x_2, x_3, before unrolling and after unrolling on the left and right respectively. Note that h_0 must be given.

dient or *exploding gradient* problems. In essence, gradients may vanish when the weights are less than one in magnitude, and explode when larger than one. The problem is particularly prevalent with recurrent neural networks compared to deep feed-forward networks due to weight sharing across time steps. Instead of the gradient calculation consisting of a chain of *different* factors, as for feed-forward networks, the chain will include several duplicates of the same weight, due to all network weights being identical across time steps. Given an expression w_i^N where N is an exponent that increases with the length of the sequence, the expression converges to zero if $abs(w_i) < 1$ and diverges if $abs(w_i) > 1$. Two common strategies that mitigate these problems are Truncated BPTT and teacher forcing.

The Truncated BPTT (TBPTT) algorithm is a small and commonly used adaptation of BPTT where sequences are divided into smaller chunks, to be computed separately. The strategy used is to compute a finite, fixed number of time steps for each inference and training step. This fragmentation does not directly affect the inference procedure, as the final state of a given round can be used as the initial state for the next computation. One can picture computing a finite, unrolled computational graph, and then "wrapping back around" to the beginning. It does, however, affect the training procedure; gradients cannot propagate across different "chunks", and the ability to learn long-term dependencies may thus be sacrificed. The number of time steps in each chunk must be chosen carefully, so that all the time-dependencies that should be learned are contained within the size of one single chunk. For example, if the sequence is divided into chunks consisting of 10 time steps, and a given output is dependent on information from 11 or more time steps back, the network cannot learn to include this information in the state.

2.1.2 Teacher forcing

The concept of teacher forcing offers an alternative approach altogether to the BPTT algorithm, but the methods are also commonly combined in practice. Teacher forcing is applicable whenever an RNN contains feedback loops from the outputs back into the network. If these feedback loops are the only recurrent loops, the different time steps of the unrolled computational graph can be decoupled completely. The output for a given time step can be substituted with the ground truth value for that time step as the input to the next step. With complete teacher forcing, this means that gradients do not propagate over time, as the effect of the weights of the last time step on the current hidden state is removed; i.e.:

$$\frac{\partial H(t)}{\partial W(t-i)} = 0 \qquad \forall i = 1, 2, ..., t-1 \tag{2.6}$$

where $W(t)$ denote the same weight W at different points in time. With teacher forcing, all time steps can be computed in parallel and learning can thus be considerably sped up.

For generative tasks[1], teacher forcing is often used in conjunction with BPTT. These typically employ feedback-loops in the hidden layer, while, at the same time, they may need to feed their outputs back as inputs. The output-to-input recurrence is then often removed during training by using teacher forcing. Bengio et al. [2015] justify this decision and show that training without teacher forcing may yield slow convergence, model instability and poor accuracy. On the other hand, Goodfellow et al. note that teacher forcing may be problematic whenever the network will be used in a closed-loop mode (during inference), with its own outputs fed back in as inputs, because it will be sensitive to errors in the outputs of earlier time steps. In this situation, small errors tend to propagate and amplify quickly. The RNN may derail from the patterns it has learned and may easily find itself in a part of state space that it has never encountered during training.

The balance between these factors depend on the learning task at hand. Two relevant factors that affect the choice of whether to use teacher forcing is the accumulation of noise and propagation of gradients from the outputs. These

[1]I.e. tasks that aim to predict x_t given $x_{t-1}, x_{t-2}, ...x_1$.

factors imply that recurrent *classification* learning tasks are more suitable for teacher forcing than recurrent *regression* learning tasks.

In regression tasks, the outputs are real values, that is, quantities. They can for instance be actuator parameters if the task is to learn a motor program. When the network must be used in the closed-loop mode during inference, and noisy outputs are fed back in as the next inputs, the noise will likely accumulate over time. Training without teacher forcing may thus be useful to condition the network to manage this noise. Furthermore, feeding back the outputs can be regarded as yet another recurrent arc, and can be treated as such during the BPTT procedure. Therefore, foregoing teacher forcing enables the network to take into account how the output at a given step affects future outputs as well.

In classification tasks, such as language modeling, the outputs are categorical values, e.g. words or characters. The output can for instance be chosen as the most likely class given the prediction of the network. If this output is fed back in as input for the next step – in the case of no teacher forcing – gradients cannot propagate over this recurrent arc, because extracting the most likely class is a non-differentiable operation. Therefore, the network is not able to adjust its outputs based on how it affects its future predictions, which implies that training without teacher forcing is less effective. Furthermore, noise is removed when a discrete value is chosen from the probability outputs of the network, and therefore does not accumulate over multiple time steps, which implies that training without teacher forcing is not as necessary. For example, say that the most likely class has only 5 % certainty. Then, by letting the next input be merely this word itself, the lack of confidence is not propagated to the future. Teacher forcing therefore typically works well for categorical tasks and is nearly always used with language modeling.

This trade-off is well recognized in the literature, and work has been done to find suitable compromises between the two learning approaches. Bengio et al. [2015] propose making the decision of whether to use teacher forcing on a per-batch basis. The choice is made stochastically according to a schedule where the likelihood decreases as a function of the training iteration. Thus, training starts in a strongly forced manner, before gradually switching to closed-loop training. The authors argue that the scheme provides the benefits of training without teacher forcing, without increasing training time or sacrificing training stability.

2.2 RNN architectures

RNNs are a rich family of architectures. The Jordan-type and Elman-type RNNs are two architectures that differ in how the feedback loops are implemented. These two variations are discussed in the next two subsections. Furthermore, different machine learning tasks require different, more specific architectures.

The specific architectures to be used for a given task depends on which inputs the system should accept and which outputs it should emit. Tasks such as machine translation requires accepting a sequence of inputs (words in the source language) and then emitting another sequence of outputs (the translation). The RNNPB on the other hand, requires an architecture that accepts a single vector x as input and then emits a sequence of vectors $y^{(1)}, y^{(2)}, ..., y^{(n)}$ as outputs. Some of the most typical variations are elaborated on at the end of this section. For completeness, the next section first presents how non-recurrent, feed-forward neural networks can, to some degree, be made context-aware, using a Time-Delay Neural Network.

2.2.1 Time-Delay Neural Networks

The feed-forward network has proved to be immensely powerful as a machine learning tool. One of its most significant assets is its ability to self-organize raw inputs into meaningful representations, in a hierarchical manner [Lipton, 2015]. Thus, whereas other machine learning models require the features to be extracted from the raw data prior to learning, neural networks can extract these features automatically as an inherent part of its learning procedure. It is therefore interesting to consider if simple feed-forward networks can be directly applied to sequences as well, with only minor modifications.

Originally, the feed-forward network assumes that the inputs are independent: no explicit state is maintained from one run of the network to the next, and the system can be described by a deterministic mapping $Y = f(X)$. This doesn't however mean that it's impossible for the feed-forward network to learn a sequence consisting of dependent inputs. Indeed, the network is *theoretically* capable of learning any sequence where a point $x^{(t)}$ is a strict function of the preceding point $x^{(t-1)}$. [2] After all, these are deterministic mappings, $x^{(t-1)} \rightarrow x^{(t)}$, or equivalently, $Y = f(X)$. The problem with this idea becomes apparent when two distinct points, $x^{(t_1)}$ and $x^{(t_2)}$, can be succeeded by the same point, i.e. $x^{(t_1+1)} = x^{(t_2+1)}$. In terms of dynamical systems theory, the states are said to be *degenerate*; a single state can be preceded by two or more distinct states. In Tani and Ito [2003], this issue is referred to in terms of sensory aliasing, and illustrated with the problem of drawing a figure eight: proceeding from the middle point, a longer history of states must be known to determine the next position[3]. The feed-forward network described here uses a history, or *context*, window of size one. This leads naturally to the generalization to larger context windows: the network can trivially be adapted to take an arbitrary context size, L, into account and learn the mapping $X^{(t-L)}, X^{(t-L+1)}, ..., X^{(t-1)} \rightarrow X^{(t)}$. The idea of using a feed-forward net with a context window summarizes the concept of Time-Delay

[2]Although in practice it would likely be beneficial to leverage a larger context of inputs.

[3]Two or more states could suffice.

nets.

While Time-Delay nets successfully incorporate history, they are generally inferior to RNNs. One drawback of the architecture is that the size of the input layer is directly proportional to the context size. Therefore, the number of weights that needs to be trained increases with the size of the context window. In terms of language modeling, it is evident that, combinatorially, the number of different (possibly malformed) sentences increases exponentially with the sentence length, which implies that the necessary data required to learn correct sentences also increases exponentially. This makes large context windows impractical for time-delay nets. RNNs, on the other hand, utilizes weight-sharing across time steps, and are therefore much more easily trained. Instead of requiring additional neurons for processing long sentences, it simply reuses the ones it has. According to Goodfellow et al., this feature allows RNNs to generalize to new sequence lengths and capture statistical properties across different positions in time.

2.2.2 The Jordan RNN

The Jordan RNN is a variant where the feedback loops are implemented from the output layer, as shown in figure 2.4 [Jordan, 1997]. Compared to other implementations, such as the Elman RNN, this is a constraint on the expressive power of the network. The output nodes must serve a dual function: both emit the output of the system and update the network's state. The network is appropriate whenever the current output is a function of the current input and all previous outputs. It can, however, not consider previous inputs, contrary to the Elman RNN. Both the Jordan RNN and Elman RNN have been employed for a wide set of tasks.

In the Jordan RNN, the output units feed back into a *context layer*[4] of units. The context units are implemented with self-loops, thus enabling the network to remember *all* previous outputs, as opposed to merely the immediately preceding output.

Jordan proposes that the context units be updated using a simple exponential average over earlier outputs,

$$c_{n+1} = \mu c_n + o_n \tag{2.7}$$

as opposed to a learnable, weighted average,

$$c_{n+1} = U c_n + V o_n \tag{2.8}$$

Here, c_n denotes the context layer and o_n the output layer values; μ is a exponential average coefficient and U and V are network weights. Jordan argues that

[4]Jordan refers to the layer as a *state layer* while Elman uses the term *context layer*. Here, context layer is used to describe both types of stateful layers.

more immediately preceding outputs are more important than outputs further away in time, which is reflected in the nature of an exponential average. Naturally, this approach assumes that vectors close in output space are similar. This doesn't necessarily hold for e.g. classification tasks, which often utilize sparse, one-hot output values. Using the learnable, weighted average in equation 2.8 is more flexible and more common today. The latter approach densely connects the output layer to the context layer, and the context layer to itself, while the former approach uses sparse connections as shown in figure 2.4.

The Jordan RNN is often informally used to describe an architecture in which the self-loops in the context layer are omitted. In this variant only the immediately preceding output is provided as the state. Thus, this architecture is less expressive than the original Jordan RNN. As Goodfellow et al. notes, unless the output layer is very high-dimensional, information about the past will usually be lost. When the network is used for an appropriate problem, however, it can be trained more efficiently with parallel training over time steps, using strict teacher forcing instead of the BPTT algorithm, as discussed in 2.1.2. This is not possible with the original Jordan RNN, because the context layer is dependent on all previous inputs, requiring that the time steps be computed sequentially.

2.2.3 The Elman RNN

Whereas the Jordan RNN's feedback loops originate from the output layer, Elman RNNs implement the feedback loop from the hidden layer back to itself [Elman, 1990]. In figure 2.5, this is illustrated by copying the hidden layer activations into the context layer to be used for the next time step. Normally, the context layer is densely connected to the hidden layer; or equivalently, the hidden layer is densely connected to itself. Elman RNNs are more expressive than Jordan RNNs, and have been proved to be Turing complete [Goodfellow et al., 2016]. While Jordan RNNs implement a mechanism to remember earlier outputs, the Elman RNN can embed rich, important historical information in the hidden activation, and will at each time step map the hidden activation to the current output. Importantly, the network can utilize all previous inputs for the current output. The higher degree of expressiveness comes at the cost of being harder to train however, and the BPTT algorithm must always be used for Elman RNNs. Still, the Elman-type RNN recurrence (from the hidden layer) is the de facto choice today.

2.2.4 Input-output relationships

Several architectures and strategies exist for implementing RNNs with different input and output characteristics. Some possible choices for the inputs include no inputs at all, a single vector as input and a sequence of vectors. Likewise, the

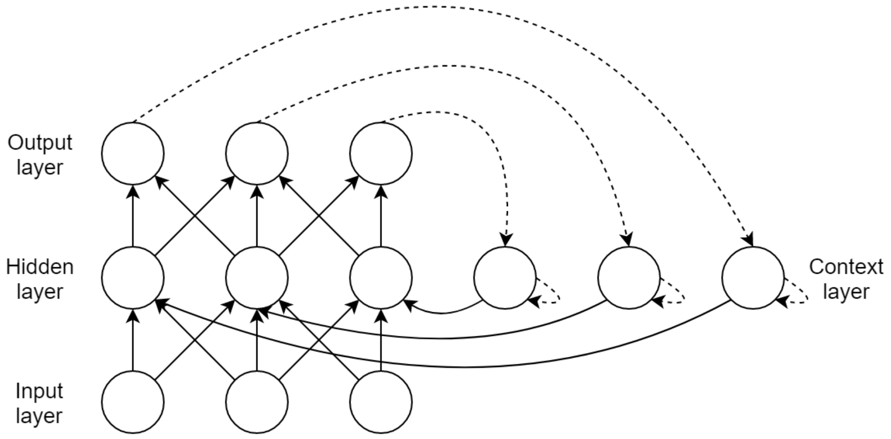


Figure 2.4: The Jordan RNN. The dashed lines indicate arcs going to the next time step. For the Jordan RNN, the context layer is updated as a function of the previous network outputs and their own previous values (notice the self-loops). The goal is to give the network the means to remember its previous outputs. Noticeably, the Jordan RNN is not able to remember previous inputs or hidden layer activations. The figure uses sparse layers only for readability.

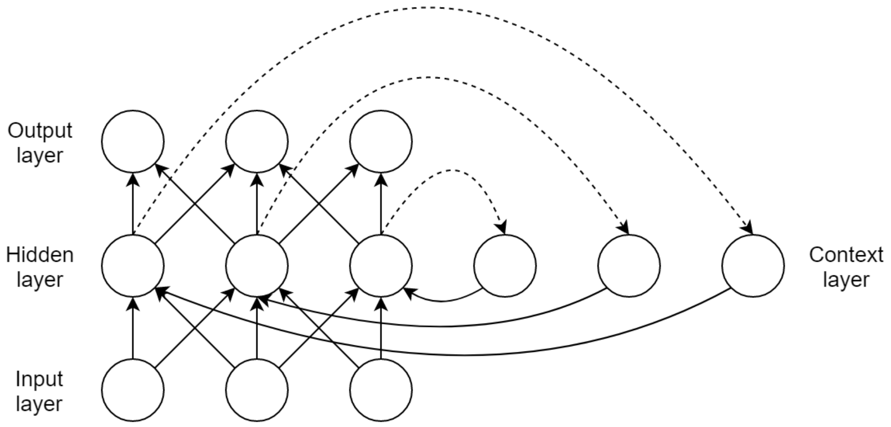
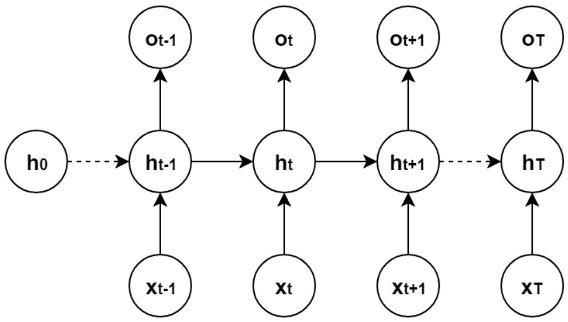
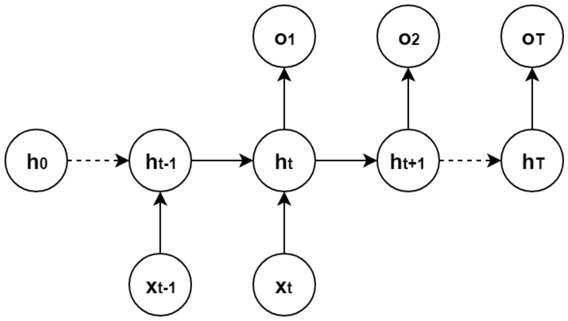


Figure 2.5: The Elman RNN. The dashed lines indicate arcs going to the next time step, and here the hidden layer values are simply copied without modification to the context units. The network is considered to be more expressive than the Jordan RNN, because the basis for the context is the hidden units, which are not constrained in the same manner as the output units.

RNN can output a single vector, sequences of vectors, and even variable-length sequences of vectors. Some of the most relevant variations are discussed below.



(a)



(b)

Figure 2.6: Two many-to-many types of RNN architectures, unrolled over time. a) For each input that is presented, an output is immediately generated. b) All inputs are presented first, and then a sequence of outputs are generated.

Many to many The many to many architecture denotes the case when both the inputs and the outputs are sequences. Two examples of this type are shown in figure 2.6. Figure 2.6-a shows perhaps the most commonly used architecture: given a sequence of inputs x_t, produce an output sequence y_t of the same length. The architecture here implies a causal relation between the inputs and outputs where an output y_t only depends on the inputs at the same time or preceding it, $x_1, x_2, ..., x_t$. This is often a realistic assumption: a common task is to predict the continuation of a sequence. For such tasks, the future is not available for inquiry.

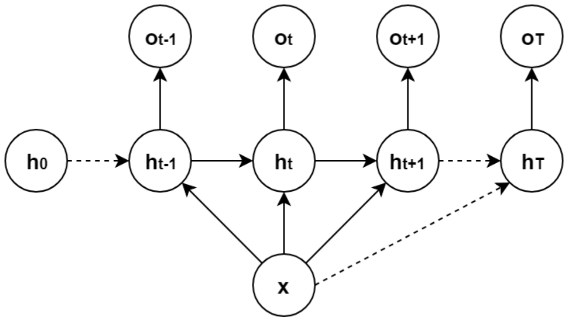
One task for which this assumption does not hold is machine translation. When translating a sentence, the entirety of the sentence must be taken into account; one cannot simply greedily translate from the beginning [5]. For this task, the architecture shown in figure 2.6-b may be appropriate. This architecture has been used for precisely this task, with the recurrent cells replaced with LSTM cells [Sutskever et al., 2014]. In this case, the first part of the graph learns to *encode* the sentence in the origin language into some fixed-size representation, and the last part of the graph learns to *decode* the representation into the target language.

One to many The one-to-many type of RNN denotes the case where the input is a single vector and the outputs a sequence of vectors. Figure 2.6-a shows an architecture where the input vector is duplicated across all time steps. This strategy can for example be used for image captioning, where the input vector is a representation of the image and the output caption consists of a sequence of words [Goodfellow et al., 2016]. Another common strategy is to utilize no inputs, and use the input vector, x, as the initial hidden state. The approaches can also be combined.

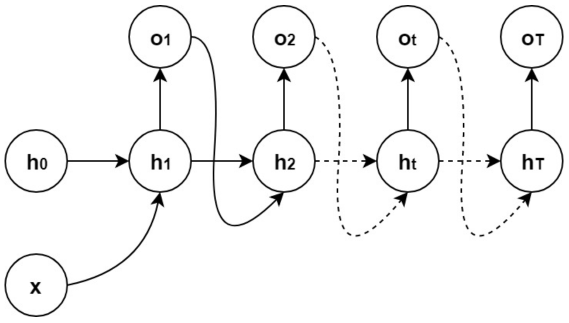
The architecture in figure 2.7-b shows how a standard many-to-many architecture can be adapted for the one-to-many case. A single input is given to start off the computation, which then proceeds in closed loop mode. This is often how language models are implemented; a special `<start>` token is used to kick off generation, after which predictions are fed back as inputs. Note that these two architectures, shown in figure 2.6-a and figure 2.7-b, can be seen as equivalent RNNs differing only in whether teacher forcing is applied or not. That is, figure 2.7-b shows the computational graph of a generative RNN, which with teacher forcing can be trained as the many-to-many architecture in figure 2.6-a. As section 2.1.2 explains, one can choose either approach for learning the RNN, or a combination.

Many to one The last case that will be described is the case with a sequence of inputs and a single output vector. This type of RNN is applicable for classification tasks: given a set of sequences, learn to classify them into categories. Figure 2.8 shows one such approach where the final state is either taken directly as the output classification or mapped to it.

[5] As an example, consider how, in German, operative verbs often occur at the end of a sentence, while this is not the case in English.



(a)



(b)

Figure 2.7: Two one-to-many types of RNN architectures, unrolled over time. a) A single input vector x is provided as input at each time step. b) An adaptation of the many-to-many architecture in figure 2.6-a where an initial vector x is used as the first input, and then the computation proceeds in closed-loop mode.

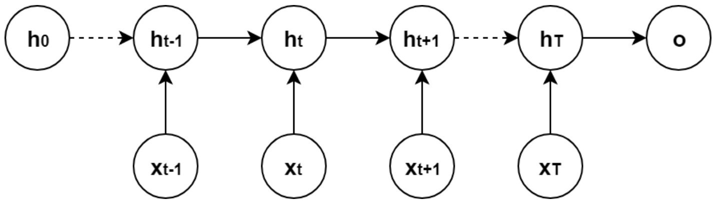


Figure 2.8: A many-to-one type of RNN, unrolled over time. In this architecture the outputs are removed altogether, and the outputs can be computed as a function of the final hidden state.

2.2.5 Training difficulties and the LSTM

While RNNs have been proven to be Turing complete in theory and should therefore be able to calculate anything a computer can calculate, they have been notoriously difficult to train in practice. The size of the unrolled graph corresponding to a long sequence simply has difficult properties, among them shared weights and repeated non-linearities. As briefly discussed in section 2.1.1, two problems often arise: vanishing gradients and exploding gradients. Both problems affect the gradients exponentially as the sequence length grows large. Exploding gradients describe how gradients can grow extremely large over large time spans, making learning unstable. Vanishing gradients describe the opposite problem, where the gradients become exponentially small, making the network unable to capture long-term dependencies.

The exploding gradient problem can normally be managed rather well. Pascanu et al. [2013] propose gradient clipping which has proven essential in training RNNs. The idea is rather simple: if the gradient exceeds a threshold value, clip it so that it equals the threshold. Specifically, it is the aggregated gradient (computed as the sum of each time-step gradient), g, that is clipped. Let τ denote the threshold, then use the clipping rule:

$$g = \begin{cases} \tau \frac{g}{\|g\|}, & \text{if } \|g\| > \tau \\ g, & \text{otherwise} \end{cases} \tag{2.9}$$

One of the most significant advances of the RNN architecture is the Long Short-Term Memory (LSTM) architecture [Hochreiter and Schmidhuber, 1997], which offers a solution to the vanishing gradient problem. Hochreiter and Schmidhuber show that the LSTM is much more capable of learning long-term dependencies. In an LSTM model, the basic recurrent units are substituted with memory cells, see figure 2.9. The memory cell implements a node, referred to as the *internal state*, which has a self-loop with weight fixed at 1. This self-loop is referred to as the *constant error carousel*: since it always has weight equal to 1, it assures that gradients can propagate across time steps without vanishing. In addition, the memory cell includes *gates* to control its state and its output. The *input gate* determines which part of the inputs to the cell will be added to the cell state, and the *output gate* determines whether the cell will emit an output. Effectively, the cell is able to ignore certain inputs, and learn when it should emit an output. If both these gates are closed, the cell state remains unchanged and does not affect the network's output; the cell is effectively dormant. Additionally, a *forget gate* is often included and determines if the state should be forgotten.

The LSTM is particularly desirable because it's easily combined with existing recurrent architectures. One can simply exchange the fundamental recurrent unit, without making any large further changes, and reap the benefits of the LSTM.

For most state-of-the-art work, the LSTM or other gated RNNs have become the standard.

2.3 Language models

A language model assigns a probability to a sequence of words. Such models play an important role in numerous fields that deal with language. Consider for instance machine translation, speech recognition or handwriting recognition: when a set of possible candidate sentences are considered, it's useful to have a model that evaluates the likelihood of each sentence occurring in the language. Intuitively, malformed sentences should be assigned a low likelihood, while sentences that frequently occur[6] should be assigned a higher likelihood. Developing an accurate language model is therefore an important task that has received much attention.

Let the probability distribution over sentences in a given language be described as $P(w_1^T)$, where w_1^T denotes a sequence of words, $w_1, w_2, ..., w_T$. Then, it is often useful to decompose $P(w_1^T)$ in the following manner:

$$P(w_1^T) = \prod_{t=1}^{T} P(w_t | w_1^{t-1}) \tag{2.10}$$

Instead of learning $P(w_1^T)$ directly, a specific language model typically learns $P(w_i | w_1^{i-1})$; the probability distribution over the vocabulary, conditioned on the preceding words in the sentence. The likelihood of the complete sentence, $P(w_1^T)$, can then be computed from the conditional probabilities using equation 2.10.

Some of the simplest language models are n-gram models, which are simple statistical models. Prior to neural models, most state-of-the-art language models utilized n-grams. These models approximate $P(w_i | w_1^{i-1})$ using a shorter conditional context. For instance, a trigram model (n = 3), would learn $P(w_i | w_{i-2}, w_{i-1})$; the likelihood of w_i given the two prior words. These conditional probabilities would simply be calculated by considering all possible trigrams in a large text corpus, and then counting the relative occurrences of trigrams:

$$P(w_i | w_{i-2}, w_{i-1}) \approx \frac{N(w_{i-2} w_{i-1} w_i)}{N(w_{i-2} w_{i-1} w*)} \tag{2.11}$$

where N is a function describing the count of a particular trigram and $w*$ describes a "wild card" word.

Despite the extensive use of n-gram models historically, they have certain drawbacks [Bengio et al., 2003]. Most noteworthy, the context taken into account

[6]According to some reference dataset.

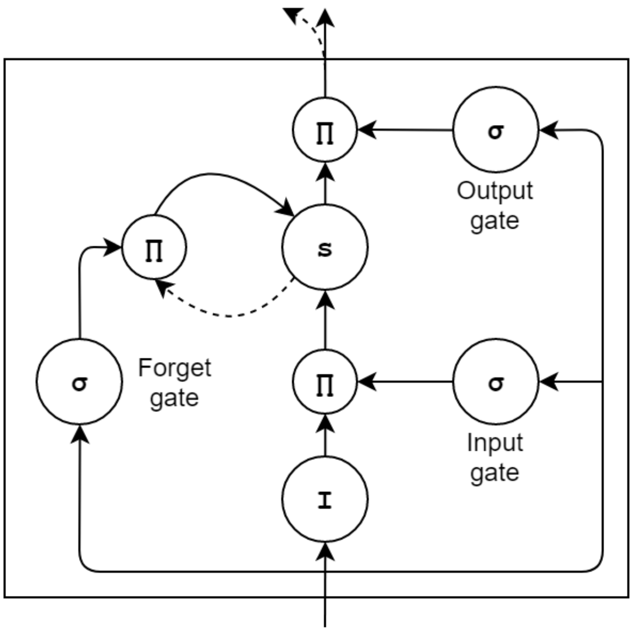


Figure 2.9: A high-level view of an LSTM memory cell with input, output and forget gates. The s node is the internal state, which, with its self-loop, implements the constant error carousel. Dashed lines go forward in time. Here, the forget gate is inserted into the self-loop to allow forgetting state when necessary. The I node is the input node and describes some input transformation. The three gates take the input signal and output a value between 0 and 1, corresponding to a closed and open gate, respectively. The input signal includes both the inputs to the LSTM layer and fully-connected recurrent connections between all the LSTM cells within the LSTM layer (see figure 2.2).

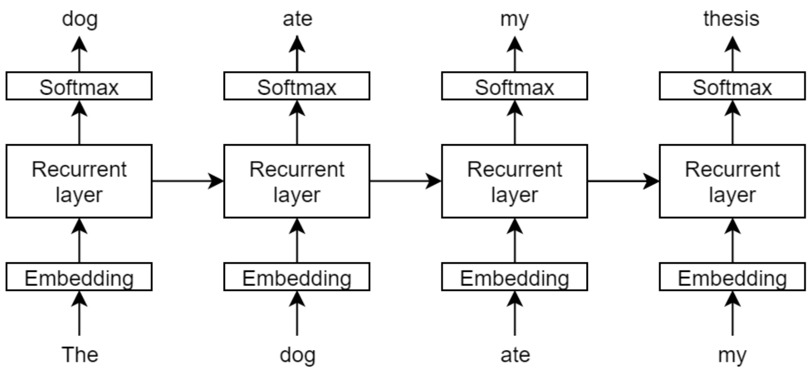


Figure 2.10: A possible architecture for an RNN language model, unrolled over time. Each word is substituted with a word embedding in the embedding layer. An LSTM layer is often used to implement the recurrent layer. The softmax outputs are the probability distribution over the set of words in the vocabulary.

is limited in size. N-gram models with $n > 3$ are seldom used, because the number of possible n-grams grows exponentially as a function of n, requiring exponentially more data. Furthermore, the models do not exploit semantic similarities between different words. In the next section, an RNN language model is presented, offering a solution to these two problems; the former through the RNN's theoretically unlimited context, and the latter through word embeddings.

2.3.1 RNN language models

A common RNN architecture for language modeling is shown in figure 2.10. The model takes a word, $x(t)$, and outputs a probability distribution over the vocabulary, estimating $P(w_{t+1}|w_1^t)$. The learning objective is to maximize the joint probability over the sentences in the training set, τ, which is achieved by finding the network weights, θ, that maximize the log-likelihood:

$$max_\theta \sum_{s \in \tau} log\hat{P}(s;\theta) \tag{2.12}$$

The softmax activation function is used in the output layer, which ensures that the probability distribution sum to 1. It should be noted that the use of the softmax output layer in this context is very expensive. The dimension of the output layer is equal to the size of the vocabulary, which can be hundreds of thousands of words. Thus, the size of the output layer can exceed the size of the other layers by orders of magnitude and become a bottleneck. Most applications therefore limit the vocabulary to e.g. the top 30K most frequent words, calculated based on the number of occurrences in the training set. With this approach, the words in the training set which are not included in the vocabulary are replaced with a special Out Of Vocabulary (OOV) token. This is a trade-off: OOV tokens cause "holes" in the sentences and too many of them are detrimental to their quality, while a vocabulary that's too large will make training infeasibly slow.

The example in figure 2.10 also uses *word embeddings*, which has become a standard technique for neural language modeling. The technique is motivated by several drawbacks of the traditional one-hot[7] categorical input encoding. The one-hot encoding is sparse, while neural networks normally prefer dense input vectors, and all vectors have uniform distance in vector space, regardless of semantic meaning. Furthermore, a one-hot encoding would impose upon the input layer the same level of complexity which torments the output layer. Word embeddings solve all these issues. When embeddings are used, all input words are substituted by real-valued vectors, $\mathbb{R}^N$, of some embedding dimension, N. The embedding vectors are typically stored in a $|V| \times N$ matrix, for a vocabulary V, and typically $N \ll |V|$, with for instance $N = 500$. The vectors are regarded as normal network weights during training and are gradually updated so that their organization in vector space reflect their semantic meaning. This encourages generalization: if the network has learned a particular sentence involving the word "dog", it will be able to infer that an alternative sentence where "dog" has been replaced by "cat" is nearly as likely, because the word embeddings for "dog" and "cat" are likely similar [Bengio et al., 2003].

2.3.2 Generating sentences

A language model can be used to generate text, which, as the reader might recall, is how the current thesis intends to do machine translation. Numerous models utilize language models that are conditioned on some extra piece of information: for machine translation, the decoder can be viewed as a language model conditioned on an encoding of the source sentence, and likewise for image captioning, only conditioned on an encoding of an image instead of a sentence. Particularly, the current work will use a language model conditioned on the values of parametric bias nodes.

[7]The one-hot encoding describes using vectors with dimension equal to the number of possible classes, where only one entry in the vector is equal to one, corresponding to the particular class, and the rest are equal to zero.

An unfortunate hurdle is, however, that language models do not provide an explicit solution for text generation. Of course, in theory one could compute

$$\hat{s} = \arg\max_{s \in S} \hat{P}(s) \tag{2.13}$$

for some finite set S containing all sentences of a sufficient length. While this would yield an optimal solution, it quickly becomes infeasible for sentence lengths longer than a few words. Two common approaches which provide acceptable, albeit non-optimal solutions, are greedy generation and beam search.

Greedy generation entails simply building the sentence iteratively left to right and choosing the most likely word at each step:

$$\hat{w}_t = \arg\max_{w \in V} \hat{P}(w|\hat{w}_1^{t-1}) \tag{2.14}$$

Typically, the generation ends when the most likely word is an end-of-sentence token. Since this problem does not possess the greedy property, a series of such locally optimal choices does not necessarily provide a globally optimal solution, i.e. $\hat{w}_1^T \neq \hat{s}$. The main argument for this approach is its simplicity.

Beam search is often used in practice and is almost always the algorithm of choice for the decoding step of neural machine translation. Beam search considers a larger set of possible sentences and selects the best solution from this set, thereby increasing the likelihood of finding an optimal or near-optimal solution. Specifically, the search maintains a set of the B best partial sentences found so far, where B is the beam size. At each step, generating left-to-right, each of the B partial sentences are expanded with the corresponding B most likely words, for a total of B^2 candidate sentences. These candidate sentences are then pruned again to the B most likely sentences before moving to the next step. Due to the nature of the RNN language model, the network's hidden states must be saved along with each candidate sentence to allow for extending the sentence further. A typical choice is $B = 10$. When $B = 1$, beam search defaults to greedy generation. Beam search is illustrated in figure 2.11.

2.4 Evaluation metrics for translation

The experiments conducted here utilize two evaluation metrics, BLEU and perplexity. The BLEU metric was proposed in Papineni et al. [2002] as a quick, inexpensive and language-independent way to automatically evaluate translations, that correlates highly with human evaluation. In the current experiments, it is used to report the final scores that are achieved in comparison with other models. Given the importance of the BLEU metric in this work, it is defined thoroughly below. Perplexity is a metric that stems from information theory and has been

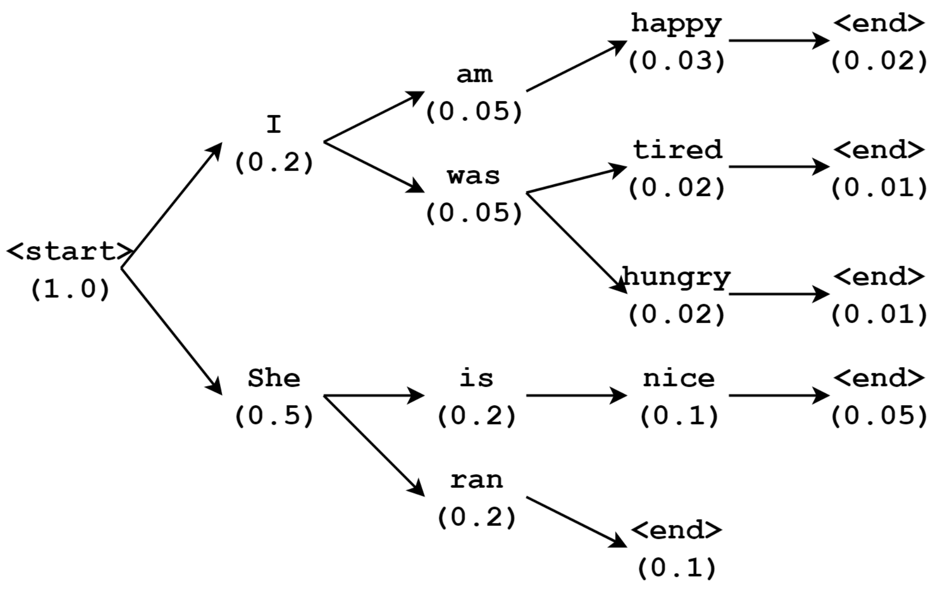


Figure 2.11: Beam search: the final beam upon completion of the search, where $B = 5$. Each node corresponds to a partial sentence, consisting of the words along the path from the root node to the node. The likelihood of the partial sentences are given in parentheses. Leaf nodes must be `<end>`-nodes, meaning that only viable, complete sentences are considered as final candidates. Note that this is not a complete search tree: other nodes would have been considered during the search, but discarded due to lower probabilities than the nodes included here. This is also why the sum of the likelihoods of child nodes do not add up to the likelihood of the parent nodes in the figure. The result of this beam search would have been the sentence "She ran", which has the highest likelihood of all complete sentences.

used extensively in work with language models and machine translation. In the current experiments, perplexity is used to evaluate the progress of training by measuring the perplexity on the training set and validation set.

2.4.1 BLEU

The BLEU (Bilingual Evaluation Understudy) metric is the de facto metric used in machine translation literature to evaluate the performance of proposed machine translation models. When multiple sources report the BLEU score on the same test dataset, the performance of their respective models can be directly compared. The motivation for the metric is due to how a given sentence typically has numerous possible good translations, and the metric must therefore be flexible enough to allow for variations. A test dataset contains sentences along with one or more corresponding *reference translations*. A machine translation model then generates *candidate translations* of the sentences in the dataset, which are scored against the reference translations using the BLEU metric.

The BLEU score is a statistical approach based on comparing n-gram counts from a candidate translation to corresponding counts in the reference translations. N-grams of multiple sizes are compared, starting from $n = 1$ (unigrams) to an upper limit, typically $n = 4$. A score is computed for each n-gram size, and the geometric mean of these scores are taken.

The score for a given n-gram size n, p_n, is computed over the entire test dataset as such [Papineni et al., 2002]:

$$p_n = \frac{\sum\limits_{C \in \text{Candidates}} \sum\limits_{\text{n-gram} \in C} \text{Count}_{\text{clip}}(\text{n-gram})}{\sum\limits_{C \in \text{Candidates}} \sum\limits_{\text{n-gram} \in C} \text{Count}(\text{n-gram})} \tag{2.15}$$

where $\text{Count}_{\text{clip}}(\text{n-gram})$ is a function that counts the occurrences of the n-gram in the candidate, but clips the value at the maximum occurrence count of the n-gram among the references.

The BLEU score is then computed as

$$\text{BLEU} = \text{BP} \cdot \exp\left(\frac{1}{N} \sum_{n=1}^{N} \log p_n\right) \tag{2.16}$$

where BP is a Brevity Penalty that penalizes too short translations. Again, a typical choice for the maximum degree of the n-grams is $N = 4$. The brevity penalty is computed globally over the set of candidate translations as

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{1-r/c} & \text{if } c \leq r \end{cases} \tag{2.17}$$

Here, c is the sum of lengths of the candidate translations, and r the sum of lengths of the corresponding reference translations. For r, if a given sentence has several reference translations, the one that is closest in length to the candidate translation is used in the sum. Equation 2.16 gives a score in the interval $[0, 1.0]$, but the score is conventionally multiplied by 100. Therefore, a BLEU score of 100 describes perfect translations.

2.4.2 Perplexity

The perplexity of some sample, given a probability model, measures how well the probability model predicts the sample. A low perplexity implies that the probability model evaluates the sample as likely. Language models can typically be evaluated by calculating the perplexity of some test set. If the test set is representative for the language, and if the language model is accurate, a low perplexity will be calculated.

When perplexity is used in the language domain, the average perplexity per word is normally used, as opposed to perplexity per sentence. In this case, the perplexity describes the number of bits that are optimally required to code each word, with respect to the language model. For instance, if the language model would require on average 10 bits to code each of the words in a sentence, the perplexity of the sentence would be calculated as $2^{10} = 1024$. Then, imagine a "dumb" language model, which considers any arrangement of words to be equally likely. Such a language model could not code each word in a sentence any more optimally than the naive coding (which affords the same number of bits to all words), which would require $\log(|V|)$ bits, for a vocabulary, V. In this case, the perplexity, $2^{\log(|V|)} = |V|$, would be equal to the size of the vocabulary. In the other extreme, consider a hypothetical language with only a single possible sentence. With such a language, an accurate language model would need zero bits to code each word in this single sentence, yielding a perplexity of $2^0 = 1$, which is the lowest possible attainable perplexity.

In general, the average perplexity of a test set, D, with regard to some language model can be computed as such:

$$\text{perplexity}(D) = \frac{1}{|D|} \sum_{x \in D} 2^{\frac{1}{T} \sum_{t=1}^{T} -log(\hat{p}(x_t))} \tag{2.18}$$

where x_t is the t'th word in the sentence x, and $\hat{p}(x_t)$ is the probability that the language model estimates for the word, x_t. Since the language model considers the previous words in the sentence, $\hat{p}(x_t)$ is taken to mean $\hat{p}(x_t|x_1^{t-1})$. Furthermore, $-log(\hat{p}(x_t))$ can be recognized as the cross-entropy function normally used as a loss function for sparse classification[8]. The experiments conducted here therefore calculate the perplexity as

$$\text{perplexity}(D) = \frac{1}{|D|} \sum_{x \in D} 2^{\frac{1}{T} L_c} \tag{2.19}$$

where L_c is the cross-entropy loss summed over time steps and T is still the length of sentence x. Due to implementation details, the perplexity is calculated using base e instead of 2 in the current experiments, which doesn't affect the result. Lastly, it is noted that perplexity is calculated in the exact same manner for a translation model compared to a language model. The only difference is that, since the decoder in an NMT model is conditioned on a representation of the source sentence, the perplexity is much lower. Ideally, a decoder needs only differentiate between possible legal, translations, and therefore a perplexity of 1 like the hypothetical example above is theoretically possible if only one possible translation exists.

2.5 Summary

This chapter defined Recurrent Neural Networks and provided motivation for their use. A few different architectures have been described, such as the Jordan and Elman RNN, and the LSTM. As the next chapter explains, the RNNPB is originally proposed as an adaptation of the Jordan-RNN, however it has also been implemented as an Elman-RNN. In this thesis, it is implemented as an LSTM network, in order to overcome the vanishing gradients problem and ease training. This chapter has discussed the BPTT algorithm and teacher forcing, two techniques which are combined in the current experiments. Different relationships between the inputs and outputs of an RNN were described. For subsequent chapters, the one-to-many relationship, which includes the RNNPB model, is particularly important. Language models were defined, and both n-gram and RNN implementations were discussed. These concepts are also important in the next chapters. Lastly, the BLEU and perplexity metrics for machine translation were introduced thoroughly. These play important roles in the experiments to come.

[8]I.e. where the targets are one-hot vectors, or in other words, where the classes are mutually exclusive.

Chapter 3

Motivation

The problem of machine translation is, without a doubt, important to solve. Few problems benefit more socially and on a global scale from good solutions. Allowing people otherwise unable to communicate with each other to do so could remove language barriers, unite different nationalities and deter xenophobia. Perhaps even more importantly, solutions to machine translation enable free flow of information across country borders. The advent of the Internet has connected the world and revolutionized information sharing, but the language barrier remains. It is vital for the development of countries with few knowledge resources that they can partake in the wealth of knowledge available on the Internet, just as freely available information is vital as a counterweight to the censorship and oppression many countries face.

> *"Knowledge is power. Information is liberating. Education is the premise of progress, in every society, in every family."*
>
> – Kofi Annan

This chapter presents related work for machine translation and the RNNPB model and puts the current work in context. In section 3.1, a Structured Literature Review protocol (SLR) is defined, which describes how the search for related work will be carried out. The results are then presented in section 3.2 and 3.3, for the RNNPB model and machine translation, respectively.

3.1 Structured Literature Review Protocol

This chapter is the result of executing a Structured Literature Review (SLR). An SLR defines a formal way of searching for and filtering the information relevant to a set of search questions. By formalizing this process, one can hopefully avoid biases in the selected material and assure that the scope of the considered material is sufficient. Furthermore, by making the process more transparent, the reader is better equipped to determine what related work has been considered, and which therefore underlie the arguments in the current work.

It is necessary to define a twofold SLR protocol given the nature of this thesis. Should one attempt to define a search for the intersection of the RNNPB model and machine translation, the resulting set of material would be empty.[1] The search must be included in the protocol for completeness, nevertheless. The SLR will thus cover the two domains separately. The next sections respectively define how the search will be conducted and how the results will be filtered.

3.1.1 Generating a set of candidate literature

In this step, all the literature that will be considered for selection is found. Here, the protocol must specify exactly what will be searched for and which sources will be searched. An important part of the SLR is to form search questions as the basis for the review. While the research questions stated in section 1.2 define the purpose of the current thesis, the search questions will define the scope of the related work inquiry. The following sets of questions will be used:

Machine translation

- What is the state-of-the-art in neural machine translation?
- What are the main challenges in the neural machine translation field?
- Which models are suitable to use as baselines to compare results against?

RNNPB model

- What work has been done with the RNNPB model?
- How has the PB binding method been used in prior work?

RNNPB for machine translation

- Has the RNNPB model or a similar model been applied to the task of machine translation in any prior work?

[1]At least at the time of writing, during the spring of 2019.

Since search engines normally perform better with keywords as opposed to complete questions, keywords are commonly derived from the search questions. Here, the notion of keyword groups is used. Groups are formed of keywords with interchangeable meaning to ensure that the scope of each group is complete. The three groupings are presented in table 3.1. A search query is then constructed using logical operators: "OR" between keywords within a group and "AND" between different groups. A subset of the first grouping could look like:

```
(Machine translation) AND (RNN OR LSTM) AND (State-of-the-art)
```

The three groupings produce three search queries. Furthermore, the following sources will be searched for each of the three queries:

- ACM digital library
- IEEE Xplore
- ISI web of knowledge
- ScienceDirect
- Springer Link
- Wiley Inter Science

3.1.2 Literature selection

This section dictates how the search results are distilled into a manageable size. Some of the search queries are rather general, so all results cannot be considered. A two-pass selection is used: first the *primary studies* are selected, and secondly, the primary studies are subject to a screening based on inclusion and quality criteria. A potential study will be discarded if it meets any of the following removal criteria:

R1. It is a duplicate of another primary study.

R2. The study is not published in English.

R3. The study is not situated in the computer science field.

Similarly, a study will be considered further only if it meets the following inclusion criterion:

I1. The study's primary concern is either machine translation for natural language, or the RNNPB model.

	Group 1	Group 2	Group 3
Term 1	Machine translation	Encoder-Decoder	State-of-the-art
Term 2		Sequence to sequence	

(a) Keywords for the machine translation domain.

	Group 1	Group 2
Term 1	RNNPB	Language
Term 2	Forwarding Forward model	Language modeling
Term 3	RNNPB PB Binding	Linguistic
Term 4		Big data
Term 5		Large dataset
Term 6		Deep learning

(b) Keywords for the RNNPB model domain. In early works, the RNNPB model was referred to as the Forwarding Forward model.

	Group 1	Group 2
Term 1	RNNPB	Machine translation
Term 2	Forwarding Forward model	
Term 3	Parametric bias	
Term 4	Mirror neuron	
Term 5	Mirror system	

(c) Keywords for the intersection of the two domains. The RNNPB model can be characterized as a mirror system, which describes systems capable of simultaneously offering forward and inverse modes of computation (generation and recognition). With broadening the scope in mind, the present approach requires two mirror models and a mechanism for binding them.

Table 3.1: Keyword groupings used for the three sets of search questions, covering the domains of machine translation and the RNNPB model.

	Domain	Criteria
I2	Both	The study is a primary study that presents empirical results.
I3	Both	The suggested machine learning strategy is fully supervised.
I4	MT	The proposed solution is based on recurrent neural networks.
I5	MT	The data utilized in learning consists only of written language.
I6	RNNPB	The task investigated has either a language aspect or makes a technical contribution to the model.
Q1	Both	The study clearly states the aim of the research.
Q2	Both	The study adequately places the work in context of related research.
Q3	Both	The algorithm/model is reproducible.
Q4	Both	The data is publicly available.
Q5	Both	The experimental set-up, including choice of hyperparameters, is adequately described.
Q6	Both	The performance metric is reproducible.
Q7	MT	For performance metric, the BLEU score is used.
Q8	MT	The suggested model beats established records.

Table 3.2: Inclusion and quality criteria for the SLR protocol. The table specifies which domain the criteria pertains to, given that the SLR searches both the domain of machine translation and of the RNNPB model.

If necessary, the size of the resulting set is restricted to 30 in the following manner: Sorted by the number of citations, select the top 10 articles published after 2015, and subsequently the top 10 non-overlapping articles published after 2010. Select the remaining 10 articles as the top non-overlapping articles after sorting by relevancy. With three search queries and seven sources, this will produce a set of primary studies of size at most 630.

The final selection is made using further inclusion criteria and quality criteria as shown in table 3.2. The included studies are assigned a score based on the quality criteria: for each of the criteria, 1p is given if the criteria is met, 0.5p if it is partially met, and otherwise 0p is given. The topmost 15 machine translation studies and 15 RNNPB studies by quality score are then chosen as the final basis for the related work. It is necessary to restrict the scope in this manner due to the limited time frame of the current work.

In the remainder of this chapter, the results of this search is now presented.

3.2 The RNNPB model

The RNNPB model [Tani, 2002] [Tani and Ito, 2003] has been employed within fields such as robotics, cognitive science and neuroscience. Within these different disciplines, the model itself largely remains the same, while the interpretations and goals of the experiments vary. This section now proceeds with a description of the RNNPB model as proposed in Tani and Ito [2003], before presenting more specific related work.

3.2.1 Definition and motivation

The model is originally motivated by the task of learning multiple temporal patterns in a single model. A standard RNN can easily be employed to learn a single temporal pattern; simply train it on the sequence or sequences exemplifying the temporal pattern, and the RNN can learn to generate the sequence in a closed-loop manner. Specifically, as illustrated in figure 2.7-b in the background chapter, the RNN can learn the mapping

$$(x_0, h_0) \xrightarrow{\theta} x_1, x_2, ..., x_T \tag{3.1}$$

where θ describes the network parameters, and x_0 and h_0 describe the initial input and hidden state which are fixed at some specific values, e.g. $(\vec{0}, \vec{0})$. Typically, what is achieved here is that the RNN has approximately learned the underlying temporal pattern from the examples, in a way that is robust against perturbations and noise. As such, the RNN may for instance generalize and predict the extension of the pattern. The benefits of this approach become clear with respect to the application in robotic experiments, which is described later in this section.

The question to which the RNNPB model offers a solution then becomes: how can *multiple* temporal patterns be learned at the same time? Two main strategies can be differentiated here as the *localist* and the *distributed* approach. In the former, each temporal pattern is learned in its own module – e.g. in its own network – and some control mechanism is employed to select the right module. In the distributed strategy, to which the RNNPB model belongs, all the patterns are embedded within a single module, here a single RNN. The latter has the generalization advantage of neural networks: having all the patterns share resources encourages generalization. However, the patterns must be differentiated in some manner. Section 2.2.4 describes two ways to achieve a one-to-many mapping with RNNs, using either x_0 or h_0 or both (from the mapping above) to encode an input vector. Such an input vector can be used to key the different temporal patterns. In this vein, Tani and Ito proposed a similar approach with the mapping

$$(x_0, h_0, p) \xrightarrow{\theta} x_1, x_2, ..., x_T \tag{3.2}$$

where p denotes the parametric biases. More specifically, there is a one-to-one mapping between parametric biases and temporal patterns, so if S_i denotes the i'th pattern, the mapping can be written as

$$(x_0, h_0, p_i) \xrightarrow{\theta} S_i \tag{3.3}$$

The complete RNNPB architecture is shown in figure 3.1. Tani and Ito describe the architecture as a Jordan-type RNN with the addition of parametric bias nodes in the input layer. This classification arises from the output-to-input-layer recurrence, but isn't fully accurate with regard to the conventional definition of the Jordan RNN. The difference is that the RNNPB embeds dedicated context units in the output layer as well as in the input layer, and importantly, the output context units are not subject to a teacher signal. The context units are therefore free to take on any suitable representation, and are thus just as expressive as in the Elman-type RNN.

An important quality of the RNNPB model is that the PB values are self-organized during training. In common with word embeddings, the p_i vectors are regarded in much the same manner as standard network weights and updated iteratively during training, starting from some initial values. Therefore, just as for word embeddings, the PB space eventually becomes organized in a regular manner, so that similar temporal patterns correspond to similar PB vectors. This quality encourages generalization, where even untrained points in PB space may correspond to useful temporal patterns. The exact procedure and equations used for training are given in detail in the next chapter, but differ from standard BPTT in only minor ways.

After training is completed, the model offers two modes of operation: **generation** and **recognition**. Generation describes the forward operation of generating a complete sequence from a PB vector, p. On the other hand, recognition describes the inverse operation where a PB vector, p, is computed from a (possibly untrained) sequence. Recognition is achieved by iteratively searching for the PB vector that generates the given sequence with the smallest error. This is merely a special case of the network training procedure; recognition is achieved by updating only the PB nodes with the BPTT procedure and the given sequence as teacher signal, while keeping the network weights fixed. Since recognition is a rather computationally expensive iterative operation, it may not be suitable for all real-time tasks.

Generation and recognition are illustrated in figure 3.2 and 3.3, respectively. In this thesis, the original RNNPB architecture from figure 3.1 is updated. The

Jordan-type recurrence is replaced with the more powerful Elman-type recurrence (hidden-layer to hidden-layer), using LSTM memory cells. Since the RNNPB is used to model language, embedding layers and softmax layers are furthermore added to the architecture. Using two such RNNPB models, one for the source language and one for the target language, this thesis intends to achieve machine translation as illustrated in figure 3.4. The concept of binding is described further in section 3.2.3.

In the next subsections, the most relevant applications of the RNNPB model are presented.

3.2.2 Dynamic object handling

In Ito et al. [2006], the RNNPB model is utilized in a dynamic object handling experiment with a robot. The goal is to execute learned behaviors while responding appropriately to the environment. Here, the temporal patterns correspond to behaviors and are represented as sensorimotor sequences, (s_t, m_t), of sensory features and motor parameters. This enables the robot to perform a suitable action at the next time step by predicting and effectuating motor parameters, m_{t+1}. Since sensory features are incorporated in the sequences as well, the robot also learns to consider its sensory inputs when it decides upon an action, thus responding to its environment.

In order to dynamically switch between behaviors in real-time, the RNNPB system continuously recognizes, by updating the PB values, a window of the immediate past. As such, if the robot's sensorimotor stream suddenly changes in nature, the PB values will be regressed to reflect the new behavior. Thus, the system continuously alternates between the generation operation, in which the next action is generated, and the recognition operation.

The experiment tests a humanoid robot's ability to handle a ball on a table with its hands in terms of two different behaviors. With the first behavior, the robot rolls the ball back and forth between its left arm and its right. With the second behavior, it grasps the ball with both arms, lifts it up and drops it down. The sensors include a camera to enable the robot to see the ball's location. After training these two behaviors, the robot was able to stably reproduce them and furthermore switch between them appropriately. For instance, if a human observer intervened to initiate the opposite ball handling behavior by manually moving the ball, the system experiences a mismatch between its sensory values and its internal prediction, starts adjusting its PB values, and eventually switches to executing the opposite behavior.

Thus, this experiment confirmed that 1) the behaviors could be successfully learned and associated with PB values, 2) the robot could successfully take the environment (e.g. the ball's position) into account, and 3) the system could

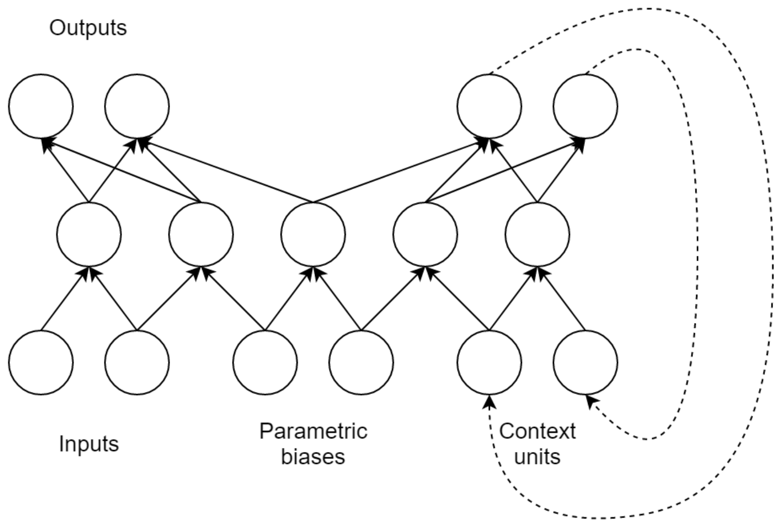


Figure 3.1: The original RNNPB architecture. Two parametric bias nodes have been placed in the input layer and encode the specific temporal pattern that should be produced. The values of the PB nodes remain fixed over time for a given pattern. The recurrence is implemented with arcs from the output layer to the input layer, using context units. The layers are normally densely connected, and the number of input, output, parametric bias and context units can be chosen (mutually) independently. By operating the network in closed-loop mode (feeding the outputs back in as inputs), an entire sequence can be generated from only the PB values and an initial input.

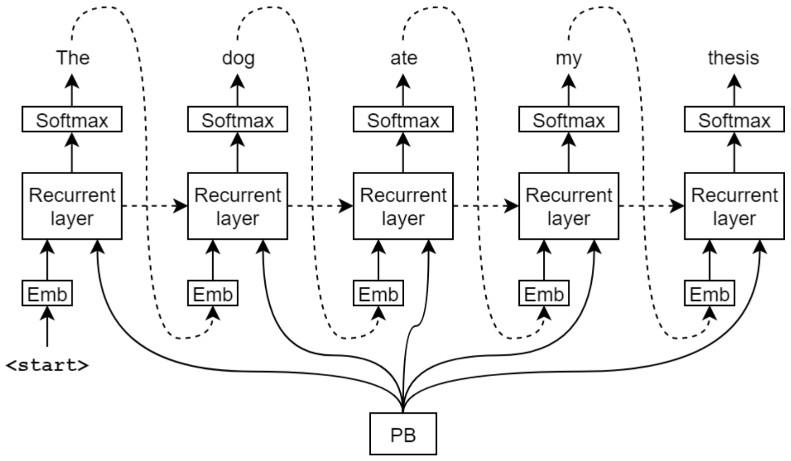


Figure 3.2: Generation mode. The figure shows an RNN language model, as presented in figure 2.10, adapted with PB nodes. In the generation mode, the network takes only a PB vector as input. The PB vector encodes a sentence fully, and the model can thus generate the sentence from the PB vector.

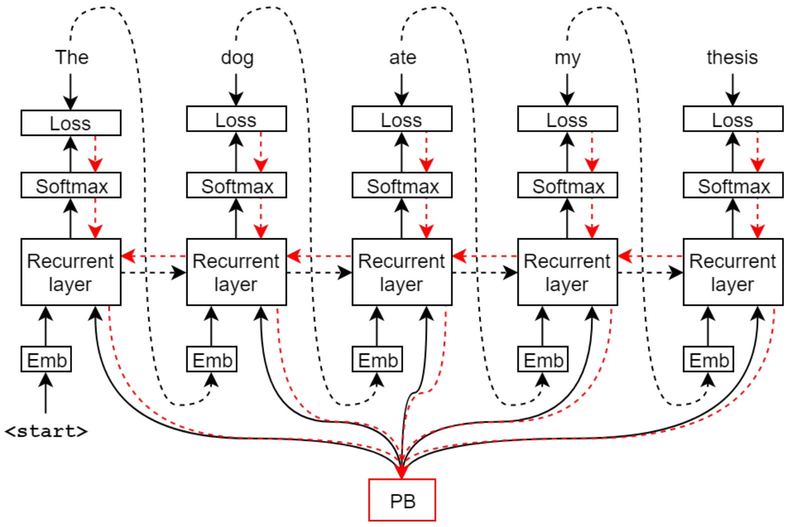


Figure 3.3: Recognition mode. A sentence is provided as input to the network, and is then used as teacher signal (at the top). The gradients (red) flow from the loss blocks to the PB block. The PB nodes are then regressed over a number of recognition iterations, until they sufficiently represent the provided sentence.

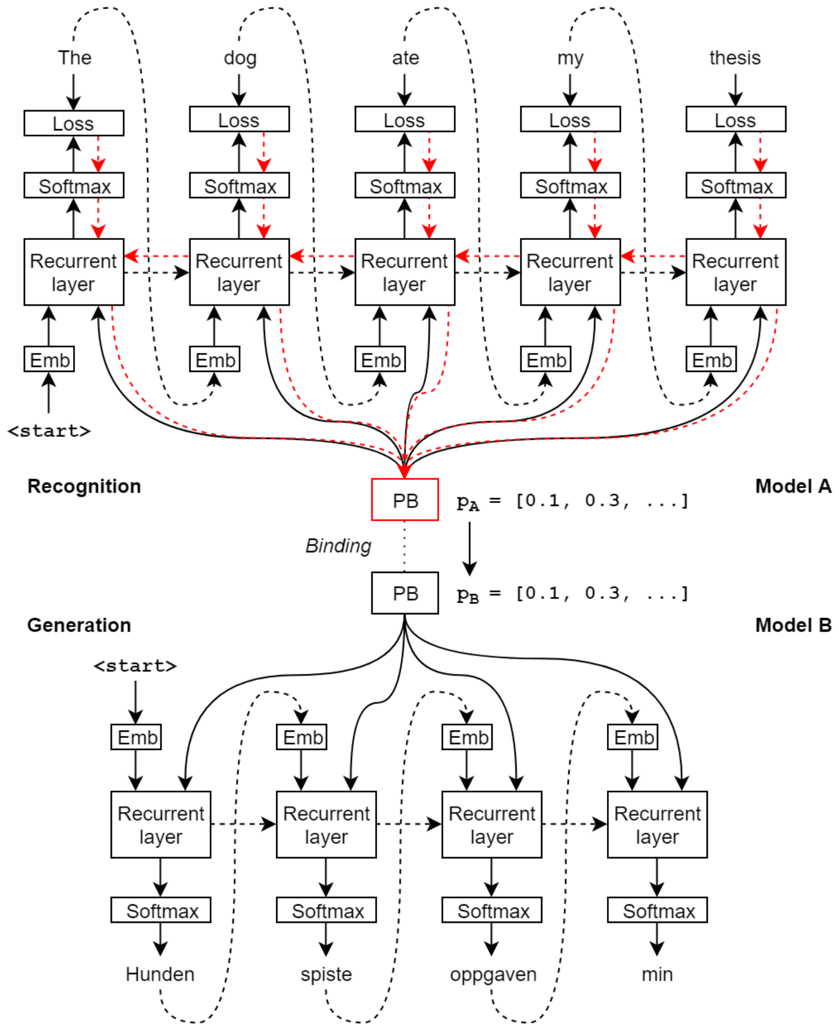


Figure 3.4: Translation. Two models, model A and model B, are bound. Binding entails that the PB vectors for corresponding sentences in model A and model B are linked during training, so that a common sentence representation emerges in the two models. Translation can then be achieved by doing recognition in model A and subsequently using the computed PB vectors for generation in model B.

automatically switch between behaviors, or in other words, achieve automatic *context switching*. In order to achieve real-time generation and recognition, the robot had to be connected to a remote cluster to carry out computation.

3.2.3 PB Binding

The experiment which this thesis is inspired by is presented in Sugita and Tani [2003] and Sugita and Tani [2005], wherein two RNNPB models are bound together. In this case, a system is built to facilitate translations between the domains of language and behavior. One RNNPB model is trained on sentences and the other is trained on sensorimotor sequences representing behaviors, in the same manner as the above experiment. The motivation here is obvious: it is appealing to be able to dictate the behavior of a robot flexibly by using natural language. Here, corresponding sentence-behavior pairs are bound using the PB binding method. Given a sentence-behavior pair, (s_i, b_i), the corresponding PB units, (p_{s_i}, p_{b_i}), are bound. The training objective is now twofold: 1) as before, the two RNNPB models should update the values of p_{s_i} and p_{b_i} in the direction of generating s_i and b_i respectively, and 2) p_{s_i} and p_{b_i} should be updated in the direction of each other. The exact equations are given in the next chapter. After training, translation of arbitrary sequences can be done by doing recognition in one module and subsequently generation in the other model using the obtained PB values. Note however that the two models can still be operated individually as well.

In the experiment, simple two-word sentences were associated with novel behaviors involving a mobile robot with wheels and a movable arm, in addition to visual sensors. The robot was placed with three objects in front of it: one object to the left, one center and one to the right. The robot was taught three actions: to push an object with its body, to point towards it with its arm but not touch it, and to hit it with its arm. In addition, the objects had different colors, with the left object always being red, the center blue and the right green. With three actions and three objects, there was a total of 9 behaviors for the robot to learn. The linguistic model learned sentences with the format "`<verb> <noun>`", for instance "hit blue" or "push right", corresponding to the different behaviors. This gives 18 possible sentences, where each behavior is described by two different but equivalent sentences.

In order to test generalization, four sentences, corresponding to two behaviors, were not trained at all. The two corresponding behaviors were trained but not bound. It was demonstrated that the system could both regenerate all sequences successfully, both linguistic and behavioral. More importantly, the robot could generate the correct corresponding behavior for all sentences, including the unlearned sentences. Sugita and Tani show that the PB vectors of equivalent sentences converge near each other in PB-space. Likewise, the PB vectors of sentence-behavior pairs converge towards becoming equal. The experiments showed that it's possible to translate between the two domains, but on a small scale. In the current experiment, the binding will be done between two language domains on a much larger scale.

The experiment is extended in Arie et al. [2010], where the RNNPB models are replaced with two Multiple Timescale RNN (MTRNN) models. PB nodes from the language and behavior models were still bound. In this experiment, two groups of sentences are trained. The first group has the format "Hold Red", similar to the original experiment, while the second group contains more complex sentences of the format "Put Red on Blue". In this case, all pairs were bound, and the model did learn to recognize all sentences and execute the corresponding behavior. However, the goal here was mainly to investigate how the PB space became organized in response to the two groups of sentences. The authors found that for the first group, the PB space exhibited a compositional structure that is necessary for generalization, which was lacking for the second group. One possible explanation for this was identified as too few training examples, and as future work, the authors describe a need to repeat the experiment on a larger scale with a higher number of words and a larger diversity of grammar. To this end, this thesis can be considered a direct contribution.

Ogata and Okuno [2013] conducted a very similar experiment with bound MTRNN language-behavior models, although they employed a more complex architecture that incorporated a Self-Organized Map (SOM) and trained sentences on the character-level. They too achieve successful language-behavior translation, and mention scaling of word vocabulary in future work.

3.2.4 Recent work with the RNNPB

Park et al. [2017] investigates the RNNPB model's ability to learn to imitate goal-directed actions. The study aims to analyze the developmental dynamics of the model; that is, how the performance develops as the model trains. Using the original RNNPB model without alterations, they test a robot arm's ability to move from an initial position to one of two different goal positions on a 2-D plane. For each of the two *goals*, they define three different *means*, i.e. types of trajectories for reaching the goal. They find that the model's ability to reproduce the actions gradually increase with training, and more importantly, that the model learns the goals of the actions before the means. Only in the later training iterations are the exact trajectories learned. In Park et al. [2014] the same authors compare similar findings with the developmental dynamics of infants; when infants learn actions through imitation, they first learn to copy the goal of the action and later the means. The authors therefore conjecture that the RNNPB model may shed light on the mechanism of infant goal-directed action learning.

Zhong and Canamero [2014] and Li et al. [2018] are two studies that investigate learning to recognize and generate emotion-aroused human behaviors with the RNNPB model. The studies model the human differently, using three-dimensional skeleton motion capture and Microsoft Kinect respectively, but both studies have humans executing a behavior while expressing different emotions, such as joy, pride, anger and sadness. Both studies have the model learn a double of each emotion-behavior to examine if the system recognizes the similarity of each pair. Based on euclidean distance between the PB-vectors, the studies show that the model clusters the same emotions together in PB space. They report the full distance matrix between emotions, which for instance shows that joy is closer in PB space to pride than it is to fear and sadness. Zhong and Canamero [2014] also tested generation by reconstructing the human skeleton in an avatar. Here, they discovered interesting generalization capabilities, where PB values corresponding to different emotions could be interpolated to generate novel emotion-behaviors situated somewhere between the two emotions. Both studies utilize an Elman-type RNNPB, instead of a Jordan-type, an adaptation which is also used in Zhong et al. [2011], where it's referred to as a Simple Recurrent Network with Parametric Biases (SRNPB). The latter motivate this by a greater ease of learning and claims that the backpropagated error is smaller.

3.3 Machine translation

This section presents some of the most relevant work in the field of machine translation and describes some of the most significant challenges. In the first section, a brief summary of the field's history is given, where the most significant approach, Statistical Machine Translation, is still relevant today. In section 3.3.2 and 3.3.3, important neural models are presented, while section 3.3.4 and 3.3.5 describe attempts at solving two important challenges: the Out-Of-Vocabulary problem and how to utilize monolingual corpora.

3.3.1 Early approaches

Machine translation has been studied since the 1950s [Costa-Jussà and Farrús, 2014] and the field has undergone a journey of different paradigms. The first systems were rule-based: they utilized bilingual dictionaries to translate word-for-word and used simple rules to correct the word order. These systems faced problems with translating context-dependent words and struggled to achieve proper word ordering; the rules for ordering the words were recognized as too ad hoc and complex [Hutchins, 2005]. A major shift occurred in the 90s with the rise of statistical machine translation (SMT). This marked the beginning of the transition from rule-based translation to example-based translation, i.e. learning to translate from parallel corpora. At this point, the problem of machine translation was formulated as finding the most likely target sentence $\hat{y}$ among all possible target sentences, given a source sentence, x. This formulation is referred to as the noisy channel approach [Costa-Jussà and Farrús, 2014] and required building a translation model, $P(\hat{y}|x)$. A common technique used was to decompose this probability model using Bayes theorem,

$$P(\hat{y}|x) \propto P(x|\hat{y}) \cdot P(\hat{y}) \tag{3.4}$$

While this decomposition still requires a translation model, $P(x|\hat{y})$, it also successfully incorporates a language model for the target language, $P(\hat{y})$, which can be trained much more thoroughly using monolingual data. The language models have typically been implemented as n-gram models, see section 2.3. So called phrase-based translation [Koehn et al., 2003] offers a method to build the translation model, whose impacts on the state-of-the-art translation models have lasted until recently. Here, phrase tables are generated from word-aligned parallel corpus[2] and the likelihood of the translation is calculated based on relative phrase occurrences [Costa-Jussà and Farrús, 2014]. This approach was then extended to incorporate and weigh additional models, such as lexical models and reordering models, by utilizing the log-linear framework [Och and Ney, 2002]. This framework allows combining several models, which are referred to as features, and compute the final conditional likelihood $P(x|\hat{y})$ as a weighted average over features,

$$P(x|\hat{y}) = \sum_i w_i P_i(x|\hat{y}) \tag{3.5}$$

In this framework, one can typically learn the translation model directly without decomposition, $P(\hat{y}|x)$, and instead incorporate the monolingual language model as one of the features. The models are developed independently, but the weights w_i are learned to maximize translation performance, typically using a minimum error rate optimizing procedure directly on the BLEU metric.

From 2014 till present, the field of machine translation has seen several advancements of the state-of-the-art from neural approaches. In the remainder of this section, the most important and relevant work for neural machine translation is presented.

[2]A parallel corpus where each source word is matched with its corresponding target word or words.

3.3.2 The Encoder-Decoder Architecture

The Encoder-Decoder architecture [Kalchbrenner and Blunsom, 2013][Cho et al., 2014b][Sutskever et al., 2014] describes a framework for machine translation with end-to-end training that's been shown to perform as well or better than conventional SMT systems. The architecture replaces the multiple, separately-tuned models common in SMT systems with a single system that models $P(y|x)$ directly. With this approach, the encoder encodes a source sentence, x, into a fixed-size continuous representation, call it c. The decoder then learns a language model conditioned on this representation, $P(y_t|y_1^{t-1}, c)$, or more intuitively: it learns to decompress this representation into a new sequence in the target language. The encoder and decoder are jointly trained to maximize the joint likelihood over the parallel corpus. The exact architecture employed and the exact use of the model vary. Three defining works are presented next.

Kalchbrenner and Blunsom [2013] were the first to employ the encoder-decoder for machine translation with the Recurrent Continuous Translation Model (RCTM). The work is motivated by certain disadvantages of SMT approaches: by modeling language in terms of discrete phrases, neither syntactic nor semantic similarities between similar phrases are captured. A model must be able to generalize better in order to overcome sparsity issues; as n-grams grow larger, they are less likely to have been seen before in the training data. Inspired by word embeddings, which have been shown to successfully capture syntactic and semantic similarity to aid generalization, the authors propose to encode the source sentence into a continuous space. In the spirit of comparison, the idea can be viewed as a *sentence embedding*, where sentences are mapped to a continuous *meaning space*. Special to this work is that the encoder is implemented using a convolutional neural network instead of a recurrent neural network. Thus, the source sentence is considered to belong to the spatial dimension instead of the temporal dimension. This is possible due to extra flexibility with source sentences: since the entire sentence can be assumed to be available at once, encoders can benefit from considering both past and future context. A large disadvantage is that some word-ordering information is lost with a convolutional approach. A maximum sentence length must also be imposed, to which all sentences must be padded. The decoder is a simple RNN without gates. The experiments test only qualitatively how well the model translates on its own, noting that significant syntactic and semantic information in English is transferred onto the French translation. They also evaluate their model quantitatively by *reranking* the candidate translations of an SMT system and concludes that the model has learned the translation distribution, although the baseline is not defeated in terms of BLUE score.

Cho et al. [2014b] advances the work of Kalchbrenner and Blunsom by proposing the RNN Encoder-Decoder. Here, the convolutional encoder has been replaced by a recurrent encoder. For the sentence representation, c, the encoder's final hidden state is used. Another important contribution here is that a memory cell is used to replace the basic RNN unit. The memory cell is inspired by the LSTM, but is simpler, implementing only a reset and update gate. This memory cell has since become known as the Gated Recurrent Unit (GRU) and has earned popularity matching the LSTM cell. Similar to Kalchbrenner and Blunsom [2013], the model is not tested for generating translations on its own, but instead incorporated into an SMT system. In this case, the model is used to evaluate the probability of phrase pairs, (p_x, p_y), by computing $P(p_y|p_x)$, replacing the estimated probability based on phrase-counts. This time, the experiments show the SMT system is improved, increasing its BLUE score.

Finally, Sutskever et al. [2014] add the last remaining piece and shows that an encoder-decoder model can indeed generate state-of-art translations on its own. The overall architecture is mostly the same as for Cho et al. [2014b], with a few differences. The architecture is shown in figure 3.5. The simplified memory cell is replaced by a complete LSTM memory cell. The authors utilize deep LSTMs with four layers for both encoder and decoder, finding that they significantly outperform shallow LSTMs, which they hypothesize is due to the increased dimensionality of the encoding, c. Lastly, the authors find that simply reversing all the source sentences is extremely valuable, increasing BLUE score from 25.9 to 30.6 on one task. The latter is contributed to introducing several short-term dependencies; if the source sentence is reversed, the last word the encoder sees is the first word of the original source sentence, which is then followed by the first word of the translation and so on, effectively decreasing the minimum distance between all dependencies. This is thought to aid the LSTM's learning ability. Beam search is subsequently used to generate translations, using a beam size of $B = 12$. Using an ensemble of encoder-decoder models, they achieved a BLUE score of 34.8 on the WMT-14 English-French dataset, beating several phrase-based models, and when using their ensemble for reranking translations from a phrase-based model, they achieved a score of 36.5, which is close to the 37.0 state-of-the-art for this task.

This shows that within a few years of their invention, these models are already on the verge of beating decades old translation systems. Several challenges have however been recognized. Cho et al. [2014a] tested the ability of the RNN Encoder-Decoder to do translation using beam search, inspired by Sutskever et al. [2014], and showed that the model suffers significantly from increasing sentence length. They hypothesize that this is due to the representation used for the source sentence, c, which is **fixed-sized**. Intuitively, the context must have some capacity limit for representing syntactic and semantic information. The authors identified a second large problem as the need to scale the architecture, specifically by increasing the vocabulary sizes. In short, translations suffer in the presence of long source sentences and unknown words. Sutskever et al. [2014] showed the op-

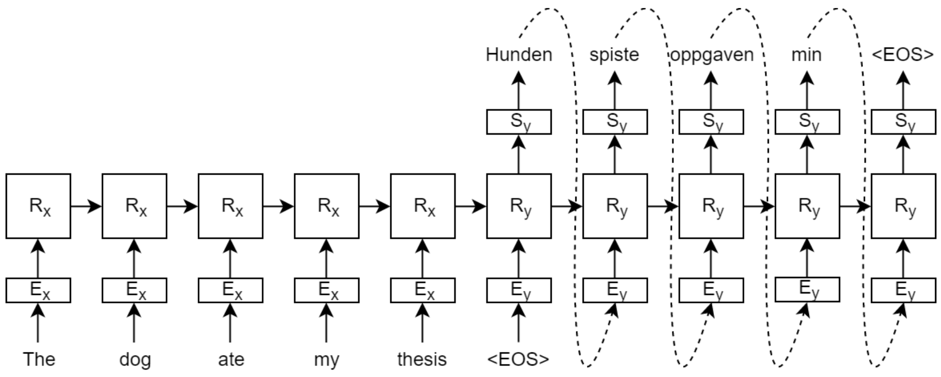


Figure 3.5: Seq-to-seq: The architecture employed by Sutskever et al. [2014]. Here, only one recurrent layer (R) is shown, instead of four. The subscripts differentiate the encoder and decoder, belonging to source language, x, and target language, y, respectively. It is apparent that the encoder and decoder form a single neural neural network that can be trained end-to-end. The inputs use an embedding layer, E, and the outputs a softmax layer, S. The dashed arcs denote the non-differentiable operation of choosing an output to be used for the next input.

posite in regard to sentence length, however: they concluded that the model did not in fact have difficulty on long sentences after investigating BLUE scores as a function of sentence length. Nevertheless, the search for a solution which scales to accommodate arbitrary sentence lengths bore fruits. The concept of *attention* as an alternative to the encoder-decoder models described above is presented next.

3.3.3 Attention

In the architecture suggested in Bahdanau et al. [2014] the whole source sentence is made available to the decoder, as opposed to just a summary, c. At each step of decoding, the decoder proceeds to calculate dynamically a context based on the most relevant words in the source sentence. On a high level, the mechanism can be described as *attention*: only the relevant source words are considered when a given target word is produced. The alignment between source words and target words is learned automatically. Some specifics are presented next. Notably, the calculation of the i-th target word, y_i, is now dependent on c_i as opposed to c.

The calculation proceeds as follows:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j \tag{3.6}$$

That is, the context is computed as a weighted average over the hidden states belonging to the encoder. In this context, the hidden states, h_j, are referred to as *annotations*. The weights, α_{ij}, are then computed as the softmax-normalization of a similarity function between each hidden state, h_j, and the previous hidden state in the decoder, s_{i-1}:

$$e_{ij} = a(h_j, s_{i-1}) \tag{3.7}$$

The normalization is not shown here. The similarity function, a, is typically implemented as a feedforward network, trained as a component of the full architecture. Note that each prediction still leverages a fixed-size context, c_i. The new contribution here is simply that each prediction utilizes a separate context, and therefore the scheme does not require that any one context vector encode the entire source sentence. Additionally, the encoder here is implemented as a Bidirectional RNN, and each annotation vector therefore summarize both past and future context.

Bahdanau et al. find that the model significantly outperforms the baseline encoder-decoder model, and that it is much more robust to sentence length. Indeed, the performance is reported to be comparable to existing phrase-based SMT models. The contribution has had a tremendous impact on the field of Neural Machine Translation, and most state-of-the-art NMT systems as of writing are attention-based. More recently, Vaswani et al. [2017] took this to the extreme and proposed an encoder-decoder model based solely on attention, without any recurrence or convolution. The proposed model used only self-attention to compute representations of its inputs. The proposal is revolutionary, surpassing state-of-the-art on the WMT 2014 English-to-German task while consuming much less training resources. The technique is not pursued any further here, as the method proposed in this thesis does not utilize attention.

3.3.4 Solving the out-of-vocabulary problem

Much work has been devoted towards finding solutions for the out-of-vocabulary (OOV) problem. As it stands, most approaches still limit the size of the vocabulary due to time constraints for training and decoding, leaving the models unable to translate very rare words. Conventional SMT systems do not suffer from vocabulary limitations: the only limitation here is the space the model oc-

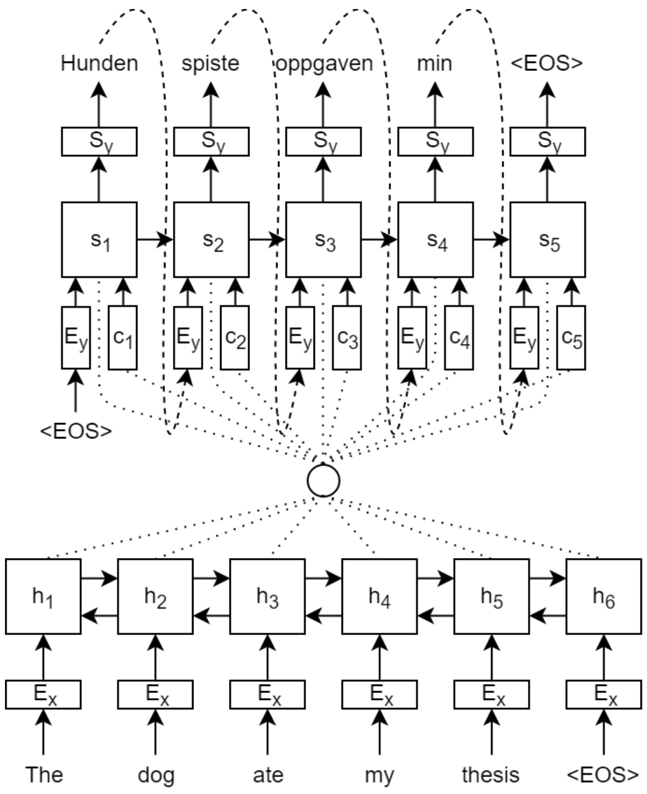


Figure 3.6: RNNSEARCH. The attention architecture employed by Bahdanau et al. [2014]. The encoder and decoder are connected through the contexts, c_i, and the dotted lines indicate the components involved in their computation. The attention mechanism enables the decoder to soft-search for relevant source words when it's translating a given word. The encoder is a bidirectional RNN, hence the bidirectional connections (a simplified illustration is shown here; a BRNN normally has two unconnected recurrent layers, processing in opposite directions, where the final hidden state is taken as the concatenation of the two directional hidden states).

cupies, which often exceeds gigabytes. To bridge this gap, both minor and major alterations have been proposed to the popular architectures.

Luong et al. [2015b] propose to simply patch in the translations of OOV words in a post-processing step. This method is appealing since it's simple and agnostic to the particular NMT model used. The challenge here is to learn the alignment between OOV words in the target sentence to the correct source words. The authors propose three schemes to learn this alignment, the most successful of which they call *PosUnk*. This scheme involves augmenting all the training data where the target sentence contains an OOV word by inserting an alignment token after each such OOV word. The alignment tokens, a_d where $d = -7, -6, ..., 6, 7$, are added to the dictionary and used to signal the relative index of the aligned source word. Thus, every time the decoder produces an UNK (unknown) token, it subsequently produces an alignment token signaling the corresponding source word. The scheme requires that alignment information be known for the training data, which can be produced using an unsupervised aligner algorithm (specifically they use the Berkeley aligner). Luong et al. notes that a technique to handle OOV words is necessary to achieve state-of-the-art, and proceeds to do so as the first NMT system to surpass all general MT systems for the WMT'14 English-to-French task.

More significant departures from established methods involve rethinking the basic unit of language. Chung et al. [2016] and Luong and Manning [2016] approach language on the character-level. The former utilize a standard encoder, but generate translations character-by-character. This reduces the size of the target-side "vocabulary" to the number of characters in the language, as opposed to words, effectively eliminating the vocabulary problem. Although this yields target sequences that are many times as long, the authors achieve strong results, outperforming non-neural approaches on three language pairs. Luong and Manning [2016] suggests a hybrid system that only falls back on character-modeling in the presence of OOV words. Their main model is thus a word-level encoder-decoder, with character-level models as a fallback. Their reasoning for using a hybrid system is that a pure character-level system is extremely slow to train[3]. On the encoder side, a character-level model computes word embeddings for words not in the source vocabulary, and on the target side, a character-level model is used to output a word character-by-character when a UNK-token is generated. The authors tested their model on the WMT'15 English-to-Czech translation task and achieved a new state-of-the-art.

He et al. [2016] also tries to handle the OOV problem, along with two other problems with NMT that they identify. The two other problems are (1) the

[3]Although the cost of a large softmax layer is avoided, the target sequences are many times longer. As mentioned in section 2.1, BPTT cannot be parallelized, which means long sequences are very ineffective to process.

problem of adequacy – that NMT lacks a mechanism to guarantee that all source words are translated – and (2) the problem of utilizing monolingual corpus. (1) describes a bias common in NMT towards shorter translated sentences, where pieces of information may be missing. (2) is especially relevant here, because it is the same problem to which the proposed model in this thesis could offer a solution. Their proposed solution pays homage to traditional MT models: they incorporate standard SMT models into their NMT model, under the log-linear framework. Treating their NMT model as one of four features, they additionally incorporate a word-translation model, an n-gram language model, and a word reward feature that rewards longer sentences. The word-translation model is able to offer translations even when the NMT model produces an UNK token. They find that OOV words are reduced by 82 %, and that incorporating the language model increases the *fluency* of the target language, thus increasing the BLEU score further. It is also worth noting that the proposed model in this thesis offers a solution for (1), since it ensures that the source sentence representation, c, is able to regenerate the source sentence fully before it's used to generate the target sentence. This is similar to Tu et al. [2016], who added a *reconstruction* component to their NMT system. The reconstructor takes the decoder hidden states and tries to reconstruct the source sentence, where the training objective was a weighing between translation accuracy and reconstruction accuracy. They commented that this improves adequacy, because it ensures that all information is transferred from the source side, similar to what the proposed RNNPB model could achieve.

The next section now proceeds to expand upon problem (2) of incorporating a monolingual corpus.

3.3.5 Using a monolingual corpus

As Gulcehre et al. [2017] note, much of recent successes with NMT is owed to large amounts of high quality parallel corpora. Such corpora are not always available, and many language pairs with few resources in terms of parallel corpora exist. Translation for these language pairs could therefore likely benefit strongly from utilizing monolingual corpora as well as parallel corpora. There has been a broad tradition for using monolingual data in SMT, but less so in NMT, as the previous section notes.

Gulcehre et al. [2017] propose two means of incorporating a (target-side) neural language model (RNNLM) into an NMT system, which they refer to as *shallow fusion* and *deep fusion*. The former *weakly* incorporates the language model: during the decoding process, both the NMT model and LM model are used to rank candidate translations in a weighted manner. With deep fusion, the hidden activations of the NMT model and language model are concatenated, and the output layer is used to map from the resulting vector. In both cases, the LM and NMT models are trained separately, but in the latter case the output layer is fine-tuned to learn to consider the LM activation additionally. Deep fusion proved superior to shallow fusion because the information from the LM could be selectively incorporated. The authors reported beating state-of-the-art on a low-resource pair, Turkish-to-English, and improving on the NMT baseline on high-resource pairs, Czech-to-English and German-to-English. They found that the improvement of utilizing monolingual data depends strongly on the similarity between the two sets of corpora, and that corpora from vastly different domains, e.g. SMS and articles, may not complement each other. The approach differs from what the proposed RNNPB method offers by utilizing monolingual language on only one side, while the RNNPB can utilize it on both. This could aid the model in learning more meaningful representations on the source side as well. Additionally, the approaches differ in the number of models that must be trained; with the RNNPB method, the translation model and language models are the same, and both corpora can be used to train at the same time.

Luong et al. [2015a] explore NMT in the multi-task framework. Using a standard encoder-decoder without attention, they show that it's possible to use many source and target languages. Furthermore, they show that it's possible to add monolingual translation paths to the model. For instance, given an English-to-French encoder/decoder, the encoder can also be intermittently connected to an English decoder to do English-to-English translation (i.e. autoencoding) with monolingual data. Similar to the current approach, monolingual data in both languages can be utilized, but requires two additional models (one extra encoder and one extra decoder). The NMT system with two autoencoders improves translation quality by 0.5 BLEU points on German-English. In common with the RNNPB approach is the drawback that attention mechanisms cannot be used.

Cheng et al. [2016] also incorporate autoencoders to make use of monolingual data to achieve semi-supervised NMT. Similar to Tu et al. [2016], mentioned in the previous section, they add a reconstruction term to the training objective, although the details and goal differ. Here, two NMT systems are trained, one source-to-target and one target-to-source. No constraint is placed on the choice of NMT system employed and attention can therefore be applied. In addition to training the NMT systems on a parallel corpus in both directions, they also train the pair together on both sets of monolingual corpora. The intuition is simple: through the transitive property, combining the two directions $A \rightarrow B$ and $B \rightarrow A$ allows for autoencoding, $A \rightarrow \hat{A}$, by stacking the two models. The proposed approach has three advantages: (1) it can be combined with any NMT system, (2) monolingual data in both languages can be used and (3) it offers bidirectional translation. The current proposed approach fulfills points (2) and (3), but not (1). Using this approach with RNNSEARCH NMT models, they

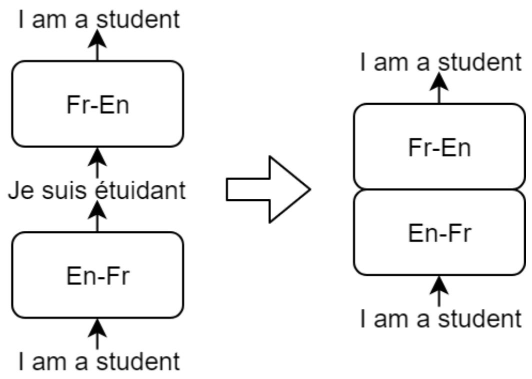


Figure 3.7: The technique used by Cheng et al. [2016] to incorporate monolingual corpus training. When the two translation models are stacked, the intermediate French translation can be considered as latent, and the models can be trained as an autoencoder on only monolingual data. The models are additionally trained on a parallel corpus.

surpass RNNSEARCH trained only on parallel sentences by 4.7 BLEU points on Chinese-English, which is a considerable improvement.

Lastly, Lample et al. [2017] also has strong resemblance with the current work, although they approach NMT with fully unsupervised learning. Just like the current approach, they train one autoencoder for each language, learning to map a sentence to a latent space and then back to itself. They too then bind these latent spaces together (with the RNNPB these are the PB spaces), but the binding mechanism is different. Here, they achieve binding by training the autoencoders to fool a discriminator that tries to guess which language an encoding belongs to. The approach suffers heavily from not using any parallel corpora, although it remains impressive that it achieves effective translation at all using only monolingual datasets.

3.4 Summary

This chapter has given an introduction to the scientific literature surrounding the RNNPB model and the field of machine translation. The RNNPB model was presented as a model that is able to learn multiple temporal patterns simultaneously by associating each with a PB vector. The model's forward and inverse operations were defined in terms of generation and recognition, respectively, and binding was described as a means of aligning the PB vectors of two different RNNPB models. Furthermore, the extension to machine translation where generation and recognition are used in conjunction for two bound RNNPB models was illustrated. The related work demonstrated some applications of the model, described some adaptations of it, and provided a few examples of how it has been applied recently.

For machine translation, a few classical approaches to machine translation and some modern neural models were described. Three early attempts at encoder-decoder architectures were presented, where the sequence-to-sequence model by Sutskever et al. [2014] is the most significant in this thesis. Attention was defined as an extension of the encoder-decoder framework, where the source sentence no longer needed to be compressed into a single context vector, c, allowing instead for the decoder to choose among the source words at each step of decoding. Although this technique is powerful, the RNNPB model cannot benefit from it. The remainder of the related work revolved around attempts at solving specific challenges with NMT models, such as the out-of-vocabulary problem, the problem of adequacy and the challenge of utilizing monolingual data. The present approach may be well-suited for solving the latter two problems, but does not attempt to solve the former, although the techniques presented here could be applicable.

Chapter 4

Model

This chapter documents the exact model that is proposed in this thesis. The model follows rather directly from the related work in the previous chapter, although the RNNPB is updated to utilize LSTM memory cells instead of basic RNN units. This adaptation is motivated by the NMT work which has found gated units to be instrumental in learning to model language on a large scale. Furthermore, details are given for training, particularly in regard to the PB units.

4.1 Architecture

The choice of network architecture is inspired by the sequence-to-sequence NMT architecture proposed in Sutskever et al. [2014]. Particularly, multiple LSTM layers are used in each RNNPB model, which Sutskever et al. found to greatly increase translation performance. Each RNNPB model is equal to the decoder LSTM used by Sutskever et al., but with the addition of PB nodes. The architecture is shown in figure 4.1.

The forward activations of the two RNNPB networks are computed individually with respect to the following equations. The two RNNPB networks are fully independent, apart from the PB binding, and share no weights. The equations below describe the forward propagation of a single RNNPB model. Below, H_1 and H_2 describe the outputs of the two LSTM layers.

$$H_1^{(t)} = \mathrm{LSTM}_1([\mathrm{emb}(X_i^{(t)}); \mathrm{PB}(i)],\ c_1^{(t-1)}) \tag{4.1}$$

$$H_2^{(t)} = \mathrm{LSTM}_2(H_1^{(t)},\ c_2^{(t-1)}) \tag{4.2}$$

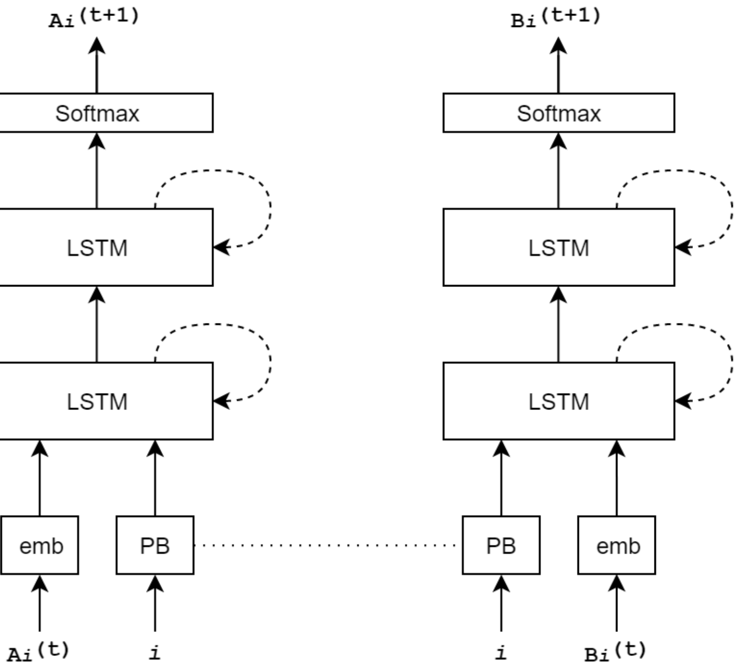


Figure 4.1: The Bound RNNPB model. The model consists of two RNNPB models that are bound, indicated by the dotted line between the PB blocks. Each RNNPB model can be recognized as a conventional RNN language model with two LSTM layers, but with the addition of PB nodes. Both the emb block and PB block are implemented as embedding layers, where the former look up word embeddings and the latter look up PB vectors. In the figure, A and B denote the two languages involved. (A_i, B_i) then denotes the i'th training pair in the corpus of parallel sentences. Each sentence A_i corresponds to a particular PB vector $p_{\mathrm{A}i}$, and likewise for B. The PB blocks take the sequence index, i, and looks up the PB vector, $p_{\mathrm{A}i}$ or $p_{\mathrm{B}i}$, which is responsible for encoding the sequence.

$$O^{(t)} = \text{softmax}(H_2^{(t)}) \tag{4.3}$$

Here, $X_i^{(t)}$ describes the $t'th$ word of sentence i. emb and PB are embedding functions that look up word embeddings and PB vectors, respectively. The word embedding and PB vector are concatenated for the input to the first LSTM layer. Each LSTM layer has a trainable function, LSTM_i, that takes the layer inputs and the layer state from the previous time step, $c_i^{(t-1)}$. The initial state of the network is a zero matrix, $[c_1^{(0)}; c_2^{(0)}] = [\vec{0}; \vec{0}]$. Finally, $O^{(t)}$ denotes the output of the network, which is the probability distribution over the vocabulary.

The definitions for the RNNPB model's three modes of operation are restated below.

Training The training procedure tries to find optimal values for both the network weights and PB vectors. The goal for the network weights is to capture the common dynamics of all the training examples, while the PB vectors specialize further and learn to encode each and every training sample in a one-to-one manner. At a given training iteration, when one training example – or several training examples in the case of mini-batches – are provided to the network, only the PB vector(s) corresponding to these examples are updated, along with the network weights.

Recognition After training is completed, the recognition operation can be used to compute a PB vector from an arbitrary sentence. This operation is exactly equal to the training operation, except that the network weights are not updated. Only the PB nodes are regressed, until the loss is minimized, after which the PB nodes now contain a representation for the sentence.

Generation Generation produces a sentence from a PB vector. As figure 4.2 shows, the PB vector is given to the network at each time step along with the current input. For the first step, the special `<start>` token is given to kick off the computation. Then, at each step the RNNPB model estimates the likely next word. In the experiments done here, beam search is used to search for the most likely sentence, given a PB vector.

4.2 Training

Training is done with teacher forcing for each RNNPB model, as shown in figure 4.2. During training, both the network weights and the current PB vectors are updated. As loss function, the cross-entropy loss between outputs and targets is used, call it L_c.

4.2.1 Binding

Originally, the PB nodes use a custom training rule to incorporate binding. For each training iteration, with a training example (A_i, B_i), model A is trained with example A_i and model B is trained with example B_i. For each of the models, the loss gradients are computed using the BPTT algorithm with regard to all network weights and relevant PB vectors. The network weights are updated normally, using for instance the ordinary gradient descent weight update rule. The PB values are updated using the following update rule, which facilitates binding:

$$p_i = p_i^{\text{old}} - \eta_p \frac{\partial L_c}{\partial p_i} + \lambda \cdot (p_{i\ \text{opposite}}^{\text{old}} - p_i^{\text{old}}) \tag{4.4}$$

p_i^{old} denotes the PB values at the previous training iteration and $p_{i\ \text{opposite}}$ denotes the PB-vector that p_i is bound to; that is, if $p_i = p_{\text{A}i}$, then $p_{i\ \text{opposite}} = p_{\text{B}i}$. λ is a hyperparameter that determines the strength of the PB binding. A separate learning rate is used for the normal network weights and the PB nodes, and η_p describes the learning rate for the PB nodes. The last term in the update equation is responsible for moving the PB vector slightly towards its paired PB vector in the opposite model. Meanwhile, the middle term is responsible for updating the PB vector in the direction of generating the sequence more accurately. After successful training, the update equations should have ensured that the PB vectors fully represent their corresponding sentences, and that each PB vector pair, $(p_{\text{A}i}, p_{\text{B}i})$, has become approximately equal.

The experiments in this thesis exploit the fact that this update equation can be rewritten in a simpler way, in order to train the whole model jointly without implementing custom weight update rules. The binding term of equation 4.4 is equivalent to adding an L2 normalization term to the loss function between the two PB vectors. The joint loss can thus be defined as

$$L = L_{\text{A}c} + L_{\text{B}c} + \alpha(p_{\text{A}i} - p_{\text{B}i})^2 \tag{4.5}$$

The equivalency is apparent from the following calculation. The first equation

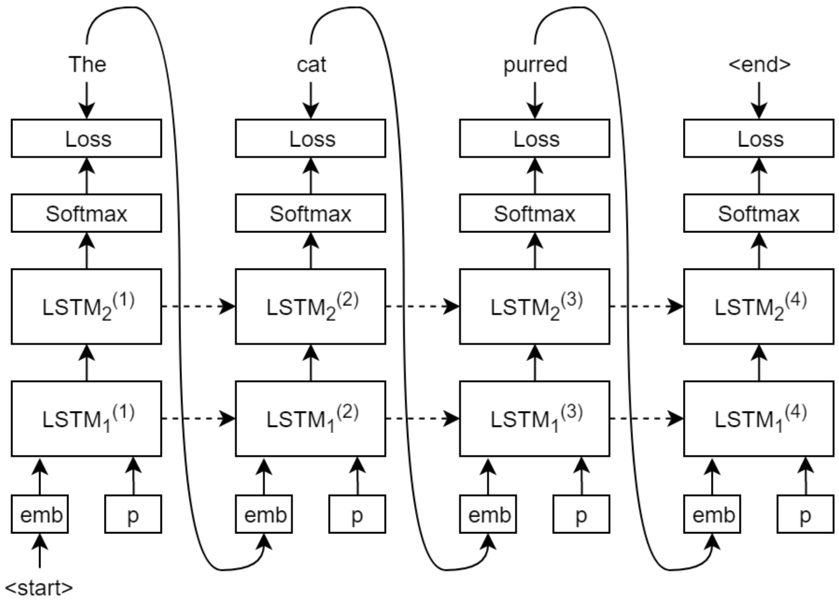


Figure 4.2: Training is performed using teacher forcing. Shown here is a single RNNPB model unrolled over four time steps. The sentence at the top is the teacher signal, which is additionally used as the inputs during training instead of the network prediction (in accordance with the teacher forcing technique). The PB vector, p, is duplicated across the time steps in the figure. The training goal here is for p to adequately represent the given sentence so that it can be regenerated fully, and to this end both p and the network weights are updated. The superscripts on the LSTM layers merely denote the current time step; the layers and all their weights are shared across time (e.g. $\mathrm{LSTM}_1^{(1)} = \mathrm{LSTM}_1^{(2)} = ... = \mathrm{LSTM}_1^{(4)}$). The binding is not shown in this figure.

is the default weight update equation used for gradient descent.

$$\begin{aligned}
p_{\mathrm{A}i} &= p_{\mathrm{A}i}^{\mathrm{old}} - \eta_p \frac{\partial L}{\partial p_{\mathrm{A}i}} \\
&= p_{\mathrm{A}i}^{\mathrm{old}} - \eta_p \frac{\partial}{\partial p_{\mathrm{A}i}} [L_{\mathrm{A}c} + L_{\mathrm{B}c} + \alpha(p_{\mathrm{A}i}^{\mathrm{old}} - p_{\mathrm{B}i}^{\mathrm{old}})^2] \\
&= p_{\mathrm{A}i}^{\mathrm{old}} - \eta_p \frac{\partial}{\partial p_{\mathrm{A}i}} L_{\mathrm{A}c} + \eta \frac{\partial}{\partial p_{\mathrm{A}i}} L_{\mathrm{B}c} + \eta \frac{\partial}{\partial p_{\mathrm{A}i}} \alpha(p_{\mathrm{A}i}^{\mathrm{old}} - p_{\mathrm{B}i}^{\mathrm{old}})^2 \\
&= p_{\mathrm{A}i}^{\mathrm{old}} - \eta_p \frac{\partial}{\partial p_{\mathrm{A}i}} L_{\mathrm{A}c} - 2\eta\alpha(p_{\mathrm{B}i}^{\mathrm{old}} - p_{\mathrm{A}i}^{\mathrm{old}})
\end{aligned} \tag{4.6}$$

which is identical to equation 4.4 when

$$\alpha = \frac{\lambda}{2\eta_p} \tag{4.7}$$

The loss function in equation 4.5 is therefore used in the experiments. Not only is this more intuitive and less error-prone, it allows for the use of more advanced optimizers such as Adagrad or Adam. Hereafter, whenever the term "binding strength" is used, it refers to the value of the α parameter in the L2 loss.

Finally, a note should be made regarding the nature of the one-to-one relation between training examples and PB vectors, which may imply a very high number of PB vectors. All the PB vectors must be saved during training, yielding a space complexity that is linear in the number of training examples. Importantly, the PB vectors can be discarded after training is complete and are no longer necessary during translation. The space complexity of the model itself post-training is therefore not affected by the number of training examples. The goal of training can be considered to shape the PB space, so that the space can represent all sentences. After training is complete, the recognition operation is used to compute a particular PB vector from a sentence. Nevertheless, to achieve this, each and every PB vector must be sufficiently regressed during training, which implies that the complexity of successful convergence likely increases linearly in the number of training examples. This may be a drawback with the proposed model.

4.2.2 Hard binding

As an alternative to the binding mechanism with L2 loss described above, the experiments done here also pursue *hard binding.* That is, a common set of PB units are used for model A and B ($\forall_i \; p_{\mathrm{A}i} = p_{\mathrm{B}i}$). This eliminates the need for the

L2 loss and the binding strength parameter altogether, and the loss can instead be taken simply as $L = L_{\mathrm{A}c} + L_{\mathrm{B}c}$.

4.2.3 Training with monolingual data

RNNPB model A and B can be trained individually on monolingual data without binding by directly optimizing $L_{\mathrm{A}c}$ and $L_{\mathrm{B}c}$, respectively. When training the Bound RNNPB model for translation while utilizing monolingual data, a simple scheme is employed with alternating between bound and unbound training. On a per-batch basis, with probability p_{mono}, train both model A and B as unbound, and else train them as bound by optimizing L. Here, p_{mono} is treated as a hyperparameter.

4.2.4 Regularization

Regularization proves to be instrumental for achieving proper binding and translation in the experiments. Three regularization techniques are investigated. Training with PB noise and intermittently resetting the PB vectors during training are two techniques that are proposed here. The last technique, dropout, is a general machine learning technique that is particularly powerful with sparse datasets. The techniques are described below.

Adding PB noise

This technique investigates adding noise to the sentence representations – the PB vectors – during training. The motivation here is twofold. First, by adding noise, each RNNPB model could learn to associate a larger region in PB space with a given sentence, thus easing recognition. Second, it could make the RNNPB models less sensitive to inaccuracies in the PB vectors, which could aid translating between them. The idea can be expressed as follows:

$$p + \mathcal{N}(0, \sigma) \rightarrow x_1, x_2, ..., x_T \tag{4.8}$$

Here, $\mathcal{N}(0, \sigma)$ describes sampling a Gaussian distribution with zero mean and with σ standard deviation. The noise is randomly generated for each step of training, and σ is treated as a hyperparameter. Table 5.7a shows the result of tests using multiple values for the noise standard deviation.

Intermittent PB reset

Recognition can be viewed as a dynamical process where the PB vectors follow a trajectory through PB space over a number of iterations. It is no help if the training procedure ensures that a given sentence can be represented by *some* point in PB space, if the recognition procedure cannot find it. Motivated by this idea, intermittent PB reset is proposed to incorporate the recognition dynamic into the training procedure. By resetting PB vectors stochastically, the vectors must be regressed anew from the starting point, just as with recognition. This forces the RNNPB model to reevaluate the position of the PB vectors, hopefully so that it readjusts the position to the currently most representative part in space, thus improving the organization of the space as a whole.

The scheme is implemented so that each PB vector is reset with probability p_{reset} at the end of each training epoch, where p_{reset} is regarded as a hyperparameter as usual.

Dropout

Dropout is a common, simple and powerful regularization technique that is often used to reduce overfitting. The technique involves stochastically turning off a subset of the nodes in a given layer at each training step; each node is switched off with a given probability. When dropout is used, it is employed for all of the LSTM layers with a dropout probability given as a hyperparameter.

4.3 Translation

After successful training with binding, translation can be achieved by combining recognition and generation in the two bound models. Define generation as

$$\text{RNNPB}_M(p) \rightarrow x \tag{4.9}$$

for a PB vector, p, sentence, x, and RNNPB model, M. Define the inverse operation of recognition as

$$\text{RNNPB}_M^{-1}(x) \rightarrow p \tag{4.10}$$

Then, translation of a given sentence, a, in language A can be computed as

$$\hat{b} = \text{RNNPB}_B(\text{RNNPB}_A^{-1}(a)) \tag{4.11}$$

In this sense, autoencoding of the same sentence, a, can be considered a special case of translation, using the equation

$$\hat{a} = \text{RNNPB}_A(\text{RNNPB}_A^{-1}(a)) \tag{4.12}$$

This form of autoencoding is investigated in experiment I in chapter 5. Experiment II will investigate a second form of autoencoding using equation 4.11, but where both RNNPB_A and RNNPB_B model the same language. Beam search is always used for $\text{RNNPB}_M(p)$, but not for $\text{RNNPB}_M^{-1}(x)$ which uses teacher forcing.

4.4 Implementation

The RNNPB model and the seq-to-seq model, which is used as a baseline, are implemented in Tensorflow 2.0 beta with python. The new version of Tensorflow is used because it allows writing models imperatively, which makes experimentation easier. It also includes new useful features for processing variable-length sequences (such as sentences) with LSTM layers on the GPU.

The models are implemented in an object-oriented manner, where the RNNPB model and the seq-to-seq model derive from a common NMT base class. This makes it possible to place most of the training, evaluation and translation code in the base class, and only implement the necessary model-specific functions in the sub-classes. This is furthermore useful for comparing the two models as fairly as possible, because most of the code is shared between them. It also makes it very easy to extend the system with a third or fourth model.

An NMT system with a command-line interface has been implemented to allow for easy training and testing of the models. All hyperparameters, datasets and configurations, including the choice of model, are given as command-line arguments, to maximize the quality and reproducibility of the results. Inspired by the Moses SMT system, three different operations are supported. Training is used by specifying the argument `--do_training`, together with the training and validation sets, all the hyperparameters, and a working directory. The model parameters, vocabulary and model weights are then saved to the working directory to ensure that the model can be restored at a later time for testing and querying, which are the two remaining supported operations. Testing and querying of a trained model can then be done very easily, e.g.:

Testing:
```
python nmt.py --do_testing --working_dir=./rnnpbnmt \
        --test_set=test.en
```
Querying:
```
python nmt.py --query="How are you?" \
        --working_dir=./rnnpbnmt
```

All the training jobs for the results given in chapter 5 use run scripts that contain the exact commands that were used, which are published in the Github repository[1], along with the source code, test translations and datasets.

[1]https://github.com/daniesso/master/

Chapter 5

Experiments and Results

This chapter describes the experiments that are conducted and their results. The experiments described here will aim to provide answers to the research questions given in chapter 1.2. Empirically, the feasibility and performance of the model presented in the previous chapter is tested. In the first section, a plan for the experiments is given. The second section provides all details about how the experiments are set up, such as datasets and network hyperparameters. Lastly, the results of the experiments are presented in the third section.

5.1 Experimental Plan

Following directly from the research questions, four experiments will be conducted to answer the following questions:

1. Can the RNNPB model successfully represent language on a large scale?
2. Can the Bound RNNPB model efficiently be employed as an encoder-decoder model for the task of autoencoding?
3. Can the Bound RNNPB model be used for large-scale machine translation?
4. Can the Bound RNNPB model, for the task of large-scale machine translation, benefit from additionally training on monolingual data?

Each of these questions correspond to a separate experiment. Experiment I and II can be seen as stepping stones for achieving large-scale translation. They describe easier yet necessary milestones on the road towards experiment III and IV.

5.1.1 Baseline models

In order to put the results achieved in this chapter into context, a few baseline models have been carefully chosen.

The first baseline model is Moses[1], which is an open source SMT system that is often used as a baseline in NMT literature. Moses can easily and quickly be trained from scratch by providing it with data. Since Moses can be used as is, the results are more objective and not affected by implementation choices made here. The comparison with Moses is especially motivated by the fact that Moses permits training on both parallel datasets and monolingual, and therefore provides a baseline also for experiment IV.

The second baseline model is the seq-to-seq NMT model from Sutskever et al. [2014], which most closely resembles the RNNPB architecture used here. Both the seq-to-seq model and RNNPB have in common that they encode a source sentence into a fixed-size representation, which is then later decompressed into a target sentence. Neither incorporate attention. The configurations of the two models are kept as similar as possible, in order to compare the models accurately.

In addition to providing context for the results, the baseline models play important roles in the development of the RNNPB system. Prior to starting development, Moses is used to evaluate the suitability of the data, including the preprocessing step, by testing whether it's able to learn to translate well using the data. Thus, possible data-related causes for failures are eliminated early on. Secondly, the seq-to-seq model is implemented and verified prior to implementing the RNNPB system. This ensures that the end-to-end NMT system – with preprocessing, translation and testing – is working properly, so as to differentiate problems with the RNNPB model with other implementation errors.

5.1.2 Experiments

The four experiments are now described in detail. One or more goals are defined for each of the experiments, embodying specifically what the experiments try to achieve. For the sake of structure, each experiment is also divided into parts. Each part either trains a new model from scratch (this is the default case), or investigates some property of a previously trained model.

Experiment I

For the Bound RNNPB model to be used for translation, a single RNNPB model must be able to compute representations for both seen (trained) and unseen sentences. A single RRNPB model is therefore trained on a monolingual dataset

[1]Moses is documented at `http://www.statmt.org/moses/` and can be downloaded from `https://github.com/moses-smt/mosesdecoder`.

in this experiment. In this context, learning to compute these sentence representations can be viewed as a form of autoencoding where the same model both produces the representation (encoding) and regenerates the sentence from it (decoding). In general, autoencoding can be viewed as a simpler version of the task of translation; in both cases the model learns to compute a fixed-size vector representation of the semantics of a source sentence, and subsequently produces a target sentence from it. Therefore, the work here considers the autoencoding performance as an upper bound for the translation performance that could theoretically be achieved, and thus examines the autoencoding performance in two different configurations in experiment I and II. In the spirit of considering autoencoding as a special case of translation, the autoencoding quality will be tested in terms of BLEU score in the exact same manner as for later translation.

The goals of this experiment can be stated as:

G11. Ascertain whether a single RNNPB model can successfully be trained on a large scale.

G12. Ascertain whether a single RNNPB model can successfully generalize to achieve autoencoding for untrained sentences.

For goal G11, a high accuracy for *trained* sentences should eventually be achieved. Goal G12 tests the properties of generation and recognition for *untrained* sentences. It is a nontrivial question whether recognition is able to compute PB vectors for such sentences. Since the recognition operation continuously examines the model's own forward operation, and since the same RNNPB model is used for encoding and decoding in this experiment, successful recognition entails successful autoencoding. For this goal, the autoencoding BLEU score on a test set will be reported.

Experiment I has two parts. In part 1, the RNNPB and the seq-to-seq models are trained on the easy subsets, and BLEU scores for the test sets are reported. Part 2 offers a qualitative examination into how recognized PB vectors relate to PB vectors obtained during training for the same training sentences.

For both experiment I and II, only the seq-to-seq model is used as a baseline. Moses is not used at this point, seeing as it is trivial for a SMT system to achieve autoencoding (it could simply copy the source sentence).

Experiment II

This experiment extends the autoencoding task in the previous experiment, using a separate RNNPB model for encoding and decoding. To facilitate this operation, binding of the two models must now be employed. The goals of this experiment are stated as follows.

G21. Ascertain whether the training PB vectors of the two bound models can be successfully aligned during training.

G22. Ascertain whether autoencoding can be achieved for untrained sentences when the recognition and generation operations are used in opposite models.

G23. Develop the model in terms of training techniques and model hyperparameters as to maximize autoencoding performance.

Goals one and two investigate how well the semantic spaces – i.e. the PB spaces – of the two models are aligned during training. The training objective aims to align the spaces at the locations of the training examples (G21), and so it is additionally necessary to examine the properties of the space for unseen sentences (G22). In this experiment, there are two independent models functioning as encoder and decoder, and thus in contrast to experiment I, successful recognition does not automatically entail successful autoencoding.

Goal G23 captures the necessary development of the model in order to achieve good autoencoding performance. This experiment investigates numerous techniques and parameters, while evaluating the performance empirically at each step. The work done in this experiment tries to justify all the major choices that are done in adapting the model. Subsequently, the best configuration of the model found in experiment II is used in experiment III and IV.

Experiment II has five parts, where the first four each investigate an aspect of the RNNPB model and the fifth reports final autoencoding results. Each part builds on the previous, using the best configuration that was found.

Part 1 In this part, the effect of the binding strength parameter, α, is investigated. Hard binding is also tested as an alternative to the soft binding with L2 loss. The part focuses on how the strength of the binding affects the quality of autoencoding and how the PB spaces are affected by the binding.

Part 2 In the second part, different numbers of PB units are tested. One notable difference between the seq-to-seq model and the RNNPB model used in part 1 is the size of the representation space. For the seq-to-seq model, the representation dimensionality is $2 \cdot$ units-per-LSTM-layer $\cdot$ number-of-LSTM-layers $= 2 \cdot 256 \cdot 2 = 1024$, while the Bound RNNPB models have representation dimensionality equal to the number of PB nodes (128 in part 1). The reader might recall that Sutskever et al. hypothesized that the increased performance of their deep LSTMs in the seq-to-seq model was owed to a larger size of the representation space. Therefore, different numbers of PB nodes are tested in this part to determine whether the RNNPB too can benefit from a larger representation space.

Part 3 This part examines the three regularization techniques introduced in chapter 4 and investigates whether they can improve the quality of the autoencoding. The techniques are used individually first, then intermittent PB reset and dropout are combined.

Part 4 Part four studies the effect of the number of recognition iterations. When two bound models are employed, it is not necessarily beneficial to continue recognition until the loss is as small as possible. For instance, if the gradients become very small, recognition may move the PB vectors by a large amount into a part of PB space that is poorly bound between the two models. Even if the recognition loss is decreasing, too many iterations may be detrimental for translation. Three schemes are tested, based on either a fixed amount of iterations or a form of early stopping.

Part 5 In the last part, the autoencoding results are reported for all the easy subsets, using the best configuration from the preceding parts.

Experiment III

This experiment tests the performance of the proposed model for translation, compared to Moses and seq-to-seq. The goal of this experiment is stated as

G31. Investigate and measure the translation performance of the RNNPB model.

In the first of two parts, experiment III picks up where experiment II left off, with the same configurations and datasets, but with English-to-German translation instead of autoencoding. The goal here is to ascertain whether the use of the model can be extended from autoencoding to translation. Then, in the second part, translation is tested with the much more difficult WMT dataset. In both parts, results are given for all three models: RNNPB, seq-to-seq and Moses. Since the WMT dataset is more complex, a larger configuration in terms of network size is also tested for the RNNPB and seq-to-seq models.

Experiment IV

In this experiment, the RNNPB model and Moses are tested with monolingual training. This is done as described in section 4.2.3, with a mixing parameter, p_{mono}. The goal of this experiment is stated as follows:

G41. Determine and measure if and to which degree the Bound RNNPB model can benefit from monolingual data for machine translation.

This experiment also has two parts. First, the final autoencoding configuration from experiment II is used to find a good value for the p_{mono} parameter, and to investigate the feasibility of adding monolingual training for the RNNPB model. In common with experiment II, the Bound RNNPB model is trained for autoencoding with the easy50000 subset. In addition, non-overlapping sentences from easy-full is used for monolingual training. In part two, the Bound RNNPB model and Moses is trained for English to German translation, using easy50000 and the WMT dataset, with additional monolingual training. Since the seq-to-seq model cannot utilize monolingual data, it will not be used in this experiment.

5.2 Experimental Setup

The NMT models will be trained using early stopping. With this scheme, training is only stopped when improvement on a validation set ceases. This ensures that the models generalize as well as possible, as opposed to overfitting on the training data. The decision to use early stopping means that the models may be trained for a different number of steps. This is a conscious choice that's been made to ensure the models are judged by their optimal performance. In order to nuance the results with the amount of computation used, the training times are reported and compared.

All models will be trained, developed and tested utilizing the standard dataset partitioning scheme with a training set, validation set and test set. The validation set is used for early stopping for the RNNPB and seq-to-seq models, and for Moses it's used for tuning. The same data is used for all models, as described below.

The next subsections now proceed to describe the datasets, preprocessing and evaluation, and model parameters used in the experiments. The subsections give all the details that are necessary to carry out these experiments.

5.2.1 Datasets

The first dataset is a parallel English-to-German dataset[2] that was originally intended for learning a foreign language (English or German). As a result, many of the sentences are simple and regular. For instance, the names Tom and Mary are consistently used as proxies for male and female persons, while few other names are used. This dataset is ideal for these experiments, where an NMT model that isn't guaranteed to work well is developed, because the dataset isn't too challenging. At the same time, the dataset contains thousands of sentences and unique words, as summarized in table 5.1. The dataset is thus sufficient

[2]It can be downloaded from http://www.manythings.org/anki/. It is also provided along with the source code.

Dataset	Sentences	Unique English words	Unique German words	Average words per sentence
Easy20000	20 000	1290	514	8.9
Easy50000	50 000	2562	1132	9.3
Easy-full	190 624	17 745	36 706	9.8

Table 5.1: The easy English-German dataset and the subsets that have been created from it.

to ascertain the RNNPB model's capability of doing machine translation. This dataset is hereafter referred to as "easy".

Furthermore, a data selection technique is used to create subsets of the dataset that limit the size of the vocabularies. Since large vocabularies are demanding, both with regard to translation performance and training time, the subsets easy20000 and easy50000 were created, containing 20000 and 50000 sentences respectively. The subsets are selected as follows. First, the frequency of each word in the complete dataset is calculated. Then, the sentences are ranked according to their least frequent word. The subsets are then created by greedily selecting the topmost 20000 and 50000 sentences (containing the sentences with the most frequent words). The subsets are chosen based on only the target-side word frequencies. The full dataset is referred to as easy-full.

Experiment I and II use only the English sentences when doing autoencoding. In this case, duplicate English sentences are removed before the subsets are created to avoid any overlap between training, validation and test sets.

Each of the three sets (easy-full, easy20000 and easy50000) are further divided into training, validation and test sets. The validation sets and test sets have sizes of 256 and 2000, respectively. The reason for using such a small validation set is that inference is very expensive for the RNNPB model, because it requires recognition, and a large validation set would therefore slow down training.

In experiment III and IV, a second set of datasets are used. These are taken from the WMT18 translation task[3]. From this task, the parallel News Commentary and monolingual News Commentary datasets are selected for training. From this data, a parallel corpus of 200 000 sentences and two independent monolingual corpora of 140 000 sentences are selected. The default WMT dataset for testing is used, News-test 2018, while News-test 2017 is used as validation set. The characteristics of this data are given in table 5.2. As the table shows, this data contains substantially longer sentences and includes a much larger vocabulary. The maximum vocabulary sizes that are used for the different experiments are given in subsection 5.2.4.

[3]The datasets can be downloaded from here http://statmt.org/wmt18/translation-task.html.

Dataset	Sentences	Unique English words	Unique German words	Average words per sentence
Parallel N.C.	200 000	74 575	156 992	27.3
Monol. N.C.	140 000	65 108	127 841	26.7
News-test 17	3004	9742	13 029	22.9
News-test 18	2998	10 374	13 342	23.9

Table 5.2: The WMT English-German datasets that are used for training, validation and testing.

5.2.2 Preprocessing and evaluation

The only preprocessing of the data is tokenization, using the standard Moses tokenizer[4], in accordance with Sutskever et al. [2014]. This mainly entails separating words and special characters with a space. The sentences retain their casing, even though this potentially doubles the vocabulary size. With this choice, the NMT models automatically learn proper capitalization. The tokenized data is used to train all the models.

Four special symbols are defined for the NMT models: `<start>`, `<end>`, `<unk>` and `<pad>`. Each sentence is prefixed with `<start>` and suffixed with `<end>`. If maximum limits are placed on the size of the vocabularies, the least frequent words that exceed the vocabulary limit are replaced with `<unk>`. Lastly, since stochastic gradient descent with mini-batches is used, all sentences in a batch must have equal length, and therefore the sentences are padded with `<pad>` to the length of the longest sentence in the batch. During training, the loss values corresponding to this padding are disregarded.

During training, the perplexity is used to evaluate the validation set after each epoch. The same mini-batch size as for training is used for the perplexity evaluation, for efficiency. The perplexity is only calculated in one direction, from source to target language, i.e. from model A to model B, even though the RNNPB is capable of bidirectional translation. First, PB vectors are computed through recognition in model A, then the perplexity is calculated in model B using the acquired PB vectors, by assessing the likelihood of the ground-truth target sentence.

Actual translations, such as for testing, use mini-batches of size 1 and do generation in model B using beam search. It should be noted that testing could be significantly sped up by doing recognition in parallel, using a larger mini-batch size. Even though beam search can only translate one sentence at the time, the majority of the time it takes to compute a translation is incurred by the recognition procedure. This optimization was however not pursued here. The

[4]The `tokenizer.perl` script is used.

results are evaluated after training by calculating the translation BLEU score on a test set. Here, the appropriate script, `multi-bleu.pl`, from Moses is used, also in accordance with Sutskever et al. [2014]. The BLEU score is measured on tokenized predictions and reference translations.

5.2.3 Training

Stochastic gradient descent with mini-batches is used for training. The cross-entropy loss for a given sentence is summed over time steps, i.e. for all words in the sentence. The sum was chosen instead of the mean because the mean isn't calculated properly when padding is used[5]. Then, the final training loss is taken as the mean of the cross-entropy losses in the batch. Here, using the mean is a popular choice because it ensures that the size of the loss is independent from the size of the mini-batches.

The network weights are initialized from a uniform distribution on some interval, where this interval is chosen with respect to the Glorot (a.k.a Xavier) initialization method. This is the default Tensorflow behavior. However, all PB vectors are specifically initialized to the zero-vector. This choice is partly motivated by ensuring that the PB vectors self-organize in space with respect to their semantic meaning only, without the initialization having any bearing. It is also likely beneficial that the recognition procedure starts with PB vectors equal to zero, so that the starting point is in the center of the space, thus avoiding bias.

The seq-to-seq model is implemented using the same values for the mutual parameters. For this model, the source sentences are reversed, which was found beneficial in the original work. Special care was taken to ensure that the hidden states computed by the Encoder model corresponds to the end of the sentences, *not* including the padding. Otherwise, the length of the padding would affect the final representation. It was not possible to achieve this efficiently with GPU computation prior to Tensorflow 2.0, which was a major motivator for adopting the new version, even though it was in alpha during the spring of 2019.

Adam optimization is used for both NMT models, and for the RNPPB model, it's used for both training and recognition. Adam was chosen because it yielded superior results and faster training for the RNNPB model. An important note should be made regarding this optimizer. While it's not in the scope of this thesis to describe different optimization algorithms, the fact that Adam maintains individual momentum terms for each network weight significantly affects the experiments conducted here. The reason for this is that all network weights are updated for each step of training, regardless of whether the weights are active

[5]The mean would be calculated by summing the loss over time steps and then dividing by the sentence length, including padding. If a mini-batch contains a long sentence, the mean loss would become very small.

with the current training batch. Therefore, all of the word embeddings and all of the PB vectors would be updated even though most have gradients equal to zero for a given batch. The first time these experiments were conducted, training took up to ten times as long due to this behavior. The experiments were then repeated with an implementation of Adam (the LazyAdam optimizer) that only updates the word embeddings and PB vectors that are active for the current training batch.

Some implementation details, such as the Adam and LSTM implementation, are given in the appendix. Lastly, training is done on a single Tesla V100 GPU, while inference (testing) is done on the CPU. Moses is trained on the author's laptop on the CPU.

5.2.4 Parameters

The values used for all parameters in the experiments are now presented. For even more detail, many of the run scripts that define each experiment are given in the appendix. While the experiments test different values for some parameters, some baseline parameters can be described that are the basis for all the configurations. These parameters are:

- LSTM layers: 2
- LSTM units: 256
- Embedding size: 128
- Learning rate: 0.001
- PB learning rate*: 0.01
- Batch size: 64
- Translation ratio: 1.5
- Gradient clip: 1.0
- Beam size: 10
- Num. PB units*: 128 / 1024
- Recognition max. iterations*: 500 / 100
- Recognition early stopping steps*: 3
- Recognition epsilon*: 0.0001
- Dropout*: 0
- σ*: 0
- p_{reset}*: 0

The parameters marked with an asterisk are only relevant for the RNNPB model and not for the seq-to-seq model. Parameters without an asterisk have equal values for both models. The number of PB units is 128 for experiment I and parts of experiment II, then 1024 for the remainder of the experiments. The maximum iterations for recognition uses 500 iterations for experiment I then 100 for experiment II-IV.

The translation ratio describes an often-used heuristic for the maximum length of a translation with respect to the length of the source sentence. In order to avoid incurring too large a BLEU penalty, unrealistically long translations should be avoided, and the translation is therefore cut off at a certain length, even if `<end>` hasn't been emitted by the model.

Recognition has three parameters here that dictate how many iterations of computation are used. In order to avoid computations that don't converge, a maximum limit of iterations is imposed. Furthermore, early stopping is used so that recognition terminates when the loss ceases to decrease significantly. With the scheme used here, recognition is stopped after k steps where the loss hasn't decreased by at least ϵ.

The configurations for each experiment is now described further.

Experiment I The baseline configuration is used as is. As stated, 128 PB units and 500 recognition iterations are used. In addition, the number of early stopping steps for training is 10. That is, training is terminated if the validation perplexity does not improve after 10 steps.

Experiment II This experiment investigates multiple parameters to find values that perform well. The experiment states which values are used for the relevant parameters at each step. At the beginning of the experiment, the parameters that deserve mention are:

- Num. PB units: 128
- Recognition max. iterations: 100
- Early stopping steps: 20
- Choice of binding mechanism: not chosen yet

At the end of experiment II, these parameters are:

- Num. PB units: 1024
- Recognition max. iterations: 100
- Early stopping steps: 20
- Choice of binding mechanism: Hard binding

- Dropout: 0.40
- p_{reset}: 0.10

Experiment III Experiment III uses the same parameters as for the end of experiment II. However, since the vocabulary sizes for this experiment are larger, limits on the size of the vocabularies are now imposed. These limits are, for both part 1 and 2:

- English vocabulary limit: 30 000
- German vocabulary limit: 30 000

In addition, since the WMT datasets are much larger, these use only 10 early stopping steps for training instead of 20.

For the WMT dataset, a large configuration of the RNNPB and seq-to-seq models is additionally tested. This large configuration makes the following changes:

- Embedding size: 512
- LSTM units: 1024
- LSTM layers: 4

Experiment IV Experiment IV uses the same parameters as experiment III, using its standard, non-large configuration. In part 1, experiment IV investigates different values for the p_{mono} parameter, which dictates the ratio with which monolingual training should be incorporated in the RNNPB model. In part 2, $p_{\text{mono}} = 0.40$ is used.

5.3 Results

In this section, the results for the four experiments are presented. In accordance with the plan given in section 5.1, experiment I and II test autoencoding in English, before experiment III and IV attempt translation. The experiments are exploratory in nature, investigating some property of the RNNPB model and building on prior results at each step. This section comments on the results as they are presented, though a full evaluation is reserved for chapter 6.

5.3.1 Experiment I

In this experiment, autoencoding is tested with a single RNNPB model. With the formalism introduced in chapter 4, this can be expressed as learning to compute

$$\hat{a} = \text{RNNPB}_A(\text{RNNPB}_A^{-1}(a)) \tag{5.1}$$

where the goal is for $\hat{a}$ to be equal to a. Only the English sentences from the easy subsets are used here.

Part 1: Autoencoding with the easy subsets

The autoencoding results for the seq-to-seq model and the RNNPB model are shown in table 5.3. These results summarize both testing and training of the two models. The results show that both models are able to autoencode very well. This is apparent from the high BLEU score; a score of 100 describes identical autoencoding (or translation), to which these results are close. The results show that the RNNPB achieves consistently better BLEU scores compared to the seq-to-seq model, for the experiment conducted here.

The models are trained for a similar number of epochs for the first two subsets, while the RNNPB model is trained for 2.8 times as many epochs for the easy-full dataset compared to the seq-to-seq model. The models have largely comparable training times, apart from for easy20000. Since this dataset is so small, the overhead caused by the RNNPB's recognition operation – which is used for testing the validation perplexity – has a large impact. For the same reason, the testing time is in total 55.4 times longer for the RNNPB model compared to the seq-to-seq model.

Part 2: Comparing trained and recognized PBs

Figure 5.1 shows a comparison between learned PB vectors and PB vectors recognized from the same sentences. The plot shows apparent correlation between trained and recognized PB vectors, although the distances between corresponding points are rather large. For instance, for many of the points in the plot, the closest neighboring point is not the one with the same color. Any such deviation between the recognized and trained PB vectors may become problematic when translations should be generated using two bound RNNPB models, because the learning objective with binding only attempts to align the semantics of the two models at the points corresponding to the training PB vectors.

5.3.2 Experiment II

This experiment tests autoencoding with the Bound RNNPB model. This can be expressed as learning to compute

$$\hat{a} = \mathrm{RNNPB}_{\hat{A}}(\mathrm{RNNPB}_{A}^{-1}(a)) \tag{5.2}$$

where the goal is for $\hat{a}$ to be equal to a, and A and $\hat{A}$ are two separate RNNPB models that model the same language. In common with experiment I, English sentences from the easy dataset are used.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB	1.01	1.01	99.43	41	2745 s	62 438 s
Seq-to-seq	1.01	1.08	93.51	31	432.5 s	889.4 s

(a) Using the easy20000 subset.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB	1.02	1.01	98.77	43	4287 s	63 016 s
Seq-to-seq	1.00	1.04	95.36	37	2026 s	1143 s

(b) Using the easy50000 subset.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB	1.05	1.09	94.44	47	10 935 s	67 527 s
Seq-to-seq	1.02	2.61	84.96	17	4085 s	1452 s

(c) Using the complete dataset.

Table 5.3: Experiment I – part 1: Autoencoding with a single RNNPB model on three datasets, compared to the seq-to-seq model. The final train perplexity denotes the perplexity of the training set at the last training epoch, while the best validation perplexity denotes the best perplexity achieved on the validation set at any epoch during training. Test time is the time elapsed processing the test set, which yields the test BLEU score. The tests are done with the network weights corresponding to the best validation perplexity. Lower perplexities scores and higher BLEU scores are better.

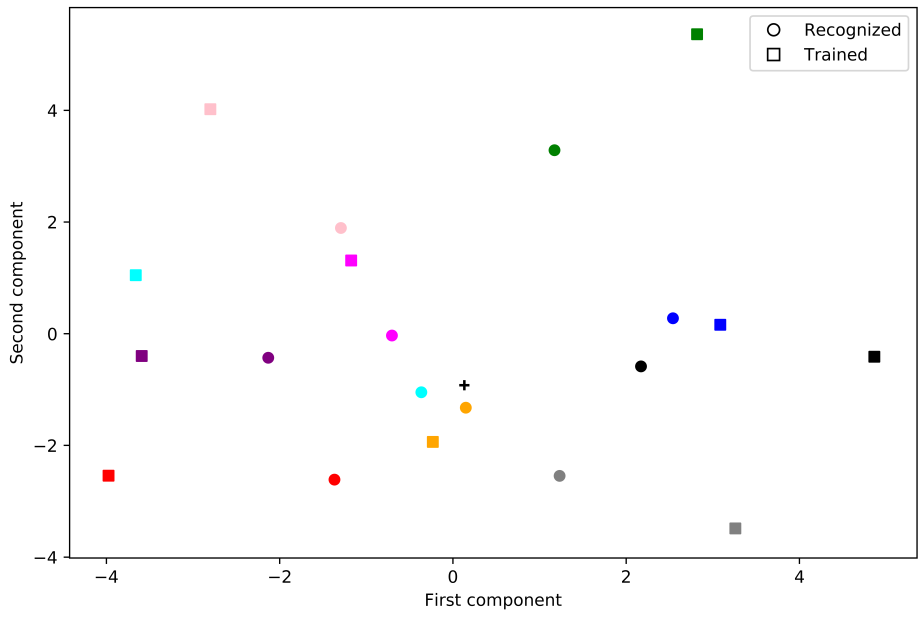


Figure 5.1: Experiment I – part 2: Comparison of PB vectors obtained during training and PB vectors recognized from the same sentences after training. To make this plot, the 64-dimensional PB vectors have been reduced to two dimensions using Principal Component Analysis (PCA). The sentences used were arbitrarily chosen as the first ten sentences in the training set for the easy20000 subset. The correspondences between trained and recognized PBs are color coded. The sentences could be generated with near perfect confidence with both sets of PB vectors, with average perplexities of 1.023 for training PBs and 1.001 for recognized PBs. This implies that multiple points in PB space may serve to encode the sentences.

Part 1 to 4 use the easy50000 subset to investigate the properties of the Bound RNNPB. This choice is motivated by the dataset having a significant number of sentences, while still limiting the size of the vocabulary, which enables carrying out multiple experiments much faster. In part 5, after concluding with the optimal configurations, results are also given for the easy20000 and easy-full sets.

Part 1: Binding strength

Table 5.4 shows the result of training with different values for the binding strength parameter, α. Additionally, hard binding is tested, where the two RNNPB models share PB vectors during training as an alternative to soft binding. The table shows that all configurations train successfully, which is apparent from the low final training perplexity. The progress of the training is further illustrated in figure 5.2. However, all but four configurations fail at improving on the validation perplexities after the first epoch. Note that this is the first validation measurement, and therefore it is apparent that these configurations fail completely at generalizing for autoencoding. Among the four configurations that improve on the initial validation perplexity, there seems to be a trend where stronger binding gives both better validation perplexities and test BLEU scores. Since hard binding gives the best BLEU score by a large margin, it is adopted in the remainder of these experiments.

Figure 5.3 examines how well binding has been achieved for the different binding strengths. The plots shows only sporadic binding for $\alpha = 0.0001$ and $\alpha = 0.001$, while for $\alpha \in [0.01, 100]$, the plot shows consistent binding. The results from the plots are reinforced numerically in table 5.5, which compares euclidean distances between bound and unbound PB vectors. Disregarding hard binding, the configuration with $\alpha = 10.0$ has the lowest bound-unbound ratio, which indicates a better binding. This configuration also has the highest test BLEU score, second to hard binding.

Part 2: Number of PB nodes

Table 5.6 shows the results of using different numbers of PB nodes. The table shows a monotonically increasing BLEU score with respect to the number of PB nodes. The best validation perplexity is also decreasing. In the remainder of the experiments, 1024 PB nodes are therefore used. All the configurations reach their best validation perplexity in the ninth epoch. At the final training epoch, the validation perplexity has become very large despite a decreasing training perplexity, which is indicative of overfitting.

α	Final train perplexity	Final val. perplexity	Best val. perplexity	Test BLEU	Best epoch	Train epochs
0.0001	1.48	631 599	34.26	-	1	21
0.001	1.50	41 712	36.06	-	1	21
0.01	1.47	807.6	35.77	-	1	21
0.1	1.03	388.43	22.41	8.00	13	33
1.0	1.10	172.2	31.88	5.97	9	29
10.0	1.14	49.95	13.42	12.52	15	35
100.0	2.96	103.2	35.07	-	1	21
hard	1.05	429.3	5.56	23.49	9	29

Table 5.4: Experiment II – part 1: Autoencoding with different binding strengths. In the last row, hard binding is employed. The epoch that achieved the best validation perplexity is given in the second to last column. This column shows that training did not succeed for half of the configurations; after the first epoch, the validation perplexities deteriorate. Since early stopping is employed, the number of training epochs exceed the best epochs by 20, as seen in the last two columns. The BLEU score was not measured for the configurations that were deemed failures.

α	Mean bound distance	Mean distance*	Ratio
0.0001	5.16	5.16	100 %
0.001	4.56	4.91	93 %
0.01	2.08	4.00	52 %
0.1	0.365	4.40	8.2 %
1.0	0.144	2.89	5.0 %
10.0	0.0607	1.33	4.6 %
100.0	0.106	0.453	23 %
hard	0.000	10.4	0 %

Table 5.5: Experiment II – part 1: Distances between bound and unbound PB vectors for different binding strengths. The mean distance between bound PB pairs versus the mean distance between all PBs, for different binding strengths. The rightmost column gives the ratio between column two and three, and can be considered as a measure for how good the binding is. *Because of high time complexity, the mean distance is sampled as the mean distance between $N = 10000$ samples from model A and N samples from model B.

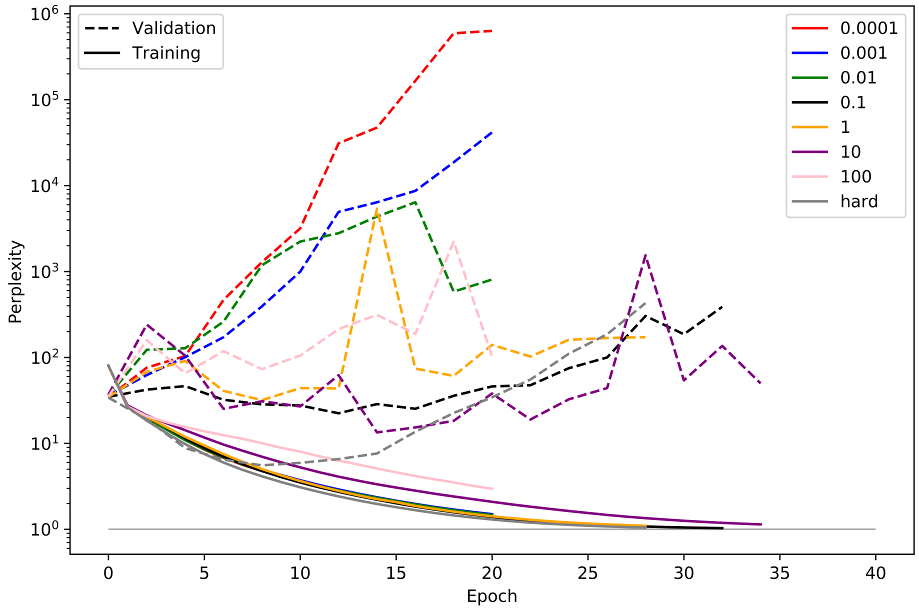
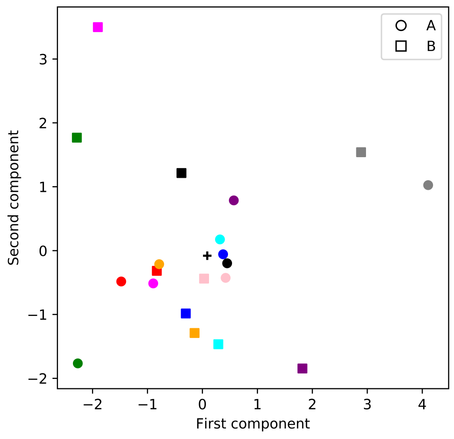


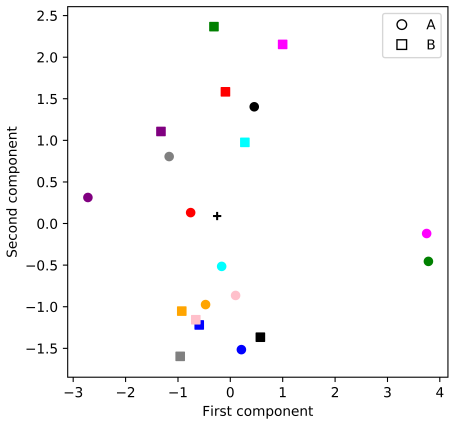
Figure 5.2: Experiment II – part 1: Perplexities of training and validation sets during training for different binding strengths. The plot is from the same results as table 5.4. The $y = 1$ line is the asymptote describing the best attainable perplexity.

Num PB	Final train perplexity	Final val. perplexity	Best val. perplexity	Test BLEU	Train epochs
64	1.17	98.63	6.21	19.83	29
128	1.05	429.3	5.56	23.49	29
256	1.02	216.4	5.53	27.87	29
512	1.01	378.9	4.96	29.90	29
1024	1.01	428.2	5.00	30.45	29

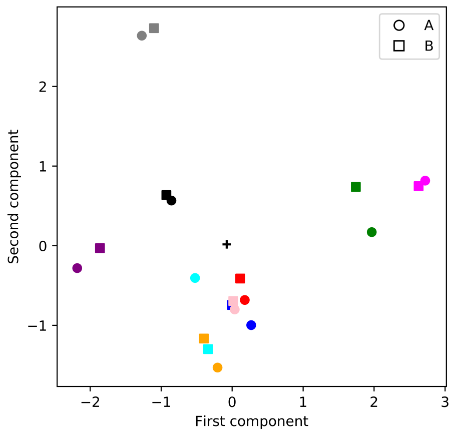
Table 5.6: Experiment II – part 2: Number of PB nodes and the effect on autoencoding quality. Since the PB nodes are tasked with encoding whole sentences, a higher number of nodes may be beneficial. All configurations use hard binding, and the second row with 128 PB nodes is copied from table 5.4. All configurations achieved their best validation perplexity in their ninth training epoch, and hence the training was terminated after 29 epochs.



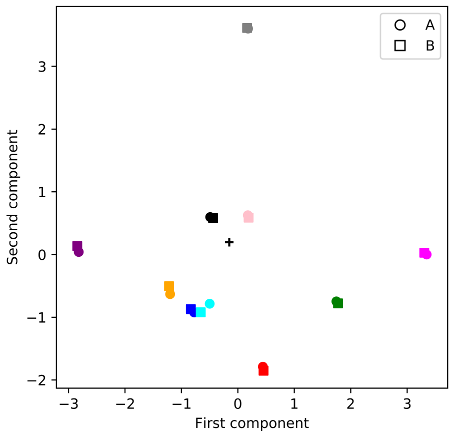
(a) $\alpha = 0.0001$



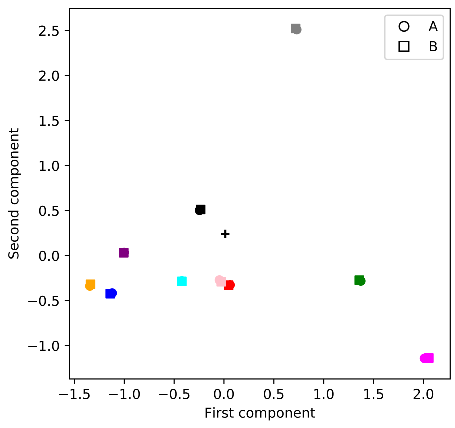
(b) $\alpha = 0.001$



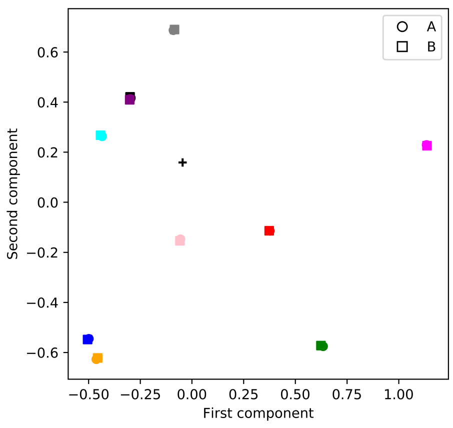
(c) $\alpha = 0.01$



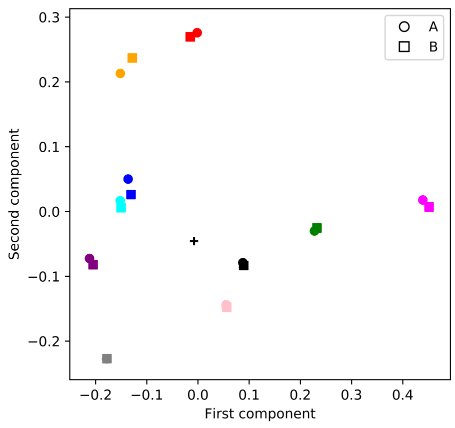
(d) $\alpha = 0.1$



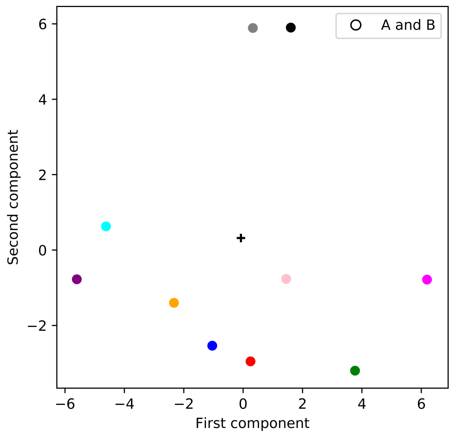
(e) $\alpha = 1$



(f) $\alpha = 10$



(g) $\alpha = 100$



(h) Hard binding

Figure 5.3: Experiment II – part 1: Plots of PB Binding with different binding strengths. As earlier, the PB vectors have been mapped to two dimensions using PCA, and the first ten training examples have been used. The final weights at the end of training were used, see table 5.4. The binding can be seen to be most consistent in figure 5.3e. A plot of hard binding is included for completeness, although it is mostly trivial.

Part 3: Regularization

The three regularization techniques are tested in table 5.7. The best results for PB noise, intermittent PB reset and dropout differ from the baseline BLEU score of 30.45 with -0.48, 28.11 and 5.25 points, respectively. Thus, intermittent PB reset gives the best results by a large margin, where the baseline score is nearly doubled. PB noise decreases the BLEU score with all the configurations that are tested and is therefore not considered any further in these experiments. It can also be seen that PB reset increases the final training perplexity drastically. Dropout has the same effect, but to a lesser extent. Both techniques also increase the number of training epochs, where PB reset is the most aggressive in this regard as well.

The two techniques are combined in table 5.8. These results show that the techniques complement each other well, increasing the BLEU score by an additional 20.83 points at the most, for a new best for the RNNPB model of 79.38 BLEU points. This score was achieved by the $p_{\text{reset}} = 0.10$, dropout $= 0.40$ configuration, which is used in the subsequent experiments[6]. The increased score now comes at a cost of the number of training epochs exceeding 100.

Part 4: Recognition iterations

Figure 5.4 shows the effect of the number of recognition iterations on the translation perplexity. That is, for each step of recognition in model A, the current PB vector is evaluated in model B compared to the reference translation, in terms of perplexity. This provides a means of evaluating the quality of the PB vectors during the progression of recognition. In this context, translation is used in the technical sense to denote the mechanism of combined recognition and generation in two separate RNNPB models, even though this experiment focuses on autoencoding. The plot shows that both too many and too few iterations can be suboptimal. Naturally, the perplexity of the computed translation is not normally available, so one cannot simply terminate the recognition upon reaching the minima in the figure. It is therefore useful to consider different schemes for determining the right number of iterations. Figure 5.5 examines whether the size of the gradients imply anything about when recognition has reached a minimum in the perplexity landscape. One visible trend in the figure is that the intersections between the gradients and the $y = 1$ line seem to correlate with the perplexity minima. A scheme where recognition is terminated when the L2 norm of the gradients decreases below 1 is therefore considered next.

The recognition procedure is tested with three different schemes in table 5.9, using the trained model from table 5.8 with $p_{\text{reset}} = 0.20$ and dropout $= 0.30$.

[6]Apart from the next section, which for circumstantial reasons uses $p_{\text{reset}} = 0.20$, dropout $= 0.30$, which looked like the most promising configuration at the time.

σ	Final train perplexity	Final val. perplexity	Best val. perplexity	Test BLEU	Train epochs
0	1.01	428.2	5.00	30.45	29
0.00001	1.01	1289	5.02	29.97	29
0.0001	1.01	2009	5.23	28.12	29
0.001	1.01	3253	5.76	21.10	27
0.01	1.01	876.5	5.31	29.22	29

(a) Adding noise to the PB vector during training.

p_{reset}	Final train perplexity	Final val. perplexity	Best val. perplexity	Test BLEU	Train epochs
0	1.01	428.2	5.00	30.45	29
0.005	2.13	32.72	5.06	25.93	27
0.01	1.57	13.46	5.21	25.63	27
0.05	2.23	6.11	5.08	30.07	29
0.10	2.53	6.10	4.61	50.69	43
0.20	3.23	5.82	3.73	58.56	43
0.30	3.81	7.42	3.44	57.16	45
0.40	4.31	5.23	3.68	51.48	45
0.50	4.96	7.92	4.72	43.89	41

(b) Resetting the PB vectors intermittently during training.

Dropout	Final train perplexity	Final val. perplexity	Best val. perplexity	Test BLEU	Train epochs
0	1.01	428.2	5.00	30.45	29
0.05	1.04	81.07	4.65	32.34	29
0.10	1.08	61.76	4.92	26.19	27
0.15	1.16	35.14	5.20	35.70	29
0.20	1.16	35.14	5.19	30.21	29
0.30	1.30	26.26	4.75	33.30	31
0.40	1.55	32.93	4.23	32.33	31
0.50	2.03	8.37	5.42	28.75	33

(c) Adding dropout in all LSTM layers during training.

Table 5.7: Experiment II – part 3: Three regularization techniques. All techniques are only enabled during training and turned off during inference. The first row of each table is copied from the 1024 PB configuration in table 5.6.

p_{reset}	Dropout	Final train perplexity	Final val. perplexity	Best val. perplexity	Test BLEU	Train epochs
0.10	0.10	2.48	2.56	2.42	67.83	63
0.10	0.20	2.50	2.52	2.02	72.86	87
0.10	0.30	2.83	2.64	2.04	73.51	69
0.10	0.40	2.90	3.92	2.14	79.39	109
0.10	0.50	3.12	2.15	1.86	75.88	119
0.20	0.10	3.26	3.79	2.70	69.29	61
0.20	0.20	3.72	2.06	1.91	70.80	55
0.20	0.30	3.60	2.05	1.85	77.53	91
0.20	0.40	3.73	2.24	1.85	78.70	113
0.20	0.50	4.06	2.14	2.09	74.73	117
0.30	0.10	4.13	2.94	2.40	66.28	51
0.30	0.20	4.12	2.83	2.26	71.99	75
0.30	0.30	4.50	2.67	2.10	72.43	71
0.30	0.40	4.47	3.54	2.36	75.50	107
0.30	0.50	4.99	2.86	2.33	68.69	95

Table 5.8: Experiment II – part 3: Combined regularization with dropout and intermittent PB reset. These regularization techniques performed best individually and the table shows that they complement each other very well.

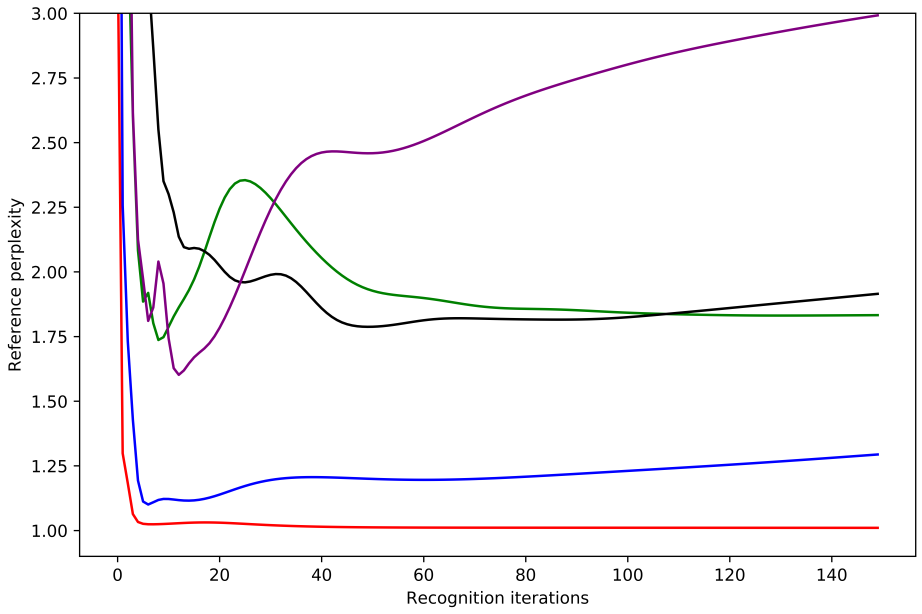


Figure 5.4: Experiment II – part 4: Perplexity of the translation as a function of the number of recognition iterations. Since binding is employed here, more recognition iterations is not necessarily better. Optimally, the recognition would be stopped at the minima in the figure, but these are not known a priori. The figure shows the first five examples in the test set.

While recognition also affects training with regard to measuring validation perplexity, only the testing step is repeated here. The schemes are: 1) Simply always use 100 recognition iterations, 2) Calculate the number of iterations that minimize the average perplexity on the validation set and then use this fixed value for the testing step, and 3) terminate recognition only when the L2 norm of the gradients decreases below 1. The first row in the table corresponds to the scheme that has been used in the experiments so far, with early stopping based on recognition loss and a maximum epoch limit, as described in section 5.2.4. The results show that there is little to gain by switching to one of the proposed recognition schemes. Scheme 1) and 2) increase the BLEU score by less than 0.1 %. However, they also increase testing time by 114 % and 53 % (not included in the table), respectively. Scheme 3) performs significantly worse. Experiment III and IV therefore continue with the original scheme with early stopping based on recognition loss.

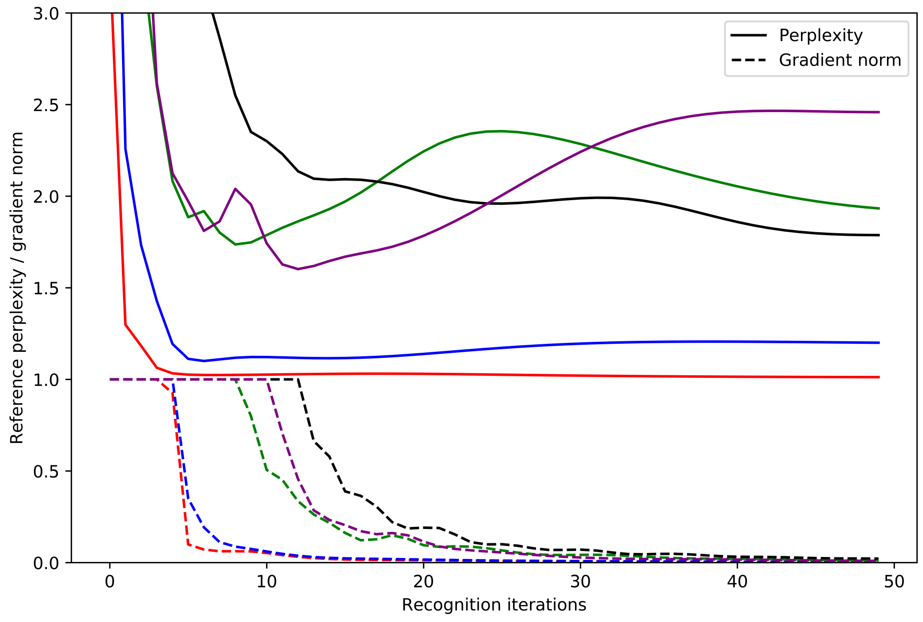


Figure 5.5: Experiment II – part 4: Perplexity of the translation as a function of the number of recognition iterations, with corresponding L2 gradient norms included. The examples are the same as in figure 5.4. The flat segments for the gradient norms at the beginning is due to the gradient clipping. The plot investigates whether the gradient norms reveal when the recognition has reached a global minimum in terms of translation perplexity. No absolute relationship is immediately apparent, although the points where the gradient norms go below 1 could be a good heuristic. This approach is tested in table 5.9.

Scheme	BLEU score
Early stopping, max 100 iterations	77.53
Constant 100 iterations	77.56
Deducing iterations from validation set: 83 iterations	77.57
Early stopping based on gradient norm	72.24

Table 5.9: Experiment II – part 4: Comparison of termination criteria for recognition. The first configuration is used in the prior experiments in this section and is copied from table 5.8. The difference between the first case of early stopping and the last is that the former terminates when the loss ceases to decrease significantly over a few steps, while the latter terminates immediately when the gradient norm drops below 1.

Part 5: Autoencoding with the easy subsets

The final results for experiment II are shown in table 5.10 for the three easy subsets, in comparison with the RNNPB and seq-to-seq results from experiment I. There is no reason to repeat the seq-to-seq tests here, as they are equal to the ones in experiment I. The results show that the Bound RNNPB model is 13.16, 15.97 and 74.71 BLEU points below the seq-to-seq model for the easy20000, easy50000 and easy-full sets. Compared to the single RNNPB model, it falls behind by 19.08, 19.38 and 84.19 points on the same datasets. The differences for the first two subsets contrast substantially with the third subset and may indicate a problem with training the Bound RNNPB model with a large dataset. Also visible in the table is an increase in training times for the RNNPB model compared to experiment I, and a decrease in testing times.

5.3.3 Experiment III

Experiment III now attempts to extend the use of the RNNPB model from autoencoding in the previous experiment to English-to-German translation. Finally, this experiment now uses the bound RNNPB in the final mode, where the goal is to learn to compute

$$b = \mathrm{RNNPB}_B(\mathrm{RNNPB}_A^{-1}(a)) \tag{5.3}$$

where A and B are two separate RNNPB models, and A models English and B models German. In both parts of this experiment, the best configuration from experiment II is used. Part 2 also tests a larger configuration of both the seq-to-seq model and RNNPB model to accommodate the more complex WMT dataset.

Part 1: Translation with the easy dataset

Table 5.11 shows the translation results for the three models – the Bound RNNPB model, seq-to-seq and Moses – on the three easy datasets. The results show that the RNNPB model achieves poor results for translation, with BLEU scores of only 18.8 %, 12.8 % and 9.57 % of what the seq-to-seq model achieves on the three datasets. Moses can furthermore be seen to achieve better BLEU scores than the seq-to-seq model for all subsets. It can be seen that the RNNPB model achieves its best scores after 7, 7 and 5 epochs, where previous experiments have shown that it needs many more epochs to give good results. Due to early stopping, training is stopped after progress has not been made for 20 epochs, similar to previous experiments. Some tests were carried out where the key configurations found in experiment II were reevaluated for English-to-German translation, using the easy20000 dataset. However, none of them improved significantly on the

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB (I)	1.01	1.01	99.43	41	2745 s	62 438 s
RNNPB (II)	2.19	1.65	80.35	119	6414 s	10 049 s
Seq-to-seq	1.01	1.08	93.51	31	432.5 s	889.4 s

(a) Using the easy20000 subset.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB (I)	1.02	1.01	98.77	43	4287 s	63 016 s
RNNPB (II)	2.90	2.14	79.39	109	88 841 s	7145 s
Seq-to-seq	1.00	1.04	95.36	37	2026 s	1143 s

(b) Using the easy50000 subset.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB (I)	1.05	1.09	94.44	47	10 935 s	67 527 s
RNNPB (II)	7.31	30.19	10.25	27	13 310 s	20 825 s
Seq-to-seq	1.02	2.61	84.96	17	4085 s	1452 s

(c) Using easy-full.

Table 5.10: Experiment II – part 5: Autoencoding with the easy subsets, using the best combinations of parameters that were found for the RNNPB model in experiment II. The seq-to-seq and RNNPB (I) results are copied from table 5.3 for experiment I, and the RNNPB (II) results for easy50000 are copied from the appropriate row in table 5.8. To summarize, the configuration used for the RNNPB model is 1) Hard binding 2) 1024 PB units 3) recognition with early stopping and max 100 epochs 4) $p_{\text{reset}} = 0.10$ and 5) dropout = 0.40.

results achieved in table 5.11, and therefore this exploration is not documented here.

Part 2: Translation with the WMT dataset

Translation results for the WMT dataset for all three models are shown in figure 5.12. Here, both the NMT models struggle with learning to translate, with less than 2 in BLEU scores. The large configurations of the RNNPB model and seq-to-seq model perform worse than their smaller counterparts. Moses achieves a much higher BLEU score of 18.03. Since the large configurations offer no improvements, the original configurations are used in experiment IV as well.

5.3.4 Experiment IV

In this last experiment, the idea of incorporating monolingual training for the Bound RNNPB model is tested. Part 1 first tries autoencoding in English just as experiment II, but with additional monolingual training. In this case, some English sentences are trained in a bound manner for the two RNNPB models and some English sentences are trained in an unbound manner in each of the models individually. These sets of sentences do not overlap. Since translation with the bound RNNPB model was mostly unsuccessful in experiment III, autoencoding may be better suited for testing the feasibility of this idea. Additionally, part 1 concludes with a value for the p_{mono} parameter. In part 2, translation with monolingual training is tested for the easy50000 and WMT datasets.

Part 1: Autoencoding with monolingual training

Table 5.13 shows that the Bound RNNPB model can benefit from monolingual training for autoencoding. The validation perplexities is decreased dramatically, and the BLEU score is increased by 7.56 points at the most, for $p_{\text{mono}} = 0.40$. The addition of monolingual training also increased the training epochs substantially, to the point where training had to be manually terminated. It is therefore possible that further improvements could have been achieved. Here, the number of epochs is counted the same way as earlier, as full rounds of the primary training dataset. Training on the monolingual dataset is simply done intermittently with respect to the p_{mono} parameter without affecting the bookkeeping. This implies that, for higher values of p_{mono}, more training steps are carried out in total each epoch. For part 2, $p_{\text{mono}} = 0.40$ is used.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB	3.76	18.26	5.31	27	1115 s	19 911 s
Seq-to-seq	1.16	8.89	28.22	27	407.2 s	887.1 s
Moses	-	-	53.56	-	465.6 s	208.8

(a) Using the easy20000 subset.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB	4.43	24.40	4.78	27	3020 s	20 210 s
Seq-to-seq	1.16	5.84	37.28	29	1134 s	910.7 s
Moses	-	-	51.16	-	420.3	175.7 s

(b) Using the easy50000 subset.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB	8.06	87.88	2.39	25	16 491 s	22 416 s
Seq-to-seq	1.45	16.65	24.96	25	4672 s	1579 s
Moses	-	-	38.95	-	687.3 s	226.5

(c) Using the easy-full subset.

Table 5.11: Experiment III – part 1: Translation results for the easy subsets, for the RNNPB model as well as the two baseline models.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB	76.03	217.05	0.17	15	8734 s	62 871 s
RNNPB (L)	29.22	178.7	0.11	13	18 611 s	47 469 s*
Seq-to-seq	17.64	72.23	1.59	17	9973 s	8219 s
Seq-to-seq (L)	14.13	149.3	0.87	15	14 382 s	35 932 s
Moses	-	-	18.03	-	4483.8 s	788.6 s

Table 5.12: Experiment III – part 2: Translation results for the WMT dataset, for the RNNPB model as well as the two baseline models. The RNNPB and seq-to-seq model test a large configuration (L) as well, with much larger networks, as specified in section 5.2. As is evident from the BLEU scores, only Moses can be said to learn to translate meaningfully with this dataset. *The large configuration of the RNNPB was so exorbitantly expensive to test on the CPU, it was tested on the GPU instead.

p_{mono}	Final train perplexity	Final val. perplexity	Best val. perplexity	Test BLEU	Train epochs
0	2.90	3.92	2.14	79.39	109
0.10	2.89	2.29	2.09	75.89	109
0.20	3.12	1.54	1.51	80.46	125
0.30	3.24	1.62	1.44	81.73	127
0.40	3.07	1.27	1.23	86.95	165*
0.50	3.15	1.42	1.32	83.04	165*
0.60	3.07	1.45	1.45	78.68	165*

Table 5.13: Experiment IV – part 1: Autoencoding results for the easy50000 subset, using non-overlapping English sentences from easy-full for additional training on monolingual data, with different values for p_{mono}. The results show that most of the configurations benefit from the additional monolingual data, with lower validation perplexities and higher BLEU scores. *The training was manually stopped at 165 epochs due to time constraints and stagnating (albeit non-zero) progress.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB	5.62	29.43	2.68 (4.78)	25	4478 s	16 221 s
Moses	-	-	49.51 (51.16)	-	317.3 s	181.8 s

Table 5.14: Experiment IV: Translation results for the easy50000 subset with incorporated training on monolingual data, using the RNNPB model as well as Moses. The monolingual data used is non-overlapping sentences from easy-full. The RNNPB model uses both English and German sentences for monolingual training, while Moses uses only German sentences. This experiment uses $p_{\text{mono}} = 0.40$ for the RNNPB model. The corresponding BLEU scores without training on monolingual data are in parentheses.

Model	Final train perplexity	Best val. perplexity	Test BLEU	Epochs	Training Time	Test Time
RNNPB	87.03	229.1	0.12 (0.17)	13	16 146 s	60 608 s
Moses	-	-	18.54 (18.03)	-	7620 s	764.5 s

Table 5.15: Experiment IV: Translation results for the WMT dataset with incorporated training on monolingual data, using the RNNPB model as well as Moses. Only the dataset is different here compared to table 5.14.

Part 2: Translation with monolingual training

Table 5.14 and 5.15 show translation results with monolingual training for the easy50000 and WMT datasets, in comparison with Moses, which also can utilize monolingual data. In this case, monolingual training provides no benefit for the RNNPB model. For easy50000, the extra monolingual data can be seen to be detrimental to Moses, while for the WMT dataset, Moses' BLEU score is increased by 0.51 points.

An additional test was done for the easy50000 dataset, where training was continued for 150 training epochs despite making no improvement. The training progress is illustrated in figure 5.6. This test shows that the problem is not simply a difficult start; the additional training epochs do not offer any improvement.

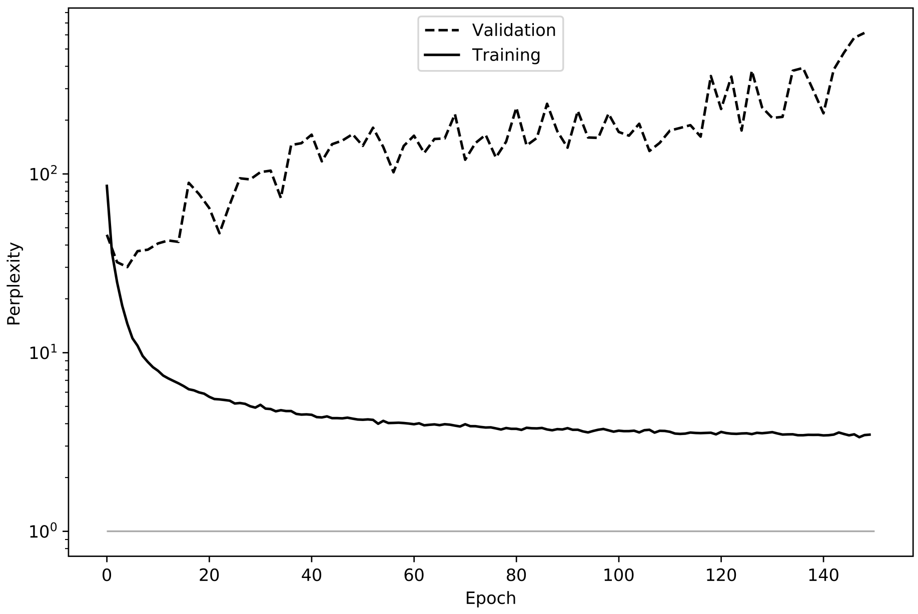


Figure 5.6: Experiment IV – part 2: The perplexity of training and validation sets during training on the easy50000 dataset, when training is continued for 150 epochs despite making no improvement.

Chapter 6

Evaluation and Conclusion

The previous chapter posed four questions and then carried out four experiments in order to answer these questions. Starting with simple autoencoding with only one RNNPB model in experiment I, the model's ability to represent language was tested. Autoencoding was then taken a step further in experiment II, where the Bound RNNPB model's ability to function as an encoder-decoder pair was investigated, with a focus on finding a good configuration for increasing the autoencoding BLEU score. In experiment III and IV, this configuration was tested on translation tasks, where the latter experiment also incorporated learning on monolingual corpora. This chapter first evaluates the results from these experiments in order in section 6.1. Section 6.2 discusses the implications of these findings and offers a conclusion, and section 6.3 summarizes the contributions of the present work. Lastly, section 6.4 gives some ideas for future work that would be interesting to pursue.

6.1 Evaluation

In regard to the three research questions posed in chapter 1, the results only affirm research question 1: the RNNPB model *can* be used to learn a large-scale language model. The second research question asked whether the Bound RNNPB model can achieve translation. The results presented here are not able to answer this question positively, although the results showed much improvement and promise with autoencoding. Lastly, the addition of monolingual data did not improve the quality of the translation in these results, failing also to affirm research question 3. However, the results still shed light on several important properties of the RNNPB model and offer contributions for its applications. The results of the experiments are now evaluated.

Experiment I

Table 5.3 gave results for autoencoding with a single RNNPB model on the easy subsets, in comparison with the seq-to-seq model. The results showed that both models were able to autoencode well.

For all three subsets, the RNNPB model achieves both better validation set perplexities and test BLEU scores by a decent margin compared to the seq-to-seq model. This is a reassuring result, although this achievement is not necessarily useful or surprising in itself. In this experiment, the two models achieve autoencoding in fundamentally different ways. The RNNPB model actively optimizes its own forward pass during recognition, and in this case, it suffices to find an arbitrary PB activation which modulates its forward pass in the best way with regard to the target sentence. For the seq-to-seq model, on the other hand, the encoder does not have access to the internals of the decoder, and it must compute the most likely representation with only a single try.

The significant result here is instead that the RNNPB is indeed able to express language well. The model is able to generalize and compute PB vectors – representations – corresponding to unlearned sentences. This is evident from its ability to regenerate the sentences from the representations with a high accuracy, as reflected by the test BLEU score. This ability is necessary for the subsequent experiments. That the model is able to achieve this on such a large scale is also a significant result of this experiment, and answers the first question posed at the beginning of chapter 5 positively.

Some observations can be made regarding the results in table 5.3. The test BLEU score can be seen to decrease for easy-full for both models. Often, it is expected that more data gives better results. However, the subsetting technique used to create easy20000 and easy50000 not only gives smaller vocabularies but is also biased towards shorter sentences. Larger vocabularies require a stronger ability to discriminate in the decoder (or during generation in the RNNPB), and longer sentences are notoriously more difficult to compress into a fixed size space. It can also be observed that the RNNPB is much slower for testing. This is, as mentioned, because testing for the RNNPB model requires recognition, which is more akin to training than inference. It may also seem odd that the RNNPB model here achieves better validation perplexity than training perplexity for easy50000. This is however to be expected: the training perplexity tests the training PB vectors, which are only regressed once per training epoch. When the validation set is tested, recognition is used with possibly many more iterations, here up to 500.

In figure 5.1, the relation between recognized PB vectors and PB vectors obtained during training from the same sentences was investigated. The plot shows a sporadic clustering of corresponding PB vectors, where the distances between corresponding points are relatively large. This plot likely explains why experiment II yields poor autoencoding results at the beginning, and provides motivation for why intermittent PB reset is so effective. It may be of little help if the binding aligns the semantics of the two models at one point in PB space, if recognition produces a PB vector at an entirely different point in PB space. The motivation for intermittent PB reset was stated as bringing the recognition dynamic into the training procedure. It is possible that it increases the performance by bridging the gap between training PB vectors and those produced by recognition.

Experiment II

In experiment II, the Bound RNNPB model is tested with autoencoding. In the first part, the binding mechanism that link the two RNNPB models is investigated. Here, the results chapter concluded that hard binding was preferable. Hard binding achieved the highest BLEU score with a large margin, where several of the configurations with soft binding fail completely.

The effect of soft binding was studied both qualitatively and numerically, in figure 5.3 and table 5.5. It was important to ascertain whether binding had indeed been successfully achieved in order to judge the performance of soft binding. The results showed that particularly binding strengths of 0.1, 1.0 and 10.0 exhibited successful binding. For binding strength $\alpha = 100.0$, table 5.5 shows that the trend turns with a decreasing binding quality, as evident from the bound-unbound distance ratio, which is likely explained by the training becoming unstable with the large binding L2 loss. It should be observed that all the configurations that achieve low distance ratios of less than 10 %, and only these configurations, are successfully able to learn autoencoding in table 5.4. This implies that the distance ratio is a meaningful metric for the quality of the binding.

The last observation made in relation to the binding is for the anomaly seen in table 5.4, where the binding strength seems to correlate significantly with the average distance between arbitrary PB vectors. Higher binding strengths seem to severely inhibit the PB vectors' dispersion through PB space. Since binding is only implemented on pairs of points, it is a bit surprising that the whole space is shrunk in this manner. A possible reason for this might be that the prediction gradients often point in opposite directions in the two models, where binding effectively achieves an averaging of the gradients. It should also be noted that the distances between PB vectors for hard binding are very large, with a mean of 10.4. This shows that the PB vectors occupy a very large part of PB space. These large distances could be attributing to slow and ineffective recognition.

In the second part of the experiment, the results show that a higher number of PB nodes is beneficial. The BLEU score increase from 23.49 by 6.96 points. The training perplexity also decreases with more PB nodes, which indicates greater ease of training. It is possible that the original configuration with 128 nodes struggled to represent all the training examples within such a small space. It's an important result that the model benefits from more PB nodes. For there to be any potential for the Bound RNNPB to work as a sequence to sequence system, the model must be able to utilize large representation spaces. One could also worry that more PB nodes would have the opposite effect by making recognition harder, due to the curse of dimensionality, which doesn't seem to be the case here.

Regularization is studied in part 3. Intermittent PB reset and dropout are showed to be effective, while training with PB noise is not. Intermittent PB reset is the most effective by far, nearly doubling the baseline score. These results may imply that this technique is essential for training the Bound RNNPB model. Although not pursued here, it would be interesting to study exactly which effect the technique has on the PB spaces and what the causal link is for the increased performance. Does it ease recognition and unite recognition trajectories between the two models, or is the mechanism simpler? One possible explanation is that intermittent reset simply shrinks the PB space by limiting the maximum distance a PB vector can travel during training. As such, it may simply constitute a regularizer on the size of the space, similar to an L2 loss (which would also have been interesting to test as a fourth technique). Likely, it's a combination of mechanisms. Nearly definitely, both dropout and intermittent PB reset inhibit overfitting, thus aiding generalization. This effect is likely instrumental for such a small dataset.

In the fourth part of the experiment, the effect of the number of recognition iterations is tested. Up until this point, the recognition scheme used has not been justified. The motivation for this inquiry was the possible risk that too many iterations were used, to the detriment of the test scores. Intuitively, recognized PB vectors should reside in the same region of PB space as the training PB vectors, to maximize the likelihood of proper binding between the two models at these points in space. Thus it could be problematic if the number of recognition iterations outnumber the training epochs, causing the recognized PB vectors to end up further away from the starting point. However, none of the proposed schemes improve the results significantly. It is possible that the two schemes with early stopping are effectively equivalent, but with different interpretations of the parameters. After all, a large gradient typically indicates that the loss changes by a lot, and vise versa[1]. Early stopping based on the gradient norm tested only a single threshold value of 1.0, while early stopping based on the recognition loss has seen some more offline testing for its epsilon value. Nevertheless, this experiment shows that the default approach is acceptable.

[1]This notion may be complicated somewhat when an adaptive optimizer (such as Adam) is used.

In the fifth and final part of the experiment, results are presented for autoencoding with the easy subsets, in comparison with the results from experiment I. The results show that the Bound RNNPB model can autoencode relatively well for the easy20000 and easy50000 datasets, at 86 % and 83 % of the BLEU points of the seq-to-seq model. However, a fundamental change occurs for the easy-full subset, where the score for the RNNPB model is reduced by 87 % to a meager 10.25 points. Where the seq-to-seq model too scores a little bit lower on this dataset, likely due to the larger vocabularies and longer sentences, the decline in performance for the RNNPB model is disproportionate. Consider for instance that the easyfull and easy50000 overlap by 50000 sentences, and that if the model only scored the same for the easy50000 sentences and 0 for the remaining 140 624 of the easy-full sentences, the average BLEU score would be 20.8. Instead it struggles to train successfully, reaching its best validation perplexity after only 7 epochs. This may be symptomatic of a larger problem that the RNNPB model has with large datasets. Furthermore, there's a considerable gap between the performance of the single RNNPB model with autoencoding from experiment I and the Bound RNNPB model in experiment II. Theoretically, if perfect binding was achieved, the scores would be equal. Instead, the binding must be described as incomplete. It is noteworthy that better binding is not achieved for the task of autoencoding, where the aligning of the two semantic spaces is subject to very little strain, which may offer foreshadowing for the subsequent disappointing translation results.

Before the evaluation of experiment II is ended, a comment regarding the methodology is due. The comparison between the Bound RNNPB model and the seq-to-seq model in this experiment cannot be interpreted as implying that the RNNPB model is almost as good as the seq-to-seq model (disregarding the easy-full results). Such an interpretation was not the intention of the experiment and would not be a fair conclusion. The results here are biased towards the RNNPB model, which has seen much more attention to optimization, development and regularization. The seq-to-seq model would for instance likely have benefited from dropout too. Indeed, the work done here could be accused of indirectly training on the test set, as it has at each step been used as justification for subsequent choices regarding the training configuration. Instead, these results should be interpreted as showing that the Bound RNNPB model *is able to* function as an encoder-decoder architecture, almost as well as a non-optimized seq-to-seq model. These results are therefore a testament to the Bound RNNPB model's maximal performance, and not to the expected performance of the two models. Thus, to conclude, these results show that the question which this experiment set out to answer is answered positively: the Bound RNNPB model *can* be employed as an encoder-decoder for the task of autoencoding.

Experiment III

Experiment III tested the Bound RNNPB model on English-to-German translation. In part 1, the easy subsets are tested. Here, the results show that the model largely struggles with learning to translate. Compared to the seq-to-seq model, which achieves acceptable test scores, the RNNPB model performs poorly on all the easy subsets. It is possible that the Bound RNNPB model is not flexible enough to utilize a common representation space for two different languages. Table 5.11 shows that the RNNPB model struggles with achieving a low training perplexity as well, even after 25-27 training epochs.

Part two shows that the BLEU scores are even worse for the WMT dataset. However, these results have less implications, seeing as the seq-to-seq model performs almost equally poorly. Most likely, these results are indicative of too little training data compared to the complex sentence structure and vocabularies. The number of training sentences here is only 5 % of the volume used in the the original work with the seq-to-seq model by Sutskever et al., who achieved a BLEU score of 30.69 (albeit on English-to-French). The easy dataset was indeed chosen to constitute a low bar for learning to translate, and upon failing for this dataset, it is unsurprising that the RNNPB model sees less success with the WMT dataset.

Experiment IV

Experiment IV shows that the RNNPB model can benefit from additional training on monolingual data, for autoencoding. In part 1, both validation perplexity and BLEU score is improved by adding monolingual training. It is again unclear what the exact causal mechanism is. Ideally, the increased scores would be due to each of the RNNPB models learning to model language better separately. Indeed, adding additional training examples could improve the organization of the PB space in this manner. However, given the unpredictable convergence of the training procedure observed in the experiments, it is just as likely that the monolingual training serves as a regularizer. Autoencoding for the easy-full dataset in experiment II showed that training is a global process, and that adding more training examples can have large impacts on the convergence. These global dependencies are inherent with the RNNPB model: consider for instance that if more examples are added, a given PB vector much wait for many more training steps between each time it's updated, and the changes in network weights that it sees between each time may be larger. This time around, the *exact same* sentences are trained, albeit 74 % without binding. This distinction causes a 76.6 point difference in BLEU scores, where the former experiment is deemed a failure and the latter a success. This goes to show that the factors dictating the convergence of the Bound RNNPB model are not well known.

Part two tests English-to-German translation with monolingual training. Here, adding monolingual training decreases the score for the RNNPB model for both easy50000 and the WMT dataset. It is interesting, although not surprising, that adding monolingual training is beneficial in part 1 of experiment IV, while it is detrimental in part 2. Also interesting is that the performance of Moses is decreased for easy50000 by adding monolingual training. This is however likely explained by the large statistical differences between the monolingual data and the easy50000 subset. This effect was described in Gulcehre et al. [2017], and reiterated in section 3.3 of this thesis. In contrast, the monolingual data used with the WMT dataset does not have such differences, and in this case Moses improves its score when monolingual data is added.

6.2 Discussion

This thesis has applied the RNNPB model to language modeling and translation at a large scale. The architecture of the RNNPB model used here replaces the original simple Jordan-type RNN with more powerful LSTM layers, which have been instrumental in recent advances with recurrent networks. Using the modern regularization technique, dropout, as well as the original regularization technique, intermittent reset of the PB vectors, the ability of the RNNPB model to express language is improved. Through four experiments, the model is tested for autoencoding, translation and for combining bound and unbound training in order to utilize monolingual data. The experiments show that the RNNPB can model language very well, and that it can achieve autoencoding with high quality when two models are bound. Furthermore, the quality of the autoencoding is increased when monolingual training is employed. However, the model fails to achieve successful translation from English to German, and for this task, monolingual training was not shown to be beneficial. The results also showed that the model may have issues training on large datasets, even for autoencoding.

The experiments have shown that the single RNNPB model can be trained with great ease. No additional modifications were required for it to learn to model the training examples well and to generalize for unseen sentences. Naturally however, two bound RNNPBs are required when translation should be achieved between two different languages. When two models were bound here, learning became much more difficult. Now, the learning objective has additionally become that all points in PB space that are meaningful for one model should have equal semantic meaning for the opposite model. The training PB vectors are responsible for aligning the semantics at some set of points. Beyond that, generalization requires that a common, regular structure emerges, so that the semantics are aligned even between these points. Ideally, more training examples would help with developing this structure, although in the current experiments, the RNNPB model isn't able to make use of such larger datasets. For the English-to-German translation, it is also possible that it's difficult to achieve a common regular structure when the two models model very different sequences.

This raises a more fundamental, philosophical question. Can two instantiations of the same sentence, in two different languages, be represented perfectly by a common representation? In theory, it's tempting to view the representation space as a semantic space where any particular sentence *meaning* can be represented as a point in space. A network should subsequently be able to express this meaning as a syntactically correct sentence in its respective language. This notion is appealing: if a perfect semantic space could be achieved, it could serve as a a lingua franca for all languages and enable powerful multi-language translation. The notion can be seen as the underlying concept behind the current application of the RNNPB model.

Language is however not a very precise matter. Sentences are often ambiguous, and particularly so without proper context. Their flexibility is not unlimited; a given sentence meaning can often be expressed better in one language than another. These properties of language could make learning such shared semantic spaces a very difficult machine learning task. Even in an ideal setting, a pair of corresponding English and German sentences are likely noisy in terms of ambiguity and not perfectly equivalent due to the constraints of the languages. It is therefore likely difficult to capture the true underlying meaning, although not necessarily impossible. With the RNNPB approach, any such inaccuracies in the model's interpretation of the sentence meaning could severely inhibit proper organization of the fragile, regular structure of the PB spaces.

This begs the question: how does this approach differ from the seq-to-seq model and similar NMT architectures? The major distinction here is that these explicitly model translation. These networks learn to effectively map a representation on the source side to a representation on the target side, in an end-to-end manner. This is likely a simpler learning task: the networks need not concern themselves to the same extent with the semantic meaning of the sentences and instead learn to map between the two syntactical manifestations of the latent semantic meaning. For the Bound RNNPB model, translation only exists as a constraint on and interpretation of the PB spaces. Individually, each model only knows about its own language, and in no way do the models learn to explicitly map from one to the other[2]. Indeed, translation is not reflected at all in the model's learning objective.

That the Bound RNNPB model struggles with learning to autoencode with the larger easy-full dataset is however more surprising than its difficulty with

[2]This could be a very interesting experiment in its own regard as an alternative to binding: Train two single RNNPB models individually, and then train a network to explicitly map from one PB space to the other. While this doesn't achieve a common semantic space, it could provide better translation results than what these experiments did.

translation. It is an unfortunate characteristic that the learning procedure for the RNNPB model depends directly on the number of training examples. When more training examples are added, each PB vector is updated relatively less frequently compared to the network weights. Particularly, this means that the choice of hyperparameters and the very convergence of the training depends on the size of the dataset, which is an undesirable trait in machine learning. It's possible that if the PB vectors "lag behind" by too much, the training procedure fails to converge. It may therefore by necessary for the future development of the model to study whether training of PB vectors and network weights can be decoupled. This would achieve a normalization of the training procedure with respect to the size of the dataset, or equivalently, the number of PB vectors. One possible scheme could be to regress all the PB vectors every N training steps and otherwise update only the network weights. Since all the PB vectors are mutually independent with respect to their gradients ($\frac{\delta}{\delta p_j}\frac{\delta L}{\delta p_i} = 0,\ \forall j \neq i$), the gradients are not dependent on the batch size, and so a very large batch size could be chosen for the PB vectors. This would allow for a small value of N without sacrificing training efficiency. In any event, it would seem necessary to achieve successful autoencoding on large datasets before one takes aim towards achieving translation. Then, it would be very interesting to study the effect of very large datasets, with millions of sentences, on the convergence of the Bound RNNPB model, where the PB spaces would be aligned in many more locations.

Experiment I likely gave the most promising results. It is interesting to consider whether this use of the single RNNPB model has practical applications in its own regard. The results showed that the model could find representations for language in order to autoencode better than the seq-to-seq model could. The practical usefulness depends on the properties of the PB vectors. Are semantic similarities captured in terms of spatial proximity? Do they offer insight into a sentence's structure and the concepts it contains, or are they mere arbitrary activations? Zhong and Canamero [2014] and Li et al. [2018] showed that the organization of PB vectors in space could reflect the similarities of emotion-aroused behaviors, which would imply the former. To this end, the current architecture and training techniques could offer a valuable contribution. As an extension of the aforementioned application, it would be interesting to consider whether PB vectors could learn to represent emotion-aroused language. Could they be used to recognize or generate hateful versus supportive utterances, or to recognize sarcasm? Here, the architecture could for instance dedicate part of the PB nodes for an emotion embedding and have the rest be free PB nodes to encode the specifics of the sentences.

As for the potential for translation, it is difficult to speak to the merit of the Bound RNNPB model. This thesis has shed light on some challenges that definitely must be solved before high-quality translation can be achieved. Even then, the theoretical foundation of the current approach could very well be inferior to the architectures that currently enjoy the most success. On the other hand, perhaps this approach – with shared semantic spaces – encourages translation on a deeper level. Should it prove to be effective, its potential for utilizing language resources across languages is definitely enticing. Although inspiring translation results could not be achieved in this thesis, the current work has taken important steps towards advancing the RNNPB model.

6.3 Contributions

This thesis' most important results is the contributions to the RNNPB model and its training procedure. Where the SLR could not find applications of the RNNPB with more powerful recurrent networks than the Elman net (i.e. Simple Recurrent Network), this thesis has demonstrated the benefits of using LSTM layers and word embeddings, as well as the Adam optimization algorithm. Experiment I and II furthermore showed that the model can be applied to problems with much larger amounts of data than what related work has done. Experiment II showed that binding of two RNNPB model can be successfully applied to achieve autoencoding for datasets up to a certain size. This experiment further contributed the techniques of hard binding and intermittent PB reset.

This use of the RNNPB model is immature in nature. It is to the best of the author's knowledge the first work that applies the RNNPB model or a similar model to the task of machine translation. As such, the mere application can be considered a contribution as well. This thesis lays an important foundation, although more work is needed to see whether there's merit for this use of the RNNPB model.

6.4 Future Work

The most pressing matter which requires future work has already been described: the Bound RNNPB model must be able to train on large datasets, and ideally, the network and training configuration should not depend on the size of the dataset. Solving these challenges are necessary to apply this model to tasks of even larger scale. Second to that, there are a few considerations that could not be included in this thesis due to its limited time frame. One of the benefits of the Bound RNNPB model that was described was the potential for bidirectional translation. The Bound RNNPB model is symmetrical and it should be able to operate in either direction. Experiments are needed to demonstrate this ability. Such an experiment will of course be more rewarding upon a successful application to actual translation. Additionally, an inquiry is due into the specific effect that intermittent PB reset and training with monolingual data have on the PB space. Through which mechanism is the performance improved? Better understanding of the convergence of the Bound RNNPB model is key to improving its performance and convergence.

A technique that was not attempted here that could be beneficial is the concept of joint recognition. Due to the nature of the Bound RNNPB model, model B could also be consulted when model A performs recognition. This achieves a similar effect as when Gulcehre et al. [2017] integrated a language model into their translation model to evaluate the quality of each translation candidate. After all, model B can at each step of recognition in model A provide commentary as to how likely it evaluates the translation corresponding to the current PB vector. Thus, once model A finds that the PB vector accurately represents the sentence in its language, it can consult model B for subsequent iterations to determine for which PB vector model B is the most confident. Such a technique where both source and target side models can participate in computing the sentence representation would be a unique feature of the RNNPB model. The resulting model would then encourage translations with both high adequacy and fluency. Perhaps it would also take steps towards bridging the gap created by imperfect binding that was observed here.

Bibliography

Arie, H., Endo, T., Jeong, S., Lee, M., Sugano, S., and Tani, J. (2010). Integrative learning between language and action: A neuro-robotics experiment. In Diamantaras, K., Duch, W., and Iliadis, L. S., editors, *Artificial Neural Networks – ICANN 2010*, pages 256–265, Berlin, Heidelberg. Springer Berlin Heidelberg.

Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv e-prints*, abs/1409.0473.

Bengio, S., Vinyals, O., Jaitly, N., and Shazeer, N. (2015). Scheduled sampling for sequence prediction with recurrent neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, pages 1171–1179, Cambridge, MA, USA. MIT Press.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

Cheng, Y., Xu, W., He, Z., He, W., Wu, H., Sun, M., and Liu, Y. (2016). Semi-supervised learning for neural machine translation. *CoRR*, abs/1606.04596.

Cho, K., van Merrienboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259.

Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Chung, J., Cho, K., and Bengio, Y. (2016). A character-level decoder without explicit segmentation for neural machine translation. *CoRR*, abs/1603.06147.

Costa-Jussà, M. R. and Farrús, M. (2014). Statistical machine translation enhancements through linguistic levels: A survey. *ACM Comput. Surv.*, 46(3):42:1–42:28.

Elman, J. L. (1990). Finding structure in time. *COGNITIVE SCIENCE*, 14(2):179–211.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Gulcehre, C., Firat, O., Xu, K., Cho, K., and Bengio, Y. (2017). On integrating a language model into neural machine translation. *Computer Speech & Language*, 45:137 – 148.

He, W., He, Z., Wu, H., and Wang, H. (2016). Improved neural machine translation with smt features. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pages 151–157. AAAI Press.

Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural Comput.*, 9(8):1735–1780.

Hutchins, J. (2005). The history of machine translation in a nutshell. `http://www.hutchinsweb.me.uk/publications.htm`.

Ito, M., Noda, K., Hoshino, Y., and Tani, J. (2006). 2006 special issue: Dynamic and interactive generation of object handling behaviors by a small humanoid robot using a dynamic neural network model. *Neural Netw.*, 19(3):323–337.

Jordan, M. I. (1997). Chapter 25 - serial order: A parallel distributed processing approach. In Donahoe, J. W. and Dorsel, V. P., editors, *Neural-Network Models of Cognition*, volume 121 of *Advances in Psychology*, pages 471 – 495. North-Holland.

Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. Seattle. Association for Computational Linguistics.

Karpathy, A. (2015). The Unreasonable Effectiveness of Recurrent Neural Networks. `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

Koehn, P., Och, F. J., and Marcu, D. (2003). Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology - Volume 1*, NAACL '03, pages 48–54, Stroudsburg, PA, USA. Association for Computational Linguistics.

Lample, G., Denoyer, L., and Ranzato, M. (2017). Unsupervised machine translation using monolingual corpora only. *CoRR*, abs/1711.00043.

Li, J., Yang, C., Zhong, J., and Dai, S. (2018). Emotion-aroused human behaviors perception using rnnpb. In *2018 10th International Conference on Modelling, Identification and Control (ICMIC)*, pages 1–6.

Lipton, Z. (2015). A critical review of recurrent neural networks for sequence learning.

Luong, M. and Manning, C. D. (2016). Achieving open vocabulary neural machine translation with hybrid word-character models. *CoRR*, abs/1604.00788.

Luong, M.-T., V. Le, Q., Sutskever, I., Vinyals, O., and Kaiser, L. (2015a). Multi-task sequence to sequence learning. *Proceedings of ICLR, San Juan, Puerto Rico.*

Luong, T., Sutskever, I., Le, Q., Vinyals, O., and Zaremba, W. (2015b). Addressing the rare word problem in neural machine translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 11–19, Beijing, China. Association for Computational Linguistics.

Och, F. J. and Ney, H. (2002). Discriminative training and maximum entropy models for statistical machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, pages 295–302, Stroudsburg, PA, USA. Association for Computational Linguistics.

Ogata, T. and Okuno, H. G. (2013). Integration of behaviors and languages with a hierarchal structure self-organized in a neuro-dynamical model. *2013 IEEE Workshop on Robotic Intelligence in Informationally Structured Space (RiiSS)*, pages 89–95.

Papineni, K., Roukos, S., Ward, T., and jing Zhu, W. (2002). Bleu: a method for automatic evaluation of machine translation. pages 311–318.

Park, J.-C., Kim, D.-S., and Nagai, Y. (2014). Developmental dynamics of rnnpb: New insight about infant action development. In del Pobil, A. P., Chinellato, E., Martinez-Martin, E., Hallam, J., Cervera, E., and Morales, A., editors, *From Animals to Animats 13*, pages 144–153, Cham. Springer International Publishing.

Park, J.-C., Kim, D.-S., and Nagai, Y. (2017). Learning for goal-directed actions using rnnpb: Developmental change of 'what to imitate'. *IEEE Transactions on Cognitive and Developmental Systems*, PP:1–1.

Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, pages III–1310–III–1318. JMLR.org.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing, MIT Press.*

Sugita, Y. and Tani, J. (2003). A holistic approach to compositional semantics: a connectionst model and robot experiments. *Advances in neural information processing systems, Cambridge, MA: MIT Press*, 17.

Sugita, Y. and Tani, J. (2005). Learning semantics combinatoriality from the interaction between linguistic and behavioral processes. *Adaptive Behav.*, 13(1):33–52.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Tani, J. (2002). Learning to generate articulated behavior through the bottom-up and the top-down interaction process. *Neural Netw.*, 16:11–23.

Tani, J. and Ito, M. (2003). Self-organization of behavioral primitives as multiple attractor dynamics: A robot experiment. *Trans. Sys. Man Cyber. Part A*, 33(4):481–488.

Tu, Z., Liu, Y., Shang, L., Liu, X., and Li, H. (2016). Neural machine translation with reconstruction. *CoRR*, abs/1611.01874.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. u., and Polosukhin, I. (2017). Attention is all you need. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates, Inc.

Zhong, J. and Canamero, L. (2014). From continuous affective space to continuous expression space: Non-verbal behaviour recognition and generation. In *4th International Conference on Development and Learning and on Epigenetic Robotics*, pages 75–80.

Zhong, J., Weber, C., and Wermter, S. (2011). Robot trajectory prediction and recognition based on a computational mirror neurons model. In Honkela, T., Duch, W., Girolami, M., and Kaski, S., editors, *Artificial Neural Networks and Machine Learning – ICANN 2011*, pages 333–340, Berlin, Heidelberg. Springer Berlin Heidelberg.

Appendices

This appendix offers some further implementation details and examples of translations and run scripts. Section .1 provides parameters for the implementations of the LSTM layer and the Adam optimizer. Section .2 provides the excerpts.

.1 Implementation details

Table A1 and A2 gives the specifics of the LSTM and Adam implementation, respectively. The parameters for both, except for dropout for the LSTM layer, use the default Tensorflow values. They are provided nevertheless since they often vary between different implementations and machine learning frameworks. The LazyAdam optimizer was previously provided together with the default package of Tensorflow, but was separated into the tensorflow_addons package for Tensorflow V2.0. The Adamax optimization algorithm was considered as an alternative, being closely related to the Adam algorithm and performing well with sparse gradient updates. However, a bug in the GPU implementation for this Tensorflow optimizer yielded NAN convergence, excluding it from consideration. Several other optimizers were tried with little success. Optimizers which are not adaptive, such as SGD, struggled to move the PB vectors from their zero-vector starting point, indicating that this may be a saddle point where the PB vectors have near-zero gradients.

Parameter	Value
Implementation	tensorflow.keras.layers.LSTM
Output activation	Tanh
Recurrent activation	Sigmoid
Dropout	†
Recurrent dropout	0
Kernel initialization	Glorot uniform
Recurrent initialization	Orthogonal

Table A1: The LSTM layer implementation used for the RNNPB model and the seq-to-seq model. In Tensorflow V2.0, this layer automatically uses the CuDNNLSTM implementation when it is available. † Dropout is used where stated in some of the experiments.

Parameter	Value
Implementation	tensorflow_addons.optimizers.LazyAdam
β_1	0.9
β_2	0.999
ϵ	10^{-7}

Table A2: The optimizer used in all experiments. LazyAdam implements the Adam optimization algorithm, but applies weight updates sparsely. That is, only the weights that are active for a current training batch are updated. This distinction is useful when the network includes embeddings; training may be drastically sped up as only relevant embeddings are updated for a given training batch. The semantics of the weight updates are also affected: with the normal Adam algorithm, momentum would be applied to all embeddings for each batch, even when their gradients are zero, effectively changing all embeddings. Similarly, the momentum terms would be updated to reflect the zero gradients, which may be inaccurate.

.2 Excerpts

Some example translations and run scripts are given here. The next section give excerpts for experiment I, from the first and final configuration of experiment II, from experiment III for all datasets, and from experiment IV only for autoencoding from its part 1. The excerpts from the run scripts have some minor modifications for brevity and formatting. The first 10 sentences from the test set are used as examples in all the subsequent tables.

Listing 1: The run script used for experiment I (part 1). The first half defines variables describing datasets and parameters, and the latter half initiates six rounds of training and testing. Encdec refers to the seq-to-seq model in the implementation. The binding strength and bind-hard parameters are given, but neither are relevant here: the autoencode flag specifies that only a single RNNPB model should be used, and therefore there is no binding. The code that has been written consists of 6 source files, where `nmt.py` is the entry point. The files contain 1482 lines of code in total. Excerpts from the test results are given in table A3-A5.

```
#!/bin/bash
mkdir rnnpb20000
mkdir rnnpb50000
mkdir rnnpbfull
mkdir encdec20000
mkdir encdec50000
mkdir encdecfull

easy20000=../../data/autoenc/easy20000
easy50000=../../data/autoenc/easy50000
full=../../data/autoenc/easy-full

train_easy20000="--train_set1=$easy20000/train.en --train_set2=$easy20000/train.en \
    --dev_set1=$easy20000/dev.en --dev_set2=$easy20000/dev.en"
train_easy50000="--train_set1=$easy50000/train.en --train_set2=$easy50000/train.en \
    --dev_set1=$easy50000/dev.en --dev_set2=$easy50000/dev.en"
train_full="--train_set1=$full/train.en --train_set2=$full/train.en \
    --dev_set1=$full/dev.en --dev_set2=$full/dev.en"
test_easy20000="--test_set=$easy20000/test.en"
test_easy50000="--test_set=$easy50000/test.en"
test_full="--test_set=$full/test.en"

run='python3 -u ../../nmt.py'

training='--do_training --embedding_size=128 --units=256 --num_layers=2 \
    --learning_rate=0.001 --batch_size=64 --max_trans_ratio=1.5 --gradient_clip=1.0 \
    --beam_size=10 --early_stopping_steps=10'
testing='--do_testing --device=cpu'

rnnpb_training='--num_PB=128 --pb_learning_rate=0.01 --max_recog_epochs=500 \
    --model=rnnpbnmt --autoencode --bind_hard --binding_strength=1.0'
encdec_training='--reverse_source --model=encdec'

train_encdec="$run $training $encdec_training"
train_rnnpb="$run $training $rnnpb_training"

# easy20000
$train_encdec $train_easy20000 --working_dir=encdec20000 &> encdec20000/train_log.txt
```

```
$train_rnnpb  $train_easy20000 --working_dir=rnnpb20000  &>  rnnpb20000/train_log.txt

# easy50000
$train_encdec $train_easy50000 --working_dir=encdec50000 &> encdec50000/train_log.txt
$train_rnnpb  $train_easy50000 --working_dir=rnnpb50000  &>  rnnpb50000/train_log.txt

# full
$train_encdec $train_full      --working_dir=encdecfull  &> encdecfull/train_log.txt
$train_rnnpb  $train_full      --working_dir=rnnpbfull   &>  rnnpbfull/train_log.txt

$run $testing $test_easy20000 --working_dir=rnnpb20000  &>  rnnpb20000/test_log.txt
$run $testing $test_easy20000 --working_dir=encdec20000 &> encdec20000/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb50000  &>  rnnpb50000/test_log.txt
$run $testing $test_easy50000 --working_dir=encdec50000 &> encdec50000/test_log.txt
$run $testing $test_full      --working_dir=rnnpbfull   &>  rnnpbfull/test_log.txt
$run $testing $test_full      --working_dir=encdecfull  &> encdecfull/test_log.txt
```

M	Sentence
T R S	Tom told me he hasn &apos;t done that yet . Tom told me he hasn &apos;t done that yet . Tom told me he asked something done that yesterday .
T R S	I want to see Tom again . I want to see Tom again . I want to see Tom again .
T R S	He was very busy all day . He was very busy all day . He was very busy all day .
T R S	We were at school together . We were at school together . We were at school together .
T R S	I &apos;m sorry for this . I &apos;m sorry for this . I &apos;m sorry for this .
T R S	Her hair is long . Her hair is long . Her hair is long .
T R S	Tom and Mary were alone in the park . Tom and Mary were alone in the park . Tom and Mary were alone in the park .
T R S	How long do I have to stay ? How long do I have to stay ? How long do I have to stay ?
T R S	You want me to go , don &apos;t you ? You want me to go , don &apos;t you ? You want me to go , don &apos;t you ?
T R S	It can &apos;t be that bad . It can &apos;t be that bad . It can &apos;t be that bad .

Table A3: Experiment I - easy20000, autoencoding with (R) RNNPB model and (S) seq-to-seq model. T denotes the true sentence. The ”&apos;” tokens arise from the tokenization behavior of Moses.

M	Sentence
T R S	Be nice to her . Be nice to her . Be nice to her .
T R S	How come you know so much ? How come you know so much ? How come you know so much ?
T R S	There &apos;s something that I want to do before I leave . There &apos;s something that I want to do before I leave . There &apos;s something that I want to do before I leave .
T R S	I can &apos;t think of anything I &apos;d want from you . I can &apos;t think of anything I &apos;d want from you . I can &apos;t think or say I &apos;d want from you .
T R S	I think it &apos;s better not to try it . I think it &apos;s better not to try it . I think it &apos;s better not not try it .
T R S	This isn &apos;t so difficult . This isn &apos;t so difficult . This isn &apos;t so difficult .
T R S	I &apos;ve looked all over for Tom , but I can &apos;t find him . I &apos;ve looked all over for Tom , but I can &apos;t find him . I &apos;ve almost came until for Tom , but I can &apos;t find him .
T R S	Is he going to make it ? Is he going to make it ? Is he going to make it ?
T R S	That hurts ! Stop it ! That loves him first night . That happened dangerous chair up !
T R S	Tom and Mary both know how to swim . Tom and Mary both know how to swim . Tom and Mary both know how to swim .

Table A4: Experiment I - easy50000, autoencoding with (R) RNNPB model and (S) seq-to-seq model. T denotes the true sentence.

M	Sentence
T R S	Use the manual override . Use the manual `<UNK>` . Use the White anthem .
T R S	The new tunnel will link Britain and France . The new tunnel will link Britain and France . The new clients will commit Britain and France .
T R S	I don &apos;t think Tom speaks French . I don &apos;t think Tom speaks French . I don &apos;t think Tom speaks French .
T R S	He sometimes goes to Tokyo on business . He sometimes goes to Tokyo on business . He sometimes goes to Tokyo on his
T R S	Tom &apos;s fingerprints were on the gun . Tom &apos;s fingerprints were on the gun . Tom &apos;s graduation were on the gun .
T R S	I want to go to the zoo with Tom . I want to go to the zoo with Tom . I want to go to the zoo with Tom .
T R S	There &apos;s no chance that he &apos;ll recover . There &apos;s no chance that he &apos;ll recover . There &apos;s no chance that he &apos;ll recover .
T R S	This colony was founded in 1700 . This colony was founded in undergrad . This wonders was founded in 1843 .
T R S	That was different . That was different . That was different .
T R S	Tom doesn &apos;t like you very much either . Tom doesn &apos;t like you very much either . Tom doesn &apos;t like you very much either .

Table A5: Experiment I - easyfull, autoencoding with (R) RNNPB model and (S) seq-to-seq model. T denotes the true sentence.

Listing 2: The run script used for experiment II, part 1. In this experiment, different binding strengths were used. An additional flag is used to specify hard binding. When this flag is given, the binding strength value is not used. Excerpts from the test results for hard binding are given in table A6.

```
mkdir rnnpb-hard
mkdir rnnpb-100
mkdir rnnpb-10
mkdir rnnpb-1
mkdir rnnpb-0.1
mkdir rnnpb-0.01
mkdir rnnpb-0.001
mkdir rnnpb-0.0001

easy50000=../../data/autoenc/easy50000

train_easy50000="--train_set1=$easy50000/train.en --train_set2=$easy50000/train.en \
    --dev_set1=$easy50000/dev.en --dev_set2=$easy50000/dev.en"

test_easy50000="--test_set=$easy50000/test.en"

run='python3 -u ../../nmt.py'

training='--do_training --embedding_size=128 --units=256 --num_layers=2 \
    --learning_rate=0.001 --batch_size=64 --max_trans_ratio=1.5 --gradient_clip=1.0 \
    --beam_size=10 --early_stopping_steps=20'

testing='--do_testing --device=cpu'

rnnpb_training='--num_PB=128 --pb_learning_rate=0.01 --max_recog_epochs=100 \
    --model=rnnpbnmt'

train="$run $training $train_easy50000 $rnnpb_training"

# easy50000
$train --binding_strength=1.0    --working_dir=rnnpb-hard \
    --bind_hard &> rnnpb-hard/train_log.txt
$train --binding_strength=100    --working_dir=rnnpb-100    &> rnnpb-100/train_log.txt
$train --binding_strength=10     --working_dir=rnnpb-10     &> rnnpb-10/train_log.txt
$train --binding_strength=1      --working_dir=rnnpb-1      &> rnnpb-1/train_log.txt
$train --binding_strength=0.1    --working_dir=rnnpb-0.1    &> rnnpb-0.1/train_log.txt
$train --binding_strength=0.01   --working_dir=rnnpb-0.01   &> rnnpb-0.01/train_log.txt
$train --binding_strength=0.001  --working_dir=rnnpb-0.001  &> rnnpb-0.001/train_log.txt
$train --binding_strength=0.0001 --working_dir=rnnpb-0.0001 &> rnnpb-0.0001/train_log.txt

# The training configurations that failed were removed from testing below.
$run $testing $test_easy50000 --working_dir=rnnpb-hard &> rnnpb-hard/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb-10   &> rnnpb-10/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb-1    &> rnnpb-1/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb-0.1  &> rnnpb-0.1/test_log.txt
```

M	Sentence
T R S	Be nice to her . Bring me to the hospital . Be nice to her .
T R S	How come you know so much ? How do you have a beer ? How come you know so much ?
T R S	There &apos;s something that I want to do before I leave . It &apos;s too difficult to do . There &apos;s something that I want to do before I leave .
T R S	I can &apos;t think of anything I &apos;d want from you . I can &apos;t know where Tom is or not . I can &apos;t think or say I &apos;d want from you .
T R S	I think it &apos;s better not to try it . I &apos;m looking for it to Tom . I think it &apos;s better not not try it .
T R S	This isn &apos;t so difficult . This isn &apos;t going on . This isn &apos;t so difficult .
T R S	I &apos;ve looked all over for Tom , but I can &apos;t find him . I &apos;ve been waiting for that for him . I &apos;ve almost came until for Tom , but I can &apos;t find him .
T R S	Is he going to make it ? Is it OK to do that ? Is he going to make it ?
T R S	That hurts ! Stop it ! That car is ! That happened dangerous chair up !
T R S	Tom and Mary both know how to swim . Tom and Mary have nothing enough . Tom and Mary both know how to swim .

Table A6: Experiment II – part 1 with hard binding, easy50000. Autoencoding with (R) RNNPB model and (S) seq-to-seq model. T denotes the true sentence. The seq-to-seq sentences are the same as in table A4. This table is included to show the evolution of the Bound RNNPB's performance with autoencoding in experiment II.

Listing 3: The run script used for experiment II, part 5. Here, the final autoencoding results are computed for the easy subsets. Excerpts from the test results are given in table A7-A9.

```
mkdir rnnpb20000
mkdir rnnpb50000
mkdir rnnpbfull

easy20000=../../data/autoenc/easy20000
easy50000=../../data/autoenc/easy50000
easyfull=../../data/autoenc/easy-full

train_easy20000="--train_set1=$easy20000/train.en --train_set2=$easy20000/train.en \
    --dev_set1=$easy20000/dev.en --dev_set2=$easy20000/dev.en"
train_easy50000="--train_set1=$easy50000/train.en --train_set2=$easy50000/train.en \
    --dev_set1=$easy50000/dev.en --dev_set2=$easy50000/dev.en"
train_easyfull="--train_set1=$easyfull/train.en --train_set2=$easyfull/train.en \
    --dev_set1=$easyfull/dev.en --dev_set2=$easyfull/dev.en"

test_easy20000="--test_set=$easy20000/test.en"
test_easy50000="--test_set=$easy50000/test.en"
test_easyfull="--test_set=$easyfull/test.en"

run='python3 -u ../../nmt.py'

training='--do_training --embedding_size=128 --units=256 --num_layers=2 \
    --learning_rate=0.001 --batch_size=64 --max_trans_ratio=1.5 --gradient_clip=1.0 \
    --beam_size=10 --early_stopping_steps=20'

testing='--do_testing --device=cpu'

rnnpb_training='--pb_learning_rate=0.01 --binding_strength=1.0 --bind_hard \
    --max_recog_epochs=100 --num_PB=1024 --model=rnnpbnmt --p_reset=0.10 \
    --dropout=0.40'

train="$run $training $rnnpb_training"

# easy50000
$train $train_easy20000 --working_dir=rnnpb20000 &> rnnpb20000/train_log.txt
$train $train_easy50000 --working_dir=rnnpb50000 &> rnnpb50000/train_log.txt
$train $train_easyfull  --working_dir=rnnpbfull  &> rnnpbfull/train_log.txt

$run $testing $test_easy20000 --working_dir=rnnpb20000   &> rnnpb20000/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb50000   &> rnnpb50000/test_log.txt
$run $testing $test_easyfull  --working_dir=rnnpbfull    &> rnnpbfull/test_log.txt
```

M	Sentence
T	Tom told me he hasn &apos;t done that yet .
R	Tom told me he hasn &apos;t done that yet .
S	Tom told me he asked something done that yesterday .
T	I want to see Tom again .
R	I want to see Tom again .
S	I want to see Tom again .
T	He was very busy all day .
R	He was very busy all day .
S	He was very busy all day .
T	We were at school together .
R	We were able to swim .
S	We were at school together .
T	I &apos;m sorry for this .
R	I &apos;m sorry for this .
S	I &apos;m sorry for this .
T	Her hair is long .
R	Her dinner is long .
S	Her hair is long .
T	Tom and Mary were alone in the park .
R	Tom and Mary were alone .
S	Tom and Mary were alone in the park .
T	How long do I have to stay ?
R	How long do I have to stay ?
S	How long do I have to stay ?
T	You want me to go , don &apos;t you ?
R	You want me to have , not we do it .
S	You want me to go , don &apos;t you ?
T	It can &apos;t be that bad .
R	It can &apos;t be that bad .
S	It can &apos;t be that bad .

Table A7: Experiment II – part 5, easy20000. Autoencoding with (R) RNNPB model and (S) seq-to-seq model. T denotes the true sentence. The seq-to-seq sentences are the same as in table A3.

M	Sentence
T R S	Be nice to her . Be nice to her . Be nice to her .
T R S	How come you know so much ? How come you know so much ? How come you know so much ?
T R S	There &apos;s something that I want to do before I leave . There &apos;s something that that I want to do before . There &apos;s something that I want to do before I leave .
T R S	I can &apos;t think of anything I &apos;d want from you . I can &apos;t think of that I was supposed from you . I can &apos;t think or say I &apos;d want from you .
T R S	I think it &apos;s better not to try it . I think it &apos;s better not to do this . I think it &apos;s better not not try it .
T R S	This isn &apos;t so difficult . This isn &apos;t so difficult . This isn &apos;t so difficult .
T R S	I &apos;ve looked all over for Tom , but I can &apos;t find him . I &apos;ve lost my watch , but I can &apos;t find him . I &apos;ve almost came until for Tom , but I can &apos;t find him .
T R S	Is he going to make it ? Is he going to make it ? Is he going to make it ?
T R S	That hurts ! Stop it ! That hurts ! That happened dangerous chair up !
T R S	Tom and Mary both know how to swim . Tom and Mary both know how to swim . Tom and Mary both know how to swim .

Table A8: Experiment II – part 5, easy50000. Autoencoding with (R) RNNPB model and (S) seq-to-seq model. T denotes the true sentence. The seq-to-seq sentences are the same as in table A4.

M	Sentence
T	Use the manual override .
R	Open your hands unattended .
S	Use the White anthem .
T	The new tunnel will link Britain and France .
R	I &apos;m tired of the world .
S	The new clients will commit Britain and France .
T	I don &apos;t think Tom speaks French .
R	I don &apos;t understand Tom anymore .
S	I don &apos;t think Tom speaks French .
T	He sometimes goes to Tokyo on business .
R	Tom often goes to school on Sundays .
S	He sometimes goes to Tokyo on his
T	Tom &apos;s fingerprints were on the gun .
R	Tom seems to be in the hospital .
S	Tom &apos;s graduation were on the gun .
T	I want to go to the zoo with Tom .
R	I want to go to Boston with Tom .
S	I want to go to the zoo with Tom .
T	There &apos;s no chance that he &apos;ll recover .
R	There &apos;s no time that you need .
S	There &apos;s no chance that he &apos;ll recover .
T	This colony was founded in 1700 .
R	This rule belongs out .
S	This wonders was founded in 1843 .
T	That was different .
R	They were young .
S	That was different .
T	Tom doesn &apos;t like you very much either .
R	Tom doesn &apos;t know Mary anymore .
S	Tom doesn &apos;t like you very much either .

Table A9: Experiment II – part 5, easyfull. Autoencoding with (R) RNNPB model and (S) seq-to-seq model. T denotes the true sentence. The seq-to-seq sentences are the same as in table A5.

Listing 4: The run script used for experiment III, part 1, testing English-to-German translation with the easy subsets. Excerpts from the test results are given in table A7-A9.

```
mkdir rnnpb20000
mkdir rnnpb50000
mkdir rnnpbfull
mkdir encdec20000
mkdir encdec50000
mkdir encdecfull

easy20000=../../data/easy/easy20000
easy50000=../../data/easy/easy50000
easyfull=../../data/easy/easy-full

train_easy20000="--train_set1=$easy20000/train.en --train_set2=$easy20000/train.de \
    --dev_set1=$easy20000/dev.en --dev_set2=$easy20000/dev.de"
train_easy50000="--train_set1=$easy50000/train.en --train_set2=$easy50000/train.de \
    --dev_set1=$easy50000/dev.en --dev_set2=$easy50000/dev.de"
train_easyfull="--train_set1=$easyfull/train.en --train_set2=$easyfull/train.de \
    --dev_set1=$easyfull/dev.en --dev_set2=$easyfull/dev.de"

test_easy20000="--test_set=$easy20000/test.en"
test_easy50000="--test_set=$easy50000/test.en"
test_easyfull="--test_set=$easyfull/test.en"

run='python3 -u ../../nmt.py'

training='--do_training --embedding_size=128 --units=256 --num_layers=2 \
    --learning_rate=0.001 --batch_size=64 --max_trans_ratio=1.5 --gradient_clip=1.0 \
    --beam_size=10 --early_stopping_steps=20 --vocab1_max=30000 --vocab2_max=30000'

testing='--do_testing --device=cpu'

rnnpb_training='--pb_learning_rate=0.01 --binding_strength=1.0 --bind_hard \
    --max_recog_epochs=100 --num_PB=1024 --p_reset=0.10 --dropout=0.40 \
    --model=rnnpbnmt'
encdec_training='--reverse_source --model=encdec'

train_rnnpb="$run $training $rnnpb_training"
train_encdec="$run $training $encdec_training"

# easy20000
$train_encdec $train_easy20000 --working_dir=encdec20000 &> encdec20000/train_log.txt
$train_rnnpb  $train_easy20000 --working_dir=rnnpb20000  &> rnnpb20000/train_log.txt

# easy50000
$train_encdec $train_easy50000 --working_dir=encdec50000 &> encdec50000/train_log.txt
$train_rnnpb  $train_easy50000 --working_dir=rnnpb50000  &> rnnpb50000/train_log.txt
```

```
# easyfull
$train_encdec $train_easyfull --working_dir=encdecfull &> encdecfull/train_log.txt
$train_rnnpb  $train_easyfull --working_dir=rnnpbfull  &> rnnpbfull/train_log.txt

# Test
$run $testing $test_easy20000 --working_dir=encdec20000 &> encdec20000/test_log.txt
$run $testing $test_easy20000 --working_dir=rnnpb20000 &> rnnpb20000/test_log.txt
$run $testing $test_easy50000 --working_dir=encdec50000 &> encdec50000/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb50000 &> rnnpb50000/test_log.txt
$run $testing $test_easyfull --working_dir=encdecfull &> encdecfull/test_log.txt
$run $testing $test_easyfull --working_dir=rnnpbfull &> rnnpbfull/test_log.txt
```

M	Sentence
TE	I was at home .
TG	Ich war zu Hause .
R	Ich war nach Boston .
S	Ich war zu Hause .
M	Ich war zu Hause .
TE	We &apos;ll wait .
TG	Wir werden warten .
R	Wir müssen gehen .
S	Wir werden warten .
M	Wir werden warten .
TE	Did you like Boston ?
TG	Hat es Ihnen in Boston gefallen ?
R	Hast du das helfen ?
S	Bist du in Boston ?
M	Hast du in Boston gefallen ?
TE	I &apos;m really sorry about that .
TG	Das tut mir wirklich leid .
R	Das kann mir leid .
S	Es tut mir leid .
M	Es tut mir wirklich leid .
TE	Tom is happy again .
TG	Tom ist wieder glücklich .
R	Tom ist nicht da .
S	Tom ist sehr glücklich .
M	Tom ist wieder glücklich .
TE	You can &apos;t buy anything if you have no money .
TG	Man kann nichts kaufen , wenn man kein Geld hat .
R	Du bist nicht mehr Zeit sein .
S	Du kannst mich besser besser , wenn du nicht war .
M	Du kannst , wenn du nichts gekauft habe kein Geld .
TE	You are not a child any more .
TG	Du bist kein Kind mehr .
R	Es ist nicht so groß .
S	Du bist nicht viel mehr .
M	Du bist kein Kind mehr .
TE	We &apos;re still very busy .
TG	Wir haben noch immer viel zu tun .
R	Es ist sehr groß .
S	Wir sind keine Angst .
M	Wir sind noch sehr beschäftigt .

TE	Did you see it ?
TG	Haben Sie es gesehen ?
R	Hast du das machen ?
S	Hast du es gesehen ?
M	Hast du das gesehen ?
TE	Tom is a dear friend .
TG	Tom ist ein lieber Freund .
R	Tom ist krank .
S	Tom ist ein Bruder .
M	Tom ist ein dear Freund .

Table A10: Experiment III - easy20000, translation with (R) RNNPB model and (S) seq-to-seq model. TE and TG denotes the true sentences in English and German, respectively.

M	Sentence
TE	I think I know what you need .
TG	Ich glaube , ich weiß , was du brauchst .
R	Ich weiß genau , was ich gesagt hat .
S	Ich glaube , du weißt , was du willst .
M	Ich denke , ich weiß , was du brauchst .
TE	I met Tom for the first time three days ago .
TG	Ich habe Tom zum ersten Mal vor drei Tagen getroffen .
R	Ich habe Tom gefunden .
S	Ich habe Tom das letzte Woche verpasst .
M	Ich traf Tom zum ersten Mal vor drei Tagen .
TE	When they &apos;re happy , we &apos;re happy .
TG	Wenn sie glücklich sind , sind wir glücklich .
R	Es ist schwierig .
S	Wenn sie glücklich wäre , ist glücklich .
M	Wann Sie sind glücklich , wir sind glücklich .
TE	I don &apos;t understand what she wants me to do .
TG	Ich verstehe nicht , was sie will , dass ich tue .
R	Ich glaube nicht , dass ich das getan hat .
S	Ich verstehe nicht , was ich will .
M	Ich verstehe nicht , was sie will mich zu tun .
TE	I can &apos;t be something I &apos;m not .
TG	Ich kann nichts sein , was ich nicht bin .
R	Ich weiß nicht , was ich bin .
S	Ich kann nicht tun , nicht .
M	Ich kann nicht bin ich nicht etwas sein .
TE	&quot; What &apos;s your name ? &quot; I asked .
TG	&quot; Wie ist Ihr Name ? &quot; , fragte ich .
R	Es ist gefährlich , oder ?
S	Mein Vater ist das erste erste Mal .
M	Wie ist Ihr Name ? " - " Ich gefragt .
TE	What are you doing here so early ?
TG	Was macht ihr hier so früh ?
R	Bist du noch böse ?
S	Was machst du hier so früh ?
M	Was machst du hier so früh ?
TE	Where can I find Tom ?
TG	Wo kann ich Tom finden ?
R	Wo hat Tom gefunden ?
S	Wo kann ich Tom finden ?
M	Wo kann ich Tom gefunden ?

TE	Tom stopped reading for a moment .
TG	Tom hörte für einen Moment auf zu lesen .
R	Tom geht ein guter Freund .
S	Tom hörte zu warten .
M	Tom hörte auf zu lesen für einen Moment .
TE	I hope you know that .
TG	Ich hoffe , Sie wissen das .
R	Ich frage mich , was ich will .
S	Ich hoffe , du weißt das .
M	Ich hoffe , du weißt das .

Table A11: Experiment III - easy50000, translation with (R) RNNPB model and (S) seq-to-seq model. TE and TG denotes the true sentences in English and German, respectively.

M	Sentence
TE	They set to work at once .
TG	Sie machten sich sogleich ans Werk .
R	Tom und Maria sind zusammen .
S	Sie weigerten sich um die Arbeit .
M	Sie machten sich umgehend an die Arbeit .
TE	He &apos;s a gambler .
TG	Er ist ein Spieler .
R	Er war ein `<UNK>` .
S	Er ist ein `<UNK>` .
M	Er ist ein Spieler .
TE	Where can I find an ATM ?
TG	Wo finde ich einen Geldautomat ?
R	Was hast du angefangen ?
S	Wo kann ich das kaufen ?
M	Wo ist der nächste Geldautomat ?
TE	She has a daughter whose name is Mary .
TG	Sie hat eine Tochter , die Maria heißt .
R	Tom hatte ein `<UNK>` .
S	Sie hat einen Freund , die Maria heißt .
M	Sie hat eine Tochter , dessen Name ist Maria .
TE	Everyone on the bus was asleep except the driver .
TG	Außer dem Fahrer schliefen alle im Bus .
R	Danke für mich .
S	Alle auf dem Fenster kam eingeschlafen .
M	Jeder in den Bus ein schlief , bis der Fahrer .
TE	You still owe me one .
TG	Du bist mir noch etwas schuldig .
R	Du hast mir ein paar Fragen gekauft .
S	Ihr habt mir noch immer einen Dummkopf .
M	Du schuldest mir noch einen .
TE	I won &apos;t be coming back .
TG	Ich werde nicht wiederkommen .
R	Ich habe Tom gesehen .
S	Ich werde nicht zurückkommen .
M	Ich werde nicht wiederkommen .
TE	Even though he apologized , I &apos;m still furious .
TG	Obwohl er sich entschuldigt hat , bin ich immer noch wütend .
R	Der Mann war sehr groß .
S	Obwohl ich wütend bin , bin ich nicht böse .
M	Obwohl er entschuldigte sich , ich bin immer noch wütend .

TE	Did I interrupt something ?
TG	Habe ich dich bei etwas gestört ?
R	Kann ich dir helfen ?
S	Habe ich etwas mitgebracht ?
M	Habe ich etwas unterbrechen ?
TE	According to the paper , it will snow tomorrow .
TG	Der Zeitung zufolge soll es morgen schneien .
R	Meine Mutter ist ein `<UNK>` .
S	Am folgenden Uhr wird es morgen schneien .
M	In der Zeitung steht , es wird morgen schneien .

Table A12: Experiment III - easyfull, translation with (R) RNNPB model and (S) seq-to-seq model. TE and TG denotes the true sentences in English and German, respectively.

Listing 5: The run script used for experiment III, part 2, testing English-to-German translation with the WMT dataset. Excerpts from the test results with the small RNNPB and seq-to-seq models are given in table A13.

```
mkdir encdec-large
mkdir rnnpb-large
mkdir encdec
mkdir rnnpb

wmt=../../data/wmt

train_wmt="--train_set1=$wmt/train.en --train_set2=$wmt/train.de \
    --dev_set1=$wmt/dev_short.en --dev_set2=$wmt/dev_short.de"

test_wmt="--test_set=$wmt/test.en"

run='python3 -u ../../nmt.py'

training='--do_training --learning_rate=0.001 --batch_size=64 --max_trans_ratio=1.5 \
    --gradient_clip=1.0 --beam_size=10 --early_stopping_steps=10 --vocab1_max=30000 \
    --vocab2_max=30000'

testing='--do_testing --device=cpu'

rnnpb_training='--pb_learning_rate=0.01 --binding_strength=1.0 --bind_hard \
    --max_recog_epochs=100 --num_PB=1024 --p_reset=0.10 --dropout=0.40 \
    --model=rnnpbnmt'
encdec_training='--reverse_source --model=encdec'

train_encdec="$run $training $encdec_training $train_wmt"
train_rnnpb="$run $training $rnnpb_training $train_wmt"

$train_encdec --working_dir=encdec-large --embedding_size=512 --units=1024 \
    --num_layers=4 &> encdec-large/train_log.txt
$train_rnnpb  --working_dir=rnnpb-large  --embedding_size=512 --units=1024 \
    --num_layers=4 &> rnnpb-large/train_log.txt

$train_encdec --working_dir=encdec --embedding_size=128 --units=256 \
    --num_layers=2 &> encdec/train_log.txt
$train_rnnpb  --working_dir=rnnpb  --embedding_size=128 --units=256 \
    --num_layers=2 &> rnnpb/train_log.txt

$run $testing $test_wmt --working_dir=encdec-large &> encdec-large/test_log.txt
$run $testing $test_wmt --working_dir=rnnpb-large  &> rnnpb-large/test_log.txt
$run $testing $test_wmt --working_dir=encdec       &> encdec/test_log.txt
$run $testing $test_wmt --working_dir=rnnpb        &> rnnpb/test_log.txt
```

M	Sentence
TE	Munich 1856 : Four maps that will change your view of the city
TG	München 1856 : Vier Karten , die Ihren Blick auf die Stadt verändern
R	`<UNK>` `<UNK>`
S	`<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>`
M	München 1856 unternommene : Vier Karten , die Veränderung Ihrer Meinung nach der Stadt
TE	A mental asylum , where today young people are said to meet .
TG	Eine Irren-Anstalt , wo sich heute Jugendliche begegnen sollen .
R	`<UNK>` `<UNK>` , vor allem in den USA , in denen die `<UNK>` in den letzten Jahren in den letzten zehn
S	Ein `<UNK>` , in dem die Menschen leben , so wie sie leben .
M	Eine psychische Asyl , wo junge Menschen sind heute zu erfüllen .
TE	A crypt chapel , where they are now digging tunnels for the S-Bahn .
TG	Eine Gruftkapelle , wo nun für den S-Bahn-Tunnel gegraben wird .
R	Das `<UNK>` ist ein `<UNK>` für das `<UNK>` .
S	Eine `<UNK>` , die sich `<UNK>` , `<UNK>` , `<UNK>` `<UNK>` .
M	Eine crypt Chapel , wo sie jetzt gräbt Tunnel für die S-Bahn .
TE	Allotment holders cultivate the soil of former farmers .
TG	Kleingärtner bewirtschaften den einstigen Grund von Bauern .
R	Eine `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` .
S	`<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` .
M	Die Inhaber kultivieren allotment der Boden des ehemaligen Bauern .
TE	The oldest official map of Munich brings captivating stories to light .
TG	Die älteste offizielle Karte Münchens fördert spannende Geschichten zu Tage .
R	Der `<UNK>` `<UNK>` `<UNK>` `<UNK>` ist in der Lage .
S	Der `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` .
M	Die älteste offizielle Landkarte von München bringt captivating Geschichten ans Licht .
TE	It is annoying when geographical maps are not up-to-date .
TG	Es nervt , wenn Landkarten nicht aktuell sind .
R	Es ist kein Zufall , warum es darum geht , dass es darum geht .
S	`<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` `<UNK>` .
M	Ist es nervig als geographische Karten sind nicht aktuellen .

TE	Anyone who has ever got worked up because the car &apos;s sat-nav is showing a green field instead of a bypass knows that .
TG	Das kennt jeder , der sich schon mal aufregen musste , weil das Auto-Navi statt einer Umgehungsstraße eine grüne Wiese anzeigte .
R	Es gibt keinen Grund , warum es darum geht , dass es darum geht , dass es darum geht , einen `<UNK>` zu verhindern .
S	Wer behauptet , ist , dass die `<UNK>` das `<UNK>` `<UNK>` , das die Menschen `<UNK>` , wenn sie `<UNK>` .
M	Jeder , der je hat funktioniert , weil das Auto der sat-nav zeigt eine grüne Feld statt weiß , dass ein umgehen .
TE	The historical maps of the digital BayernAtlas , an offering from the State Government &apos;s Geoportal Bayern , are anything but up-to-date - and yet it is precisely for this reason that they are so informative .
TG	Die historischen Landkarten des digitalen Bayern-Atlases , ein Angebot des Geoportals Bayern der Staatsregierung , sind alles andere als aktuell - doch gerade deshalb sehr aufschlussreich .
R	Die `<UNK>` der `<UNK>` von `<UNK>` und `<UNK>`
S	Die größten `<UNK>` der `<UNK>` , die `<UNK>` `<UNK>` , `<UNK>` `<UNK>` , ist nicht so gut wie möglich .
M	Die historische Landkarten der digitalen BayernAtlas , ein , von der Regierung die Geoportal Bayern , sind alles andere als aktuelle - und dennoch ist es genau aus diesem Grund , dass sie so informative .
TE	Especially when one compares them with current online maps .
TG	Besonders wenn man sie mit aktuellen Online-Karten vergleicht .
R	Angesichts der Tatsache , dass sie in der Lage sind , sich in der Lage sind
S	Im Gegensatz zu einem gewissen Grad an den `<UNK>` .
M	Insbesondere , wenn man sie mit aktuellen online Karten vergleicht .
TE	Then it becomes clear how the towns and municipalities in the distribution area of Munich &apos;s Merkur newspaper have changed since the 19th century .
TG	Dann wird deutlich , wie sich Städte und Gemeinden im Verbreitungsgebiet des Münchner Merkur seit dem 19. Jahrhundert verändert haben .
R	Zunächst ist der Ansicht , dass die `<UNK>` `<UNK>` und `<UNK>` `<UNK>` `<UNK>` `<UNK>` .
S	Dann ist es wichtig , dass die `<UNK>` und `<UNK>` in den späten 1970er Jahren in der Geschichte des 20. Jahrhunderts `<UNK>` .
M	Dann wird es klar , wie die Städte und Gemeinden in der Verteilung von München Bereich der Merkur Zeitung geändert haben seit dem 19. Jahrhundert .

Table A13: Experiment III - WMT, translation with (R) RNNPB model and (S) seq-to-seq model. TE and TG denotes the true sentences in English and German, respectively.

Listing 6: The run script used for experiment IV, part 1, autoencoding with monolingual training with the easy50000 dataset. Excerpts from the test results are given in table A14.

```
mkdir rnnpb-0.60
mkdir rnnpb-0.50
mkdir rnnpb-0.40
mkdir rnnpb-0.30
mkdir rnnpb-0.20
mkdir rnnpb-0.10

easy50000=../../data/autoenc/easy50000

train_easy50000="--train_set1=$easy50000/train.en --train_set2=$easy50000/train.en \
    --dev_set1=$easy50000/dev.en --dev_set2=$easy50000/dev.en"
train_easymono="--mono_set1=$easy50000/mono.en --mono_set2=$easy50000/mono.en"

test_easy50000="--test_set=$easy50000/test.en"

run='python3 -u ../../nmt.py'

training='--do_training --embedding_size=128 --units=256 --num_layers=2 \
    --learning_rate=0.001 --batch_size=64 --max_trans_ratio=1.5 --gradient_clip=1.0 \
    --beam_size=10 --early_stopping_steps=20 --vocab1_max=30000 --vocab2_max=30000'

testing='--do_testing --device=cpu'

rnnpb_training='--pb_learning_rate=0.01 --binding_strength=1.0 --bind_hard \
    --max_recog_epochs=100 --num_PB=1024 --p_reset=0.10 --dropout=0.40 \
    --model=rnnpbnmt'

train="$run $training $rnnpb_training $train_easy50000 $train_easymono"

$train --working_dir=rnnpb-0.60 --p_mono=0.60 &> rnnpb-0.60/train_log.txt
$train --working_dir=rnnpb-0.50 --p_mono=0.50 &> rnnpb-0.50/train_log.txt
$train --working_dir=rnnpb-0.40 --p_mono=0.40 &> rnnpb-0.40/train_log.txt
$train --working_dir=rnnpb-0.30 --p_mono=0.30 &> rnnpb-0.30/train_log.txt
$train --working_dir=rnnpb-0.20 --p_mono=0.20 &> rnnpb-0.20/train_log.txt
$train --working_dir=rnnpb-0.10 --p_mono=0.10 &> rnnpb-0.10/train_log.txt

$run $testing $test_easy50000 --working_dir=rnnpb-0.60 &> rnnpb-0.60/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb-0.50 &> rnnpb-0.50/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb-0.40 &> rnnpb-0.40/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb-0.30 &> rnnpb-0.30/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb-0.20 &> rnnpb-0.20/test_log.txt
$run $testing $test_easy50000 --working_dir=rnnpb-0.10 &> rnnpb-0.10/test_log.txt
```

M	Sentence
T	Be nice to her .
R	Be nice to her .
S	Be nice to her .
T	How come you know so much ?
R	How come you know so much ?
S	How come you know so much ?
T	There &apos;s something that I want to do before I leave .
R	There &apos;s that that I want to do before I leave .
S	There &apos;s something that I want to do before I leave .
T	I can &apos;t think of anything I &apos;d want from you .
R	I can &apos;t think I &apos;d want to say in you .
S	I can &apos;t think or say I &apos;d want from you .
T	I think it &apos;s better not to try it .
R	I think it &apos;s better not to try it .
S	I think it &apos;s better not not try it .
T	This isn &apos;t so difficult .
R	This isn &apos;t so difficult .
S	This isn &apos;t so difficult .
T	I &apos;ve looked all over for Tom , but I can &apos;t find him .
R	I &apos;ve looked all over for Tom , but I can &apos;t find him .
S	I &apos;ve almost came until for Tom , but I can &apos;t find him .
T	Is he going to make it ?
R	Is he going to make it ?
S	Is he going to make it ?
T	That hurts ! Stop it !
R	That is coming !
S	That happened dangerous chair up !
T	Tom and Mary both know how to swim .
R	Tom and Mary both know how to swim .
S	Tom and Mary both know how to swim .

Table A14: Experiment IV. Autoencoding with (R) RNNPB model and (S) seq-to-seq model, using monolingual data for the RNNPB model. T denotes the true sentence. The seq-to-seq sentences are the same as in table A4.