Master's thesis

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

Janine Rugayan

A Deep Learning Approach to Spoken Language Acquisition

Master's thesis in Electronic Systems Design
Supervisor: Torbjørn Karl Svendsen
June 2021

Norwegian University of
Science and Technology

Janine Rugayan

A Deep Learning Approach to Spoken Language Acquisition

Master's thesis in Electronic Systems Design
Supervisor: Torbjørn Karl Svendsen
June 2021

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Electronic Systems

Norwegian University of
Science and Technology

Abstract

The process of human spoken language acquisition is still being studied up to this day—the most popular theory from B.F. Skinner describes the language learning of infants as a verbal behavior controlled by consequences. This thesis explores the possibility of applying the same principle to machines by creating a system that simulates spoken language acquisition using reinforcement learning.

The developed system is mainly comprised of unsupervised word segmentation and language learning. Vector-Quantized Autoregressive Predictive Coding (VQ-APC) model is utilized to implement unsupervised word segmentation. While the language learning part is implemented using the reinforcement learning method called deep Q-network (DQN). The input to the system is a combined sound file consisted of randomly shuffled utterances of digits ”zero” to ”nine”, and various background noises. It is akin to what an infant would hear during the early stages of learning a language. The virtual agent learns the meanings of the discovered spoken digits through accomplishing the task of ”reciting” them in ascending order.

Different experiments were executed to test the system. The best results for word segmentation were achieved using the VQ-APC model with the WordSeg Adaptor Grammar (AG) algorithm. Moreover, increasing the recognition rate of the word segmentation was observed to improve the reinforcement learning results only to a certain degree. Finally, it was found that large action space sizes can hinder DQN model convergence.

In summary, the thesis achieved spoken language acquisition in machines in line with Skinner’s theory by performing unsupervised word segmentation on a long speech clip and employing reinforcement learning to ground the discovered spoken words. Moreover, it managed to utilize VQ-APC for unsupervised word segmentation and discovered factors that can affect reinforcement learning performance.

Keywords: Reinforcement learning, unsupervised word segmentation, deep-Q network, spoken language acquisition, vector-quantized neural networks

Acknowledgment

Firstly, I would like to sincerely thank my supervisor, professor Torbjørn Karl Svendsen for all the guidance and help he has provided throughout this master thesis. I would also like to extend my appreciation to Andy Chung and Hermann Kamper for answering all my queries and sharing their knowledge with me.

Next, I would like to thank Matúš Košút for all the support he has provided me. You've been a rock when things get super stressful. Also, thanks for being a nerd and helping me when I get errors in my code.

Also, thanks to my second family here in Europe, my friends Kevin, Romeo, Patrick, Eunice, Marvin, Jorge, and Priye. Thanks for all the adventures we had, and looking forward to a lot more in the future.

Finally, I would like to thank my family, especially my parents, for their undying love and support, and my sister for her never-ending teasing that makes my life fun. None of this would have been possible without them.

Contents

List of Figures . . . xi

List of Tables . . . xiii

1 Introduction . . . **1**

1.1 Problem Description . . . 1

1.2 Motivation . . . 2

1.3 Thesis Scope . . . 2

1.4 Thesis Outline . . . 3

2 Theoretical background . . . **5**

2.1 Reinforcement Learning . . . 5

2.1.1 Markov Process . . . 6

2.1.2 Markov Reward Process . . . 6

2.1.3 Markov Decision Process . . . 8

2.2 Q-Learning . . . 10

2.2.1 Tabular Q-learning . . . 11

2.2.2 Deep Q-Network . . . 11

2.3 Vector-Quantized Autoregressive Predictive Coding . . . 13

2.3.1 Autoregressive Predictive Coding . . . 13

2.3.2 Vector-Quantized Autoregressive Predictive Coding . . . 14

2.4 VQ Segmentation . . . 16

2.5 WordSeg . . . 18

2.5.1 Transitional Probabilities . . . 18

2.5.2 Adaptor Grammar . . . 19

2.6 Embedded Segmental K-means Model 21
3 Related work **24**
4 Methodology **27**
4.1 The Task 27
4.2 Learning Method 27
5 Implementation **29**
5.1 Speech Data 29
5.2 Architecture 29
5.2.1 Unsupervised Word Segmentation 29
5.2.2 Language Learning 33
6 Experiments and results **37**
6.1 Modifying the code book size of the VQ-APC model 37
6.1.1 Setup 37
6.1.2 Results 38
6.2 Simulated word segmentation results 45
6.2.1 Setup 45
6.2.2 Results 46
6.3 VQ-APC versus ES k-means for word segmentation 55
6.3.1 Setup 55
6.3.2 Results 55
7 Discussion and Conclusion **58**
7.1 Discussion 58
7.2 Conclusion 61
7.3 Future work 61
Appendices **66**
A Diagrams and Plots **67**

B Tables **69**

List of Figures

1 Model for spoken language acquisition in machines. 2
2 General overview of the system developed for the master's thesis. . . . 3
3 Reinforcement learning diagram. 5
4 Illustration of MDP transition matrix from Lapan, *Deep Reinforcement Learning Hands-On*. 8
5 VQ-APC diagram from Chung et.al., *Vector-Quantized Autoregressive Predictive Coding*. 14
6 Embedded segmental K-means diagram from Kamper et al., *An embedded segmental K-means model for unsupervised segmentation and clustering of speech*. 21
7 Diagram of spoken language acquisition using reinforcement learning. . 28
8 General overview of the system architecture. 30
9 Reinforcement learning results using WordSeg AG for codebook sizes 128, 256, and 512. 39
10 Reinforcement learning results using WordSeg TP for codebook sizes 128, 256, and 512. 42
11 Comparison of all reinforcement learning results for codebook sizes 128, 256, and 512. 44
12 Reinforcement learning results for simulated word segmentation with recognition rate from 10% to 100%. 50
13 Reinforcement learning results for simulated word segmentation with recognition rate from 10% to 20%. 51
14 Reinforcement learning results for simulated word segmentation with recognition rate from 50% to 100%. 52

15 Reinforcement learning results for simulated word segmentation with different action space sizes. 53
16 Reinforcement learning results for simulated word segmentation with action space sizes from 1000 to 1300. 54
17 Reinforcement learning results for VQ segmentation and ES k-means. . . 57
18 System architecture overview showing the main processes and the corresponding input and output. 68

List of Tables

1 Default parameters for WordSeg AG. 20
2 Parameters for the VQ-APC model initialization and training. 30
3 Parameters for the phone segmentation algorithm. 32
4 Parameters for WordSeg TP segment function. 32
5 Parameters for WordSeg AG segment function 33
6 Hyperparameters for the deep Q-network. 35
7 Segmentation results using WordSeg AG and codebook size 128. 38
8 Segmentation results using WordSeg AG and codebook size 256. 38
9 Segmentation results using WordSeg AG and codebook size 512. 40
10 Average of segmentation results using WordSeg AG. 40
11 Summary of segmentation results using WordSeg TP. 40
12 Settings for simulating word segmentation results with varying recognition rates. 45
13 Settings for simulating word segmentation results with varying number of segments or action space sizes. 46
14 Settings for simulating recognition rates 12% to 18%. 47
15 Mean and standard deviation of reinforcement learning results from the simulated word segmentation with varying recognition rates. 48
16 Settings for simulating action space sizes 1100 to 1300. 48
17 Mean and standard deviation of reinforcement learning results from the simulated word segmentation with different action space sizes. 49
18 Segmentation results using ES k-means for word segmentation. 55
19 Average segmentation results of VQ segmentation and ES k-means. . . 56

20 Segmentation results using WordSeg AG and codebook size 128. 69
21 Segmentation results using WordSeg AG and codebook size 256. 70
22 Segmentation results using WordSeg AG and codebook size 512. 70
23 Segmentation results using WordSeg TP. 71
24 Segmentation results using ES k-means for word segmentation. 71

Abbreviations

ACORNS Acquisition of communication and recognition skills. 25

AG Adaptor grammar. 18, 19, 20, 32, 33, 37, 38, 41, 43, 55, 58, 61

APC Autoregressive Predictive Coding. 13, 14, 15, 31

API application programming interface. 33

ASR automatic speech recognition. 28, 33

BTP backward transitional probabilities. 18, 19

DQN Deep Q-Network. 2, 3, 11, 12, 28, 34, 35, 37, 38, 41, 43, 45, 46, 55, 59, 61

ES Embedded Segmental. 2, 21, 22, 55, 56, 60, 61

FTP forward transitional probabilities. 18, 19, 32

LM Language models. 13

MDP Markov Decision Process. 8, 9

MI mutual information. 18, 19

MP Markov Process. 6

MRP Markov Reward Process. 6, 7, 8, 9

RNN Recurrent Neural Network. 13, 30, 31

SGD stochastic gradient descent. 11, 12, 35, 59

TP Transitional probabilities. 18, 19, 32, 33, 37, 41, 58, 59

VQ vector quantization. 3, 15, 16, 30, 31, 32, 55, 56, 58, 60, 61

VQ-APC Vector-Quantized Autoregressive Predictive Coding. 3, 13, 14, 15, 16, 29, 37, 60, 61

1 Introduction

Human spoken language acquisition is still being researched up to this date. A theory that has been widely accepted is B.F. Skinner's verbal behavior. It explains how children can learn a language from scratch. It extends his general theory of conditioning called operant conditioning, wherein the organism releases an operant even without perceiving a stimulus. The operant or response is learned by the organism through reinforcement. A basic example is an infant that needs to drink milk. It learns to say "want milk" due to repeated occurrences of receiving milk whenever it utters the phrase. The verbal behavior is conditioned through the reward received by the infant.

Based on Skinner's theory, verbal behavior can be considered just like any other behavior wherein the outcome controls it. It is established through positive reinforcement or rewards, or it is diminished by negative reinforcement or punishments [1].

Can we use the same principle to teach robots a language? This master's thesis explores the possibility.

1.1 Problem Description

Two main facets have to be dealt with to delve into spoken language acquisition in machines, and Figure 1 shows a general model for it.

First, it is necessary to have the ability to discover words from a continuous speech in an unsupervised manner. The idea is to mimic the scenario of infants constantly hearing speech and eventually picking up some of the words on their own. The next thing to do is to make sense of these words. The second facet of the endeavor is establishing an approach that would attach meanings to the discovered words. The machine's task is to identify which word segments are valid, learn what they mean, and discard the non-valid segments.

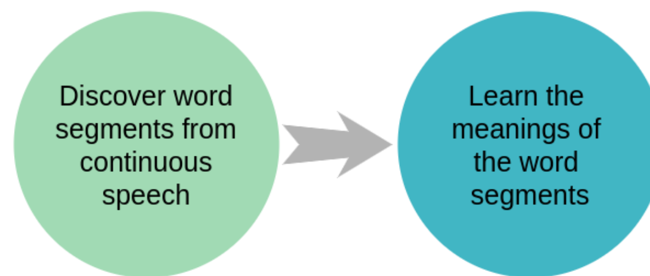


Figure 1: Model for spoken language acquisition in machines.

1.2 Motivation

To this date, there is no governing theory that explains human spoken language acquisition, and the equivalent research in machines is an emerging field itself. There has been much work when it comes to simulating language learning from text. However, it is pretty uncommon to find one that executes it directly from speech signals.

As previously mentioned, the master's thesis adopts Skinner's theory of reinforcing verbal behavior to simulate language learning in machines. The thesis aims to develop a system that would effectively segment continuous speech in an unsupervised manner and simulate language learning using reinforcement learning. The objective is to examine the process of spoken language acquisition in machines and the factors that can influence its performance.

1.3 Thesis Scope

The paper *Spoken Language Acquisition Based on Reinforcement Learning and Word Unit Segmentation* [2] from the Tokyo Institute of Technology is the inspiration for the thesis. They have proposed to implement the two parts of the model in Figure 1 in the following ways:

(i) Word discovery is implemented using Embedded Segmental (ES) K-means model [3] to segment the combined speech file in an unsupervised way.

(ii) Language learning is implemented using reinforcement learning through the Deep Q-Network (DQN) algorithm [4] which combines Q-learning and a deep neural network.

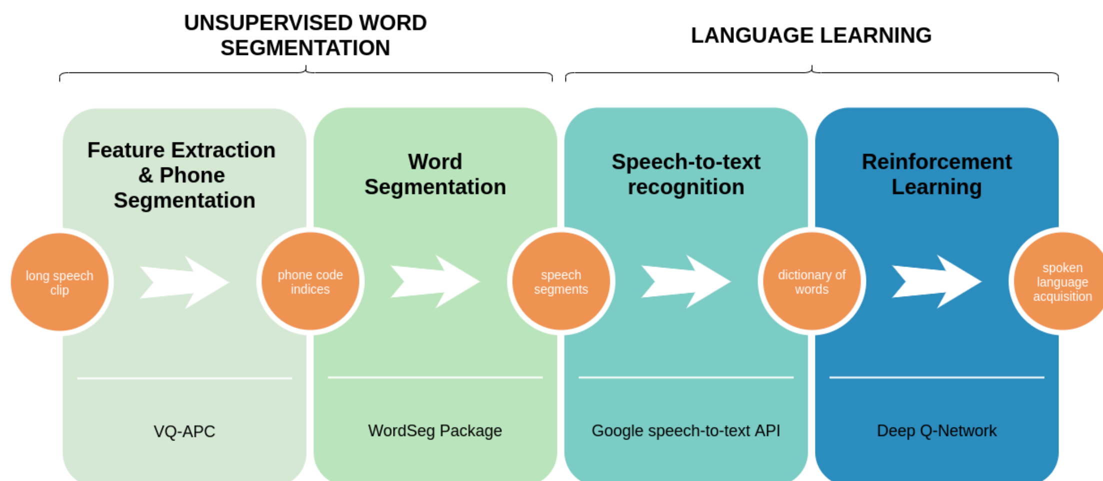


Figure 2: General overview of the system developed for the master's thesis.

The master's thesis involves the modification of these two parts. A general overview of the system developed is illustrated in Figure 2.

For the word discover part, the thesis implements a segmentation method based on vector quantization (VQ). A novel architecture called Vector-Quantized Autoregressive Predictive Coding (VQ-APC) [5] is used to train a vector-quantized neural network. The trained model would be used for feature extraction of the speech signal, while the model's codebook is used to perform phone segmentation and assign code indices to each phone segment. Subsequently, word segmentation is performed with the Word-Seg package [6] using the phone segment indices as input.

For the language learning part, a new task is defined for the agent wherein it needs to enumerate in ascending order the digits "zero" until "nine". As such, the definitions of the agent and the environment for the DQN algorithm are modified. It should be noted that the speech-to-text recognition block in Figure 2 transforms an audio waveform into its symbolic equivalent, which is text. It does not attach any meaning to the words.

1.4 Thesis Outline

The rest of the thesis is organized in the following manner. Section 2 presents the theoretical background discussing concepts related to reinforcement learning and vector-quantized neural networks. Related works follow it in section 3. Then, the details for the methodology, such as the task and the learning method, are elaborated in sec-

tion 4. The implementation is discussed in section 5, wherein crucial parameters are highlighted. Then, various experiments are documented in section 6. The setup for each experiment is defined and followed by the presentation of the results. Finally, a concluding discussion is given in section 7, along with some proposals for future work. There are supplementary diagrams and tables included in the appendices.

2 Theoretical background

2.1 Reinforcement Learning

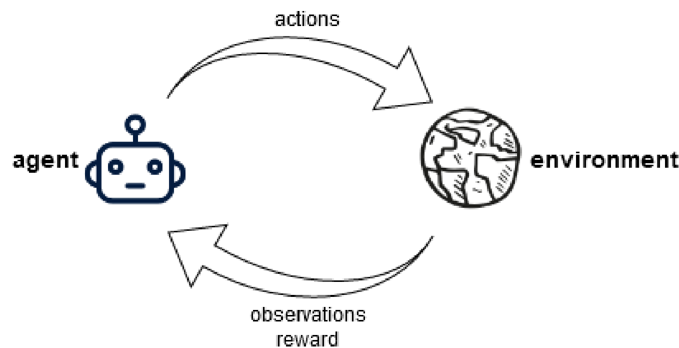


Figure 3: Reinforcement learning diagram.

Reinforcement learning allows the improvement of machine performance over time. It is an approach wherein intelligent programs, called agents, perform actions in a known or unknown environment [7].

The agent and the environment are the two major components of reinforcement learning. The agent interacts with the environment through performing actions and gathering observations. It's aim is to solve a problem and perform the solution in the most efficient way. On the other hand, the environment is everything outside the agent. It provides observations and feedback to the agent. It can be 2-dimensional or 3-dimensional worlds, or game-based scenarios [8].

The environment provides either positive or negative feedback to the agent depending on the action it performed. Through continuous interaction with the environment, the agent adapts and learns based on the feedback it has received [7].

Other components of reinforcement learning are actions, rewards and observations. These are illustrated in Figure 3. Actions, which can either be discrete or continuous,

are the things that can be executed in the environment by the agent. While rewards and observations are communication channels between the agent and the environment.

Rewards is a way for the environment to provide feedback to the agent about the success of its latest activity. It is a scalar value that can be negative or positive. The agent obtains rewards whenever it interacts with the environment, or they can be given by the environment during specific timestamps. The goal of the agent is to accumulate the largest total reward through the series of actions it executes. This is what motivates the learning process of the agent. While the second communication channel for the agent is observations, through which the environment provides the agent information about what is happening around it [8].

The following sections discuss concepts that form the theoretical foundation of reinforcement learning. It starts with the simplest one, Markov Process, which, when expanded to include rewards, turns into a Markov reward process. Another layer of complexity is added by including actions in the Markov reward process, transforming it to a Markov Decision Process.

2.1.1 Markov Process

The Markov Process (MP), also known as the Markov chain, is a system that conforms to the Markov property. Any observations made of the system is referred to as **states**. While *state space* is the set of all the possible states for the system. In MP, the state space needs to be finite. Over time, a sequence of observations forms a chain of states which is referred to as *history*.

The Markov property states that from any observable state, the future dynamics of the system is dependent only on the state itself. As such, the property requires unique and distinguishable states. By fulfilling the Markov property, the future dynamics of the system can be modelled with just one state, and not requiring the whole history. A **transition matrix** is used to summarize the transition probabilities between states into a square matrix. The size of the matrix is NxN, where N denotes the number of states. In row i and column j of the matrix, each cell holds the probability of the system moving from state i to state j [8].

2.1.2 Markov Reward Process

The MP model is expanded by adding value to the transitions form one state to another. In this way, rewards are introduced, and the simple MP becomes a Markov Reward Process (MRP).

In particular, there are two components added to the model, namely - reward and discount factor. Reward is just a number which can be large or small, positive or negative. It can take on different types of representation. However, the most common way is to present it as a matrix like the transition matrix. Row i and column j contains the rewards for changing over from state i to state j [8].

On the other hand, the discount factor γ (gamma) is a single number that depicts the agent's foresightedness. Its value can range from 0 to 1. To understand its purpose, a return value at time t for every episode is to be examined. **Return** is calculated using the formula [8]:

$$G_t = R_{t+1} + \gamma R_{t+2} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}. \tag{2.1}$$

Equation 2.1 computes the return for every time step as a sum of succeeding or future rewards. However, rewards that are k time steps away from the starting point t are multiplied by the discount factor γ raised to k. Inspecting the function of γ, it can be seen that if $\gamma = 1$, then the return value G_t would just be equal to the sum of all future rewards. This means that the agent can perfectly see any future rewards. Conversely, if $\gamma = 0$, then the return value G_t would just be the immediate reward without any consideration for any future rewards. The agent has complete *short-sightedness*. In functional applications, the values for the discount factor is commonly set in between 0 and 1, like 0.9 or 0.99. The discount factor can be thought of as a measure of how much the agent looks into the future when estimating the future return. As γ gets closer to 1, more of the future steps are taken into consideration [8].

The **return** value is not very practical because it is defined for every chain observed from the MRP. As such, it can extensively diverge even for the same state. A much more practical quantity is the **value of state**. It is defined as the mathematical expectation of return for any state [8], where:

$$V(s) = \mathbb{E}[G|S_t = s]. \tag{2.2}$$

Equation 2.2 shows that the value of state $V(s)$ for every state s is the expected or average return acquired by going through the MRP.

2.1.3 Markov Decision Process

In order to transform MRP into a Markov Decision Process (MDP), actions are added into the model. The first consideration is to have a finite set of actions, also referred to as the agent's *action space*.

An extra dimension is required for the transition matrix in order to include action. The agent is no longer an uninvolved observer of the state transitions, but now has the power to choose with action to take at every time step [8].

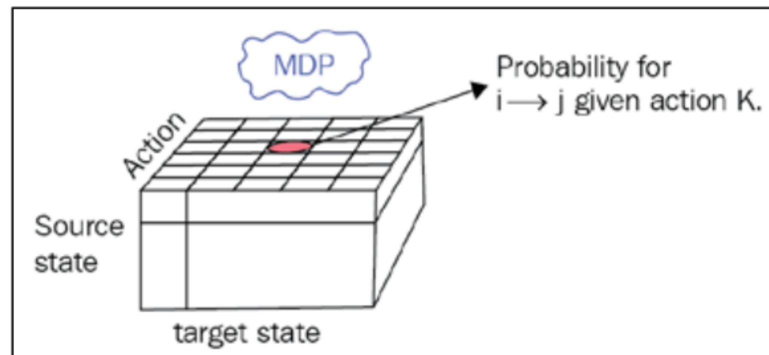


Figure 4: Illustration of MDP transition matrix from Lapan, *Deep Reinforcement Learning Hands-On*.

To better visualize the addition of the action dimension, Figure 4 shows a 3-dimensional transition matrix. The depth dimension encompasses the possible actions (k) the agent can choose to take. The height dimension is the source state (i), and the width is the target state (j). When the agent chooses an action, the probabilities of the target states can be altered. By having a 3D transition matrix, the MDP can cover all the intricacies of the environment and its range of possible feedback to the agent's actions.

Furthermore, to completely turn MRP into a MDP, the reward matrix is updated as well with the addition of actions, like that of the transition matrix. As such, the attainable reward is dependent on the agent's state and the action it has chosen to end up in this state [8].

Another main concept for MDP and reinforcement learning is the **policy**. It is defined as the set of rules that determine how the agent acts in the environment. It determines the amount of return obtained by the agent. So, it is vital that a good policy is found since it ensures that the agent's goal of accumulating the largest return is achieved.

The formal definition of policy is as follows:

$$\pi(a|s) = P[A_t = a|S_t = s], \tag{2.3}$$

wherein it is the probability distribution over actions given every possible state [8]. If the policy is constant, the MDP reduces to MRP. The transition and reward matrices will not have the action dimension.

2.2 Q-Learning

The reinforcement learning method used for this project is Q-learning. Its basic principle is encompassed by the Bellman Equation defined as

$$V_0 = \max_{a \epsilon A} E_{s \sim S}[r_{s,a} + \gamma V_s], \tag{2.4}$$

where V_0 is the value of the state, $r_{s,a}$ is the reward, γ is the discount factor, and V_s is the value of the next state [8]. Equation 2.4 characterizes the ideal value of the state V_0 as the action which maximizes the immediate expected reward $r_{s,a}$ plus the discounted one-step long-term reward V_s. These values of the state not only provides the best attainable reward but also the best policy that achieves this reward. With the knowledge of every state's value, the agent will be able to map the actions that will lead to earning the largest possible reward.

In Q-learning, the value of action $Q(s, a)$ is considered. It indicates the total reward that can be earned in state s by executing action a. It is defined by the equation,

$$Q(s, a) = E_{s' \sim S}[r_{s,a} + \gamma V_{s'}], \tag{2.5}$$

wherein the Q-value is equivalent to the expected immediate reward $r_{s,a}$ plus the discounted long-term reward $\gamma V_{s'}$ for the target state [8]. By using the Bellman approximation, the resulting Q-values are frequently very similar because the current state and the target state are only one step apart. The Q-value of the state-action pair can also be expressed via itself through the following equations [8]:

$$V_s = \max_{a \epsilon A} Q_{s,a} \tag{2.6}$$

$$Q(s, a) = r_{s,a} + \gamma \max_{a' \epsilon A} Q(s', a'). \tag{2.7}$$

The value of state can defined using the value of action as seen in Equation 2.6, wherein it is equivalent to the action that maximizes the Q-value. Using this same principle for the value of the destination state, it is seen in Equation 2.7 that the Q-value of the state-action pair can be expressed via itself.

2.2.1 Tabular Q-learning

One method of Q-learning is tabular Q-learning wherein a mapping of the states and their corresponding Q-values are stored in a table. The algorithm starts with an empty table for the Q-values. Then, during each interaction with the environment, the agent acquires the data for the state, action, reward, and new state. At this point, the agent decides which action to take. Then the Q-values are updated using the Bellman approximation with the learning rate α incorporated as follows [8]:

$$Q_{s,a} \leftarrow (1-\alpha)Q_{s,a} + \alpha(r + \gamma \underset{a' \epsilon A}{max} Q_{s',a'}). \tag{2.8}$$

The learning rate allows the old and new Q-values to be combined. Its value can range from 0 to 1. As seen in Equation 2.8. The old Q-values are incorporated as $(1-\alpha)Q_{s,a}$. While the new Q-values are incorporated as $\alpha(r + \gamma \underset{a' \epsilon A}{max} Q_{s',a'})$. Simply replacing the old Q-values with the new ones can cause training to become unstable. The whole process is repeated until the condition for convergence is met [8]. However, tabular Q-learning struggles when the state space is very large. For this case, deep-Q learning is more suitable .

2.2.2 Deep Q-Network

In deep Q-learning, values are mapped to state-action pairs using a non-linear representation, which is approximated and trained using deep neural networks [8]. Henceforth, it is referred to as Deep Q-Network (DQN). For a successful training, the epsilon-greedy method, replay buffer, and target network need to be implemented.

Firstly, the epsilon-greedy method solves the exploration versus exploitation dilemma. The epsilon-greedy algorithm makes it possible for the agent to switch between deciding randomly and deciding based on the policy Q-network [8]. At the beginning of training when the Q-values are still not fine-tuned, it is better for the agent to act randomly as it allows the gathering of information about the environment states in a uniformly distributed manner. However, as the training progress, the Q-values are more calibrated, and makes it more efficient to decide based on this rather than acting randomly.

Next, the replay buffer enables the implementation of the stochastic gradient descent (SGD) algorithm for updating the Q-values [8]. The training data available for the SGD update does not fulfill the requirement of being independent and identically distributed. The data samples are gathered during the same episode, thus, making

them very close to each other. Moreover, the training data available does not have the same distribution as the sample data of the optimal policy, but instead has a distribution based on the current policy. The replay buffer mitigates this problem by storing past experiences from different episodes, and using this buffer as source for the training data instead of sampling it from the latest experience.

Lastly, the target network makes training of the neural networks more stable by using a copy of the policy network for the target Q-values [8]. As mentioned previously, the Q-values in the Bellman approximation are usually very similar because they are only one step apart. By synchronizing the target network with the policy network only once every N steps, the target network will have Q-values that are N steps apart from the policy network Q-values.

The whole DQN algorithm used in this project is based on the paper of Mnih et al. [4]. It uses two deep neural networks to estimate the Q-values. One is used for the policy Q-network, and the other is used for the target network. The policy Q-network Q is used to decide which action to take. It has weights denoted by θ. On the other hand, the target network $\hat{Q}$ is used to generate the target Q-values for learning. It has weights denoted by θ^{-}. Every X number of updates, the weights θ from the policy network Q are copied to the target network $\hat{Q}$.

$$y = r + \gamma \underset{a' \epsilon A}{max} \hat{Q}(s', a'; \theta^{-}) \tag{2.9}$$

$$L(\theta) = (y - Q(s, a; \theta))^2 \tag{2.10}$$

Equation 2.9 [2] above denotes the Bellman approximation of the target Q-value, where the reward for the current action a is r, the discount factor is γ, and the expected state and action for the next step are s' and a', respectively. SGD is used to update the weights θ of the policy network Q. The goal is to minimize the loss given in Equation 2.10 [2] as the difference between the target Q-value y, and the current Q-value.

2.3 Vector-Quantized Autoregressive Predictive Coding

VQ-APC is a model from Chung et al. that produces encoded representations wherein the amount of information contained can be modified based on the size of the codebook that quantizes the speech signal [5]. As the name implies, it is based on Autoregressive Predictive Coding (APC), with the addition of having quantization layers.

2.3.1 Autoregressive Predictive Coding

APC is an architecture developed to facilitate unsupervised learning of speech representations. It focuses on predicting the spectrum of a future frame. Language models (LM) for text highly influences its methodology. Given a sequence of N tokens $(t_1, t_2, ..., t_N)$, a LM assigns a probability to the entire sequence. This probability is derived by modeling the probability of token t_k as:

$$P(t_1, t_2, ..., t_N) = \prod_{k=1}^{N} P(t_k|t_1, t_2, ..., t_{k-1}), \tag{2.11}$$

wherein $t_1, t_2, ..., t_{k-1}$ are the previous tokens prior to t_k. Training is done by minimizing the negative log-likelihood which is defined as,

$$\sum_{k=1}^{N} -logP(t_k|t_1, ..., t_{k-1}; \theta_t, \theta_{rnn}, \theta_s), \tag{2.12}$$

wherein θ_t, θ_{rnn}, and θ_s are the parameters for optimization. The look-up table for mapping tokens into a vector is denoted by θ_t. On the other hand, the history of token sequences up to the current time step is summarized by a Recurrent Neural Network (RNN) denoted by θ_{rnn}. Finally, at the output of each RNN time step, a Softmax layer is appended to estimate the probability distribution over the tokens. This is denoted by θ_s [9].

The concept of neural LM described above is the inspiration for APC. The acoustic sequence's temporal information is modeled by a RNN. However, the look-up table is not required in APC because each frame in the speech data is considered as one token t_k, instead of considering each word or character. These frames are then fed directly into the RNN θ_{rnn}. Furthermore, the Softmax layer is replaced by a regression layer θ_r because there is no fixed set of target tokens. This results to using linear mapping at each time step as the RNN output tries to match the target frame. In summary, the parameters to optimize in APC are θ_{rnn} and θ_r [9].

The APC model is set to predict n frames into the future instead of just predicting the next one. It ensures that the model has a more universal structure, and not focus on the localized information. The speech signal's feature vectors $(x_1, x_2, ..., x_T)$ is the input to the model. For each feature vector x_t from the sequence, the model outputs a prediction y_t. These two vectors have the same dimension. Optimization of the model is implemented by minimizing the L1 loss denoted as:

$$\sum_{i=1}^{T-n} |x_{i+n} - y_i| , \tag{2.13}$$

wherein $(x_1, x_2, ..., x_T)$ is the sequence of input feature vectors, $(y_1, y_2, ..., y_T)$ is the predicted sequence, and T is the sequence length [9]. It is illustrated in Equation 2.13 that L1 loss is the difference between the target future frame and the predicted frame.

2.3.2 Vector-Quantized Autoregressive Predictive Coding

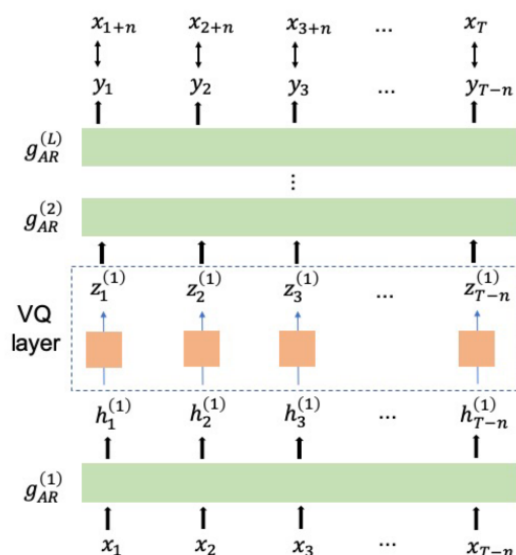


Figure 5: VQ-APC diagram from Chung et.al., *Vector-Quantized Autoregressive Predictive Coding*.

By adding quantization layer(s) to APC, the VQ-APC architecture is achieved. As discussed earlier, APC uses an autoregressive neural model g_{AR} to capture the temporal information of the acoustic sequence. Features can be extracted by taking the hidden representations of g_{AR} after it is trained.

To create VQ-APC, consider g_{AR} is made up of L layers, and the l-th layer is denoted as $g_{AR}^{(l)}$. Each $g_{AR}^{(l)}$ layer is able to produce a sequence of hidden vectors $h^{(l)}$ based on the input feature vector sequence to g_{AR}. In addition, a VQ layer is appended after any of the $g_{AR}^{(l)}$ layers. This transforms elements of the hidden vectors into an equivalent one based on a codebook. For example, at time t, the hidden vector element $h_t^{(l)}$ becomes $z_t^{(l)}$ which is one of the elements in codebook $c_1, ..., c_V$. The next layer $g_{AR}^{(l+1)}$ receives the resulting quantized hidden vectors as the input. The feed-forward process then continues. An example is shown in Figure 5, where the VQ layer is inserted after the first layer. VQ-APC is trained the same way as APC wherein the objective is to minimize the L1 loss [5].

To determine the discrete codebook variables, Gumbel-Softmax with the straight-through estimator [10] is used such that it can be done in a fully differentiable way. Explicitly, the hidden vector $h_t^{(l)}$ is mapped to a vector $r \,\epsilon\, \mathbb{R}^V$ using a linear layer. During testing, the codebook variable is chosen by taking the largest element in vector r. During training, the i-th code variable c_i is selected with a probability p_i denoted as:

$$p_i = \frac{e^{(r_i+v_i)/\tau}}{\sum_{j=1}^{V} e^{(r_j+v_j)/\tau}}, \tag{2.14}$$

where $v = -ln(-ln(u)) \,\epsilon\, \mathbb{R}^V$. By sampling $\mu(0,1)$ uniformly, the value of u is obtained. On the other hand, the approximation's closeness to argmax is determined by the value of τ. The code c_k is chosen during the forward pass based on $k = argmax_i p_i$. While the true gradients of the Gumbel-Softmax outputs are used during the backward pass [5].

2.4 VQ Segmentation

The previously discussed VQ-APC model is utilized to extract the feature vectors of speech signals. In order to perform segmentation on the continuous sequence of feature vectors, a constrained optimization problem is to be resolved.

One way to implement segmentation is to divide the continuous speech representation based on minimizing the sum of the squared Euclidean distances between the feature vectors and the representative code of each segment. However, if this is the only criterion followed, then the best segmentation would be to place each feature vector into its own segment, and assign it the code which is closest to the feature vector. Likewise, it ends up functioning as a standard VQ layer. As such, a constraint in the form of duration penalty is introduced to encourage longer and fewer segments. Additionally, a maximum limit on the number of contiguous frames or feature vectors in the segment is put into place.

The VQ segmentation algorithm looks for the optimal segmentation that would minimize the error function, $argmin_{s_{1:M}} E(\boldsymbol{z}_{1:T}, s_{1:M})$, where $(\boldsymbol{z}_1, \boldsymbol{z}_2, ..., \boldsymbol{z}_T)$ signifies the sequence of feature vectors, and $(s_1, s_2, ..., s_M)$ are the segments produced. Each segment s_i is an aggregation of $|s_i|$ feature vectors from the sequence $\boldsymbol{z}_{1:T}$, and it is assigned a corresponding code vector $\hat{\boldsymbol{z}}_{s_i}$ from the VQ codebook. The following equation elaborates on the error function to be minimized:

$$E(\boldsymbol{z}_{1:T}, s_{1:M}) = \sum_{s_i \epsilon s_{1:M}} \sum_{\boldsymbol{z}_j \epsilon s_i} [\|\boldsymbol{z}_j - \hat{\boldsymbol{z}}_{s_i}\|^2 + \lambda pen(|s_j|)]. \tag{2.15}$$

As seen from Equation 2.15, a penalty term is added to the squared Euclidean distance between the feature vector and the code vector of the segment it belongs to. The term $pen(|s_j|)$ is the penalty for $|s_j|$ frames in the segment. While, λ denotes the penalty weight [11].

Finding the reasonable segment lengths and minimizing this error function is a constrained optimization problem. It is solved using dynamic programming. Forward variables α_t is defined as $min_{s_1:M_t} E(\boldsymbol{z}_{1:t}, s_{1:M_t})$ which is the optimal segmentation's error up to step t. The following equation is used to calculate this:

$$\alpha_t = \min_{j=1}^{t} \left\{ \alpha_{t-j} + \min_{k=1}^{K} \sum_{\boldsymbol{z}_i \epsilon \boldsymbol{z}_{t-j+1:t}} \left[\|\boldsymbol{z}_i - \boldsymbol{e}_k\|^2 + \lambda pen(j) \right] \right\}. \tag{2.16}$$

It is done recursively and starts with $\alpha_0 = 0$. While the succeeding α_t for step $t = 1, ..., T-1$ are calculated according to Equation 2.16. For each α_t, the resulting arg min is noted. Then, from the final position $t = T$ and moving towards $t = 0$, optimal boundaries are chosen repeatedly. This process achieves the overall optimal segmentation [11].

2.5 WordSeg

WordSeg is an open-source software package that is aimed towards the standardization of unsupervised word segmentation from text by allowing the easy reproduction of results, and stimulating the growth of cumulative science in this field of study. It has two main use cases. First, it can be used for the development of another unsupervised word segmentation algorithm. Second, it can be used by linguists and other cognitive scientists for their study of early language acquisition [12].

The package accepts as input a prepared text containing the phonemized or syllabified version of the original text or transcription. Afterwards, the segmentation process is modelled based on the chosen segmentation algorithm. There are six algorithms available in the package. Lastly, the package also includes evaluation tools to assess the performance of algorithms [6].

This section discusses Transitional probabilities (TP) and Adaptor grammar (AG), which are two types of word segmentation algorithms from the WordSeg package. Each one is used separately to perform experiments in section 6. They are utilized for unsupervised word segmentation with various input scenarios, and the results from each algorithm is examined and compared.

2.5.1 Transitional Probabilities

TP is a sublexical algorithm which primarily bases its word segment boundaries on local cues like the occurrence of particular sound sequences around word boundaries. It works by differentiating among phone or syllable sequences which are approximately *internally cohesive* [12].

There are three ways to calculate TP. For a given sequence XY, the forward transitional probabilities (FTP), backward transitional probabilities (BTP), and mutual information (MI) can be calculated. The FTP is acquired by taking the frequency of XY and dividing it by the frequency of X. The BTP is acquired by taking the frequency of XY and dividing it by the frequency of Y. Lastly, the MI for XY is denoted by:

$$log_2\left(\frac{frequency\,XY}{(frequency\,X)(frequency\,Y)}\right) \tag{2.17}$$

Moreover, there are two options for identifying word boundaries. The first option is a *relative* threshold which uses relative dips in TP to determine the word boundaries. For example in the phone sequence ABCD, a boundary is assumed to occur between

B and C if the TP for sequence AB and CD are higher than that of BC. The second option is an *absolute* threshold which uses the average TP of the entire corpus as the limit for boundary detections. For both of these options, it is not required to have any knowledge of the word boundaries [12].

WordSeg package's TP accepts as input a prepared text file containing the phonemized or syllabified version of the original transcript. The *segment* function of TP starts with creating the test units using the prepared text input. If there is no train text, the test units are used as the train units. Afterwards, the transition probabilities are estimated using the train units. This is done by first calculating and counting all the unigrams and bigrams in the sequence. Next, it calculates the transitional probabilities using the train units based on the chosen dependency, which can be FTP, BTP, or MI.

Then, the prepared text input is segmented using the calculated transitional probabilities for all the bigrams. It takes the test units and identifies word boundaries based on the chosen threshold, which can be *relative* or *absolute*. This is done continuously until all the units are inspected. Finally, the *segment* function returns a set of phones or syllables grouped together as words [6].

2.5.2 Adaptor Grammar

AG is a lexical algorithm wherein deduced probabilities of how a set of ”grammar” rules is used for the creation of the corpus posits the manner by which the corpus will be parsed [12]. For example, there are particular words that would more likely appear consecutively and the algorithm exploits this. It parses the whole utterance again such that there is a minimum number of recombinable units.

By default, the package is able to generate the simplest and most universal grammar which is generated through various rewrite rules. One of the rules is that ”sentences are one or more words”, and another is that ”words are one or more basic units”. Lastly, one is a set of rewrite rules that describes the basic units for all the possible terminals.

Furthermore, there are three subprocesses that comprise the segmentation of the corpus using this algorithm. First, the corpus is parsed based on a set of rules and subrules. This would be done for a number of iterations to account for senseless or wrong parses. Moreover, the first and last iterations are dropped, and only one in a few will be retained. The next subprocess can be considered as the actual segmentation process wherein the parses from the first subprocess are applied once more to the corpus. Finally, the third subprocess uses minimum Bayes risk to find the most prevalent sample segmentations and use this as basis for choosing the solution [12].

There are many parameters that can be set for AG, but it has default values that were based on experiments done on English, Japanese and French adult and child corpora. The parameters are shown in Table 1.

Parameter	Value
number of runs	8
number of sweeps per run	2000
number of sweeps that are pruned	100 at the beginning and end, 9 in every 10 in between
Pitman-Yor a parameter	0.0001
Pitman-Yor b parameter	10000
Rule probability (theta)	estimated using Dirichlet prior

Table 1: Default parameters for WordSeg AG.

These settings are based on what was commonly found in adaptor grammar papers. Number of runs is the amount of times the algorithm is executed before finalizing on the word boundaries. Number of sweeps per run is the number of iterations done for each execution of the algorithm. The Pitman-Yor values are for the Pitman-Yor process which controls the balance between creating and reusing the subrules [12].

The AG segment function starts with creating the test text from the prepared text input. If there is no train text, the test text is used as the train text. First, the function sets up to ignore the first parses produced by the algorithm, and ensures that a different random seed is used for each run. Then, the algorithm generates grammar from the set of phones in the prepared text input and saves it in a temporary file. Using this grammar file, along with the test text and train text, the algorithm is executed repeatedly based on the declared number of runs and number of iterations for each run. Due to the lower accuracy of the first iterations of AG, these are dropped. After each run, the counter for the number of parses produced is updated. At the end of executing all runs, the function returns the chosen segmentation based on the most common parses produced [6].

2.6 Embedded Segmental K-means Model

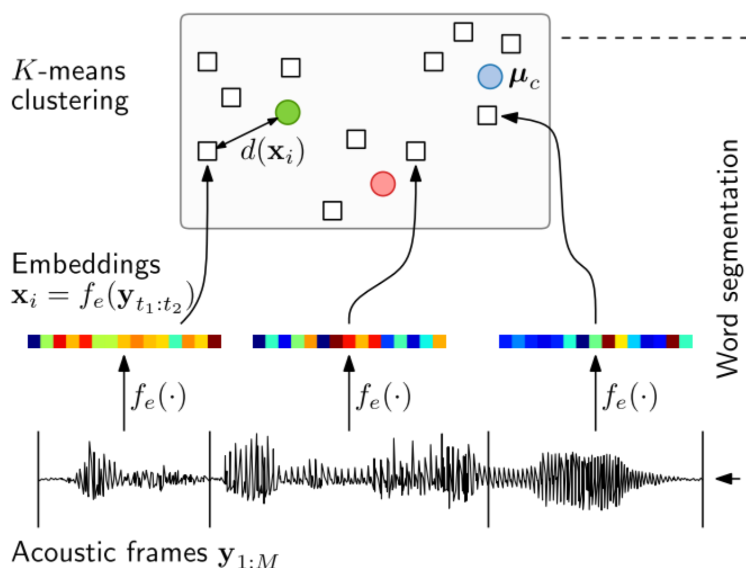


Figure 6: Embedded segmental K-means diagram from Kamper et al., *An embedded segmental K-means model for unsupervised segmentation and clustering of speech*.

In the paper of Gao et al. [2], unsupervised word segmentation is implemented using the ES K-means model which uses hard clustering and segmentation to segment and cluster unlabelled speech in an unsupervised manner.

The objective of the model is to break up a sequence of acoustic frames $\boldsymbol{y}_{1:M} = \boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_M$ (i.e., MFCCs) into word-like segments, and to collect them into assumed word types. If the position of word boundaries are already known, like as shown at the bottom of Figure 6, then an approach to compare these variable-length vector sequences is required in order to cluster them [3].

The ES K-means model adopts an *acoustic word embedding* approach [13, 14, 15] to cluster the segments. Each variable-length speech segment is mapped using an embedding function f_e to a embedding vector $\boldsymbol{x} \epsilon \mathbb{R}^D$ situated in a fixed-dimensional space. To illustrate, word segment with feature vectors $\boldsymbol{y}_{t_1:t_2}$ is mapped to an embedding vector $\boldsymbol{x}_i = f_e(\boldsymbol{y}_{t_1:t_2})$, represented as the colored horizontal figures in Figure 6. The central concept here is that acoustically similar speech segments should be situated close to each other in $\mathbb{R}^D$ [3]. Moreover, the model uniformly downsamples any segment such

that each one is represented by the same quantity of vectors, which are then flattened to acquire the embedding [13].

A set of vectors $X = \{\boldsymbol{x}_i\}_{i=1}^{N}$ is formed after embedding all the segments in the data set. The next step is to group together these segments into K hypothesized word types using K-means, illustrated at the top of Figure 6. In standard K-means, the sum of squared Euclidean distances to each cluster mean is minimized:

$$min_{\boldsymbol{z}} \sum_{c=1}^{K} \sum_{\boldsymbol{x} \epsilon X_c} \|\boldsymbol{x} - \boldsymbol{\mu}_c\|^2 , \tag{2.18}$$

where the cluster means is denoted by $\{\boldsymbol{\mu}_c\}_{c=1}^{K}$, all vectors belonging to cluster c is denoted by X_c, and the cluster to which $\boldsymbol{x}_i$ is assigned to is denoted by the elements of $\boldsymbol{z}$ [3]. This method can only be used if the segmentation is already established. However, this is not the case. Depending on the current segmentation, the set of embeddings X may vary. Given a data set of S utterances, $Q = \{\boldsymbol{q}_i\}_{i=1}^{S}$ represents the segmentations, where the boundaries of utterance i is specified by $\boldsymbol{q}_i$. The embeddings under the current segmentation is represented by $X(Q)$.

The goal of the ES K-means algorithm is to mutually optimize the segmentation Q and the cluster assignments $\boldsymbol{z}$ as:

$$\min_{Q,\boldsymbol{z}} \sum_{c=1}^{K} \sum_{\boldsymbol{x} \epsilon X_c \cap X(Q)} len(\boldsymbol{x}) \, \|\boldsymbol{x} - \boldsymbol{\mu}_c\|^2 . \tag{2.19}$$

A score per frame is assigned as equal to the score achieved by the segment to which the frame belongs to. This suggests the influence of segment duration on the segment scores, thereby resulting to Equation 2.19 showing $len(\boldsymbol{x})\|\boldsymbol{x} - \boldsymbol{\mu}_c\|^2$ as the score of embedding $\boldsymbol{x}$. The term $len(\boldsymbol{x})$ signifies the number of frames in the acoustic sequence used to calculate embedding $\boldsymbol{x}$ [3].

In summary, Kamper et al. describes the ES K-means algorithm as

> The overall ES K-means algorithm starts with randomly assigned word boundaries. It then optimizes Equation 2.19 by going back and forth between optimizing the segmentation Q while keeping the cluster assignments $\boldsymbol{z}$ and means $\{\boldsymbol{\mu}_c\}_{c=1}^{K}$ fixed (top to bottom in Figure 6), and then optimizing the cluster assignments and means while keeping the segmentation fixed (bottom to top in Figure 6) [3].

When the cluster assignments are fixed, then the optimization goal in Equation 2.19

transforms to:

$$\min_{Q} \sum_{\boldsymbol{x} \epsilon X(Q)} len(\boldsymbol{x}) \left\| \boldsymbol{x} - \boldsymbol{\mu}_x^* \right\|^2 = \min_{Q} \sum_{\boldsymbol{x} \epsilon X(Q)} d(\boldsymbol{x}), \tag{2.20}$$

where $\boldsymbol{\mu}_x^*$ is the mean of the current cluster to which x belongs to, and $d(\boldsymbol{x})$ is the score of embedding $\boldsymbol{x}$ [3]. Equation 2.20 is optimized by finding the boundaries $\boldsymbol{q}$ for each utterance that results to the minimum total score of $X(Q)$, the embeddings under the current segmentation. The optimal segmentation is found by using dynamic programming which implements the shortest-path algorithm (Viterbi).

Conversely, when the segmentation Q is fixed, the optimization goal in Equation 2.19 transforms to:

$$\min_{\boldsymbol{z}} \sum_{c=1}^{K} \sum_{\boldsymbol{x} \epsilon X_c \cap X(Q)} len(\boldsymbol{x}) \left\| \boldsymbol{x} - \boldsymbol{\mu}_c \right\|^2 . \tag{2.21}$$

Standard K-means is adopted to find the best assignment of the embeddings to clusters when the means $\{\boldsymbol{\mu}_c\}_{c=1}^{K}$ are fixed [3]. Since the distance between an embedding and its assigned cluster means will never increase, the reassignments are expected to further optimize Equation 2.19.

Eventually, the cluster assignments $\boldsymbol{z}$ are fixed, then the means are updated:

$$\boldsymbol{\mu}_c = \frac{1}{\sum_{\boldsymbol{x} \epsilon X_c} len(\boldsymbol{x})} \sum_{\boldsymbol{x} \epsilon X_c} len(\boldsymbol{x}) \boldsymbol{x} \approx \frac{1}{N_c} \sum_{\boldsymbol{x} \epsilon X_c} \boldsymbol{x}, \tag{2.22}$$

where N_c is the quantity of embeddings currently belonging to cluster c. The mean of cluster c is also expected to further optimize Equation 2.19. The approximation in Equation 2.22 is used since it equates to the exact calculation when the duration for all the segments is identical [3].

3 Related work

The thesis is analogous to the research about the *grounded language acquisition* problem, which pertains to finding a way to learn the meaning of a language predicated on its application to the physical world [16]. Without any substantial interpretation, human language is just a collection of symbols. It acquires its value when it is learned, understood, and utilized in the physical world where humans exist. The related works presented in this chapter aim to perform embodied language learning through virtual agents.

The work of Matuszek conveyed that natural language processing and robotics could improve their efficiency and efficacy if language learning is considered a grounded language acquisition problem. It argued that using concrete applications of the language improves the way it is learned and that robots perform better when the world where they run in is depicted and disambiguated by language. The paper revolved around a case study wherein unconstrained natural language is used by people to teach a robot. Statistical machine learning approaches were formulated such that the robot learns about the objects and tasks in its environment and attains semantics of the language through constant interaction with users [16].

Virtual environments are commonly used as a tool to ground linguistic tokens. Sinha et al. created 2D and 3D environments wherein an agent is tasked to navigate to an object in the environment and follow natural language instructions. They developed an attention mechanism for combining the visual and textual information received by the agent such that it learns to accomplish the given tasks, and it achieves language grounding [17].

Likewise, Hermann et al. presented an agent that learns the language by successfully completing a set of tasks in a 3D environment. The agent received written instructions and was trained through a combination of reinforcement and unsupervised learning. It earned positive rewards if it efficiently worked in the environment while concurrently learning the meanings of phrases and their relationship to the visual cues observed. Additionally, they found that new words were learned faster when some words were already learned [18].

Yu et al. used a 2D maze-like world to teach a virtual agent the language based on two cases. The agent followed navigation instructions and answered questions. The agent had visual information of its surroundings and the textual instructions or questions from a virtual teacher. It received rewards based on the actions it took. These components led the agent to learn about the visual representation of the simulated world, the language, and the action control, all at the same time. Moreover, they found that the agent can predict the meaning of new words or word combinations after learning [19].

All of the previously mentioned work is grounding language using text as the input. On the contrary, the thesis aims to simulate spoken language acquisition. Roy proposed to do this by applying an architecture that would process multi-sensory data. A computational model called CELL (Cross-Channel Early Lexical Learning) was formulated. It learns words by training on untranscribed microphone and camera input and forming a dictionary of audio-visual items. The lexical items were acquired by discovering words from continuous speech, acquiring visual categories, and developing the connection between the word and visual models [20].

Similarly, Yu et al. used multi-sensory inputs and developed a system that mimics the way adults teach children names of objects. Users introduced objects, where they are located, and how they are used. The multimodal learning system collected visual and speech data from the users and automatically learned to construct a mapping between the words and the objects. Furthermore, it learned to put the visual features of the objects into categories by using the corresponding linguistic information as guide [21].

Chauhan et al. tackled spoken language grounding with a learning and categorization approach. There are no predefined sets of words and meanings in their architecture, which leaves it open-ended. Through constant interaction with a user, the virtual agent obtained new words and their corresponding meanings. Much like the research mentioned above, their work revolved around naming objects. The virtual agent was equipped with a camera and a microphone while a user presents objects and uses speech to introduce them. Using the multi-sensory input, the agent learned the meaning of the words. Additionally, their approach was able to exploit the homogeneity of word categories for organizing the object categories [22].

On the other hand, the Acquisition of communication and recognition skills (ACORNS) project, funded by the European Commission, aspired to simulate human language learning in artificial agents by utilizing the memory-prediction model of natural intelli-

gence for speech processing. With this model, speech representations with rich detail are first stored in the lower levels of the neo-cortex. In contrast, speech patterns are saved at higher levels. When sensory inputs consistent with parts of the pattern are detected, the brain 'predicts' and activates the complete pattern. It is also through this approach that new patterns may be detected and saved in the memory. It was intended for the project to come up with a new way to develop virtual agents that can learn human-like verbal communicative behavior [23].

All of the work discussed so far was implemented in a supervised manner in one way or another and did not genuinely correspond to the way humans learn language from an early age. However, another paper from the Tokyo Institute of Technology can simulate spoken language acquisition in an approach that coincides with B.F. Skinner's theory. Zhang et al. proposed a spoken language acquisition system that uses images to make unsupervised learning more focused and implement pre-training. Their method made the reinforcement learning process more efficient. The sound-image grounding concept was inspired by how infants learn by observing the world around them. Their experiments showed that the reinforcement learning's speed is improved and that the software robot successfully acquired spoken language from spoken prompts with dialogues and images [24].

4 Methodology

4.1 The Task

The spoken language acquisition task exhibits the application of Skinner's verbal behavior to machines. The agent is given an ordering task to utter the ten digits in ascending order, from ”zero” to ”nine”.

The agent's initial condition is that it has not ”spoken” any of the digits for the task. It is given a long speech clip containing some noise, and the utterances for ”zero”, ”one”, ”two”, ”three”, ”four”, ”five”, ”six”, ”seven”, ”eight”, and ”nine”. It needs to identify valid words from the speech clip. If the agent utters the correct digit, the environment responds by acknowledging the spoken word. Otherwise, the environment does not respond. The correctness of the utterance depends on what has been spoken already by the agent. For example, during the initial stage, the correct digit that the environment will recognize is ”zero”. On the other hand, if the agent has already uttered ”zero”, then the correct utterance accepted by the environment is ”one”. This scheme goes on until the agent has uttered ”nine”, the last digit in the sequence.

In summary, the agent must independently learn to choose the correct digit to ”speak” based on its current state such that it utters all the ten digits in ascending order most efficiently.

4.2 Learning Method

The process of spoken language acquisition can be summarized in three major steps: forming observations, processing the observations, and grounding the observations. The methodology from Gao et al. [2] is used as a guide for this section. The agent and environment descriptions are tailored for the task defined in section 4.1.

The environment is an empty list, wherein an agent needs to fill up with digits from ”zero” to ”nine”. The agent's state refers to how much of the digit list it has filled up. An action is each instance the agent ”speaks” to the environment. The initial condition for the agent is akin to that of a newborn child wherein it does not have any existing knowledge of the language.

Firstly, the agent's observation of the environment is represented by the long speech input it receives. This observation is comparable to what babies usually hear when they start learning a language. Then, the agent processes this observation by identifying possible words and segments the long speech input. This stage is implemented using an unsupervised word segmentation based on vector quantization. Lastly, the agent needs to ground the observations through reinforcement learning. The interaction of the agent with the environment is realized through the DQN. The learning loop runs until the agent can perform the task efficiently by correctly choosing the words to ”speak” based on its current state.

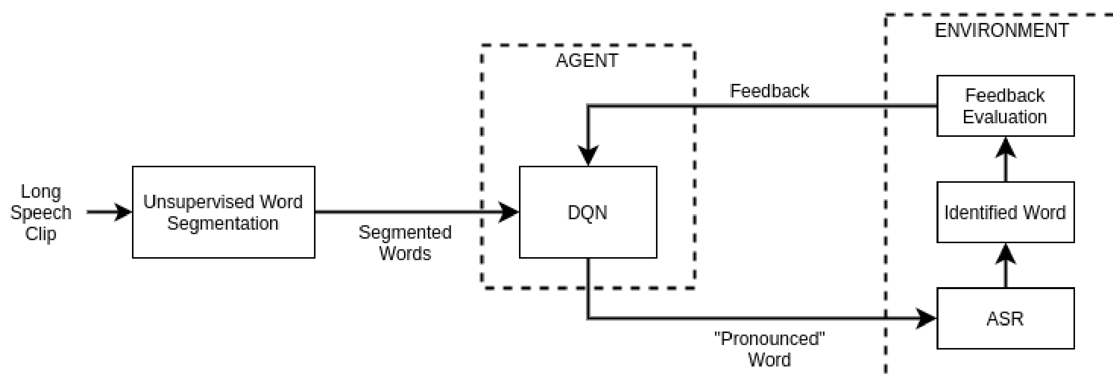


Figure 7: Diagram of spoken language acquisition using reinforcement learning.

The learning loop initializes with the segmented words as the DQN algorithm's action space. The agent makes an ”utterance” to the environment with a segmented word chosen based on a DQN which is still in exploration phase, meaning that decisions are made randomly.

The environment responds to each utterance by providing feedback. The automatic speech recognition (ASR) in the environment is responsible for recognizing the agent's ”spoken” word. The identified word is forwarded to a feedback evaluation algorithm that determines how the agent's state will or will not change.

The agent then evaluates the reward obtained based on the received feedback and on its current state. The reward calibrates the DQN such that better decisions are made. Therefore, as the agent ”speaks” more, the DQN gets more refined, and the agent gets to decide more based on the policy instead of just doing it randomly. Each episode terminates once the agent has enumerated all the digits ascending from ”zero” to ”nine”.

5 Implementation

5.1 Speech Data

The speech samples used for testing the system are from the Google Speech Commands data set[1] (version 2). It is made up of one-second-long utterances of 35 English words spoken by thousands of different people. It also contains a collection of various background noises.

A combined sound file using speech samples from the data set is created. It is comprised of a total of 500 utterances. There are 50 speech samples for each of the following words - ”zero”, ”one”, ”two”, ”three”, ”four”, ”five”, ”six”, ”seven”, ”eight”, and ”nine”. In addition, 50 short segments of background noise from the data set are inserted as well. All of these audio segments are shuffled randomly and concatenated into one file.

The combined sound file is used as the input speech signal for the experiments in section 6.1 and section 6.3.

5.2 Architecture

The system is comprised of two main parts, namely - unsupervised word segmentation and language learning (see Figure 8). The word segmentation part is further subdivided into feature extraction, phone segmentation, and word segmentation. The detailed diagram of the system architecture is shown in Figure 18 under Appendix A.

5.2.1 Unsupervised Word Segmentation

Feature Extraction

The feature vectors of the input speech signal are extracted using a trained VQ-APC model. The reference used for model training is the code repository[2] for [5]. LibriSpeech dataset’s *train-clean-360*, which contains 360 hours of ”clean” speech, is used for training. While *dev-clean*, which is the development set containing ”clean”

[1] http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz
[2] https://github.com/iamyuanchung/VQ-APC

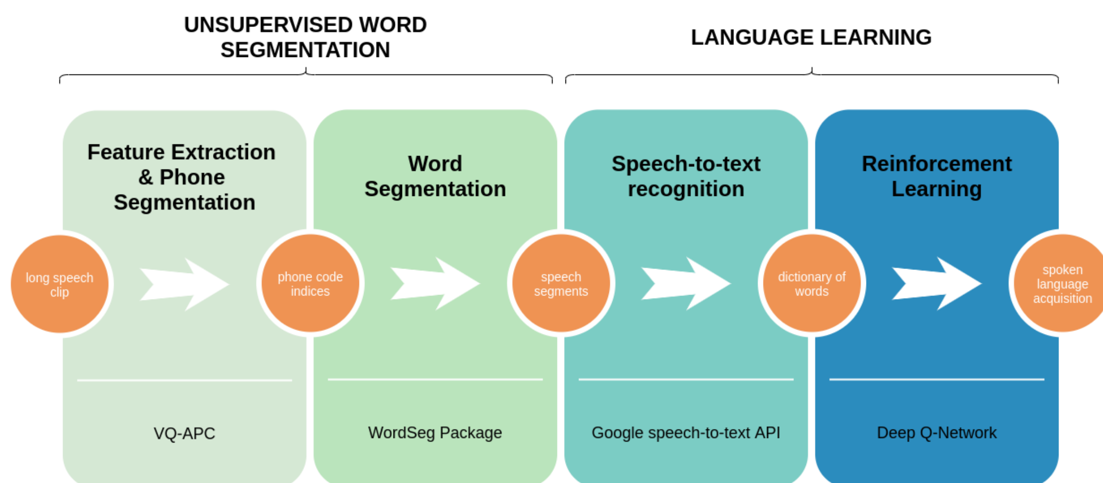


Figure 8: General overview of the system architecture.

speech, is used for validation. Training is run in a machine with two GPUs, and both of them are utilized.

Models with codebook sizes 128, 256, and 512 are trained for 2000 epochs. The parameters used for model initialization and training are listed in Table 2.

Parameters	Value
rnn_num_layers	3
rnn_hidden_size	512
rnn_dropout	0.1
rnn_residual	True
codebook_size	128, 256 or 512
code_dim	512
gumbel_temperature	0.5
apply_VQ	False, False, True
optimizer	adam
batch_size	32
learning_rate	0.0001
epochs	2000
n_future	5

Table 2: Parameters for the VQ-APC model initialization and training.

The model is set to have three RNN layers. The hidden layer size is 512. In comparison, the input layer size is 80, which is equivalent to the input feature dimension. Of the three network layers, the VQ layer is appended after the third one as indicated

by $(False, False, True)$ under the $apply_VQ$ parameter. It is found that inserting the VQ layer after the third RNN layer gave the most improvement over the regular APC in terms of phone error rate [5]. It is also worth noting that the model is set to predict 5 frames into the future.

While the codebook size is varied for each model trained, the vector dimension is fixed at 512. Both the codebook size and vector dimension control the amount of information that the VQ layer lets through [5]. The effect of varying codebook sizes on the downstream tasks of the system is investigated in the succeeding chapter.

Once the model is trained, it is used for feature extraction. First, the input speech signal is divided into portions with a maximum duration of 10 seconds. It is necessary to do this preprocessing due to the input duration limit found when running the trained models for feature extraction.

Next, the 80-dimension log Mel spectrogram of each portion is generated. The spectrogram is normalized to zero mean and unit variance per portion processed. The *fbank* function of the *torchaudio.compliance.kaldi* module[3] is utilized to create the log Mel spectrograms. The frame shift is set to 10 milliseconds, and the window type is set to ”hamming”. The module makes it possible to perform Kaldi[4] operations with *torchaudio*. The function used matches the output of Kaldi's *compute-fbank-feats*. It is important to do this since the model is trained on LibriSpeech dataset's log Mel spectrograms that were extracted using Kaldi scripts[5]. It is found that going with the same method results to better feature extraction performance than doing it otherwise.

Finally, the pre-trained model is loaded and set to evaluation mode. The learned codebook of the model is obtained by taking the weights of the VQ layer. Then, the 80-dimension log Mel spectrograms are used as input to the trained model. The resulting RNN hidden representation of the last layer during the forward pass is taken and considered as the feature vectors of the speech signal. These feature vectors and the model's corresponding codebook are used in the subsequent phone segmentation algorithm.

The execution done by Kamper et al. in [11] is used as a reference for implementing the phone and word segmentation.

[3] https://pytorch.org/audio/stable/compliance.kaldi.html
[4] https://github.com/kaldi-asr/kaldi
[5] https://github.com/iamyuanchung/Autoregressive-Predictive-Coding

Phone Segmentation

Phone segmentation is implemented by following the VQ segmentation algorithm presented in section 2.4. The parameters set for the phone segmentation are shown in Table 3.

Parameter	Value
minimum number of frames	0
maximum number of frames	100
duration penalty weight	36

Table 3: Parameters for the phone segmentation algorithm.

The maximum number of frames is set to 100, limiting the number of continuous frames contained in one phone segment. Each frame is a feature vector representing 10 milliseconds of the speech utterance. On the other hand, the duration penalty weight value dictates the significance of having longer segments. If the value is high, then the resulting phone segments become longer. It should be noted that the values for both of these parameters are chosen as seen fit based on the development data. Experimentation is done on a small sample of the input speech signal in order to decide on these parameter values.

The representative code for each phone segment is assigned based on which code from the codebook generates the lowest summed distance with respect to the feature vectors in the given phone segment.

Word Segmentation

The code indices assigned for the phone segments are concatenated into a prepared text which serves as input for the word segmentation algorithm. The WordSeg package [6] is used for word segmentation. The chosen algorithms from the package are TP and AG. They are run independently to perform various experiments in section 6. The results from both algorithms are compared.

Parameter	Value
threshold	relative
dependency	FTP

Table 4: Parameters for WordSeg TP segment function.

The parameters used when calling the *segment* function of TP are listed in Table 4. The test units are created from the prepared text input. Since there is no train text,

the test units are used as train units. The chosen dependency and threshold values are both default settings of the algorithm. By choosing *relative* threshold, the function takes four units at a time and checks for the TP of the bigrams. If a relative dip is found, then the midpoint of the bigram with the lower TP is taken as a word boundary. The process is done continuously until all the units are inspected. At the end of running the algorithm, the *segment* function returns a list of code indices grouped as words. [6].

Parameter	Value
nruns	8
njobs	3
args	”-n 2000”

Table 5: Parameters for WordSeg AG segment function

On the other hand, the parameters used when calling the *segment* function of the AG algorithm are listed in Table 5. The test text is created from the prepared text input. Since there is no train text, then the test text is used as the train text. The parameter *nruns* uses the default value. It indicates the number of times the AG algorithm is executed. While *njobs* signifies the number of subprocesses to run in parallel. The last parameter is for additional arguments. In this case, ”-n” defines the number of iterations per run and is set to 2000 by default. At the end of executing the algorithm, the *segment* function returns a list of code indices grouped as words [6].

After running a WordSeg algorithm, the list of phone code indices grouped into word candidates is used to find the word boundaries. Then, the list of boundaries is used for splitting the sound file. The resulting speech segments are then fed to the next part of the system.

5.2.2 Language Learning

Speech-to-text recognition

SpeechRecognition[6] python package is used to execute the ASR. It is a wrapper that supports several engines and application programming interface (API) and comes with a default API key for the Google Speech-to-Text API[7], which is used for this project.

The speech segments are fed into the ASR and may result in valid or non-valid recognized words. A word is considered valid if it is relevant to the task defined in

[6]https://pypi.org/project/SpeechRecognition/
[7]https://cloud.google.com/speech-to-text

section 4.1, which means any of the digits from ”zero” to ”nine”.

A dictionary records the total number of segments and each valid word with the corresponding quantity of recognized words. It is used as input to the reinforcement learning part, which essentially grounds the discovered words. Speech-to-text recognition only transforms speech signals into their symbolic equivalent and does not attach any meaning to them.

DQN

The implementation code[8] from Gao et al. [2] is used as reference for implementing the DQN algorithm.

The input to the DQN is the dictionary containing the total number of segments and each valid word with their corresponding quantity. The total number of segments signifies the action space size or the number of actions available to the agent. On the other hand, the quantity of each valid word is used as the number of actions that represent ”speaking” that word.

The agent and environment class definitions are defined based on the task described in section 4.1. The agent is initialized with an empty list signifying that it has not ”spoken” any digits. The environment is initialized with the input dictionary to the DQN. The act of speaking is simulated by the agent performing an action in the DQN. Every time the agent ”speaks” or performs an action, the environment responds by acknowledging valid words or actions and ignoring any non-valid ones. If the agent ”speaks” the correct digit, the environment responds by acknowledging to the agent that it has indeed ”spoken” the digit. If it ”speaks” incorrectly, the environment does not give any feedback.

The reward $r(t)$ for each time the agent performs an action is calculated as:

$$r(t) = SL(t) - SL(t-1), \tag{5.1}$$

where $SL(t)$ and $SL(t-1)$ stand for the satisfaction level of the agent at its current and previous states, respectively. The agent's state refers to how much of the list it has filled up. The agent's satisfaction level is given by the negative of the Levenshtein[9] distance between the current list and the target list. The target list is the digits ”zero” to ”nine” in ascending order. The Levenshtein distance is simply a measure of the

[8] https://github.com/tttslab/spolacq
[9] https://folk.idi.ntnu.no/mlh/hetland_org/coding/python/levenshtein.py

difference between the current digit sequence and the target digit sequence.

Additionally, when the agent does not gain any reward for the action performed, then the reward value is set to go down further at $r(t) = -10$. It serves as a punishment to discourage the agent from performing actions that do not merit any reward.

The agent performs a set of actions from the initial state until the target list is reached. This set of actions comprise one episode. The DQN learning loop is designed such that the agent performs 50 episodes over 100 random seeds.

Hyperparameter	Value
batch_size	128
gamma	0.5
eps_start	0.9
eps_end	0.05
eps_decay	200
target_update	10

Table 6: Hyperparameters for the deep Q-network.

The hyperparameters set for the DQN are shown in Table 6. The hyperparameters *batch_size* and *gamma* are used for optimizing the model. The *batch_size* value refers to the size of the sample taken from the replay memory, which stores past experiences from different episodes. The sample taken from this buffer is used as training data for the SGD update. On the other hand, the *gamma* value pertains to the discount factor used in calculating the expected Q-values. The value is chosen such that the future reward does not outweigh the current step's reward. It ensures that the model converges and does not deviate too much during the initial stages when the Q-values are still random.

The hyperparameters *eps_start*, *eps_end*, and *eps_decay* are used to implement the epsilon-greedy method, which solves the exploration versus exploitation dilemma. Whenever the agent needs to select an action, the epsilon threshold is calculated as:

$$eps_threshold = eps_end + (eps_start - eps_end)\left(e^{-\frac{steps_done}{eps_decay}}\right). \tag{5.2}$$

If the threshold is overcome, the action will be selected based on maximizing the expected reward; otherwise, the action is selected randomly.

As seen from Equation 5.2, the epsilon threshold changes based on the number of steps done or number of actions. The value of the term $e^{-steps_done/eps_decay}$ goes from 1

and decreases to minimal values as more steps are done, hence, the epsilon threshold value becomes infinitely small.

The value of the *eps_decay* hyperparameter affects the epsilon threshold calculation as well. When the *eps_decay* value is lower, the epsilon threshold value decreases faster as more steps are done. Conversely, when it is higher, the epsilon threshold value decreases slower as more steps are done.

Lastly, the *target_update* hyperparameter defines the episode interval at which the target network copies the weights from the policy network. Updating the target network once every *N* steps makes the training of the neural networks more stable.

6 Experiments and results

Three experiments are exploring different aspects of the system, namely - the relationship of the VQ-APC codebook size to the system performance, the relationship of word segmentation results to reinforcement learning results, and how the system fares in comparison to an existing one. This chapter introduces each experiment and describes the hypotheses to be tested. Afterward, the setup is elaborated, and the results are presented.

6.1 Modifying the code book size of the VQ-APC model

This experiment examines the word segmentation and reinforcement learning results versus different codebook sizes used for the VQ-APC model. The following hypothesis is tested:

H1 Increasing the codebook size positively affects the word segmentation results and the reinforcement learning results.

6.1.1 Setup

The input to the system is the combined sound file previously described in section 5.1 which contains utterances of the digits from ”zero” to ”nine”. The experiment is run using models with codebook sizes 128, 256, and 512. In line with this, the system implementation changes in the following ways:

- Feature extraction uses a VQ-APC model which will change depending on the chosen code book.
- Phone segmentation uses the chosen code book for segmenting the feature vector sequence and assigning the representative code to each segment.

Furthermore, for the word segmentation algorithm, the experiment uses WordSeg AG and TP independently. When using WordSeg AG, the whole system runs five times. While for WordSeg TP, it runs once. The reason for this is AG uses a different random seed for each run which results in different parses. For each of these runs, the DQN

learning loop executes 50 episodes over 100 random seeds. Lastly, the DQN hyperparameters are unchanged.

6.1.2 Results

The word segmentation result tables present the number of segments produced, the number of valid words recognized, and three other metrics. The recognition rate is the percentage of recognized valid words out of the actual quantity in the input. Over segmentation is the rate by which the system produces more words or segments than the actual amount in the input. Finally, the table provides the ratio of recognized valid words to the number of segments.

The reinforcement learning result figures show the number of actions executed for each episode of the learning loop. The quantity for each episode is the average number of actions over the 100 random seeds. It should be noted that for results relating to WordSeg AG, the figures show the average considering all the five runs.

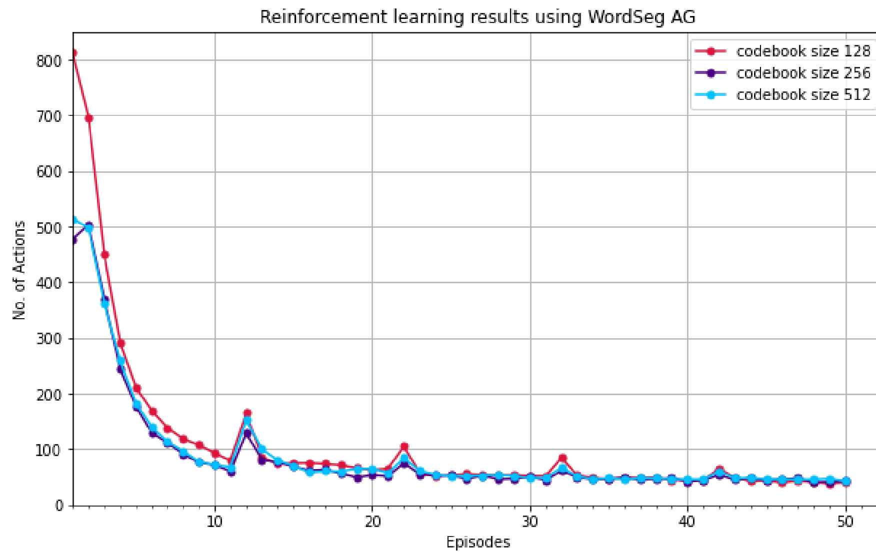
The results are presented by category of word segmentation algorithm used. First, the results using WordSeg AG are examined. Table 7, Table 8 and Table 9 present the word segmentation results for codebook sizes 128, 256 and 512, respectively. While, Table 10 summarizes the results of the three codebook sizes by presenting the average from all five runs. For a detailed breakdown on the quantity of recognized words, appendix B can be referred to.

Results	Run 1	Run 2	Run 3	Run 4	Run 5
Number of segments	251	243	249	249	239
Recognized valid words	77	65	76	72	66
Recognition rate	15.40%	13.00%	15.20%	14.40%	13.20%
Over segmentation	-49.80%	-51.40%	-50.20%	-50.20%	-52.20%
Valid words / segments	30.68%	26.75%	30.52%	28.92%	27.62%

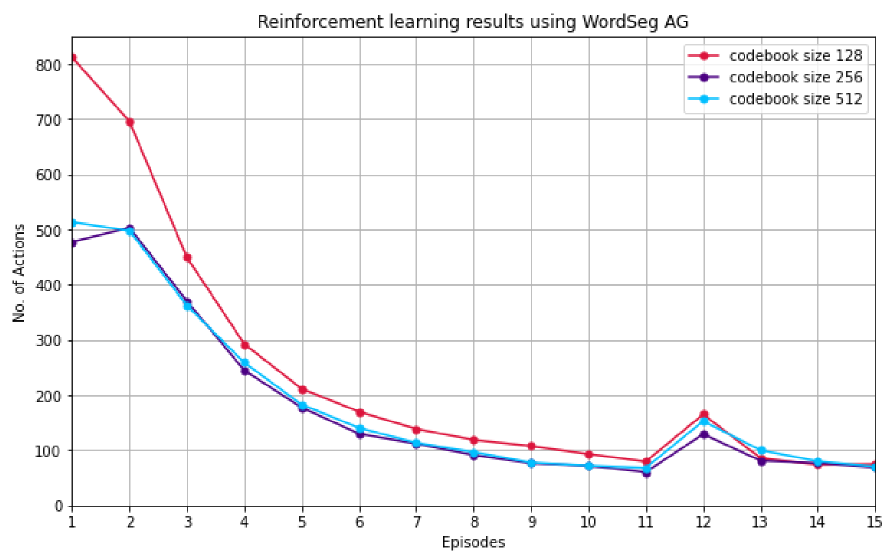
Table 7: Segmentation results using WordSeg AG and codebook size 128.

Results	Run 1	Run 2	Run 3	Run 4	Run 5
Number of segments	478	478	486	480	483
Recognized valid words	157	159	165	160	163
Recognition rate	31.40%	31.80%	33.00%	32.00%	32.60%
Over segmentation	-4.40%	-4.40%	-2.80%	-4.00%	-3.40%
Valid words / segments	32.85%	33.26%	33.95%	33.33%	33.75%

Table 8: Segmentation results using WordSeg AG and codebook size 256.



(a) Result of 50 episodes.



(b) Result of first 15 episodes.

Figure 9: Reinforcement learning results using WordSeg AG for codebook sizes 128, 256, and 512.

Results	Run 1	Run 2	Run 3	Run 4	Run 5
Number of segments	453	455	453	457	454
Recognized valid words	152	150	152	145	153
Recognition rate	30.40%	30.00%	30.40%	29.00%	30.60%
Over segmentation	-9.40%	-9.00%	-9.40%	-8.60%	-9.20%
Valid words / segments	33.55%	32.97%	33.55%	31.73%	33.70%

Table 9: Segmentation results using WordSeg AG and codebook size 512.

Results	code size 128	code size 256	code size 512
Number of segments	247	481	455
Recognized valid words	72	161	151
Recognition rate	14.40%	**32.20%**	30.20%
Over segmentation	-50.60%	-3.80%	-9.00%
Valid words / segments	29.15%	**33.47%**	33.19%

Table 10: Average of segmentation results using WordSeg AG.

As seen from Table 10, using codebook size 256 generates the most number of segments and achieves the highest recognition rate among the three codebook sizes. This observation is surprising since it is assumed that size 512 would be the one generating the best segmentation results. Although, the results from both sizes 256 and 512 are very close. In addition, the system under segments the combined sound file, and the results appear to be consistent across all five runs for all the codebook sizes.

The plots in Figure 9 illustrate how the agent has performed during reinforcement learning. Compared to the results of codebook size 128, it shows the agent took 37% fewer actions in the first episode when codebook size 512 is used. While, when codebook size 256 is used, it shows the agent took 44% fewer actions in the first episode. Closer inspection of Figure 9b shows that during the initial episodes, results of sizes 256 and 512 appear to coincide and maintain a small margin from the results of size 128.

Results	code size 128	code size 256	code size 512
Number of segments	231	424	411
Recognized valid words	75	111	118
Recognition rate	15.00%	22.20%	**23.60%**
Over segmentation	-53.80%	-15.20%	-17.80%
Valid words / segments	**32.47%**	26.18%	28.71%

Table 11: Summary of segmentation results using WordSeg TP.

For the next section, Table 11 summarizes the word segmentation results when WordSeg TP is used. For a detailed breakdown on the quantity of recognized words, Table 23 in appendix B can be referred to.

It is apparent from Table 11 that using codebook size 512 achieves the highest recognition rate. While the most number of segments is obtained when using codebook size 256. Again, only a small difference is observed between the results of codebook sizes 256 and 512. With all three sizes, the system under segments the combined sound file.

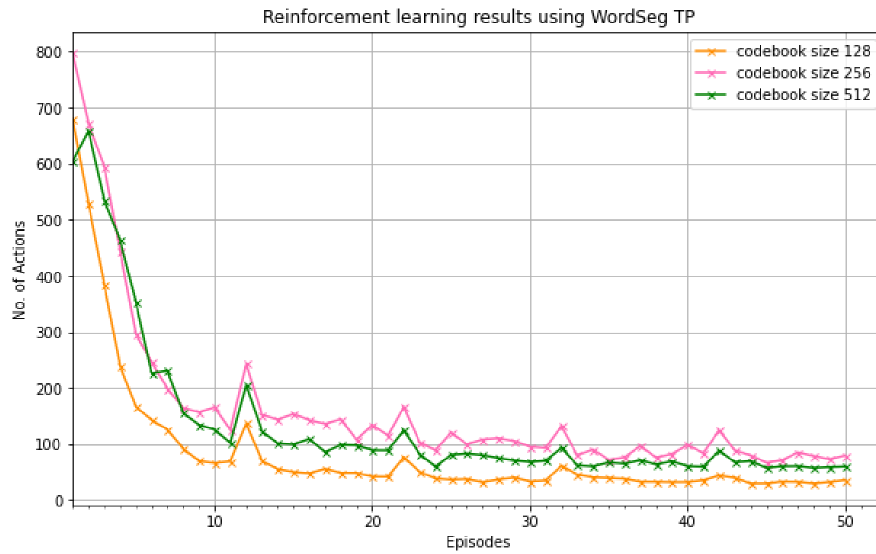
Comparing the data from Table 11 and Table 10, it shows that recognition rates are lower when using WordSeg TP compared to AG. However, for the case of codebook size 128, there is not much difference in the recognition rates when using either of the two word segmentation algorithms. Furthermore, all codebook sizes produced lower number of segments when using TP.

The plots in Figure 10 represents the agent's performance when using the word segmentation results under WordSeg TP. In general, it shows the agent took the least number of actions when codebook size 128 is used. Interestingly, Figure 10b shows in the first episode that the agent took around 12% more actions with codebook size 128 compared to size 512. On the other hand, it shows the agent took the most number of actions when codebook size 256 is used.

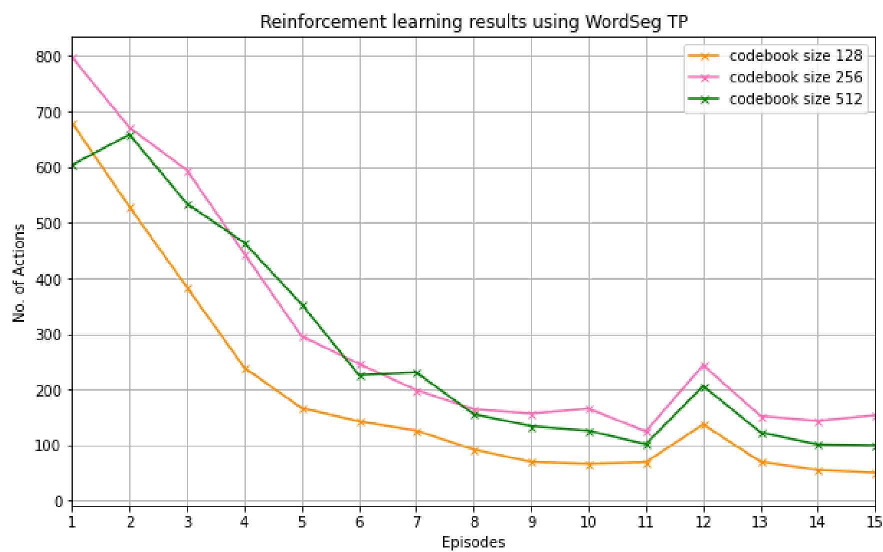
Finally, Figure 11 shows all the cases in one plot. Prominent peaks are occurring at episodes 12, 22, 32, and 42. It can be noted that the agent is initialized the same way each time an episode starts. The peaks may be explained by the interval at which the target network synchronizes with the policy network. The DQN hyperparameters in Table 6 show the target network getting updated every 10 episodes. The peaks seem to occur after these updates.

Figure 11 also shows that using codebook size 256 with WordSeg AG starts the learning loop with the lowest number of actions taken during the first episode. While in the later episodes, the plot shows that the lowest number of actions are taken when using codebook size 128 with WordSeg TP.

In summary, hypothesis **H1** is tested to be true to some degree only. Results show that increasing the codebook size has positive effects on the recognition rate of the word segmentation results. It is strictly observed when increasing the codebook size from 128 to 256. However, when going from codebook size 256 to 512, the difference between their recognition rates is not that significant. When using WordSeg TP, size



(a) Result of 50 episodes.

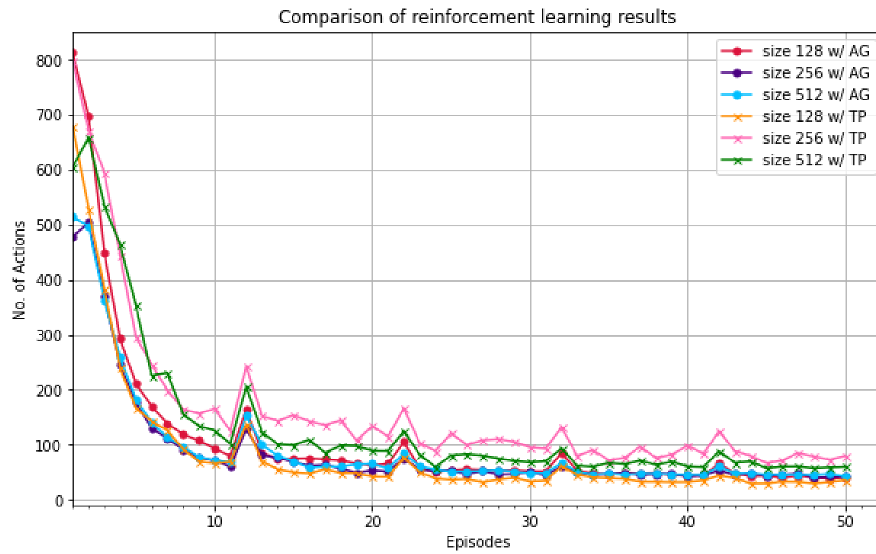


(b) Result of first 15 episodes.

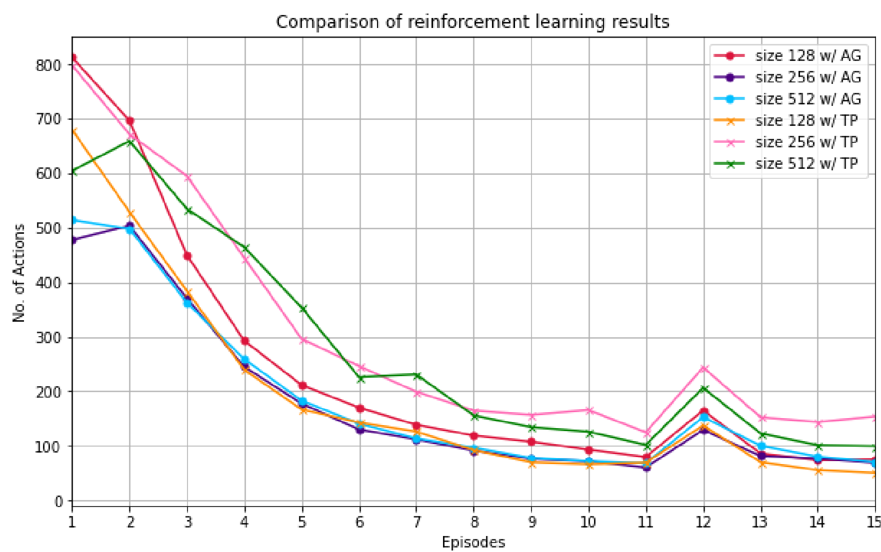
Figure 10: Reinforcement learning results using WordSeg TP for codebook sizes 128, 256, and 512.

512 achieves slightly better recognition rate. While the opposite is observed when using WordSeg AG.

Nonetheless, hypothesis **H1** cannot be justified for the reinforcement learning results. There are instances in the experiments wherein the lower codebook size case managed to achieve better reinforcement learning results. It is observed that there seems to be a correlation between the ratio of recognized valid words to the number of segments and the performance in the reinforcement learning part. Interestingly, cases with higher ratios develop to have generally lower number of actions. It is to be remarked that this relationship is only anecdotal and not to be considered as generally applicable to other DQN implementations.



(a) Result of 50 episodes.



(b) Result of first 15 episodes.

Figure 11: Comparison of all reinforcement learning results for codebook sizes 128, 256, and 512.

6.2 Simulated word segmentation results

This experiment simulates different word segmentation results and investigates the corresponding response of the reinforcement learning algorithm. The following hypotheses are tested:

H2 A high recognition rate in the word segmentation has a positive effect on the reinforcement learning results.
H3 Increasing the DQN action space size may result in the deterioration of the agent's performance during the learning loop.

6.2.1 Setup

Two cases are considered based on the hypotheses mentioned above. The first case tackles hypothesis **H2** which involves simulating word segmentation results with varying recognition rates. It is implemented by changing the number of recognized valid words while the total number of segments produced remains constant. Table 12 summarizes ten different settings considered for the simulation of the word segmentation results.

Property	1	2	3	4	5	6	7	8	9	10
Number of segments	500	500	500	500	500	500	500	500	500	500
Number of recognized valid words	50	100	150	200	250	300	350	400	450	500
Recognition rate	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
Valid words / segments	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%

Table 12: Settings for simulating word segmentation results with varying recognition rates.

The number of segments produced is fixed at 500 for each setting. At the same time, the total number of recognized valid words is incremented from 50 until 500. Each digit from "zero" to "nine" is assigned equal amounts of recognized words. Moreover, it is considered that there is a total of 500 valid words in the hypothetical speech signal. As a result, the recognition rate changes from 10% to 100%. Likewise, the ratios of recognized valid words to the number of segments have the same resulting values. A dictionary representing each scheme is created and used as input to the reinforcement

learning algorithm. The learning loop iterates 50 episodes over 100 random seeds, and the DQN hyperparameters are unchanged.

On the other hand, the second case tackles hypothesis **H3** which involves simulating word segmentation results with varying action space sizes. It is implemented by changing the number of segments while the number of recognized valid words remains constant. Table 13 summarizes nine different settings for simulating the word segmentation results. The number of segments denotes the action space size for the DQN.

Property	1	2	3	4	5	6	7	8	9
Number of segments (action space size)	200	300	400	500	600	700	800	900	1000
Number of recognized valid words	200	200	200	200	200	200	200	200	200
Recognition rate	40%	40%	40%	40%	40%	40%	40%	40%	40%
Valid words / segments	100%	67%	50%	40%	33%	29%	25%	22%	20%

Table 13: Settings for simulating word segmentation results with varying number of segments or action space sizes.

The number of recognized valid words is set to 200, while the number for segments produced is incremented from 200 to 1000. A total of 500 valid words is considered in the hypothetical speech signal. As such, the recognition rate is fixed at 40%. On the other hand, the ratio of recognized valid words to the number of segments decreases from 100% to 20%. A dictionary representing each action space size is created and used as input to the reinforcement learning algorithm. The learning loop iterates 50 episodes over 100 random seeds, and the DQN hyperparameters are unchanged.

6.2.2 Results

The reinforcement learning result figures show the number of actions executed for each episode of the learning loop. It should be noted that the quantity shown for each episode is the average number of actions over the 100 random seeds. The mean and standard deviation of the number of actions over the 50 episodes are calculated as well.

The reinforcement learning results for the first case are shown in Figure 12. As the recognition rate decreases, the agent takes more actions during the initial episodes of the reinforcement learning.

Property	11	12	13	14
Number of segments	500	500	500	500
Number of recognized valid words	60	70	80	90
Recognition rate	12%	14%	16%	18%
Valid words / segments	12%	14%	16%	18%

Table 14: Settings for simulating recognition rates 12% to 18%.

For recognition rates 40% and below, a more significant increase in the number of actions throughout the episodes is observed as the recognition rate decreases. As there is a big gap between the results of 10% and 20%, additional simulations are done in this range. Table 14 summarizes the settings used. Figure 13 shows that as you decrease the recognition rate from 20%, more variability in the number of actions is observed. Additionally, it shows that the model fails to converge when the rate is 12% and lower.

For recognition rates 50% to 100%, Figure 14 shows that the initial episodes follow the general observation of increasing number of actions as the rate decreases. Interestingly, the models eventually converge at approximately the same episode and within a small range for the number of actions.

Table 15 lists down the mean and standard deviation values calculated for the reinforcement learning results of the first case. In general, the mean value increases as the recognition rate decreases. However, it does not change linearly. When the rates decrease from 100% to 50%, mean values slowly increase. While mean values rapidly rise when rates are falling from 40% to 10%. This observation supports the convergence perceived in the plots for higher recognition rates (50% and above). The standard deviation values also show an increasing behavior for decreasing recognition rates. It supports the observed increase in variability of the number of actions as the recognition rate decreases.

Overall, hypothesis **H2** is valid for the conditions described in the experiment setup. Higher recognition rates do improve the reinforcement learning performance. Although, it is observed that the overall improvement becomes less significant at some point. From recognition rates 50% and above, substantial improvements are only apparent in the initial episodes of the learning loop. It is important to note that the underlying

Recognition rate	Mean	Standard deviation
10%	1521.04	405.67
12%	986.02	307.90
14%	514.16	239.30
16%	291.60	202.38
18%	268.18	199.05
20%	169.66	159.59
30%	88.04	99.91
40%	68.80	73.66
50%	56.76	57.70
60%	53.24	46.11
70%	49.42	39.06
80%	48.12	35.34
90%	46.78	30.00
100%	46.52	29.42

Table 15: Mean and standard deviation of reinforcement learning results from the simulated word segmentation with varying recognition rates.

condition for this observation is that the number of segments produced is equal to the total number of valid words in the hypothetical speech signal. Additionally, 20% is the lower limit found for the recognition rate such that the reinforcement learning results are acceptable.

Turning now to the second case, Figure 15 shows that increasing the action space size results in a general increase in the number of actions throughout all the episodes. It also indicates a notable increase in the variability of the number of actions as the action space size increases.

Property	10	11	12
Number of segments (action space size)	1100	1200	1300
Number of recognized valid words	200	200	200
Recognition rate	40%	40%	40%
Valid words / segments	18%	17%	15%

Table 16: Settings for simulating action space sizes 1100 to 1300.

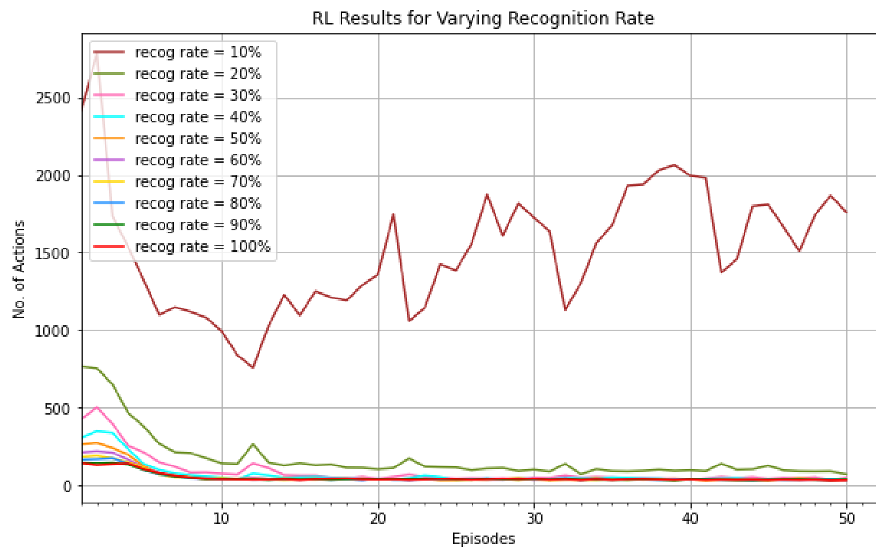
Additional simulations using more extensive action space sizes examine the limits for obtaining acceptable reinforcement learning results. Table 16 summarizes the settings used. Figure 16 shows that starting from 1100, not only is there more variability in the number of actions, but also elevated propensity for the model not to converge. This

observation becomes more evident as the action space size increases. On the other hand, Table 17 shows that both the mean and standard deviation values increase as the action space size increases, which supports the general trend observed in the plots.

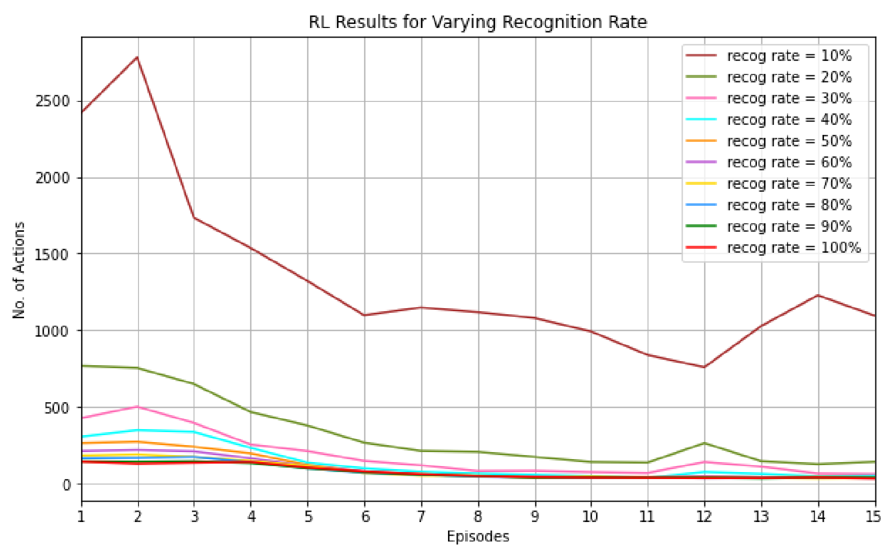
Action space size	Mean	Standard deviation
200	38.78	29.77
300	46.30	43.15
400	58.48	61.04
500	68.80	73.66
600	105.14	100.60
700	106.92	108.49
800	144.04	131.37
900	160.02	152.38
1000	242.60	159.49
1100	458.10	188.27
1200	604.84	199.49
1300	832.98	214.39

Table 17: Mean and standard deviation of reinforcement learning results from the simulated word segmentation with different action space sizes.

Hence, hypothesis **H3** is true for the condition stated in the experiment set up - a recognition rate of 40% for a hypothetical speech signal with 500 valid words. Results show that increasing the action space size causes negative effects on the reinforcement learning results in varying degrees. For action space sizes until 1000, the number of actions throughout the episodes increases as the action space size increases. While, for sizes larger than 1000, significant deterioration in the agent's performance is evident through the overall increase in the number of actions and the model's failure to reach and maintain convergence.

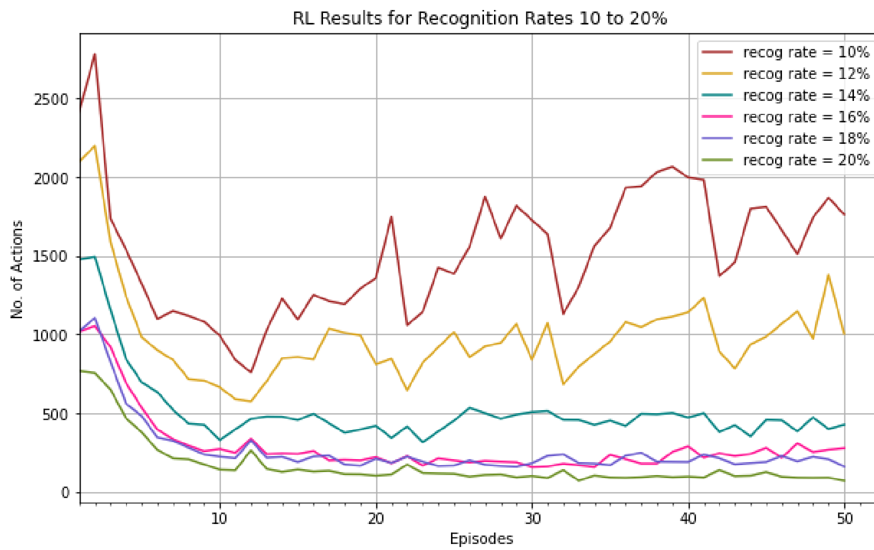


(a) Result of 50 episodes.

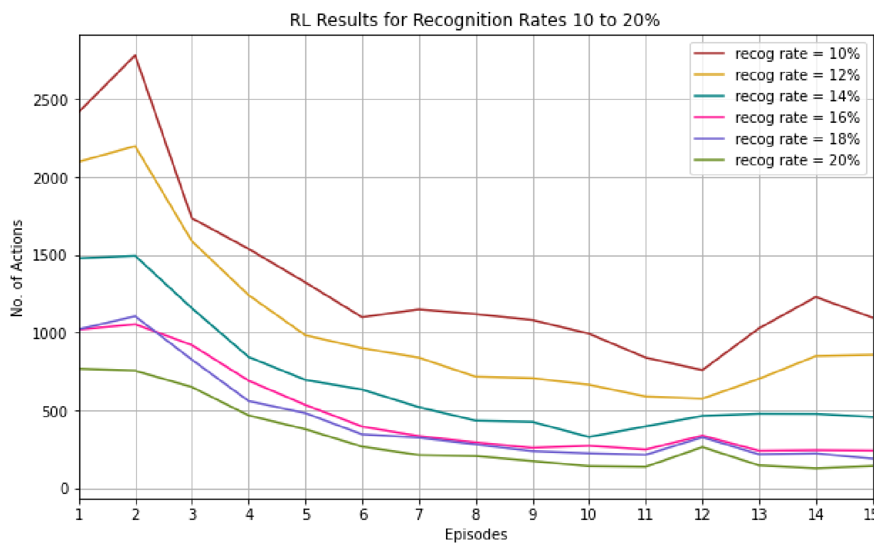


(b) Result of first 15 episodes.

Figure 12: Reinforcement learning results for simulated word segmentation with recognition rate from 10% to 100%.

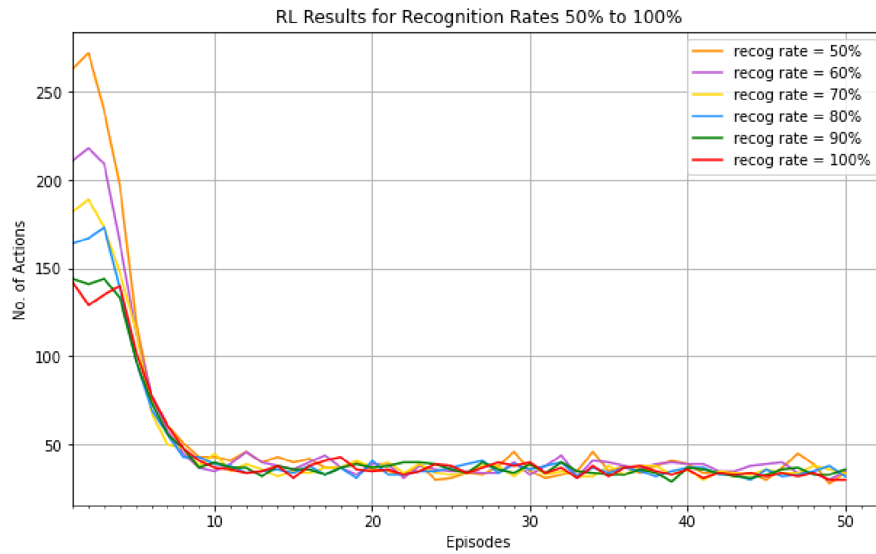


(a) Result of 50 episodes.

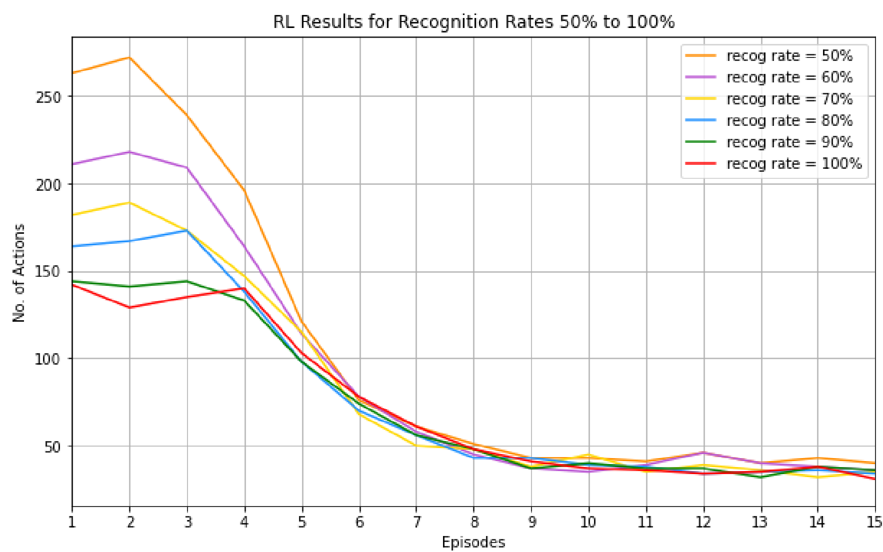


(b) Result of first 15 episodes.

Figure 13: Reinforcement learning results for simulated word segmentation with recognition rate from 10% to 20%.

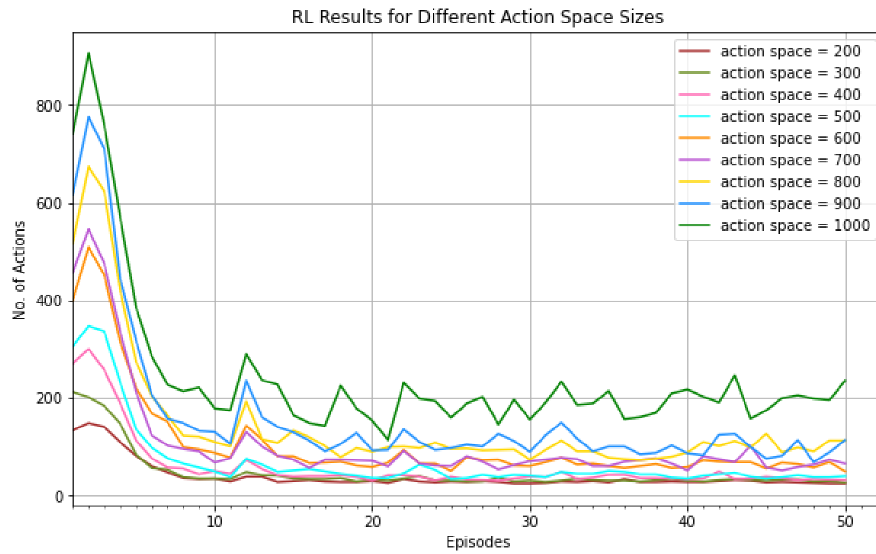


(a) Result of 50 episodes.

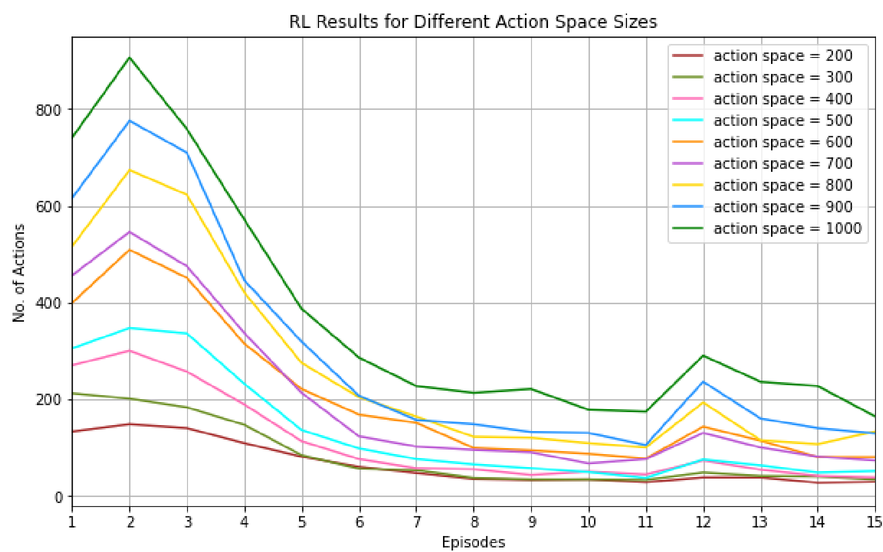


(b) Result of first 15 episodes.

Figure 14: Reinforcement learning results for simulated word segmentation with recognition rate from 50% to 100%.



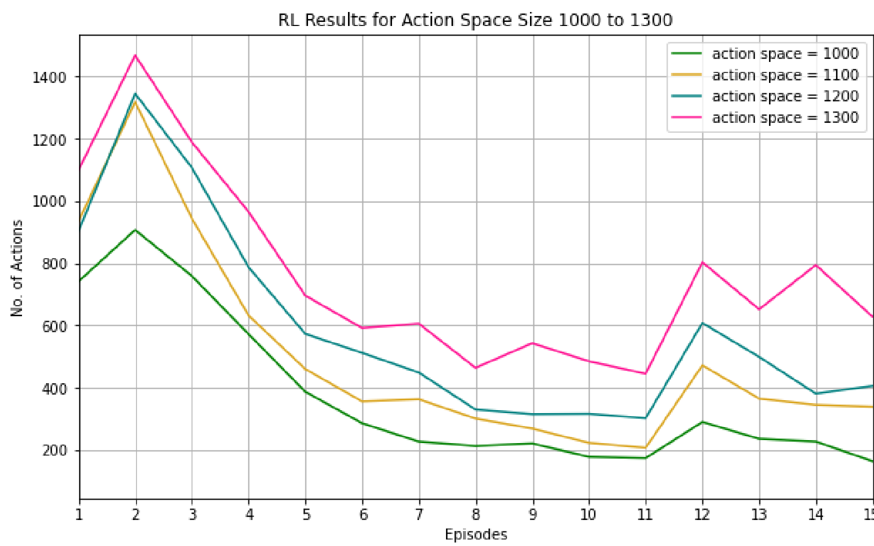
(a) Result of 50 episodes.



(b) Result of first 15 episodes.

Figure 15: Reinforcement learning results for simulated word segmentation with different action space sizes.

(a) Result of 50 episodes.



(b) Result of first 15 episodes.

Figure 16: Reinforcement learning results for simulated word segmentation with action space sizes from 1000 to 1300.

6.3 VQ-APC versus ES k-means for word segmentation

This experiment compares the performance of the system to an existing one that uses ES k-means for the word segmentation algorithm. It examines the word segmentation and reinforcement learning results from these two systems.

6.3.1 Setup

The input to both systems is the same combined sound file previously described in section 5.1 which contains utterances of numbers zero to nine.

Results using codebook size 512 with WordSeg AG from section 6.1 is used as the representative of the system. It is referred to as VQ segmentation in the results.

The existing system from Gao et al. [2] is used for generating the results for the ES k-means algorithm case. It is run five times, with the learning loop iterating 50 episodes over 100 random seeds. The DQN hyperparameters are unchanged.

6.3.2 Results

The word segmentation and reinforcement learning results are presented in the same way as in section 6.1. For a detailed breakdown on the quantity of recognized words for ES k-means, Table 24 in appendix B can be referred to.

Table 18 shows that the ES k-means algorithm segments the combined sound file to around 50% more words. To compare it with VQ segmentation, Table 19 summarizes the results of both algorithms by presenting the average from all 5 runs. It is inspected that ES k-means achieved much higher recognition rate compared to VQ segmentation.

Results	Run 1	Run 2	Run 3	Run 4	Run 5
Number of segments	736	712	741	746	736
Recognized valid words	285	290	279	274	278
Recognition rate	57.00%	58.00%	55.80%	54.80%	55.60%
Over segmentation	47.20%	42.40%	48.20%	49.20%	47.20%
Valid words / Segments	38.72%	40.73%	37.65%	36.73%	37.77%

Table 18: Segmentation results using ES k-means for word segmentation.

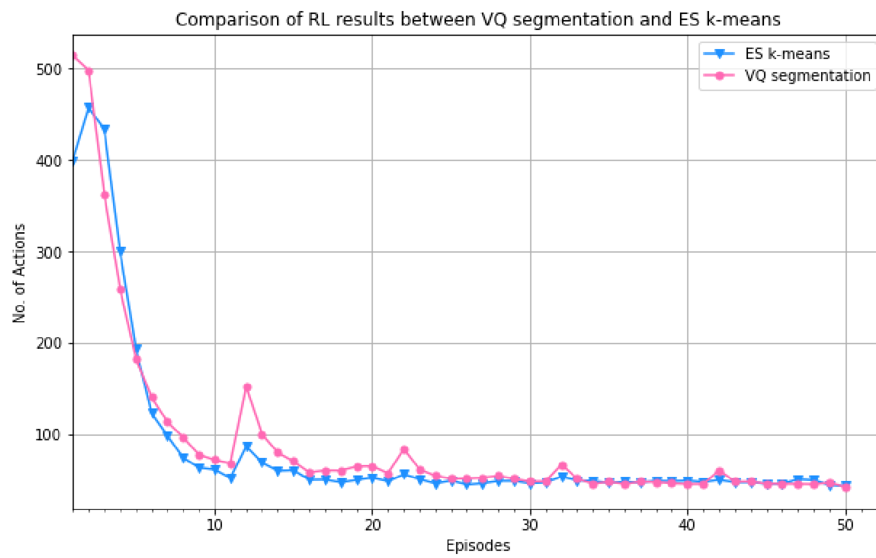
As exemplified in section 6.1, there is an observed relationship between the ratio of recognized valid words to number of segments and the reinforcement learning results. Word segmentation results with higher ratios tend to be followed by better performance in the reinforcement learning part. Seeing in Table 19 that ES k-means

Results	VQ-seg (code size 512 w/ WordSeg AG)	ES k-means
Number of segments	455	735
Recognized valid words	151	282
Recognition rate	30.20%	**56.40%**
Over segmentation	-9.00%	47.00%
Valid words / segments	33.19%	**38.37%**

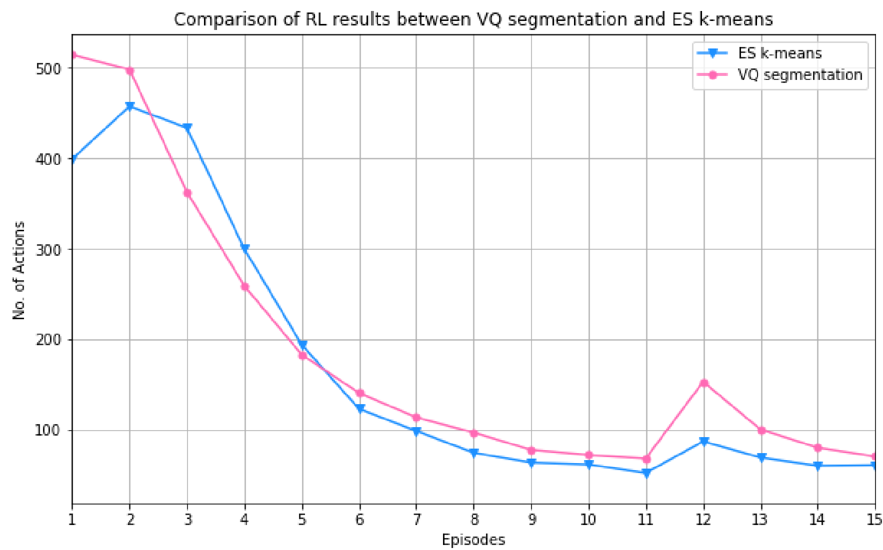
Table 19: Average segmentation results of VQ segmentation and ES k-means.

attained a higher ratio, it is anticipated to have better reinforcement learning results. Figure 17 reveals that with ES k-means the agent took 22% less actions in the first episode compared to the VQ segmentation case. Surprisingly, Figure 17b shows that in episodes 3 to 5, VQ segmentation achieved lesser number of actions. In the subsequent episodes, the resulting number of actions for ES k-means are generally lower, but not by a huge margin.

In summary, the system using ES k-means algorithm achieves better results in both word segmentation and reinforcement learning. Thus, there is room for improvement for the implementation of VQ segmentation.



(a) Result of 50 episodes.



(b) Result of first 15 episodes.

Figure 17: Reinforcement learning results for VQ segmentation and ES k-means.

7 Discussion and Conclusion

7.1 Discussion

The experiment testing hypothesis **H1** demonstrated that bigger codebook sizes produce higher recognition rates for the word segmentation results. It was significantly evident when comparing codebook size 128 with either 256 or 512. However, the resulting recognition rates were very close between codebook sizes 256 and 512.

It is anticipated that bigger codebook sizes would result in better accuracy. The goal of VQ segmentation is to minimize the error function in Equation 2.15 which compares the squared Euclidean distance between the feature vectors and the code vector assigned to each phone segment. If the codebook is large, each phone segment would be assigned its corresponding code more accurately such that the error function is minimized. Better distinction between phone segments improves word discovery.

Referencing back to Equation 2.15, the duration weight value assigned for the penalty term also affects the results of the segmentation. As previously discussed in subsection 5.2.1, the duration weight value of 36 is chosen based on experimenting with the development data. The results show that it was a good choice since a reasonable amount of valid words were produced.

A possible explanation for the closeness of results between codebook sizes 256 and 512 is the small vocabulary size of the input speech signal. The combined sound file is comprised of only ten different words, which means that there will not be a large set of phonemes to identify. The model with a codebook size of 256 is adequate for the combined sound file based on the results.

Moreover, the experiment on hypothesis **H1** showed that bigger codebook sizes did not necessarily conclude with better reinforcement learning results. Instead, anecdotal evidence suggested that higher ratios of recognized words to the number of segments indicated improved reinforcement learning performance. It was shown that the choice for the word segmentation algorithm also mattered. As presented in Table 10, higher ratios were found with the results of codebook sizes 256 and 512 when WordSeg AG is used. Conversely, codebook size 128 had the higher ratio when WordSeg TP is used,

as shown in Table 11. The smaller codebook size took advantage of the TP algorithm in that less variety in the phone sequences was able to produce fewer segments that contained a viable amount of valid words.

In reality, the recognition rate is the frequency of recognizing the spoken words intended to be learned. The total number of segments or action space size can be thought of as our world where one constantly hears speech and noise. Only a portion of this world would have the target spoken words, and the rest would be noise or irrelevant spoken words that can cause confusion and hinder language learning. In applying spoken language acquisition to machines, it would make sense to consider optimization based on both the recognition rate and the action space size.

The simulated word segmentation experiment examined how the word segmentation results affect the reinforcement learning performance. Hypothesis **H2** dealt with the recognition rate. It was proven that having a higher recognition rate does improve the reinforcement learning performance as long as the total number of segments produced remains the same. However, there was a limit found to the improvement that can be attained. At some point in increasing the recognition rate, significant improvements only manifested in the initial episodes of the DQN learning loop.

On the other hand, hypothesis **H3** dealt with the number of segments produced or equivalently the action space size for the DQN. The experiment tested that increasing the action space size while the recognition rate is held constant causes more unsatisfactory reinforcement learning results. Up to a certain extent of increasing the size, the deterioration was limited to an increased number of actions. However, after some threshold, increasing the size also caused the model convergence to fail.

It should be recalled that DQN is implemented with the epsilon-greedy method to deal with the exploration versus exploitation dilemma. During the beginning of the learning loop, the agent is in exploration mode and acts randomly. As the learning loop progresses, the agent goes into exploitation mode and acts based on the policy network. When the action space is much larger than the number of valid actions, the agent has a higher chance of choosing more non-valid actions and developing the wrong policy network. Eventually, the model may not converge.

Conversely, when the action space size is small, it is easier for the agent to choose the correct actions and develop the most efficient policy network. It should also be noted that the replay buffer stores the past experiences of the agent and uses a sample from this buffer to implement the SGD update of the policy network weights. If the buffer is mainly filled up with past experiences based on non-valid actions, then there

is a higher chance that the policy network will not be appropriately developed.

This experiment demonstrated how the recognition rate and action space size independently influence the reinforcement learning results. Improving the recognition rate to a level that facilitates successful reinforcement learning seemed to be sufficient enough. In addition, the action space size limit needed to be watched out to ensure model convergence. As previously discussed, looking into the ratio of recognized valid words to action space size can offer additional insights on the possible constraints of grounding spoken language in machines using reinforcement learning.

The last experiment compared VQ segmentation and ES k-means algorithm. It demonstrated that there is still room for improvement with the current VQ segmentation setup. The system using ES k-means achieved better results in both word segmentation and reinforcement learning. However, there was not a huge gap between the number of actions of the two cases during the later episodes of the learning loop.

Unsupervised word segmentation in this thesis relies on the VQ-APC model to capture the acoustic differences in the speech signal. As such, the extent to which the model is trained will affect the results. The thesis uses models that were trained for 2000 epochs. It is possible that training them further can help improve the word segmentation results. It can be recalled that VQ segmentation breaks down the combined sound file into phone segments. Then, an algorithm from the WordSeg package is used to detect words from the sequence of phones. Therefore, word discovery is highly dependent on the quality of the phone segments.

On the other hand, ES k-means starts with random word boundaries and then goes back and forth, mutually optimizing the segmentation and cluster assignments. The goal is to group acoustically similar speech segments. One remark on this method is that some knowledge on the speech signal is required to declare a sufficient number of K hypothesized words that the algorithm uses as a basis for the clustering.

Comparing the two segmentation methods, it is apparent that there is more flexibility in the way ES k-means assigns the word boundaries. However, even though VQ segmentation may be more rigid in its approach, it has the advantage of not requiring any hypothesis on the number of words in the speech signal. A theory that can explain why ES k-means worked better is that it was able to work around the rather long silent gaps in between the utterances in the combined sound file. The VQ-APC model may not necessarily have a good representation of these silences in its codebook since it trained on a massive amount of continuous speech.

7.2 Conclusion

One of the significant contributions of this work is utilizing the VQ-APC model for unsupervised word segmentation. By training the model using the LibriSpeech dataset for 2000 epochs, it effectively extracted the feature vectors of the speech signal and generated a codebook capable of producing acceptable results with VQ segmentation. Based on the comparison done with an existing system using ES k-means algorithm, results revealed that there is still room for improvement.

On the other hand, the DQN reinforcement learning algorithm was successfully utilized for the language learning task defined in this thesis. It proved to be an effective tool in simulating grounded language acquisition. The agent's verbal behavior was established through positive rewards gained by accomplishing the task of "speaking" the digits in ascending order.

Various experiments were performed that tested both the word segmentation and reinforcement learning performance of the system. Through them, it was discovered that the best results for word segmentation were achieved when the VQ-APC model is used with WordSeg AG. However, having the best word segmentation results was not enough to ensure the best reinforcement learning results. It was found that looking into other parameters, such as the ratio of recognized valid words to the total number of segments, gave a better idea of how the language learning performance would be. It should be noted that these were based on anecdotal evidence particular to the task and DQN setup in this thesis. However, it can shed some additional insight for other reinforcement learning implementations of similar nature.

In summary, the thesis achieved spoken language acquisition in machines in line with Skinner's theory by performing unsupervised word segmentation on a long speech clip and employing reinforcement learning to ground the discovered spoken words. Moreover, it was able to utilize the novel VQ-APC model for unsupervised word segmentation and managed to discover factors that can influence the reinforcement learning performance.

7.3 Future work

For the word segmentation part, the VQ-APC model can be trained for more epochs and see if significant improvements in the results can be achieved. Moreover, another speech corpus containing labels for phonetic boundaries can be used to validate the system's performance at the phone segmentation level.

It would also be interesting for the reinforcement learning part to see an implementation that deals with a more extensive vocabulary and more complicated tasks for the agent. Additionally, multi-sensory input for the system can also be considered. Currently, the system is only working with speech signals and establishing their meanings through tasks performed in a virtual environment. Visual input can also be included, leading to a more well-rounded way to learn the language.

Bibliography

[1] Brown, H. D. 2006. *Principles of Language Learning and Teaching*. Pearson Education, 5th edition.

[2] Gao, S., Hou, W., Tanaka, T., & Shinozaki, T. 2020. Spoken language acquisition based on reinforcement learning and word unit segmentation. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 6149–6153. doi:10.1109/ICASSP40776.2020.9053326.

[3] Kamper, H., Livescu, K., & Goldwater, S. 2017. An embedded segmental k-means model for unsupervised segmentation and clustering of speech. In *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 719–726. doi:10.1109/ASRU.2017.8269008.

[4] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., & Hassabis, D. February 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533. doi:10.1038/nature14236.

[5] Chung, Y.-A., Tang, H., & Glass, J. 2020. Vector-Quantized Autoregressive Predictive Coding. In *Proc. Interspeech 2020*, 3760–3764. URL: http://dx.doi.org/10.21437/Interspeech.2020-1228, doi:10.21437/Interspeech.2020-1228.

[6] Bernard, M. October 2018. bootphon/wordseg: wordseg-0.7.1. URL: https://doi.org/10.5281/zenodo.1471532, doi:10.5281/zenodo.1471532.

[7] Nandy, A. & Biswas, M. 2018. *Reinforcement learning with Open AI, TensorFlow and Keras Using Python*. Number 1. doi:10.1007/978-1-4842-3285-9.

[8] Lapan, M. 2018. *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd.

[9] Chung, Y.-A., Hsu, W.-N., Tang, H., & Glass, J. 2019. An Unsupervised Autoregressive Model for Speech Representation Learning. In *Proc. Interspeech*

2019, 146–150. URL: http://dx.doi.org/10.21437/Interspeech.2019-1473, doi:10.21437/Interspeech.2019-1473.

[10] Jang, E., Gu, S., & Poole, B. 2017. Categorical reparametrization with gumble-softmax. In *International Conference on Learning Representations (ICLR 2017)*. OpenReview. net.

[11] Kamper, H. & van Niekerk, B. 2020. Towards unsupervised phone and word segmentation using self-supervised vector-quantized neural networks. *CoRR*, abs/2012.07551. URL: https://arxiv.org/abs/2012.07551, arXiv:2012.07551.

[12] Bernard, M., Thiolliere, R., Saksida, A., Loukatou, G. R., Larsen, E., Johnson, M., Fibla, L., Dupoux, E., Daland, R., Cao, X. N., et al. 2020. Wordseg: Standardizing unsupervised word form segmentation from text. *Behavior research methods*, 52(1), 264–278. doi:10.3758/s13428-019-01223-3.

[13] Levin, K., Henry, K., Jansen, A., & Livescu, K. 2013. Fixed-dimensional acoustic embeddings of variable-length segments in low-resource settings. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, 410–415. doi:10.1109/ASRU.2013.6707765.

[14] Levin, K., Jansen, A., & Van Durme, B. 2015. Segmental acoustic indexing for zero resource keyword search. In *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 5828–5832. doi:10.1109/ICASSP.2015.7179089.

[15] Kamper, H., Wang, W., & Livescu, K. 2016. Deep convolutional acoustic word embeddings using word-pair side information. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 4950–4954. doi:10.1109/ICASSP.2016.7472619.

[16] Matuszek, C. 2018. Grounded language learning: Where robotics and nlp meet (invited talk). *Proceedings of the International Joint Conference on Artificial Intelligence*. URL: https://par.nsf.gov/biblio/10066404.

[17] Sinha, A., Akilesh, B., Sarkar, M., & Krishnamurthy, B. 2019. Attention based natural language grounding by navigating virtual environment. In *2019 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 236–244. doi:10.1109/WACV.2019.00031.

[18] Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., Wainwright, M., Apps, C., Hassabis, D., & Blunsom, P. 2017. Grounded language learning in a simulated 3d world. *CoRR*, abs/1706.06551. URL: http://arxiv.org/abs/1706.06551, arXiv:1706.06551.

[19] Yu, H., Zhang, H., & Xu, W. 2018. Interactive grounded language acquisition and generalization in a 2d world. *CoRR*, abs/1802.01433. URL: http://arxiv.org/abs/1802.01433, arXiv:1802.01433.

[20] Roy, D. 2003. Grounded spoken language acquisition: experiments in word learning. *IEEE Transactions on Multimedia*, 5(2), 197–209. doi:10.1109/TMM.2003.811618.

[21] Yu, C. & Ballard, D. H. 2004. On the integration of grounding language and learning objects. In *AAAI*, volume 4, 488–493.

[22] Chauhan, A. & Lopes, L. S. 2011. Using spoken words to guide open-ended category formation. *Cognitive processing*, 12(4), 341–354.

[23] Boves, L., ten Bosch, L., & Moore, R. 2007. Acorns - towards computational modeling of communication and recognition skills. In *6th IEEE International Conference on Cognitive Informatics*, 349–356. doi:10.1109/COGINF.2007.4341909.

[24] Zhang, M., Tanaka, T., Hou, W., Gao, S., & Shinozaki, T. 2020. Sound-Image Grounding Based Focusing Mechanism for Efficient Automatic Spoken Language Acquisition. In *Proc. Interspeech 2020*, 4183–4187. URL: http://dx.doi.org/10.21437/Interspeech.2020-2027, doi:10.21437/Interspeech.2020-2027.

Appendices

A Diagrams and Plots

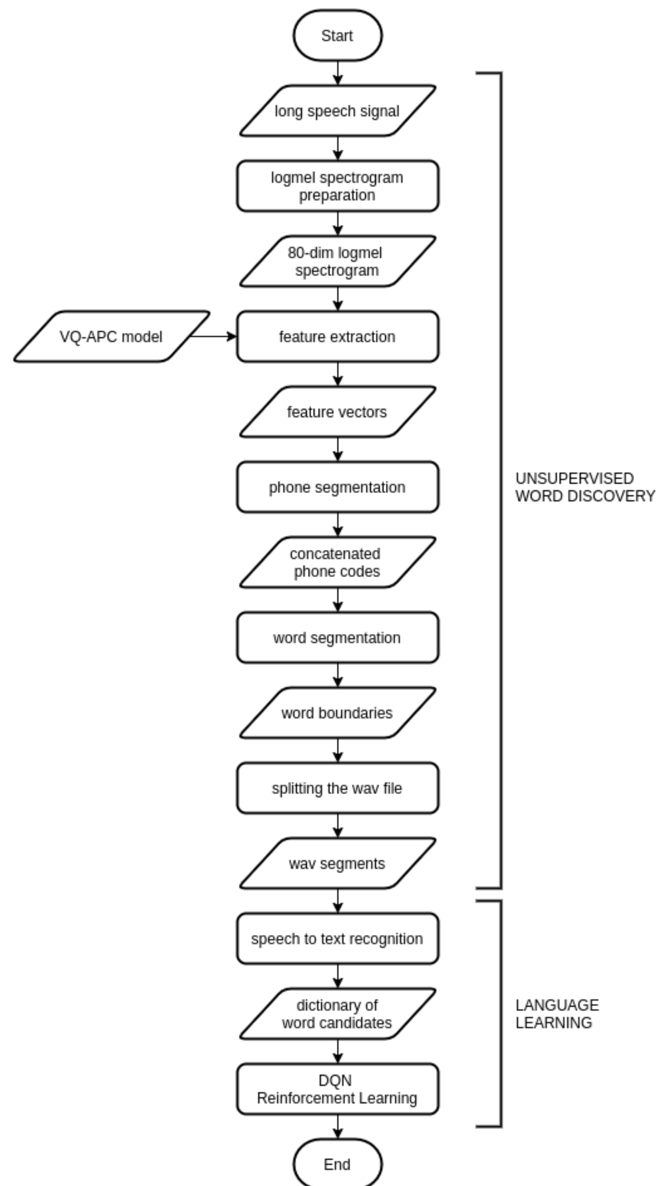


Figure 18: System architecture overview showing the main processes and the corresponding input and output.

B Tables

Results	Run 1	Run 2	Run 3	Run 4	Run 5
Number of segments	251	243	249	249	239
zero	12	11	11	12	8
one	8	7	11	7	6
two	9	8	7	8	8
three	11	8	9	10	7
four	6	6	6	7	6
five	6	6	6	4	7
six	8	6	9	8	8
seven	7	5	6	8	7
eight	3	3	3	2	2
nine	7	5	8	6	7
Recognized valid words	77	65	76	72	66
Recognition rate	15.40%	13.00%	15.20%	14.40%	13.20%
Over segmentation	-49.80%	-51.40%	-50.20%	-50.20%	-52.20%
Valid words / Segments	30.68%	26.75%	30.52%	28.92%	27.62%

Table 20: Segmentation results using WordSeg AG and codebook size 128.

Results	Run 1	Run 2	Run 3	Run 4	Run 5
Number of segments	478	478	486	480	483
zero	22	23	23	22	24
one	19	20	19	21	20
two	15	15	17	16	15
three	12	13	16	14	16
four	13	14	15	13	13
five	15	16	16	15	14
six	11	11	10	11	11
seven	17	19	18	18	19
eight	9	6	8	7	7
nine	24	22	23	23	24
Recognized valid words	157	159	165	160	163
Recognition rate	31.40%	31.80%	33.00%	32.00%	32.60%
Over segmentation	-4.40%	-4.40%	-2.80%	-4.00%	-3.40%
Valid words / segments	32.85%	33.26%	33.95%	33.33%	33.75%

Table 21: Segmentation results using WordSeg AG and codebook size 256.

Results	Run 1	Run 2	Run 3	Run 4	Run 5
Number of segments	453	455	453	457	454
zero	22	20	22	21	21
one	16	14	14	15	15
two	19	19	20	20	20
three	10	11	11	11	11
four	20	21	21	19	20
five	13	13	13	12	13
six	13	12	12	11	13
seven	14	16	14	14	15
eight	7	7	8	6	8
nine	18	17	17	16	17
Recognized valid words	152	150	152	145	153
Recognition rate	30.40%	30.00%	30.40%	29.00%	30.60%
Over segmentation	-9.40%	-9.00%	-9.40%	-8.60%	-9.20%
Valid words / segments	33.55%	32.97%	33.55%	31.73%	33.70%

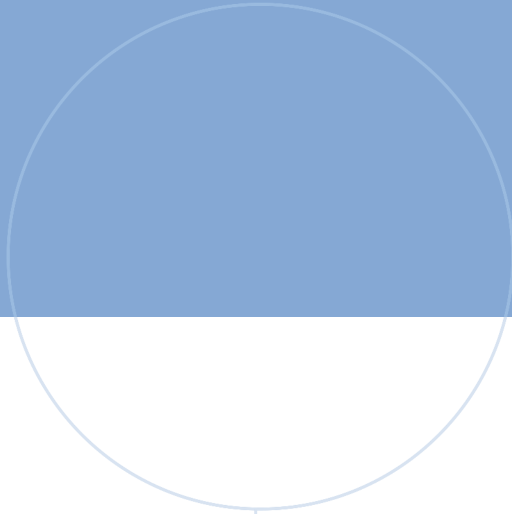
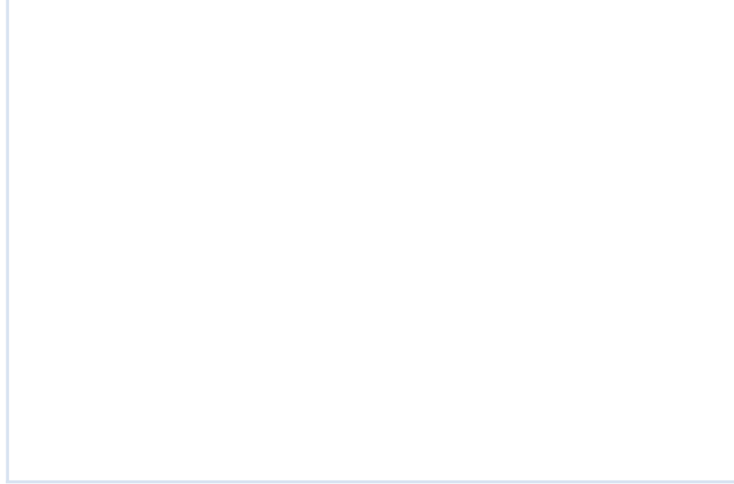
Table 22: Segmentation results using WordSeg AG and codebook size 512.

Results	code size 128	code size 256	code size 512
Number of segments	231	424	411
zero	10	17	14
one	8	13	12
two	7	10	9
three	8	7	12
four	10	12	12
five	6	13	12
six	4	9	5
seven	11	13	12
eight	6	4	11
nine	5	13	19
Recognized valid words	75	111	118
Recognition rate	15.00%	22.20%	23.60%
Over segmentation	-53.80%	-15.20%	-17.80%
Valid words / segments	32.47%	26.18%	28.71%

Table 23: Segmentation results using WordSeg TP.

Results	Run 1	Run 2	Run 3	Run 4	Run 5
Number of segments	736	712	741	746	736
zero	40	41	40	41	41
one	33	35	32	33	33
two	26	26	26	25	26
three	28	28	30	27	28
four	29	31	29	29	29
five	26	26	24	26	26
six	25	25	20	18	23
seven	34	35	35	30	32
eight	16	16	15	17	13
nine	28	27	28	28	27
Recognized valid words	285	290	279	274	278
Recognition rate	57.00%	58.00%	55.80%	54.80%	55.60%
Over segmentation	47.20%	42.40%	48.20%	49.20%	47.20%
Valid words / Segments	38.72%	40.73%	37.65%	36.73%	37.77%

Table 24: Segmentation results using ES k-means for word segmentation.

NTNU

Norwegian University of Science and Technology