



Norwegian University of
Science and Technology

Neural networks for sentiment analysis in AsterixDB

Johan Morten Kristoffer Finckenhagen

Master of Science in Computer Science

Submission date: July 2018

Supervisor: Herindrasana Ramampiaro, IDI

Norwegian University of Science and Technology
Department of Computer Science

Summary

As data is generated at an ever increasing rate, and social media are getting larger and more comprehensive than ever, the availability of these data and the possibility to analyze them, is growing as well. The capability of storing large masses of data has become trivialized over the years, and massive amount of information is stored every second. There is competitiveness in being able to extract meaningful information from data that were previously thought of as insignificant. Sentiment analysis is methods of retrieving an authors attitude towards the topic discussed. Having knowledge about the sentiment of the masses can be of significant market value, especially in i.e. knowing customers happiness with a product or service provided. Other useful areas can be mining twitter and follow opinions about the latest trends to develop new market strategies. Together with the late success of deep learning, it poses great interest to observe how these practices can be combined to analyze big data.

Previously the technique for processing natural language has been to create vocabularies of arbitrary order to use for further processing, but by introducing multi-dimensional vectors we can store semantic and syntactic information about words in a vector-space. These vectors has proven incredibly good for natural language processing, and more so as input format to deep learning algorithms as neural networks.

In this study several neural network models will be evaluated up against traditional classification algorithms, measuring accuracy, on sentiment classification of twitter messages. Each model will be tested for prediction speed on big data using AsterixDB, a big data management system support ingestion of streaming data. The results from this study gives the best accuracy score 84,02%, and the fastest networks can handle an average of about 10 000 tweets per second.

Sammendrag

Ettersom data blir produsert i stadig økende grad, og sosiale medier blir større og mer omfattende enn noen gang er tilgangen til informasjon fra disse kildene, samt muligheten for å analysere disse, også økt. Lagringskapasitet har økt i tilgjengelighet og det lagres enorme mengder data hvert sekund, og det er stor konkurranse i å trekke ut meningsfull informasjon fra informasjon som tidligere ble sett på som verdiløs. Analyse av sentiment er metoder for uthenting av forfatters sentiment ovenfor et objekt det skrives om og kan ha utrolig markedverdi i dagens marked, spesielt innenfor å måle kunders fornøydhets med produkter og å analysere trender utifra f.eks twittermeldinger. Sammen med den seneste suksessen ved dyp læring er det interessant og se hvordan disse kan kombineres for å analysere store mengder data.

Fra tidligere har det vært vanlig å å lage arbitrære vokabular for å utføre diverse typer prosessering av naturlig språk, men ved å integrere ord i flerdimensjonale vektorer kan vi lagre semantisk informasjon i et vektorrom. Disse vektorene viser seg å gi ekstremt gode resultater når det kommer til språkprosessering, spesielt som input til dype læringsalgoritmer som nevralt nettverk.

I denne undersøkelsen vil flere nevralt nettverksmodeller bli evaluert opp mot tradisjonelle klassifiseringsalgoritmer hva angår nøyaktighet ved analyse av twittermeldinger. Nettverksmodellene testes i AsterixDB, et databasesystem som støtter analyse av strømningsdata, og klassifiseringshastigheten til de ulike nettverkene blir målt opp mot hverandre. Nøyaktighetsmålingene på twitterdata når opptil 84,02%, og de raskeste nettverkene håndterer gjennomsnittlig rundt 10 000 tweets per sekund.

Preface

The thesis is written by Johan Morten Kristoffer Finckenhagen as a final project for a masters degree in Computer Engineering at the Norwegian University of Science and Technology.

Acknowledgements

I want to thank my supervisor Heri Ramampiaro for continous help and feedback on the project, as well as Thor Martin Abrahamsen for help regarding AsterixDB. Additionally I want to thank Ian Maxon and Xikui Wang for fast responses and answers to questions regarding AsterixDB.

Table of Contents

Summary	i
Sammendrag	ii
Preface	iii
Acknowledgements	iii
Table of Contents	vi
List of Tables	vii
List of Listings	ix
List of Figures	xi
Abbreviations	xii
1 Introduction	1
1.1 Project background	1
1.2 Project description and goal	2
1.3 Approach	2
1.3.1 Research Method	2
1.3.2 Limitations	3
1.4 Chapter outlines	3
2 Background Theory	5
2.1 Data Representation	5
2.1.1 Word2Vec	5
2.2 Machine Learning	6
2.2.1 Deep Learning	7
2.2.2 Neural Networks	7

2.2.3	Keras	11
2.2.4	Tensorflow	11
2.2.5	Deeplearning4j	12
2.3	AsterixDB	12
2.3.1	User Defined Functions	13
2.4	Task and Domain	13
2.4.1	Sentiment Analysis	13
2.4.2	Twitter	13
3	Survey	15
3.1	Related work	15
3.1.1	Related Research	15
3.1.2	Related Frameworks	16
4	Theoretical Solution	19
4.1	The goals of the experiment	19
4.2	Network Models	19
4.2.1	The training data	20
4.2.2	Training the network models	20
4.2.3	Hyper parameters	23
4.2.4	GPU or CPU for inference?	23
4.3	The AsterixDB Feed ingestion pipeline	24
4.4	Testing the system	26
4.4.1	The feed generator	26
4.4.2	Setting up the AsterixDB cluster	26
5	Results	29
5.1	Training results	29
5.2	Streaming results	30
6	Evaluation and Discussion	35
6.1	Evaluation	35
6.2	Discussion	36
7	Conclusion and Future Work	37
7.1	Conclusion	37
7.2	Future Work	38
	Bibliography	38

List of Tables

2.1	The formulae and plot of the different activation functions.	11
4.1	AsterixDB cluster system specifications.	26
4.2	Feed generator system specifications.	26
5.1	Results from the Sentiment140 project.	29
5.2	CNN model summary.	30
5.3	Data acquired from the AsterixDB feeds.	30

List of Listings

4.1	A preview of the created vocabulary	20
4.2	Sequence representation of the string "this is a tweet". Notice that "a" is left out, as the tokenizer does not carry on single characters.	21
4.3	Matrix representation of the embedding layer where l equals the length of the word_index, and m is the embedding size.	21
4.4	The query for initializing the feed in AsterixDB.	27
4.5	The dataformat of the tweets fed into the UDF and the output from the UDF stored in AsterixDB.	27

List of Figures

2.1	How the word embeddings are trained when using CBOW and Skip-gram.	6
2.2	The connections between nodes in a FCNN.	8
2.3	An overview of a CNN with two convolutional layers, one pooling layer, a fully connected layer and a binary classifier.	9
2.4	A simple recurrent neural network unfolded.	9
2.5	The LSTM (left) and GRU (right) layer structure.	10
2.6	An overview of the AsterixDB system	13
4.1	The training procedure for the neural network models.	21
4.2	A simplified overview of the FCNN architecture.	22
4.3	A simplified overview of the CNN architecture.	22
4.4	A simplified overview of the parallel CNN architecture.	22
4.5	A simplified overview of the CNN-RNN architecture.	23
4.6	Performance of CPU and GPU on the larger CNN analyzing a total of 2.7 million tweets.	24
4.7	An overview of the AsterixDB feed ingestion pipeline.	25
5.1	Results from the FCNN with 64 neurons in the hidden layer.	31
5.2	Results from the FCNN with 256 neurons in the hidden layer.	31
5.3	Results from the CNN with 100 feature maps and a FC layer with 64 neurons.	32
5.4	Results from the CNN with 200 feature maps and a FC layer with 128 neurons.	32
5.5	Results from the parallel CNN network.	33
5.6	Results from the model combining convolutional and recurrent layers.	33

Abbreviations

ADM	=	Asterix Data Model
AI	=	Artificial Intelligence
BD	=	Big Data
BDMS	=	Big Data Management System
CBOW	=	Continuous Bag Of Words
CC	=	Cluster Controller
CFM	=	Central Feed Manager
CNN	=	Convolutional Neural Network
DAG	=	Directed Asyclic Graph
DL	=	Deep Learning
DM	=	Data Mining
DNN	=	Deep Neural Network
FC	=	Fully Connected
FCNN	=	Fully Connected Neural Network
FM	=	Feed Manger
GRU	=	Gated Recurrent Unit
LSTM	=	Long short term memory
ML	=	Machine Learning
NB	=	Naive Bayes
NC	=	Node Controller
NLP	=	Natural Language Processing
RNN	=	Recurrent Neural Network
SST	=	Stanford Sentiment Treebank
SVM	=	Support Vector Machine
TPS	=	Tweets Per Second
UDF	=	User Defined Function

Chapter 1

Introduction

In this chapter the overall context of the project will be presented. It will include a description of the project, the background and goals of the project, and lastly there will be an outline of the forthcoming chapters.

1.1 Project background

Over the last decade Big Data (BD) has come to be something that is generated from everything around us, and data has gone from being static to something that is vital for all businesses survival. Phones, tablets, watches and access cards, everything creates data that is being mined and used by companies to personalize our perceptions and surroundings. What advertisement we are being shown, and what we ultimately end up purchasing can be heavily influenced by prosecutors sitting on information about our search history, where we have been or what we have bought earlier.

As the capability of storing immense amounts of data has become feasible over the last decade, new areas of data analysis have seen the light of day. Data Mining (DM) has become a prevalent study within computer science, and sheds light on what information can be generated from existing data. Within this practice Machine Learning (ML) has gotten a good foothold, and helps us extract useful information from otherwise fruitless data. ML algorithms has the ability to train and learn from gathered data, and in turn produce rules that helps us make educated estimations and predictions. They are able to learn strong patterns used for classification in both text, images and other multimedia. They provide a user friendliness in that they do not require feature extraction, but rather teach themselves useful features.

There are already an abundance of readily available tools and techniques for ML and DM, and one that has gotten a lot of consideration over the last couple of years is Deep Learning (DL), more precisely Deep Neural Networks (DNN). Neural Networks (NN) mimics our understanding of the human brain, and has proven to give extraordinary results when applied to classification.

The study is based on a previous preliminary study on NNs used to process twitter

datastreams. The preliminary project used a framework called Deeplearning4j and performed sentiment analysis on tweets using a Recurrent Neural Network (RNN) with Long Short Term Memory (LSTM) layers. Deeplearning4j was, and still is the only DL framework completely written in Java. In this study a transition to Keras with the Tensorflow backend for building models has been made, and the Tensorflow Java API is used for inference in AsterixDB. The reasoning behind this is explained in chapter 4.

1.2 Project description and goal

The main goal of this project is to perform a comparative study on the performance of different ML algorithms applied on streaming data. The main focus will be put on NN-based algorithms, comparing their differences in terms of speed performance and scalability. As a case, the study will be done on sentiment analysis on a twitter-stream using AsterixDB feeds.

Sentiment analysis has for a long time been performed using traditional classification algorithms such as Naive Bayes (NB) and Support Vector Machines (SVM). These are in general quite fast and not too computationally expensive. DNNs offer tools to incorporate a larger part of the Natural Language Processing (NLP) research field into its architecture, and are capable of learning more complex features than can be manufactured with traditional classification algorithms. For simple classification tasks, simple, shallow and efficient NNs can be crafted and perform incredibly well. But as more complex features need to be extracted, they also require deeper and more computationally expensive networks.

With this in mind this study will take a look at different NNs used for sentiment analysis, and what each component of these networks brings to the table in terms of speed and accuracy, as well as how similarities can be drawn between these components and traditional classification and NLP fundamentals. The study will try to answer the following questions.

What are the different building blocks that make up a good neural network for sentiment analysis, and what does each element add to performance and efficiency?

Are neural networks a good alternative to classic ML algorithms for sentiment analysis when processing huge amounts of real-time data?

1.3 Approach

1.3.1 Research Method

The first parts of the study is review of relevant literature, taking a closer look at what is the state-of-the-art network models, and what results have been achieved in terms of sentiment analysis. This part will also present alternative frameworks for datastreaming as well as several DL frameworks. The latter part will present some standard DL architectures and

two models inspired by some of the research papers. There will be performed an analysis of performance in regards to both accuracy and speed on each network.

1.3.2 Limitations

The models presented in the literature study are not specified in great detail, but in more general terms regarding the architecture of the models. Thus, the models presented in the latter part of this study will take inspiration from these models, but they do not necessarily reflect the original models capabilities. The training and testing data are also different, and making a direct comparison to the original findings is nonsensical. The results from this study will be compared to results from other models on the same training and test data in terms of accuracy. As for a measure of speed, only one other study has done similar testing.

Due to hardware limitations the experiment in the latter part of the study is done on a single computer simulating a small cluster, while the datastream is generated and sent from another machine.

Twitter has since the dataset was gathered increased the maximum length of a tweet from 140 to 280 characters, thus tweet lengths of 140 characters will be used for the entirety of the experiment.

1.4 Chapter outlines

In chapter 2 a brief explanation of the concepts used in the project will be presented, as well as an introduction to each of the technologies used to implement the system. A literature study of similar frameworks and relevant research will be presented in chapter 3. Chapter 4 will outline the theoretical solution and how the experiments have been executed. The results from the experiments will be presented in chapter 5, and chapter 6 will demonstrate an evaluation and discussion around the results presented in chapter 5. Lastly, chapter 7 will conclude the thesis and present some future work.

Background Theory

In the case of this study ML will be used to classify tweets, and therefore we need a good data representation of words.

2.1 Data Representation

Data comes in several different formats, and one of the more prominent ones is textual data. This form of sequential data is often conformed by a sequence of either words or characters, and by analyzing these sequences much can be learned about their contents. We can use ML on text data for a number of purposes, such as sentiment analysis, document classification and to a degree question answering. As NNs do not comprehend text data, there is need for a good data representation.

Vectorization is used in NLP as a means to represent sequential text data as real number vectors. The simple way of doing this is through one-hot encoding, in which a vocabulary is generated from the input data, and the vector representation of a word corresponds to the length of the vocabulary. The index of the word in the vocabulary corresponds to where in the vector there will be a 1, and the rest of the vector will be 0 valued. "One hot encodings" are sparse, hard coded vectors that consume a lot of memory as the vocabulary grows. A way of reducing the dimensionality of the representation is by using dense, trained word embeddings.

2.1.1 Word2Vec

Word2Vec is an implementation of a NN that learns word embeddings. The two layer NN was developed by a team at Google led by Tomas Mikolov. Previously a popular approach to statistical language modelling has been N-grams and Bag Of Words (BOW). N-gram looks at the n adjacent word or tokens in a sentence to find the semantic context between them. BOW collects all words in a document in a list, but does not keep memory of the location of the words in relation to each other. BOW and n-grams can be combined to Bag of n-grams as well, but some of the positional information is still lost. Word2Vec is

a more complex language model than its predecessors and outperforms n-grams Mikolov et al. (2013).

Word2Vec proposes two different models trained by either Continuous Bag of Words (CBOW) or Skip-gram. The CBOW model is trained with a log-linear classifier using an input of 8 words, or word embeddings. These 8 words are divided into 4 future and 4 history words, and the network is trained to predict the word between these groups of 4. The skip-gram model is trained similarly, but while CBOW uses neighbouring words as input, the skip-gram model uses the middle word to predict its neighbours as shown in Figure 2.1.

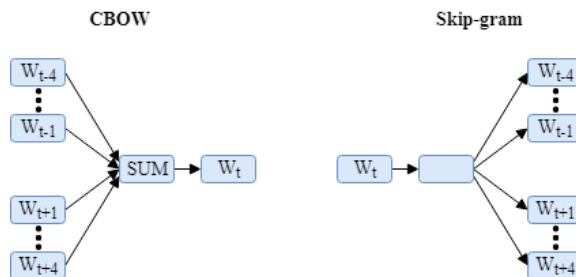


Figure 2.1: How the word embeddings are trained when using CBOW and Skip-gram.

An interesting aspect of the Word2Vec model is how we can query the words in the corpus and the different kinds of relationships we can find. Words that have similar semantics e.g. different cities, will appear close to each other in the vector space and at the same time have similar distances to words they are connected to e.g. the countries in which they are located. This means that we can query a Word2Vec model and get results like Oslo : Norway :: Stockholm : Sweden where ":" is denoted as "is to" and "::" is denoted as "what". We can then read the result of the query as "Oslo is to Norway what Stockholm is to Sweden". Using the offset between words in the vectorspace, the Word2Vec model is able to discover more than just syntactic similarities, and it has been shown that performing simple algebraic equations on vectors yield interesting results. The equation $vec(king) - vec(man) + vec(woman) = vec(queen)$ is an example of such an algebraic operation.

2.2 Machine Learning

ML differs from traditional programming in that instead of providing rules and input data to get an answer, the user provides answers and input data to generate rules, which can subsequently be used to generate answers to new input data. In ML we train an algorithm to find a structure in the training data from which rules to automate the task can be generated. ML can be defined as "searching for useful representations of some input data, within a predefined space of possibilities, using guidance from a feedback signal", as stated in Chollet (2017). There are two different types of training; supervised and unsupervised. When using supervised learning training datasets are given which has previously been manually classified, and the algorithm are able to learn from the training data how to

classify new data. In unsupervised training the algorithms will learn features in the data and be able to separate data from each other into clusters. What defines these clusters are on the other hand not always apparent. There are three prerequisites that need to be in place to use ML. First we need to have a set of data points, secondly we need examples of expected output and lastly a way to measure performance.

DL is subfield of ML, where the focus is on creating deep, layered ML algorithms, where each successive layer outputs increasingly purified features.

2.2.1 Deep Learning

NN and Artificial Intelligence (AI) have for the last couple of years gotten a lot of attention in the media, and are expanding fields of interest both academically and business related. Self-driving cars, intelligent personal assistants like Siri and Alexa and chatbots are all over the news. NNs plays an important role in many of these concepts, and makes an effort to create an abstraction of how we believe the human brain makes decisions. NNs consists of up to billions of connected processor nodes called neurons each having an activation function that activates upon receiving the appropriate signal from other neurons in the network. We divide NNs into two sub-categories, shallow and deep NNs. Shallow networks have been around for along time, while DNNs date back to the 1960s. Shallow networks are defined by having two or less layers, while DNNs has more.

To understand what decides a given network's momentum, there is a necessity to understand the different architectures and concepts of the NNs. The next sections will give a brief explanation of some of the core concepts used in the models presented later in this study.

2.2.2 Neural Networks

NNs are built up by four main concepts. Firstly, different network layers that are stacked on each other is what makes up the core model. The input data and labels are needed for a network to train itself. A loss function measures the performance of the model and lastly an optimizer tells the network how it should react to the loss function, and how training of the network proceeds.

Fully Connected Neural Networks

The Fully Connected Neural Networks (FCNN) is the simplest form of a NN, with a number of fully connected (FC) hidden layers. Data fed to the input layer is sent to all nodes in the successive layer, and output from all nodes are sent to every node in the next layer until it finally arrives at the output layer. As every input is connected to every node in the hidden layer, and every layer is FC to each consecutive layer, as input size increases these networks become increasingly inefficient and computationally expensive.

Convolutional Networks

Convolutional Neural Networks (CNN) was pioneered by LeCun et al. (1998) and is implemented by combining local receptive fields, shared weights and spacial subsampling.

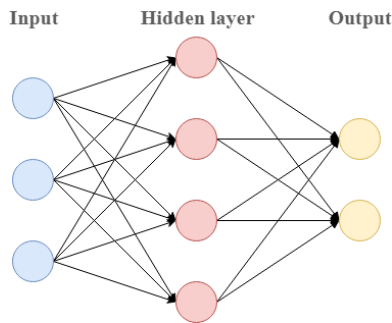


Figure 2.2: The connections between nodes in a FCNN.

Neurons use local receptive fields to extract visual features which are subsequently fused to create features of higher order. CNNs assume that inputs are n -dimensional tensors, often images, or matrices consisting of word embeddings in the case of this study. As a tensor is input into a CNN it is usually read as a tensor of 3 or 4 dimensions; batch size, width, height and channels. Channels are used primarily for colour images, and is not necessary in the case of this study. The receptive field, or filter size, of a neuron is connected to a local region of the input, and there may be one or more neurons looking at the same receptive field, thus finding different features for each field. Convolution layers are initialized with several hyperparameters; the filter size, depth, stride and padding. The filter size decides how big the receptive field of each neuron will be, the depth is how many filters are applied, stride is how big steps are taken while sliding the filter and padding is whether or not the input will be padded, as the convolution operation will eat from the input size as shown in Figure 2.3

Just as with images convolutional layers can be used on NLP tasks such as sentiment analysis. A one dimensional convolution uses filters of width equal to the input width and a variable height given by hyperparameters. This gives one dimensional convolutional layers the ability to extract features of n -grams, or sequences of n trailing words, and thus be able to learn local patterns in a sequence of words in the same way two dimensional convolutional layers can learn about features in an image given a neighbourhood of pixels.

Both FCNN and Convolutional Neural Networks are what is called feedforward networks, which the topology is that of a Directed Acyclic Graph (DAG).

Max-Pooling Layer

Pooling layers are an often used technique in CNNs to reduce the spatial size of the output, also known as downsampling. The most common usage is max pooling with a kernel size of 2 and stride 2, which effectively removes 75% of all activations from the previous layer, but there is also discussion around whether pooling layers are necessary, or if they should be discarded in favor of higher stride values on some convolutional layers as suggested in Springenberg et al. (2014).

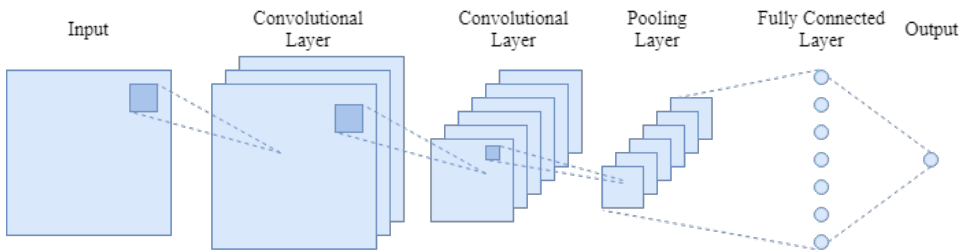


Figure 2.3: An overview of a CNN with two convolutional layers, one pooling layer, a fully connected layer and a binary classifier.

Recurrent Neural Networks

RNNs differentiate themselves from other neural network configurations in that they look at data sequentially, just like humans read sequentially. This specific ability makes RNNs great for NLP tasks. RNNs work in timesteps, and at each timestep an input is processed and produces an output that is sent as input to the next timestep, thus the input at the next timestep is a concatenation of the output from the last timestep and the current input as shown in Figure 2.4. Considering the two sentences "I'm sad" and "I'm not sad", by having information about what term that occurs prior to "sad", RNNs are capable of seeing the difference in sentiment between these sentences. One problem with these networks are, as Bengio et al. (1994) mentions, when the distance between two dependent timesteps becomes long, the RNNs struggle to find them.

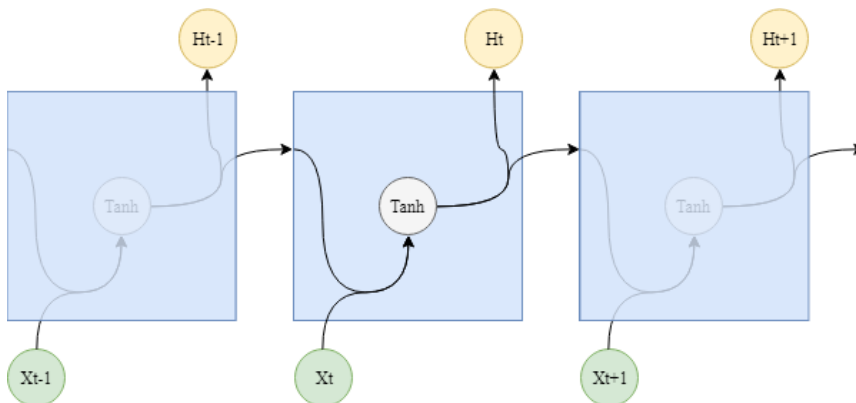


Figure 2.4: A simple recurrent neural network unfolded.

Long Short Term Memory and Gated Recurrent Units

To combat the difficulty of long-term dependency Hochreiter and Schmidhuber (1997) introduced the Long Short Term Memory network. While the regular RNN consists of one neural network layer, an LSTM layer incorporates 4 layers within repeating module, three

sigmoid layers and a tanh layer as shown in Figure 2.5. The uppermost line in the layer, called the conveyor belt, can easily transmit data down the network in a cell state, while the sigmoid and tanh gates provides a way for the network to decide how much of the input to let through. The leftmost sigmoid layer, the "forget gate", uses the previous output h_{t-1} and the input x_t to decide how much of the conveyor belt shall be forgotten. The next sigmoid and tanh layers, using a pointwise multiplication, decide what new information should be added to the cell state, and it is added with a pointwise addition. The last part of the LSTM decides what will be output from the layer at the current timestep.

The Gated Recurrent Unit (GRU), introduced by Cho et al. (2014), works in a similar way as the LSTM network, though with a few key differences. The GRU simplifies the module by combining the forget and update units, and removing the conveyor belt entirely.

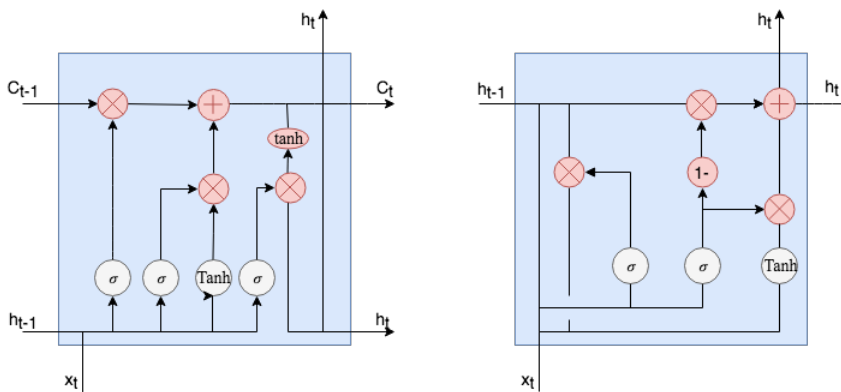


Figure 2.5: The LSTM (left) and GRU (right) layer structure.

Activation Functions

Every node in a neural network has an activation function that computes the output of the node. The simplest neuron, the perceptron has a simple step function that outputs either 1 or 0, while others usually have more intricate functions. Different activation functions serve different purposes in neural networks. Three fundamental activations that see extensive use in neural networks are the sigmoid, relu and tanh activation functions shown in Table 2.1.

Loss functions

The training of neural networks comprises of several forward and backward passes. A forward pass makes an inference from the given input and a backward pass computes the gradients of the network using the chain rule. During the training stage a loss function uses the deviation between the output from an inputs forward pass and the expected output to calculate the loss of the given forward pass. There are several loss functions readily available. Mean Squared Error (MSE), Categorical Cross Entropy and Binary Cross Entropy are the most used loss functions in today's neural networks.

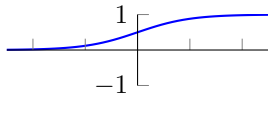
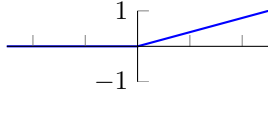
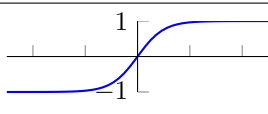
Activation	Function	Plot
Sigmoid	$f(x) = \frac{1}{1+e^{-x}}$	
ReLU	$f(x) = \begin{cases} 0 & : x < 0 \\ x & : x \geq 0 \end{cases}$	
tanh	$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	

Table 2.1: The formulae and plot of the different activation functions.

Optimizers

Optimizers decide how weights are updated during the backward passes, or backpropagation, of the training stage. Several great optimizers has been proposed over the last years, all stemming from Gradient Descent. Gradient descent involves computation of the gradients between every layer in the network, and the way this is performed is by calculating the derivative between nodes backwards in the layer, and moving slightly in the opposite direction of the tangent. The step taken "downhill" in the gradient is decided by the learning rate of the optimizer.

2.2.3 Keras

Keras¹, distributed under the MIT licence (<https://opensource.org/licenses/MIT>), is a deep-learning library for python, initially developed as a research tool for scientists to easily experiment and produce neural network models. Since then it has become the preferred community choice as it provides a high level API that produce a user-friendly, modular DL experience. It supports implementation of both CNNs and RNNs, and a plethora of different activation functions, convenience layers and techniques for optimization and regularization. Keras does not concern itself with the low-level operations on tensors, instead it is supported by three different backends, Google Tensorflow, Theano and Microsoft CNTK.

2.2.4 Tensorflow

Tensorflow² is the second interface and implementation for expressing and executing ML algorithms released by Google following DistBelief, the outcome of early works on The Google Brain project. Tensorflow grant a powerful tensor library for dealing with data representation, as well as training and inference algorithms for deep neural networks. It is a

¹<https://keras.io/>

²<https://www.tensorflow.org/>

system developed with usability and flexibility in mind, as well as being high performance for usage in deployment.

2.2.5 Deeplearning4j

Deeplearning4j³ is an open source project and toolkit for DL in Scala and Java. It is mainly being developed by a company called Skymind. It offers several tools for DL, more notably the components Deeplearning4j and ND4j, which will be used in this project. The Deeplearning4j component supplies everything needed for setting up a neural network and ND4j supplies a mathematical library for processing matrix data.

2.3 AsterixDB

AsterixDB⁴ is a Big Data Management System (BDMS) developed by faculty, staff and students at UC Irvine and UC Riverside. The development was initiated in 2009, and the first open source release dates back to 2013. Since then it is continuously being worked on as part of the Apache Software Foundation (2018). AsterixDB aims to provide a rich feature set that will distinguish it from the abundance of BD platforms that have emerged over the last decade, and one of the major features is that it is a parallel database system, that provide functionality both for analysis and storage of data. The idea behind AsterixDB is to provide a system that support ingestion, indexation and management of, while also providing tools for querying and analyzing, huge amounts of semi-structured data. An overview of the AsterixDB system can be seen in Figure 2.6

AsterixDB comes shipped with a flexible data model called the Asterix Data Model (ADM), which builds on and expands the traditional JSON-format with e.g. spacial and timely formats, which are common in today's BDMS. To store data in AsterixDB a Dataverse must first be created, which serves as what would be defined as a database in relational database management systems (RDBMS). Datasets are collections of datatype instances within a dataverse, and each datatype must be defined as either open or closed. Open datatypes has some attributes decided "a priori", while additional attributes can be added. Closed datatypes on the other hand have a strict set of attributes that can not be altered. To support this AsterixDB has developed its own SQL-like query language called SQL++ which support data ingestion among other extra features not available in regular SQL.

Data ingestion in AsterixDB is done through the feed adapter, which creates a connection to the feed, receives, parses and translates data to the ADM. Additionally AsterixDB performs pre-processing of the data, e.g. adding/removing attributes, data analysis or feature extraction. The feed is initialized with a policy, which can be either of the following; Basic, Spill, Discard, Throttle or Elastic. The Basic policy stores overflow in its buffer memory, Spill flushes overflow to disk for later processing, Discard discards the overflow and processes only the data it has capacity to. Throttle randomly filters out records arriving in the stream while Elastic scales the system accordingly, so it will be capable of receiving the arriving records. The feed ingestion pipeline will be explained in greater detail later in the thesis.

³<https://deeplearning4j.org/>

⁴<https://asterixdb.apache.org/>

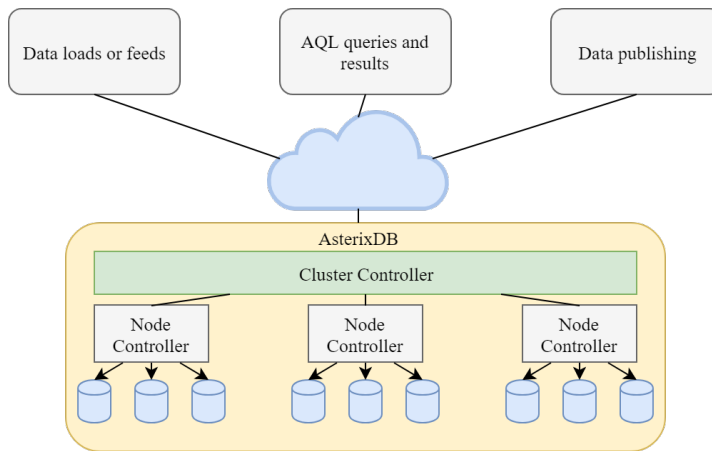


Figure 2.6: An overview of the AsterixDB system

2.3.1 User Defined Functions

As part of the pre-processing of data coming in through the feed adapter, AsterixDB supports what is called User Defined Functions (UDF). AsterixDB supplies some predefined UDFs, but users are also able to create their own UDFs in Java and install them on their AsterixDB instance. As a feed is initialized a chosen function may be applied to all incoming data, specifying the input and output format of each data element. This gives AsterixDB a great and versatile tool for data analysis, which will be utilized in this project.

2.4 Task and Domain

2.4.1 Sentiment Analysis

Sentiment analysis, also known as opinion mining, denotes the process of extracting an authors sentiment towards the subject of a given text, and is one of the most common text classification tools.

Businesses are eager to learn what the consumers joint sentiment towards their product, or more so what the sentiment towards competing products are. A shift in sentiment towards certain products or services can be an indication of a change in trends, and having knowledge about trends can in return make a company adaptable and able to come up with new strategies for tackling this shift.

2.4.2 Twitter

Twitter⁵ has over the recent years marked its position as one of the largest Social Media Platforms with over 330 million active users daily Inc. (2017). It is a social network where

⁵<https://about.twitter.com/>

users are able to post "tweets", which are messages of a maximum length of 280 characters (as of 7. November 2017). Tweets are grouped into topics by the use of hashtags, i.e. *#BigData*, and thus tweets can easily be sorted into different categories for further processing. One of the main challenges with analyzing tweets are the varying language and structure, use of slang and abbreviations not often used in other literature.

Chapter 3

Survey

3.1 Related work

Significant traction in 2012 has instigated several breakthroughs in sentiment analysis using DNNs, pushing the boundaries of what is state-of-the-art. With a high focus on increasing the accuracy of the networks, little to none research has been done concerning the time-efficiency of NNs and while accuracy is of great importance, when handling streaming data, time is also an important factor. The following sections will explore state of the art NNs, frameworks for sentiment analysis, as well as explore some of the latest data stream processing tools.

3.1.1 Related Research

Sentiment Analysis and Neural Networks

There has been done thorough research on how to efficiently speed up the training phase of NNs, but little to none research on how well they perform while predicting with focus on speed. The reasoning behind this is a huge focus on accuracy, and making the best possible prediction. When classifying streaming data on the other hand, speed becomes a concern as well, and is an important factor when creating appropriate models. This chapter will give insight into some of the state of art models for sentiment analysis and how well they perform, as well as look at how well some different backends perform against each other. After several years of being the state of art, and community choice of classification tools, several frameworks and libraries for NNs has emerged. Along with Keras, Tensorflow and Deeplearning4j, frameworks like PyTorch¹, Caffe² and Microsoft Cognitive Toolkit (CNTK)³ are eminent choices. Exploring the many available tools and deciding what to use can be difficult, and there can be prominent differences in performance on various

¹<https://pytorch.org/>

²<https://caffe2.ai/>

³<https://www.microsoft.com/en-us/cognitive-toolkit/>

tasks. Kovalev et al. (2016) has done a thorough comparison between several de facto frameworks on FCNN, and recorded result both on accuracy, training time, prediction time and size of the source code, coming to the conclusion that Theano with Keras are ranked on top, followed by Tensorflow, and ranking Deeplearning4j on bottom.

As both CNN and RNN models have proven significant usefulness Ouyang et al. (2015) proposes a model consisting of 3 convolutional layers, dropout layers, max-pooling and normalization layers, followed by the last FC fine grained classification layer, classifying texts into 5 different categories (very negative, negative, neutral, positive and very positive). The word embeddings used are pretrained embeddings from Word2Vec. The performance outclasses several well known neural nets, as well as classic classification methods as NB and SVM. dos Santos and Gatti (2014) proposes a network using not only word embeddings, but also character embeddings, using convolutional layers to extract character to sentence level features. This Character to Sentence Convolutional Neural Network (CharSCNN) slightly outperforms the Sentence Convolutional Neural Network (SCNN) they compare it to, as well as the SVM, NB and RNN classifications proposed by Socher et al. Similar to the SCNN solution implemented by Cicero, Kim (2014) proposes several CNN networks based on word embeddings, both with randomized, static, non-static and multi-channeled word embeddings. On the Stanford Sentiment Treebank with binary classification, the multi-channel network provides outstanding results compared to other state of art classification models. Both Wang et al. (2016a) and Wang et al. (2016b) incorporates a model combining convolutional layers with LSTM and GRU layers getting state of art result on the "Rotten Tomatoes" Movie Review dataset and Stanford Sentiment Treebank (SST) dataset.

3.1.2 Related Frameworks

Deep learning

Just in the last couple of years, several toolkits and frameworks for computing artificial NNs have emerged, most of them still in their early stages and undergoing continuous development. One of the earlier implementations of DL frameworks, Theano⁴, developed by the Montreal Institute of Learning Algorithms, and while being discontinued since late 2017, has been an inspiration to the development of Tensorflow, and is offered as a backend possibility in Keras. Also worthy a mention is Torch⁵ and Caffe⁶, which are well performing frameworks, but lack community support and active development. Torch also has the downside of being programmed in Lua, a language that has seen less use in recent years, thus the community support is limited.

One of the main competitors to Keras and Tensorflow, PyTorch, has become one of the community choices over the years and is often observed in high ranking entries on Kaggle⁷ leaderboards. It provides a strong tensor library just like Tensorflow, and an easy to use python API like Keras. PyTorch supports dynamic computational graphs as opposed to static graphs provided by Tensorflow. Braun (2018) provides a study of PyTorch

⁴<http://deeplearning.net/software/theano/>

⁵<http://torch.ch/>

⁶<http://caffe.berkeleyvision.org/>

⁷<https://www.kaggle.com/>

performance on LSTM networks while comparing to both Tensorflow and Keras showing similar performance as the Tensorflow backend both GPU and CPU.

Caffe2, built as an improvement by Facebook on the former Caffe framework, and provides multi-GPU support and is aimed to be used in industrial-strength applications, while PyTorch, similar to Keras and Tensorflow, excels at providing great tools for research and experimentation with more advanced NN models. Caffe2 also provides functionality for exporting all models in pure C++, giving the user the ability to deploy models without python. These models can thus also be used with Java with additional software e.g. JavaCPP⁸.

Microsoft offers an open source DL solution for commercial-grade application in their CNTK. It supports python, C and C++, while also offering a stand-alone tool supplied with its own language, BrainScript.

Apache MXNet⁹ is another outstanding framework, supported by both Microsoft and Amazon. It offers API for JavaScript, R, Go, Python and C++, being one of the most accessible frameworks in terms of programming languages. Together Microsoft and Amazon have launched an interface for MXNet called Gluon¹⁰ that simplifies building and prototyping NNs, much like how Keras works with the Tensorflow backend.

Big Data Streams

BD is rapidly and asynchronously generated, and often involves unbounded data of various dimensions. These kinds of datasets is unfitted for the traditional relational databases and DM frameworks designed to handle homogenous data. The traditional way of DM, where data is loaded as a whole into main memory and processed, either on one machine or on small simple clusters, proposes problems as the amount of data becomes overwhelmingly large, and keeping it all in-memory at once becomes an impossibility. To counter this obstacle several elastic and virtualized cloud based frameworks for dataprocessing has been implemented, moving from the store-then-process paradigm to ad-hoc queries processing streaming data in real-time.

In recent years the state-of-the-art systems for processing datastreams has been implementations of softwares stacks, folding complementing frameworks into a merged BDMS. One of the community choices for this type of setup is Apache Spark¹¹ and Apache Cassandra¹², where Apache Spark handles streams in near-real-time by processing batches with small window sizes, and Apache Cassandra handles the persistent storage.

Another choice proposed by Grover and Carey (2015) which they used for comparison with AsterixDB regarding scalability and fault-tolerance is Apache Storm¹³ with MongoDB¹⁴. Apache Storm is a real-time stream processing engine, that uses spouts and bolts as datasources and operators respectively, and MongoDB provides resilient storage in the form of key-value stores. Storm has also seen extensive use at Twitter for processing

⁸<https://github.com/bytedeco/Javacpp>

⁹<https://mxnet.apache.org/>

¹⁰<https://gluon.mxnet.io/>

¹¹<https://spark.apache.org/>

¹²<http://cassandra.apache.org/>

¹³<http://storm.apache.org/>

¹⁴<https://www.mongodb.com/>

tweets after the acquisition in 2011, and open-sourcing in 2012 Toshniwal et al. (2014). The extensive use and need for increasingly fast stream processing has led to the development of Twitter Heron¹⁵, which provides Storm compatibility and improvements on latency, throughput and CPU core usage Kulkarni et al. (2015).

Apache Flink¹⁶ is also one of the newer platforms introduced in recent years and quickly got traction, being used by large scale companies as Uber, Zalando and Alibaba¹⁷. Flink provides both streaming and batch processing tools, and supports connections to several other frameworks as Kafka, ElasticSearch, HDFS, Amazon Kinesis Stream and Cassandra¹⁸.

One of the main drawbacks of many of these implementations is the glued topology consisting of different layers of frameworks. This is where AsterixDB is different, as it provides one system for both handling streams and persisting data.

¹⁵<https://apache.github.io/incubator-heron/>

¹⁶<https://flink.apache.org/>

¹⁷<https://flink.apache.org/poweredby.html>

¹⁸<https://ci.apache.org/projects/flink/flink-docs-master/dev/connectors/>

Theoretical Solution

The system implemented provides tools for training and testing NNs offline using word embeddings and a series of different model proposals. The models will be installed and used in AsterixDB UDFs to classify tweets arriving through AsterixDB feed adapter. The following chapter will present the models, thoroughly explain the AsterixDB feed ingestion pipeline, as well as how the experiment has been carried out. The tool used for modelling the NN in this experiment is Keras, as it is easy to set up, use and provides a great and userfriendly API for experimentation. More importantly it runs on the Tensorflow backend, which in turn provides a Java API, so the models can be used for inference in the AsterixDB UDF. The study provided by Kovalev et al. (2016) on different framework speed also provides reasoning behind the switch to Tensorflow, as it outperformed Deeplearning4j on every front.

4.1 The goals of the experiment

The aim of the experiment is to document whether or not NNs are applicable to perform sentiment analysis on huge amounts of data arriving from a continuous datastream. The experiment will serve as a proof of concept on how well these algorithms scale when used in conjunction with AsterixDB as UDFs. There will be put effort into creating models that are fast, as well as accurate. The architecture of each model will be presented, and the experiments conducted will give an implication of how the different components of each model affect performance.

4.2 Network Models

In this study several NN models are presented and measured up against each other, and while accuracy is not the main focus, it is taken into account. Some general purpose models will be proposed, as well as some models inspired by the papers presented in chapter 3. The training of the network is performed in Keras with a Tensorflow backend,

and the models are exported and imported in the UDF with the Tensorflow Java API.

4.2.1 The training data

The training data used in this study is a collection of 1 600 000 tweets gathered by students at Stanford University as a research project called Sentiment140¹, presented in Go et al. (2009). The dataset is available as a .csv file with tweets labeled as 1 or 0, 1 translating to positive and 0 to negative. Tweets have been collected by querying the twitter API for tweets that contain positive or negative emoticons, and the emoticons act as a noisy label for determining the sentiment of the tweet. Every tweet containing both positive and negative emoticons are pruned, as they are considered neutral along with non-emoticon tweets. The tweets stored in the dataset have had their emoticons removed as to give pure textual input for training. One feature that makes the Sentiment140 project a great tool is automated aspect of gathering training data and performing sentiment classification through distant supervision. The study also proposes several forms of text preprocessing needed to achieve high classification accuracy. Lastly the study also provides some accuracy measures for sentiment analysis using some traditional classification algorithms.

4.2.2 Training the network models

Before we start training the models the dataset needs to be properly cleaned up. A Data-Cleaner processes all data by removing URLs, transforming everything to lowercase and remove punctuation. A tokenizer creates tokens from every tweet in the training set, while simultaneously creating a `word_index` that carries information about the l most occurring terms as shown in Listing 4.1. Every tweet is then assembled into a sequence of integers equivalent to their position in the `word_index` and padded with zeroes to match the set length of input n into the networks Listing 4.2. Figure 4.1 shows pipeline for the training of the models.

```
{
  "to" : 1,
  "the" : 2,
  ...
  "ybca" : 99999,
  "polska" : 100000
}
```

Listing 4.1: A preview of the created vocabulary

An inspection of the training data shows that the average tweet consists of 11,7 words, and that 99,9% of all the tweets in the dataset has 28 words or less. The longest tweet in the corpus is 40 words long. The NNs proposed later will need an input of fixed size, hence an upper limit to the size of each individual tweet must be set. Setting a maximum length of 40 words makes sure that almost every new tweet will have all its words included

¹<http://help.sentiment140.com/>

when performing sentiment analysis. Having 99,9% of all tweets have the tweet as a whole included seems justifiable as well, and will make each network lighter and improve performance in relation to speed.

"this is a tweet" : [0 0 0 ... 0 24 8 217]

Listing 4.2: Sequence representation of the string "this is a tweet". Notice that "a" is left out, as the tokenizer does not carry on single characters.

An approach called k-fold was used to split the dataset into training, validation and test sets respectively, thus having a training set of 1 568 000 tweets, validation of 16 000 tweets and a test set of 16 000 tweets. Similar for all models are the use of word embeddings provided by Word2Vec. Several word2vec models has been tested for training. Two 100-dimensional models trained on the Sentiment140 data with CBOW and skip-gram respectively has been tested. The second model is a 200 dimensional concatenation of the previous two, thus giving a combination of CBOW and skip-gram. The last word2vec model is the pre-trained Google News 300-dimensional vectors². As each model is created, the first layer, the embedding layer, uses the word_index to create the word embeddings by taking the numerical value for each term and pairing it with the corresponding word embedding from the pre-trained Word2Vec model, see Listing 4.3.

$$\begin{array}{l} 1 : [x_1 \quad x_2 \quad x_3 \quad \dots \quad x_m] \\ 2 : [x_1 \quad x_2 \quad x_3 \quad \dots \quad x_m] \\ \vdots \\ l : [x_1 \quad x_2 \quad x_3 \quad \dots \quad x_m] \end{array}$$

Listing 4.3: Matrix representation of the embedding layer where l equals the length of the word_index, and m is the embedding size.

For every model several hyper-parameters have been tested to find the model that gives the best result, or with hyper-parameters that are mentioned in the relevant literature. The loss function for every model is binary crossentropy as there is only binary classification conducted in this study. Kingma and Ba (2014) explores the differences in the most popular optimizers, stating that Adam is the most versatile, and gives overall good results on varied classification problems. The Adam optimizer will be used for all models presented in this study.

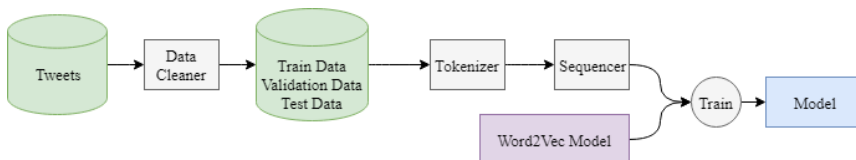


Figure 4.1: The training procedure for the neural network models.

²<https://code.google.com/archive/p/word2vec/>

FCNN

The first proposed network is a FCNN. It consists of an embedding layer, followed by a flattening layer and a FC layer. Second to last is a dropout layer to reduce overfitting, and lastly a 1 neuron FC classification layer using a sigmoid activation. This model will serve as a point of reference regarding speed, and several versions of the model will be tested with different hyper-parameters to see how it affects the performance regarding speed efficiency. A simplified version of this model can be seen in Figure 4.2.

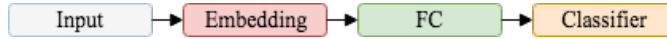


Figure 4.2: A simplified overview of the FCNN architecture.

CNN

The first proposed CNN network implements only one convolutional layer following the embedding layer. The convolutional layer is initialized with a window size of 3, and the feature maps are forwarded to a max over time pooling layer that reduce the number of activations and transforms the 2D tensor output from the convolutional layer to 1D tensor which is input to the FC layer. The FC layer is in the same manner as in the FCNN model followed by a dropout layer to reduce overfitting. The last layer is a sigmoid classifier. The full model can be seen in Figure 4.3.

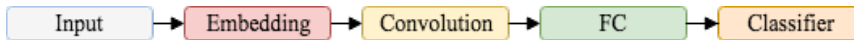


Figure 4.3: A simplified overview of the CNN architecture.

Parallel CNN

A parallel CNN model is proposed, where several convolutional layers with different filter size are concatenated before being forwarded to the FC layer. The filter sizes used in the model is 3, 4 and 5 as proposed by Kim (2014) in his non-static model. Each of the parallel convolutional layers are followed by a max-over-time pooling which reduces the dimensionality. This topology gives the network the ability to extract features in a way similar to trigrams, 4-grams and 5-grams and use the resulting concatenation of features as input to a FC layer, an addition to Kim Yoons proposed model. As with the other models, the FC layer is followed by a dropout layer and a sigmoid classifier with a l2-norm regularization with value 3.

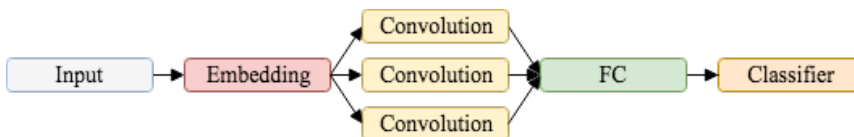


Figure 4.4: A simplified overview of the parallel CNN architecture.

CNN-RNN

The last model is proposed by Wang et al. (2016b) and uses a recurrent layer instead of a FC layer as the penultimate layer of the model. The model uses a GRU with 100 outputs, the best scoring model from the paper on binary classification. Preceding the GRU layer is two parallel convolutional layers with filter size of 4 and 5.

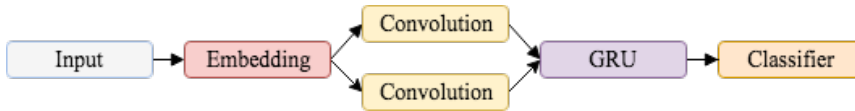


Figure 4.5: A simplified overview of the CNN-RNN architecture.

4.2.3 Hyper parameters

One of the downsides of using NNs is often their complexity, and that they are computationally expensive. The choice of hyper-parameters play an important role when the efficiency of both training and inference are concerned. A FCNN with an input size of 45, and an embedding size of 300, produces a flattened input of size 13 500. With a FC layer of size 256 the total number of weights in the network amounts to 3 456 256. Reducing the input and embedding size on the other hand will minimize the network. For comparison a FCNN with input and embedding size of 28 and 100 respectively, will have a total of 665 856. A further reduction in the size of the FC layer to 128 neurons will reduce the amount of weights to 332 928. As input and layer size increases, the FCNN struggles with an overload of parameters, thus the prediction efficiency will also drop drastically.

CNN on the other hand has a lot less parameters due to the trait of parameter sharing, but the convolution operation is on the other hand more costly. The CNN models also incorporate a FC layer at the end, and after the convolution operation the input to the FC layer is considerably smaller in size. A CNN with 100 featuremaps, a FC layer of 256, input size 45 and embedding size of 300 has a total of 116 213 weights.

In this study several variants of the FCNN and CNN will be tested to determine what parts of the NNs makes it slower or faster. To answer the question on what parts of the neural networks affect the time efficiency, the FCNN and CNN will be tested with different sizes of each layer to see how it affects the inference speed. For all experiments the input size will be 28.

4.2.4 GPU or CPU for inference?

GPUs has shown to be a major benefit to DNNs over the last year, and especially when it comes to training CNNs. Chetlur et al. (2014) reports increases in both backward and forward propagation of CNNs when using GPU rather than CPU, and an overall speedup of 30%. The test done in the study were done on networks with 5 consecutive convolutional layers, which are more that what is tested here. A test was performed on a stream of 1500 tweets per second for 180 seconds, thus a total of 2,7 million tweets on the largest CNN model presented previously. From the result shown in Figure 4.6 it is evident that the

network running on CPU is actually faster than the one running on GPU, thus the rest of the experiments will be performed on CPU as well.

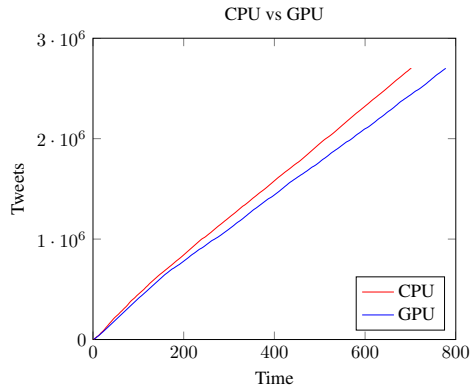


Figure 4.6: Performance of CPU and GPU on the larger CNN analyzing a total of 2.7 million tweets.

4.3 The AsterixDB Feed ingestion pipeline

To establish connections to data sources, AsterixDB has implemented feed adaptors, harboring functionality to ingest, parse and translate data from source to storable ADM records. Feed adaptors operate in two different modes, either push or pull, depending on whether the intention is to establish a connection in one request to receive data continuously, or to send a request each time one wishes to receive data respectively. As a feed is initialized, there is a possibility to specify the inclusion of a UDF, which can be written in SQL++ or defined through Java. Simpler functions, similar to those known from SQL can be defined in SQL++, while more elaborate functions will need to be implemented in Java and installed as AsterixDB libraries, then used in SQL++. As well as applying a UDF, one of several ingestion policies must be chosen, deciding how the feed adaptor will handle potential bottlenecks and failures. As the feed is initialized it is translated to a Hyracks job, and a dataflow referred to as a feed ingestion pipeline is generated. A big part of the feed ingestion pipeline is comprised of a Hyracks data operator, which is responsible for executing custom logic on partitions of data, and create partitioned output. This output is made available for the consuming operator instances after repartitioning in the Data Connectors. Along the pipeline several feed joints may be located along the pipeline and helps route the data along several simultaneous paths. Feed joints are usually placed at the output of an operator producing records, or at the output of a preprocessing compute operator.

The feed ingestion pipeline is a three stage workflow, including the intake, compute and store stages. The intake stage involves the establishment of a feed adaptor and initializing the datatransfer and transformation into ADM records. If supplied, the preprocessing functions are applied to data during the compute stage, and subsequently persisted during the store stage. Each stage is assigned a specific data operator, known as the intake, compute and store operator.

An AsterixDB cluster is made up from a manager node and several worker nodes. The Central Feed Manager (CFM) is responsible for scheduling as well as tracking load distribution across the cluster, and is hosted by the manager node. The worker nodes hosts Feed Managers (FM) that send periodic reports to the CFM with vital information such as CPU usage. For each pipeline a certain degree of parallelism must be chosen through some cardinality or location constraint. For the intake operator the constraints are regulated by the feed adaptor, while the constraints for the store is predetermined by the nodegroup associated with the target dataset. The amount of parallelism for the compute operator is on the other hand determined by the rate of data arrival and complexity of the UDF.

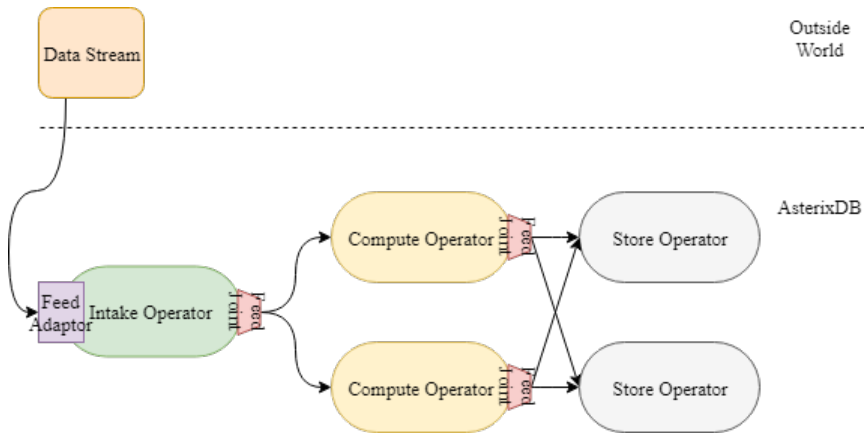


Figure 4.7: An overview of the AsterixDB feed ingestion pipeline.

As the rate of arrival increases or the UDF function gets more complex a need for excessive amounts of resources may arise, thus creating a back-pressure which in turn might lead to a complete lock of the data flow. In the case of a high rate of arrival AsterixDB tackles this problem by preventing back-pressure at the originating operator, securing the rest of the pipeline from being affected. As back-pressure evolves at an operator a MetaFeed operator, wrapped around the core operator, makes sure records are buffered in memory while also tracking the rate of arrival and rate of processing done by the core operator. Congestion caused by UDF complexity is transmitted from the MetaFeed operator, through the local FM, to the CFM which can re-structure the pipeline to accommodate the need for more processing power at the compute stage.

The MetaFeed operator is also responsible for handling software failures, and as the core operator is operating in a sandbox-like environment failures are handled by the MetaFeed operator copying the incoming data feed frame, slicing away the record causing and error, and re-feed the core operator with the augmented data frame.

Hardware failures are handled differently. Each node in the cluster periodically sends a heartbeat to the CFM telling it that it is alive, and thus the CFM can detect dead nodes. As soon as a failure is detected all nodes in the feed pipeline are notified and their FMs make sure their output buffers are saved before they terminates themselves. Intake operators spill or buffer their inputs so they are not forwarded down the pipeline. Consequently,

each operator assumes a hand-off stage, where all stored states are retrieved and sent to a new identical feed ingestion pipeline. The failed node is substituted by the CFM, which has an overview of each nodes load distribution. Store node failures are handled by an early termination of the given feed, and uses a log based recovery to restore itself and re-join the cluster

4.4 Testing the system

This section will serve as a review of how the experiments have been carried out in practice. The system will be tested on a machine with the specifications shown in Table 4.1.

AsterixDB local cluster	
Processor	3.5 GHz Intel Core i7 QuadCore
Memory	16 GB 1600 MHz DDR3
Graphics	GeForce GTX 1060 6GB

Table 4.1: AsterixDB cluster system specifications.

4.4.1 The feed generator

For testing purposes a feed generator has been implemented in Python to serve as a twitter feed, generating a specific amount of tweets per second. The generator is set to connect to a given IP address and port number, and will continuously push tweets to the socket at the given rate until a certain threshold is met. If the AsterixDB feed ingestion queue is full and can no longer receive tweets, the tweet generator will halt its stream, and continue as soon as there is room in the queue. The threshold for the tweet generator is set 180 seconds. The feed generator will be running on the machine shown in Table 4.2.

Macbook Pro Retina 2015	
Processor	2,7 GHz Intel Core i5
Memory	8 GB 1867 MHz DDR3
Graphics	Intel Iris Graphics 6100 1536 MB

Table 4.2: Feed generator system specifications.

4.4.2 Setting up the AsterixDB cluster

As an AsterixDB cluster is deployed it is comprised of a Master Node and several Worker Nodes. The master node runs a service called Cluster Controller (CC), while the worker nodes run services known as Node Controllers (NC). The CC manages the workload and distributes the work among the other nodes as well as itself.

AsterixDB supports deployment of several nodes to a single working machine, as they are logical concepts rather than individual machines running AsterixDB. In this study we will deploy 2, 4 and 8 nodes to a single machine, as the processor has 4 cores and 8 threads.

The query for starting an ingestion feed is shown in Listing 4.4. First a dataverse called feed for our incoming feed is created. As to minimize storage over several tests the feed, dataset and types are dropped before being created again. The feed is set up to listen for incoming data of datatype TextType and is connected to the dataset TweetDataset, using the getSentiment function in the compute stage of the feed pipeline. As the query is executed AsterixDB compiles the query written in SQL++ to an algebraic Hyracks executable program.

```
CREATE DATAVERSE feed
USE feed;
DROP FEED SocketFeed IF EXISTS;
DROP DATASET TweetDataset IF EXISTS;
DROP TYPE TextType IF EXISTS;
CREATE TYPE TextType AS open { text: string };
CREATE DATASET TweetDataset(TextType) PRIMARY KEY text;
CREATE FEED SocketFeed USING socket_adapter (
    ("sockets"="127.0.0.1:10001"),
    ("address-type"="IP"),
    ("type-name"="TextType"),
    ("format"="adm")
);
CONNECT FEED SocketFeed TO DATASET TweetDataset
APPLY FUNCTION classlib#getSentiment USING POLICY Basic;
START FEED SocketFeed;
```

Listing 4.4: The query for initializing the feed in AsterixDB.

As the tweets arrive in the feed they will be processed by the installed getSentiment function applied to the stream in the AsterixDB query shown above. As a tweet is ingested into the function, it will be tokenized by a method called textToSequence which processes the tweet in the same way as done in training. Each tweet will subsequently be evaluated by the NN, and an inference will be computed by the network. The output of the getSentiment function is an ADM shown in Listing 4.5.

```
input = {text: "this is a tweet text"}

output = {text: "this is a tweet text",
          sentiment: "positive"}
```

Listing 4.5: The dataformat of the tweets fed into the UDF and the output from the UDF stored in AsterixDB.

Results

In this chapter the results from training and running the twitter stream into the AsterixDB feed ingestion pipeline are presented. Every test was done 5 times and the average is displayed in the following graphs.

5.1 Training results

The results from Go et al. (2009) are used as a reference point to see how well the models are performing. Sentiment140 was tested with several combinations of features and with three different algorithms. The algorithms tested was NB, Max Entropy and SVM. The results are shown in Figure 5.1

Features	NB	MaxEnt	SVM
Unigram	81,3%	80,5%	82,2%
Unigram + Bigram	82,7%	83,0%	81,6%

Table 5.1: Results from the Sentiment140 project.

As the training set is huge in size, the models presented in the previous chapter was trained for a total of 10 epochs each. A callback method keeps track of the validation accuracy for each epoch and saves the best performing model to file. Each network was trained with the 4 different Word2Vec embedding explained earlier, and the 200 dimensional concatenation of CBOW and Skip-gram vectors outperformed the 100 dimensional vectors of both CBOW and skip-gram trained vectors, as well as the 300 dimensional Google News Vectors.

For the FCNN and the CNN several models were trained with different hyperparameters getting different result. The FCNN with 64 neurons in the hidden layer got a test accuracy of 81,44%, while with 128 and 256, the accuracy increased to 81,64% and 82,00% respectively. For the CNN a combination of 100 convolution filters and a FC layer with 64 neurons were tested, producing a test accuracy of 83,22%, while the model with 200

filters and a 128 neuron FC layer scored a test accuracy of 83,27%.

The hyperparameters for the parallel CNN and the CNN-RNN were kept as close to the original studies as possible, except for the addition of a penultimate FC layer with 64 neurons in the parallel CNN which increased test accuracy from 83,05% to 83,71%. The CNN-RNN proposed by Wang et al. (2016b) had less information about the hyperparameters applied to the model, but both the LSTM and GRU model were tested with 100 convolution filters and 128 as the output size from the LSTM and GRU layer. Just as in the study, the GRU implementation acquired the best test accuracy of 84,02%, the best scoring model in this study. An overview of the accuracy scores can be seen in Table 5.2.

Model	Accuracy
Small FCNN	81,44%
Large FCNN	82,00%
Small CNN	83,22%
Large CNN	83,27%
Parallel CNN	83,71%
CNN-RNN	84,02%

Table 5.2: CNN model summary.

5.2 Streaming results

In Figure 5.1-5.6 the results from running the query on the various models are presented. For each model some initial test were done to find a reasonable amount of tweets per second to push from the stream generator. Each model has been tested 5 times on each cluster size, and the results shown are the averages. Lastly Table 5.3 shown an overview of each models total tweets classified, the finish time of each model and the increase in tweets per second when increasing the cluster size to either 4 or 8 nodes.

Model	Total Tweets	Finish Time			TPS increase
		2 nodes	4 nodes	8 nodes	
small FCNN	2 700 000	280s	210s	204s	37,25%
large FCNN	900 000	338s	314s	318s	7,62%
small CNN	1 800 000	322s	256s	210s	53,33%
large CNN	1 260 000	332s	258s	252s	31,75%
Parallel CNN	720 000	374s	262s	300s	42,75%
CNN-RNN	360 000	418s	304s	242s	72,82%

Table 5.3: Data acquired from the AsterixDB feeds.

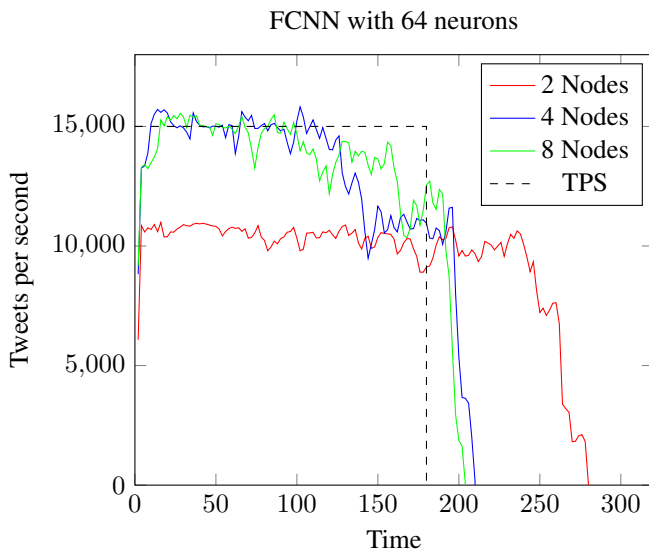


Figure 5.1: Results from the FCNN with 64 neurons in the hidden layer.

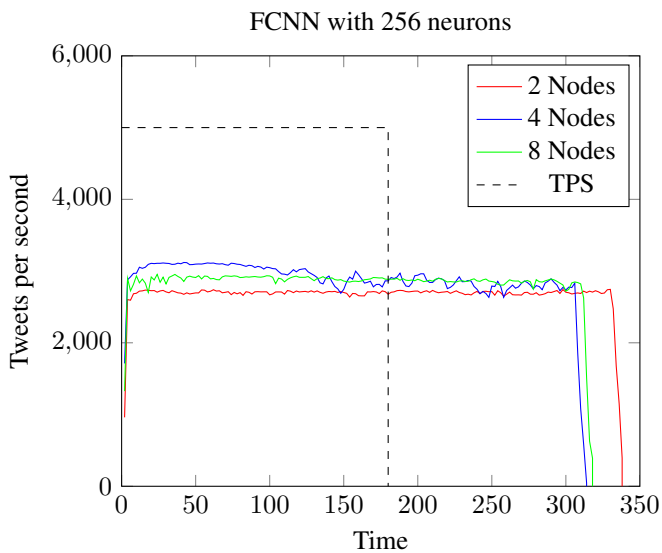


Figure 5.2: Results from the FCNN with 256 neurons in the hidden layer.

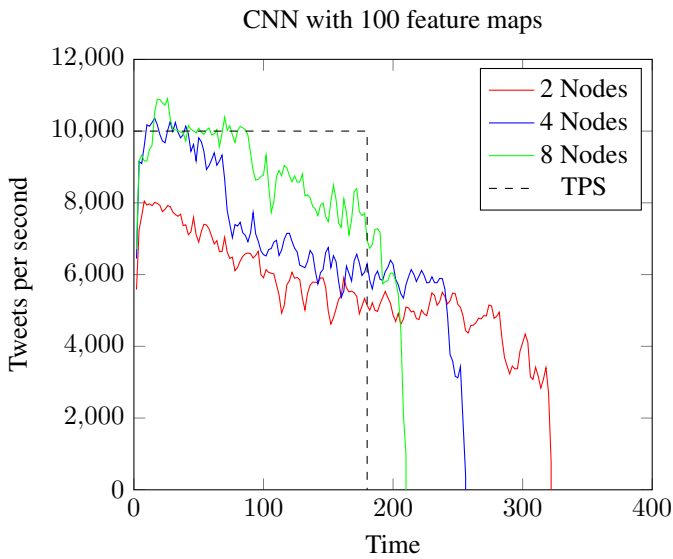


Figure 5.3: Results from the CNN with 100 feature maps and a FC layer with 64 neurons.

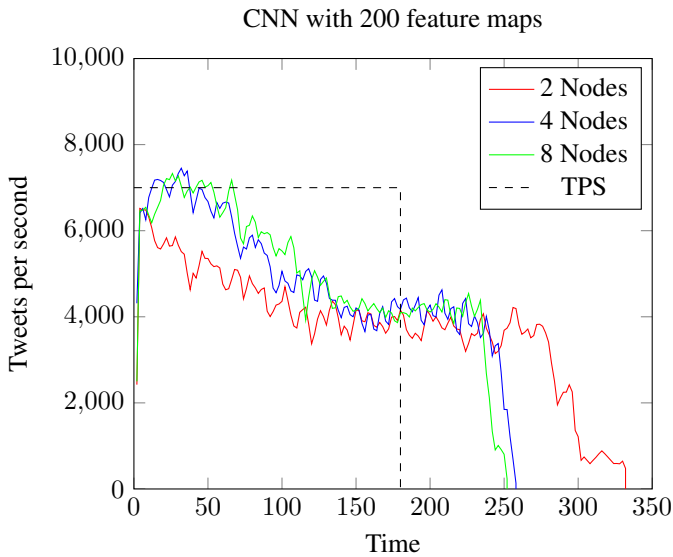


Figure 5.4: Results from the CNN with 200 feature maps and a FC layer with 128 neurons.

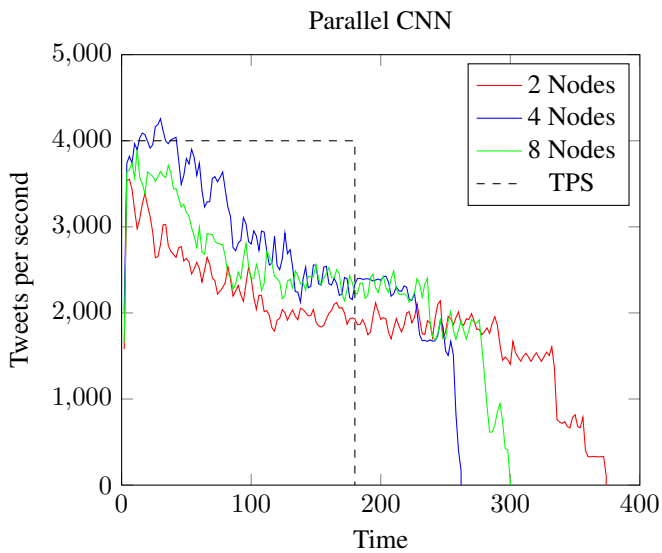


Figure 5.5: Results from the parallel CNN network.

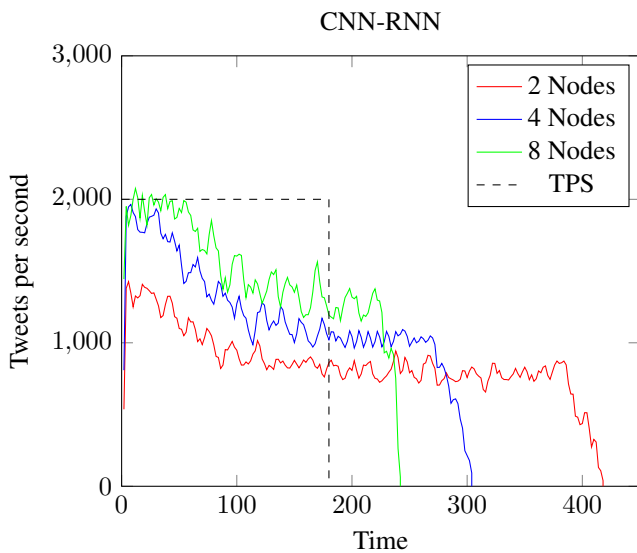


Figure 5.6: Results from the model combining convolutional and recurrent layers.

Evaluation and Discussion

In this chapter an evaluation and discussion around the previous results will be presented.

6.1 Evaluation

The accuracy measures presented in the last chapter clearly outperforms the results provided in the literature from Sentiment140. The CNN-RNN presented in this study increases the best gained accuracy from the Max Entropy classification algorithm by 1,02%, a significant increase. Both the CNN and parallel CNN outperformed the best of the traditional algorithms as well.

Interestingly the larger FCNN has significantly better accuracy than the smaller FCNN, with an increase of 0,56%. Meanwhile the larger CNN barely increases the accuracy from the smaller, presenting an increase of only 0,05%.

Each model has been tested on a simulation of a cluster with 2, 4 and 8 nodes, as to give an estimation as to whether these models will scale or not. For the smaller FCNN the time it took to process the entirety of the stream was reduced from 280 to 204 seconds, an improvement in tweets per second from 9643 to 13 235, or roughly 37,25%. Meanwhile the larger FCNN shows almost no improvement in terms of speed by increasing the amount of nodes, increasing the TPS by only 7,62%. The larger FCNN also had a marginally larger increase in TPS when using 4 nodes than 8.

Both the small and large CNNs show improvement from adding more nodes to the virtual cluster, and increase their tweets per second from 5590 to 8571 for the smaller CNN and from 3795 to 5000 respectively. This gives an overall increase in TPS by 53,33% for the small model and 31,75% for the larger.

Likewise there is a similar increase on the parallel CNN increasing the tweets per second from 1925 to 2748, an increase of 42,75% when increasing the cluster to 4 nodes. The 8 node cluster on the other hand showed a smaller improvement of only 24,68%. Meanwhile, the CNN-RNN model is the model showing the best improvement from increasing the cluster size, starting at 861 TPS on a 2 node cluster, and ending on 1488 TPS on the 8 node cluster, posing an increase in TPS of 72,82%.

By evaluating the statistics generated from running the different networks, several models are showing clear signs of scalability. A closer inspection of the statistics of each NC, shows clearly that the CC evenly distributes the workload among the NCs, independent on the cluster size.

6.2 Discussion

This study had as a goal to see how well different NNs performed on the classification task of sentiment analysis compared to traditional methods. By taking full advantage of the powerful word embedding, results outperforming all of the traditional classification methods were achieved by some of the networks. The lightest networks struggled with getting a high accuracy, but displayed competitiveness towards the simplest classification algorithms such as NB.

Furthermore the goal was to achieve a competitive accuracy in response to what have been achieved with the traditional algorithms, and there is probably more work that can be applied to reaching even higher accuracy with the NNs. There are endless combinations of hyperparameters that can be tested, and since the dataset was big, the only regularization applied was dropout layers. Different variations of regularization, such as batch normalization and constraints on the layers might prove to generalize the models more, reduce overfitting and result in even higher accuracy measures.

The CNN shows better improvement than the FCNN when creating slightly more complex models with added feature layers and neurons in the hidden layers. The convolution and pooling layers used in succession reduce the dimensionality of the data propagating through the network, and thus creates network capable of managing a higher throughput.

An interesting finding among the results is how the larger FCNN with a hidden layer size of 256 neurons barely improves when increasing the cluster size, as shown in Figure 5.2. Every other model has a significant increase in TPS. This might have to do with the amount of weights, and consequently the amount of floating point operations taking place. Inspecting the amount of processing power used while running the stream, the larger FCNN struggled with utilizing the full potential of the CPU regardless of the cluster size. As for the other networks, the CPU usage was usually recorded at 45-75% on the 2 node cluster, and increased to 100% utilization on the 4 and 8 node clusters.

Every network apart from the larger FCNN follows a similar curvature of the TPS over time. Each network is initially able to process a reasonably high amount of TPS, which is also increased by the cluster size for most models. Over time each model experiences a decrease in the TPS it is capable of processing, eventually stagnating at a certain amount.

Conclusion and Future Work

This chapter will conclude the presented study, and present potential future work.

7.1 Conclusion

The aim of the study was to establish whether or not a number of different NN models could be competitive with traditional classification algorithms on the task of detecting sentiment within twitter messages. The study proposed 6 different neural networks. The first two layers were two FCNN with 64 and 256 neurons in the hidden layer respectively. Additionally two CNN networks were proposed. The first CNN included a convolutional layer of 100 feature maps followed by a FC layer with 64 neurons, and the second 200 feature maps and a FC layer of size 128. The penultimate model was inspired by one of the articles presented in the related research section in chapter 3, and introduces three parallel convolutional layers extracting features in a similar way as n-grams does in traditional computational linguistics. The last model proposed is also inspired by the previous research, and incorporates parallel convolutional layers, followed by a consecutive GRU layer. The models were compared to models proposed by a study from Stanford University in terms of accuracy, and 4 out of 6 models had improved accuracy. This confirms that NN are indeed competitive in the area of sentiment analysis. The two FCNN models were the only models not increasing accuracy over any of the models proposed by the previous study.

The second part of the study set out to explore how well the different models performed on streaming data. Several great stream processing engines exist, but AsterixDB is still the only framework providing stream handling as well as persistent storage, and thus, AsterixDB was chosen for this project. AsterixDB supports UDFs which lets the user define functions written in Java, that are applied to the streaming data. The results, shown in Figure 5.1-5.6, proved that several of the models are sustainable for larger amounts of data. Further testing the networks on local clusters of different sizes show that most of the models are scalable as well.

7.2 Future Work

The natural next step for this kind of experiment is to test the models on an actual cluster and not a simulated one running on a single machine. As the machine running the cluster reached its maximum processing capability reasonably fast as the cluster size increased, the results from the larger cluster are not necessarily representable for a real cluster.

Bibliography

- Apache Software Foundation, 2018. Apache software foundation.
URL <https://www.apache.org/>
- Bengio, Y., Simard, P., Frasconi, P., Mar. 1994. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.* 5 (2), 157–166.
URL <http://dx.doi.org/10.1109/72.279181>
- Braun, S., Jun. 2018. LSTM Benchmarks for Deep Learning Frameworks. ArXiv e-prints.
- Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., Shelhamer, E., 2014. cudnn: Efficient primitives for deep learning. arXiv preprint arXiv:1410.0759.
- Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y., Jun. 2014. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. ArXiv e-prints.
- Chollet, F., 2017. Deep Learning with Python, 1st Edition. Manning Publications Co., Greenwich, CT, USA.
- dos Santos, C., Gatti, M., August 2014. Deep convolutional neural networks for sentiment analysis of short texts. In: *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*. Dublin City University and Association for Computational Linguistics, Dublin, Ireland, pp. 69–78.
URL <http://www.aclweb.org/anthology/C14-1008>
- Go, A., Bhayani, R., Huang, L., 2009. Twitter sentiment classification using distant supervision. CS224N Project Report, Stanford 1 (12).
- Grover, R., Carey, M. J., 2015. Data ingestion in asterixdb. In: *EDBT*. pp. 605–616.
- Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. *Neural computation* 9 (8), 1735–1780.
- Inc., T., 2017. Selected company metrics and financials.
URL <http://files.shareholder.com/downloads/AMDA-2F526X/>

5678890017x0x961126/1C3B5760-08BC-4637-ABA1-A9423C80F1F4/
Q317_Selected_Company_Metrics_and_Financials.pdf

- Kim, Y., 2014. Convolutional neural networks for sentence classification. CoRR abs/1408.5882.
URL <http://arxiv.org/abs/1408.5882>
- Kingma, D. P., Ba, J., Dec. 2014. Adam: A Method for Stochastic Optimization. ArXiv e-prints.
- Kovalev, V., Kalinovsky, A., Kovalev, S., 2016. Deep learning with theano, torch, caffe, tensorflow, and deeplearning4j: Which one is the best in speed and accuracy?
- Kulkarni, S., Bhagat, N., Fu, M., Kedigehalli, V., Kellogg, C., Mittal, S., Patel, J. M., Ramasamy, K., Taneja, S., 2015. Twitter heron: Stream processing at scale. In: Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data. ACM, pp. 239–250.
- LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., November 1998. Gradient-based learning applied to document recognition. Proceedings of the IEEE 86 (11), 2278–2324.
- Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. CoRR abs/1301.3781.
URL <http://arxiv.org/abs/1301.3781>
- Ouyang, X., Zhou, P., Li, C. H., Liu, L., Oct 2015. Sentiment analysis using convolutional neural network. In: 2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing. pp. 2359–2364.
- Springenberg, J. T., Dosovitskiy, A., Brox, T., Riedmiller, M., Dec. 2014. Striving for Simplicity: The All Convolutional Net. ArXiv e-prints.
- Toshniwal, A., Taneja, S., Shukla, A., Ramasamy, K., Patel, J. M., Kulkarni, S., Jackson, J., Gade, K., Fu, M., Donham, J., Bhagat, N., Mittal, S., Ryaboy, D., 2014. Storm@twitter. In: Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data. SIGMOD '14. ACM, New York, NY, USA, pp. 147–156.
URL <http://doi.acm.org/10.1145/2588555.2595641>
- Wang, J., Yu, L.-C., Lai, K. R., Zhang, X., 2016a. Dimensional sentiment analysis using a regional cnn-lstm model. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). Vol. 2. pp. 225–230.
- Wang, X., Jiang, W., Luo, Z., 2016b. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. In: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers. pp. 2428–2437.