Capsule Neural Networks for Text Classification

Akmal Khikmatullaev

Matriculation number: 2754116

April 15, 2019

Master Thesis

Computer Science

Supervisors:

Prof. Dr. Jens Lehmann
Dr. Kuldeep Singh

Institut für Informatik III

Rheinische Friedrich-Wilhelms-Universität Bonn

Declaration of Authorship

I, Akmal Khikmatullaev, declare that this thesis, titled "Capsule Neural Networks for Text Classification", and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at University of Bonn.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. Except for such quotations, this thesis is entirely my own work. I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Acknowledgements

I would like first to thank Prof. Jens Lehmann and Dr. Steffen Lohmann for allowing me the opportunity to write my master thesis at the Computer Science department in the University of Bonn.

I would like to express my sincere gratitude to my supervisor Mr. Kuldeep Singh who guided me all the way even from before starting my thesis, helped me to understand the task, speculated about different ways of solutions and supported me during the doing master thesis.

Finally, I must express my very profound gratitude to my parents for providing me with unfailing support and continuous encouragement throughout my years of study, gave me the opportunity to study in University of Bonn and encourage through the process of researching and writing this thesis. This accomplishment would not have been possible without them. Thank you a lot.

Akmal Khikmatullaev

Contents

List of Figures

List of Tables

Abstract

In the contemporary era, a large amount of unstructured data is produced, stored in digital form for further analysis and processing. This volume of data is so large that it is impossible to identify useful information in a manual manner. Text classification problem plays an especially important role in structuring upcoming information. In order to solve this issue, machine learning approaches from simple statistical models to different types of neural networks are used for automation.

Capsule Neural Network is an absolutely new type of neural networks which was introduced by Geoffrey Hinton for image recognition task. The general idea is to imitate mimic biological neural organization by using "capsule" which is a group of neurons representing parameters of an object or an object part. Moreover, Geoffrey Hinton introduced Dynamic Routing and EM Routing as algorithms which allow Capsule Neural Network to forward between low-level and high-level capsules.

This thesis presents an empirical exploration of using Capsule Neural Network with Dynamic Routing and Capsule Neural Network with EM Routing for text classification problem. Seven different datasets were used for getting the accuracy and the loss of neural networks. Also, pre-trained word vectors such as glove, word2vec, and fasttext are used in order to prepare all datasets for feeding to Capsule Neural Network. Finally, results were obtained in all datasets and discussed.

Keywords: Text Classification, Deep Learning, Neural Networks

Chapter 1

Introduction

Text classification forms an essential part of many Natural Language Processing applications including Sentiment Analysis, Question Answering, and Text Categorisation and Clustering, etc. Over the years, several approaches have been employed to achieve text classification ranging from the use of classification rules to machine learning approaches. Application of rules is an intuitive approach that formed part of the early attempts to text classification ([12, 13]). Rule-based approaches classify text by either using a set of handcrafted linguistic rules [12] or rules that have been automatically learned via rule induction [14].Per contra, both probabilistic and statistical Machine Learning approaches have been shown to perform well on the task of text classification. Some of the common machine learning approaches to text classification include Naive Bayesian[15], Logistic Regression Classifiers. Support Vector Machines (SVM)[16] and Neural Networks Based Models. In general, machine learning approaches heavily rely on the quality of feature engineering and the availability of training data.

Recent advances in Deep Learning has seen neural network based models became state of the art approaches for many NLP tasks including text classification. Neural network models started being employed as Multilayer Perceptron(MLP), which gradually transformed into current more parametric approaches in the form of Convolutional Neural Networks (CNNs), Recurrent Neural Networks, and attention based models. It is common practice in research, that Neural Network techniques which have seen successful in other domains such as image analysis and computer vision have also been utilized for text processing. For instance, the CNN family of algorithms were originally deployed for image processing and computer vision[17] until recently when researchers have experimented with these architectures for text classification as seen in[1]. The Capsule Neural Network(CapsNet)[7, 8] is a recent architecture of this family of algorithms that seeks to alleviate some

of the challenges of the CNN by introducing the concept of capsules in which several convolutional layers are stacked up together in a single layer to form a capsule that output a vector-based representation of the state of an image. Such vector representation is produced by routing the agreement method which is superior to the current mechanism like max-pooling that has been employed in CNNs. As a result, the capsule networks are able to provide a better understanding of finer grained objects in an image.

In this thesis, we experiment with the CapsNet in a new domain for text classification from the original application in computer vision. Early research in using this approach for text classification have been just reported [18, 19] and indicates that this approach has potential to improve the state of the art in this field. Our evaluations, presented in this thesis, demonstrate that capsule networks are a viable architecture to improve the performance of text classification in various NLP tasks.

1.1 Motivation

Unstructured data anywhere: tweets, chats, emails, web pages, etc. For that reason, the central task today is text classification which plays for business a crucial role. Text classification finds massive application in several areas. In this thesis, we motivate our work in two areas namely sentiment analysis and question answering. In the following sections, we describe each area and how CapsNet is able to use.

Sentiment Analysis

Perhaps, sentiment analysis is the most significant part of text classification: the process of automatically determining a text is positive, negative or neutral. Various machine learning models have been used for sentiment analysis from naive bayes to a different type of neural networks such as CNN.

Originally, CNN was built as a check-recognition system to read handwritten digits[20]. However, later CNN performed well in different tasks in computer vision such as object detection, object tracking, object recognition and with RNN in video and image captioning.

Nonetheless, this paper[1] demonstrates the way, how CNN is able to be used for text classification. Figure (1.1) shows the architecture of CNN.

The general idea of using CNN in sentence classification is a word vector presentation of the text by using internal embedding or using pre-trained word vectors for each of the words. In this case, the prepossessing is a significant part, since the choice of embedding affects the final performance

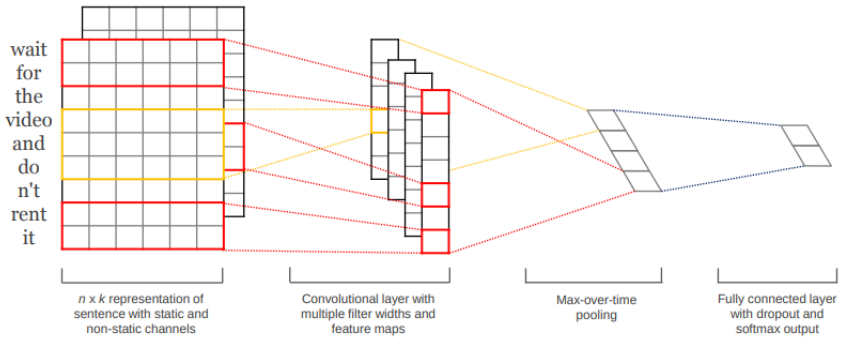


Figure 1.1: CNN model architecture with two channels[1]

of CNN.

However, Geoffrey Hinton in 2017 has introduced Capsule Neural Network[7] and demonstrated that it is significantly better than CNNs in Image Recognition task. Thus, one of the motivations of this thesis is to use CapsNet for text classification and determine is it better than CNN or not.

Question Answering

Singh et al. [2, 21, 22] has introduced the concept of dynamic composition of QA systems. The concept of dynamic composition of QA pipelines adduces that the combination of components to process a given question should be determined during runtime depending on the nature of the question.

In collaborative development of QA systems, several tools exist that perform similar tasks within the pipeline but each of these tools varies in the method by which it handles different kinds of questions. Within the QA process, there are five major tasks to be performed. At every step, the most suitable component is selected for the given task. It follows from this that analysis and typification of the question are critical in determining which pipeline should be used in processing.

For frameworks such as Frankenstein, that aim for dynamic composition of QA pipelines, the question understanding or classification module is necessary to achieve an appropriate combination of existing components into a suitable pipeline able to deduce the best answer.

To motivate our work, let us consider two natural language questions as follows: i) *"What is the capital of Ireland?"* and ii) *"Name the municipality of Roberto Clemente Bridge?"*. Figure (1.2) shows how the pipelines are selected for answering these two different questions within the Frankenstein Framework. To achieve appropriate characterization of questions in order to

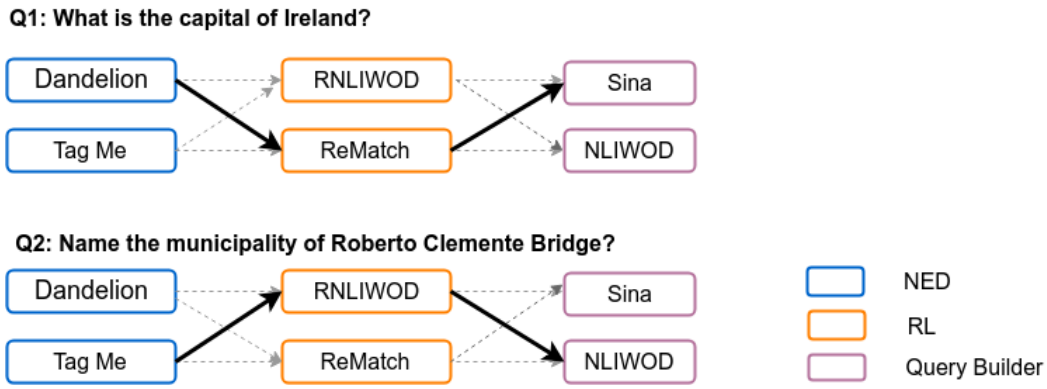


Figure 1.2: Two sample NL questions answered successfully by different pipelines formed of 2 NEDs, 2 RLs, and 2 QBs components. The most suitable pipeline for each question is highlighted in bold Arrows. Questions obtained from Frankenstein[2]

decide what chain of components is suitable for the given question, Singh et al. [2] elaborate a set of syntactical features to be extracted from the natural language questions.

For Instance, for our example questions, the following features will be extracted:

Features	**Questions Analysis**	
	What is the capital of Ireland?	*Name the municipality of Roberto Clemente Bridge?*
Question word	What	Descriptive
Question length	7	8
Answer Type	String	String
POS	(#DT:1), (#IN:1), ...	(#VB:1), (#DT:1), ...

For the *"What is the capital of Ireland?"* the question type is *"What"*, for the *"Name the municipality of Roberto Clemente Bridge?"* the question type is *"Descriptive"*. This is the question classification task which is a subset of text classification. In order to automate the process of determining a question work trained CapsNet can be used for improving the performance of the QA system.

1.2 Problem Statement

The mathematical form of text classification can be defined as follows:

$\mathcal{D}$ – a domain of documents

$\mathcal{C} = \{c_1, c_2, \dots c_{|\mathcal{C}|}\}$ – a set of pre-defined *categories*

$\forall d \in \mathcal{D}$ assign a Boolean value $b_i \in \{0, 1\}, \forall \langle d, c_i \rangle \in \mathcal{D} \times \mathcal{C} \Leftrightarrow$

$$\begin{aligned} \langle d, c_1 \rangle &= b_1 \\ \langle d, c_2 \rangle &= b_2 \\ &\dots \\ \langle d, c_{|\mathcal{C}|} \rangle &= b_{|\mathcal{C}|} \end{aligned} \tag{1.1}$$

More formally, this is the task of a ***target function approximation***:

$$\Phi : \mathcal{D} \times \mathcal{C} \to \{0, 1\} \tag{1.2}$$

Φ – is called the *classifier* or *rule*, or *hypothesis*, or *model*.

A variety of conditions is able to apply, depending on an application requirement. For example, for a given integer number n elements of $\mathcal{C}$ have to be assigned to each element of $\mathcal{D}$.

Multi-label case: $\forall$ n $\geq 2 \Leftrightarrow$

$$\begin{aligned} \langle d, c_1 \rangle &= b_1 \\ \langle d, c_2 \rangle &= b_2 \\ &\dots \qquad \exists b_{i_1}, b_{i_2}, \dots b_{i_n} = 1, others = 0 \\ \langle d, c_{|\mathcal{C}|} \rangle &= b_{|\mathcal{C}|} \end{aligned}$$

Single-label case: $n = 1 \Leftrightarrow$

$$\begin{aligned} \langle d, c_1 \rangle &= b_1 \\ \langle d, c_2 \rangle &= b_2 \\ &\dots \qquad \exists i : b_i = 1 \vee b_j = 0, \forall j \neq i \\ \langle d, c_{|\mathcal{C}|} \rangle &= b_{|\mathcal{C}|} \end{aligned}$$

Moreover, if $|\mathcal{C}| = 2$, this is a ***binary text classification task***.

In this thesis, a single-label case has been considered. The purpose of this work is to use two types of CapsNets for the central problem(text classification) in Natural Language Processing(NLP). Originally, CapsNet was introduced for image recognition task, however, from the successful use of CNN in NLP it can be concluded that any neural network can solve problems in NLP.

1.3 Objectives

CapsNet is an absolutely new type of neural networks which is better in modeling hierarchical relationships and significantly imitates more the biology of natural neural networks. The crucial part of CapsNet is a *capsule* which is a group of neurons whose activity vector represents parameters of an object or an object part. Capsules save together all significant information about the state of the feature and store as a vector.

The purpose of this work is to use CapsNet for an approximation of (1.2) function. Since originally CapsNet was introduced for computer vision area, the task is to modify the original architecture of CapsNet and/or represent sentences as an image(used pre-trained word vectors) for text classification problem.

1.4 Thesis Structure

Chapter 1 Introduction

This chapter brings the overview of text classification, describes the motivation and defines in mathematically way the problem. Furthermore, notices the model that will be used and the contributions. The outline of this work is presented.

Chapter 2 Background

This chapter provides a background developing Neural Networks from a historical view. Also, describes "Supervised Learning" and the most famous algorithm in machine learning - "Back-Propagation Algorithm" and illustrates in details Recurrent Neural Network, Convolution Neural Network and two different types of Capsule Neural Networks.

Chapter 3 Related Work

This chapter covers the various machine learning models that have been applied for text classification tasks such as Naive Bayes, K-Nearest Neighbors, Decision Tree and Support Vector Machine.

Chapter 4 Approach

In this chapter outline data and preprocessing, modification of Capsule Neural Networks. Seven different datasets which were used for experiments and tree type of pre-trained vectors such as glove, fasttext, and word2vec.

Chapter 5 Implementation

This chapter illustrates in details the implementation Capsule Neural Network with Dynamic Routing in Keras library and Capsule Neural Network with EM Routing in PyTorch library.

Chapter 6 Evaluation

The chapter describes the environment setup for the experiments, demonstrates the performance(the accuracy and the loss graphs) of Capsule Neural Network with Dynamic Routing and Capsule Neural Network with EM Routing for each dataset.

Chapter 7 Conclusion

In this chapter, the thesis is summarized the performance of models and the future work is discussed.

Chapter 2

Background

The human brain has the unique ability to analyze, learn and make conclusions based on experience. From a biological point of view, millions of neuron folds connected in a complicated way allow learning. The brain is extremely powerful and faster than any computer. A lot of neuron layers are communicated with each other thanks to electrical signals that are passed through neurons and, importantly, this happens in parallel.

The human brain has the ability to solve a variety of complex problems like face detection, motion, mathematical calculations, etc. For the past 50 years, computer scientists have been trying to simulate and imitate the architecture of brain neurons(simplified model) in order to automate processes and tasks that require a human presence.

Artificial **N**eural **N**etworks (ANN) is one of the attempts to imitate the human brain and thanks to many biologists, mathematicians and computer scientists successfully cope with problems of computer vision, recommendation system, natural language processing, etc.

In the following sections, we take into account the history of ANN: an inspiration and biological background, remarkable stages of developing and various attempts of making models. Further, outline the Supervised Learning and Back-Propagation Algorithm that lets training ANNs. A general schema is being described as examples of ANN, the principle of working, having an analysis in detail Recurrent Neural Network(GRU and LSTM), Convolutional Neural Network and Capsule Neural Network.

2.1 History of ANN

A *neuron* is an information processing unit that is fundamental to the operation of neural networks. A neurophysiologist Warren Sturgis McCulloch

together with a mathematician Walter Pitts proposed the first mathematical model of a neuron that imitate the biological neuron. In 1943[23] authors published a paper where they described the principle of a neuron.

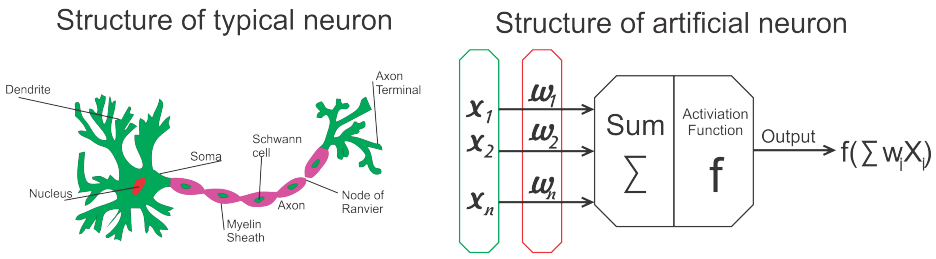


Figure 2.1: Biological Neuron vs Artificial Neuron

Figure (2.1) demonstrates their perception of a neuron compared to a real biological illustration of a biological neuron.

Getting electro-chemical inputs from other connected neurons at the dendrites is the major rule of the biological work and then sum the inputs, the neuron is activated if the sum is notable or exceed a definite threshold, and sends or transfers an electrochemical signal along the axon to other neurons[23].

Contrary to what the biological neuron does, the McCulloch-Pitts vision summarises the multiplications of inputs with internal *weights* and a new signal would be sent to other connected neurons as an output[24] with a condition the sum is more than specific value or threshold. In the original version of the model, $inputs \in (0, 1)$ and $weights \in (-1, +1)$ which correspond to the strength of the connection.

McCulloch and Pitts established that this structure could construct the boolean *"NOT"* and *"AND"* functions which allows to get *"OR"*. Hence, any boolean function can be expressed by using these 3 functions(disjunctive or conjunctive normal form).

In 1949 Donald Hebb offered the Hebb's rule of learning[25], which states that the connection between two neurons is bent to change, It means that, if there are more signals between neurons, connections between them will be stronger. According to Hebb's rule, the concept of weight w_{ij} attributes to the weight between neuron i and j.

Then, the importance of adapting weights was realized by mathematics and computer scientist, the ability for each neuron to adjust weights in a certain way to perform the target output. It was an inspiration for Window and Hoff to design a learning procedure that gives to opportunity to adapt

the weight of a neuron. While working together they created the Windrow-Hoff learning rule, which was based on distributing the error of some random neuron to the whole network.

The Winrow-Hoff rule was a foundation for the famous learning algorithm "Back-propagation of Error" giving an opportunity to achieve a local and/or global minimum of a function which was based on gradient descent. Finally, the Widrow-Hoff learning rule was advanced to multiple layers ANN by a group of researches in 1986 and the most famous "Back-propagation of Error" method in machine learning was published.

In our modern world, computer scientists do research in the field of neural network models and learning technique, nevertheless, the most important difficulty facing neural networks which can be met is the hardware efficiency, because of the processing power which must have tradings in deep structures(days, weeks).

2.2 Supervised Learning and Back-Propagation Algorithm

Supervised learning is one of the three fundamental types of learning that are used in machine learning, along with unsupervised learning and reinforcement learning.

The central unit of any neural networks is the *"neuron"* with an internal function that sums inputs multiplying by weights value w_{ij}. Within each neuron, the *"bias"* is included with default value 1 and was introduced by Bernard Widrow[26] to shift the output of the neuron's activation function to be around zero and get rid of the threshold parameter.

The Hebb's rule was the first attempt to adjust weights and add the concept of an activation function announced non-linearity proving to be good because in practice there is very little data that depends linearly.

The input of an activation function is a linear combination of inputs with weights, as follow:

$$z = \varphi(w_0 b + w_1 x_1 + w_2 x_2 + \cdots + w_n x_n) \tag{2.1}$$

where: z: the output, $\varphi(.)$: the activation function, n: the total number of inputs, b: the bias value, w_i: the weight value, x_i: the input value.

The following functions are able to use as an activation function:

- $ReLU(z) = max(0, z)$
- $Tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$

- $Sigmoid(z) = \frac{1}{1+e^{-z}}$

A feed-forward network is a batch of connected to each other layers(a collection of neurons); each layer has its neurons connected to all other neurons in the next layer. All these connections are weighted via a set of weights to tune the strength of the connections.

Usually, feed-forward networks have three types of layers:

- input: a buffer for the input values and the bias;
- output: produce the final value of the network;
- hidden: in between the input and output.

Supervised learning[27] is the method that allows imitating learning with a *"teacher"*. In neural networks, it is a process of attempting to approximate the same output that a teacher gives, the performance of it can be evaluated by comparing the output of the system with the correct output of the teacher. This comparison method is performed via an *"error/loss"* function.

As an error function **M**ean **S**quared **E**rror (MSE) and **M**ean **A**bsolute **E**rror (MAE) is able to use are able be used:

$$MSE_p = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{2.2a}$$

$$MAE_p = \frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i| \tag{2.2b}$$

Where n is the input dimension, Y_i is the predicted output, and $\hat{Y}_i$ is the correct teacher output. Indeed, these equations calculate the difference between neural network predicted value with teacher value only for one sample, however usually in practice, there can be billions of data points. Therefore an objective function(global error) is defined as follows:

$$G_{MSE} = \sum_{i=1}^{p} MSE_p \tag{2.3a}$$

$$G_{MAE} = \sum_{i=1}^{p} MAE_p \tag{2.3b}$$

Where p - the data points number.

Gradient descent is an iterative algorithm that allows finding a local/-global minimum of an arbitrary function by using a function's gradient descent. From the mathematical analysis, it is known that the gradient of the function is directed towards the growth of the function and based on it if we use the opposite direction of the gradient, we can approach the minimum iteratively. This is the general idea of a gradient descent algorithm.

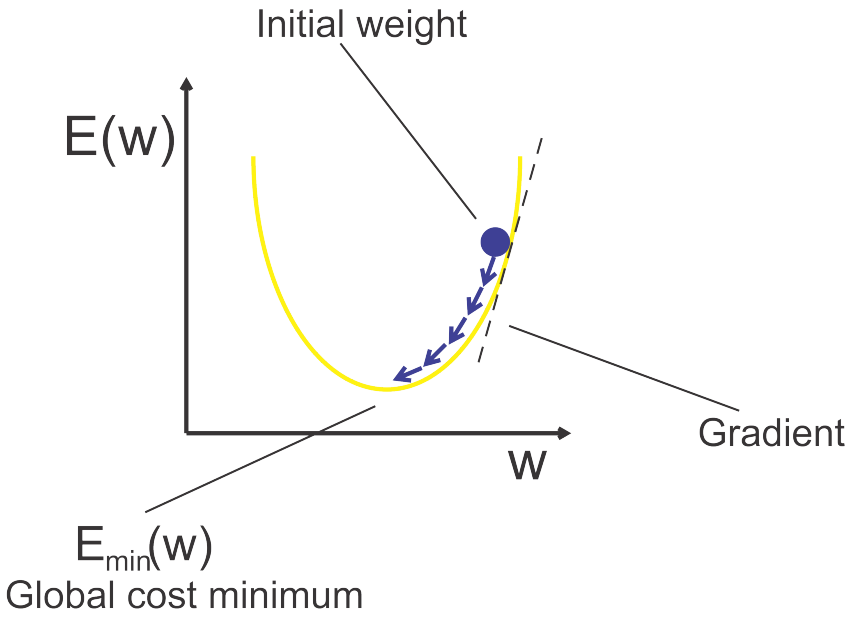


Figure 2.2: Gradient descent

Figure (2.2) demonstrates how the tune of the weights w_i with the gradient minimizes the error value to achieve the global minimum and with that the learning process of the neurons is complete. The gradient is denoted as the symbol ∇.

Back-Propagation Algorithm is an implementation of a gradient descent algorithm in neural networks and it was introduced in 1986 by a group of researchers by extending the Hebbian learning rule for multiple layer neural networks[28].

Algorithm (2.1) demonstrates all steps of the back-propagation algorithm that adjusts weights with respect to the error value calculated with the loss function of a neural network.

The algorithm consists of three major parts, which indicates the same procedure explained via equation (2.1) to calculate the output of the neural

Algorithm 2.1 Back-Propagation of error

1: Weights initialization
2: Pick a pattern i, $(X^i, \hat{Y^i})$
3: Forward phase through the network, perform the prediction Y^i
4: Compare the prediction Y^i to the target $\hat{Y^i}$
5: Backward phase through the network
6: Compute δ_o at the output layer, and all Δw_{ho}
7: **repeat**
8: Compute δ_h at the hidden layer, and all Δw_{kh}
9: **until** input layer is faced
10: Update weights: $w_{ij} = w_{ij} + \Delta w_{ij}$
11: **if** Exit Condition **then**
12: Stop
13: **else**
14: Go to the step 2
15: **end if**

network.

In the 2nd step, we pick the pattern in order to train the neural network.

In steps 3 and 4 it produces the prediction and then calculates the difference between the output value with the predicted value.

The 5th steps the backward phase starts.

Furthermore, in steps 6 and 8 are a calculation of δ which is the change of weights of output and hidden layers respectively. δ consists of the error derivation with respect to the output multiplied by the learning rate η(a learning rate) for which there are many strategies to change it. Choosing the right value of η is able to impact the whole process of learning since with a small value can be achieved a local minimum(instead of global) and with a huge value may lead to the irreversibility of adapting the values to their optimal.

2.3 Recurrent Neural Network

Not in all cases, the structured data can be the same size, for instance in text translation the input size is not always fixed. Exactly for such cases, Recurrent Neural Networks(RNNs) were invented.

Traditionally, for the text classification problem the input is considered as *one-hot* vector, where each component represents the index(number) in the internal vocabulary. However, this approach has problems:

1. Inputs and outputs can be different lengths in different examples and datasets.

2. Not share features learned across different positions of the input text.

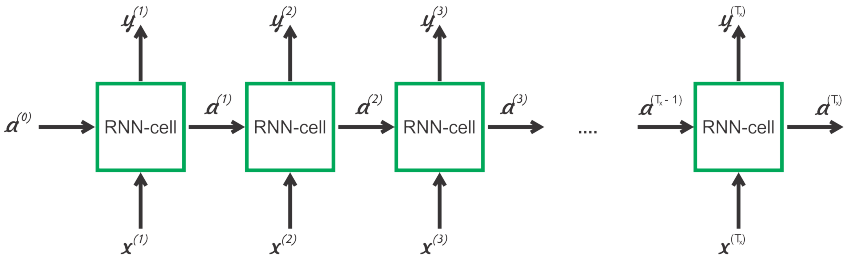


Figure 2.3: Recurrent Neural Network

Recurrent Neural Network(RNN) does not have these disadvantages. In contrast, RNN takes the x_1 word and tries to predict the output y_1, then for the x_2 word instead of just predicting y_2 it also gets the input from the previous step y_1. All recurrent neural networks have the form of a chain of repeating modules of a neural network.

2.3.1 Gated Recurrent Unit

A Gated Recurrent Unit(GRU) is a basic modification of RNN in the hidden layer that captures long-range connections much better and allows to avoid the gradient descent problem called vanishing.

GRU consists of 4 general steps:

1. Update Gate: calculation the **update gate z_t for the time step t**. The update gate helps to understand how much of the past information needs to be passed.

2. Reset Gate: calculation the r_t **for the time step t**. The reset gate helps to understand how much of the past information needs to be forgotten.

3. Cell Gate: calculation the memory content for the time step t by this formula:

$$h'_t = tanh(Wx_t + r_t \odot Uh_{t-1})$$

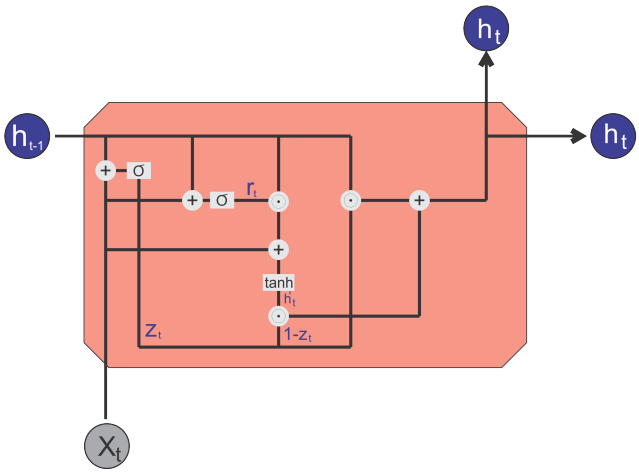


Figure 2.4: Gated Recurrent Unit

4. Output Gate: the network needs to calculate h_t — vector which holds information for the current unit and passes it down to the network:

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h'_t$$

2.3.2 Long Short Term Memory

A Long Short Term Memory(LSTM) is a special kind of RNN and more complicated than GRU. It was introduced by Hochreiter Schmidhuber in 1997[29] and were refined and popularised by many in NLP area.

The crucial aspect of LSTM is the cell state. It runs straight down the entire chain without any changes. The LSTM is not able to add or remove information to the cell state rather than in GRU.

LSTM also consists of 4 general steps:

1. Forget Gate: the gate decides how much of the information should be kept or ignored. The f_t indicates the keeping value or not.

2. Input Gate: in this step, it decides what fraction of new information should be stored in the cell state. Calculation of vector C'_t that can be added to the cell state and i_t is the output of the gate.

3. Cell State: update the C_{t-1} state by this formula:

$$C_t = f_t * C_{t-1} + i_t * C'_t$$

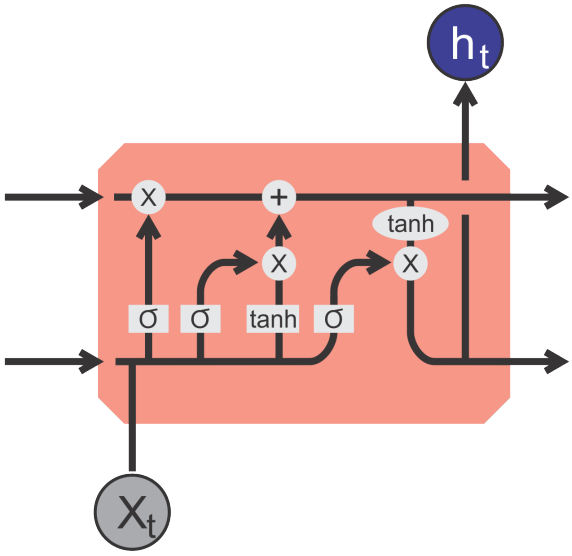


Figure 2.5: Long Short Term Memory

4. Output Gate: the output is based on our cell state, but will be a filtered version.

$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

2.4 Convolution Neural Networks

In 1995, the concept of Convolution Neural Network(CNN) was introduced by Yann LeCun[20] and the main purpose of it was to build a check-recognition system to read handwritten digit. CNN is a biologically-inspired result of multiple layer perceptrons.

Hubel and Wiesel's on there works on the cat's visual cortex[30], demonstrates the visual cortex has two types of cells. Such kind of cells was extremely sensitive to mini-regions of the visual field, named a receptive field. The mini-regions are tiled to cover the entire visual field.

These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images.

Figure (2.6) depicts the first version of CNN - Le-Net5.

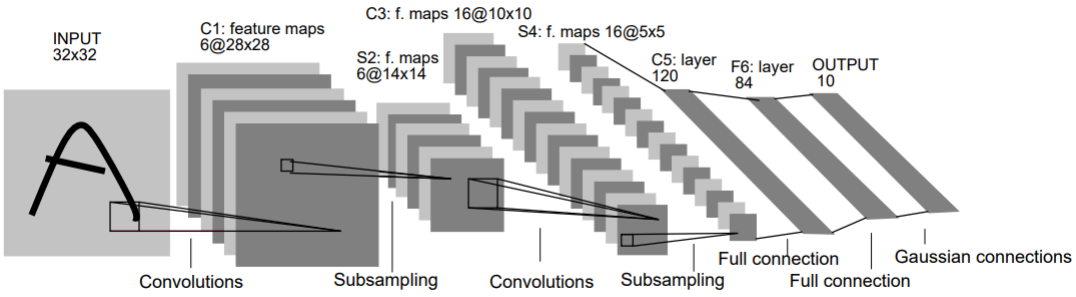


Figure 2.6: Le-Net5 model

During the research, two types of main cells had been identified:

1. Simple cells: respond maximally to specific edge-like patterns within their receptive field;

2. Complex cells: have larger receptive fields and are locally invariant to the exact position of the pattern;

Obviously, evolution had much more time to create complex systems like a brain that are efficiently adopted by many neuronal models. Researchers make a lot of attempts to imitate, LeNet-5[20], NeoCognitron[31], and HMAX[32] are examples of these assumptions.

2.4.1 Convolution and Filters

From the Latin convolver, "to convolve" means to roll together. From the mathematical perspective, a convolution is an operation on two functions (f and g) to make a third function that expresses how the shape of one is modified by the other. Convolution is able to express as follows:

$$(f * g)[x, y] = \sum_{i=-\infty}^{\infty} \sum_{j=-\infty}^{\infty} f[i, j]g[x - i, y - j] \tag{2.4}$$

Convolution layer is the core of CNN and computes the output of neurons that are connected to local regions in the input called a receptive-field, each computing is a dot product between their weights and a receptive-field. This may result in volume such as [32x32x12] if we decided to use 12 filters.

Mainly, as an activation function, ReLU or Tanh are commonly used. Figure (2.7) shows the filtering basic principle in convolution layer.

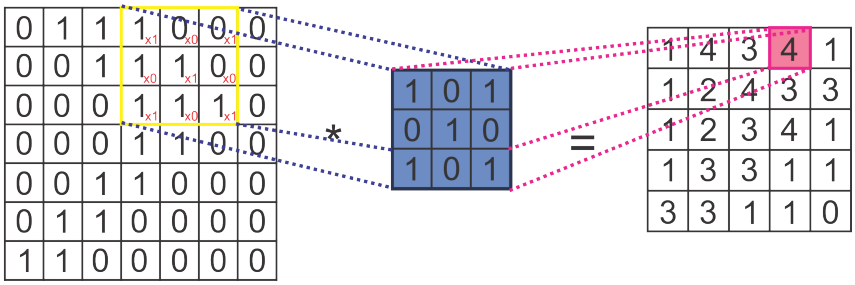


Figure 2.7: CNN filtering process

2.4.2 Pooling Layer

Pooling Layer performs a reducing/downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12] and it also serves against overfitting. This technique is done by passing a kernel on the image and producing the maximum number in its receptive field. As a reducing/downsampling operation mostly a *max* function is used, however variety of applications applied *min* and *average*.

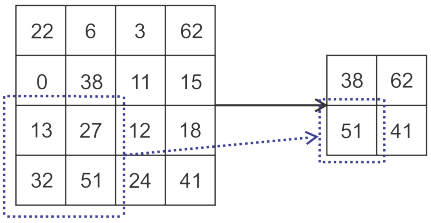


Figure 2.8: Max-pooling concept

2.4.3 Fully-Connected Layer

Fully-Connected layer computes the class scores, resulting in a volume of size [1x1x10], where each of the 10 numbers corresponds to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

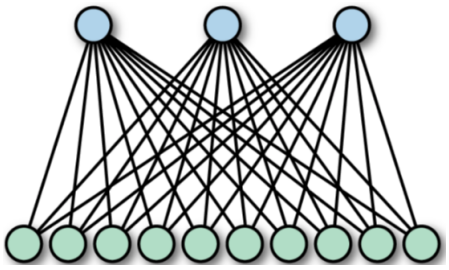


Figure 2.9: Fully-Connected Layer

2.4.4 The sequence of layers in CNN

Each CNN model consists of a different combination of the following layers:

1. **Input layer**: generally, this is the buffer layer;
2. **Convolutional layer**: performs the convolution operations for each kernel with the receptive-field(dot product with weights);
3. **Dropout layer**: this layer randomly disables according to a parameter a small or big subset of neurons during the learning process. Generally, this layer is used against *overfitting*[33].

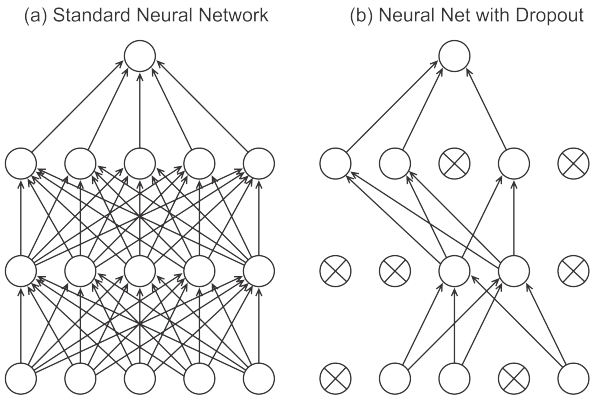


Figure 2.10: Drop-out concept

4. **Pooling layer**: this layer reduces the input by using one of *min*, *max* or *average* functions;
5. **Feed-forward layer**: calculates the probability distribution for each pre-defined class.

2.5 Capsule Neural Network

In this section, we in details describe Capsule Neural Network with Dynamic Routing and Capsule Neural Network with EM Routing.

All further information will be taken from these two publications: *'Dynamic Routing Between Capsules', Sara Sabour, Nicholas Frosst, Geoffrey E Hinton, cs.CV 2017*[7] and *'Matrix capsules with EM routing', Geoffrey E Hinton, Sara Sabour, Nicholas Frosst, ICLR 2018*[8].

Also, these articles had been used to describe models[34, 35, 36, 4, 9] and a series of articles from Max Pechyonkin[3, 5, 6, 37].

2.5.1 Drawbacks of CNN

Currently, CNN has been the state-of-the-art machine learning model for Image recognition and classification tasks. However, CNN has limitations and disadvantages.

Consider the face recognition task. After feeding the image to CNN, a set of kernels(filters) in the convolutional layer generates features by computing convolutional operation (2.4.1) and one of the non-linear function transforms the features. Then, the pooling layer(max pooling) (2.4.2) helps to reduce the training time by taking the most significant information for each sub-region.

At the first layers detect low-level features like edge and color gradient, whereas last layers detect high-level features like nose and eye. Finally, the fully-connected layers mix all high-level features and perform the classification prediction with a soft-max classifier.

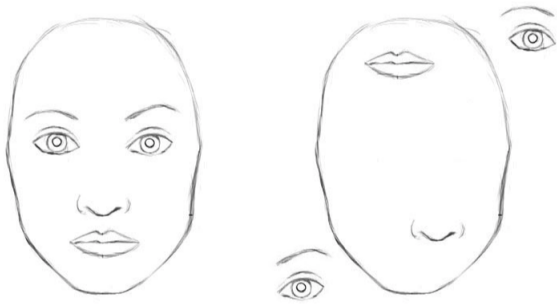


Figure 2.11: CNN detects both images as a face[3]

This example above demonstrates two faces with the same amount of eyes, lips, and eyes, however, well-trained CNN will recognize two faces what is the error.

The high-level features combine low-level features by multiplying with weights, then the activation function produces the output. It is important to note that **pose**(translational and rotational) relationship between low-level features is not taken into account. To solve this issue, CNN uses the max-pooling layer for reducing the spacial size of the data thereby increase *'field of view'* of high-level neuron's. This allows finding high-ordered features in a bigger region in the input image. In this case, CNN losses information about a relationship between part of the complex object.

Also, another significant disadvantage is fooling CNN with adding small unnoticeable changes to an image.

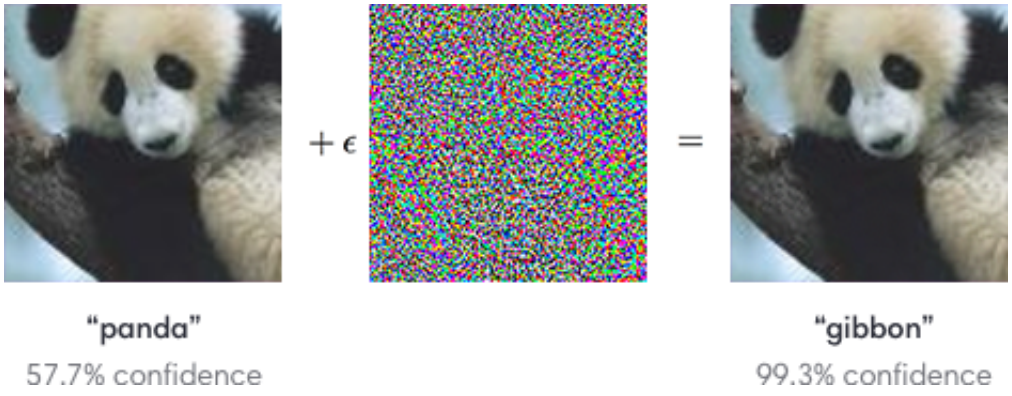


Figure 2.12: Miscategorization of a panda as a gibbon[4]

Figure (2.12) classifies the left object as 'panda', however with adding some small changes demonstrates misclassification.

2.5.2 Capsule Neural Network with Dynamic Routing

Capsule

A ***capsule*** is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part[7]. In other words, capsules encapsulate all important information about the state of the feature they are detecting in vector form[5].

Since the capsule is a vector, the length of it is a probability of detection of a feature, which means that even the detected object has rotated, the length of the vector will be the same(the probability still stays the same), but the vector direction will change in the direction of change.

For example, let assume, that the current capsule has detected a face within an input image with probability **0.9**. When the face starts to change the location across the image, the capsule's vector will change the direction

which means that it still detects the face, however, the length will be the same. Hinton refers to it as *activities equivariance*: neuronal activities will change when an object *'moves over the manifold of possible appearances'* in the image[5]. This is exactly the form of invariance, which is not the max-pool offer in CNN.

Figure (2.13) below demonstrates the difference between capsules and traditional neurons.

Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector($\mathbf{u}_i$)	scalar(x_i)
Operation	Affine Transform	$\widehat{\mathbf{u}}_{j\|i} = \mathbf{W}_{ij}\mathbf{u}_i$	–
	Weighting	$\mathbf{s}_j = \sum_i c_{ij}\widehat{\mathbf{u}}_{j\|i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1+\|\mathbf{s}_j\|^2}\frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$	$h_j = f(a_j)$
Output		vector($\mathbf{v}_j$)	scalar(h_j)

Figure 2.13: Capsule Vs Traditional Neuron[5]

The capsules have in general 4 different steps of working:

1. **Matrix multiplication of input vectors**:

   As an input for the current capsule is a vector $u = (u_1, u_2, \ldots, u_n)$, as mentioned above before, the length of the capsule is the probability lower-level capsules detected their corresponding objects. Assume that low-level capsules detect nose, mouth and eyes and high-level capsule detects a face.

   In this case, the matrix W encode the important relationships between low-level features(eyes, mouth, and nose) and high-level feature(face). For instance, u_2 detects the nose. Hence, the matrix W_{2j} encodes the relationship between nose and face: the position of the nose is a center within the face and so one.

   After affine transform, $\widehat{u}_2$ represents where the face should be according to the detected position of the nose.

   Figure 2.14 demonstrates the intuition that if the predictions of low-level capsules(eyes, nose, and mouth) correspond approximately to the same position, hence the face is there.

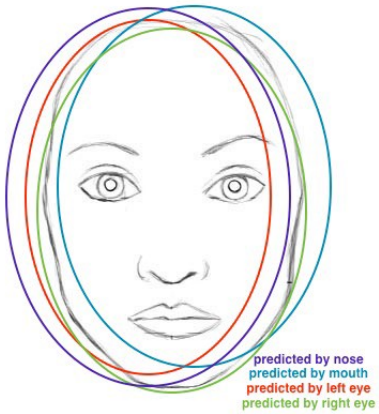


Figure 2.14: Face predictions of nose, mouth and eyes[5]

2. **Scalar weighting of input vectors**:

   Figure (2.15) shows for the low-level capsule, choosing between high-level capsules **J** and **K**. The choice between high-level capsules is done by the dynamic algorithm that adjusts the internal matrix **C** in such way that the wights in **C** corresponding to capsule **K** will be high, however, the weights corresponding to capsule **J** will be low.

   The Dynamic Routing algorithm will be described in the next section.

3. **Sum of weighted input vectors**:

   The same as for traditional neuron and it is the sum of all inputs.

4. **Vector-to-vector non-linearity**:

   This is the new non-linear function that was introduced by Hinton which takes the output of the capsule and *'squashes'* it, i.e scale to [0, 1] and does not change the direction.

Dynamic Routing algorithm

The significant part of dynamic routing is *'agreement'* between low-level and high-level capsule. This is the essence of the dynamic routing algorithm. An explanation of this term will be indicated in the description of the dynamic algorithm below.

For each high-level capsule, j executes the routing algorithm and performs the result $\mathbf{v}_j$.

The 1st line says that the input: all low-layer capsules in l layer and their respective outputs $\widehat{\mathbf{u}}_{j|i}$ and the number of routing iterations r.

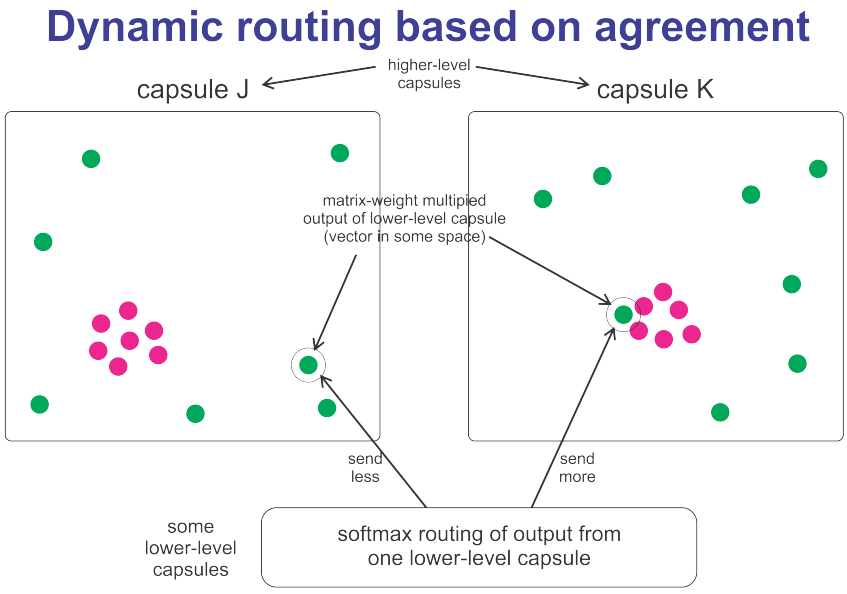


Figure 2.15: Dynamic Routing

$$\mathbf{v}_i = \frac{||s_j||^2}{1 - ||s_j||^2} \frac{s_j}{||s_j||^2}$$

Figure 2.16: Non-linear "squashing" function

2nd line shows initialization of the variable $b_{ij} = 0$, which iteratively updates and the final value will be assigned to c_{ij}.

The 3rd line starts the loop with r number of iterations.

4th line calculates for all low-level capsules c_i by using softmax which allows getting non-negative values and the sum of them will be 1.

For instance, if we have 4 low-layer capsules and 4 high-layer capsules, then all c_{ij} will be equal to 0.25(because initially b_i=0). Next 5th line calculates the linear combination of all low-level capsules with coefficients c_{ij} which allows to scale up or down the final value $\mathbf{s}_j$. Further, line 6 squashes s_j and the direction of the vector will be the same, but the maximum length is 1.

Lines 7-8 are crucial. It is here that the routing happens. We update all values b_{ij} for the current high-level capsule by dot product of current output $\mathbf{v}_j$ of capsule j with the input to this capsule $\widehat{\mathbf{u}}_{j|i}$.

Figure (2.17) demonstrates the intuition of the routing. Consider, we

Algorithm 2.2 Routing Algorithm.

procedure ROUTING($\widehat{\mathbf{u}}_{j|i}$, r, l)
  for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow 0$.
  for r iterations **do**
    for all capsule i in layer l: $\mathbf{c}_i \leftarrow softmax(b_i)$
    for all capsule j in layer $(l+1)$:: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\widehat{\mathbf{u}}_{j|i}$
    for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \text{squash}(\mathbf{s}_j)$
    for all capsule i in layer l and capsule j in layer $(l+1)$:
      $b_{ij} \leftarrow b_{ij} + \widehat{\mathbf{u}}_{j|i}\mathbf{v}_j$
  return $\mathbf{v}_j$
end procedure

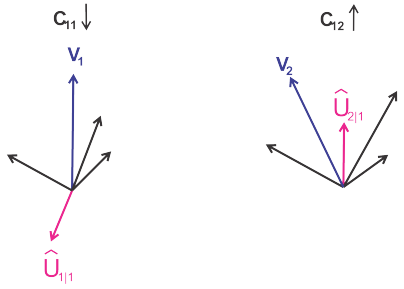


Figure 2.17: Two higher level capsules with their outputs represented by purple vectors, and inputs represented by black and orange vectors. Lower level capsule with orange output will decrease the weight for higher level capsule 1 (left side) and increase the weight for higher level capsule 2 (right side)[6]

have 2 high-level capsules $\mathbf{v}_1$ and $\mathbf{v}_2$. The orange vector represents one of the low-level capsules and black vectors are the rest.

The value c_{11} will decrease, because vectors $\mathbf{v}_1$ and $\widehat{\mathbf{u}}_{1|1}$ have opposite directions hence the dot product will be a negative number. In contrast, the value c_{21} will increase, because vectors $\mathbf{v}_2$ and $\widehat{\mathbf{u}}_{2|1}$ have the same direction, hence the dot product will be a positive number. For each high-level capsules, the procedure will be executed and all c_{ij} will be updated. This is how dynamic routing works.

CapsNet Architecture

The CapsNet consists of two major parts: encoder and decoder.

Encoder: Takes the input of 28x28 MNIST digit images and construct

capsules and do routing algorithm. The output of this part is 10-dimensional vectors of lengths of DigitCaps'.

Decoder: Takes the correct output from DigitCaps and attempts to reconstruct the digit. Also, a decoder is used as a regularization method, which takes the loss function as a Euclidean distance between the reconstructed image and the input image.

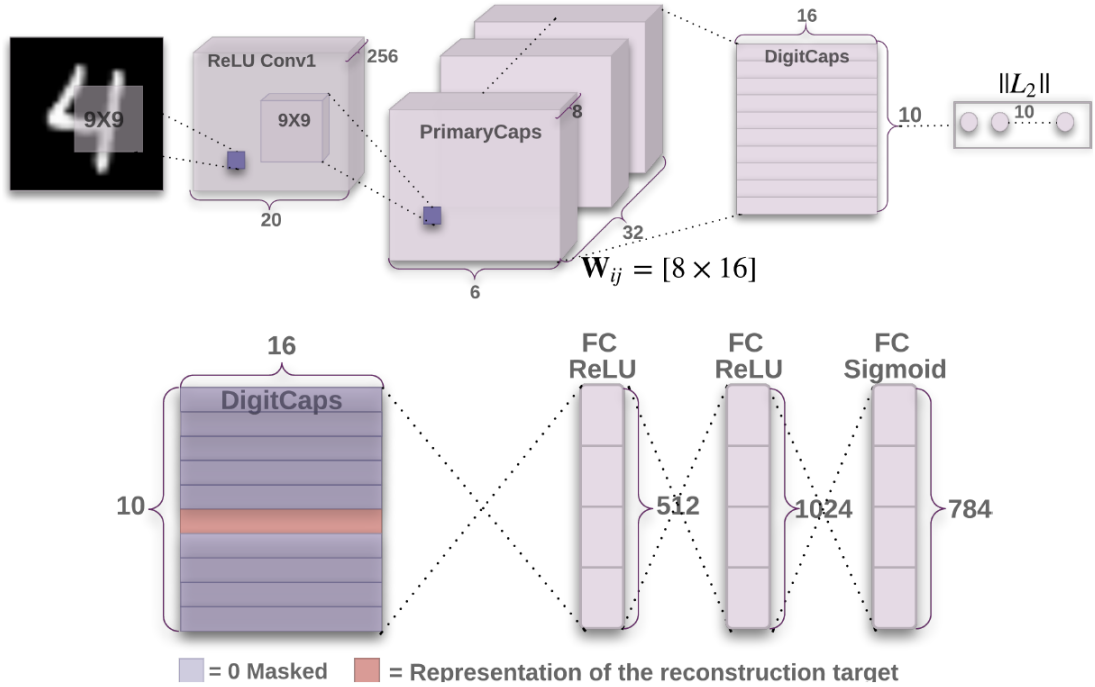


Figure 2.18: CapsNet Architecture[7]

The first three layers of CapsNet is an encoder, the remaining three layers are decoder. In this publication *'Dynamic Routing Between Capsules', Sara Sabour, Nicholas Frosst, Geoffrey E Hinton, cs.CV 2017*[7] layers of CapsNet were introduced as follows:

1. **Convolutional layer**:

   - Input: 28x28 image (one color channel).
   - Output: 20x20x256 tensor.
   - Number of parameters: 20992.

   The first layer task to detect basic features within an input 28x28 image. 256 kernels with the size of 9x9x1 and stride 1 are used with ReLU activation function.

2. **PrimaryCaps layer**:

   - Input: 20x20x256 tensor.
   - Output: 6x6x8x32 tensor.
   - Number of parameters: 5308672.

   The 32 low-level capsules are formed in this layer with 9x9x256 convolutional kernels (with stride 2). The main task is to combine the basic features into capsules.

3. **DigitCaps layer**:

   - Input: 6x6x8x32 tensor.
   - Output: 16x10 matrix.
   - Number of parameters: 1497600.

   Since MNIST is a dataset, therefore, this layer has 10-digit capsules. Dynamic Routing by Agreement executes here. To allow for multiple digits, we use a separate margin loss, L_c for each digit capsule, c:

$$L_c = T_c max(0, m^+ - ||\mathbf{v}_c||)^2 + \lambda(1 - T_c)max(0, ||v_c|| - m^-)^2$$

Figure 2.19: CapsNet Loss Function

   Where, $T_c = 1$ iff a digit of class c is present and $m^+ = 0.9$ and $m^- = 0.1$. The λ down-weighting of the loss with value = $\lambda = 0.5$[7].

4. **Fully connected #1**:

   - Input: 16x10.
   - Output: 512.
   - Number of parameters: 82432.

   This layer and next two layers form decoder which is the main task is to reconstruct the original image.

5. **Fully connected #2**:

   - Input: 512.
   - Output: 1024.

- Number of parameters: 525312.

6. **Fully connected #3**:
   - Input: 1024.
   - Output: 784 (which after reshaping gives back a 28x28 decoded image).
   - Number of parameters: 803600.

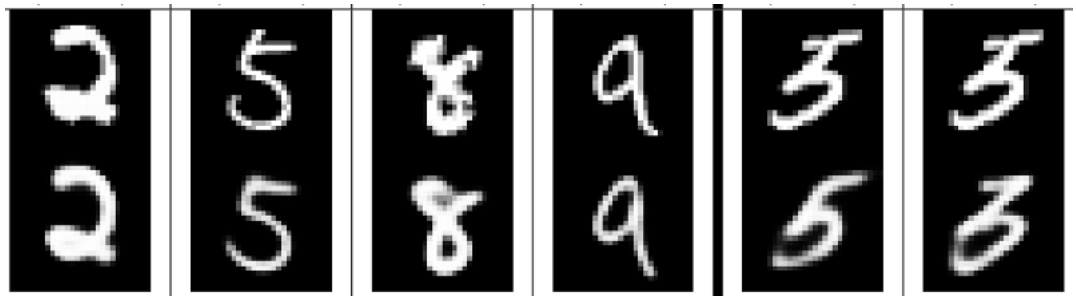


Figure 2.20: Top row: original images. Bottom row: reconstructed images[7]

Total number of trainable parameters: 8238608

2.5.3 Capsule Neural Network with EM Routing

Matrix capsule

Matrix capsule is the new type of capsules which was introduced[8] with containing 4x4 matrix which is able to find between to an entity and the viewer(the pose) the relationship and an activation probability denoted as a.

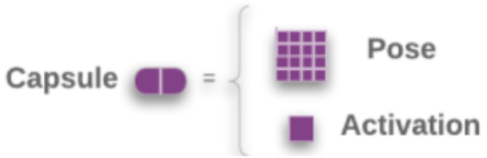


Figure 2.21: Matrix Capsule[8]

The pose matrix allows getting the positional arrangement of all smaller parts it consists of multiplication with a transformation matrix.

P: pose matrix of the person. M: the spatial relationship between face and person N: represents the relationship between mouth and face. M' and N: pose matrices for the face and the mouth.

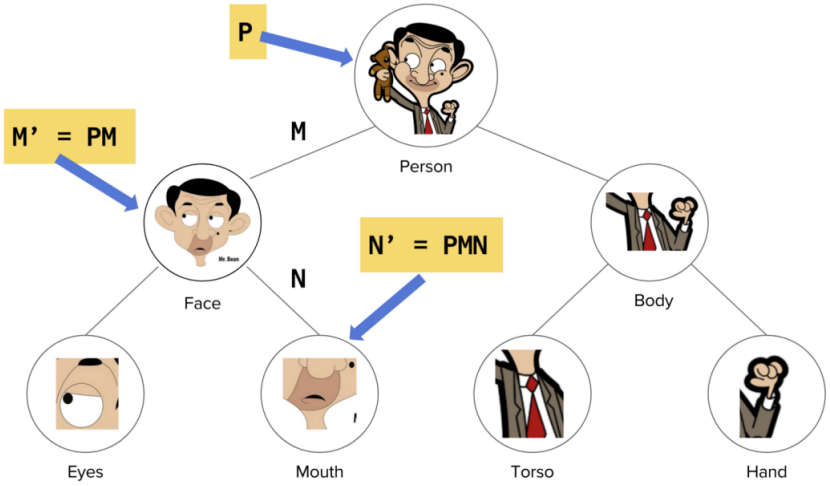


Figure 2.22: Pose matrices representing the hierarchical relationship[9]

Working of Matrix Capsules

Ω_L - a set of all capsules in L layer. As described above, each capsule has a M 4x4 pose matrix with an activation probability a. W_{ij} - a discriminatively trainable *transformation matrix* which is between a capsule i in L layer and a capsule j in $L+1$.

The pose matrix of capsule i is transformed by W_{ij} to cast a vote $V_{ij} = M_i W_{ij}$ for the pose matrix of capsule j[8]. Calculations of poses and activations for all capsules in layer $L+1$ is done by using a non-linear routing procedure where the input are V_{ij} and a_i $\forall i \in \Omega_L, \forall j \in \Omega_{L+1}$.

The non-linear procedure is a modification of the Expectation-Maximization algorithms which originally was introduced for Gaussian Mixture Model.

Gaussian Mixture Model. Expectation-Maximization Algorithm

In statistics, the Gaussian Mixture Model(GMM) is a probabilistic model which expresses the normally distributed subgroup within an overall group. GMM is a generalization of clustering K-means algorithm and accordingly, it is a sample of unsupervised learning.

GMM is useful for modeling data that comes from one of several groups: the groups might be different from each other, but data points within the same group can be well-modeled by a Gaussian distribution.

GMM is adjusted by two types of parameters: number of components K and for each component k^{th} a mean $\overrightarrow{\mu_k}$ and co-variance matrix Σ_k. Moreover, number of components is defined as ϕ_k for each component C_k witch

condition: $\sum_{i=1}^{K} \phi_i = 1$.

GMM model:

$$
\begin{aligned}
p(\vec{x}|\mu, \Sigma, \phi) &= \sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x}|\vec{\mu_i}, \Sigma_i) \\
\mathcal{N}(\vec{x}|\vec{\mu_i}; \Sigma_i) &= \frac{1}{\sqrt{2\pi|\Sigma|}} exp(-\frac{1}{2}(\vec{x} - \vec{\mu_i})^T \Sigma_i^{-1}(\vec{x} - \vec{\mu_i})) \\
&\sum_{i=1}^{K} \phi_i = 1 \wedge \forall i : 0 \leq \phi_i \leq 1
\end{aligned}
\tag{2.5}
$$

μ - the mean, Σ - co-variance matrix.

Usually, in probability theory to find the parameters of a model using **maximum likelihood estimation** techniques, which attempts to maximize the likelihood of the observed data a given model parameters. The maximum likelihood of **??** is:

$$
\begin{aligned}
ln(p(X|\mu, \Sigma, \phi)) &= \sum_{n=1}^{N} ln(p(\vec{x_n}|\mu, \Sigma, \phi)) = \\
&= \sum_{n=1}^{N} ln\{\sum_{i=1}^{K} \phi_i \mathcal{N}(\vec{x_n}|\vec{\mu_i}, \Sigma_i)\}
\end{aligned}
\tag{2.6}
$$

However, in the case of GMM is usually analytically impossible for finding. Therefore, **Expectation-Maximization(EM)** Algorithm had been used, which allows finding the parameters of GMM iteratively. EM at each iteration rigorously increases the maximum likelihood of the data with a guarantee of reaching a *local maximum*.

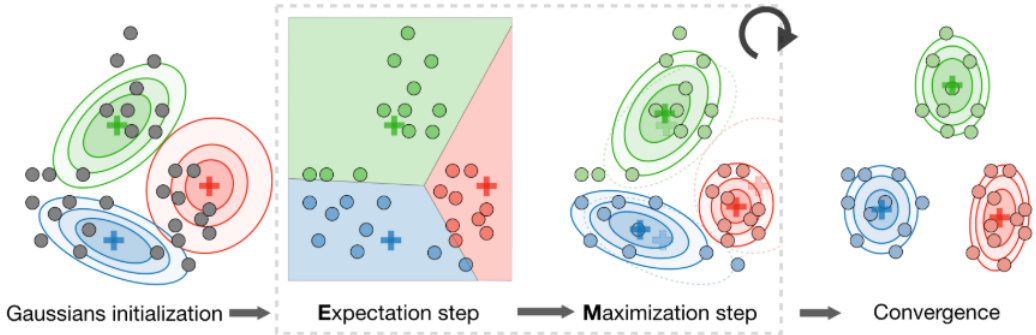


Figure 2.23: Expectation-Maximization algorithm[10]

Figure (2.23) illustrates EM algorithm which consists of an initialization and two major steps:

1. **Initialization**:

   - randomly initialization of $\hat{\mu_1}, \hat{\mu_2} \dots \hat{\mu_k}$ for each C_k from a given dataset X;
   - calculate the variance with the same value:

   $$\hat{\sigma_1}^2, \hat{\sigma_2}^2 \dots \hat{\sigma_K}^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2, \bar{x} = \frac{1}{N}\sum_{i=1}^{N} x_i$$

   - set $\forall i$: $\hat{\phi_i} = \frac{1}{N}$

2. **E-step**: $\forall C_k$ calculate the expectation(mean) value for $\forall x_i \in X$ with parameters ϕ_k, μ_k and σ_k.

   More formal: calculate i, k:

   $$\hat{\gamma}_{ik} = p(C_k|x_i, \hat{\phi}, \hat{\mu}, \hat{\sigma}) = \frac{\hat{\phi_k}\mathcal{N}(x_i|\hat{\mu_k}, \hat{\sigma_k})}{\sum_{j=1}^{K}\hat{\phi_j}\mathcal{N}(x_i|\hat{\mu_j}, \hat{\sigma_j})}$$

   Where $\hat{\gamma}_{ik}$ represents x_i is generated by component C_k.

3. **M-step**: maximization the expectation(mean) value with respect to the model parameters: ϕ_k, μ_k and σ_k.

   More formal: with this $\hat{\gamma}_{ik}$ value, calculate ϕ_k, μ_k and σ_k.

   $$\hat{\phi_k} = \sum_{i=1}^{N}\frac{\hat{\gamma}_{ik}}{N}$$
   $$\hat{\mu_k} = \frac{\sum_{i=1}^{N}\hat{\gamma}_{ik}x_i}{\sum_{i}^{N}\hat{\gamma}_{ik}}$$
   $$\hat{\sigma_k} = \frac{\sum_{i=1}^{N}\hat{\gamma}_{ik}(x_i - \hat{\mu_k})^2}{\sum_{i=1}^{N}\hat{\gamma}_{ik}}$$

These two steps will continue until the algorithm converges and the result is a maximum likelihood estimate.

Routing by agreement based on EM

Consider the task of face detection. A high-level capsule in layer $L+1$, which detects the face by looking for agreement between votes from all capsules in the layer L. For the current capsule j the votes v_{ij} calculate from all capsules i in the layer L by multiplying the pose matrix M_i of capsule i by a transformation matrix(viewpoint invariant) W_{ij}:

$$v_{ij} = M_i W_{ij}$$

So, we have a vector of votes $(v_{o_1 j}, v_{o2j} \dots v_{o_k j})$. The calculation of the probability that a capsule i is grouped into capsule j is determined based on the proximity of the vote v_{ij} to other votes. W_{ij} is learned during the backpropagation with loss function and also it learns what a face is composed of and makes sure the pose information of the capsule j matched with its sub-components after some transformation.

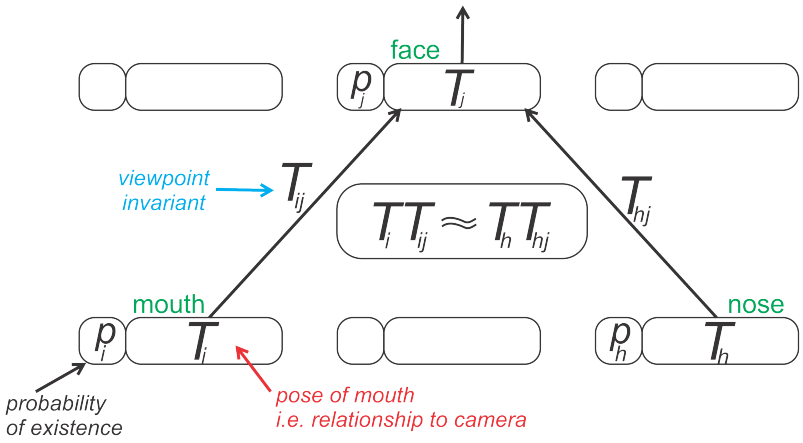


Figure 2.24: EM routing visualization

Figure (2.24) visualizes the routing by agreement with matrix capsules. The crucial point is grouping capsules with similar votes: $T_i T_{ij} \approx T_h T_{hj}$ where T_i and T_h the pose matrices of capsules i and J and $T_{ij}(W_{ij})$, $T_{hj}(W_{hj})$ viewpoint invariant transformations.

Figure (2.25) shows in spite of a face rotating(red dots change to pink dots), EM routing is based on proximity, hence can still cluster the same children capsules together. The pose matrices and the votes coordinate to each other when the viewpoint is changed. Therefore, the transformation

Figure 2.25: Face rotation[4]

matrices W will not change for any viewpoints of the object which means W is a viewpoint invariant.

Also, EM routing makes clustering in runtime to form a height level cluster. Meanwhile, the **assignment probabilities** r_{ij} which represents a strength of a connection between high-level capsule and low-level capsules is calculating during the runtime.

There is a difference in the calculation of the output value between a capsule and a classical neuron. EM clustering represents data points by a Gaussian distribution and models a parent's pose matrix with Gaussian also. In total, 16 = 4x4 components in a pose matrix hence EM routing models with a 16 Gaussians with 16μ and 16σ and each μ represents a pose matrix's component.

Let v_{ij} - the vote from the capsule i for the parent capsule j and v_{ij}^h - h-th component of v_{ij}. The calculation of the v_{ij}^h probability belonging to the capsule j's Gaussian model is done as follows:

$$p_{i|j}^h = \frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} exp(-\frac{(v_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2})$$

Let's take ln:

$$ln(p_{i|j}^h) = ln\frac{1}{\sqrt{2\pi(\sigma_j^h)^2}} exp(-\frac{(v_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2})$$
$$= -ln(\sigma_j^h) - \frac{ln(2\pi)}{2} - \frac{(v_{ij}^h - \mu_j^h)^2}{2(\sigma_j^h)^2}$$

Let $cost_{ij}$ - the cost to activate the parent capsule j by the low-level capsule i. The smaller the value the greater the likelihood of activation and vice versa which is able to represent as follow:

$$cost_{ij}^{h} = -ln(P_{i|j}^{h})$$

Since capsules are not equally linked with capsule j, the runtime **assignment probabilities** r_{ij} are used as follows:

$$cost_{j}^{h} = \sum_{i} r_{ij} cost_{ij}^{h} = \sum_{i} -r_{ij} ln(P_{i|j}^{h})$$

$$= \sum_{i} r_{ij} (ln(\sigma_{j}^{h}) + \frac{ln(2\pi)}{2} + \frac{(v_{ij}^{h} - \mu_{j}^{h})^{2}}{2(\sigma_{j}^{h})^{2}})$$

$$= \frac{\sum_{i} r_{ij} (\sigma_{j}^{h})^{2}}{2(\sigma_{j}^{h})^{2}} + (ln(\sigma_{j}^{h}) + \frac{len(2\pi)}{2}) \sum_{i} r_{ij} = (ln(\sigma_{j}^{h}) + k) \sum_{i} r_{ij}$$

Where k - a constant.

The following equation is used to make a solution about activation of the capsule j:

$$a_{j} = sigmoid(\lambda(b_{j} - \sum_{h} cost_{j}^{h}))$$

Where b_j - the cost of describing the mean and variance of capsule j.

During EM routing r_{ij}, μ, σ are computed. λ - the inverse temperature parameter and helps for tuning r_{ij} to better with higher sensitivity to vote discrepancy in this region.

Figure (2.26) demonstrates the EM Routing algorithm which calculates the pose matrix and the activation of all output capsules.

The input: a - the activation and V - the votes from the children capsules.

The output: a - the activation and M - the pose matrix of all capsules.

2nd line shows the initialization of r_{ij} which uniformly distributed.

3rd line shows the loop with r iteration

4th line performs M-step which re-calculate the Gaussian models' values (μ, σ) and parent activation a_j from a, V based on r_{ij}. Moreover, M-step re-calculate $cost^h$ and the activation a_j for the parent capsules. β_v and β_α is trained discriminatively.

5th line performs E-step which recomputes the new r_{ij} based on the new Gaussian model and the new a_j for all lower-level capsules.

Procedure 1 Routing algorithm returns **activation** and **pose** of the capsules in layer $L+1$ given the **activations** and **votes** of capsules in layer L. V_{ij}^h is the h^{th} dimension of the vote from capsule i with activation a_i in layer L to capsule j in layer $L+1$. β_a, β_u are learned discriminatively and the inverse temperature λ increases at each iteration with a fixed schedule.

procedure EM ROUTING($\boldsymbol{a}$, V)
  $\forall i \in \Omega_L, j \in \Omega_{L+1}$: $R_{ij} \leftarrow 1/|\Omega_{L+1}|$
  for t iterations **do**
    $\forall j \in \Omega_{L+1}$: M-STEP($\boldsymbol{a}$, R, V, j)
    $\forall i \in \Omega_L$: E-STEP(μ, σ, $\boldsymbol{a}$, V, i)
  return $\boldsymbol{a}$, M

procedure M-STEP($\boldsymbol{a}$, R, V, j) ▷ for one higher-level capsule, j
  $\forall i \in \Omega_L$: $R_{ij} \leftarrow R_{ij} * \boldsymbol{a}_i$
  $\forall h$: $\mu_j^h \leftarrow \frac{\sum_i R_{ij} V_{ij}^h}{\sum_i R_{ij}}$
  $\forall h$: $(\sigma_j^h)^2 \leftarrow \frac{\sum_i R_{ij} (V_{ij}^h - \mu_j^h)^2}{\sum_i R_{ij}}$
  $cost^h \leftarrow \left(\beta_u + log(\sigma_j^h)\right) \sum_i R_{ij}$
  $a_j \leftarrow logistic(\lambda(\beta_a - \sum_h cost^h))$

procedure E-STEP(μ, σ, $\boldsymbol{a}$, V, i) ▷ for one lower-level capsule, i
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{p}_j \leftarrow \frac{1}{\sqrt{\prod_h^H 2\pi(\sigma_j^h)^2}} \exp\left(-\sum_h^H \frac{(V_{ij}^h - \boldsymbol{\mu}_j^h)^2}{2(\sigma_j^h)^2}\right)$
  $\forall j \in \Omega_{L+1}$: $\boldsymbol{R}_{ij} \leftarrow \frac{\boldsymbol{a}_j p_j}{\sum_{k \in \Omega_{L+1}} \boldsymbol{a}_k p_k}$

Figure 2.26: EM Routing[8]

Loss Function

Choosing the loss function is significant, since matrix capsules expects to train W, β_v and β_α. The spread loss is used as a loss function for the backpropagation. The loss from the class i (other than the true label t):

$$L_i = (max(0, m - (a_t - a_i)))^2$$

Where a_t - target class's activation, a_i - the activation for class i.
The objective function:

$$L = \sum_{i \neq t} L_i$$

If the difference between between the true label and the wrong class is smaller than m, in this case we penalize by $(m-(a_t-a_i))^2$. Initially, $m = 0.2$ and each epoch is increased by 0.1 until 0.9 value.

CapsNet Architecture

Figure (2.27) demonstrates the original architecture that was introduced in *'Matrix capsules with EM routing', Geoffrey E Hinton, Sara Sabour, Nicholas*

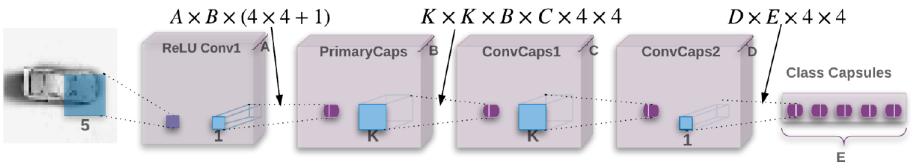


Figure 2.27: Capsules Architecture[8]

Frosst, ICLR 2018[8] as follows:

1. **ReLU Conv1**:

   CNN layer with 5x5 kernels, 32($A = 32$) output channels, stride = 2, ReLU activation.

   Output shape: (14,14, 32)

2. **PrimaryCaps**:

   Use 1x1 convolution filter to transform each of the 32 channels into 32 ($B = 32$) primary capsules, stride = 1.

   Output shape: pose (14, 14, 32, 4, 4), activations (14, 14, 32)

3. **ConvCaps1**:

   Capsule convolution with 3x3 filters (K=3) with stride = 2, no padding.

   EM routing happens here.

   Output shape: poses (6, 6, 32, 4, 4), activations (6, 6, 32)

4. **ConvCaps2**:

   Capsule convolution with 3x3 kernels, strides 1 and no padding.

   EM routing happens here.

   Output shape: poses (4, 4, 32, 4, 4), activations (4, 4, 32)

5. **Class Capsules**: Capsule with 1x1 kernel.

   EM routing happens here.

   Output shape: poses (10, 4, 4), activations (10)

Chapter 3

Related Work

Before introducing Neural Networks, a variety of approaches are used to text classification from probabilistic models, regression models, decision tree and decision rule learners, ensemble learning(random forest), batch and incremental learners of linear classifiers, example-based models, support vector machine and hidden Markov models, classifier committees(which included boosting methods).

Originally, these machine learning models were not created to solve text classification problem, however, in statistics, there is a common aphorism ***'All models are wrong, but some are useful'***, which means that any model may be applied to any task. Therefore in the machine learning area, usage of models is not strictly limited to certain domains.

Despite the fact that many machine learning algorithms have been proposed, text classification is still one of the most researched areas in NLP, because currently, automated text classifiers not faultless and still needs improvement.

In the following sections Naive Bayes, K-Nearest Neighbors, Decision Tree and Support Vector Machine are considered, basic principles are outlined and their efficiency is demonstrated on various datasets.

3.1 Naive Bayes

Naive Bayes is one of the simplest machine learning models based on the Bayes Theorem and the adjective *naive* comes from the assumption that the features in a dataset are independent. The main idea is to compute the probability that the text belongs to the class.

In order to calculate the probability that a text t belongs to a class s, it is necessary to summarise the probability that each lexical term $l_1^t, l_2^t \dots l_n^t$

belongs to s class[38].

The following steps demonstrate how Naive Bayes works:

1. For each class calculate the probability vector(w_1, w_2, ..., w_n) of feature terms;

2. For a new text based on feature terms, calculate the probability of t_i belongs to C_j:

$$P(C_j|t_i) = \frac{P(C_j)P(t_i|C_j)}{P(t_i)} = \frac{P(C_j)\prod_{k=1}^{n} p(W_k|C_j)}{P(t_i)}$$

$$\text{Laplacian smoothing} : \hat{p}(W_k|C_j) = \frac{1 + N_{kj}}{M + \sum_{i=1}^{M} N_{ij}}$$

   The denominator $\mathrm{P}(t_i)$ can be neglected, because it is fixed for the new text; where $\hat{p}(W_k|C_j)$ can be unbiased estimation $p(W_k|C_j)$.

3. Return the maximum probability among all classes that the new text belong to.

$$C_{MAP} = \arg\max_{c_j \in C} P(c_j)P(t_i|c_j)$$

In general, two ***generative*** models have been used for making the "Naive Bayes assumption": ***Multi-variate Bernoulli*** and ***Multinomial*** event models. Both of them are called "Naive Bayes" by their practitioners[39][40].

When a dataset is huge, a multinomial event model is preferable to use. However, there are several issues: rough parameter estimated and an issue with classes that consists of only few training documents. As an alternative, a Poisson even model is able to be used for over-passing the drawbacks of a multinomial even model by using weight reinforcing method to boost the performance of rare categories[15].

Naive Bayes is easy, fast for implementation and calculation. Nevertheless, it has a probability classification model, two assumptions:

1. All attributes of the category must take independent values, which means all attributes fully independent;

2. The texts length are independent of their categories. Practically, it appears rarely, therefore it is the main reason for the low accuracy Naive Bayes sometimes has.

3.2 K-Nearest Neighbors

K-Nearest Neighbors(KNN) is a lazy learning algorithm based on a distance such as Euclidean or Cosine[41]. For the text t to be classified it's k nearest neighbors are extracted, and these form a *neighbourhood of t*. KNN stores all training samples in n-dimensional space.

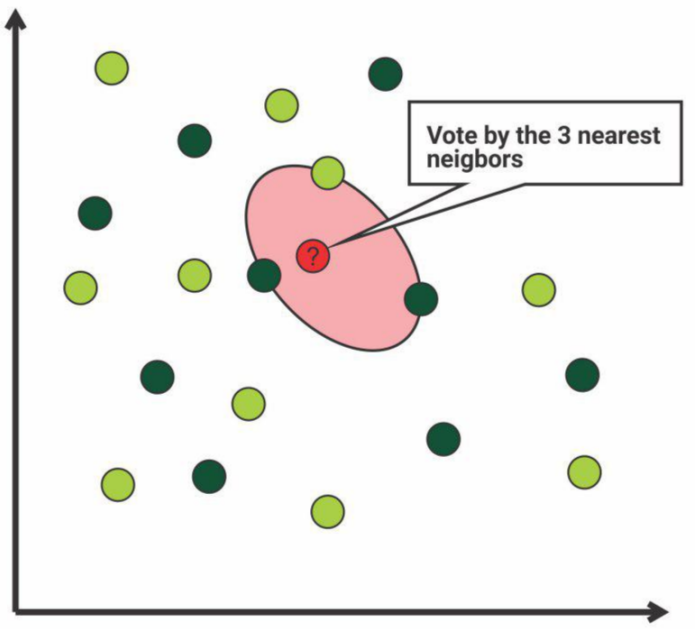


Figure 3.1: K-Nearest Neighbors Classification

The most significant for this algorithm is choosing a relevant distance function. Usually, ***Euclidean distance*** has been used for KNN, however there are many different distance functions[42].

Let's $A = (x_1, x_2, \ldots x_m)$ and $B = (y_1, y_2, \ldots y_m)$ are feature vectors in a m-dimensional feature space. For calculation a distance between A and B ***Normalized Euclidean*** metric is used:

$$dist(A, B) = \sqrt{\frac{\sum_{i=1}^{m}(x_i - y_i)^2}{m}}$$

However, ***Cosine Similarity*** measure is typically used to measure similarity values between documents in text retrieval[43]:

$$dist(A, B) = \frac{\vec{A} \cdot \vec{B}}{|\vec{A}| \cdot |\vec{B}|}$$

In different circumstances, other distance functions are also available for KNN classification, such as ***Minkowsky***[1], ***Correlation*** and ***Chi square***[2].

$$dist\_Minkowsky(A,B) = (\sum_{i=1}^{m} |x_i - y_i|^r)^{1/r}$$

$$dist\_correlation(A,B) = \frac{\sum_{i=1}^{m}(x_i - \mu_i)(y_i - \mu_i)}{\sqrt{\sum_{i=1}^{m}(x_i - \mu_i)^2 \sum_{i=1}^{m}(y_i - \mu_i)^2}}$$

$$dist\_Chi\text{–}square(A,B) = \sum_{i=1}^{m} \frac{1}{sum_i}(\frac{x_i}{size_Q} - \frac{y_i}{size_I})^2$$

Many applications have used this method[44], because it is effective, simple to implement, non-parametric and usually robust to data noise. Notwithstanding, the process of making a prediction is too long and it is a great issue to find an optimal value for k. The best choice of k depends on data. Moreover, KNN is not scalable with respect to increasing of k, because more computation occurs to find nearest neighbors and therefore KNN is most often used for low-level data dimensional.

3.3 Decision Tree

Decision Tree is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree. One of the most important advantages of Decision Tree over other machine learning algorithms is that it is readable for human and represents sets of if-then rules[45]. The rules are learned once while learning by using a training dataset. All tuples will be deleted during the learning which is covered by a learned rule. This process is continued on the training set until meeting a termination condition.

Decision Tree can be used for text classification. In this case, internal nodes are labeled by terms, branches departing from them are labeled by test on the weight and the leaf node are present the matching class label. Decision Tree can classify the input text by going through the structure of a tree from the root to the corresponding leaf node, which represents the goal for the classification of the input text.

One of the significant disadvantages is that most of the training data is not able to fit in memory Decision Tree construction. To resolve this issue[46]

[1]Minkowsiki distance is typically used with r being 1 or 2, where the former is sometimes known as the Manhattan distance and the latter is the Euclidean distance

[2]`http://archive.ics.uci.edu/ml/index.php`

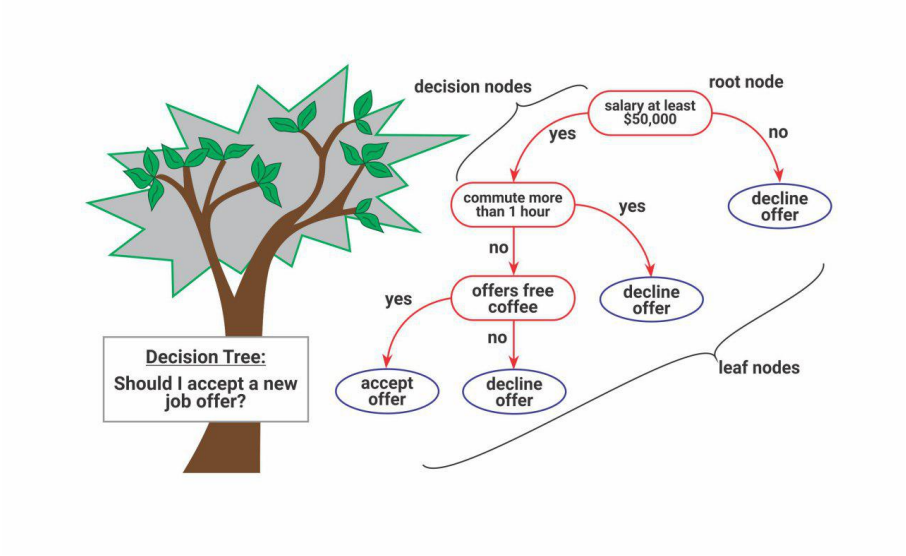


Figure 3.2: Decision Tree Classification

is proposed the new method which allows managing numeric and categorical data.

The new method[47] is introduced Fast Decision Tree Induction (FDI) to puzzle out the multi-label text which decreases the cost of induction and [48] involves the novel combination of:

1. a fast decision tree induction algorithm especially appropriated to text data;

2. a new method for converting a decision tree to a rule set that is simplified, but still logically equivalent to, the original tree.

3.4 Support Vector Machine

Support Vector Machine (SVM) is based on the *Structural Risk Minimization* principle[49] from computational learning theory.

The idea of structural risk minimization is to obtain a hypothesis h for which we can guarantee the lowest true error. The true error is the probability that h makes an error on an unseen and an arbitrarily picked test sample. An upper bound is able to be used for relating the true error of a hypothesis h with the error of h on the training dataset and the complexity of H (measured by VC-Dimension), the hypothesis space containing[49].

In this paper[50] SVM has been proposed for text classification and commonly needs the binary class labels for the construction of the best separates

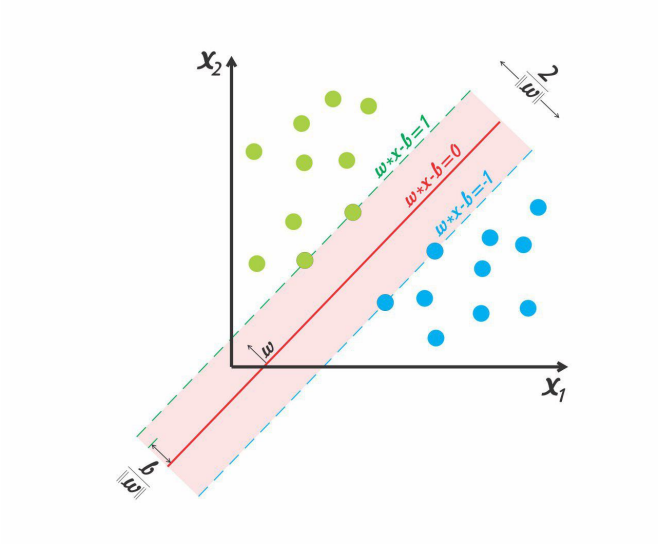


Figure 3.3: Support Vector Machine Classification

the positive from the negative data in the n-dimensional space, so-called the hyperplane.

Structured Support Vector Machine(SSVM) is introduced to solve multi-label classification problem[51]. Structured SVM is a machine learning algorithm that generalizes SVM and can be used not only for classification and regression but also allows training of a classifier for general structures like sequences, trees or sets. For the following problems with a compound output, SSVM can be perfectly applied: Natural Language Parsing, Sequence Alignment in protein homology detection and Markov models for Parts Of Speech Tagging.

Chapter 4

Approach

This chapter describes all the datasets that were used with a detailed description and also provides graphs of sentences length distribution. Additionally, illustrates in details data preparation and using pre-defined vectors like glove, word2vec, and fasttext. Furthermore, covers all modifications of CapsNet with Dynamic Routing and CapsNet with EM Routing.

4.1 Data and Preprocessing

The main task of the supervised learning is for a machine learning model and for the current dataset and try to learn it for finding patterns and automatically improve their performance on certain tasks and/or adapt to changing circumstances over time.

Therefore, the data preparation process is prominent in machine learning and in general takes more than half-time of the whole project. Because of this, there are lots of different approaches how to collect the data and prepare it. In this chapter one of them will be considered.

The preparation process can be divided into three steps:

1. **Select Data**: selecting appropriate datasets that will be fed to the machine learning model. As the best case consider as many datasets as possible because the performance directly depends on the number of datasets.

   4.1.1 section describes 7 different datasets.

2. **Preprocess Data**: getting selected datasets in an appropriate form for feeding to the machine learning models. There are lots of preprocessing steps like formatting and cleaning.

   4.1.2 section describes all steps used for this thesis.

3. **Transform Data**: transformation of the data for exact machine learning model. In the case of neural networks, the internal/external embedding is used for feeding the data.

   4.1.3 describes 3 different pre-trained word vectors.

4.1.1 Datasets

Choosing the right dataset is the first important task because of the performance of a machine learning model. These following 7 different datasets are public, the majority of them are used for sentiment-analysis. In the following sections, each of them will be described and put graphs about the distribution of the sentence's length.

Data	Classes	ASL	DS	VS	TS
MR	2	10	10662	18008	CV
SST-1	5	10	11855	17471	2210
SST-2	2	10	9163	1741	1821
SUBJ	2	13	10000	20491	CV
TREC	6	5	5952	8328	500
ProcCons	2	6	45875	8972	CV
IMDB	2	120	50000	99455	CV

ASL - Average Sentence Length, DS - Dataset Size
VS - Vocabulary Size, TS - Test Size, CV - no standard train/test split

Table 4.1: Overall information about datasets

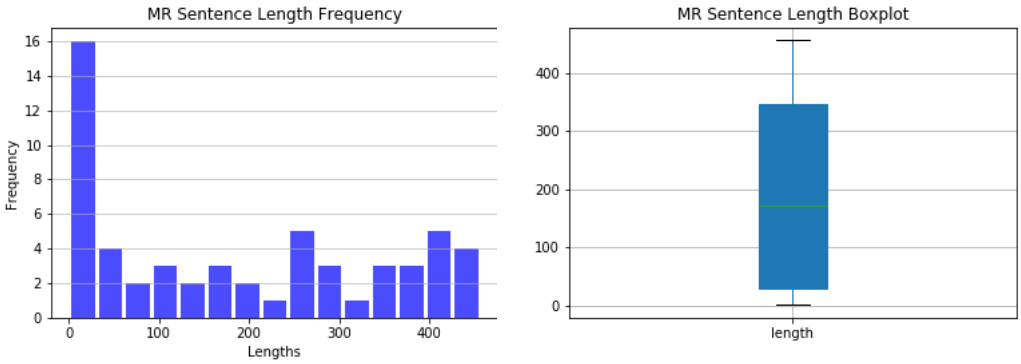
MR

Movie Review - a dataset with one sentence for each review and consists of collections of movie-review documents with their overall sentiment polarity (positive or negative) labels and sentences labeled with respect to their subjectivity status (subjective or objective) or polarity. This dataset is one of the most popular and commonly used in the state-of-the-art for sentiment analysis[1].

MR was introduced in "Proceedings of EMNLP" 2002[52] and until today was used more than a hundred publications. MR includes exactly 5331 positive and 5331 negative sentence, hence for the classification task, we have in total 10662 sentences.

[1]https://www.cs.cornell.edu/people/pabo/movie-review-data/

The following graphs display the length frequency of sentences and the distribution of MR.



As noted, more than 50 percent of all sentences are between 0 and 100 lengths, it means that the majority of sentences are short and informative.

SST-1, SST-2

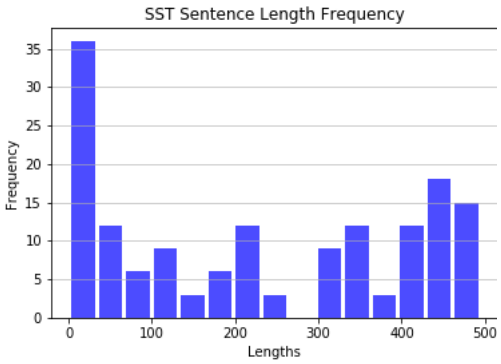
Stanford Sentiment Treebank - an extension of MR, however, splits into train/dev/test and provides fine-grained labels (very positive, positive, neutral, negative, very negative)[2].

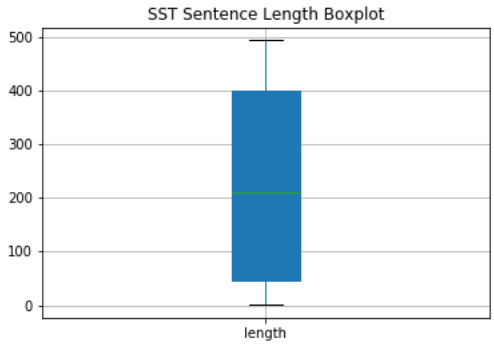
Indeed, lots of machine learning models do not use directly sentence level. Therefore, SST-1 provides also a different level of sentences - phase-level and we can train models on both phrases and sentences, but score on sentences at test time. Thus the training set is an order of magnitude larger than listed in the above table.

SST-2 is the same as SST-1, nevertheless, all neutral reviews deleted. This dataset is better for sentence classification.

The following graphs display the length frequency of sentences and the distribution of SST.

[2]http://nlp.stanford.edu/sentiment/





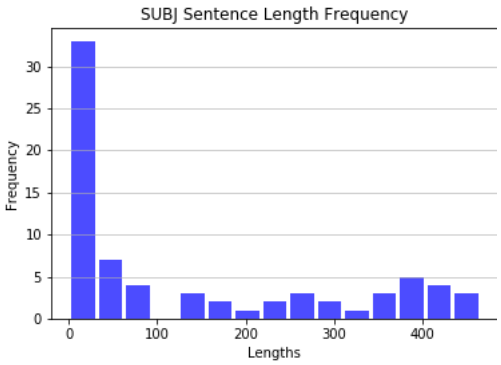
As noted, SST contains more sentences than MR, because lots of neutral reviews are added for sentiment-analysis task, consequently, SST is usually the first dataset to feed new machine learning models.

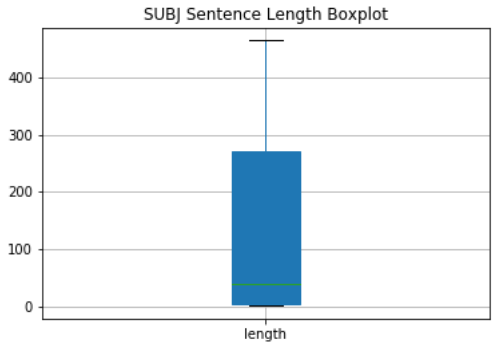
SUBJ

SUBJ - Subjectivity dataset[3] contains all data used for classification a sentence as being subjective or objective. This data was first used in Bo Pang and Lillian Lee, "A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts"[53].

The starting points for data acquisition were snippets of movie reviews from Rotten Tomatoes (http://www.rottentomatoes.com/) and plot summaries for movies from IMDB database. The dataset was split into "subj" and "obj" files, hence binary classification problem is considered.

The following graphs display the length frequency of sentences and the distribution of SUBJ.





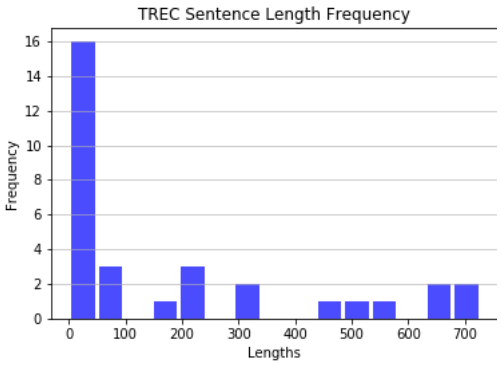
[3]https://www.cs.cornell.edu/people/pabo/movie-review-data/

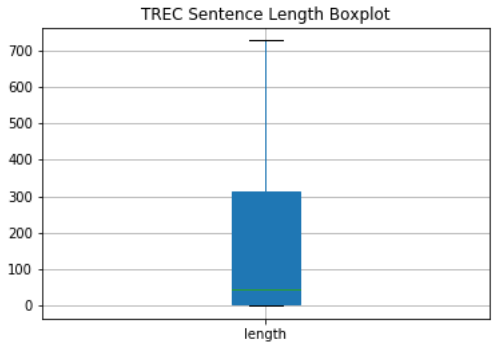
TREC

TREC question dataset—task involves classifying a question into 6 question types (whether the question is about a person, location, numeric information, etc.).

The first time was used by Xin Li and Dan Roth[16]. This data collection contains all the data used in classification experiments, which has question class definitions, the training and testing question sets, examples of preprocessing the questions, feature definition scripts and examples of semantically related word features.

The following graphs display the length frequency of sentences and the distribution of TREC.





There are 5 different classes:

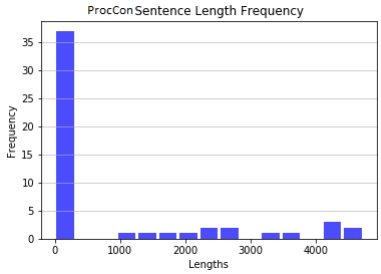
1. ABBR - abbreviation;
2. ENTY - entity: animal, body, currenty and etc;
3. DESC - description: definition, manner, reason and etc;
4. HUM - human: group, individual and etc;
5. LOC - city, country and etc;
6. NUM - code, count, date and etc.

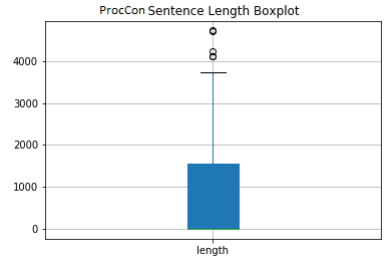
ProcCons

ProcCons is a dataset[4] that was used or determining context (aspect) dependent sentiment words.

[4]https://www.cs.uic.edu/ liub/FBS/sentiment-analysis.html

The following graphs display the length frequency of sentences and the distribution of ProcCons.



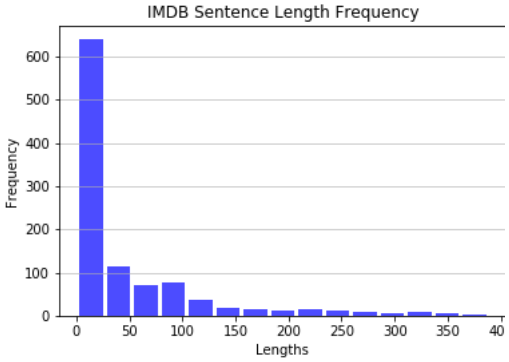


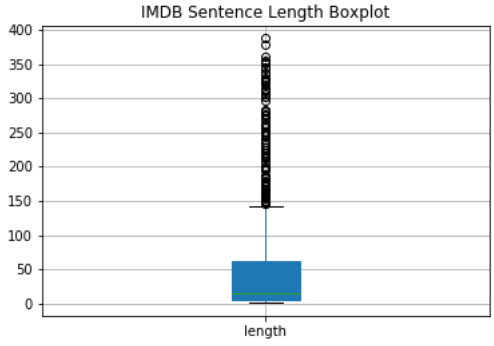
IMDB

Originally, IMDB dataset is formed from IMDB - an online database of information related to films, television programs, home videos, and video games, and internet streams, including cast, production crew, and personal biographies, plot summaries, trivia, and fan reviews and ratings.

The core dataset contains 50000 reviews split into 25000 train and 25000 test. The overall distribution of labels is balanced(25000 positive and 25000 negative). In this thesis, in order to use IMDB dataset properly all reviews shuffled and split to 45000 train and 5000 test sets.

The following graphs display the length frequency of sentences and the distribution of IMDB.





Within the entire collection, if a movie has more than 30 reviews they tend to have correlated ratings. Additionally, the train and test sets consist of a disjoint set of movies, therefore there is an opportunity to use cash during the train of machine learning models.

IMDB dataset is able to use for supervised and unsupervised learning. In the labeled train/test sets, a negative review has a score of < 5 out of 10,

and a positive review has a score > 6 out of 10. Thus reviews with more neutral ratings are not included in the train/test sets. In the unsupervised set, reviews of any rating are included and there is an even number of reviews > 4 and < 6.

4.1.2 Preprocessing

The majority of data in the real world is significantly incomplete and noisy, with high probability contains extraneous and redundant data and errors. Therefore, data preprocessing is of the most crucial part of a machine learning approach which helps transform the raw data to an appropriate format for the following feeding to machine learning models.

Tokenization

Tokenization is the step when the whole sentence will be split into small pieces or tokens. The text itself can be tokenized into paragraphs, each paragraph can be split into sentences, each sentence can be tokenized into words, etc. All datasets are already split into sentences, therefore only splitting into tokens is necessary.

Normalization

Normalization is also an important step in the preprocessing. Normalization consists of converting whole text to the same case (upper or lower), deleting punctuation, converting any Normalization puts all words on equal footing and allows processing to proceed uniformly.

Stemming

Stemming is the process of erasing affixes(suffix, prefix, infix, circumfix) in order to get a word stem.

$$read, reads, reading, readable \rightarrow read$$

Lemmatization

Lemmatization is also a part of normalization and can capture canonical forms based on a word's lemma. For example, stemming the word "worse" would fail to return its citation form (another word for lemma); however, lemmatization would result in the following:

$$worse \rightarrow bad$$

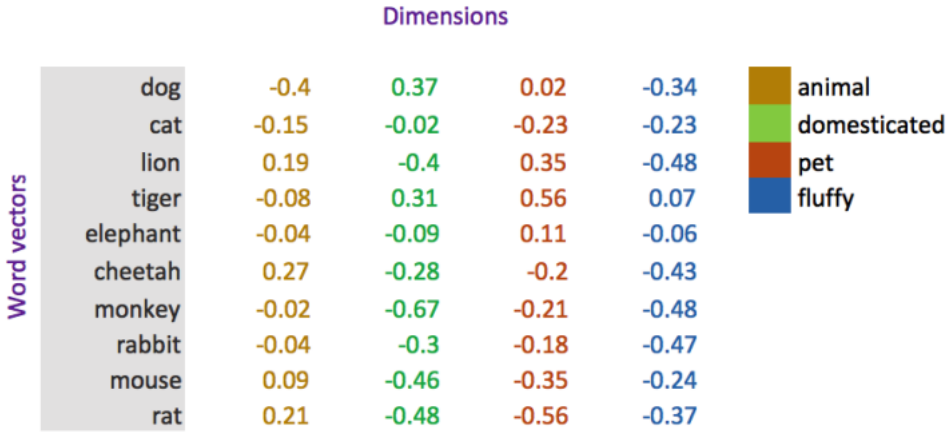
Stop words removal

Removing the most frequent words of the language from the source input text (i.e. the, at). Such words do not have a special meaning, because of this they can be neglected.

4.1.3 Embedding

Word embedding is of the most popular technique for NLP and the purpose is to learn low-dimensional vector representations of words from text[54]. The capability of capturing semantic and syntactic word relationships, word embedding algorithms such as Glove and Word2Vec has been proven to facilitate various NLP tasks, such as POS tagging, aspect extraction, and sentiment analysis. The crucial idea of distributional hypothesis, which states that words occurring in the same contexts tend to have similar meanings.

Actually, the word embedding represents the hidden relationship between words that can be used during training on the data.



Word embedding methods try to study a real-valued vector representation from a corpus of text with fixed vocabulary size. The learning process can be with the neural network task for text classification or can be an unsupervised process, using some text statistics.

In this thesis, for the first part of the approach*Embedding Layer* has been used within Capsule Neural Network. The word embedding is learned jointly with the model. The data should be clean and prepared so that each word is one-hot encoded.

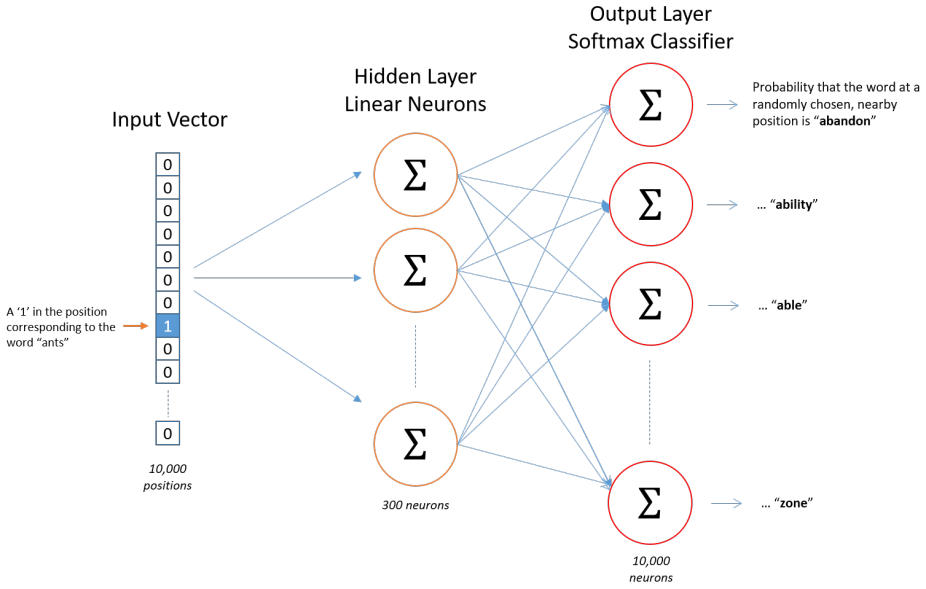
The size of the vector space can be tuned from 50D to 300D and the random values are used as default.

However, for the second part Glove, Word2Vec and Fasttext pre-defined word vectors are used and in the following sections will be described.

Word2Vec

Word2Vec is a statistical method for accurately studied a standalone word embedding from a text corpus. This method was introduced by Tomas Mikolov et al. at Google in 2013 as a response to make the neural-network-based training of the embedding more efficient and since then has become the de-facto standard for developing pre-trained word embedding.

The model is a two-layer neural network the processes text into vectors. Although Word2Vec is not a deep neural network but map the text into a feature space which deep neural networks can understand.



Measuring cosine similarity, no similarity is expressed as a 90-degree angle, while total similarity of 1 is a 0-degree angle, complete overlap; Here is a list of words associated with "Sweden" using Word2vec, in order of proximity:

Word	Cosine distance
norway	0.760124
denmark	0.715460
finland	0.620022
switzerland	0.588132
belgium	0.585835
netherlands	0.574631
iceland	0.562368
estonia	0.547621
slovenia	0.531408

Glove

The Global Vectors for Word Representation, or GloVe, the algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford[55]. This is an unsupervised learning algorithm for getting vector representations for words within a corpus of text.

Glove provides different models from 25, 50, 100, 200 and 300 dimensional base on 2, 6, 42, 840 billion tokens. For the construction of embedding, word-word co-occurrence probability is used i.e two words are co-existed much time, hence both words have similar meanings, hence the matrix will be closer.

X - is the matrix of word-word co-occurrence counts.

X_{ij} - the number of times word j occurs in the context of word i.

$P_{ij} = P(j|i) = X_{ij}/X_i$ be the probability that word j appear in the context of word i.

Consider two words $i =$ ice and $j =$ steam. The relationship of these words can be examined by studying the ratio of their co-occurrence probabilities with various probe words, k. For words k related to ice but not steam, say $k =$ solid, we expect the ratio P_{ik}/P_{jk} will be large.

Probability and Ratio	k = solid	k = gas	k = water	k = fashion
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Table 4.2: Co-occurrence probabilities for target words ice and steam with selected context words from a 6 billion token corpus

The above argument suggests that the appropriate starting point for word vector learning should be with ratios of co-occurrence probabilities rather than the probabilities themselves.

Noting that the ratio P_{ik} / P_{jk} depends on three words i, j, and k, the most general model takes the form:

$$F(w_i, w_j, w_k) = \frac{P_{ik}}{P_{jk}}$$

where $w \in \mathbb{R}^d$

FastText

Fasttext is a library for learning of word embedding and text classification created by Facebook's AI Research (FAIR) lab[56]. It is designed to be simple to use for developers, domain experts, and students. Its speed allows you to iterate quickly and refines your models without specialized hardware. FastText models can be trained on more than a billion words on any multicore CPU in less than a few minutes and can classify half a million sentences with hundreds of thousands of classes in less than a minute.

FastText can achieve better performance than the popular word2vec tool, or other state-of-the-art morphological word representations, and includes many more languages. FastText will receive future improvements from the FAIR team and FastText community making it more accessible.

	Corpora	MRPC	MR	ProcCons	SUBJ	Average
GloVe	Wiki+news	71.9/81.0	75.7	78.1	91.5	79.7
GloVe	Crawl	72.0/80.7	78.0	79.6	91.8	82.0
FastText	Wiki+news	72.9/**81.6**	77.8	80.3	92.2	82.5
FastText	Crawl	**73.4/81.6**	**78.2**	**81.1**	**92.5**	**82.7**

Table 4.3: Comparison of different pre-trained models on supervised text classification tasks

4.1.4 PCA: Dimensionality Reduction

At the moment, every day a huge amount of data is generated, which is not enough computational abilities to process everything. Therefore, data dimensional reduction algorithms have been developed such as the Principal Component Analysis(PCA). PCA is a technique that allows to reduce the data dimension and uses only unlabelled data. PCA[57] is a linear technique which transforms whole data into uncorrelated variables called *principal components*.

Assume, we need to reduce the dimension of data to k. In this case, PCA consists of six general steps:

1. Take the whole d-dimensional dataset;
2. Make a normalization with a mean of 0 and a standard deviation of 1:

$$\overline{x}_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}, \mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}, \sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2$$

$$\overline{X}_i = [\overline{x}_1 \overline{x}_2 \dots \overline{x}_n], X_{mean} = [\overline{X}_1 \overline{X}_2 \dots \overline{X}_n]$$

3. Calculate the co-variance matrix:

$$cov = \begin{pmatrix} \mu_{11}^2 & \mu_{12}^2 & \dots & \mu_{1n}^2 \\ \mu_{21}^2 & \mu_{22}^2 & \dots & \mu_{2n}^2 \\ \vdots & \vdots & \ddots & \vdots \\ \mu_{n1}^2 & \mu_{n2}^2 & \dots & \mu_{nn}^2 \end{pmatrix}$$

4. Calculate the eigenvalues and corresponding eigenvectors of cov:

$$\det(\lambda I - cov) = 0$$

   After solving this equation, the matrix will result in n possible values for λ: $\lambda_1, \lambda_2, \dots, \lambda_n$

   Sort all eigenvalues in descending order, than take the first k eigenvalues and computer corresponding eigenvectors:

$$(\lambda_i I - cov) * V_i = 0$$

   After all calculations we get eigenvectors: $V_1, V_2, \dots, V_k$

5. Compute the basis vectors:

$$X_b = X_{mean} * [V_1 V_2 \dots V_n]$$

6. Represent each sample as a linear combination of the basis vectors:

$$X_{new} = (X - \overline{X})^T * X_b$$

4.2 CapsNet with Dynamic Routing

A detailed description of the current neural network is covered in Capsule Neural Network with Dynamic Routing subsection in Background chapter.

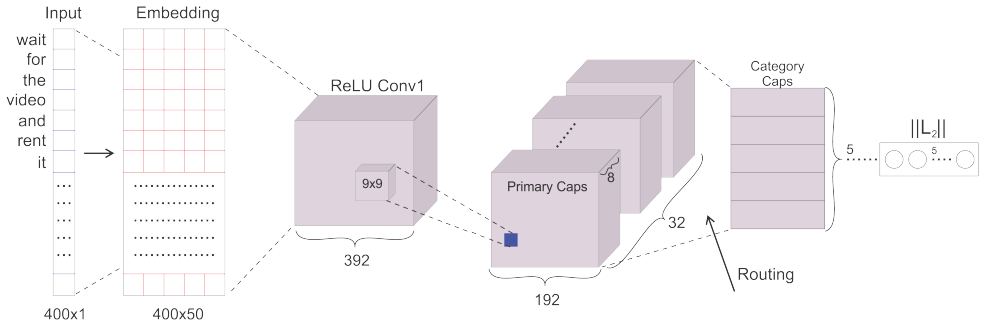


Figure 4.1: CapsNet Architecture for Text Classification

Figure (4.1) demonstrates the architecture that was changed.

Neural Networks work with digit inputs not with sentences, audio or video. Because of this, a possible solution is to encode text into digit inputs. There are different ways to encode text and one of however we use internal word embedding encode which means within CapsNet the input text is encoded. In this case, in order to represent a word as a vector, 400-dimensional size is used with internal embedding 50. In this case, the assumption is that such dimensions will be enough to effectively train CapsNet.

Consider the sentiment analysis with positive and negative classes then low-level capsules should catch the most important words in the text, which greatly affects the classification. After that, two high-level capsules(positive and negative classes) with using Dynamic Routing should detect dependencies between significant words and try to combine them in order to get the right prediction i.e to maximize their lengths.

Consequently, the assumption is that low-level capsules will detect significant words that significantly affect the classification of the text and high-level capsules will detect these capsules and maximize the prediction value.

2D convolution was used in the original architecture since the input is an image and it is significant to consider pixels because surrounding pixels bring additional information. However, in the case of text no need to consider surrounding pixels since it is not an image. In this case, the suggestion is to use 1D convolutional operation at the first and primary layers.

In the original architecture[7] the decoder structure reconstructs the input image and also the decoder is used as a regularization method. Nevertheless,

in text classification task, no reason to use it because the task is only to classify the input to pre-defined categories, therefore from the proposed architecture the decoder structure was removed. Instead of using the decoder as a regularization method, we use a dropout layer against overfitting.

Summarizing all that was said above, the following steps were made to the architecture of CapsNet:

1. **Added Internal Embedding Layer**:

   Embedding Layer allows to encode the input text as a vector which initialized with random weights and during the training phase is able to learn.

2. **1D Convolution Layer**:

   Since with Internal Embedding, any word is represented as a vector, no need to use two-dimensional convolution operation, therefore as a second layer in CapsNet 1D Convolutional Layer is used in order to reduce the computation of CapsNet.

3. **Added Dropout Layer**:

   As described in this section 2.4 dropout layer is a regularization method against overfilling. Since with internal embedding, this architecture has more than a million trainable parameters, Dropout Layer is necessary.

4. **Deleted Decoder**:

   Decoder structure in the original paper reconstructs the input image. However, in the case of text classification, there is no need to do this, thus decoder has been removed.

4.3 CapsNet with EM Routing

A detailed description of the current neural network is covered in Capsule Neural Network with EM Routing subsection in Background chapter.

Figure (4.2) demonstrates the architecture and the input is two-dimensional text representation with based on pre-trained vectors.

We used the original architecture[8] for text classification. Inspiration for using the same structure is from this paper[1]. In this case, CNN is adapted be preprocessing text with using pre-trained word embedding word2vec. Since original CapsNet with EM Routing is better than CNN for image classification, the assumption that the original architecture in text classification also would be better than CNN.

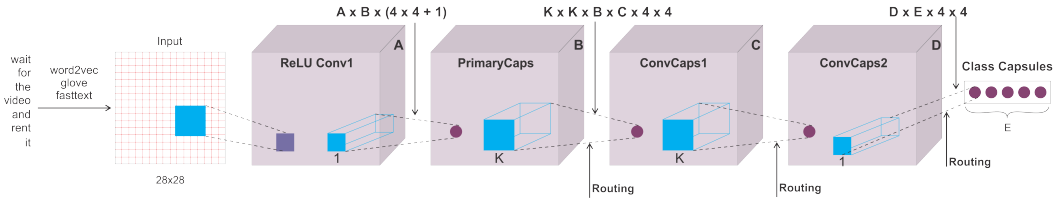


Figure 4.2: CapsNet Architecture for Text Classification

Since originally CapsNet with EM Routing was introduced for image classification therefore, in order for CapsNet with EM Routing to work, it is necessary to represent the text as an image. Because of this, Word2Vec, Glove, and Fasttext are used for word embedding.

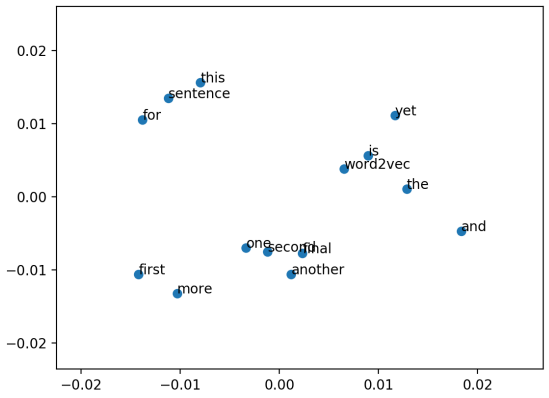


Figure 4.3: Glove Word Embedding[11]

Figure (4.3) illustrates the word embedding in 2-dimensional space. 300-dimensional pre-trained vectors of Word2Vec, Glove, and Fasttext were used in order to compare them. However, it is computationally expensive to calculate a convolutional operation because if the prepossessed sentence length is 20 words we got 20x300-tensor. Because of this, PCA was used for reducing the dimensional of word embedding. All datasets were divided into two groups: small datasets(total samples $\leq$ 11000) and big datasets(total samples about 50000). For small dataset, the dimensional of pre-trained word vectors were reduced to 28(as in the original paper) and for big datasets were reduced to 52. Since it was necessary to represent the sentence as a square image, padding was used and zero-vector $(0, 0, \ldots 0)$ was used for padding.

Consider the sentiment analysis with 5 different classes: positive, less

positive, neutral, more negative and negative. In this case, from layer to layer we collect the information in capsules and with EM Routing algorithm *construct* clusters where points represent the most important part of a text that will help to classify the whole sentence analogically to person recognition in figure (2.21).

EM Routing algorithm plays a significant role because it allows distinguishing clusters within detected words by low-level capsules. Since pre-trained word vectors allow to calculate the distance between two words and it allows reducing the training time. For instance, if we established, that the word *bad* belongs to the negative class, in this instance all words that are close to it with high probability belong to the negative class.

Chapter 5

Implementation

This chapter provides a detailed description of the implementation of the models which was introduced in Approach chapter. Firstly, the tools that were used, namely Python language with Scikit-Learnt, Keras and PyTorch libraries. Secondly, we illustrate in details how the data was prepared, i.e internal and external embedding. Finally, we depict CapsNet with Dynamic Routing by using Keras library and CapsNet with EM Routing by using PyTorch library.

5.1 Tools

Tools are used to develop, research and apply ANN for getting the performance on a real task. That is why the choice of the right tools is one of the key factors of the final performance.

In this section, Python language and Scikit-learn, Keras and PyTorch will be described in detail with examples of using.

5.1.1 Python

Python language is an interpreted, high-level programming language. It was introduced by Guido van Rossum and first released in 1991.

According to this article[58] Python is the best language for Data Science and Machine Learning. Python has a lot of different already integrated libraries which allow with minimal effort get the desired result.

Implemented a variety of functions for data preparation and visualization, construction of machine learning models including Deep Neural Networks. Furthermore, Python syntax is similar to the mathematical notation, therefore the threshold of entry is low.

In this thesis, the 3.5 version of the language was used.

5.1.2 Scikit-learn

Scikit-learn is an open-source library which is simple to use, effective enough to use it for Data Mining and Data Analysis[59]. It consists of six general parts: classification, regression, clustering, dimensionality reduction, model selection and preprocessing.

In this thesis, scikit-learn was mainly used for data processing and dimensional reduction.

5.1.3 Keras

Another open-source library is Keras which provides a high-level neural networks API[60], supported by Google.

The library was implemented with a focus on fast experimentation and making prototype of future models.

Most research associates prefer to use Keras library since the threshold of entry is very low and several hours are enough to create the first prototype of a neural network. Moreover, running experiments with CPU and GPU are available and well-written documentation is provided.

5.1.4 PyTorch

PyTorch[1] is an open-source machine learning library based on Torch[2], supported by Facebook. It provides two high-level features[3]:

1. Tensor computation (like NumPy) with strong GPU acceleration;
2. Deep neural networks built on a tape-based autograd system.

PyTorch enables fast, flexible experimentation and efficient production through a hybrid front-end, distributed training, and ecosystem of tools and libraries[4].

One of the advantages of using PyTorch instead of Keras is enough high level of abstraction, which allows to quickly implement the first prototypes and, simultaneously, a fairly low level of abstraction that gives an opportunity to implement the most complex structures of neural networks.

[1] `https://pytorch.org/`
[2] `http://torch.ch`
[3] `https://github.com/pytorch/pytorch`
[4] `https://pytorch.org/features`

5.2 CapsNet with Dynamic Routing

In this section, all layers of the neural network and their description will be shown in details. The full code can be found here[5].

The implementation of CapsNet with Dynamic Routing is presented below with Keras library.

```
from config import cfg
from keras import layers, models
from capsule_layers import CapsuleLayer, PrimaryCap, Length, Mask

def CapsNet(
    input_shape,
    n_class,
    num_routing,
    vocab_size,
    embed_dim,
    max_len):
    """
    :param input_shape: data shape
    :param n_class: number of classes
    :param num_routing: number of routing iterations
    :param vocab_size: the vocabulary size
    :param embed_dim: the embedding dimension size
    :param max_len: the maximum length of the input
    :return: a prediction of model
    """
    x = layers.Input(shape=input_shape)
    embed = layers.Embedding(vocab_size, embed_dim, input_length=max_len)(x)

    # Layer 1: Conv1D layer
    # Detects the basic features of the input
    conv = layers.Conv1D(
        filters=256,
        kernel_size=9,
        strides=1,
        padding='valid',
        activation='relu',
        name='conv1')(embed)

    # Layer 2: Dropout regularization
    # This method is againts overfitting
```

[5] https://github.com/khikmatullaev/CapsNet-Keras-Text-Classification

```
    dropout = layers.Dropout(cfg.regularization_dropout)(conv)

    # Layer 3: Primary layer with ‘squash‘ activation,
    # then reshape to [None, num_capsule, dim_vector]
    # Low-level capsules: detect significant words
    primary_caps = PrimaryCap(dropout,
        dim_vector=8,
        n_channels=32,
        kernel_size=9,
        strides=2,
        padding='valid',
        name="primary_caps")

    # Layer 4: Capsule layer. Dynamic Routing algorithm works here.
    # High-level capsules: based on Dynamic Routing detect
    # the class of the input
    category_caps = CapsuleLayer(
        num_capsule=n_class,
        dim_vector=16,
        num_routing=num_routing,
        name='category_caps')(primary_caps)

    # Layer 5: This is an auxiliary layer
    # To replace each capsule with its length.
    # The length of each capsule is the probability
    # that corresponding class is detected
    out_caps = Length(name='out_caps')(category_caps)

    return models.Model(input=x, output=out_caps)
```

5.3 CapsNet with EM Routing

In this section, all layers of the neural network and their description will be shown in details. Additionally, the significant part in this implementation is transforming text to 2-dimension tensor and using PCA for data reduction, this part of implementation is also provided.

The full code can be found here[6].

The implementation of CapsNet with EM Routing and Data Reduction are presented below with PyTorch library.

[6] https://github.com/khikmatullaev/CapsNet-EM-Matrix-Pytorch-Text-Classification

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import numpy as np
import math

class CapsNet(nn.Module):
    """
        :param A: output channels of normal conv
        :param B: output channels of primary caps
        :param C: output channels of 1st conv caps
        :param D: output channels of 2nd conv caps
        :param E: output channels of class caps (i.e. number of classes)
        :param K: kernel of conv caps
        :param P: size of square pose matrix
        :param iters: number of EM iterations
        ...
    """
    def __init__(self, A=32, B=32, C=32, D=32, E=10, K=3, P=4, iters=3):
        super(CapsNet, self).__init__()

        # Layer 1:
        # Detects basic features of the input
        self.conv1 = nn.Conv2d(
            in_channels=1,
            out_channels=A,
            kernel_size=5,
            stride=2,
            padding=2)

        # Layer 1 an activation of the output
        self.relu1 = nn.ReLU(inplace=False)

        # Layer 2: EM Routing works here
        # Detects low-level features and
        # Composes into low-level capsules
        self.primary_caps = PrimaryCaps(A, B, 1, P, stride=1)

        # Layer 3: EM Routing works here
        # Combines low-level capsules into high-level capsules
        self.conv_caps1 = ConvCaps(B, C, K, P, stride=2, iters=iters)
```

```
        # Layer 4: EM Routing works here
        # Combines high-level capsules
        self.conv_caps2 = ConvCaps(C, D, K, P, stride=1, iters=iters)

        # Layer 5:
        # Gets the final predictions
        self.class_caps = ConvCaps(D, E, 1, P, stride=1, iters=iters,
            coor_add=True,
            w_shared=True)

    # Forward function shows how to make the forward through the network
    def forward(self, x):
        x = self.conv1(x)
        x = self.relu1(x)
        x = self.primary_caps(x)
        x = self.conv_caps1(x)
        x = self.conv_caps2(x)
        x = self.class_caps(x)
        return x

def capsules(**kwargs):
    """Constructs a CapsNet model.
    """
    model = CapsNet(**kwargs)
    return model
```

5.3.1 Data Preprocessing: PCA

This is the implementation of data reduction. After data preparing it is necessary to reduce the dimension in order to increase the performance of the neural network.

```
    def prepare_x(x_text, dimension):
        """
            The function that returns reduced dimension
            from the vocabulary of the input dataset

            :param x_text: the input dataset
            :param dimension: the preferred dimension size
            :return: reduced pre-trained vectors with preferred
            dimension size
```

```
    """

    # Convert a collection of text documents
    # to a matrix of token counts
    vec = CountVectorizer(tokenizer=clean_str)
    vec.fit_transform(x_text)
    total_empty = 0

    # Take the full vocabulary from the input dataset
    vocab_names = vec.get_feature_names()
    vocab_values = np.zeros((len(vocab_names), 300))

    # Form the output array
    for i in range(len(vocab_names)):
        embedding_vector = embeddings_index.get(vocab_names[i])

        if embedding_vector is not None:
            vocab_values[i] = embedding_vector
        else:
            total_empty += 1

    # Use PCA for reduce dimension
    pca = PCA(n_components=dimension)
    vocab = pca.fit_transform(vocab_values)
    x_final = [clean_str(sent) for sent in x_text]

    # If the word is not in pre-trained vectors, full by zero-vector
    X = np.zeros((len(x_final), 1, dimension, dimension))

    for i in range(len(x_final)):
        x = x_final[i]
        text = np.zeros((dimension, dimension))

        for j in range(dimension):
            if j < len(x):
                if x[j] in vocab_names:
                    text[j] = vocab[vocab_names.index(x[j])]
            else:
                break

        X[i][0] = text

    return X, total_empty
```

Chapter 6

Evaluation

In the current chapter, the experimental setup will be described. Details are about the process of an evaluation with all datasets that were introduced Datasets section in Approach chapter. Finally, provides all final results for both proposed models with different hyper-parameters which is set for each of the models separately.

6.1 Experimental Setup

The training computations for both CapsNets have been run on operation system Ubuntu 16.04 with INTEL CORE i7 CPU, NVIDIA GeForce GTX 1080 Ti GPU and 32GM of RAM. The total amount of training time for CapsNet with Dynamic Routing is approximately 24 hours, however for CapsNet with EM Routing is roughly 40 hours.

6.2 Experiments

For the purpose of evaluating CapNets, the dataset is split into 3 parts:

1. **Train**: 70% of a dataset is used for training;
2. **Dev**: 15% of a dataset is used during training to get results;
3. **Test**: 15% is used for getting a final performance of a model.

All databases were divided into two groups:

1. **Small Datasets**: MR, SST-1, SST-2, SUBJ, TREC
2. **Big Datasets**: ProcCons, IMDB

All results in each Experiment were obtained for **Test** since the performance of any machine model is measured on data that was not used during the training phase.

6.2.1 Experiment 1: CapsNet with Dynamic Routing

In this experiment, 4 different hyper-parameters are used such as:

1. **o**: Optimizer:

   a - **Adam** optimizer: an optimization algorithm that updates the classical stochastic gradient descent based on adaptive estimates of lower-order moments which was introduced in 2015[61].

   na - **Nadam** optimizer: Nesterov Adam optimizer is a combination of Adam with Nesterov accelerated gradient(NAG) which is superior to vanilla momentum and was introduced in 2016[62].

2. **en**: Epoch Number: 10 or 20 epoch numbers were;

3. **bz**: Batch Size: 200, 500 batch sizes were used;

4. **lr**: Learning Rate Scheduler: a function that reduces a learning rate during the training phase in order to achieve a global minimum of a loss function.

   Three different functions were used:

   *l*1 - lambda1

   *l*2 - lambda2

   sd - step_decay.

```
def lambda1(epoch):
    return 0.001 * np.exp(-epoch / 10.)

def lambda2(epoch):
   initial_lrate = 0.1
   k = 0.1
   return k*initial_lrate/np.sqrt(epoch + K.epsilon())

def step_decay(epoch):
    initial_lrate = 0.1
    drop = 0.5
    epochs_drop = 5
    lrate = initial_lrate * pow(drop, floor((1+epoch)/epochs_drop))
    return lrate
```

o	en	bz	lr	MR	SST-1	SST-2	SUBJ	TREC
a	10	200	l1	71%	34%	**73.8%**	87.9%	73%
a	10	200	l2	20%	11%	50%	18%	3%
a	10	200	sd	0%	30%	52.2%	33%	10%
a	10	500	l1	66%	22%	72.5%	**89%**	43%
a	10	500	l2	27%	16%	51%	56%	3%
a	10	500	sd	45%/99%	12%	51,7%	12%	29%
a	20	200	l1	70%	33%	69.9%	82%	76%
a	20	200	l2	30%	25%	50.4%	28%	3%
a	20	200	sd	0%	16%	48.2%	44%/99%	30%
a	20	500	l1	69%	32%	72.3%	86%	61%
a	20	500	l2	3%	11%	49.7%	7%	7%
a	20	500	sd	58%	16%	50%	25%	10%
na	10	200	l1	**73%**	**34%**	71.6%	87%	75.5%
na	10	200	l2	26%	11%	51.4%	21%	3%
na	10	200	sd	0%	22%	48.2%	0%	3%
na	10	500	l1	57%	34%	73%	87%	50%
na	10	500	l2	0%	13%	52.7%	45%/99%	12%
na	10	500	sd	44%/100%	16%	53%	45%/99%	3%
na	20	200	l1	71%	34%	70%	84%	**76.5%**
na	20	200	l2	2%	18%	53%	25%	3%
na	20	200	sd	44%/100%	18%	46%	45%/99%	3%
na	20	500	l1	61%	34%	67%	82%	66%
na	20	500	l2	1%	14%	50%	13%	18%
na	20	500	sd	45%/99%	22%	46%	4%	3%
				73%	**34%**	**73.8%**	**89%**	**76.5%**

Table 6.1: Experiment 1: Small Datasets

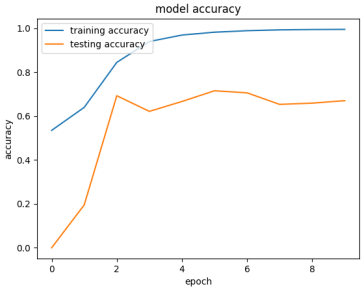
Table (6.1) demonstrates results of tuning hyper-parameters of CapsNet with Dynamic Routing for small datasets.

o	en	bz	lr	ProcCons	IMDB
a	10	200	l1	90.95%	81%
a	10	200	l2	29%	15%
a	10	200	sd	45%/100%	44%
a	10	500	l1	91.63%	79%
a	10	500	l2	19.42%	25%
a	10	500	sd	40%	45%/100%
a	20	200	l1	90.19%	82%
a	20	200	l2	37.23%	38%
a	20	200	sd	63.01%	39%
a	20	500	l1	90.65%	78%
a	20	500	l2	40.98%	31%
a	20	500	sd	45%/100%	11%
na	10	200	l1	90.43%	80%
na	10	200	l2	1.55%	30%
na	10	200	sd	18.79%	34%
na	10	500	l1	90.41%	79%
na	10	500	l2	45%/100%	25%
na	10	500	sd	40%/100%	15%
na	20	200	l1	90.24%	80%
na	20	200	l2	13.54%	30%
na	20	200	sd	40%/100%	40%/100%
na	20	500	l1	90.61%	77%
na	20	500	l2	29.01%	26%
na	20	500	sd	45%/100%	73%
				91.63%	**82%**

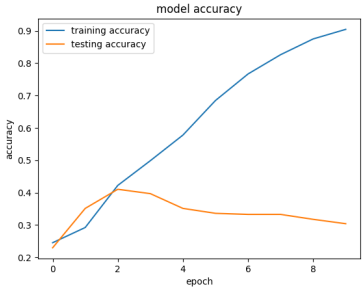
Table 6.2: Experiment 1: Big Datasets

Table (6.2) demonstrates results of tuning hyper-parameters of CapsNet with Dynamic Routing for big datasets.

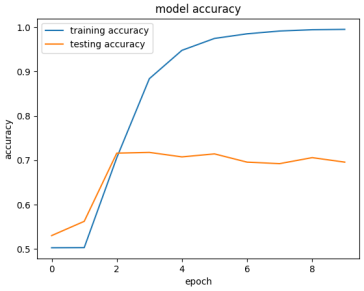
Red cells are those in which the CapsNet with hyper-parameters gives 100% accuracy on the test set and 45-50% on train set which cannot be served as a reliable result. This is an *underfitting* problem because the accuracy in the test set is higher than the accuracy in the train set.



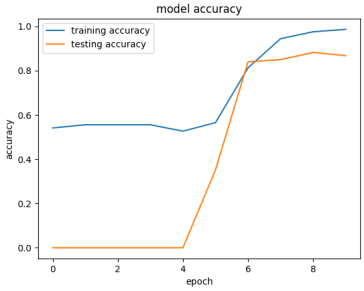
(a) MR



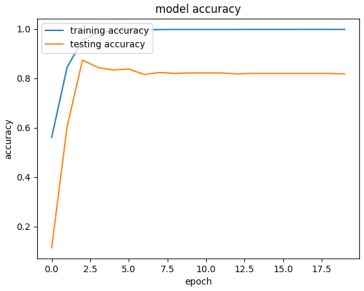
(b) SST-1



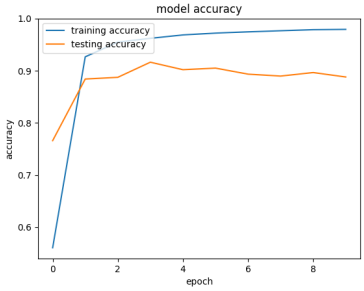
(c) SST-2



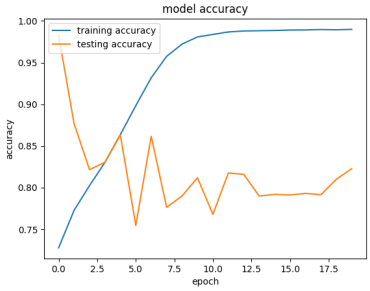
(d) SUBJ



(e) TREC

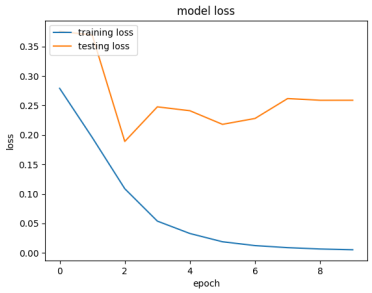


(f) ProcCons

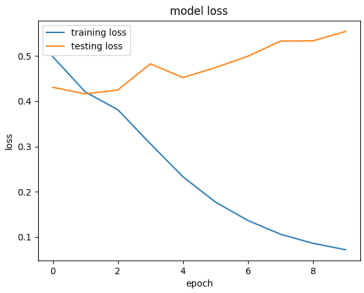


(g) IMDB

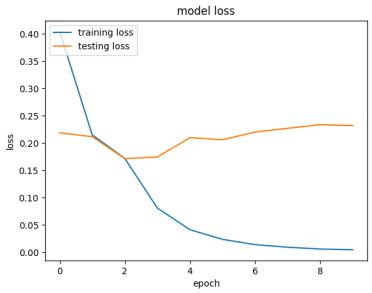
Figure 6.1: Experiment 1: Accuracy graphs with the best hyper-parameters



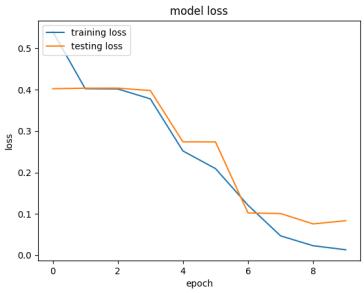
(a) MR



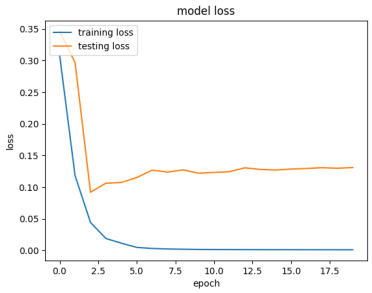
(b) SST-1



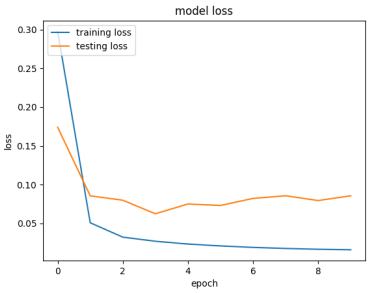
(c) SST-2



(d) SUBJ



(e) TREC



(f) ProcCons

(g) IMDB

Figure 6.2: Experiment 1: Loss graphs with the best hyper-parameters

6.2.2 Experiment 2: CapsNet with EM Routing

Features of PyTorch library and only one NVIDIA GeForce GTX 1080 Ti GPU did not allow conducting a full-fledged experiment(not well optimized and needs more GPUs) with the same hyper-parameters as in Experiment 1.

Because of that, this experiment was conducted only with a small dataset with fixed **Epoch Number = 20** and **Batch Size = 64**.

In this experiment, 3 different hyper-parameters are used such as:

1. **em**: Embedding:

   glove: the algorithm is an extension to the word2vec method for efficiently learning word vectors, developed by Pennington, et al. at Stanford[55];

   word2vec: a statistical method for accurately studied a standalone word embedding from a text corpus;

   fasttext: a library for learning of word embedding and text classification was created by Facebook's AI Research (FAIR) lab[56].

2. **o**: Optimizer:

   a - **Adam** optimizer: an optimization algorithm that updates the classical stochastic gradient descent is based on adaptive estimates of lower-order moments which was introduced in 2015[61].

   ag - **Adagrad** optimizer: an optimization algorithm that adapts the learning rate to the parameters, smaller updated for more frequent features and bigger update for infrequent features. It was introduced in 2011[63].

3. **lr**: Learning Rate Scheduler: a function that reduces a learning rate during the training phase in order to achieve a global minimum of a loss function.

   Two different functions were used:

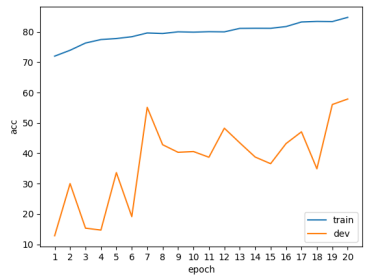
   rp - **ReduceLROnPlateau**: decrease learning rate when a metric has stopped improving.

   slr - **StepLR**: initialize the learning rate parameter with input *lr* and was reduced by *gamma* every step.

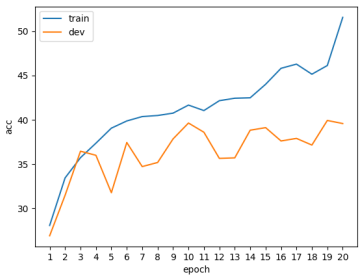
em	o	lr	MR	SST-1	SST-2	SUBJ	TREC
glove	a	rp	44.56%	39.17%	70.37%	78.66%	65.86%
glove	a	slr	43.00%	35.7%	69.38%	77.51%	65.74%
glove	ag	rp	44.12%	34.72%	66.72%	77.44%	65.62%
glove	ag	slr	47.50%	33.39%	66.49%	79.6%	64.9%
word2vec	a	rp	**57.87%**	**39.35%**	70.94%	75.88	59.61%
word2vec	a	slr	47.06%	38.25%	71.58%	74.04	61.89%
word2vec	ag	rp	49.43%	36.34%	71%	73.02%	60.57%
word2vec	ag	slr	50%	33.66%	70.08%	73.23%	59.61%
fasttext	a	rp	47.5%	35.99%	71.12%	**80.7%**	65.14%
fasttext	a	slr	47%	33.55%	**71.7%**	73.64%	**66.58%**
fasttext	ag	rp	47%	33.78%	70.25%	75.13%	64.9%
fasttext	ag	slr	49.56%	33.7%	69.38%	73.36%	62.98%
			57.87%	**39.35%**	**71.7%**	**80.7%**	**66.58%**

Table 6.3: Experiment 2: Small Datasets

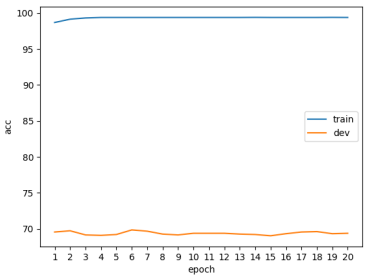
Table (6.3) demonstrates results of tuning hyper-parameters of CapsNet with EM Routing for small datasets.



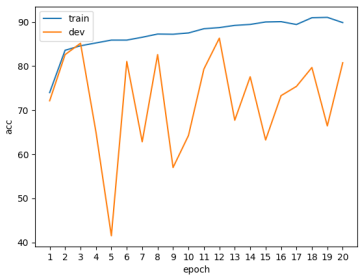
(a) MR



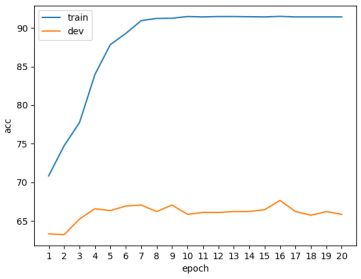
(b) SST-1



(c) SST-2

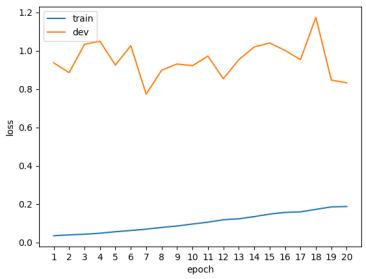


(d) SUBJ

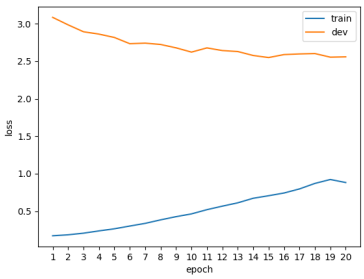


(e) TREC

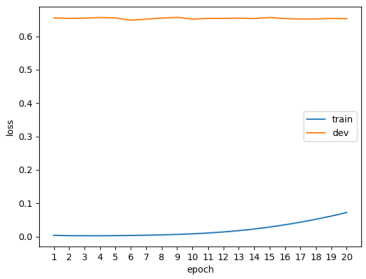
Figure 6.3: Experiment 2: Accuracy graphs with the best hyper-parameters



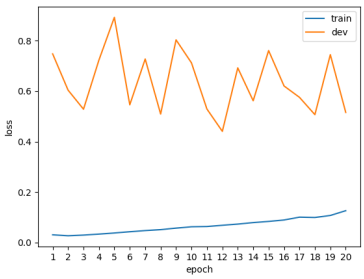
(a) MR



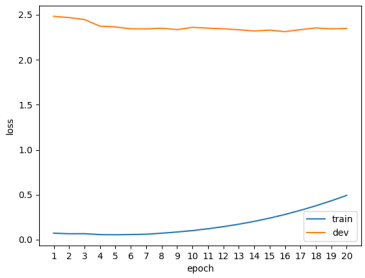
(b) SST-1



(c) SST-2



(d) SUBJ



(e) TREC

Figure 6.4: Experiment 2: Loss graphs with the best hyper-parameters

6.3 Discussion

In this section, we will compare the performance of CapsNets with a state-of-the-art result and discuss the results of CapsNet with Dynamic Routing and CapsNet with EM Routing. Also, possible reasons for such results are performed by CapsNets.

6.3.1 Discussion of the results

Models	MR	SST-1	SST-2	SUBJ	TREC	ProcCons	IMDB
SA by Caps[18]	**83.8%**	**49.3%**	-	-	-	-	-
CNN for SC[1]	81.1%	47%	88.1%	**90%**	91.2%	-	-
CRDLM for SC[64]	-	48%	**89.2%**	-	-	-	**93.4%**
CapsNet, DR	73%	34%	73.8%	89%	76.5%	**91.63%**	82%
CapsNet, EMR	57.87%	39.35%	71.7%	80.7%	66.58%	-	-

Table 6.4: Comparison with state-of-the-art

Table (6.4) demonstrates the comparison with different models.

As can be seen from the table above, only in ProcCons dataset the accuracy is desirable, however, in other cases, it is much less. Possible reasons will be described for each model below.

6.3.2 CapsNet with Dynamic Routing

In this implementation, the *overfitting* problem has been faced since the internal embedding was used for transformation text to vectors. Since the input text primarily represented by a 400-dimensional vector (with adding padding if is it necessary) and the internal embedding transforms it to 400x50 tensor.

An important fact is that this embedding layer is trainable, therefore, the total number of trainable parameters are close to 1.2 million. As it can be seen from the table above CapsNet with Dynamic Routing performs worse accuracy in small datasets. Nonetheless, in big datasets, the accuracy is higher and close to the state-of-the-art results.

Also, the mistake was that 400 dimension input layer was used in order to encode a vocabulary in each dataset, since after analyzing all datasets it was found that for small datasets the average text length was 10 and for one big dataset(IMDB) about 120. However, when choosing the input dimension, the average lengths for other databases was not taken into account and hence lot's of 0 values in an input vector.

The overfitting problem is often encountered in practice since there is not always a sufficient amount of data and often machine learning models are too complex. In order to solve it, a dropout layer was used as a regularization method and even with **90%** of randomly disabling neurons within layer the accuracy of CapsNet did not increase. One of the solutions for the overfitting problem is to collect more data in the domain of the dataset and then try to train CapsNet with Dynamic Routing with collected data. Against the overfitting problem another possible method is to use L1(add the absolute value of the parameters) or L2(add the squared value of the parameters) regularization that penalizes the objective function thereby reduce the complexity of CapsNet.

Another possible solution to reduce the number of trainable parameters is to use pre-trained word vectors like Glove, Word2Vec or Fasttext as it was used for the second experiment with CapsNet with EM Routing.

Further, it is important to recall about using 1D convolution operation. In the case of internal embedding, there is no guarantee that between two close words the cosine distance will be close to zero. Therefore, 1D convolution operation was used, however, the internal embedding is learned during the training phase, therefore probably 2D convolution operation will improve the performance because it considers also surround pixels which store additional useful information after the training phase.

6.3.3 CapsNet with EM Routing

For this implementation, the preprocessing part was significant since 300-dimensional pre-trained word vectors were used for text transformation. Nevertheless, there was a limitation of using only one NVIDIA GeForce GTX 1080 Ti GPU, therefore, it was necessary to reduce 300-dimensional pre-trained word vectors to 28-dimensional for small datasets(as well as in the publication) and 52-dimensional for big datasets.

One of the most important reasons for unsatisfactory results was the incorrect choice of the final dimensional size of pre-trained word vectors, as it can be seen from the table (4.1) in Datasets section that the average length size for small datasets is 10. In this case, zero-padding is used which does not add meaningful information. Furthermore, we lost the information

from pre-defined word vectors when we used PCA which means that the distance measure did not work properly. Because of this, the clustering by EM Routing did not partition clusters. And in order to train this complex architecture, it is necessary to have longer text and do not reduce the pre-trained word embedding.

The second reason is that the original CapsNet with EM Routing was introduced for image recognition, on that account, consists of a lot of high-level matrix capsule layers and as a result, the proposed model is too complex for text classification. For instance, for the person recognition (2.22) is it necessary to recognize eyes, mouth, torso, and hand. Then by using EM Routing represents a face as a combination of eyes and a month and a body as a combination of a torso and a hand. Finally again with EM Routing represents a person as a combination a face with a body. However, in the case of text classification, there is no need for several numbers of layers in CapsNet.

One possible solution is to reduce the number of high-level matrix capsule layers to two: one low-level capsules layer which will detect significant words within input text and one high-level capsule layer that detects dependencies between significant words and tries to combine them in order to get the right prediction(analogically with CapsNet with Dynamic Routing). Analogically with this paper[1] they used only one convolution layer and one max-pooling layer in proposed CNN. It is also worth noting that reducing the number of capsule layers will decrease the training time because between capsule layers EM Routing algorithm spends a lot of time, because of a number of iterations.

Despite the 66,000 total number of parameters, the regularization method is also able to use. For instance, the dropout layer between low-level capsule layer and high-level capsule layer. However, in this case, we will have to think about how it will affect on EM Routing algorithm and therefore it will be better to use L1 or L2 regularization which penalize the loss function of CapsNet in order to reduce the complexity of the proposed model.

Chapter 7

Conclusion

In this chapter, a summary of the thesis is presented. It discusses adapting Capsule Neural Network with Dynamic Routing and Capsule Neural Network with EM Routing architectures for text classification problem and results of the experiments. Finally, some suggestions are about the future direction of research.

7.1 Summary

Text classification problem is one of the central problems in Natural Language Processing. Consequently, success in solving this problem will greatly affect real-life applications such as Question Answering and Document Analysis. For this reason, different approaches from statistical methods to a new type of neural networks are used for getting better performance on all datasets. Thus, the goal of this thesis is to adapt Capsule Neural Network which is the latest type of a neural network that was introduced by Geoffrey Hinton.

We introduced seven different datasets and conventionally split into two groups: small datasets(total samples ≤ 11000) and big datasets(total samples about 50000). In preprocessing part we removed all stop words(the, a, etc) and do tokenization, normalization, lemmatization and stemming. After cleaning all datasets for CapsNet with EM Routing we represented text as an image by using pre-trained word vectors glove, word2vec, and fasttext. Since there is a computational restriction(only one GPU) we used PCA for data dimension reduce.

In CapsNet with Dynamic Routing we changed the architecture: added internal embedding layer(encode text), used 1D convolutional operation, added dropout layer and removed a decoder structure. For the big datasets, the results are satisfactory(close to the state-of-the-art results). For the small

datasets, we got unsatisfactory results since faced the overfitting problem. Also using 1D convolution operation did not help to increase the performance of CapsNet. One of the reasons for the overfitting problem was using internal embedding because it is a trainable layer.

In CapsNet with EM Routing, we left the original architecture[8]. Moreover, for text encoding, we used pre-trained word vectors glove, word2vec, and fasettext for representing all datasets as a collection of images. For the small datasets, we got unsatisfactory results since faced with overfitting. The first reason for it is that the architecture of CapsNet with EM Routing too complex since there are three high-level capsule layers. The second reason is using PCA for reducing of dimension, because during the process of reducing we lost additional information in vectors of words, therefore, EM Routing algorithm decreased the performance of the proposed model.

7.2 Future Work

For Capsule Neural Network with Dynamic Routing the possible direction of research is to use pre-trained word vectors like glove, word2vec or fasttext for data preparation instead of internal trainable embedding layer for CapsNet with Dynamic Routing.

For Capsule Neural Network with EM Routing, the possible direction of research is a simplification of the architecture, since the original one is too complex for the text classification task. First of all, it is necessary to reduce the number of high-level capsule layer to two: one for low-level capsule layer and one for high-level capsule layer. Moreover, it is vital to use regularization methods like dropout layer or L1 and L2 regularization.

It is also necessary to analyze the accuracy and loss of Capsule Neural Network with Dynamic Routing and Capsule Neural Network with EM Routing for as large datasets as possible in order to be the state-of-the-art solution.

Bibliography

[1] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[2] K. Singh, A. S. Radhakrishna, A. Both, S. Shekarpour, I. Lytra, R. Usbeck, A. Vyas, A. Khikmatullaev, D. Punjani, C. Lange, M. Vidal, J. Lehmann, and S. Auer, "Why reinvent the wheel: Let's build question answering systems together," in *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, pp. 1247–1256, 2018.

[3] M. Pechyonkin, "Understanding hinton's capsule networks. part 1. intuition.," 2017.

[4] J. H. blog, ""understanding matrix capsules with em routing (based on hinton's capsule networks)"," 2017.

[5] M. Pechyonkin, "Understanding hinton's capsule networks. part 2. how capsules work.," 2017.

[6] M. Pechyonkin, "Understanding hinton's capsule networks. part 3. dynamic routing between capsules.," 2017.

[7] S. Sabour, N. Frosst, and G. E. Hinton, "Dynamic routing between capsules," in *Advances in neural information processing systems*, pp. 3856–3866, 2017.

[8] G. E. Hinton, S. Sabour, and N. Frosst, "Matrix capsules with em routing,"

[9] C. Lüth, "Matrix capsules with em-routing,"

[10] A. Amidi, "Unsupervised learning cheatsheet," 2017.

[11] J. Brownlee, "How to develop word embeddings in python with gensim," 2017.

[12] H. Han, E. Manavoglu, C. L. Giles, and H. Zha, "Rule-based word clustering for text classification," in *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Informaion Retrieval*, SIGIR '03, (New York, NY, USA), pp. 445–446, ACM, 2003.

[13] H. Li and K. Yamanishi, "Text classification using esc-based stochastic decision lists," in *Proceedings of the Eighth International Conference on Information and Knowledge Management*, CIKM '99, (New York, NY, USA), pp. 122–130, ACM, 1999.

[14] W. W. Cohen, "Fast effective rule induction," in *In Proceedings of the Twelfth International Conference on Machine Learning*, pp. 115–123, Morgan Kaufmann, 1995.

[15] S.-B. Kim, K.-S. Han, H.-C. Rim, and S. H. Myaeng, "Some effective techniques for naive bayes text classification," *IEEE Trans. on Knowl. and Data Eng.*, vol. 18, pp. 1457–1466, Nov. 2006.

[16] X. Li and D. Roth, "Learning question classifiers," in *Proceedings of the 19th international conference on Computational linguistics-Volume 1*, pp. 1–7, Association for Computational Linguistics, 2002.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[18] Y. Wang, A. Sun, J. Han, Y. Liu, and X. Zhu, "Sentiment analysis by capsules," in *Proceedings of the 2018 World Wide Web Conference*, WWW '18, (Republic and Canton of Geneva, Switzerland), pp. 1165–1174, International World Wide Web Conferences Steering Committee, 2018.

[19] J. Kim, S. Jang, S. Choi, and E. L. Park, "Text classification using capsules," *CoRR*, vol. abs/1808.03976, 2018.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, vol. 86, pp. 2278–2324, 1998.

[21] K. Singh, I. Lytra, A. S. Radhakrishna, A. Vyas, and M.-E. Vidal, "Dynamic composition of question answering pipelines with frankenstein,"

in *The 41st International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '18, (New York, NY, USA), pp. 1313–1316, ACM, 2018.

[22] K. Singh, A. Both, A. S. Radhakrishna, and S. Shekarpour, "Frankenstein: A platform enabling reuse of question answering components," in *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, pp. 624–638, 2018.

[23] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, no. 4, pp. 115–133, 1943.

[24] R. Kruse, C. Borgelt, F. Klawonn, C. Moewes, M. Steinbrecher, and P. Held, *Computational Intelligence - A Methodological Introduction.* Texts in Computer Science, Springer, 2013.

[25] D. O. Hebb *et al.*, "The organization of behavior," 1949.

[26] B. Widrow and M. E. Hoff, "Adaptive switching circuits," in *1960 IRE WESCON Convention Record, Part 4*, (New York), pp. 96–104, IRE, 1960.

[27] A. Ng, "Cs229 lecture notes - supervised learning." 2012.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–, Oct. 1986.

[29] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[30] D. H. Hubel and T. N. Wiesel, "Receptive fields and functional architecture of monkey striate cortex," *Journal of Physiology (London)*, vol. 195, pp. 215–243, 1968.

[31] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological Cybernetics*, vol. 36, pp. 193–202, 1980.

[32] T. Serre, L. Wolf, S. M. Bileschi, M. Riesenhuber, and T. A. Poggio, "Robust object recognition with cortex-like mechanisms.," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 29, no. 3, pp. 411–426, 2007.

[33] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.

[34] N. Bourdakos, "Capsule networks are shaking up ai—here's how to use them," 2017.

[35] J. H. blog, "Understanding dynamic routing between capsules (capsule networks)," 2017.

[36] D. De, "What is a capsnet or capsule network?," 2017.

[37] M. Pechyonkin, "Understanding hinton's capsule networks. part 4. capsnet architecture.," 2018.

[38] S. Yong-feng and Z. Yan-ping, "Comparison of text categorization algorithms," *Wuhan University Journal of Natural Sciences*, vol. 9, pp. 798–804, Sep 2004.

[39] D. D. Lewis, "Naive (bayes) at forty: The independence assumption in information retrieval," in *Machine Learning: ECML-98* (C. Nédellec and C. Rouveirol, eds.), (Berlin, Heidelberg), pp. 4–15, Springer Berlin Heidelberg, 1998.

[40] A. McCallum, K. Nigam, *et al.*, "A comparison of event models for naive bayes text classification," in *AAAI-98 workshop on learning for text categorization*, vol. 752, pp. 41–48, Citeseer, 1998.

[41] G. Guo, H. Wang, D. Bell, Y. Bi, and K. Greer, "Knn model-based approach in classification," in *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE* (R. Meersman, Z. Tari, and D. C. Schmidt, eds.), (Berlin, Heidelberg), pp. 986–996, Springer Berlin Heidelberg, 2003.

[42] L.-Y. Hu, M.-W. Huang, S.-W. Ke, and C.-F. Tsai, "The distance function effect on k-nearest neighbor classification for medical datasets," *SpringerPlus*, vol. 5, no. 1, p. 1304, 2016.

[43] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*, vol. 39. Cambridge University Press, 2008.

[44] E. Aramaki, T. Imai, K. Miyo, and K. Ohe, "Patient status classification by using rule based sentence extraction and bm25 knn-based classifier," in *i2b2 Workshop on Challenges in Natural Language Processing for Clinical Data*, 2006.

[45] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, Inc., 1 ed., 1997.

[46] M. Mehta, R. Agrawal, and J. Rissanen, "Sliq: A fast scalable classifier for data mining," in *Advances in Database Technology — EDBT '96* (P. Apers, M. Bouzeghoub, and G. Gardarin, eds.), (Berlin, Heidelberg), pp. 18–32, Springer Berlin Heidelberg, 1996.

[47] P. Vateekul and M. Kubat, "Fast induction of multiple decision trees in text categorization from large scale, imbalanced, and multi-label data," *2009 IEEE International Conference on Data Mining Workshops*, pp. 320–325, 2009.

[48] D. E. Johnson, F. J. Oles, T. Zhang, and T. Goetz, "A decision-tree-based symbolic rule induction system for text categorization," *IBM Systems Journal*, vol. 41, no. 3, pp. 428–437, 2002.

[49] V. Vapnik, *The nature of statistical learning theory*. Springer science & business media, 2013.

[50] T. Joachims, "Text categorization with support vector machines: Learning with many relevant features," in *European conference on machine learning*, pp. 137–142, Springer, 1998.

[51] J. B. Chrystal and S. Joseph, "Multi-label classification of product reviews using structured svm,"

[52] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," in *Proceedings of EMNLP*, pp. 79–86, 2002.

[53] B. Pang and L. Lee, "A sentimental education: Sentiment analysis using subjectivity summarization based on minimum cuts," in *Proceedings of the ACL*, 2004.

[54] Y. Li, Q. Pan, T. Yang, S. Wang, J. Tang, and E. Cambria, "Learning word representations for sentiment analysis," *Cognitive Computation*, vol. 9, no. 6, pp. 843–851, 2017.

[55] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation.," in *EMNLP*, vol. 14, pp. 1532–1543, 2014.

[56] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *Proceedings*

of the International Conference on Language Resources and Evaluation (LREC 2018), 2018.

[57] X. Wei, M. Kleinsteuber, and H. Shen, "Invertible nonlinear dimensionality reduction via joint dictionary learning," in *Proceedings of the 12th International Conference on Latent Variable Analysis and Signal Separation - Volume 9237*, LVA/ICA 2015, (Berlin, Heidelberg), pp. 279–286, Springer-Verlag, 2015.

[58] "What is the best programming language for machine learning?." https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7. Accessed: 2018-07-14.

[59] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[60] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[61] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[62] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[63] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[64] A. Hassan and A. Mahmood, "Convolutional recurrent deep learning model for sentence classification," *IEEE Access*, vol. 6, pp. 13949–13957, 2018.