Expanding on the end-to-end memory network for goal-oriented dialogue

Peter Arentz Taraldsen and Vegard Vatne

SUPERVISORS
Raheleh Jafari
Morten Goodwin

Master's Thesis
University of Agder, 2019
Faculty of Engineering and Science
Department of ICT

UiA
University of Agder
Master's thesis

Faculty of Engineering and Science
Department of ICT

Abstract

A series of end-to-end models have been proposed in order to satisfy the requirements of the Dialog System Technology Challenge: building an end-to-end dialog system for goal-oriented applications. While these models have proven to be a good solution for such tasks, they perform worse when dealing with out-of-vocabulary tasks and none-synthetic data. Additionally, they rely heavily on the use of an underlying knowledge base to achieve good results.

We propose two new models that build on the end-to-end memory network architecture. The goal of these two models is to better handle out-of-vocabulary tasks and none-synthetic data. The first model changes the bag-of-words representation of the data, into a paragraph vector representation treating all data sentences as unseen sentences. We call this model the distributed bag-of-words end-to-end memory network (DbowN2N). The second model adds a bidirectional long short-term memory layer at the beginning of the model used for named entity recognition, to capture the keywords in the sentences before feeding it through the memory network. We call this network the key-tagging end-to-end memory network (KTN2N).

In our experiments, the DbowN2N model achieves similar results to that of the state of the art regular memory network, suggesting that bag-of-words representation of the sentences are as effective as Distributed bag-of-words representations for dealing with tasks like this. The KTN2N model achieves a considerable increase in accuracy over the plain memory network and comparable results with state of the art memory networks such as the gated memory network and unified weight-tying memory network.

Preface

Expanding on the end-to-end memory network for goal-oriented dialogue is a master thesis at the University of Agder. This project was suggested by Raheleh Jafari. We would like to thank Raheleh Jafari for her tremendous work guiding us through this project. Additionally, we would like to thank Dr. Morten Goodwin for his valuable feedback on this project.

Peter Arentz Taraldsen and Vegard Vatne
Grimstad, May 2019

Table of Contents

Glossary

binary classification problems Binary classification problems consider assigning an individual to one of two categories, by measuring a series of attributes [1]. 19, 22

deep learning Learning techniques allowing transformation of raw input into higher level abstract representations for non-linear modules [2]. 19, 21

feedforward neural networks Model containing computational neurons connected with weights arranged in a layer-by-layer basis [3]. 19

gradient a vector of slopes (derivatives/partial derivatives) for each dimension in the input space [4]. 28

gradient saturation The linear value of neurons become either very big or very small causing slow updates during training. 19, 21, 55

hidden layers A layer of neurons which is only connected to the input of other neurons. xii, 20

nonlinear function A function that is not linear (does not follow a straight line). 18, 19

one-stage object detection Predicting object in image without filtering first, as done in two stage object detection. 10

probability distribution A distribution of values between 0 and 1 where the sum of all the values equals to 1. 9, 10, 12, 18, 22

recurrent neural network Recurrent neural networks perform the same task for every element of a sequence, with the output being depended on the previous computations. [5]. 21

root mean square square root of the average of set values squared. 33

shallow network A neural network consisting of a low number of hidden layers. 19

support Vector Machine Classifyer separating labels with a hyperplane [6]. 11

List of Figures

List of Tables

List of Publications

1. Taraldsen, P.A., Vatne, V., Jafari, R., Goodwin, M., Granmo, O-C. (2019): End-to-End Memory Networks: A Survey. In: 19th Annual UK Workshop on Computational Intelligence, September 4-6, 2019, Portsmouth, United Kingdom (under review).

2. Vatne, V., Taraldsen, P.A., Jafari, R., Goodwin, M., Granmo, O-C. (2019): Dialogue Systems using End-To-End Memory Networks: Divorce Bot. In: Workshop: Chatbots for Social Good, September 3, 2019, Paphos, Cyprus (under review).

3. Taraldsen, P.A., Vatne, V., Jafari, R., Goodwin, M., Granmo, O-C. (2019): A novel structure for end-to-end memory networks in a goal-oriented domain. In: Journal of Intelligent & Fuzzy Systems (planning to submit).

Part I

Research Overview

Chapter 1

Introduction

There are many benefits of using goal-oriented dialog systems and their applications can be used in a wide range of domains. Among them is the case of children of divorce, as it is a little researched domain and could benefit from automation. The child may wish to gain certain legal information regarding their circumstance and the system will query a database for the appropriate information.

Constructing rule-based systems for goal-oriented dialogue is very effective and not particularly challenging when dealing with synthetic data. However, when dealing with real-world data the task becomes exponentially more difficult. This can be remedied by using end-to-end networks, where all components are learned from the dialogues themselves. In [16] Bordes et al. introduce a testbed to train and evaluate goal-oriented dialog systems. The goal of the dialog is to make a restaurant reservation based on a series of user queries. A series of different learning systems has been proposed in order to tackle this problem and among them is the end-to-end memory network, which produced promising results.

End-to-End Memory networks, which stores dialog information into a writable memory, shows high potential in solving goal oriented dialog problems. From it, a series of iterations have been developed, such as gated memory network and unified weight-tying memory network.

The work on this thesis has resulted in 2 publications, which are currently under review. Additionally, we plan another publication where we combine

the two proposed architectures described in chapter 4.

1.1 Problem statement

State of the art End-to-End Memory Networks has proven to be successful when applied to goal oriented dialog task for synthetically generated data, which can be seen in Table 3.1. The state of the art models results are above 99 % for test. However, the model has worse performance when operating with real-world data. Also, when the networks were dealing with entity words previously not seen under training, the accuracy scores would drop considerably.

1.2 Thesis Definition

We aim to solve Memory Networks current issues by adding extensions to the model. Our theory is that changing the way data is represented in the memory network may allow it to surmise a more accurate meaning of the queries. We have two proposed methods to achieve this. First is to use a long short-term memory layer to identify keywords and predict their meaning. Secondly, we will attempt to change the vector representation of the sentences from a traditional bag-of-words representation to a paragraph vector representation. We believe this could result in the network being able to infer meaning to previously unseen words based on its surrounding words. This will hopefully result in better results when dealing with the out-of-vocabulary problem and when dealing with non-synthetically generated data.

Lastly, correctly tuning the model should slightly improve the overall accuracy of the model, so finding the best methods for learning is imperative.

1.3 Report Outline

Chapter 2 presents the underlying loss and optimization functions often used in learning networks. Chapter 3 presents the memory network which our work originates from. Additionally, we look at some of the more successful modified versions of the memory network, gated memory network and unified gate-tying memory network and compare their findings. Furthermore we will explore the use of memory networks in the domain of children of divorce. In chapter 4 we will present our two proposed models; distributed bag-of-words memory network and bi-Lstm memory network. In chapter 5 we will present the datasets used and our results. In chapter 6 we will summarize our findings and present our thoughts on them.

Chapter 2

Optimization algorithms

There are different ways of optimizing the training of a prediction model. Machine learning can be applied to a vast amount of different domains. As different domains may require different machine learning algorithms, they may also be improved by how the machine learning algorithm is optimized. Choosing the best suited algorithms may improve results and efficiency. This chapter will explore some of the loss functions and optimization algorithms that are commonly used to train machine learning algorithms.

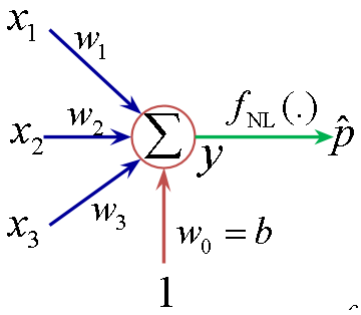

Figure 2.1: A single layer perceptron [7]

Figure 2.1 shows how each input in a feature vector [x1, x2, x3] is assigned a weight in a single layer perceptron. These weights are summarized to produce the output prediction. Some of these weights are more important

than others and therefore needs to have a higher value. It is necessary to have some functions that are able to find how wrong the current weights are (loss functions), and how they should be updated (Optimizers) [7].

2.1 Loss functions

Loss functions are used in machine learning to measure the error of a prediction in comparison to the true label (correct output value of the input data), and is in most cases the distance between the prediction and the true label [7].

$$J(W) = p - \hat{p} \tag{2.1}$$

W can be interpreted as the input weights of a neural network layer. There are multiple loss functions that return different loss and therefore might impact the training considerably. Loss functions fall in the different categories, as shown in Figure 2.2 [8].

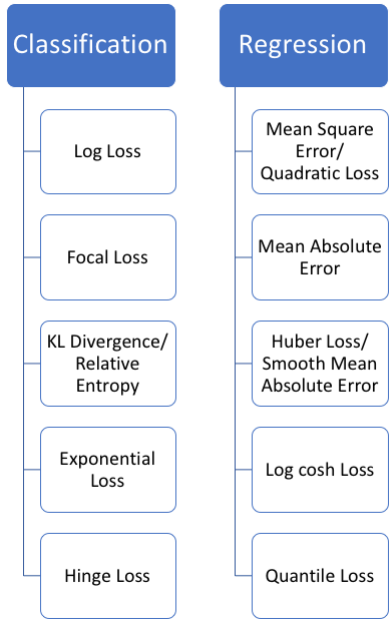

Figure 2.2: Loss function categories [8]

Classification task in machine learning is to approximate a mapping function from input variables to discrete output variables (called classes or categories) [20]. For example, a classification task can approximate a function of some input to a specific color in a set of colors.

Regression task in machine learning is to approximate a mapping function from input variables to a continuous output variable, such as an integer or float [20]. A regression task could be, for example, calculating mathematical expressions.

There is however some overlapping between classification tasks and regression tasks, such as an integer does exist in the real number set which means that it is possible to treat an integer as a class in a set of classes. Also, a classification algorithm may predict a continues value, but this value will exist in the class label probability [20].

2.1.1 Classification loss functions

Cross entropy and log loss are two loss functions that are mathematically different. However, when calculating error rates between 0 and 1 they resolve to the same [9]. Cross entropy calculates an error by interpreting a probability distribution. For each index/weight in the distribution, an error will be given depending on whether that index is the true label index or not. Cross entropy is given by:

$$-\frac{1}{N}\sum_{c=1}^{N}(y\log(p)+(1-y)\log(1-p)) \tag{2.2}$$

where N is the number of classes, log is the natural logarithm, y is the label truth value (1 if the correct class, 0 otherwise) and p is the prediction for class c. Only one part after the + sign is added for summation, since y can only be 0 or 1. If the prediction value is high for the correct class, the algorithm would result in a low error value. A low prediction value would yield a high error value. Conversely, if the prediction value is high for the wrong class, it would yield high error. The errors are distributed as log loss, shown in Figure 2.3 [9].

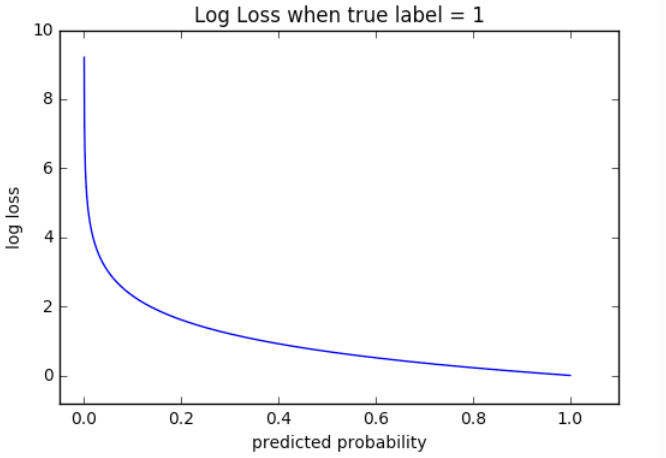

Figure 2.3: log loss w.r.t predicted probability [9]

The error output of cross entropy drastically increases when probability gets very low, as shown if figure 2.3. The purpose of increasing the error drastically is to penalize confident wrong answers higher than when the probability is spread, such that a probability distribution as [0.99, 0.01], with last index being true label, would give an error of ≈ 4.6, while a distribution of [0.5, 0.5] would give an error ≈ 0.7.

Focal loss was first introduced to address the one-stage object detection scenario. Focal loss can be considered as an extension of cross entropy and is defined by:

$$p_t = \begin{cases} p, & \text{if } y > 1 \\ 1 - p, & \text{otherwise} \end{cases} \tag{2.3}$$

$$FL(p_t) = -\alpha(1 - p_t)^{\gamma} \log(p_t) \tag{2.4}$$

Focal loss includes a tunable factor $\gamma \geq 0$ for focusing hard negative examples during training. $\gamma = 2$ is said to give best results [21]. Also focal loss comprise the balancing factor α from balanced cross entropy, to balance the importance of positive/negatives. Figure 2.4 show the foreground and background loss distribution dependant on the γ.

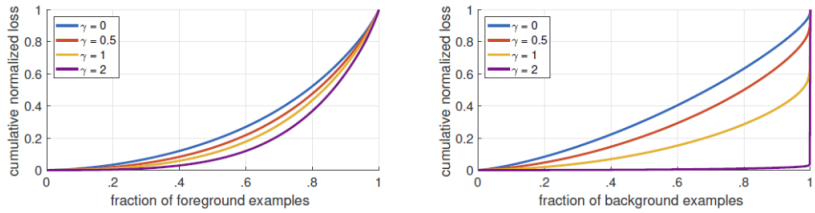

Figure 2.4: focal foreground and background loss [10]

KL (Kullback-Leibler) Divergent or Relative Entropy can be used as a loss function to estimate the loss of information from input q to prediction p. The KL Divergent is given by:

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \log(\frac{p(x_i)}{q(x_i)}) \tag{2.5}$$

KL Divergent can be used as a loss function for constructing the input as shown in Figure 2.5 [11].

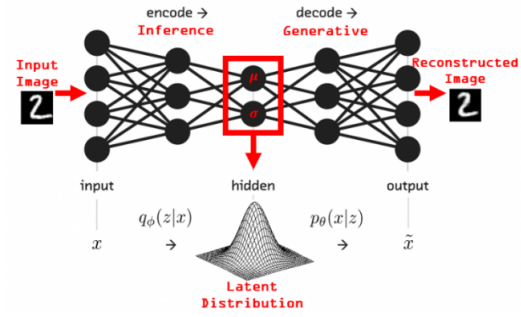

Figure 2.5: neural network generation of image [11]

Hinge loss is a loss function typically used in support Vector Machine. Hinge loss is given by:

$$\mathcal{L}(p) = \max(0, 1 - y \bullet p) \tag{2.6}$$

where p is the score or the probability distribution and y is the degree of truth to the label (1 for true, -1 for false).

Hinge loss is said to be the preferred choice for classification tasks [22].

2.1.2 Regression loss functions

A commonly used regression loss function is Mean square error (MSE)[8]. MSE is given by:

$$MSE = \frac{\sum_{i=1}^{n}(y_i - y_i^p)^2}{n} \tag{2.7}$$

and is the summation of the squared distance between the true label and predicted value. Figure 2.6 demonstrates MSE loss function where the predicted values ranges between -10 000 and 10 000, and the true value is 100.

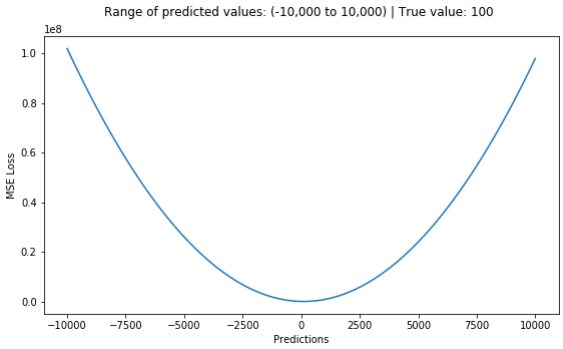

Figure 2.6: MSE loss w.r.t predicted integer value [8]

In MSE, the further away the predicted value is from the true value the faster the error grows. A loss function that produces a constant increase in error is Mean Absolute Error (MAE). In MAE the absolute value of the distance between the predicted value and the true label is summarized to produce the error as [8]:

$$MAE = \frac{\sum_{i=1}^{n} |y_i - y_i^p|}{n} \tag{2.8}$$

MAE is able to measure the average magnitude of a set of predictions errors, not considering directions. Mean Bias Error (MBE) considers directions when measuring the average magnitude of errors in a set of predictions [8].

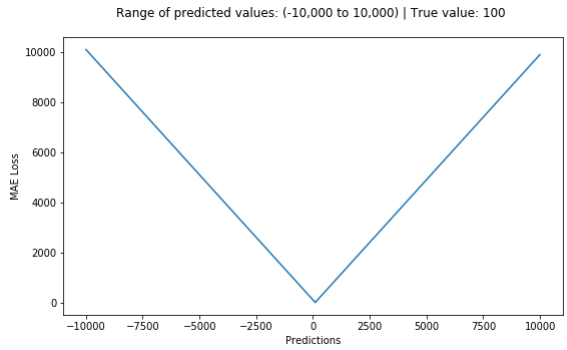

Figure 2.7: MAE loss w.r.t predicted integer value [8]

Figure 2.7 shows MAE loss function for values between -10 000 and 10 000 with true label 100. The figure shows that MAE error grows at a constant rate when moving away from the true label. Considering the optimizing method gradient descent, minimizing the loss of MSE and MAE, both would travel towards value 100 with the decrease of loss. But, as further described below in section 2.2.1, with a high learning rate or "travel distance" MAE might travel past the optimal point (value 100) as shown in Figure 2.8[8].

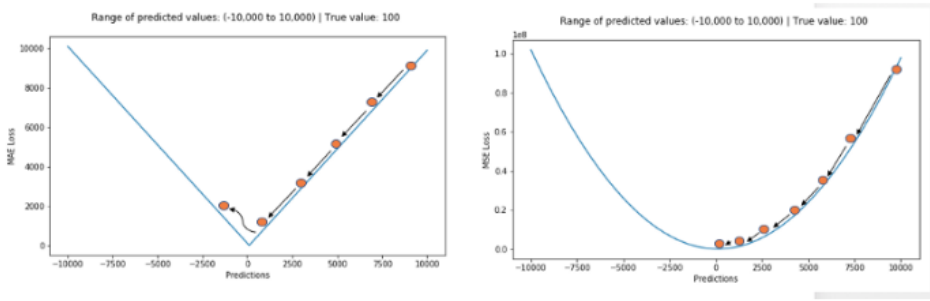

Figure 2.8: MAE vs MSE gradient descent with fixed learning rate [8]

The reason one can experience traveling past the optimal point with MAE loss function, is that the gradient stays the same even though the loss is decreasing. With MSE the gradient and the loss will decrease while closing to the optimal point, causing the travel distance to decrease for each step taken. Another property that is different between MSE and MAE is that MSE tends to weight outlying weights higher than MAE, which in turn means that in a set of data, rare points will be considered higher by MSE than MAE. MSE should be considered when outlying data points are considered useful, while MAE should be considered if outlying data points are considered as corrupt data. [8].

Considering a dataset that is heavily biased towards one value, as having 95% of the dataset being the value 1000, both MSE and MAE will fail. MAE would most likely cause a prediction of 1000 of all inputs, while MSE would produce more than 5% predictions that are not 1000. One could transform the target variables in such a way that MSE or MAE are feasible solutions. However, another possibility is to use the loss function Huber Loss instead.

Huber loss can be considered as a smooth MAE since it attains much of the same attributes of MAE, but Huber loss takes a parameter (delta) which is used as the starting point of smoothing as MSE. Huber loss is given by:

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2, & \text{for } |y - f(x)| \leq \delta \\ \delta|y - f(x)| - \frac{1}{2}\delta^2, & otherwise \end{cases} \tag{2.9}$$

Figure 2.9 shows how the Huber loss function acts as MAE until it reaches δ, then it acts as MSE. The figure shows Huber with $\delta = 0.1$, 1 and 10 with

the true value being 0 [8].

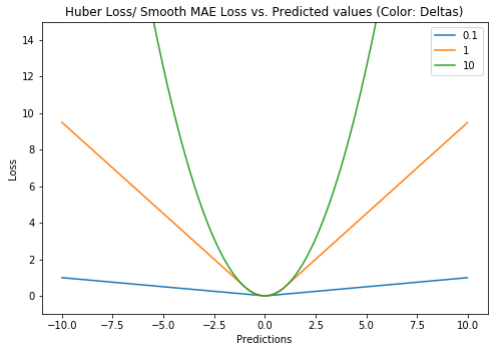

Figure 2.9: Huber loss with at delta 0.1, 1 and 10 w.r.t predicted float value [8]

The benefit of Huber loss over MAE is that Huber loss is able to slow down and converge towards the optimal point where loss is zero. Also, it is less sensitive to outlying data points, but is more time consuming since the parameter δ has to be tuned.

Another regression loss function is Log-Cosh loss which is the logarithm of the hyperbolic cosine of the predicted error [8] as given by:

$$L(y, y^p) = \sum_{i=1}^{n} \log(cosh(y_i^p - y_i)) \tag{2.10}$$

Log-Cosh inherits all the benefits of Huber loss and decreases that rate of wrong rare data point predictions as is typical of MSE, but does not exclude them totally. Figure 2.10 shows how Log-Cosh error is distributed with the true label being 0 [8].

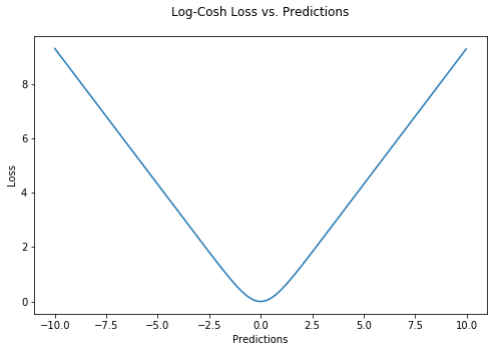

Figure 2.10: Log-cosh loss w.r.t predicted float value [8]

The last regression loss function that will be mentioned is Quantile loss. Quantile loss is typically useful when predicting intervals instead of point values. Quantile is able to estimate intervals with the non-constant variance of values and can be considered as an extension of MAE. Quantile loss takes a parameter quantile γ, which is used to increase or decrease penalizing of over and underestimations and ranges between 0 and 1. Quantile loss is given by:

$$L_\gamma(y, y^p) = \sum_{i=y_i<y_i^p} (\gamma - 1) \bullet |y_i - y_i^p| + \sum_{y_i \geq y_i^p} (\gamma) \bullet |y_i - y_i^p| \tag{2.11}$$

Below, in Figure 2.11, one can see how a low γ favours underestimation while high γ favours overestimation.

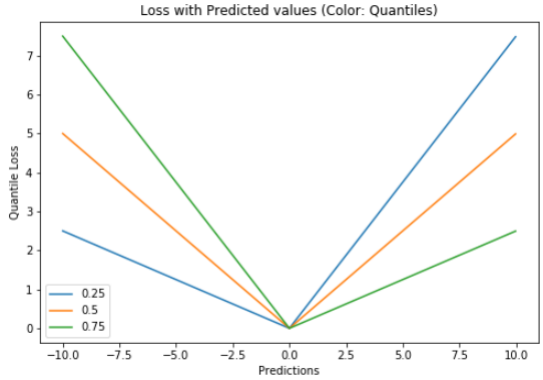

Figure 2.11: Quantiile loss w.r.t predicted float value [8]

Figure 2.12 shows a loss comparison of the regression models described in this section.

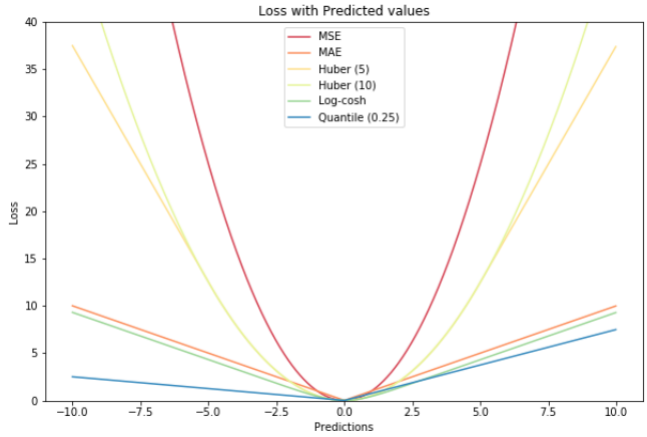

Figure 2.12: loss w.r.t predicted float value [8]

2.1.3 Activation functions

Nonlinear functions, also often referred to as activation functions, are able to transform vectors values and is used in different layers of the network depending on the output value ranges. For example, in order to calculate the loss in classification tasks, the predicted output of the machine learning algorithm should be transformed into a probability distribution. Some activation functions are:

- Sigmoid function $f(x) = \frac{1}{1+e^{-x}}$
- Hyperbolic Tangent function (Tanh) $f(x) = (e^x - e^{-x})/(e^x + e^{-x})$
- Softmax function $f(x_i) = \exp(x_i)/\sum_j \exp(x_j)$
- Softsign $f(x) = x/(|x| + 1)$
- Rectified Linear Unit (ReLU) function $f(x) = \max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i < 0 \end{cases}$
- Softplus function $f(x) = log(1 + \exp^x)$
- Exponential Linear Units (ELUs) $f(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha \exp(x) - 1, & \text{if } x \leq 0 \end{cases}$
- Maxout function $f(x) = \max(w_1^T x + b_1, w_2^T x + b_2)$
- Swish Function $f(x) = x \bullet sigmoid(x) = x/(1 + e^{-x})$
- ELiSH $f(x) = \begin{cases} x/(1 + e^{-x}), & \text{if } x \geq 0 \\ (e^x - 1)/(1 + e^{-x}), & \text{if } x < 0 \end{cases}$

Sigmoid function

The Sigmoid activation function, also referred to as the squashing or logistic function, is a nonlinear function used, for example, in feedforward neural networks [2]. A Sigmoid function is described as:

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

and is used in the output layer of deep learning models. Sigmoid function has been successfully applied to neural network domains, such as binary classification problems. The sigmoid activation function is highlighted as a highly used activation function in shallow networks and is easy to understand, but suffers major drawbacks as gradient saturation and slow convergence. Over the years, research has produced:

- Hard Sigmoid Function
- Sigmoid-Weighted Linear Units (SiLU)
- Derivative of Sigmoid-Weighted Linear Units (dSiLU)

as three Sigmoid functions that are used in deep learning [2]. The Hard Sigmoid funcion is described as:

$$f(x) = \text{clip}(\frac{x+1}{2}, 0, 1) \tag{2.13}$$

and offers less computational cost compared to the soft Sigmoid function, also showing promising results on deep learning based binary classification problems [2].

The SiLu is an approximation function based on reinforcment learning and is given by:

$$a_k(s) = z_k \alpha(z_k) \tag{2.14}$$

where s is the input vector and z_k is given by:

$$z_k = \sum w_{ik} s_i + b_k \tag{2.15}$$

with bias b_k and weight w_i connected to hidden unit k. SiLu function is only applicable in hidden layers of reinforcement based systems and is said to outperform ReLU function 2.1.3 [2] as shown in figure 2.13.

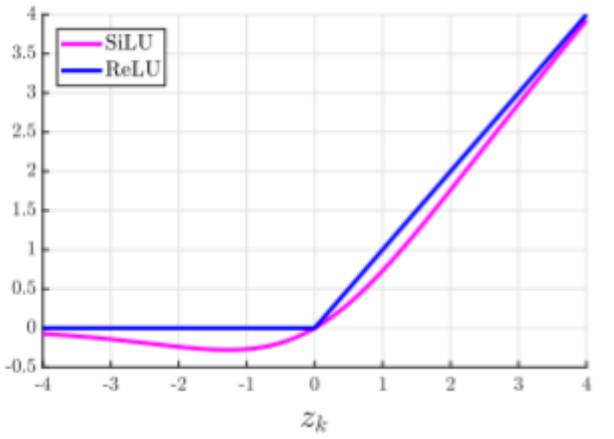

Figure 2.13: response comparison of SiLU and ReLU [2]

The derivative of SiLU (dSiLU) is used in gradient descent 2.2.1 for learning updates of weight parameters. The function is given by:

$$a_k(s) = \alpha(z_k)(1 + z_k(1 - \alpha(z_k))) \tag{2.16}$$

and is said to outperform the standard sigmoid function significantly [2] as shown in figure 2.14.

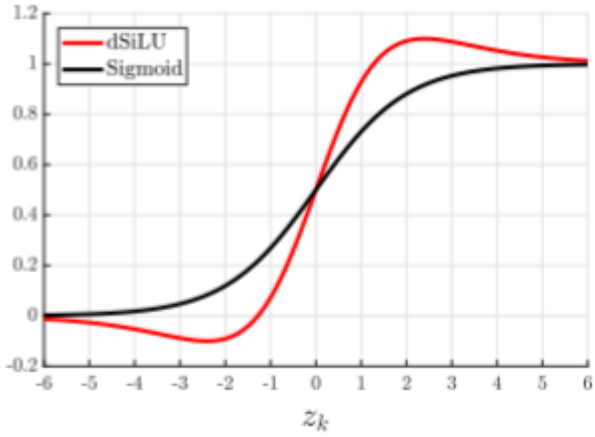

Figure 2.14: response comparison of dSiLU and Sigmoid [2]

Hyperbolic Tangent

The Hyperbolic Tangent (Tanh) activation function is a zero-based function that ranges between -1 and 1. The function is described as:

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.17}$$

and performs better than Sigmoid functions for multi-layer neural networks used in deep learning. The main advantage of the Tanh function is that the output is centralized around zero, which is preferable during the back-propagation process. However, as with the Sigmoid function, the Than function does not solve the gradient saturation problem. The Tanh functions are typically used in a recurrent neural network for speech recognition and language processing tasks [2].

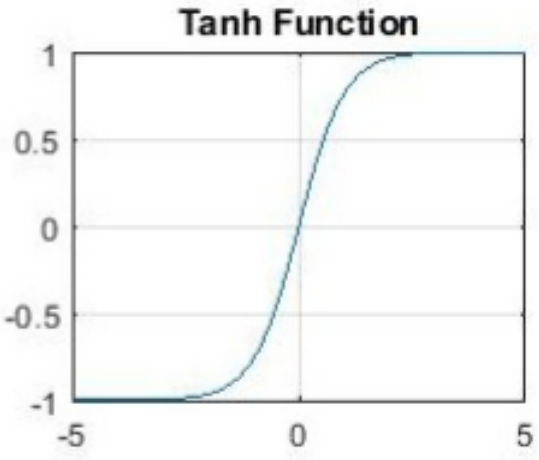

Figure 2.15: Hyperbolic Tangent function response representation [2]

Hard Hyperbolic Tangent (Hardtanh) function is a more computational efficient variant of the Tanh activation function. The Hardtanh function has been applied to natural language applications and is said to be an improvement in speed and accuracy over the regular Tanh function [2]. The Hardtanh is given by:

$$f(x) = \begin{cases} -1, & \text{if } x < -1 \\ x, & \text{if } -1 \geq x \leq 1 \\ 1, & \text{if } x > 1 \end{cases} \tag{2.18}$$

Softmax

The Softmax activation function is used to compute the probability distribution for a vector of real numbers and is mostly used in output layers of neural networks. The Softmax function can be used to produce the probability of a class in a set of classes. While Sigmoid functions are used in binary classification problems, Softmax is used when there exist more than two classes [2].

$$f(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{2.19}$$

Rectified Linear Unit

The rectified linear unit (ReLU) function eliminates the gradient saturation problem, which is present in the activation functions above. To achieve this, ReLU performs a threshold operation on each of the input elements, setting all input elements with values less than zero to zero:

$$f(x) = \max(0, x) = \begin{cases} x_i, & \text{if } x_i \geq 0 \\ 0, & \text{if } x_i < 0 \end{cases} \tag{2.20}$$

The main advantage of using the ReLU activation function is that it offers faster computation as it does not compute exponentials or divisions. The first limitation of the ReLU activation function is that it overfits easily compared to the sigmoid function. To reduce the effect of over-fitting, the dropout technique has been adopted. The second limitation is that ReLU units are fragile during training and can result in 'dead' neurons. A large gradient flowing through the network can cause the the weights to update in such a way that the neurons will never activate again, thereby causing the gradients flowing through these units to always be zero. However, the ReLU has overall improved the performance of deep neural networks [2].

In order to deal with the dead neuron problem the leaky ReLU activation function was proposed:

$$f(x) = \alpha x + x = \begin{cases} x_i, & \text{if } x_i > 0 \\ \alpha x, & \text{if } x_i \leq 0 \end{cases} \tag{2.21}$$

The leaky RelU introduces a small negative slope, in order to keep the weight updates alive during the propagation process.

Softplus

The Softplus function is a smooth primitive of the sigmoid function and shares similarities with the ReLU function with its nonzero gradient properties. The smoothing properties gives it improved stabilization and performance compared to Sigmoid and ReLU functions, where it required less epochs to converge under training [2].

$$f(x) = log(1 + exp^x) \tag{2.22}$$

The Softplus function has mainly been used in statistical applications, but has also seen some use in speech recognition systems.

Exponential Linear Units (ELUs)

The Exponential linear unit (ELU) proposed in [23] is given by

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(exp(x) - 1) & \text{if } x \leq 0 \end{cases} \quad , \quad f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ f(x) + \alpha & \text{if } x \leq 0 \end{cases} \tag{2.23}$$

where α is a hyperparameter which controls the value to which the ELU saturates for negative net inputs. As with ReLUs and LReLUs, the ELU activation function diminish the vanishing gradient effect. The reason for this is that the positive part is the identity, which means the derivative is one instead of contractive. Tanh and sigmoid is on the other hand contractive almost everywhere [23].

However, as opposed to ReLUs, the ELUs have negative values , which pushes the mean of the activations closer to zero. This enable faster learning, as they bring the gradient closer to natural gradient [23].

Although α can be set to any value, if $\alpha! = 1$ the function would be not be differentiable at $x = 0$. This means that learning the parameter α would break differentiability at $x = 0$. For this reason an alteration was proposed in [24] called parametric exponential linear unit (PELU) and is given by

$$f(x) = \begin{cases} \zeta x & \text{if } x \geq 0 \\ \alpha(exp(\frac{x}{\beta}) - 1) & \text{if } x < 0 \end{cases}, \quad \alpha, \beta, \zeta > 0 \tag{2.24}$$

This activation function has reportedly better performance than the ELU activation function used in CNN's on the CIFAR-10/100 and ImageNet datasets[24].

Maxout

The Maxout activation function applies non-linearity as the dot product between the weights and the data of the network. It inherits the properties of ReLU and LReLU where there are no dying neurons or saturation. The big drawback of the Maxout function is that it is computationally expensive, as it doubles all the parameters used in the neurons [2]. The function is given by

$$f(x) = \max(w_1^T x + b_1, w_2^T x + b_2) \tag{2.25}$$

where w = weights, b = biases.

Swish

The Swish activation function is unlike the previous activation functions in the sense that it is the first which combines two activation functions into one. It combines the sigmoid and input function to create a hybrid activation function. The swish function provides smoothness and is bounded below zero and unbounded above zero and is given by [2]:

$$f(x) = x \cdot sigmoid(x) = \frac{x}{1 + e^{-x}} \tag{2.26}$$

The Swish function does not suffer from vanishing gradient problems, but still attains good propagation of information during training. Reportedly, it performs better than ReLU on deep learning classification tasks [2].

ELiSH

The Exponential linear Squashing activation function (ELiSH) is the most recently proposed activation function included and shares properties with, and is highly inspired by the Swish function. It is based on the sigmoid and ELU functions and is given by:

$$f(x) = \begin{cases} \frac{x}{1+e^{-x}}, & \text{if } x \geq 0 \\ \frac{e^x-1}{1+e^{-x}}, & \text{if } x < 0 \end{cases} \tag{2.27}$$

The similarities with swish is apparent, as the negative part of the function is a multiplication of the Sigmoid and ELU functions and the positive part remains the same as Swish [25].

HardELiSH is a variation of the ELiSH function. The name is derived from the fact that the negative part is a multiplication of the HardSigmoid and ELU functions and the positive part is a multiplication of the Linear and the HardSigmoid functions [2] and is given by

$$f(x) = \begin{cases} x \times \max(0, \min(1, \frac{x+1}{2})), & \text{if } x \geq 0 \\ (E^X - 1) \times \max(0, \min(1, \frac{x+1}{2})), & \text{if } x < 0 \end{cases} \tag{2.28}$$

The motivation behind these activation functions was to take advantage of composite functions. As with Swish, the Sigmoid functions improves the informational flow and the Linear removes the issues with vanishing gradients [2].

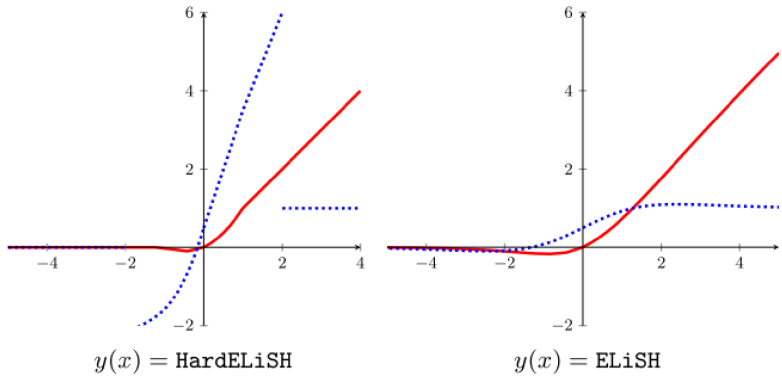

Figure 2.16: The activation functions HardELiSH and ELiSH function (red), and their derivatives (blue dotted) [2]

2.2 Optimization

Optimization can be considered the learning of most machine learning algorithm. The optimization task is the task of minimizing the loss calculated by the loss function in such a way that the mapping function between input and output variables has the lowest loss, zero if possible. Typically optimizing in machine learning is altering the value of the weights in such a way that the loss travel towards zero.

There are multiple ways for optimizing the mapping function, for example, one may generate a set of random weights in hope that one of the sets will be the correct. Generating random sets of weights will, given time, find the best mapping function, but that is given time (most cases, very long time). Another way is to increment or decrement each weight respectively and check if the loss for the weights is higher or lower. Considering a simple scenario, walking on a mountain where you want to get to the top (without knowing where the top is), you could take one step and check if you are higher above water, if not, step back and try another direction. By stepping a certain amount of time one should on average be able to reach the top faster than randomly landing all across the mountain.

Stepping seems like a good optimizing function, but finding the direction is

cumbersome if one simply randomly chooses the direction. Some function that describe the slope of the mountain is needed to optimize each step in the stepping function. It turns out that there is a way of optimizing what direction to take, one may consider it as a feeling of the mountain slope, but in machine learning its following the gradient of the loss function [4]. Figure 2.17 below shows the steepest direction given by the gradient.

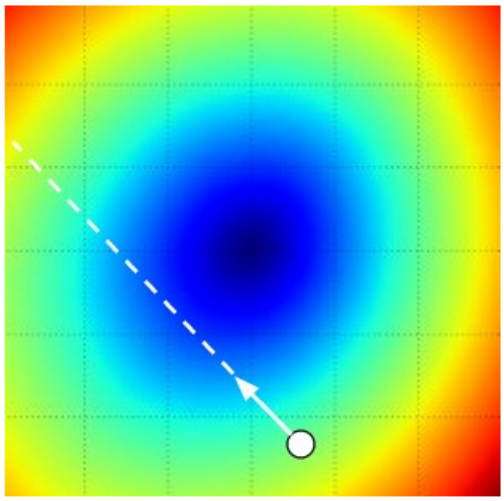

Figure 2.17: update direction given by gradient [4]

2.2.1 Gradient Descent

Gradient descent is a common algorithm used to update the mapping function between input and output variables, and uses loss functions loss output to determine the best update. Gradient descent takes a tunable parameter called learning rate, also called step, which determines the update size. The rate of learning may deeply impact the minimizing function, since a high learning rate will for sure cause the model to step over the optimal learning point, while a low learning rate will increase the time it takes to learn a task. Consider climbing a mountain and the learning rate to be the number of meters you take each step. If the top of the mountain is 10 meters away, but your step size is 20 meters, you will never be able to stand on the top, you will simply overstep each time. On the other hand if the top is 100 meters away, but your step size is 0.0001 meter, it will take many hours to

reach the top.

There are multiple versions of gradient descent, where batch gradient descent can be considered the first version, or vanilla algorithm, and is given by:

$$\theta = \theta - \eta \bullet \nabla_\theta J(\theta) \tag{2.29}$$

where η is the learning rate an ∇ is the gradient.

Batch gradient descent computes the descent of the whole training set to make one update, which is not preferable for 3 reasons:

- size of memory used for one update
- time consumed per update
- no updates after initial training

Stochastic gradient descent on the other hand computes a gradient for each sample x with true label y and is given by:

$$\theta = \theta - \eta \bullet \nabla_\theta J(\theta; x^i; y^i) \tag{2.30}$$

Since stochastic gradient descent computes the gradient at each example, it is able to converge to a minima much faster. Stochastic gradient descent also enables updates after initial training. Since stochastic gradient descent updates weights at each example, the fluctuation of the travel path is larger than the batch gradient descent. High fluctuation enables discoveries of new possible best paths, but is prone to overshoot the optimal point. On the other hand, it has been shown that decreasing the learning rate over time will enable stochastic gradient descent to converge [12].

The third variant of gradient descent is mini-batch gradient descent. Mini-batch gradient descent performs updates for small blocks of the training data, and so can be considered to have the advantages of both vanilla gradient descent and stochastic gradient descent [12]. Mini-batch gradient descent is given by:

$$\theta = \theta - \eta \bullet \nabla_\theta J(\theta; x^{i:i+n}; y^{i:i+n}) \tag{2.31}$$

where n is the the number of training example per batch.

Mini-batch gradient descent, because of its reduced variance and high efficiency gradient matrix computation, can be considered the best choice of the three described above. On the other hand, the original mini-batch gradient descent came with some challenges. Getting trapped in a sub optimal solution is one of the key challenges. Imagine climbing a mountain blindfolded. You know that you are climbing a mountain, but do you know if the climb ends in the actual top. Even though the mountain slopes down in every direction you turn, the mountain may slope back up again and bring you higher than the point at which you are currently standing.

A similar issue occurs when there are saddle points. Saddle points are when one dimension slopes up while another slopes down as shown in figure 2.18.

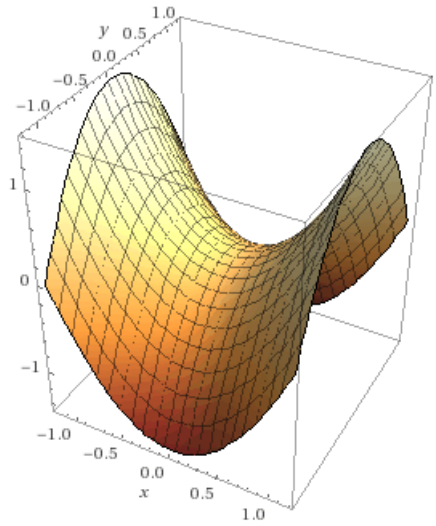

Figure 2.18: saddle point of $z = x^2 - y^2$

The saddle points are typically surrounded with the same error causing the gradient to go towards zero in all dimensions. There are also other similar challenges that arise from using the gradient. Since the travel path is always

along the steepest route, as shown in figure 2.19 below, one may be walking up and down ravines without traveling in the optimal direction [12].

Figure 2.19: travel path of stochastic gradient descent [12]

A method called momentum dampens the constant shifting directions, while accelerating the relevant directions. Stochastic gradient descent with momentum is given by:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta) \tag{2.32}$$

$$\theta = \theta - v_t \tag{2.33}$$

where γ is used to get a fraction of the update vector from the previous update. γ is typically set to 0.9 [12]. Efficiency for ravine scenarios is increased when using momentum as shown in figure 2.20.

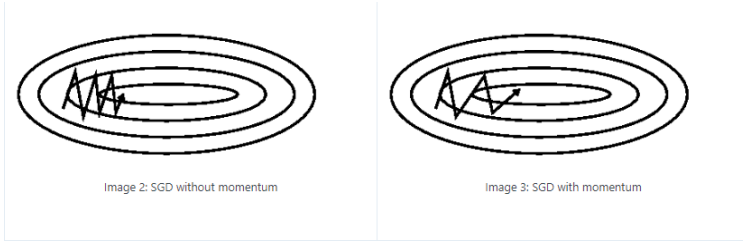

Figure 2.20: travel path of stochastic gradient descent with and without momentum [12]

A challenge with momentum is the increase in speed, in that the acceleration might cause overstepping, which might take additional steps to correct.

Another model similar to momentum is Nesterov accelerated gradient. Nesterov accelerated gradient looks at the approximate future position, in addition to the last position from the momentum model. Nesterov accelerated gradient is given by:

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta - \gamma v_{t-1}) \tag{2.34}$$
$$\theta = \theta - v_t \tag{2.35}$$

Figure 2.21 show how normal momentum (blue arrow) sharply accelerates in a direction, while Nesterov accelerated gradient first makes an initial calculation (brown error), and based on the approximate next position (red arrow), makes an update (green arrow) that is more precise than the normal momentum [12].

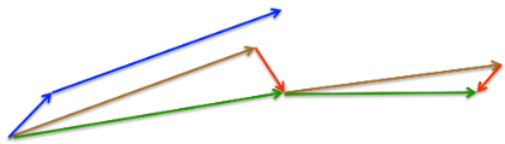

Figure 2.21: momentum and Nesterov accelerated gradient updates [12]

Another challenge of gradient descent variants are tuning the learning rate. The learning rate is statically set as a value, either high or low, at the start of training when using the gradient descent methods described above. In terms a high learning rate might cause the model to not converge, while a low learning rate will cause a longer learning time. The better solution would be to start with a high learning rate and decrease it over time. Adagrand aims to solve the learning rate issue by applying a learning rate update during training, while, in addition, enabling individual learning rates per parameter. Adagrand is given by:

$$g_{t,i} = \nabla_\theta J(\theta_{t,i}) \tag{2.36}$$
$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \odot g_{t,i} \tag{2.37}$$

where $g_{t,i}$ is the gradient with respect to parameter θ_i, $G_t \in \mathbb{R}^{d \times d}$ is a matrix containing the sum of the gradients squares, and ϵ is a value to avoid division by zero, typically $1e-8$. The method is said to perform much worse without the square root operation [12].

One flaw of Adagrand is that the accumulated sum in the denominator grows rapidly, causing the learning rate to rapidly approach zero. When learning rate is infinitely small, the learning itself becomes absent.

Another method called Adadelta reduces the aggressive decrease in learning rate by restricting the accumulated gradients. Adadelta restricts accumulated past gradients to a fixed size w. As opposed to storing the previous squared gradients, Adadelta stores the decaying average of the past squared gradients. Adadelta is given by:

$$\nabla\theta_t = -\frac{RMS[\nabla\theta]_{t-1}}{RMS[g]_t} g_t \tag{2.38}$$

$$\theta_{t+1} = \theta_t + \nabla\theta_t \tag{2.39}$$

where RMS is the root mean square, replacing learning rate as an initial parameter. Observing figure 2.22 below, one could easily see that Adadelta is outperforming the others for saddle point scenarios, while also performing better on a regular loss surface [12].

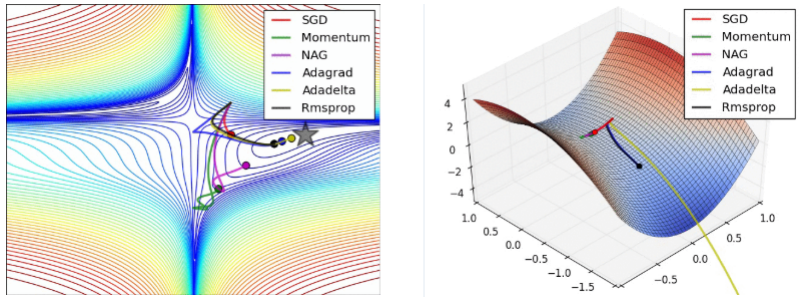

Figure 2.22: comparison of optimizing methods [12]

Adam is a method that extends on Adadelta. Adam stores decaying averages of past squared gradients v_t as Adadelta, while also, similar to momentum, keeping an exponentially decaying average of past gradients m_t [12].

Adam can typically be seen as applying friction, such that flat surfaces in the error spaces are preferred. Adam is given by:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t}}\hat{m}_t \tag{2.40}$$

Where $\hat{v}_t$ and $\hat{m}_t$ are the bias-correction of the past squared gradients (v_t) and exponential past squared gradients (m_t).

Adam is said to work well in practice, but a challenge of Adam is that due to exponential averaging, informative mini batches that occur rarely are diminished [12].

2.3 Conclusion

In this chapter we have explored the supervised, unsupervised and semi-supervised learning methods. We have presented some of the more prominent loss functions, including classification loss functions, regression loss functions and activation functions. Lastly, we have explored some of the more prominent optimization functions using gradient descent.

Chapter 3

End-to-End Memory Networks

3.1 Introduction

Instructing machines that can converse like a human for real-world objectives is possibly one of the crucial challenges in artificial intelligence. In order to construct a significant conversation with human, the dialog system is required to be qualified in the perception of natural language, constructing intelligent decisions as well as producing proper replies [26, 27, 28]. Dialog systems, recognized as interactive conversational agents, communicate with the human through natural language in order to aid, supply information and amuse. They are utilized in an extensive applications domain from technical support services to language learning tools [29, 30].

In artificial intelligence area [31, 32, 33], end-to-end dialog systems has gained interest because of the current progress of deep neural networks. In [14] a gated end-to-end trainable memory network is proposed which is learning in an end-to-end procedure without the utilization of any extra supervision signal. In [16] the original task is broken down into short tasks where they should be individually learned by the agent, and also built in order to perform the original task. In [34] a long short term memory (LSTM) model is suggested which learns in order to interact with APIs on behalf of the user. In [35] a dynamic memory network is introduced which contains

tasks for part-of-speech classification as well as question answering, also uses two gated recurrent units in order to carry out inference. In [36] the memory network has been implemented which needed supervision in every layer of the network. In [37] a set of four tasks in order to test the capability of end-to-end dialog systems has been introduced which focuses on the domain of movies entities. In [38] a word-based method to dialog state tracking utilizing recurrent neural networks (RNNs) is proposed which needs less feature engineering. Even though neural network models include a tiny learning pipeline, they need a remarkable content of the training. Gated recurrent network (GRU) and LSTM units permit RNNs to deal with the longer texts needed for question answering. Additional advancements to be mentioned as attention mechanisms, as well as memory networks, permit the network to center around the most related facts.

In this chapter, the applications of different types of memory networks are studied on data from the Dialog bAbI. The performance results demonstrate that all the proposed techniques attain decent precision on the Dialog bAbI datasets. The best performance is obtained utilizing UN2N.

3.2 Memory Networks

3.2.1 End-to-End Memory Network with Single Hop

We will first explain the end-to-end memory network (N2N) using a single hop (shown in its entirety in figure 3.1) and in the next section, we will explain how this structure is expanded into multiple hops. The N2N model takes a discrete number of inputs $(\tilde{x}_1, \tilde{x}_2 ... \tilde{x}_i)$, which it writes to the memory, a query $\tilde{q}$ and outputs an answer $\tilde{a}$. The set of $\{\tilde{x}_i\}$ is converted into memory vectors $\{\tilde{m}_i\}$ by using an embedding matrix $\tilde{A}$, with the size $(d \times V)$, where d is the vector size and V is the number of words in the vocabulary. In the same sense, the question $\tilde{q}$ is converted into the vector $\tilde{u}$, using the embedding matrix $\tilde{B}$ with the same dimensions as $\tilde{A}$.

The matrices inner product is utilized in order to calculate the match between each the memory inputs $\tilde{m}_i$ and the question $\tilde{u}$, which will cause the creation of the attention. Then, by performing a softmax operation on the attention, the probability distribution $\tilde{p}_i$ across all the words from the memory input is created.

$$\tilde{p}_i = Softmax(\tilde{u}^T \tilde{m}_i) \tag{3.1}$$

Each of the input sentences $\tilde{x}_i$ has a corresponding output vector $\tilde{c}_i$, which is created using a third embedding matrix $\tilde{C}$. The response vector $\tilde{O}$ is created by the sum of output vectors weighed with the probability distribution of the memory input.

$$\tilde{O} = \sum_i \tilde{p}_i \tilde{c}_i \tag{3.2}$$

Finally, in order to generate the predicted answer, the sum of the output vector and the question embedding is passed through the weight matrix $\tilde{W}$ with a subsequent softmax operation.

$$\tilde{a} = Softmax(\tilde{W}(\tilde{O} + \tilde{u})) \tag{3.3}$$

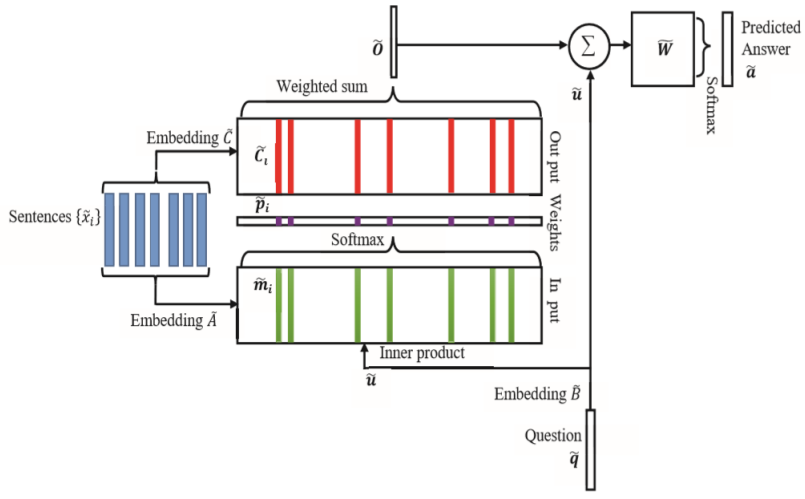

Figure 3.1: End-to-end memory network with a single hop[13]

3.2.2 End-to-End Memory Network with Multiple Hops

The N2N architecture contains two major components: supporting memories and final answer prediction [13]. Supporting memories consist of a set of input and output memory represented by memory cells. In complicated tasks with the requirement of multiple supporting memories, the model can be built to accommodate more than one set of input-output memories by stacking a number of memory layers. Each memory layer in the model is called a hop, also the input of the $(K+1)^{th}$ hop is the output of the Kth hop:

$$\tilde{u}^{k+1} = \tilde{o}^k + \tilde{u}^k \tag{3.4}$$

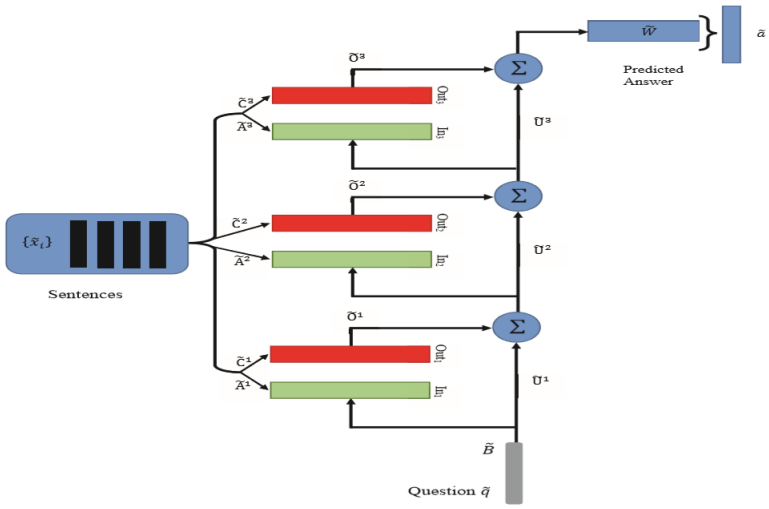

Figure 3.2: A three layer end-to-end memory network[13]

Each layer contains its own embedding matrices $\tilde{A}^K$ and $\tilde{C}^K$, which is utilized to embed the memory inputs $\tilde{x}_i$. The prediction of the answer to the question $\tilde{q}$, is carried out by the last memory hop:

$$\tilde{a} = softmax(\tilde{W}(\tilde{o}^k + \tilde{u}^k)) \tag{3.5}$$

where $\tilde{a}$ is taken to be the predicted answer distribution, $\tilde{W}$ (of size $V \times d$) is considered to be a parameter matrix for the model in order to learn, also K is the total number of hops. The N2N architecture with three hop operations is shown in Figure 3.2. The hard max operations within each layer are substituted with a continuous weighting from the softmax. The method takes a discrete set of inputs $\tilde{x}_1, ..., \tilde{x}_n$ which are stored in the memory, a question $\tilde{q}$, also outputs a reply $\tilde{a}$. The model can write all $\tilde{x}$ to the memory up to a fixed buffer size, also it obtains a continuous demonstration for $\tilde{x}$ and $\tilde{q}$. Afterward, the continuous demonstration is processed with multiple hops in order to generate $\tilde{a}$. This permits backpropagation of the error signal through multiple memory accesses back to the input while training.

In addition to the architecture above, the model implements something called match-features.

3.2.3 Gated End-to-End Memory Network

The gated end-to-end memory network (GN2N) adopts the idea of an adaptive gating mechanism introduced in Highway Networks in [39]. Highway networks can be understood as an extension of a feedforward neural network consisting of n layers, where the n^th layer applies a non-linear transform H ($\tilde{W}_H$) on its input x_n, to produce the output y_n. Which makes x_n the input to the network and y_n as the output:

$$y_n = H(x, \tilde{W}_H) \tag{3.6}$$

In [39] Srivastava et al. additionally define two additional non-linear transformations $T(x, \tilde{W}_T)$ and $C(x, \tilde{W}_C)$:

$$y_n = H(x, \tilde{W}_H) \cdot T(x, \tilde{W}_T) + x \cdot C(x, \tilde{W}_H) \tag{3.7}$$

These transforms are called the transform and carry gate (C and T), as they express how much of the output is produced by transforming and carrying the input. Furthermore, for simplicity, the carry gate is set as $C = 1 - T(x)$:

$$y_n = H(x, \tilde{W}_H) \cdot T(x, \tilde{W}_T) + x \cdot (1 - T(x, \tilde{W}_T)) \tag{3.8}$$

Liu and Perez [14] argues that residual networks can be seen as a special case of Highway Networks, where the transform and carry gate are substituted by the identity mapping function. Which leaves us with:

$$y = H(x) + x \tag{3.9}$$

Equation 3.4 is in fact very similar, where one can view $\tilde{o}^k$ as the residual function and $\tilde{u}^k$ as the shortcut connection. As opposed to hard-wired skip connections in Residual networks, highway networks offer and adaptive gating mechanism. This mechanism was therefore adopted into the memory network (N2N), creating the gated memory network (GN2N). This makes the network able to dynamically conditioning the memory reading operation on the controller state $\tilde{u^K}$ at every hop, see Figure 3.3. In GN2N, formula 3.4 is reformulated as below [14],

$$T^K(\tilde{u}^k) = \sigma(\tilde{W}_T^K \tilde{u}^K + \tilde{b}_T^K) \tag{3.10}$$

$$\tilde{u}^{K+1} = \tilde{o}^K \odot T^K(\tilde{u}^k) + \tilde{u}^k \odot (1 - T^K(\tilde{u}^K)) \tag{3.11}$$

where $\tilde{W}_T^K \tilde{u}^K$ and $\tilde{b}_T^K$ are taken to be the hop-specific parameter matrix and bias term for the k^{th} hop respectively. $T^K(\tilde{x})$ is the transform gate for the K^{th} hop and $\odot$ is the Hadamard product.

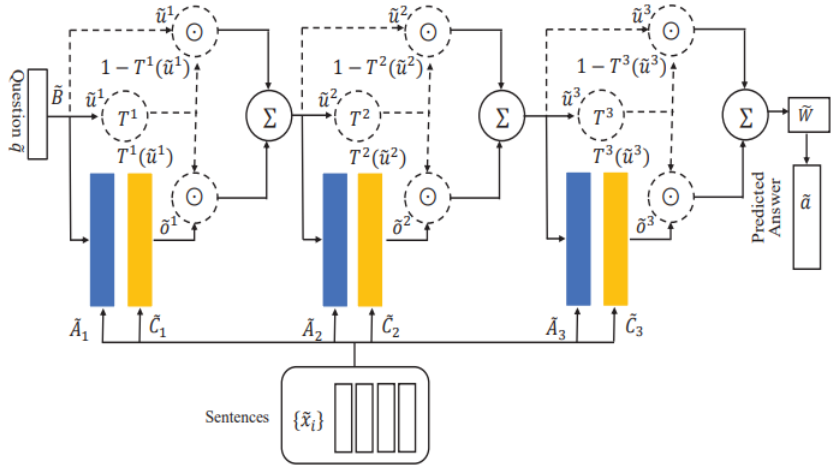

Figure 3.3: Gated end-to-end memory network [14]

3.2.4 End-to-End memory Networks with Unified Weight Tying

In [13], two kinds of weight tying are proposed for N2N, namely adjacent and layer-wise. Layer-wise approach portions the input and output embedding matrices across various hops (i.e., $\tilde{A}^1 = \tilde{A}^2 = ... = \tilde{A}^K$ and $\tilde{C}^1 = \tilde{C}^2 = ... = C^K$). Adjacent approach portions the output embedding for a given layer with the corresponding input embedding (i.e., $A^{\tilde{K}+1} = \tilde{C}^K$). Furthermore, the matrix $\tilde{W}$ which predicts the answer, as well as the question embedding matrix $\tilde{B}$, are developed as $\tilde{W}^T = \tilde{C}^K$ and $\tilde{B} = \tilde{A}^1$. These weight tying mechanisms are shown in figure 3.4, where the dotted lines indicate the adjacent approach and the lines indicate the layer-wise approach.

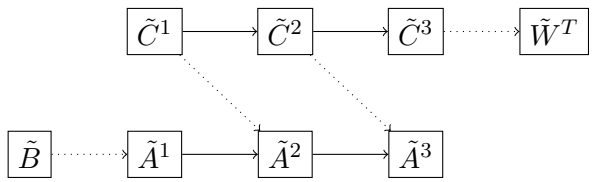

Figure 3.4: Illustrating the two different weight tying mechanisms in on a N2N memory network with 3 hops [15]

Both of these techniques reportedly [15] perform well, albeit inconsistently on some tasks. Therefore in [15], a dynamic mechanism is designed which permits the model to choose which of the two weight tying techniques on the basis of the input. Therefore, the embedding matrices are developed dynamically for every instance which makes UN2N more efficient compared with N2N and GN2N where one weight tying technique is used throughout all the embeddings. In UN2N a gating vector $\tilde{z}$, described in equation 3.15, is used in order to develop the embedding matrices, $\tilde{A}^K, \tilde{C}^K, \tilde{B}$, and $\tilde{W}$. The embedding matrices are influenced by the information transported by $\tilde{z}$ related to the input question $\tilde{u^0}$ and the context sentences in the story $\tilde{m_t}$. Therefore,

$$\tilde{A}^{K+1} = \tilde{A}^K \odot \tilde{z} + \tilde{C}^K \odot (1 - \tilde{z}) \tag{3.12}$$

$$\tilde{C}^{K+1} = \tilde{C}^K \odot \tilde{z} + \tilde{C}^K \odot (1 - \tilde{z}) \tag{3.13}$$

where $\odot$ is taken to be the column element-wise multiplication operation, also $\tilde{C}^{K+1}$ is the unconstrained embedding matrix. In 3.12 and 3.13, the large value of $\tilde{z}$ leads UN2N towards the layer-wise approach and the small value of $\tilde{z}$ leads UN2N towards the adjacent approach. In UN2N, at first, the story is encoded by reading the memory one step at a time with a gated recurrent unit (GRU) as below,

$$\tilde{h}_{t+1} = GRU(\tilde{m}_t, \tilde{h}_t) \tag{3.14}$$

such that t is considered to be the recurrent time step, also $\tilde{m}_t$ is taken to be the context sentence in the story at time t. Afterward, the following relation is defined,

$$\tilde{z} = \sigma(\tilde{W}_{\tilde{z}} \begin{bmatrix} \tilde{u}^0 \\ \tilde{h}_T \end{bmatrix} + \tilde{b}_{\tilde{z}}) \tag{3.15}$$

where $\tilde{h}_T$ is the last hidden state of the GRU which presents the story, ($\tilde{W}_{\tilde{z}}$ is considered as a weight matrix, $\tilde{b}_{\tilde{z}}$ is bias term, σ is taken to be the sigmoid function and $\begin{bmatrix} \tilde{u}^0 \\ \tilde{h}_T \end{bmatrix}$ is the concatenation of $\tilde{u}^0$ and $\tilde{h}_T$. A linear

mapping $G \epsilon R^{d \times d}$ is added for updating the connection between memory hops as below,

$$\tilde{u}^{K+1} = \tilde{O}^K + (G \odot (1 - \tilde{z}))\tilde{u}^K \tag{3.16}$$

3.3 Experiment and results

3.3.1 Experiment Setup

In this section, the data results were gathered from a range of publications which tackled the tasks of this dataset. Bordes et al.[16] provided the results for rule-based systems (RBS), TF-IDF match, Nearest Neighbor (NN), Supervised Embeddings and Memory network (N2N). Liu & Perez provided the results for the gated end-to-end memory network (GN2N) [14]. Liu et al. provided the results for unified weight tying end-to-end memory network (UN2N)[15].

3.3.2 Dataset

In *Learning End-to-End Goal-Oriented Dialog*[16], Bordes, et al. introduced a testbed to train and evaluate end-to-end goal oriented dialog systems. The dataset requires the systems to manipulate sentences and symbols in order to conduct a conversation, perform API calls and use the information from these API calls. The dataset is set in the context of restaurant reservation, consisting of five tasks and includes a knowledge base (KB) containing different restaurants and their properties. These tasks cover different segments of the conversation and will test the model's ability to manage dialogues, query the KB, interpreting these queries to continue the dialogue and dealing with restaurant properties previously not seen under training (OOV tasks).

Task 1: Issuing API calls The chatbot asks questions in order to fill the missing areas, and finally produces a valid corresponding API call. The questions asked by the bot is for collecting information in order to make the prediction possible.

Task 2: Updating API calls In this part users update their requests. The chatbot asks from users if they have finished their updates, then chatbot generates updated API call.

Task 3: Demonstrating options The chatbot provides options to users utilizing the corresponding API call.

Task 4: Generating additional information User can ask for the phone number and address and the bot should use the knowledge bases facts correctly in order to reply.

Task 5: Organizing entire dialogues Tasks 1-4 are combined in order to generate entire dialogues.

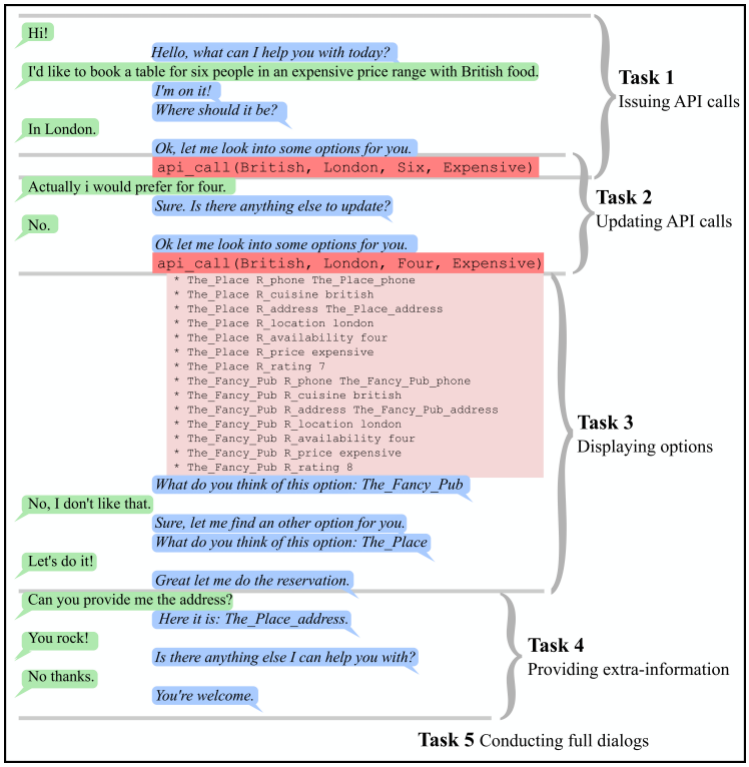

Figure 3.5: The different goal-oriented dialog tasks. A user (green) chats with a dialog system. Where the dialog system predicts responses (blue) and API calls (red), giving the API call results(light red) [16]

These five tasks are all generated from the same KB and it contains 1000 dialogues for training, validation and test[16]. In addition to these five tasks, a sixth task was created. This task is similar to task five, as it contains full dialogues, but instead of being generated, it is converted from the second dialog state tracking challenge [40] and contains dialog extracted from a real online concierge service. Task one to five also contains two test sets, one which contains only entities which are used in the training set and one which uses entities which are not seen in the training set.

3.3.3 Match features

Words that denote an entity have two important traits: 1. Exact matches are usually more appropriate than approximate matches. 2. They frequently appear as OOV words. This leads to two problems. Firstly, it is challenging to differentiate between exact matches and approximate matches in a lower embedding space. Secondly, words that has not been seen under training will not have any available word embedding, which usually results in failure [16]. In order to deal with the issues regarding entity types, the different end-to-end memory networks also implement something called match features, suggested by Bordes et al.[16]. With this, they augment the vocabulary with entity type words, one for each of the KB entity types. If a word is found in Input/Memory, candidate and also is considered as a KB entity, the candidate representation will be augmented with this KB entity type word.

3.3.4 Experiment Results

Efficiency results on Dialog bAbI tasks are demonstrated in Table 1, with rule-based systems, TF-IDF, nearest neighbor, supervised embedding, N2N, GN2N, and UN2N. As is shown in Table 1, the rule-based system has a high performance on tasks 1-5. However, its performance reduces when dealing with DSTC-2 task. TF-IDF match has poor performance compared with other methods on both the simulated tasks 1-5 and on the real data of task 6. The performance of the TFIDF match with match type features considerably increases but is still behind the nearest neighbor technique. Supervised embedding has higher performance compared with TF-IDF match and nearest neighbor technique. In task 1, supervised embedding is fully successful

but its performance reduces in task 2-5, even with match type features. GN2N and UN2N models outperform the other methods in DSTC-2 task and Dialog bAbI tasks respectively.

Task	RBS	TF-IDF		NN	-match				match			
		no type	type		S-emb	N2N	GN2N	UN2N	S-emb	N2N	GN2N	UN2N
1.Issuing API calls	100	5.6	22.4	55.1	100	99.9	100	100	83.2	100	100	100
2.Updating API calls	100	3.4	16.4	68.3	68.4	100	100	100	68.4	98.3	100	100
3.Displaying options	100	8.0	8.0	58.8	64.9	74.9	74.9	74.9	64.9	74.9	74.9	74.9
4.Generating additional information	100	9.5	17.8	28.6	57.2	59.5	57.2	57.2	57.2	100	100	100
5.Full dialogs	100	4.6	8.1	57.1	75.4	96.1	96.3	99.2	76.2	93.4	98.0	99.4
Avarage	100	6.2	14.5	53.6	73.2	86.1	85.7	86.3	70.0	93.3	94.6	94.8
1.(OOV) Issuing API calls	100	5.8	22.4	44.1	60	72.3	82.4	83	67.2	96.5	100	100
2.(OOV) Updating API calls	100	3.5	16.8	68.3	68.3	78.9	78.9	78.9	68.3	94.5	94.5	94.5
3.(OOV) Displaying options	100	8.3	8.3	58.8	65	74.4	75.3	75.2	65	75.2	75.1	76.3
4.(OOV) Generating additional information	100	8.8	17.2	28.6	57	57.6	57	57	57.1	100	100	100
5.(OOV) Full dialogs	100	4.6	9	48.4	58.2	65.5	66.7	67.8	64.4	77.7	79.4	79.5
Avarage	100	6.4	14.7	49.6	61.7	69.7	72.1	72.4	64.4	88.8	89.7	89.8
6. Dialog state Tracking 2	33.3	1.6	1.6	21.9	22.6	41.1	47.4	42.4	22.1	41	48.7	42.9

Table 3.1: The accuracy results of rule-based systems (RBS), TF-IDF, nearest neighbour (NN), supervised embedding (S-emb), N2N, GN2N and UN2N methods.

3.4 Conclusion

End-to-end learning scheme is suitable for constructing the dialog system because of its simplicity in training as well as effectiveness in model updating. In this chapter, the applications of various memory networks are studied on data from the Dialog bAbI. The performance results demonstrate that all the proposed techniques attain decent precision on the Dialog bAbI datasets. The best performance is obtained utilizing UN2N. In order to evaluate the true performance of the proposed methods, extra experimentation are required utilizing wide non-synthetic dataset.

Part II

Contributions

Chapter 4

Proposed approach

In chapter 2 and 3 we have spoken of different optimization techniques and memory networks. When doing our own testing of the memory networks we found them to perform worse when dealing with non-synthetic data and dealing with OOV-problems. In order to mitigate these problems, we are going to explore different ways of representing the data in memory networks. As is usual with machine learning algorithms like these, the data is required to be represented as a vector of a fixed length. Among the most common of these fixed length representations are bag-of-words and bag-of-n-grams [41]. Bag-of-words representation is used in all of the memory networks mentioned in chapter 3 and is often used for its simplicity and accuracy. However, this representation is not without issues. In particular, as the sentence representation is a concatenation of a series of word vectors, the word order in a sentence is lost. As such, two different sentences can have the exact same representation as long as the same words are used. Bag-of-n-grams do consider word order in a short context, but it suffers from data sparsity and high dimensionality [42]. There are two different representations that are proposed, which not only preserve word order, but also seek to attain some semantic meaning from the words. These models are the continuous word vector model introduced by Mikolov et al. [17] and the paragraph vector model introduced by Le & Mikolov [42].

Previous work [43] suggests that integrating a continuous word vector architecture into the Memory network resulted in no noticeable benefit. However, we will briefly explain its architecture in this chapter as it serves to

gain an understanding of the paragraph vector architecture which builds on ideas from it. We will propose a new memory network model used for goal-oriented dialog and evaluate it by using the Dialog bAbI dataset [16].

4.1 Distributed bag-of-word memory network

4.1.1 Continuous word vectors

Mikelov et al. [17] aimed to create a natural language processing (NLP) system which could produce high-quality word vectors from bigger datasets compared with the work that was previously done by NLP systems which produced continuous word representations. The goal was not only to have similar words produce resembling vectors, but also that words can have multiple dimensions of similarities as shown previously in [44]. As the goal was to create a system which could handle bigger datasets, the system would need lower training complexity than the competing systems.

The continuous bag-of-words model (CBOW) consists of input, projection and output layers. In the input layer, the N previous words are encoded by using 1-of-V coding, where V is the size of the vocabulary and the words representations are of size D. The input layer is then projected onto the projection layer which has a dimension of $N \times D$, where the projection layer is shared among all the words [17].

The skip-gram model is much like the CBOW model, but instead of predicting a word based on the context, it will take the current word and use it to predict words within a specified range of that word. Both models can be seen in figure 4.1[17].

Mikelov et al. [17] report that the continuous skip-gram model performs better with semantic word relations and the CBOW model perform better with syntactic word relations. Both of them perform considerably better than previous continuous NLP systems.

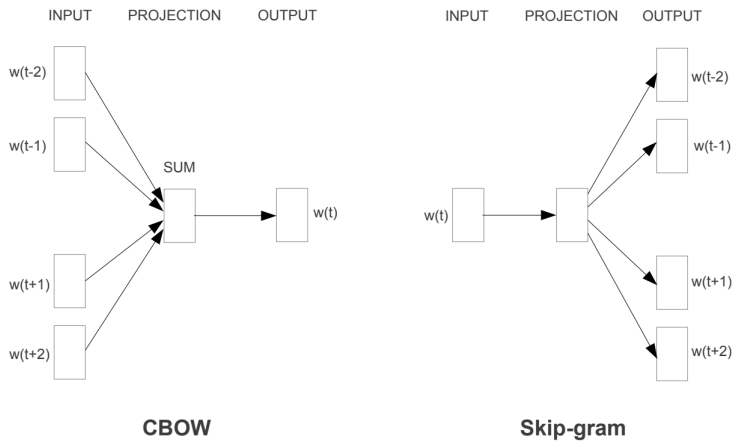

Figure 4.1: CBOW architecture predicts the current word based on the context, and the Skip-gram predicts the surrounding words from the given current word [17]

4.1.2 Paragraph vectors

Unlike the two previous NLP systems, where a continuous vector representation of a word is learned, the paragraph vector learns a continuous vector representation for text pieces. These pieces can be as small as sentences or entire documents. However, similarly to the previous, there are two related architectures that mirror each other in shape. As with word vectors, in the paragraph vector model word vectors are asked to contribute to a prediction of a word. In this way, even though the word vectors are randomly initialized, the model will eventually be able to learn the semantics of the words as an indirect result of these prediction tasks.

The first model, Distributed Memory Model of Paragraph Vectors (PV-DM) represents all the different paragraph vectors as a column in matrix D and all the word vectors as a column in matrix W. Then, the word vectors and paragraph vectors are concatenated or averaged in order to predict the next word in the context.

The second model, Distributed bag-of-words of Paragraph Vectors (PV-DBOW) is much like the skip-gram model. Instead of predicting based on

the context of the words, it will predict words randomly sampled from the paragraph.

Which leads to our first proposed model.

4.1.3 Proposed model

The first of the two new models we propose in this paper is the Distributed bag-of-words memory network. The name is derived from the usage of the DBOW architecture for vector representations of documents.

It utilizes the base structure of the end-to-end memory network with one key change. The three word embeddings A, B and C is replaced with the use of a pre-trained distributed bag-of-words (DBOW) NLP. The idea is that by passing these sentences to a paragraph vector NLP will alow the network to learn the semantics of words. Specifically we hope it will be able to identify key words such as restaurant names, locations and numbers as this is essential to solve the goal oriented dialogue task set (more on this later). All the queries and answers are passed through DBOW, where it is inferred a sentence vector. In a sense, every sentence is treated as a never before seen sentence. This means we can pass through sentences with words never before seen by the memory network. As each of the sentences gets an inferred vector, sentences that appear multiple times in the data will have a slight variance in their vector value. However, increasing the training epochs of the DBOW inference will result in more uniform results. This added random noise will hopefully make it easier for the memory network to generalize.

The reason we chose to use the DBOW architecture over the PV-DM is that for the experiments, which we will explain more in depth later, word ordering is not particularly important. Applying the PV-DM architecture would allow us to expand our model to generate sentence answers from the output from the memory network, as opposed to predicting a response from a set of responses. We will however leave this for future work. The model with a single hop can be seen in figure 4.2.

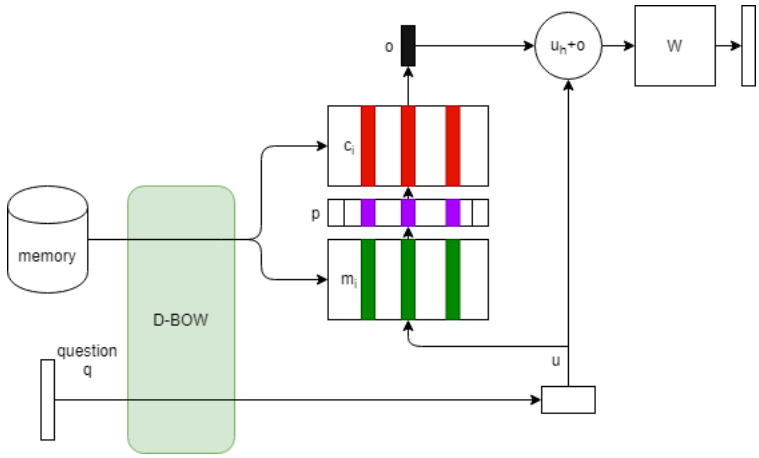

Figure 4.2: The DbowN2N model

4.2 Key-tagging Memory Network

DbowN2N is proposed to address the N2N problems that arise from OOV and real-world data, but the synthetic data can also be improved. Studying results from existing end-to-end Memory Networks in Table 3.1, one can conclude that models using match-feature give better results than those that do not. The next proposed model, the Key-tagging end-to-end Memory Network (KTN2N), is inspired by the match-features 3.3.3. The KTN2N uses a Bi-directional Long Short Term Memory to predict entity properties for each word in the input query and memory.

4.2.1 Bi-directional LSTM

Long Short Term Memory Recurrent Neural Network (LSTM) models are typically used to address problems where information storing is essential. Word generation, as an example, is very hard when information of previous words are lost. It would essentially be a guess, where the most common word would be predicted. The LSTM architecture consists of memory blocks, where each block consists of one or more memory cells [45]. A memory cell is structured to contain relevant information in its state C. Each memory cell takes an input ($[h_{t-1}, x_i]$), which is a concatenated list of sequence input

x_i and previous memory cell output $h_{t-1} = LSTM(h_{t-2}x_{i-1})$. The input is used to decide what to keep from the previous state (C_{i-1}), how to update that state, and what to output.

The memory cell first decides what to keep from the previous state (C_{t-1}) using a forget gate. The forget gate is a Sigmoid function 2.1.3 over the input, producing a new vector f_t where each value is either zero or one.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{4.1}$$

W_f is the forget gate weight matrix that is multiplied with the input. The vector f_t is later multiplied with previous C_{t-1} element-wise. When an element of C_{t-1} is multiplied with a zero-element in the f_t vector, it is set to zero, in other words, forgotten.

$$C'_{t-1} = C_{t-1} \bullet f_t \tag{4.2}$$

The next step of the LSTM architecture is deciding what parts of the state should be updated, and what values the state should be updated with. Both actions are done in the input gate, where a Sigmoid function (i_t) is used to decide what values to update:

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4.3}$$

while a Hyperbolic Tangent 2.1.3 is used on the input to generate an update vector $\tilde{C}_t$ with values between -1 and 1.

$$\tilde{C}_t = tahn(W_c \cdot [h_{t-1}, x_t] + b_i) \tag{4.4}$$

W_i and W_c are separate weight matrices for the input gate actions. The update vector $\tilde{C}_t$ is then multiplied with the filter vector i_t to generate the new update. The update is then added on the filtered previous state C'_{t-1} to produce the new state C_t:

$$C_t = C'_{t-1} + (i_t \bullet \tilde{C}_t) \tag{4.5}$$

The new sate C_t is then passed directly to the next memory cell, but also to the output layer of the memory cell to produce output (h_t).

The output layer acts as a filter of the current state, deciding which values to output using another Sigmoid function o_t,

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_i) \tag{4.6}$$

while also transforming the state vector with another Hyperbolic Tangent function h_t.

$$h_t = o_t \bullet tahn(C_t) \tag{4.7}$$

The output, which is now a vector of values between -1 and 1 are passed to the next memory cell, which first, concatenates the output (h_t) and the new input x_{i+1}, then apply the forget gate, input gate and lastly output gate as shown in Figure 4.3.

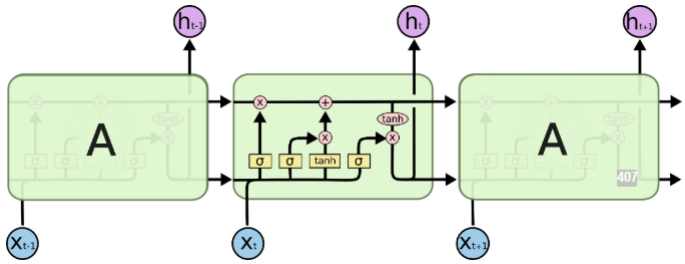

Figure 4.3: memory cell architecture [18]

LSTM has been shown to successfully address the gradient saturation problems typical to Recurrent Neural Network architectures [45]. Bi-directional LSTM is, in essence, two different LSTM layers that are facing in different directions. In other words, input x_0 is provided for memory cell $LSTM_0$ in one layer, and in $LSTM_n$ in the other. The output of A_0 and A_n is first concatenated, then applied an activation function, to form output y_0 as shown in Figure 4.4 [19].

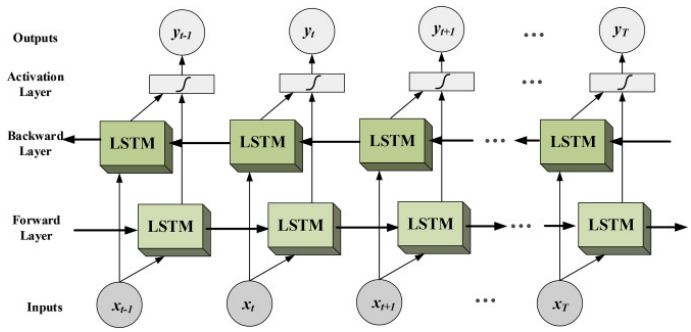

Figure 4.4: bi-directional LSTM architecture [19]

Bi-directional LSTM is typically used to both store information of the past in the first LSTM, while also store information about the future in the second LSTM. As an example, given input sequence: "I speak fluent French", the word "French" would be sent as the first input to the second LSTM, and "fluent" as the second input and so on.

In the proposed model, the input is given as sequences of characters instead of words. Having sequences of characters, the model should be able to perform well, even when there exists out of vocabulary words in the sequence. This was also confirmed during experiments, where the Bi-directional LSTM was able to predict entity properties perfectly across the entire dataset.

4.2.2 Proposed Model

The match feature results from previous N2N architectures were overall better, which means that adding more information to the network should increase the accuracy. The dialog-babi includes a file containing all restaurants for the synthetic data, where each restaurant is mapped to keywords. For example, "resto_london_moderate_british_4stars" is mapped to:

- cuisine: british
- location: london
- price: moderat
- rating: 4

- phone: resto_london_moderate_british_4stars_phone
- address: resto_london_moderate_british_4stars_address

The file can be considered a database for all restaurants in the synthetic data, and can be used to get restaurants from API calls. Also, all other restaurants contain these entity properties, such as Indian is a cuisine of resto_london_expensive_indian_1stars. Indian can be considered a keyword for the N2N network, as essentially, the memory network needs to isolate words that map to entity properties. Typically a sentence containing the word Indian would have something to do with cuisine, while cheap would have something to do with the prize. If one word can represent the entire query, one can assume that tagging the sentence with the entity property of this word would increase the accuracy of the N2N model. This assumption was the original motivation for the proposed KTN2N model.

The KTN2N model is extended from the plain N2N model without match-features and includes a Bi-directional LSTM that predicts the entity properties for each word in the sentence. Each entity property are indexed, where 0 is used for the pad, and 1 is used for words that do not have a entity property representation in the KB file. The sentence: "I would like to order Italian food" would, for example, have a entity property vector: [1, 1, 1, 1, 1, 2, 1], where the value "2" represents the entity property "cuisine".

Both the user queries and the bot replies are fed through the Bi-directional LSTM. Predicting entity properties for bot replies should be important, as this will create a stronger correlation between the API call responses in the memory and the correct candidate. Also, some of these entity properties, as phone and address, are only seen in the bot replies. These entity properties would not exist in the memory if the bot replies did not get passed through the Bi-directional LSTM.

In one of our earlier iterations of the proposed model, we experimented with appending the largest value from the entity property vector to the input sentence. Quickly it was discovered that only appending one entity property was not feasible, since one sentence could include multiple entity properties, such as API calls. The solution was simply operating with the whole entity property vector.

The first operation made with the entity property vector was a multiplication of embedded word x_i and the entity property for that word t_i. Multiply-

ing the word would increase the value of the words that had a feature in the KB file, while having all other words, stay the same. Testing showed that the multiplication confused the model more than it helped, and the final accuracy was considerably decreased in comparison with the regular N2N. Summation of input sentence and entity property vector was also tested, but did not yield good results.

In our tests, concatenation of the reduced embedded word vector and the entity property vector showed potential, as the test results showed that the model's accuracy was similar to the regular N2N. At this point, the N2N did not contain any learning over the entity properties. And the latter part of the concatenated vectors would only contain integers between zero and the number of entity properties + 1. A weight matrix T was introduced, to let the N2N model do learning on the entity properties, which also enabled the entity property vector to have values between $-\infty$ and ∞. The weight matrix T is in the shape [t_size + 1, d], where t_size is the number of entity properties, and d is the embedding size. The weight matrix is used on the entity properties to produce the internal query entity property vector q_t and the memory entity property matrix m_t as:

$$q_t = \sum_i Tz(q)_i \tag{4.8}$$

$$m_{ti} = \sum_j Tz(s)_{ij} \tag{4.9}$$

where z denote the Bi-directional LSTM output, q denote input queries, and s denote input memory. q_t is then concatenated with the reduced weight vector q^{emb} of queries to produce the initial (u^0) as:

$$q^{emb} = \sum_i Aq_i \tag{4.10}$$

$$u^0 = q^{emb} \frown q_t \tag{4.11}$$

where $\frown$ denotes concatenation of the vectors. Now that u^0 is extended to shape $2d$, the W output weight matrix is extended to have shape $V \times 2d$. m_t is concatenated with the memory as:

$$m_i^{emb} = \sum_j As_{ij} \tag{4.12}$$

$$m_i = m_i^{emb} \frown m_{ti} \tag{4.13}$$

Figure 4.5 shows the proposed KTN2N architecture.

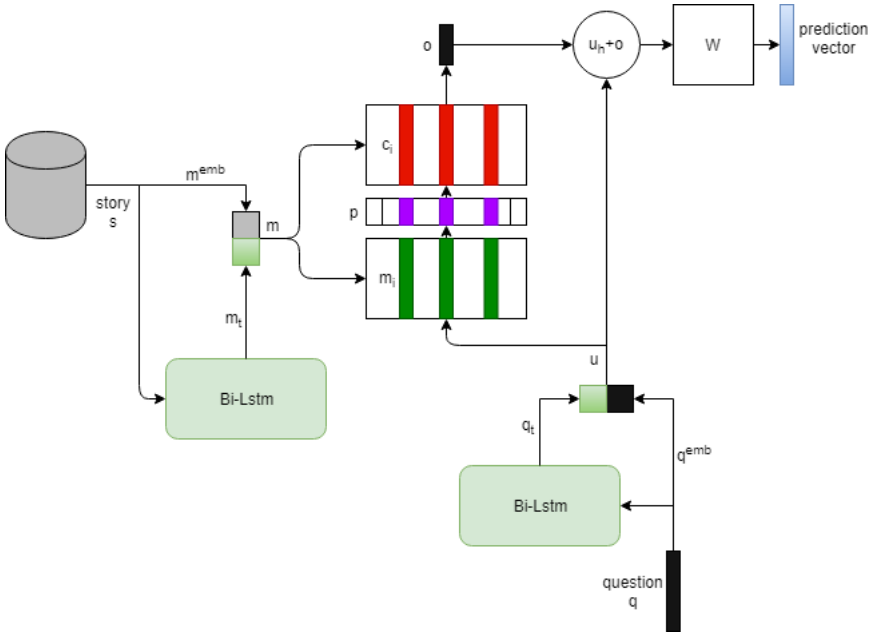

Figure 4.5: High level KTN2N architecture

The proposed model is similar to the model proposed by Byoungjae et al. in [43]. The Bi-directional Memory Network model proposed in [43] concatenates the state of the forward LSTM with the state of the backward LSTM to form u^0 and m. However, in our model we form u^0 and m with the concatenation of both the input and output of the LSTM layer.

Part III

Experiments and Results

Chapter 5

Experiments with Proposed models

5.1 Introduction

Our experiments mainly deal with the babi dataset proposed by Bordes, et al. [16], which was also used in the experiments on the memory networks in chapter 3 and a more thorough explanation can be found there. Additionally, we have some tests with a smaller dataset we created ourselves. This dataset is in the domain of children with divorced parents, where the goal is to find answers to legal questions connected to this.

We perform a series of experiments using both models. We attempt to keep the datasets as similar to the work done previously with memory networks, so that a good comparison between our two new models and the already existing ones can be made.

5.2 Experiments

Initial tests with the DbowN2N network were done using vector embeddings of size 20, similarly to that of other memory networks [16][14][15]. These test results were underwhelming and we attempted additional test by increasing

the embedding size incrementally. The paper proposing paragraph vectors [42] used a vector size of 400. We ended up on a more modest size of a 100, which is still a significant increase of the initial size.

As opposed to the other memory network models, the DbowN2N model appeared to benefit from using up to as many as 8 hops. This is a substantial increase from what was observed previously, where the networks would get diminishing returns after 2 to 4 hops depending on the task. We assume there is a correlation between the increase of embedding size used in this model and the optimum number of hops. As was suggested in the first publication of memory networks [36], more complex tasks require more hops.

Training of both models was run over 100 epochs. The best model was chosen from the validation set and the accuracy results were found using the test set. The training on the Dbow component and the memory component was done separately, running for 100 epochs each.

5.3 Results

5.3.1 Tuning

The hyper parameters studied were:

- sentence size
- memory size
- embedding size
- batch size
- hops
- learning rate

Both sentence size and memory size contributes to set a maximum number of sentences and memory respectively. As long as all sentences and memories are below the maximum, the sentence and memory size will not matter because all padded sentences and memories do not contribute to the

embedded vector representations in the model (u,m). But if sentences are longer than sentence size, keywords might get lost, since the sentence would have to be cut down to sentence size. And if memories are longer than memory size, sentences in the memory would get cut, which might cause loss of information. Setting a large sentence size is preferable to avoid loss of information, but memory size should be tuned such that there is minimal loss of information, while also not having too much information. In our case, a memory size of 50 was better than a memory size of 100, with an average increase of 1 % prediction accuracy.

The embedding size slightly impacted the overall accuracy of the network. An embedding size of 32 was found to give the highest results.

Interestingly batch size impacts were larger than first assumed. A batch size of 256 resulted in quick convergence, but a batch size of 64 resulted in a better convergence. The result could vary as much as 5 % prediction accuracy, where a batch size of 64 was found to give the highest accuracy.

The number of hops also impacted the network overall, and, as shown in Table 5.1, 5 hops resulted in a higher accuracy at task 5.

Hops	Test accuracy
1 hop	95.8
3 hops	98.6
5 hops	99.0
10 hops	96.0

Table 5.1: KTN2N hop comparison

The learning rate was tuned to 0.001. Learning rate over 0.001 would over step as described in 2.2.1.

Cross entropy loss and Hinge loss was tested with Adam optimizer, which revealed that cross entropy loss with Softmax activation function was the way to go. Adadelta optimizer was tested as well, but underperformed in comparison with Adam.

Lastly, we tested the model with data shuffling, where the dataset was shuffled at every epoch, but found no real impact on the prediction accuracy after training.

5.3.2 DbowN2N

In addition to testing the capabilities of paragraph vector representation, we wanted to confirm that randomly initialized embeddings worked better than word2vec initialized embeddings for these tasks. The results for our test can be seen in Table 5.2 and seem to conform with results in previous work done by Byoungjae et al. [43].

Task	RIE	Word2Vec
5. Organizing entire dialogues calls	96.1	67.2

Table 5.2: Comparison of Random Initialized Embedding vs Word2Vec Embedding

As seen in Table 5.3, the DbowN2N network performed very similarly to that of the baseline memory network without match features. However, it is important to note that in order to get similar results, the DbowN2N network required a considerable larger embedding size. In spite of this, the computational time was not changed to drastically. We can assume that this is because it only needs to train a single embedding for the sentences, instead of 3.

We can observe that it still is not able to solve task 4 and the OOV-tasks as we were hoping. One explanation for this could be that the dbow model is not able to infer meaning to the keywords as we intended. The reason for this could be that the variation of the sentence structure in the datasets is very limited. In order to explore this issue, we did a test where the Dbow component was trained using the training set from task 6, containing data from real dialogues. This means that every dialogue from both the training set and test set contained unknown restaurant properties and the sentence structures varied much more. Additionally, the training set is a much larger training set. The results can be seen in Table 5.4 and show some improvement.

Task	plain N2N	DbowN2N
1. Issuing API calls	99.9	99.2
2. Updating API calls	100	99.4
3. Displaying options	74.9	74.1
4. Generating additional information	59.5	57.2
5. Organizing entire dialogues calls	96.1	95.7
1. (OOV) Issuing API calls	72.3	68.7
2. (OOV) Updating API calls	78.9	77.8
3. (OOV) Displaying options	74.4	61.8
4. (OOV) Generating additional information	57.6	57.0
5. (OOV) Organizing entire dialogues calls	65.5	59.9
6. Dialog state tracking 2		41.9

Table 5.3: Results of The DbowN2N network on the babi dataset

Task	DbowN2N	
	internal dataset	external dataset
1. (OOV) Issuing API calls	68.7	81.3
2. (OOV) Updating API calls	77.8	78.9
3. (OOV) Displaying options	61.8	72.1
4. (OOV) Generating additional information	57.0	57.0
5. (OOV) Organizing entire dialogues calls	59.9	65.7

Table 5.4: Comparison of using training data from the respective tasks and an external dataset with the DbowN2N

5.3.3 KTN2N

Before the final model of KTN2N were established, many experiments were done, but they were all outperformed by the plain N2N model without match type. Eventually, we came to the current proposed model giving intriguing results, which can be seen in Table 5.5.

Task	N2N	N2N w/match	KTN2N
1. Issuing API calls	99.9	100	99.6
2. Updating API calls	100	98.3	100
3. Displaying options	74.9	74.9	74.9
4. Generating additional information	59.5	100	57.2
5. Organizing entire dialogues calls	96.1	93.4	99.0
average	86.1	93.3	86.1
1. (OOV) Issuing API calls	72.3	96.5	78.4
2. (OOV) Updating API calls	78.9	94.5	78.9
3. (OOV) Displaying options	74.4	75.2	73.4
4. (OOV) Generating additional information	57.6	100	57.0
5. (OOV) Organizing entire dialogues calls	65.5	77.7	65.6
average	69.7	88.8	70.7

Table 5.5: KTN2N vs baseline N2N with and without match features

However, we observe that KTN2N still fails taks 4 and that it does not provide as good of a solution to the OOV problem as the match-type feature provides.

Interestingly without including API call responses in the data, the KTN2N performed better on OOV task. A test on task 5 resulted in a score of 98.8 % accuracy for the test set, and 67.5 % for the out-of-vocabulary test set. The theory is that the Network is able to match the API calls sentences to the actual restaurant response, so including the API call responses might be ambiguous or confusing. However, This is not the intended way, and looking at the data, one can see that there is no order of the API call responses. Ordering them from, for example, lowest to highest rated might result in higher performance of the network.

5.3.4 Final Comparison

Below follows a table determining how the two proposed models compare against the baseline end-to-end memory network (N2N), the Gated end-to-end (GN2N), and the Unified weight tying end-to-end memory network (UN2N). The proposed models are compared against the N2N models with match features, seeing that the baseline N2N results were better on average with match features.

Task	N2N	GN2N	UN2N	DbowN2N	KTN2N
1. Issuing API calls	100	100	100	99.2	99.6
2. Updating API calls	98.3	100	100	99.4	100
3. Displaying options	74.9	74.9	74.9	74.1	74.9
4. Generating additional information	100	100	100	57.2	57.2
5. Organizing entire dialogues calls	93.4	98.0	99.4	95.7	99.0
average	93.3	94.6	94.8	85.1	86.1
1. (OOV) Issuing API calls	96.5	100	100	68.7	78.4
2. (OOV) Updating API calls	94.5	94.5	94.5	77.8	78.9
3. (OOV) Displaying options	75.2	75.1	76.3	61.8	73.4
4. (OOV) Generating additional information	100	100	100	57.0	57.0
5. (OOV) Organizing entire dialogues calls	77.7	79.4	79.5	59.9	65.6
average	88.8	89.7	89.8	65	70.7

Table 5.6: Comparison of proposed models and state of the art N2N for synthetic data

As shown in Table 5.6, both the DbowN2N and the KTN2N fails on task 4, but when using the validation set as training data, KTN2N is able to get an accuracy of 92 %. The reason for the sharp drop when using the validation set as training set is not yet determined, but might suggest that there are some key differences between the training and validation set in task 4.

Although the KTN2N failed on task 4, it showed a good performance for task 5, containing the whole dialog. With an accuracy of 99 %, the KTN2N model was only outperformed by the UN2N model on task 5. OOV results on the other hand were not promising, as both the DbowN2N and the KTN2N were outperformed on all of these tasks.

5.3.5 Additional Experiment

We also tried to compare our proposed models against the baseline N2N model using another dataset. The dataset was obtained from a youth information service, with 50 dialogues between children with divorced parents and counselor. The dataset does not include task 4, as seen in the babi dataset, but includes task 1, 2, 3, and 5. As shown in Table 5.7, the dataset is too small to be able to get any real value, but KTN2N seems to outperform the N2N on average.

Task	N2N	KTN2N	DBowN2N
1. Issuing API calls	71.4	71.4	57.1
2. Updating API calls	50	50	50.0
3. Displaying options	57.1	71.4	57.1
5. Organizing entire dialogues calls	60	60	50.0
average	59.6	63.2	53.6

Table 5.7: KTN2N and DBowN2N versus regular N2N over Divorce-dialog dataset

Chapter 6

Conclusion

There are many benefits of using goal-oriented dialog systems and their application can be used in a wide range of domains. Building such a system with rule-based architecture can be complex when dealing with real world dialog as data. Another approach to building such a system is by using the end-to-end memory network architecture, which has shown promising results.

The state of the art end-to-end Memory Networks (N2N) uses bag-of-words embeddings, where each word in the vocabulary is randomly initialized. Using a pre-trained Word2Vec model's embedding to initialize the embeddings of N2N should give more correlations between the words. However, initializing the word embeddings with Word2Vec did not yield good results. In fact, the accuracy of the N2N network decreased considerably with a Word2Vec embedding. The reason for the accuracy drop is not yet determined, but a likely candidate is that randomly initialized embeddings can be considered better than Word2Vec, for N2N.

The results from our distributed bag-of-word memory network (DBOWN2N) was very similar to that of the plain N2N network which it originates from. This suggests that the word embeddings are equally effective as paragraph vectors for these tasks. However, the results by using randomly initialized bow-embeddings can vary significantly from each execution as you could get lucky or unlucky with the initialization. The results that are presented from the plain memory network are the highest from several tests, and the

DbowN2N results stay much more similar over multiple tests.

Another thing to consider is that one of the main strengths of the paragraph vector architecture is its ability to process a large amount of unlabeled data. We limited ourselves to using only the data for each respective task for the pre-trained DBOW. However, our experiments show that adding more data from a different source improves the networks ability to solve these kinds of tasks, especially the OOV-tasks.

Tagging key-words shows promising results, where the accuracy varies in terms of how the tagging information is presented to the memory network. Tagging a whole sentence as a single tag did not produce a higher accuracy than the baseline N2N model. However, concatenating the sentence vectors with a tag vector, containing the placement and type of key words, increases the baseline accuracy overall. Character-based Bi-directional LSTM provided high accuracy for predicting tags. Concatenating the input sentences with the corresponding tag vector produced higher accuracy than an element-wise summation. The proposed model Key-tagging end-to-end Memory Network (KTN2N) had an accuracy of 99.0 % for full dialog data (task 5), which is an increase of 5.6 % over the baseline memory network with match features and is only narrowly beaten by the unified weight-tying network.

References

[1] G. Parmigiani, "Decision theory: Bayesian," in *International Encyclopedia of the Social & Behavioral Sciences* (N. J. Smelser and P. B. Baltes, eds.), pp. 3327 – 3334, Oxford: Pergamon, 2001.

[2] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," *arXiv e-prints*, p. arXiv:1811.03378, Nov 2018.

[3] V. K. Ojha, A. Abraham, and V. Snášel, "Metaheuristic design of feedforward neural networks: A review of two decades of research," *Engineering Applications of Artificial Intelligence*, vol. 60, pp. 97 – 116, 2017.

[4] A. Karpathy, A. Simpelo, O. Caglayan, M. Bosnjak, and J. Chan, "Optimization: Stochastic gradient descent." [online] www.cs231n.github.io, Available at: `http://cs231n.github.io/optimization-1/`. [Accessed 10 May. 2019].

[5] D. Britz, "Recurrent neural networks tutorial, part 1 – introduction to rnns." [online] www.wildml.com, Available at: `http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/`. [Accessed 02 May. 2019].

[6] OpenCV, "Introduction to support vector machines." [online] www.docs.opencv.org, Available at: `https://docs.opencv.org/2.4.13.7/doc/tutorials/ml/introduction_to_svm/introduction_to_svm.html`. [Accessed 10 May. 2019].

[7] A. Agrawal, "Loss functions and optimization algorithms. demystified.." [online] www.medium.com, Available at: `https:`

//medium.com/data-science-group-iitr/loss-functions-and-optimization-algorithms-demystified-bb92daff331c. [Accessed 02 May. 2019].

[8] P. Grover, "5 regression loss functions all machine learners should know." [online] www.heartbeat.fritz.ai, Available at: https://heartbeat.fritz.ai/5-regression-loss-functions-all-machine-learners-should-know-4fb140e9d4b0. [Accessed 03 May. 2019].

[9] B. Fortuner, M. Viana, and B. Kowshik, "Loss functions." [online] www.ml-cheatsheet.readthedocs.io, Available at: https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html. [Accessed 06 May. 2019].

[10] S.-H. Tsang, "Review: Retinanet - focal loss (object detection)." [online] www.towardsdatascience.com, Available at: https://towardsdatascience.com/review-retinanet-focal-loss-object-detection-38fba6afabe4. [Accessed 10 May. 2019].

[11] R. Sagar, "Decoding kl divergence and its significance in machine learning." [online] www.analyticsindiamag.com, Available at: https://www.analyticsindiamag.com/decoding-kl-divergence-and-its-significance-in-machine-learning/. [Accessed 10 May. 2019].

[12] S. Ruder, "An overview of gradient descent optimization algorithms." [online] www.ruder.io, Available at: http://ruder.io/optimizing-gradient-descent/. [Accessed 10 May. 2019].

[13] S. Sukhbaatar, A. Szlam, J. Weston, and R. Fergus, "End-to-end memory networks," *Proceedings of Advances in Neural Information Processing Systems (NIPS 2015)*, vol. 1, pp. 2440 – 2448, 2015.

[14] F. Liu and J. Perez, "Gated end-to-end memory networks," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers*, (Valencia, Spain), pp. 1–10, Association for Computational Linguistics, Apr. 2017.

[15] F. Liu, T. Cohn, and T. Baldwin, "Improving end-to-end memory networks with unified weight tying," in *Proceedings of the Australasian Language Technology Association Workshop 2017*, (Brisbane, Australia), pp. 16–24, Dec. 2017.

[16] A. Bordes, Y.-L. Boureau, and J. Weston, "Learning End-to-End Goal-Oriented Dialog," *arXiv e-prints*, p. arXiv:1605.07683, May 2016.

[17] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *arXiv e-prints*, p. arXiv:1301.3781, Jan 2013.

[18] C. Olah, "Understanding lstm networks," Aug 2015.

[19] O. Yildirim, "A novel wavelet sequence based on deep bidirectional lstm network model for ecg signal classification," *Computers in Biology and Medicine*, vol. 96, pp. 189 – 202, 2018.

[20] J. Brownlee, "Difference between classification and regression in machine learning." [online] www.machinelearningmastery.com, Available at: `https://machinelearningmastery.com/classification-versus-regression-in-machine-learning/`. [Accessed 03 May. 2019].

[21] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal Loss for Dense Object Detection," *arXiv e-prints*, p. arXiv:1708.02002, Aug 2017.

[22] L. Rosasco, E. De Vito, A. Caponnetto, M. Piana, and A. Verri, "Are loss functions all the same?," *Neural computation*, vol. 16, pp. 1063–76, 06 2004.

[23] D.-A. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)," *arXiv e-prints*, p. arXiv:1511.07289, Nov 2015.

[24] L. Trottier, P. Giguère, and B. Chaib-draa, "Parametric Exponential Linear Unit for Deep Convolutional Neural Networks," *arXiv e-prints*, p. arXiv:1605.09332, May 2016.

[25] M. Basirat and P. M. Roth, "The Quest for the Golden Activation Function," *arXiv e-prints*, p. arXiv:1808.00783, Aug 2018.

[26] T. Araujo, "Living up to the chatbot hype: The influence of anthropomorphic design cues and communicative agency framing on conversational agent and company perceptions," *Computers in Human Behavior*, vol. 85, pp. 183 – 189, 2018.

[27] J. Hill, W. R. Ford, and I. G. Farreras, "Real conversations with artificial intelligence: A comparison between human–human online conversations and human–chatbot conversations," *Computers in Human Behavior*, vol. 49, pp. 245 – 250, 2015.

[28] S. Quarteroni, "A chatbot-based interactive question answering system," *11th Workshop on the Semantics and Pragmatics of Dialogue*, pp. 83 – 90, 2007.

[29] S. Young, M. Gašić, B. Thomson, and J. D. Williams, "Pomdp-based statistical spoken dialog systems: A review," *Proceedings of the IEEE*, vol. 101, pp. 1160–1179, May 2013.

[30] B. Shawar and E. Atwell, "Chatbots: Are they really useful?," *LDV Forum*, vol. 22, pp. 29–49, 01 2007.

[31] R. Jafari and W. Yu, "Uncertainty nonlinear systems control with fuzzy equations," in *2015 IEEE International Conference on Systems, Man, and Cybernetics*, pp. 2885–2890, Oct 2015.

[32] I. Sutskever, J. Martens, and G. E. Hinton, "Generating text with recurrent neural networks," *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1017–1024, 01 2011.

[33] O. Vinyals, A. Toshev, S. Bengio, and D. Erhan, "Show and Tell: A Neural Image Caption Generator," *arXiv e-prints*, p. arXiv:1411.4555, Nov 2014.

[34] J. D. Williams and G. Zweig, "End-to-end LSTM-based dialog control optimized with supervised and reinforcement learning," *arXiv e-prints*, p. arXiv:1606.01269, Jun 2016.

[35] A. Kumar, O. Irsoy, P. Ondruska, M. Iyyer, J. Bradbury, I. Gulrajani, V. Zhong, R. Paulus, and R. Socher, "Ask Me Anything: Dynamic Memory Networks for Natural Language Processing," *arXiv e-prints*, p. arXiv:1506.07285, Jun 2015.

[36] J. Weston, S. Chopra, and A. Bordes, "Memory Networks," *arXiv e-prints*, p. arXiv:1410.3916, Oct 2014.

[37] J. Dodge, A. Gane, X. Zhang, A. Bordes, S. Chopra, A. Miller, A. Szlam, and J. Weston, "Evaluating Prerequisite Qualities for Learning End-to-End Dialog Systems," *arXiv e-prints*, p. arXiv:1511.06931, Nov 2015.

[38] M. Henderson, B. Thomson, and S. Young, "Word-based dialog state tracking with recurrent neural networks," in *SIGDIAL 2014 Conference*, pp. 292–299, 01 2014.

[39] R. K. Srivastava, K. Greff, and J. Schmidhuber, "Highway Networks," *arXiv e-prints*, p. arXiv:1505.00387, May 2015.

[40] M. Henderson, B. Thomson, and J. D. Williams, "The second dialog state tracking challenge," in *Proceedings of the 15th Annual Meeting of the Special Interest Group on Discourse and Dialogue (SIGDIAL)*, (Philadelphia, PA, U.S.A.), pp. 263–272, Association for Computational Linguistics, June 2014.

[41] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[42] Q. V. Le and T. Mikolov, "Distributed Representations of Sentences and Documents," *arXiv e-prints*, p. arXiv:1405.4053, May 2014.

[43] B. Kim, K. Chung, J. Lee, J. Seo, and M.-W. Koo, "A bi-lstm memory network for end-to-end goal-oriented dialog learning," *Computer Speech & Language*, vol. 53, pp. 217 – 230, 2019.

[44] T. Mikolov, W. tau Yih, and G. Zweig, "Linguistic regularities in continuous space word representations," in *HLT-NAACL*, 2013.

[45] Z. Yu, V. Ramanarayanan, D. Suendermann-Oeft, X. Wang, K. Zechner, L. Chen, J. Tao, A. Ivanou, and Y. Qian, "Using bidirectional lstm recurrent neural networks to learn high-level abstractions of sequential features for automated scoring of non-native spontaneous speech."

UiA University of Agder
Master's thesis
Faculty of Engineering and Science
Department of ICT