IMPLEMENTATION OF NEURAL EMBEDDING BASED MODELS FOR SENTENCE CLASSIFICATION

[1]Siddharth Sundararajan [2]Shirish Shevade

[1,2]Department of Computer Science and Automation,
Indian Institute of Science, Bangalore - India

WORKING DRAFT

Abstract. *In this report, we consider the sentence classification problem using neural embedding based models. We experiment with and without attention mechanisms. Model parameters and word embeddings are optimized by implementing five gradient descent based algorithms using mini-batch method. Three datasets are used to evaluate the performance of various models.*

1. Introduction

Categorization of documents or short texts arise in text applications. A sentence classification problem is defined as predicting a class label for few sentences. Example applications include classifying movie or restaurant or product reviews and detecting subjectivity or objectivity of sentences [4]. Depending on the class label information available, the problem can be posed as a binary or multi-class problem. In a binary classification problem, an example belongs to one of two categories, e.g., {good, bad} encoded as {+1,-1}. In a multi-class problem, the number of categories is more than two, e.g., {good, bad, neutral} encoded as {1,2,3}.

A data corpus comprises of a collection of examples that are used to build classifier models. A vocabulary $\mathcal{V}$ representing the collection of unique words present is useful for data representation. To build good classifier models that generalize well, we need a good method to represent sentences. As sentences consists of words, representation of words and methods to find sentence representation by combining representations of words are necessary. Two popular methods are Bag Of Words (BOW) and Word Embeddings [1]. In the BOW method, each document is represented by a vocabulary dimensional vector as normalized frequency of words in a sentence. Word embedding refers to representing each word in the vocabulary as a d-dimensional vector. For example, Google's word2vec comprises of a 300-dimensional vector for nearly 3 million English words and phrases [1] learned from a huge News corpus data using Skip-gram method. Given word embeddings, how do we represent sentences? A few commonly used methods to represent a sentence with the help of word representations are simple linear combiner, many variants of Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN) [4][5].

To get a good representation for a sentence, it is important to know that not all words in a sentence are important. For instance, a review - "The movie was bad and disappointing." should focus on the phrase 'bad and disappointing' to understand that type of review. Therefore, the problem of identifying words or phrases that are important to get the class label is important. A mechanism that finds the right weight (i.e, learn the importance) to be provided for each word is called attention based mechanism [5]. In this work, we consider the problem of designing attention based classifier models. In particular, we represent a sentence as a linear combination of word embeddings with attention weights designed using a neural network model. We experiment with and without attention based models. As a first step, we have currently implemented a linear classifier model without attention based mechanism. Our code is currently available in github [1].

To get the right label for a sentence, the model has to learn the best parameters. There are three sets of parameters: classifier model parameters, sentence representation parameters (along with word embeddings) and attention model parameters; and, optimization can be done for various combinations of these sets. In our implementation so far, classifier model parameters and word embeddings are optimized by using an alternate optimization algorithm.

This report is organized as follows. In section 2, we give details about sentence representation and the model implemented. As part of section 3, we talk about optimizing the model parameters and word embeddings. This is followed by experiments conducted and corresponding results. We conclude with next steps where a brief discussion on non-linear models with attention mechanism is given.

2. Attention based sentence representation

2.1. Sentence Representation

Each sentence is represented by combining representation of the words present in it. All the unique words present in the data corpus are stored in a vocabulary and each word is represented as a single point in a d-dimensional space. In this work, a 300 dimensional vector is used to represent each word [1].

A vocabulary containing m words is given as,

$$\mathcal{V} = \{\langle z_k, v_k \rangle : k = 1, 2, 3 \dots m\} \tag{1}$$

where $v_k \,\epsilon\, R^d$, d = 300 and v_k is the word representation for the word z_k.

There are three cases we can consider in a linear combiner model. They are:

Case 1: All words have equal weights
The i^{th} sentence is represented as:

$$x_i = \frac{1}{l_i} \sum_{j=1}^{l_i} \mathcal{V}(z_{i,j}) \tag{2}$$

where l_i is the length of the i^{th} sentence and $\mathcal{V}(z_{i,j})$ is the word embedding for the j^{th} word $z_{i,j}$ in the i^{th} sentence.

[1] https://github.com/Siddharthss500/Implementation-of-Neural-Embedding-models-for-Text

Case 2: Context independent weights
We associate a weight for each word in the vocabulary; i.e,

$$\mathcal{A} = \{\langle z_k, a_k \rangle : k = 1, 2, 3 \ldots m\} \tag{3}$$

The i^{th} sentence is represented as:

$$x_i(a) = \frac{1}{\sum_j a(z_{i,j})} \sum_{j=1}^{l_i} a(z_{i,j}) \mathcal{V}(z_{i,j}) \tag{4}$$

where $a(z_{i,j})$ is the weight associated with the word $z_{i,j}$.

Case 3: Context dependent weights
We associate a context dependent weight for each word where the context comes from the way words are used in a sentence. The i^{th} sentence is represented as:

$$x_i(a) = \frac{1}{\sum_j a_i(z_{i,j})} \sum_{j=1}^{l_i} a_i(z_{i,j}) \mathcal{V}(z_{i,j}) \tag{5}$$

Note that the weight factor $a_i(z_{i,j})$ has a subscript i indicating the context dependency on the i^{th} sentence.

Case 1 and case 2 are used as baseline models to compare the attention based model (case 3). The goal is to learn the attention model weights as part of training. So far, case 1 is implemented.

2.2. Classifier model

A classifier model is built to classify a sentence into its corresponding class label. There are two types of classifier models, namely, linear and non-linear classifier models. We will be experimenting with both models. As the first step, a linear binary classifier model is implemented to classify a sentence as {+1,-1}. Subsequent implementations will support non-linear models such as neural networks. For classifying a sentence as {+1,-1}, a logistic regression model is implemented with the sentence representation as input.

$$P(y_i|x_i; w, v, a, \bar{w}) = \frac{1}{1 + \exp(-y_i w^T \phi(x_i(v, a); \bar{w}))} \tag{6}$$

where y_i is the class label, w is the logistic regression classifier model parameter, a is the weight vector with elements specifying the weight to each word for a sentence, $x_i(v, a)$ is the sentence representation vector and $\bar{w}$ is the model parameter for the function $\phi(.)$. The function $\phi(.)$ represents an identity or some parameterized non-linear transformation function such as a neural network (e.g., RNN, CNN).

Based on the optimized model parameters and word embeddings, the class label for the i^{th} sentence is given by:

$$y_i = \begin{cases} +1, & \text{if } P(y_i|x_i; w, v, a, \bar{w}) > 0.5 \\ -1, & \text{if } P(y_i|x_i; w, v, a, \bar{w}) \leq 0.5 \end{cases} \tag{7}$$

2.3. Objective function

The general objective function of the model is defined using a logistic loss function as follows:

$$L(w, v, a, \bar{w}) = \frac{1}{n}\sum_{i=1}^{n} L_i(y_i, \phi(x_i(v, a); w); \bar{w}) + \lambda\|w\|^2 \tag{8}$$

where $L_i(y_i, \phi(x_i(v, a); w); \bar{w}) = log(1 + \exp(-y_i w^T \phi(x_i(v, a); \bar{w})))$

We have added a regularization term $\lambda\|w\|^2$ to control the complexity of the classifier; λ is the regularization constant. Similar terms can be added for $\bar{w}$ as well.

As the first baseline, we consider the identity model.

$$\phi(x_i(v, a)) = x_i(v, a) \tag{9}$$

We experiment with three cases described earlier (i.e, equations (2) - (4)). Later, we will support parameterized non-linear transformation model such as neural network with parameters $\bar{w}$.

Currently, no weight regularization is used in our experiments. Also, only model parameters and word embeddings are being optimized using an alternate optimization technique.

When we compare neural embedding based model model to BOW based model, it is observed that neural embedding based model cannot outperform BOW based model under certain conditions as explained below.

A weight vector $\widetilde{W}$ with k^{th} element representing the weight for word z_k is,

$$\widetilde{W} = \{\langle z_k, \widetilde{w}_k\rangle : k = 1, 2, 3 \ldots m\} \tag{10}$$

is learned in a BOW representation based classifier model.

The comparison of scores for each sentence represented by neural embedding based model and BOW model can be made as follows,

For the BOW model:

$$S(x_i) = \frac{1}{l_i}\sum_{j=1}^{l_i} \widetilde{w}(z_{i,j}) \tag{11}$$

where l_i is the length of the i^{th} sentence. Here, the BOW representation is assumed to be a normalized frequency of words with normalization done using the length of the sentence.

For neural embedding based model:

$$S(x_i) = \frac{1}{l_i}\sum_{j=1}^{l_i} w^T \mathcal{V}(z_{i,j}) \tag{12}$$

where l_i is the length of the i^{th} sentence, w is the classifier model parameter and $\mathcal{V}(z_{i,j})$ is the word embedding for the j^{th} word $z_{i,j}$ in the i^{th} sentence.

When we look at equations (11) and (12), it is seen that the terms $\widetilde{w}(z_{i,j})$ and $w^T\mathcal{V}(z_{i,j})$ are comparable. This is because the term $w^T\mathcal{V}(z_{i,j})$ can be seen as the weight for the word z_{ij} in the neural embedding based model. Let W^* and $\widetilde{W}^*$ be the optimal weight vectors for the neural embedding based model and the BOW model respectively. With V representing the word embedding matrix, VW^* is in the search space of BOW model. Therefore, it is possible that $\widetilde{W}^*$ is a better solution than VW^* for fixed V. This holds true for case 2 (refer equation (4)) also. As long as the neural embedding based model is linear using individual word representations, it will be similar to the conventional BOW model. On the other hand, case 3 (refer equation (5)) is implemented by adding context dependent weights and this makes the model non-linear. More analysis will be done in the upcoming versions.

3. Optimization of model parameters and word embeddings

In our implementation, we use several gradient based algorithms (e.g., momentum, RMSProp, AdaGrad, etc.,) [3] to optimize parameters with samples used as mini-batches. There are several variants available in the literature [3]. They are briefly summarized in appendix.

The gradient for the loss function is obtained with respect to model parameters and word embeddings. The gradient with respect to the model parameters for mini-batch method is given as:

$$\frac{\partial L(w,v,a)}{\partial w} = \frac{1}{B}\sum_{i=1}^{B}\frac{-y_i x_i(v,a)(\exp(-y_i w^T x_i(v,a)))}{1+\exp(-y_i w^T x_i(v,a))} \tag{13}$$

where the gradient is calculated by taking B examples at a time from the entire dataset for case 1.

The gradient with respect to the word embeddings for the sentence x_i is given by:

$$\frac{\partial L_i(w,v,a)}{\partial v_k} = \begin{cases} \frac{1}{l_i}(\frac{-y_i w^T(\exp(-y_i w^T x_i(v,a)))}{1+\exp(-y_i w^T x_i(v,a))}), \text{when the word is present in the } i^{th} \text{ sentence} \\ 0, \text{when the word is not present in the } i^{th} \text{ sentence} \end{cases} \tag{14}$$

where l_i is the length of the i^{th} sentence.

We use an alternate optimization technique when both model parameters and word embeddings are optimized in an alternative manner. The details are given in algorithm 1.

Algorithm 1 Alternate Optimization Technique

```
Require: Gradient descent algorithm parameters = Constants $\{\epsilon, \alpha, \gamma\}$; Learning rates $\{\eta_e, \eta_w\}$
Require: $N$ (# outer iterations), $T_w$ (# epochs for classifier model parameter optimization), $T_e$ (# epochs for word embedding optimization)
Initialize Word embeddings using Google word2vec
Initialize Model parameters using Random number generator
for t = 0:N do
    for t = 0:$T_w$-1 do
        Compute the gradient : $g_t$ (For the model parameters using equation 13)
        Apply the update : $w_t$ (Using a chosen gradient descent algorithm e.g; Momentum,Adagrad using gradient descent algorithm parameters)
    end for
    for t = 0:$T_e$-1 do
        Compute the gradient : $g_t$ (For all word embeddings using equation 14)
        Apply the update for all word embeddings : $\mathcal{V}_t$
        Compute the sentence vector representation for all sentences : $x_t$
    end for
end for
```

4. Experiments

In this section, we present details of various experiments conducted.

4.1. Sentence representation

To represent each word present in the data corpus, word embeddings represented by 300 dimensional vectors learnt from Google News data are used as a starting point [1]. For all the words not present in the data corpus, a randomly generated 300 dimensional vector is assigned. Subsequently, a sentence vector is computed by averaging (refer equation (2)) the word representations of that sentence. To classify the sentences as {+1,-1}, a simple logistic regression model is implemented. The performance of the model is determined by computing the test accuracy. We optimize the objective function with respect to model parameters and word embeddings. We compare different variants of gradient descent algorithms on several datasets.

4.2. Datasets

Three datasets are used in our experiments. They are : Amazon Product Review, IMDB Movie Review and Yelp Restaurant Review [6]. Each dataset has positive and negative examples labelled as +1 or -1 corresponding to positive or negative review. There are 1000 examples in each dataset with 500 positive and negative examples. The datasets are pre-processed similar to the method used by Mikolov [1]. Each of the three datasets are separated into training, validation and test data. The training dataset consists of 60% of the data and, validation and test datasets comprise of 20% each.

4.3. Training

As learning rate η plays a crucial role in the optimization problem, different learning rates were tested. For each gradient descent algorithm, the optimal learning rates were chosen by comparing the accuracy on the validation set for different η values. Further, since we do not use any regularization it is possible for the model to overfit after certain number of iterations. Therefore, we chose the best model over the iterations using validation accuracy.

There are two implementations. In the first implementation only classifier model parameters are optimized with word embeddings fixed using Google word2vec. In the second implementation, we also optimize word embeddings initialized with Google word2vec. Model parameters and word embeddings are optimized using an alternate optimization technique (refer algorithm 1). We have implemented five popular gradient descent based algorithms [3] with samples used as mini-batches.

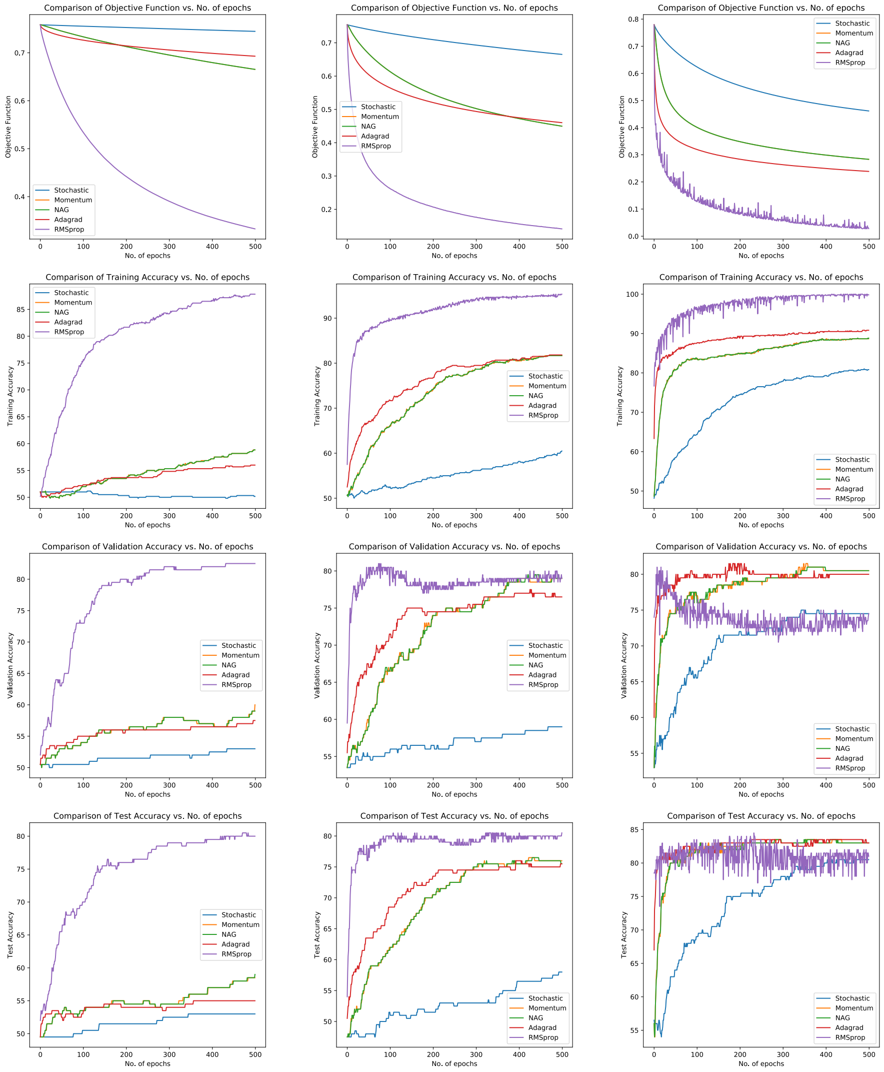

Figure 1. Optimization Results by optimizing Model paramaters and Word embeddings (From left to right the η values used are 0.001, 0.01 and 0.1)

4.4. Observations

In this subsection, we present our results and make key observations.

Figure 1 shows the comparison of optimization algorithms for the amazon product review dataset across three values of η, which are, 0.001, 0.01 and 0.1. The first row in the figure shows the objective function value decrease as a function of iterations. The second, third and fourth rows compare the training, validation and test accuracies against the number of iterations. We observe that learning rate η plays a crucial role while optimizing parameters.

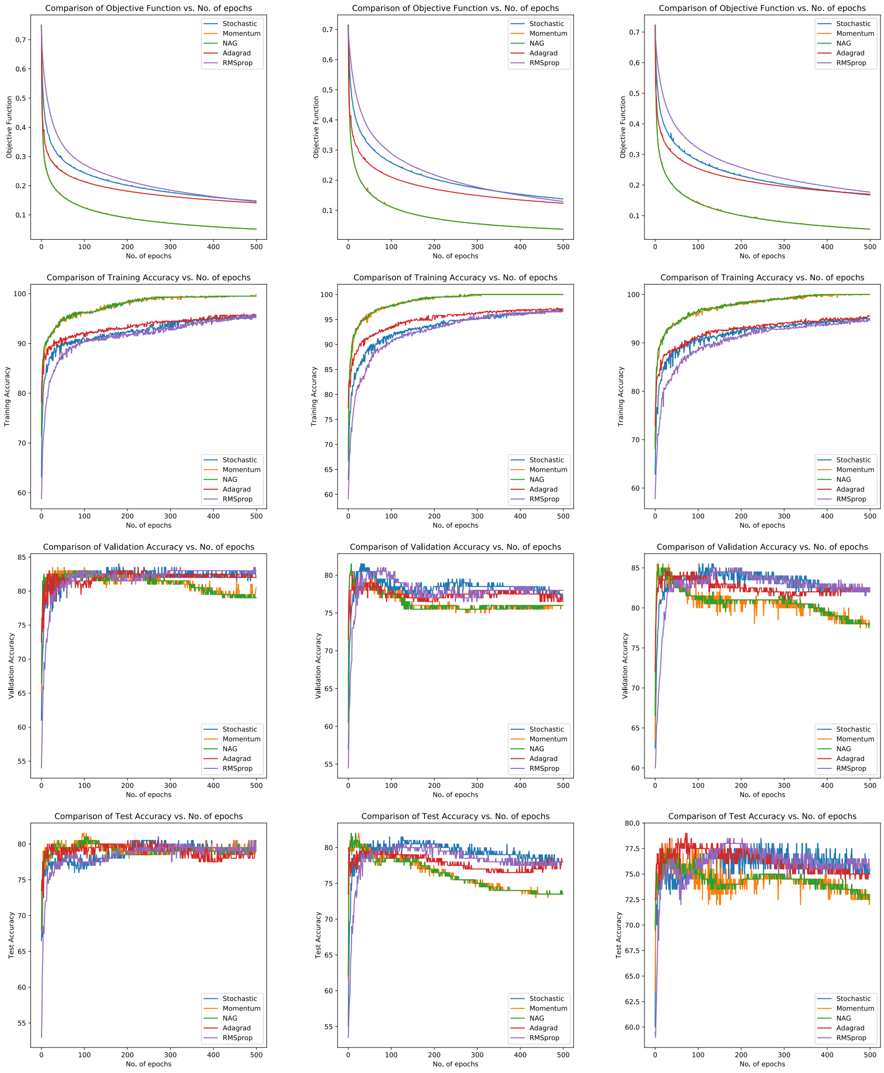

Figure 2. Optimization Results by only optimizing Model Paramaters (From left to right - Amazon movie review dataset, IMDB movie review dataset, Yelp restaurant review dataset)

In Table 1, we compare the performance of five different gradient based algorithms across three η values. It is seen that when we increase the value of η various algorithms tend to converge faster. However, we cannot indefinitely increase the η value as the algorithm will become unstable. This experiment was initially conducted to identify the best learning rates for different gradient descent based algorithms.

In Figure 2, we see the performance of the model for five different gradient descent algorithms across mini-batch gradient method while optimizing the model parameters for three datasets. It is observed that the objective function value is decreasing steadily over the iterations run. Consequently, the training, validation and test accuracy are also improving as the objective function value improves. It is seen that Momentum and Nesterov Accelerated Gradient (NAG) perform better compared with other methods. We also observe that overfitting takes place, but this part is taken care by considering the best validation accuracy as seen in the table 2.

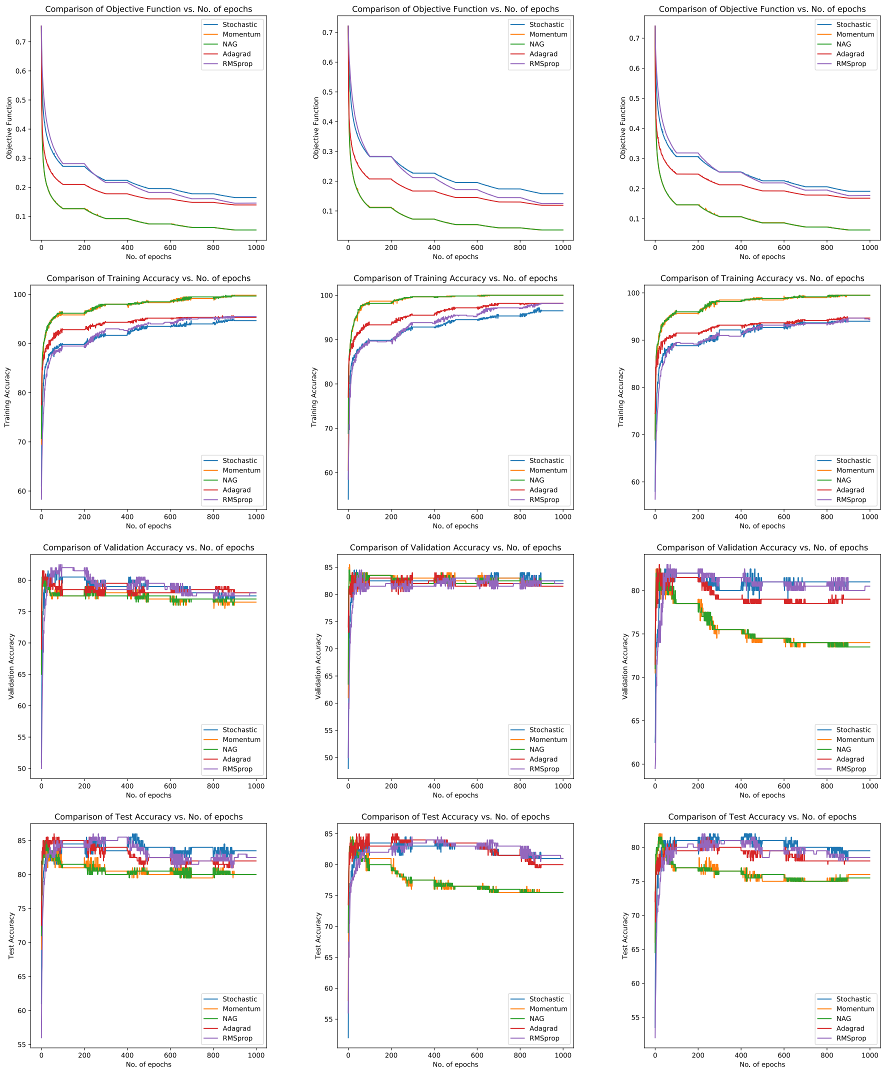

Figure 3. Optimization Results by optimizing Model paramaters and Word embeddings (From left to right - Amazon movie review dataset, IMDB movie review dataset, Yelp restaurant review dataset)

Table 2 compares the training, validation and test accuracies across five gradient descent based algorithms for three datasets. The datasets are Amazon product review, IMDB movie review and Yelp restaurant review. The epoch number corresponds to the first time the best validation accuracy is observed. The test accuracy seen is the value at that particular epoch number. From the table, it is observed that for similar validation accuracies Nesterov Accelerated Gradient (NAG) performs well considering the number

Eta values	(Best Validation Accuracy,Epoch Number, Corresponding Test Accuracy)				
	Gradient Descent	Momentum	NAG	Ada grad	RMS Prop
$\eta = 0.001$	(53,434,53)	(60,499,58.5)	(59,492,58.5)	(57.5,495,55)	(82.5,431,80)
$\eta = 0.01$	(59,467,57)	(79,397,75.5)	(79.5,420,76)	(77.5,425,75.5)	(81,73,79)
$\eta = 0.1$	(75,342,79.5)	(81.5,348,83)	(81,356,83.5)	(81.5,174,83)	(81,7,81)

Table 1. Comparison of Best validation accuracy for different gradient descent algorithms across different η values

of epochs and test accuracy.

Dataset Name	(Best Validation Accuracy,Epoch Number, Corresponding Test Accuracy)				
	Gradient Descent	Momentum	NAG	Ada grad	RMS Prop
Amazon Product Review	(84,180,80)	(83.5,26,79)	(83,31,80)	(83.5,225,78.5)	(83.5,303,77.5)
IMDB Movie Review	(81.5,29,78.5)	(81,7,80)	(81.5,7,82)	(80,9,78.5)	(81,68,79.5)
Yelp Restaurant Review	(85.5,102,76)	(85.5,5,75)	(85.5,7,73.5)	(85,24,76)	(85,132,76.5)

Table 2. Comparison of Best validation accuracy for different gradient descent algorithms across three datasets

In Figure 3, we see the performance of the model for five different descent algorithms with the mini-batch gradient descent method while optimizing model parameters alternatively with word embeddings for three datasets. It is observed that the model improves as seen by the increase in the training, validation and test accuracy. But, as the no. of iterations increase, the model starts to overfit. To overcome this, the validation accuracy is also calculated and the best validation accuracy is chosen for each optimization algorithm. Also, it is seen that when compared with figure 2, there is not much improvement in the model performance during the word embeddings optimization phase. This maybe because the Google word2vec chosen initially is itself a good representation for this particular problem and requires further investigation.

5. Next Step

We have experimented so far only with linear models without attention mechanism. The linear models are built to serve as baselines. Our next goal is to work with non-linear models and attention mechanisms [5]. These models will be implemented using tensorflow.

References

[1] Tomas Mikolov, Kai Chen, Greg Corrado, Jeffrey Dean. *Efficient Estimation of Word Representations in Vector Space*, arXiv:1301.3781

[2] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. *Distributed Representations of Words and Phrases and their Compositionality*, arXiv:1310.4546

[3] Sebastian Ruder. *An overview of gradient descent optimization algorithms*, arXiv:1609.04747

[4] Yoon Kim. *Convolutional Neural Networks for Sentence Classification*, arXiv:1408.5882

[5] Dzmitry Bahdanau, KyungHyun Cho, Yoshua Bengio. *Neural Machine Translation by Jointly Learning to Align and Translate*, arXiv:1409.0473v7

[6] Sentiment Labelled Sentences Data Set; A published dataset source: https://www.kaggle.com/rahulin05/sentiment-labelled-sentences-data-set

Appendices

Appendix A Optimization Algorithms

In this report, we consider different gradient descent algorithms [3] in combination with various gradient descent variants to optimize the objective function for a sentence classification problem. The most basic approach that is used to compute the weight update is the gradient descent algorithm given by:

$$\Delta w = -\eta g_t \tag{15}$$

where g_t is the gradient of the parameters at the t^{th} iteration $\frac{\partial L(w_t)}{\partial w_t}$ and η is a learning rate which controls how much it must move in the negative gradient direction.

A.1 Gradient Descent Variants

Gradient descent variants [3] are chosen based on the sample size of the dataset that is considered while computing the gradient of the objective/loss function. The three different variants are:

A.1.1 Batch Gradient Descent Method

In this method, the gradient is calculated by taking the entire dataset at a time. This can be represented as :

$$w_{t+1} = w_t - \eta \bigtriangledown_{w_t} L(w_t) \tag{16}$$

As the complete dataset is taken at a time to compute the gradient, this method can be slow due to the computations for large datasets. Hence, this method is generally avoided while considering large datasets.

A.1.2 Stochastic Gradient Descent Method

In this method, the gradient is calculated by taking a random example from the dataset at a time. This is represented below for the example x_i and label y_i as:

$$w_{t+1} = w_t - \eta \bigtriangledown_{w_t} L(w_t; x_i; y_i) \tag{17}$$

As only one example is taken at a time to compute the gradient, the fluctuations are more and hence the objective function value smoothens only over a period of time.

A.1.3 Mini-Batch Gradient Descent Method

In this method, the gradient is computed by taking 'B' examples at a time from the dataset. The representation for this method taking 'B' mini-batches at a time is:

$$w_{t+1} = w_t - \eta \bigtriangledown_{w_t} L(w_t; x_{(i:i+B)}; y_{(i:i+B)}) \tag{18}$$

This method is considered to be best of Batch and Stochastic Gradient descent methods. It reaches its minima in a smoother way as mini-batches of data are taken at a time to compute one gradient update.

A.2 Update Algorithms

Five optimization algorithms [3] are implemented in this experiment. They are:

A.2.1 Gradient Descent

The Gradient Descent method is a simple and elementary method used to optimize the objective function as seen below:

$$w_{t+1} = w_t - \eta \bigtriangledown_{w_t} L(w_t) \tag{19}$$

A.2.2 Momentum

Momentum method is a slightly modified version of the basic gradient descent method. A simple variable is added in this method which helps in accelerating the gradient descent in the right direction in order to reach the optimum point faster. By adding the new update variable the momentum equation can be seen as:

$$\begin{aligned} \Delta w_t &= \alpha \Delta w_t - \eta \bigtriangledown_{w_t} L(w_t) \\ w_{t+1} &= w_t + \Delta w_t \\ w_{t+1} &= w_t - \eta \bigtriangledown_{w_t} L(w_t) + \alpha \Delta w_t \end{aligned} \tag{20}$$

The term α mentioned is generally set to around the value 0.9. One major advantage of this method is that the added term '$\alpha \Delta w$' helps in acceleration and reaches the convergence faster.

A.2.3 Nesterov Accelerated Gradient

Nesterov Accelerated Gradient or NAG is an improved version of the Momentum Algorithm. The improvement is that the rate of the acceleration is controlled in order to slow down once it is near the optimal value. The equation for NAG is seen as:

$$\begin{aligned} \Delta w &= \alpha \Delta w - \eta \bigtriangledown_{w} L(w_t + \alpha \Delta w) \\ w_{t+1} &= w_t + \Delta w \\ w_{t+1} &= w_t - \eta \bigtriangledown_{w} L(w + \alpha \Delta w) + \alpha \Delta w \end{aligned} \tag{21}$$

Similar to the Momentum algorithm the term α is generally set to a value around 0.9. When compared to the Momentum algorithm it is faster as it corrects the direction of the gradient movement for every iteration. Hence, it has a slightly better performance compared to the earlier method.

A.2.4 Adaptive Gradient

Adaptive Gradient or Adagrad is an algorithm which has an adaptive learning technique. This algorithm is generally considered when the data present is limited. The major difference in Adagrad is that it has a different learning rate η for every update. This is because the learning rate is higher for frequent parameters and comparatively lesser for infrequent parameters. The Adagrad equation can be seen as:

$$\begin{aligned}
\Delta w &= \frac{\eta}{\sqrt{r_{i,i} + \epsilon}} \odot \bigtriangledown_w L(w_t) \\
w_{t+1} &= w_t + \Delta w \\
w_{t+1} &= w_t - \frac{\eta}{\sqrt{r_{i,i} + \epsilon}} \odot \bigtriangledown_w L(w_t)
\end{aligned} \tag{22}$$

where $r_{i,i}$ is a diagonal matrix where every diagonal element i, i is the sum of the squares of the gradients w.r.t. w_i up till the update t. The ϵ is called the smoothing term that avoids division by zero error. The ϵ value is generally set to $1e^{-08}$. One major advantage is that the learning rate need not be manually changed for every update. But, the disadvantage is also that over many iterations the learning rate becomes infinitesimally small which affects the learning.

A.2.5 RMS Propagation

RMS propagation model or RMSprop is one of the two methods that were made to overcome the learning rate pitfall problem in the Adagrad algorithm. The flaw is overcome by making the learning rate η dependant only on the recently found gradients. This helps as the decaying rate of η is taken care by this slight modification. The RMSprop equation as proposed by Geoff Hinton is:

$$\begin{aligned}
E[L^2(w)]_t &= 0.9E[L^2(w)]_{t-1} + 0.1L^2(w)_t \\
\Delta w &= \frac{\eta}{\sqrt{E[L^2]_t + \epsilon}} . \bigtriangledown L(w) \\
w_{t+1} &= w_t + \Delta w \\
w_{t+1} &= w_t - \frac{\eta}{\sqrt{E[L^2]_t + \epsilon}} . \bigtriangledown L(w)
\end{aligned} \tag{23}$$

where the term $\sqrt{E[L^2]_t + \epsilon}$ is called the Root Mean Square(RMS). This term is updated based on the formula stated on the first line above. As seen in Adagrad, in order to avoid the *divide-by-zero error* the smoothing term ϵ is added and is generally set to $1e^{-08}$. Also, the gamma value is commonly considered to be 0.9. RMSprop is said to adapt well with the mini-batch method. It also reaches convergence moving in an expedite manner.