TOWARDS AN OPTIMAL SET OF INITIAL WEIGHTS FOR A DEEP NEURAL NETWORK ARCHITECTURE

A. Saadi[*], *H. Belhadef*[†]

Abstract: Modern neural network architectures are powerful models. They have been proven efficient in many fields, such as imaging and acoustic. However, these neural networks involve a long-running and time-consuming process. To accelerate the training process, we propose a two-stage approach based on data analysis and focus on the gravity center concept. The neural network is first trained on reduced data represented by a set of centroids of the original data points, and then the learned weights are used to initialize a second training phase of the neural network over the full-blown data. The design of deep neural networks is extremely difficult, and the primary objective is to achieve high performance. In this study, we apply the Taguchi method to select good values for the factors required to build the proposed architecture.

Key words: *Deep Neural Networks, centroids, arabic language processing, big data, morphosyntactic tagging, machine translation, regression*

Received: January 24, 2019 **DOI:** 10.14311/NNW.2019.29.025
Revised and accepted: December 30, 2019

1. Introduction

At present, deep neural networks (DNNs) are considered the best method for learning data with considerable accuracy. DNNs are widely used in many artificial intelligence applications, including machine translation, robotics and autonomous cars and drones. In fields such as pattern recognition, convolution neural networks are becoming increasingly accurate in identifying objects. Recurrent neural networks (RNNs) have been successfully used in many natural language processing applications, particularly in language models, and recently, in machine translation.

Training this type of neural networks can be regarded as an optimization problem. It involves searching for the local minimum that is close to the global minimum. When the local minimum is close to the global minimum, the performance is acceptable and the training process is successful. DNN performance depends on two concepts: convergence speed and generalization capability. Convergence

[*]Abdelhalim Saadi – Corresponding author; Faculty of Technology, Department of Basic Studies in Technology, University of Ferhat Abbas Setif 1, Algeria, E-mail: h.saadi@yandex.com

[†]Hacene Belhadef; NTIC Faculty, Abdelhamid Mehri Constantine 2 university, Algeria, E-mail: hacene.belhadef@univ-constantine2.dz

speed refers to the number of iterations required to reach a predefined value for errors to stop the training process. The two major factors that can influence the performance of DNNs are weight initialization value and learning rate. The weight initialization value is critical for convergence speed because different numbers of iterations will be required to converge for the same local minimum depending on the weight initialization value. Generalization capability is the ability to handle unseen data. Many successful convergences with different local minima can be found. Here, the generalization will be different as we follow the local minimum. Therefore, performance will also be different.

In this study, we present an original approach based on data analysis [2, 17] to find the best weights for starting the training of large corpora. This approach consists of dividing the training corpus into small subsets and then finding the center of gravity [15], which is actually a synthetic value, for each subset. All the centroids are collected, and a small training set is established. The learned optimal weights will be used to train the full-blown data. The training process converges rapidly because the sum of the distances between the centroids and the other elements is optimal.

Building or designing DNN architecture is the same as designing an engineered system. Perfection is achieved when all the inputs are transformed accurately to create the right results. We used the Taguchi method, which is a structured approach for determining the best combination of factors to produce a product or service, as the parameter optimization method. The Taguchi method can considerably reduce the cost of time in a simulation process. It is based on a design of experiments (DOE) methodology for determining parameter levels. DOE or robust design is an experimental method for achieving process quality by integrating insensitivity to noise using statistical approaches.

The scope of our architecture covers the Arabic part-of-speech (POS) tagging. POS tagging is the process by which a specific tag is assigned to each word in a sentence to indicate the function of that word in its specific context. Part-of-speech tagging demonstrates the syntactic category of words such as noun, verb, pronoun, adverb, adjective, or other tags, to resolve lexical ambiguity [5]. The process of POS tagging is a critical task in text parsing. It is considered one of the fundamental tools in natural language processing and is frequently part of a higher-level application, such as machine translation, speech recognition, and information retrieval. Our work is divided into two parts. The first part involves Arabic POS tagging and the neural network architecture. The second part involves describing our approach to find the best initial weights for our proposed architecture.

Natural language processing involves analyzing huge volumes of data. Therefore, parallel computing, such as multithreading and the use of the processing capacity of graphics cards, can help achieve the aforementioned requirements. In our architecture, we have implemented multithreading on a multicore multithreaded CPU and multithreading using the capacity of a GPU via Nvidia CUDA technology.

2. Related works

The weight initialization problem and its proposed solutions have existed for as long as neural networks have. To design a neural network, we must first answer several questions. For example, how many hidden layers do we need to solve our problem? How many hidden neurons are found in each hidden layer? What are the best values for the learning rate and momentum? What is the best value for weight initialization? To date, machine learning experts have been using randomly initialized weights as the starting point of the training process. They were not aware that the initial values of these weights are critical for finding the global minimum [10] of a deep neural network cost function.

2.1 Zero initialization

The simplest method for weight initialization is zero initialization. This method does not perform symmetry breaking. Initializing weights by zeros makes the complexity of the deep neural network similar to that of a simple neuron because all the outputs of all the layers perform the same calculation.

2.2 Random initialization

To date, the only means to initialize weights for deep neural networks is through random initialization. When random values are used instead of zeros, symmetry breaking is performed. Performance varies in this case because neurons in different layers conduct various computations. Random initialization has been used for a long time. In [23], a method based on the minimum bound of weight initialization was proposed. In this method, every weight should be lower than the square root of the learning rate divided by the number of neurons in the previous layer. In [8], another method for weight initialization based on the threshold of a unit was presented. In this method, the threshold highly depends on the activation function, and the random values selected in this case should all be positive and higher than the threshold. In [13], another weight initialization method based on the importance of every input was developed. An input with high importance should be initialized with a large weight. This method requires statistical techniques to select inputs with high and low importance to assign weights according to their importance. Considering the characteristics of the information transformation system of a unit, Shimodaira [9] proposed an approach based on equations and certain parameters. Shimodaira used the sigmoid function, while we used the *tanh* function. Therefore, we modified the first parameter according to the activation function that we used in our architecture. In [16], a weight initialization method called statistically controlled activation weight initialization (SCAWI) was developed. The authors presented two equations for calculating the initial weights for the input and hidden layers. In SCAWI, the authors adopted the concept of paralyzed neuron percentage, which is defined by the number of times that a neuron is saturated and the magnitude of neurons (a least one) in the output layer.

In recent years, several methods have been presented, and the best known appeared in [22]. In this work, Glorot proposed a formula for estimating standard deviation based on the number of input and output channels of the layers under

the assumption that no nonlinearity occurs between layers. The initial values of the weights for a hidden layer i should be uniformly sampled from a symmetric interval that depends on an activation function. In [3], an initialization approach based on Gaussian noise with a mean equal to zero and a standard deviation set to 0.01 was developed. A bias equal to one was added to some layers. In [8], a new method for weight initialization was proposed in 2015 by He et al. This method is similar to Glorot initialization, with a factor multiplied by two. This method is highly efficient for rapidly attaining a global minimum of the cost function.

3. Architecture of the POS tagging neural network

Neural networks have many possible architectures, including multilayer perceptron, convolutional neural networks [19], restricted Boltzmann machine [1], and RNNs [4]. Our architecture is described in Fig. 1. It is inspired by the work of [18]. In this section, we discuss the different architecture layers: input, hidden, and output layers. In our work, we used the word2vec, which is a neural-based model and application introduced by [21] to represent words into real-valued vectors.

The input layer represents a window of words, which denotes the word on which training is performed surrounded by a number of words. Each word is represented by the continuous bag of words (CBOW) model concatenated by word features. For the surrounding words, we add the probability of being included into each grammatical class. Subsequently, we discuss the mechanisms of propagation and backpropagation.

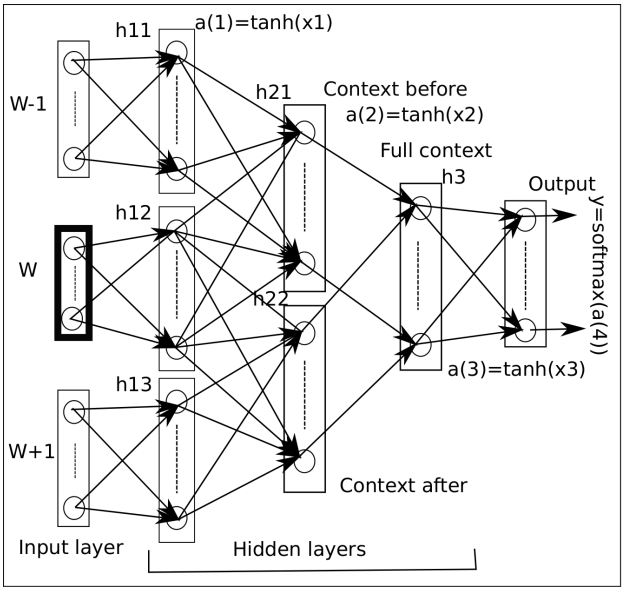

Fig. 1 *Neural network architecture.*

3.1 Preprocessing of datasets

The preprocessing of datasets is the first step in preparing words to obtain a manageable representation that we can use in our neural network. This step is divided into three parts: the extraction of word features, the calculation of probabilities, and the CBOW representation for each word. The Morphological Analysis and Disambiguation for Arabic (MADA)[1] is the main toolkit used for data preprocessing. We used MADA to extract all the information and analysis about a word, including stem, POS, gender, number, affixes, and gloss. The features of MADA are Aspect, Case, Gender, Mood, Number, Person, State, Voice, Stem, POS, and affixes. The word structure in MADA follows this pattern:

[PRC3 [PRC2 [PRC1 [PRC0 – BASEWORD – ENC0]]]].

PRC3 is a question proclitic or QUES, PRC2 is a conjunction proclitic or CONJ, PRC1 is a preposition proclitic or PREP, PRC0 is an article proclitic or ART, an ENC0 are pronominals enclitics or PRON. We create the first look-up table LW1 by extracting the aforementioned features. Then, we use word2vec to obtain word representation in vector space, which is the second look-up table LW2. The final step in preparing data to feed the neural network is computing probabilities. To find these probabilities, we train a simple feedforward neural network. The input layer is for word representation, whereas the output layer is for the probability that a word will belong to each grammatical class.

The probabilities are listed in LW3, which is the third look-up table. The vector input of the word w is the concatenation of all the values extracted from the three look-up tables.

3.2 Propagation

To learn the network, we use the backpropagation method with the stochastic gradient descent algorithm. Accordingly, the first step is the propagation and calculation of the error. Then backpropagation is performed to update weights and minimize the error E.

For each hidden layer, we apply a nonlinear function, namely, the tanh function, to the result. We use the following classic formula described in [12] to limit the saturation phenomenon:

$$y = 1,716 \times \tanh\left(\frac{2}{3}x\right), \tag{1}$$

because the tanh function tends rapidly toward 1 or -1, and its derivative is close to 0. Therefore, the backpropagation of the error is stopped. This phenomenon is known as saturation.

In the output layer, we apply the softmax function, which is a log-linear classification model, to obtain the probabilities in the output.

$$y_k = p(T \mid W_C) = \frac{\exp(a_k^{(4)})}{\sum_{j=1}^{n_2} \exp(a_j^{(4)})}, \tag{2}$$

[1]MADA+TOKAN version 3.2 using Aramorph version 1.2.1

where T is the set of tags, W_C is the input words, and $a_j^{(4)}$ is the output value in position j in the output layer.

3.3 Backpropagation

The objective of training is to maximize $p(T \mid W_C)$, the conditional probability of observing the actual output, the set of tags for the word w_I (denote its index in the output layer as y_k^*), surrounded by the words of the context (all the explanation based on Fig. 1). To maximize $p(T \mid W_C)$ we use the following equation:

$$\max(p(T \mid W_C)) = \max(y_k^*) = \max \log y_k^*. \tag{3}$$

We define our loss function E as

$$E = -\log p(T \mid W_C). \tag{4}$$

Our objective is to minimize E. Notably, this loss function can be considered a special case of cross-entropy measurement between two probabilistic distributions.

The input vector of the word in the context is $x = [x_1, x_2, \ldots x_{n_0}]$. We have C input vectors, where C is the number of words in the context. We have the output vector of the desired output, i.e., $y^{\text{des}} = \left[y_1^{\text{des}}, y_2^{\text{des}}, \ldots, y_{n_2}^{\text{des}}\right]$. The error is given by the following equation:

$$e_k = y_k^{\text{des}} - y_k. \tag{5}$$

The loss function E for an example is:

$$E_{\text{example}} = \frac{1}{2} \sum_{k=1}^{n_2} e_k^2. \tag{6}$$

For all examples, we have:

$$E = \sum_{l=1}^{N} E_l, \tag{7}$$

where N is the number of all examples.

4. Centroids for fighting dimensionality

Motivated by the fact that a deep neural network process can simultaneously run several logistic regressions, we use regression techniques to select the best initial weights to train our neural network for the global training set. The concept here is to divide the training set into subsets $E = \{E_1, E_2 \ldots E_m\}$. The subsets may have different sizes. Every subset E_i can be represented by a cloud of n points $\{(x_i, y_i), i = 1, 2 \ldots n\}$. For each subset found, we identify its *gravity center*, which is the mean of $\{x_i\}$, and the mean of $\{y_i\}$, as follows:

$$(\mu(x), \mu(y)) = \left(\frac{1}{n} \sum_{i=1}^{n} x_i, \frac{1}{n} \sum_{i=1}^{n} y_i \right). \tag{8}$$

Minimizing the overall error is the result of minimizing the error of all the subsets:

$$E = \sum_{i=1}^{m} e_i. \tag{9}$$

Our idea consists of finding the best starting point for minimizing the local error for each subset, and then finding a model to join these points to reduce the global error of the training set. At the level of each subset, we define the total dispersion (TD) as the sum of the dispersion due to the regression (RD) and the dispersion around the regression line or residual dispersion (AD) as shown in Fig. 2.

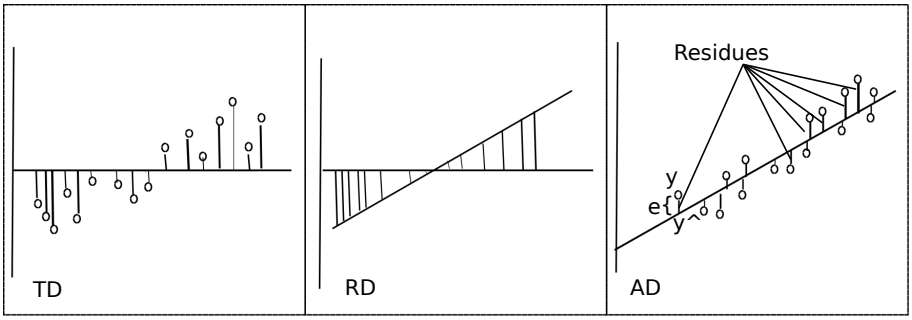

Fig. 2 *Total dispersion, dispersion due to the regression, and residual dispersion.*

Fig. 2 illustrates the formula $TD = AD + RD$. The horizontal line passes through the gravity center of the cloud of points. The first figure represents the TD, the second shows RD (null if the line slope of the least-squares is null and important if this slope is strong), and the third presents AD.

We have

$$TD = AD + RD. \tag{10}$$

Therefore, we can write

$$\sum_{i=1}^{n} \left(y_i^{\text{des}} - \mu(y)\right)^2 = \sum_{i=1}^{n} \left(y_i^{\text{des}} - y_i\right)^2 + \sum_{i=1}^{n} \left(y_i - \mu(y)\right)^2. \tag{11}$$

From Eq. 11, we can derive

$$\sum_{i=1}^{n} e_i = \sum_{i=1}^{n} \left(y_i^{\text{des}} - y_i\right)^2 = \sum_{i=1}^{n} \left(y_i^{\text{des}} - \mu(y)\right)^2 - \sum_{i=1}^{n} \left(y_i - \mu(y)\right)^2. \tag{12}$$

From Eq. 12, we have $\sum_{i=1}^{n} \left(y_i^{\text{des}} - \mu(y)\right)^2$, which represents the sum of distances between the gravity center and the other points. This distance is optimal and is the smallest because $\mu(y)$ is the gravity center of these points, in fact it is the barycenter because all the words have the same importance. If we take three points A, B, C in the Euclidean plane $(O, \vec{i}, \vec{j})$, we can say that the point G is the barycenter of the points A, B and C if we find $a, b, c \in R$ and $a + b + c \neq 0$. We have:

$$a\vec{OA} + b\vec{OB} + c\vec{OC} = (a+b+c)\vec{OG}. \tag{13}$$

Therefore,

$$x_G = \frac{ax_A + bx_B + cx_C}{a+b+c}, \tag{14}$$

$$y_G = \frac{ay_A + by_B + cy_C}{a+b+c}. \tag{15}$$

If we take $a = b = c = 1$ because all the words have the same importance, we can find $x_G = \mu(x_i)$ and $y_G = \mu(y_i)$, which is in fact the gravity center.

We also have $\sum_{i=1}^{n} (y_i - \mu(y))^2$, which is always a positive number. Therefore, the sum of errors is also optimal (Eq. 12).

This idea helps us start training from the center of gravity of each subset. Subsequently, we need to connect all these centroids by using a model. To achieve this, we create a training set. Fig. 3 illustrates how the model passes through the centroids. The size of this set is the number of subsets because we find the gravity center $(\mu(x), \mu(y))$ for each subset. The training space here is defined as $\{(\mu(X_i), \mu(Y_i), i = 1 \dots m\}$.

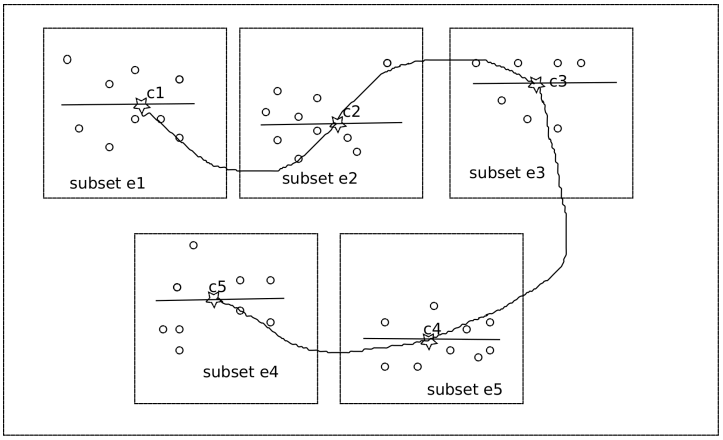

Fig. 3 *Training the neural network with the set of centroids.*

Training the neural network with the set of centroids is too fast because the size of the set is too small, depending on the size of the subsets.

After training the neural network with the set of gravity centers, the optimal weights found will be used as an initial set of weights to train the global training set. This process guarantees two situations. First, the error in each center of gravity is $\simeq 0$ because we train the neural network with the set of centroids. Second, starting the training from the gravity center of each subset ensures that the minimum error for that subset is obtained.

4.1 Size of subsets

The objective of the centroid approach is optimizing the convergence of the training process. A suitable choice for the size of the subsets is critical. If we regard the size of the training set as N and the size of the subsets as M, then we will have the following equation in case all the subsets have the same size:

$$| \text{ Centroids } | = \left[\frac{N}{M} \right]. \tag{16}$$

From Eq. 12, we derive

$$\sum_{i=1}^{n} e_i = \sum_{i=1}^{n} \left(y_i^{\text{des}} - \mu(y) \right)^2 - \sum_{i=1}^{n} \left(y_i - \mu(y) \right)^2. \tag{17}$$

As shown in Eqs. 16 and 17, a small value for M decreases the error for every subset. Therefore, the sum of errors also decreases, but the size of the centroid set will be increased and the time for training the centroid set will be significant. In such case, we exert considerable effort on the centroid set and less effort on the full-blown data. By contrast, a large value for M increases the error for every subset because it becomes larger, and thus, the error also increases, and the centroid set will be small. In such case, we exert considerable effort on full-blown data and less effort on the centroid set. Therefore, the suitable subset size is small, because we prefer exerting less effort on full-blown data due to their large size. However, this value should not be too small to prevent the size of the centroid set from being too important.

In our experiments, we set the size of the subsets to be 5 % the size of full-blown data. Fig. 4 illustrates the convergence speed using three sizes of subsets. As shown in the figure, good results are obtained for very small subsets (2.5 % of the size of the training corpus). However, if we consider the time allotted to train the centroid set, then the appropriate subset size will be 5 % the size of the training corpus.

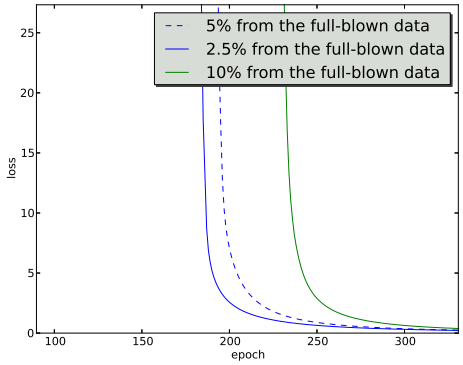

Fig. 4 *Impact of the size of subsets on convergence speed.*

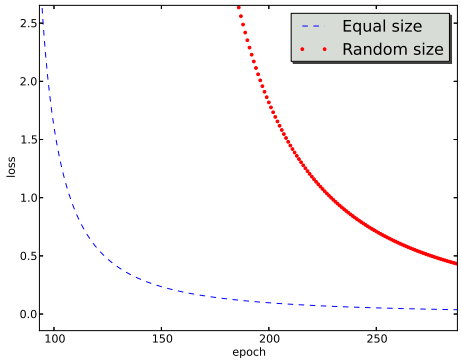

Fig. 5 *Equal vs different sizes of subsets.*

In the first case, we assume that all the subsets have the same size. In another case, we compare the results obtained using equal and different sizes with the same

number of subsets, which denotes the same size as the centroid set. Fig. 5 illustrates the advantage of using equal-sized subsets, given that subsets may range from large to extremely small. In case of large subsets, the error will also be large, whereas small subsets exert no influence on reducing loss in the training corpus.

Algorithm 1 Centroid algorithm.

Require: - A set of examples $\{(x_i, y_i^{\text{des}}), i = 1, 2 \dots N\}$
- Neural network: a feedforward neural network(our architecture)
- M: size of subsets (in case they are all equal)

Ensure: The model with the best initial weights: λ
Following our architecture, the model here is
$\lambda = \{MW_{h11}, MW_{h12}, MW_{h13}, MW_{h21}, MW_{h22}, MW_{h23}, MW_O\}$, where MW_i is the matrix of weights of the hidden layer i.
for each subset$_j \in \left[1, \left[\frac{N}{M}\right]\right]$ **do**
Centroid$_j \leftarrow (\mu(x_k), \mu(y_k))$, $k = 1, 2 \dots M$
end for
Collect all Centroid$_j$ in the same set.
Initialize all the weights of the architecture by zero to avoid using random methods .
Train the architecture with the set of centroids.
Save the best weights found in the model λ_{optimal}.
Load the global training set.
Load the optimal model λ_{optimal}.
Train(network).

5. Theory and methodology

To verify the effectiveness of our method, we compared it with best-known methods in the field of weight initialization of deep neural networks. First, we present the mathematical background of these methods. Then, we provide a comparison among all the methods.

5.1 Glorot initialization

In this method, the initial values of the weights of the hidden layer i should be uniformly sampled from a symmetric interval that depends on an activation function. For the tanh function, the results obtained in [22] shows that the interval should be

$$w_{ij} \in \text{Uniform}\left[-\sqrt{\frac{6}{f_{an_{\text{in}}} + f_{an_{\text{out}}}}}, \sqrt{\frac{6}{f_{an_{\text{in}}} + f_{an_{\text{out}}}}}\right], \tag{18}$$

where $f_{an_{\text{in}}}$ is the number of units in the previous layer, and $f_{an_{\text{out}}}$ is the number of units in the next layer.

5.2 He initialization

This method is similar to Glorot initialization with the factor multiplied by two [8]. In this method, the weights are initialized by the size of the previous layer

$$w_{ij} \in \text{Random}\left[2 \times \sqrt{\frac{6}{f_{an_{\text{in}}} + f_{an_{\text{in}}}}}\right]. \tag{19}$$

5.3 Krizhevsky initialization

In this method, the weights are initialized with random Gaussian values with standard deviation $\delta = 0.01$ for all layers [3].

$$w_{ij} \in \delta \times \text{Random}(f_{an_{\text{in}}}, f_{an_{\text{out}}}) + \mu, \tag{20}$$

where μ is the mean of the weights.

5.4 Shimodaira initialization

In our study, we tested all the random methods cited in the related works section. From the previous methods we found that the Shimodaira initialization [9] achieved good results. This method, which is based on equations, represents the characteristics of the information transformation mechanism of a neuron. This method can be summarized through the following steps.

1. Calculate b using the formula $b = |f^{-1}(-1+\varepsilon) - f^{-1}(1-\varepsilon)|$, where f is the tanh function. We modified the original formula for calculating b because Shimodaira used the sigmoid function in his works. ε has an extremely small value. We used $\varepsilon = 0.1$ in our experiments.
2. Calculate $w' = \frac{b}{\sqrt{2} \times k \times n}$, where $k = 8$ and n is the number of units in the lower layer.
3. Calculate $w = w' \times \sqrt{a_i + 1}$, where a_i is a random value between -0.6 and 0.6.

6. Experimental results

6.1 Experiments by using the holdout method for the validation

The objective of this research is to apply a new initialization method and to determine its usefulness. We applied our approach to five categories: culture, economy, localnews, international news, and sports for 100 000 examples taken from the KALIMAT corpus [7]. We compared our results with the best-known methods in this field. All the methods described in the theory and methodology section are random. That is, performance will generally vary in different trials. Therefore, we performed eight trials for every initialization method in our experiments, and the mean was regarded as the performance value. Our experiments were based

on the root mean square error (RMSE), and we set the learning rate at 0.01 and the momentum at 0.5. We evaluated two effects: convergence speed and accuracy. Convergence speed is measured using the number of epochs required to reach a threshold of error to stop the training. In our case, the threshold is 0.05. The accuracy is based on the generalization performance, and it is calculated using the test set, we used the holdout method because we have made our experiments on large datasets, thus the computational time will be very large (many experiments). In [20] a comparative study has been made between k-Fold cross-validation over hold-out validation on colossal datasets for quality classification. The results found show that for large datasets both methods give very close results. For instance, if we take 10 folds in the cross-validation method, it is like to take 90 % from the size of the dataset as the training set and 10 % for the testing set in the holdout method.

Fig. 6 presents the results obtained in the context of three words. In this case, the neural network did not converge in all the methods. In terms of convergence speed, centroid and Glorot methods are the best. Fig. 7 shows the results in the context of five words. In this case, all the methods converged. However, the centroid method is considerably better than the other methods in terms of convergence speed. In the context of seven and nine words, the centroid method still achieved the best performance, followed by the Glorot and He methods. Fig. 10 illustrates the convergence speed of the five methods in the context of 11 words. In this case, the neural network did not converge in all the methods. For the convergence speed, the centroid method is considerably better than the other methods. The Shimodaira and Krizhevsky methods exhibited considerably worse performance than the centroid, He, and Glorot initializations. Particularly Krizhevsky initialization, we even added biases to the hidden layers and set extremely small values for μ. Nevertheless, we always obtain the same results.

The second factor for performance measurement is generalization. In Tab. I, the columns present three headers in every context: "Acc%" for the mean value of the accuracy measurement, "Epo" is the mean value of the convergence speed or the number of epochs required to reach the threshold error, and "Cv" denotes if the neural networks have converged. The centroid method is the best in one category, whereas He initialization is the best in one case and Glorot is the best in one. In addition, in the cases where the centroid method is not the best, it obtains an accuracy that is extremely close to those of He and Glorot's initializations.

If we simultaneously consider convergence speed and accuracy for performance measurement, then centroid initialization is clearly the best because even in the cases where He and Glorot initializations achieve the best accuracy, the accuracy of centroid initialization is extremely close to theirs. Moreover, the convergence speed of the centroid initialization is considerably better.

To make a fair comparison of the convergence speed between the approaches, to the time of the training of the centroid approach we added the time of the training of the centroids set in each context. In Tab. II, we find the time of the training of the centroids set, and in Tab. III, we find the time of training of the different approaches. From Tab. III, we can see that the time of training of the centroids set has no effect and can be neglected.

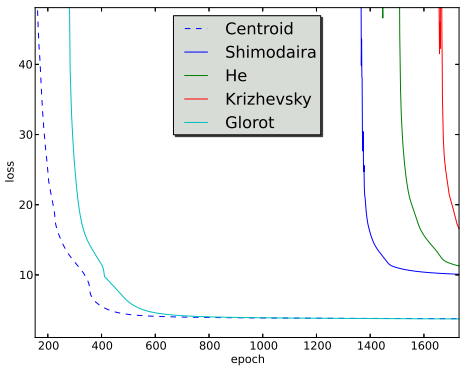

Fig. 6 *Training loss in the context of three words.*

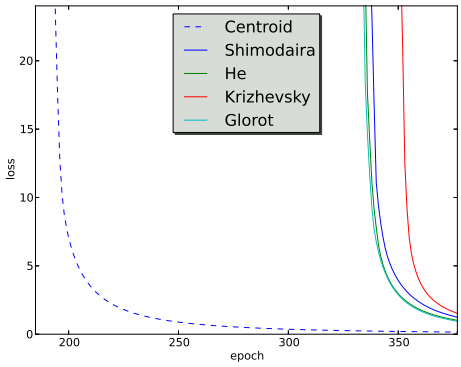

Fig. 7 *Training loss in the context of five words.*

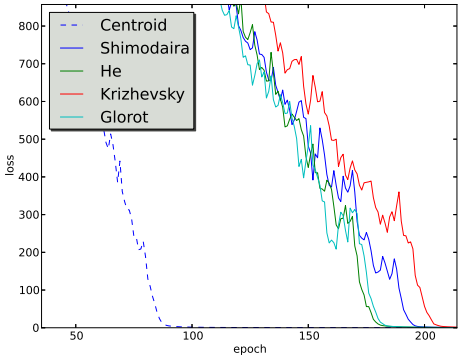

Fig. 8 *Training loss in the context of seven words.*

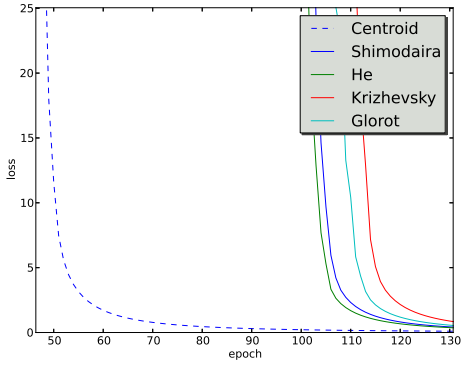

Fig. 9 *Training loss in the context of nine words.*

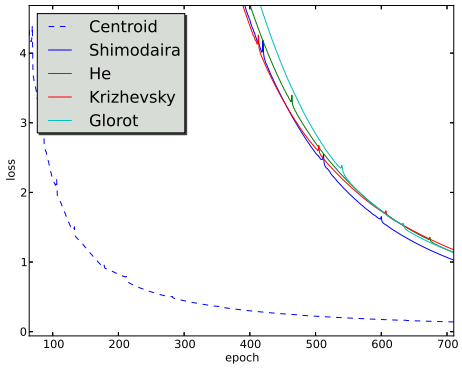

Fig. 10 *Training loss in the context of 11 words.*

Context		3			5			7	
Performance	Acc%	Epo	Cv	Acc%	Epo	Cv	Acc%	Epo	Cv
Centroid	–	–	No	90.10	**950**	Yes	91.14	**825**	Yes
Glorot	–	–	No	90.21	1147	Yes	**93.78**	1012	Yes
He	–	–	No	**91.66**	1088	Yes	91.03	1051	Yes
Shimodaira	–	–	No	85.28	1389	Yes	86.94	1140	Yes
Krizhevsky	–	–	No	80.37	1412	Yes	82.79	1381	Yes

Context		9			11	
Performance	Acc%	Epo	Cv	Acc%	Epo	Cv
Centroid	**94.92**	**742**	Yes	–	–	No
Glorot	92.08	935	Yes	–	–	No
He	93.89	905	Yes	–	–	No
Shimodaira	88.13	1096	Yes	–	–	No
Krizhevsky	84.01	1238	Yes	–	–	No

Tab. I *Performance of the centroid method against the other methods.*

	Centroid set 5 % from the Full-blown data			Full-blown data 100 000 examples
Context	1 Epoch / [s]	Epochs	CV	1 Epoch / [s]
3	2.2173	–	No	711.31
5	4.7721	750	Yes	1176.23
7	8.3911	537	Yes	1590.1
9	11.8415	440	Yes	1825.24
11	18.7352	–	No	2508.74

Tab. II *Duration and number of epochs for training the centroids set.*

Context	3		5		7	
Performance	T. Training	Cv	T. Training	Cv	T. Training	Cv
Centroid	–	No	**13 days 00h 49m**	Yes	**15 days 07h 18m**	Yes
Glorot	–	No	15 days 16h 25m	Yes	18 days 16h 59m	Yes
He	–	No	14 days 21h 02m	Yes	19 days 10h 19m	Yes
Shimodaira	–	No	18 days 23h 49m	Yes	21 days 01h 51m	Yes
Krizhevsky	–	No	19 days 07h 27m	Yes	25 days 06h 34m	Yes

Context	9		11	
Performance	T. Training	Cv	T. Training	Cv
Centroid	**15 days 19h 18m**	Yes	–	No
Glorot	19 days 20h 09m	Yes	–	No
He	19 days 04h 57m	Yes	–	No
Shimodaira	23 days 06h 14m	Yes	–	No
Krizhevsky	26 days 06h 34m	Yes	–	No

Tab. III *The training time of the different approaches against the centroid initialization method including the training time of the centroids set.*

6.2 Experiments by using the cross-validation method

In this kind of validation, we create a K-fold partition of the dataset. For each of the K experiments, we use K−1 folds for training and a different fold for testing. The advantage of K-Fold cross-validation is that all the examples in the dataset are eventually used for both training and testing. The true error is estimated as the average error test examples as follows:

$$E = \frac{1}{k}\sum_{i=1}^{k} E_i \tag{21}$$

We applied our approach to 10 000 examples taken from [6] using 4-fold in each experiment.

Through the results obtained in Tabs. IV, V and VI, we can observe that as in the experiments on big data the model does not converge in the case of three and 11 words. In terms of converge speed the centroid method still the best, and it has the best accuracy in the context of seven and nine words. We also can see that the Glorot method yields very close results to the centroid method, it has the best accuracy in the context of five words and the second-best convergence speed, we can see that in Fig. 11 and 12. The He method comes as the third-best method we can see that in Fig. 13 and Tab. VI.

	Centroid		Shimodaira		HE		Krizhevsky		Glorot	
	Acc%	Epo	Acc%	Epo	Acc%	Epo	Acc%	Epo	Acc%	Epo
Fold A	90.01	541	88.14	588	92.07	592	86.01	599	92.15	567
Fold B	89.74	539	90.13	584	92.11	598	84.23	599	92.20	583
Fold C	91.51	537	90.01	582	91.44	542	90.12	554	91.88	553
Fold D	90.35	556	87.13	567	92.13	543	84.17	588	91.96	546
Average	90.40	**543**	88.85	580	91.94	568	86.13	585	**92.05**	562

Tab. IV *Performance of the centroid method against the other methods in context of five words.*

	Centroid		Shimodaira		HE		Krizhevsky		Glorot	
	Acc%	Epo	Acc%	Epo	Acc%	Epo	Acc%	Epo	Acc%	Epo
Fold A	92.87	496	90.62	510	92.94	500	90.03	512	92.88	507
Fold B	93.20	485	90.84	506	91.13	489	89.29	510	92.93	490
Fold C	92.91	513	90.01	492	91.83	505	87.10	499	92.84	494
Fold D	92.96	486	92.65	494	91.87	505	90.78	518	92.73	493
Average	**92.92**	**495**	91.03	500	91.94	500	89.30	510	92.85	496

Tab. V *Performance of the centroid method against the other methods in context of seven words.*

	Centroid		Shimodaira		HE		Krizhevsky		Glorot	
	Acc%	Epo	Acc%	Epo	Acc%	Epo	Acc%	Epo	Acc%	Epo
Fold A	93.97	305	93.38	328	93.78	336	93.82	350	92.66	567
Fold B	93.85	308	93.88	338	93.84	341	93.93	361	93.98	583
Fold C	93.77	301	93.49	347	93.47	338	93.60	344	92.47	553
Fold D	93.50	307	93.67	341	93.50	341	93.31	350	93.39	546
Average	**93.77**	**305**	93.61	339	93.65	339	93.67	351	93.13	343

Tab. VI *Performance of the centroid method against the other methods in context of nine words.*

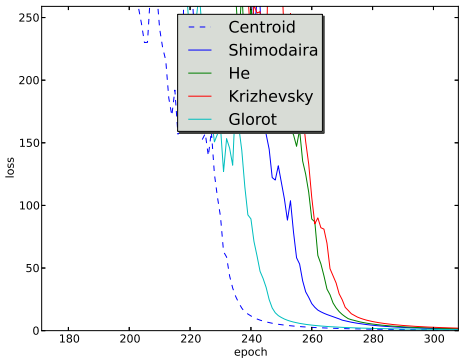

Fig. 11 *Training loss in the context of five words.*

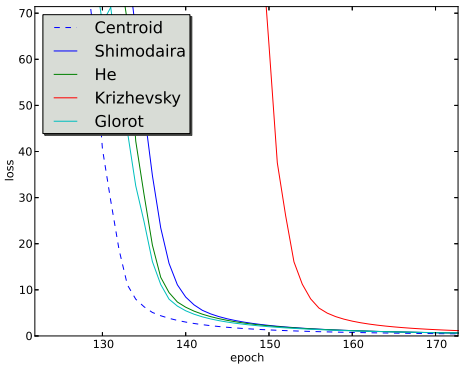

Fig. 12 *Training loss in the context of seven words.*

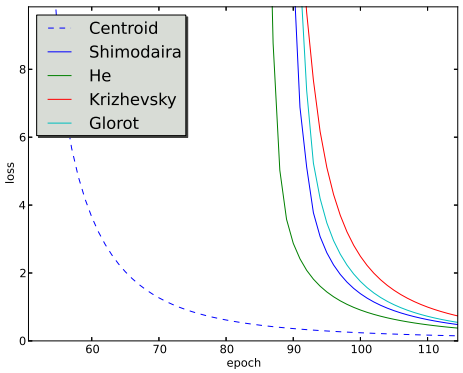

Fig. 13 *Training loss in the context of nine words.*

7. Taguchi method

The Taguchi method is a structured approach for determining the best combination of inputs to produce a product or service. It was proposed in the 1950s by Dr. Genichi Taguchi in Japan [14] based on DOE methodology for determining parameter levels. The objective of this method is to use a few experimental data for systematic analysis. Instead of using full factorial experiments, which are time-consuming, the Taguchi method uses an orthogonal array. In such case, each experiment should no longer be implemented and accuracy can be reached within a short period. In our study, we selected four factors: the context, which is the number of words in the input layer; learning rate; weight initialization method; and momentum. We selected four levels without considering the interaction among the factors when choosing the four-level standard. In our study, we selected $L_{16}(4^4)$. Our experiment is based on RMSE. Therefore, signal-to-noise smaller is better were chosen. The objective is to obtain a smaller error and higher accuracy.

7.1 Levels of factor design

We selected the following four factors as the primary configurations of our neural network

1. *Context*: We selected four values: 5, 7, 9, and 11. We used a context with three words, but the model did not converge.

2. *Weight initialization*: Four methods were selected: centroid, He, Glorot, and Shimodaira.

3. *Momentum*: We used four values: 0.25, 0.50, 0.75, and 1.00.

4. *Learning rate*: One of the most important hyperparameters for tuning neural networks is the learning rate. A good learning rate can transform a model that does not learn anything into a model that provides state-of-the-art results. First, we need to determine the effect of the learning rate on optimization in deep learning. If the learning rate is too high, then the parameters will go back and forth between points, thereby easily overshooting the minima and resulting in a large loss. If the learning rate is too low, then the training process will be extremely slow. Low learning rates not only make the training process long, but they can also even degrade the performance of the model. Current conventional knowledge indicates that high learning rates increase generalization ability. High learning rates are typically good at finding general areas with good solutions. However, low learning rates are better at finding the best solution in that area.

   We used two fixed values, namely, 0.01 and 0.001, and two adaptive learning rate methods. The first method is based on an exponential decay (MED), whereas the second method is based on a linear decay (MLD). Motivated by a concept from [11], we modified the method by adding a decay for maxLr and minLr after each cycle.

- MED: In this method, we fixed ranges for the maximum minimum learning rates using the following formula to find the learning rate in every epoch:
  initmaxLr = (0.1); initminLr = (0.06); endmaxLr = (0.01);
  endminLr = (0.001);
  $Ti = 1000$ defines the number of epochs in a cycle
  $T\text{current} = \text{modulo}(i, Ti)$ represents the number of epochs since the last restart.
  $l = i/(\text{epochs})$; i is the i-th epoch, epochs denotes the maximum number of iterations.
  $\text{maxLr} = \text{endmaxLr} + (\text{initmaxLr} - \text{endmaxLr}) \times \exp^{-7 \times l}$
  $\text{minLr} = \text{endminLr} + (\text{initminLr} - \text{endminLr}) \times \exp^{-7 \times l}$
  $\text{learningRate} = \text{minLr} + 1/\text{double}(2) \times (\text{maxLr} - \text{minLr}) \times (1 - \cos((1 - 2 \times T\text{current}/(Ti)) \times \pi))$

- MLD: In this method, we changed the formula for calculating maxLr and minLr as follows:
  $\text{tgalpha} = (\text{initmaxLr} - \text{endmaxLr})/\text{epochs}$
  $\text{tgbeta} = (\text{initminLr} - \text{endminLr})/\text{epochs}$
  $\text{maxLr} = \text{tgalpha} \times (\text{epoch} - i) + \text{endmaxLr}$
  $\text{minLr} = \text{tgbeta} \times (\text{epoch} - i) + \text{endminLr}$
  $\text{learningRate} = \text{minLr} + 1/\text{double}(2) \times (\text{maxLr} - \text{minLr}) \times (1 + \cos(T\text{current}/\text{double}(Ti) \times \pi))$

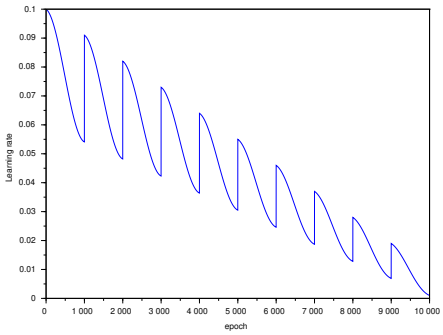

Fig. 14 *Learning rate with linear decay.*

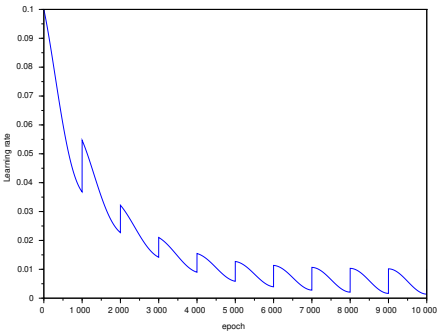

Fig. 15 *Learning rate with exponential decay.*

Tab. VIII illustrates the SNRs among factors using the (smaller is better method). We calculated SNR for each factor in each level using Minitab software[2]. We can also see the delta value, which is the difference between the highest and lowest ratios for the same factor. For all the factors, we ranked the delta value from the highest to the lowest. In our case, the context factor is ranked as 1 and momentum

[2]Minitab 18: Statistical Software Free Trial.
http://www.minitab.com

Level/Factor	Context	Weight Initialization	Learning Rate	Momentum
1	5	Centroid	MED	0.25
2	7	Glorot	MLD	0.50
3	9	He	0.01	0.75
4	11	Shimodaira	0.001	1.00

Tab. VII *Factors and levels for a Taguchi plan.*

is ranked as 4. The factor ranked as one exerts the most considerable effect on building neural networks.

Level	Context	Weight Initialization	Learning Rate	Momentum
1	13.199	9.069	9.213	12.782
2	9.166	11.182	15.942	12.691
3	6.113	12.560	13.570	14.052
4	20.641	16.308	10.393	9.594
Delta	14.529	7.239	6.729	4.458
Rank	1	2	3	4

Tab. VIII *Response table for signal to noise ratios, smaller is better.*

Fig. 16 illustrates the optimized configuration for a neural network. Our model will assume the following values for the factors: context (nine words), learning rate (MED adaptive LR with exponential decay), weight initialization (centroid), and momentum (1.00).

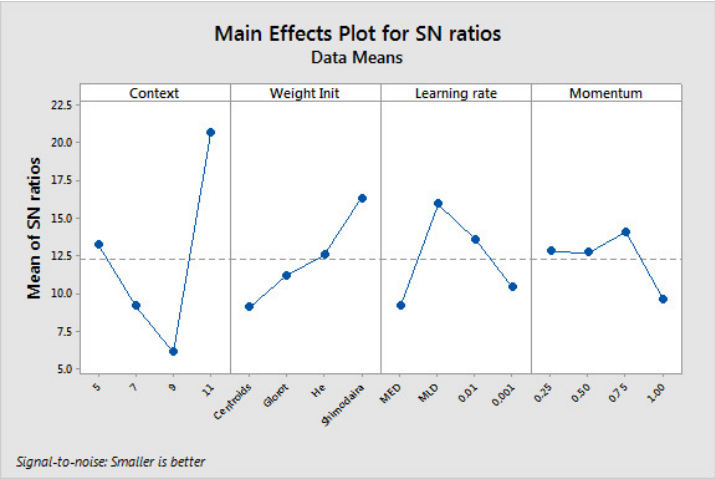

Fig. 16 *Optimal parameters for the factors.*

7.2 Optimal configuration choice

The optimal configuration for our model is context (nine words), Learning rate (MED adaptive LR with exponential decay), weight initialization (centroid), momentum (1.00). To predict the correspondent theoretical rate of defect T (Theo), we simply have to add to the average defect rate (11.35). The different effects as follows:
Effect(Context = 9 words) $= 6.113 - 11.35 = -5.24$
Effect(LR = MED) $= 9.069 - 11.35 = -2.28$
Effect(weight initialization = centroid) $= 9.213 - 11.35 = -2.14$
Effect(Momentum = 1.00) $= 9.594 - 11.35 = -1.76$
Ttheo $= 11.35 - 5.24 - 2.28 - 2.14 - 1.76 = -0.07$
This negative result may be unexpected, but it is a theoretical calculation and a 0 rate will be highly acceptable.

To validate our model, we performed eight trials using the optimal configuration. The results confirm the theoretical outcome, thereby indicating that the experimental design is successful.

8. Parallel computing

8.1 Parallel computing using CPU

This section has no relation to our work presented in sections 4, 5 and 6, we have added this section to show that our architecture supports parallel programming, hence the improvement in computational time. Deep neural networks involve many learning layers arranged as a network and working together to create one large model. This model typically has many layers of learning, and each layer learns new patterns from the data from the previous layer. Deep neural networks involve a long-running and time-consuming process. To accelerate the process, we split it over multiple threads on multi-core processors. Thus, our neural network architecture supports the parallel aggregation pattern and the following conditions are satisfied.

1. We can split the architecture into multiple independent parts. For each hidden layer, we can compute h^l independently. The first hidden layer behaves similar to filters in convolution neural networks. Each component will produce a partial result that will be used in the second hidden layer.

2. In our architecture, the result of each part is a vector h^l that is not related to the results of the other parts.

3. The partial results from each worker thread will be aggregated when all the threads are finished.

We conducted our experiments on an Intel Core i7-2670QM CPU 2.20 GHz × 8 with 8 GB RAM and GeForce GT 525M/PCIe/SSE2 Nvidia graphic card. The number of logical cores in the machine is eight-core processors between physical and hyper-threading enabled. In general, the operating system performs a 1-to-1 mapping of the application threads to the cores. Therefore, we can simultaneously calculate the hidden layers, which accelerates the process.

8.2 Parallel computing using GPU

In neural networks, most calculations are multiplication operations between matrices. To use CUDA, we created a kernel that calculates the multiplication of a matrix by a vector. We first define the number of blocks required to perform a multiplication operation. In general, each block contains 512 threads. The number of threads is typically equal to the number of elements in a multiplication result. When a kernel is called, all the threads will be launched simultaneously, and the result will be obtained immediately.

After finding the best configuration using the Taguchi method. We determined that the best context is nine words. On the basis of our architecture, our model will have the following weights:

- $9 \times 80 \times 100 = 72,000$, where 80 is the size of a word vector, and 100 is the size of each unit in the first hidden layer.
- $10 \times 100 \times 100 = 100,000$, from the first hidden layer to the second hidden layer.
- $34 \times 100 = 3,400$, from the third hidden layer and the output layer.

We have a model with 175,400 weights that should be adjusted during the training process. We compared between CPU and GPU and we applied the two methods on 100,000 examples (900 000 words) from [7].

We found that the average time for an epoch using CPU is 1873.78 s, and the total number of epochs required to reach convergence is 742. In summary, training takes approximately 16 days 2 hours 12 minutes using CPU. The average time for an epoch using GPU is 1825.24 s. The training process in this case will take 15 days 16 h and 12 min.

In conclusion, a hardware accelerator, such as GPU, is important in deep neural networks models. In deep learning projects, using GPUs is highly required particularly when the size of the tensors is important. Consequently, we find that most deep learning frameworks, such as: Tensorflow, PyTorch, Keras, and Caffe, use GPUs in deep learning applications.

9. Conclusion

In this study, we proposed an original weight initialization approach for a deep neural network architecture. This architecture was based on the automatic extraction of the vector representations of words and deep learning theories. The architecture contains five layers: the input layer, three hidden layers, and the output layer. This architecture is extensible to increase the size of the input layer, thereby enabling it to work with a large window of words. The output layer is also extensible and can be used in other natural language processing tasks, such as, language models.

Weight initialization and learning rate are the most important hyperparameters for tuning neural networks. A good initialization weight can be the reason for obtaining state-of-the-art results. A poor choice can be the reason why a model fails to learn anything.

Unlike other methods, which focus on the architecture for weight initialization, our approach focus on data. It consists of dividing the training corpus into small subsets, finding the centroids of these subsets, and training the neural network with this centroid set. The learned weights will be the initial weights for the full-blown data. We compared our approach with recent methods in this field and found that it was considerably better.

Our architecture comprises a set of factors with different levels. Selecting the best configuration among these factors is highly difficult and costly in time. If we select 4 factors with 4 levels, then we need 256 experiments. Every experiment requires many trials, and each trial can take hours (or even days in some cases); thus, the time cost will be considerable. To address this problem, we used the Taguchi method, which is based on DOE. We can reduce the number of experiments by using an orthogonal array with only 16 experiments $L_{16}(4^4)$.

Training this type of architecture is a time-consuming process. To address this challenge, we used parallel computing to reduce training time. In our project, we compared between multithreading CPU and GPU. We determined that GPU hardware accelerators are highly required in deep learning applications.

References

[1] FISHER A., IGEL C. An introduction to restricted boltzmann machines. In: *Proceedings of the 17th Iberoamerican Congress on Pattern Recognition*, 7441, pp. 14–36, 2012.

[2] GELMAN A., GOODRICH B., GABRY J., VEHTARI A. R-squared for bayesian regression models. *The American Statistician*, 0(0), pp. 1–7, 2019. doi: 10.1080/00031305.2018.1549100.

[3] KRIZHEVSKY A., SUTSKEVER I., HINTON G.E. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-net\protect\discretionary{\char\hyphenchar\font}{}{}works.pdf.

[4] MANNING C., KLEIN D., SINGER Y. Feature-rich part-of-speech tagging with a cyclic dependency network. In: *HLT-NAACL*, 2003.

[5] JURAFSKY D., MARTIN J.H. *Speech and Language Processing: An Introduction to Speech Recognition, Computational Linguistics and Natural Language Processing*. Prentice Hall, 2nd edition, 2008.

[6] KAIS D. Quranic arabic corpus (plain text, version 0.2). *Kais Dukes GNU Public License*, 2010.

[7] MAHMOUD E., KOULALI R. Kalimat a multipurpose arabic corpus. https://sourceforge.net/projects/kalimat/files/kalimat/Part%20of%20Speech%20Tagged%20Corpus, 2013.

[8] KAIMING H., XIANGYU Z., SHAOQING R., JIAN S. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, pp. 1026–1034, Washington, DC, USA, 2015. IEEE Computer Society. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.123. http://dx.doi.org/10.1109/ICCV.2015.123.

[9] SHIMODAIRA H. A weight value initialization method for improving learning performance of the backpropagation algorithm in neural networks. In: *Proceedings Sixth International Conference on Tools with Artificial Intelligence. TAI 94*, pp. 672–675, Nov 1994. doi: 10.1109/TAI.1994.346429.

[10] SUE ELLEN H., ANTONELLO P., CAREN M. *Artificial Intelligence Methods in the Environmental Sciences*. Springer Netherlands, 1st edition, 2009.

[11] LOSHCHILOV I., HUTTER F. SGDR: stochastic gradient descent with restarts. *CoRR*, abs/1608.03983, 2016. http://arxiv.org/abs/1608.03983.

[12] BOTTOU L., LECUIN Y. The backpropagation cookbook. In: *NIPS workshop: Trick of the Trade*, 1996.

[13] CHEN C.L., NUTTER R.S. Improving the training speed of three-layer feedforward neural nets by optimal estimation of the initial weights. In: *[Proceedings] 1991 IEEE International Joint Conference on Neural Networks*, pp. 2063–2068, 3, Nov 1991. doi: 10.1109/IJCNN.1991.170691.

[14] HUEI-HUANG L. Taguchi methods: Principles and practices of quality design. *Gau lih book Co. Ltd, Taiwan*, 2008.

[15] PAL M., BHARATI P. Introduction to correlation and linear regression analysis. In: *Applications of Regression Techniques*. Springer, Singapore, 2019.

[16] DRAGO G.P., RIDELLA S. Statistically controlled activation weight initialization (scawi). *IEEE Transactions on Neural Networks*, 3(4), pp. 627–631, Jul 1992. ISSN 1045-9227, doi: 10.1109/72.143378.

[17] SUR P., CANDÈS J.E. A modern maximum-likelihood theory for high-dimensional logistic regression. *Proceedings of the National Academy of Sciences*, 116(29), pp. 14516–14525, 2019. ISSN 0027-8424. https://www.pnas.org/content/116/29/14516.

[18] COLLOBERT R., WESTON J. A unified architecture for natural language processing: Deep neural networks with multitask learning. *Proceedings of the 25 th International Conference on Machine Learning*, 25, pp. 160–167, 2008.

[19] COLLOBERT R., WESTON J., BOTTOU L., KARLEN M., KAVUKCUOGLU K., KUKSA P. Natural language processing (almost) from scratch. In: *the Journal of Machine Learning Research*, pp. 2461–2505, 2011.

[20] YADAV S., SHUKLA S. Analysis of k-fold cross-validation over hold-out validation on colossal datasets for quality classification. In: *2016 IEEE 6th International Conference on Advanced Computing (IACC)*, pp. 78–83, Feb 2016. doi: 10.1109/IACC.2016.25.

[21] MIKOLOV T., CHEN K., CORRADO G., DEAN J. Efficient estimation of word representations in vector space. *Proceedings of Workshop at ICLR*, 2013.

[22] GLOROT X., BENGIO Y. Understanding the difficulty of training deep feedforward neural networks. In: Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pp. 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR. http://proceedings.mlr.press/v9/glorot10a.html.

[23] KIM Y.K., RA J.B. Weight initialization for improving training speed in the backpropagation network. *Proceedings of the International Joint Conference on Neural Networks*, 3, pp. 2396–2401, 1991.