Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2019) 27: 4418 – 4428

doi:10.3906/elk-1903-65

Improving word embeddings projection for Turkish hypernym extraction

Savaş YILDIRIM*
Department of Computer Engineering, Faculty of Engineering and Natural Science, İstanbul Bilgi University, İstanbul, Turkey

Received: 11.03.2019 • **Accepted/Published Online:** 27.06.2019 • **Final Version:** 26.11.2019

Abstract: Corpus-driven approaches can automatically explore is-a relations between the word pairs from corpus. This problem is also called hypernym extraction. Formerly, lexico-syntactic patterns have been used to solve hypernym relations. The language-specific syntactic rules have been manually crafted to build the patterns. On the other hand, recent studies have applied distributional approaches to word semantics. They extracted the semantic relations relying on the idea that similar words share similar contexts. Former distributional approaches have applied one-hot bag-of-word (BOW) encoding. The dimensionality problem of BOW has been solved by various neural network approaches, which represent words in very short and dense vectors, or word embeddings. In this study, we used word embeddings representation and employed the optimized projection algorithm to solve the hypernym problem. The supervised architecture learns a mapping function so that the embeddings (or vectors) of word pairs that are in hypernym relations can be projected to each other. In the training phase, the architecture first learns the embeddings of words and the projection function from a list of word pairs. In the test phase, the projection function maps the embeddings of a given word to a point that is the closest to its hypernym. We utilized the deep learning optimization methods to optimize the model and improve the performances by tuning hyperparameters. We discussed our results by carrying out many experiments based on cross-validation. We also addressed problem-specific loss function, monitored hyperparameters, and evaluated the results with respect to different settings. Finally, we successfully showed that our approach outperformed baseline functions and other studies in the Turkish language.

Key words: Projection learning, word embeddings, hypernym relation, deep learning

1. Introduction

Hypenymy indicates an is-a semantic relation between two nouns such as “cat-animal” or “Paris-city”. Computational approaches can automatically deduce such relations from raw text by applying corpus-driven approaches. This is called either a hypernym classification problem when classifying if two words given are in hypernym relation or hypernym extraction when pulling out the pairs based on some corpus statistics. Formerly, some studies utilized lexico-syntactic patterns [1]. They employed language-specific syntactic rules that matched the patterns embodying the word pairs. On the other hand, many studies have used distributional approaches to extract the pairs having such semantic relations. The distributional hypothesis relies on the idea that similar words share similar contexts and neighbors. They do not use predefined sources such as dictionary or linguistics rules. Rather, distributional behaviors of words in a corpus are taken into account [2, 3]. Former distributional approaches have applied one-hot encoding, also called bag-of-words (BOW). A word is represented by a vector that keeps count of how many times it cooccurs with its nearby words. However, such an approach raised the

*Correspondence: savas.yildirim@bilgi.edu.tr

problems of high dimensionality and sparsity. Recently, the dimensionality curse has been solved by various neural network approaches [4–8]. These studies showed that words could be represented in very short and dense vectors by means of neural layers, namely word embeddings.

In this study, we use word embeddings representation to solve the problem. We build a supervised model that extracts hypernym pairs from a corpus. Our design consists of an optimized supervised architecture that learns projection or mapping functions linking the word embeddings of hypernym pairs. It learns the function through word pairs that are in hypernym relations. The architecture mainly has two phases: learning word embeddings and learning projection function. We used our trained word embeddings built from a big corpus and also used pretrained embeddings resources for comparison. Word embeddings are produced by supervised neural networks. The network topology of embeddings is based on the idea that the model optimizes its parameters (word embeddings) by predicting a word by using its K nearby words to the left and right. Such topologies adapt the unsupervised nature of textual data to supervised learning. The optimized parameters are literally the vectors of the words at the end, or word embeddings.

We utilize the deep learning optimization methods to improve the performance. Other features such as loss function and epoch size are examined and tuned to learn a better projection function that would be capable of mapping embeddings of a hyponym to that of its hypernym. We discuss our results by carrying out many experiments, tuning hyperparameters, and evaluating them with respect to different settings. Finally, we show that our proposed approach outperforms other models employing baseline functions and other related studies in the Turkish language.

2. Methodology

2.1. Background

Neural networks have gained great attention by virtue of three driving technical forces: hardware, dataset, and algorithmic advances. Algorithmic advances in this field became possible just after a huge amount of data and computational power of hardware became available, and the methods using many neural layers have been referred to as deep learning. In the last few decades, very basic but important algorithmic improvements shaped the study of the field. Different activation functions, several optimization algorithms, better regularizations, and improved initialization methods have been studied and applied successfully to many problems. As an important phase of deep learning, recently many studies adapted efficient optimization algorithms by varying classical gradient descent, redefining regularization, varying momentum and adaptive learning factors, and so on [9, 10].

2.1.1. Optimizations

Gradient descent (GD) is one of the most popular methods to optimize neural networks. Deep learning libraries such as TensorFlow, Keras, and CNTK have several variants of GD for effective and faster optimization. The classical gradient descent algorithm uses a differentiable loss function, $J(\theta)$, and computes the update $\Delta\theta$ for each parameter θ by applying the formula $-\eta\frac{\partial J(\theta)}{\partial\theta}$. Even though this approach has been successfully applied to many problems so far, recently it has been found very slow as the data size increases. Therefore, it evolved in many ways. Another problem is that successive updates of gradient descent at each epoch may be very different, which causes high oscillations and prevents the model from convergence. The momentum approach accumulates an exponentially decaying moving average of past gradients m_{t-1} and continues to smooth the update instead of the gradient computed, as shown in Algorithm 1.

$$\begin{aligned} m_t &\leftarrow \frac{\partial J(\theta)}{\partial\theta} + \mu m_{t-1} \\ \Delta\theta &\leftarrow -\eta m_t \end{aligned} \tag{1}$$

Nesterov's accelerated gradient (NAG) is an alternative implementation of utilizing the momentum term for the gradient as shown in Algorithm 2 [11]. It uses a kind of improved momentum and has been found better than gradient descent. Some studies have provided empirical evidence that NAG could be superior to GD [12].

$$\begin{aligned} m_t &\leftarrow \frac{\partial J(\theta - \eta\mu m_{t-1})}{\partial \theta} + \mu m_{t-1} \\ \Delta\theta &\leftarrow -\eta m_t \end{aligned} \tag{2}$$

The fundamental problem in machine learning is that many models suffer from overfitting (variance) or underfitting (bias). The models are prone to memorizing the data and fail to generalize the problem, or they miss the global minima and underfit the data. Therefore, all studies regarding optimization are deeply involved in providing effective solutions to these problems. One solution to prevent overfitting is adding a regularization term to the loss function, $J(\theta)$. L2 is a norm regularization method to prevent the model from overfitting. In L2, the norm of the weight, $\theta^T\theta$, is added to the loss function as a regularization term in order to prevent the model from memorization and reduce error in validation. It is computed by the total value of the square of weights. L1 regularization is similar but it does not use the square, $|\theta|$. However, empirical studies showed that L2-norm is better than L1-norm regularization [10, 12]. Another important factor of the training phase of neural networks is adaptive learning. GD uses the learning factor η (mostly in the range of [0–0.2]) in order to specify the magnitude of update for the parameter. GD takes η as hyperparameter and does not change it during training. On the other hand, as the loss is reducing, adaptive learning increases η, which helps the model quickly find the minima. Likewise, it decreases the η as loss is increasing.

Some studies addressed that standard GD, NAG, and momentum could be very slow and they do not immediately converge under some conditions [9, 12]. Therefore, some alternatives have been developed using additional hyperparameters. Two widely used optimization algorithms, AdaGrad and RMSProp, use better L2-norm regularizations and adaptive learning [13, 14]. The AdaGrad algorithm accumulates the square of past gradients and uses it as a normalization factor, as shown in Algorithm 3.

$$\begin{aligned} grad &\leftarrow \frac{\partial J(\theta)}{\partial \theta} \\ norm_t &\leftarrow norm_{t-1} + grad^2 \\ \Delta\theta &\leftarrow -\eta \frac{grad}{\sqrt{norm_t} + \epsilon} \end{aligned} \tag{3}$$

One problem with AdaGrad is that learning becomes slower as the L2 norm vector, $norm_t$, increases. This prevents the model from reaching the local minimum. As a solution, RMSProp, as shown in Algorithm 4, uses an additional parameter, v, that controls the norm vector magnitude and allows the model to learn continuously [12, 14].

$$\begin{aligned} grad &\leftarrow \frac{\partial J(\theta)}{\partial \theta} \\ norm_t &\leftarrow v \cdot norm_{t-1} + (1-v) grad^2 \\ \Delta\theta &\leftarrow -\eta \frac{grad}{\sqrt{norm_t} + \epsilon} \end{aligned} \tag{4}$$

As an alternative solution to RMSProp, adaptive moment estimation (Adam) [15] successfully combines the momentum and RMSProp as formulated in Algorithm 5. Adam exploits the idea of momentum that simply directs the algorithm in a better direction and the idea of RMSProp that adapts how far the model moves in that direction. It has been found computationally efficient and it has fewer memory requirements [12]. The classical momentum is alternated by a decaying mean instead of a decaying sum, where the hyperparameters μ and v control the exponential decay rates of the moving averages. It has been shown that it improves performance for many problems [15].

$$\begin{aligned} t &\leftarrow t+1 \\ grad &\leftarrow \frac{\partial J(\theta)}{\partial \theta} \\ m_t &\leftarrow \frac{\mu m_{t-1} + (1-\mu) grad}{1-\mu^t} \\ norm_t &\leftarrow \frac{v \cdot norm_{t-1} + (1-v) grad^2}{1-v^t} \\ \Delta\theta &\leftarrow -\eta \frac{m_t}{\sqrt{norm_t + \epsilon}} \end{aligned} \tag{5}$$

Nesterov-accelerated adaptive moment estimation (Nadam) is another optimization technique that applies NAG to Adam, also called Adam with Nesterov momentum [12]. It simply ignores the initialization bias correction terms for the moment, as shown in the Algorithm 6.

$$\begin{aligned} t &\leftarrow t+1 \\ grad &\leftarrow \frac{\partial J(\theta)}{\partial \theta} \\ grad &\leftarrow \frac{grad}{1-\prod_i \mu_i} \\ m_t &\leftarrow \frac{\mu_t m_{t-1} + (1-\mu_t) \cdot grad}{1-\prod_i \mu_i} \\ m_t &\leftarrow (1-\mu_t) \cdot grad + \mu_{t+1} \cdot m_t \\ norm_t &\leftarrow \frac{v \cdot n_{t-1} + (1-v) \cdot grad^2}{1-v^t} \\ \Delta\theta &\leftarrow -\eta \frac{m_t}{\sqrt{norm_t + \epsilon}} \end{aligned} \tag{6}$$

2.2. Loss function

The loss function $J(\theta)$, a.k.a. the objective function, is the the measure that needs to be minimized during training. Each application needs a specific loss function. Generally speaking, while the binary classification application needs a binary cross-entropy loss function, multiclass or multilabel classification problems require categorical cross-entropy loss functions. Regression problems are mostly evaluated by mean of square error (MSE) or mean of absolute error (MAE) [10, 16]. The design of the loss function helps us to both update

the parameters and improve the model performance. Problem-specific hints and penalization could be helpful especially for overfitting, which needs to be added to the loss function. In this study, we observe that the most suitable loss function is cosine similarity-based loss function, which is cosine proximity as given in Equation 7. It simply shows that if the two vectors get close to each other, the loss value approximates to zero.

$$J(\theta; X, y) \leftarrow 1 - \frac{f(X;\theta) \cdot y}{||f(X;\theta)||_2 \cdot ||y||_2} \tag{7}$$

Another loss function for hypernym projection would be MSE or MAE, but since that we target an embedding vector, the most suitable one is cosine proximity that maximizes the similarity between predicted vector $f(X;\theta)$ and expected vector y.

2.3. Word embeddings

Advances in the field of deep learning help to build better word representation. Although one-hot encoding representation has been found capable of solving many NLP problems, some studies recently found these approaches ambiguous and insufficient [6–8, 17]. They proposed novel methods and applications, such as word2vec or glove, for dense and short representations inspired by neural-network topology, namely word embeddings. Even though word2vec is not considered as an example of deep learning due to the lack of deep layers, it still exploits many algorithmic advances in the field such as optimization, initialization, and so forth. It was proven that these modern representations are capable of understanding semantic and syntactic problems. The most famous example is that the difference (offset) between the vectors of *queen* and *king* is equal to that of pair *woman-man*. It has been shown that the model can differentiate gender concepts, which is not explicitly encoded in the corpus. Moreover, the word embeddings representation has also been applied to many NLP problems ranging from POS tagging to sentiment analysis [18–20].

2.4. Embeddings projection

We rely on the idea that the offset, or the difference between the embeddings, of two words that are in a hypernym relation can encode this relation of those words because it has been shown that offset can have a capacity of encoding semantic or syntactic relations [6–8]. In this study, we propose that embeddings of hypernym pairs can be converged to each other by means of the mapping function $f : \mathbb{R}^N \rightarrow \mathbb{R}^N$. The real valued vectors $v \in \mathbb{R}^N$ can be linearly or nonlinearly converged to each other. This idea might be true for other types of relations. In order to see the potential of offset, we tested the offset hypothesis that offsets between hypernym pairs are more similar than those of randomly selected word pairs. To see this, the offsets in two sets are projected to a 2D space by applying t-distributed stochastic neighbor embedding (t-SNE), which is a technique for dimensionality reduction [21]. t-SNE is particularly effective and widely used for the visualization of high-dimensional data such as word embeddings. As we notice from Figure 1, the offsets in two sets can be linearly separated from each other. The blue points (+) that are the offsets of hypernym pairs stand clear of the red circle points that are the offsets of random pairs.

If we provide negative examples, the problem can be solved as binary classification, but we take a more challenging task into account, which is an embedding projection problem that maps the vector of a word to that of its hypernym. The algorithm learns the projection model by using correct word hypernym pairs. Once converged, it can predict the target vector, $y \in \mathbb{R}^N$, by projecting source embeddings $x \in \mathbb{R}^N$. During the test

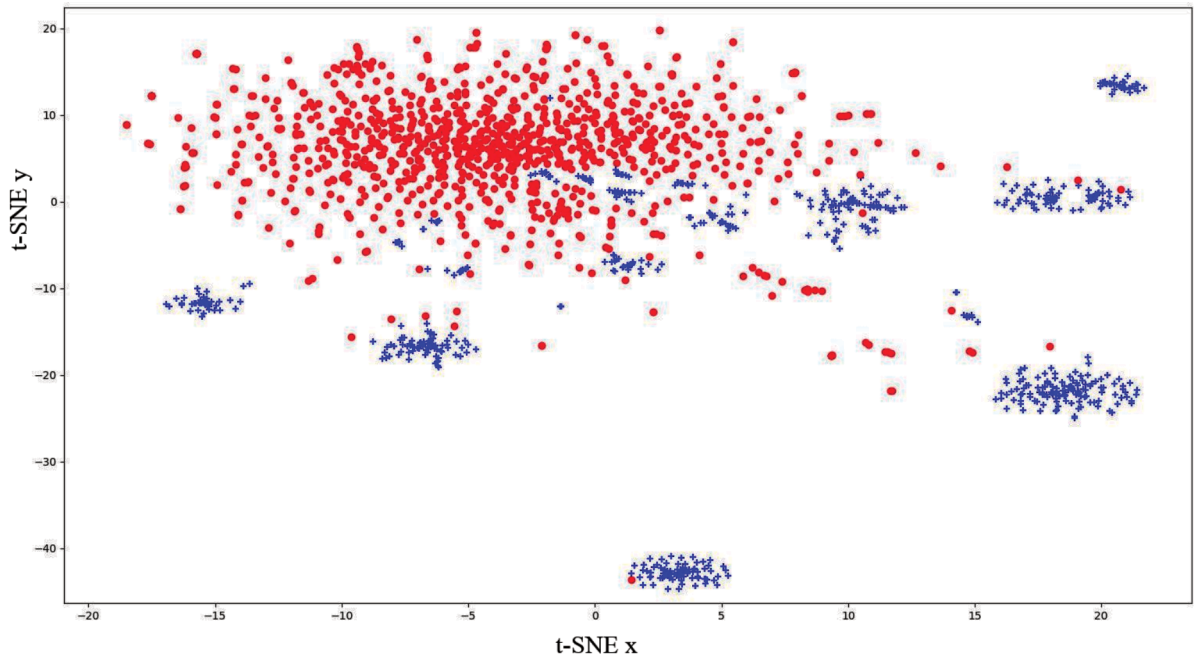

Figure 1. t-SNE mapping of hypernyms (blue) and random pairs (red).

phase, it of course could not precisely predict the exact location of target embeddings, but it can ideally predict a point that is closer to the target word than others. If the problem has linear characteristics, it could be solved by any of the regression algorithms. If not, nonlinear models can achieve the mapping. The training dataset includes a list of hyponym–hypernym pairs (e.g., "cat-animal"). The word embeddings of the hyponym (cat) are used as a source of input variables and those of the hypernym (animal) are used and targeted as output variables. The model learns is-a generalization during the training phase such that any given noun in the test data can be mapped to a closest point to its hypernym in the test phase. Deep learning offers a variety of architectures such as RNN, GRU, LSTM, CovnNet, and feedforward neural network (FFNNs) to solve NLP problems. We suggested and experimentally showed that FFNN is the most suitable architecture to effectively estimate embeddings vectors, as shown in Figure 2. The model takes the embeddings of a hyponym (e.g., "bird") and gradually transforms them through as many hidden layers as needed and finally estimates the hypernym (e.g., "animal"). The number of hidden layers and the number of units in each layer are problem-dependent and need to be tuned. We discuss the issue in further experimental sections.

During the training phase several optimization algorithms, regularizations, activation functions, and other hyperparameters have been evaluated. One can say that the disadvantages of the neural network is that it has many parameters that cannot be learned during the training phase, or hyperparameters, when compared to other learning paradigms such as kernel machines and Bayesian theory. This bottleneck is mostly solved by cross-validation and an additional secondary test set, namely a validation set. The entire dataset is divided into three sets: the training set, validation set, and test set. Training sets are used to train models and validation sets are used to monitor the performance of parameters. Once the hyperparameters are tuned, the model needs to be tested against test data to measure the realistic performance. We conducted our experiments on the basis of this evaluation criterion where cross-validation uses 5 folds and the validation split ratio is 0.2.

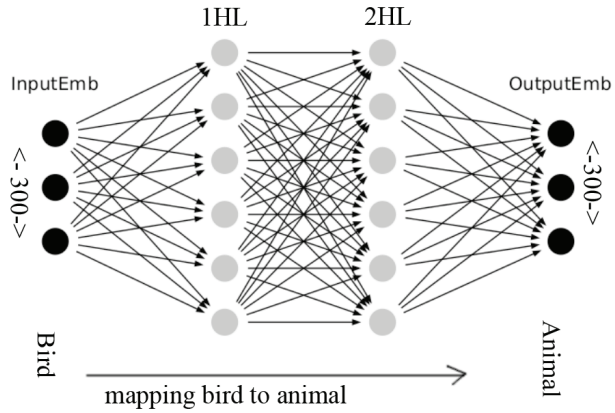

Figure 2. The architecture of projection FFNN.

3. Experimental results and discussion

We assessed the performance of SGD, NAG, Nadam, Adagrad, RMSProp, and Adam optimizers. These algorithms have been found very successful in recent studies [9, 12, 13]. The network architectures were implemented using TensorFlow API[1] and the Keras wrapper library.[2] We tuned some hyperparameters such that we found the optimum number of epochs as 15, which maximizes validation accuracy. The unit size of the hidden layer has been efficiently found as 300, which is equal to the size of input and output. The ReLU activation function outperformed the other functions and is selected as the activation function. Besides, we mostly used default settings of those hyperparameters that we do not need to tune. The learning factor, η, of the algorithm is set to 0.01 for RMSProp, Adam, and Nadam and 0.02 for AdaGrad. During the training phase, the batch size, which is the number of instances taken per update, is set to 32. The weight initialization scheme was random uniform, which generates unit weights with a uniform distribution within the range (-0.05 and 0.05). The validation set split is set to 0.2.

In the benchmark dataset there are 735 noun hypernym pairs, which are taken from another study for the Turkish language [22]. Pairs such as ($burdur \rightarrow il$) and ($doktor \rightarrow meslek$) have been randomly selected for a Turkish corpus, which is a natural language resource for Turkish. It contains about 2G tokens, compiled from the 5-year digital archives of a news agency. It was manually annotated in several categories such as economics, sports, and so forth. This resource is adequately representative for deep learning approaches in terms of length and coverage.

Two different word embeddings models have been tested. The first embedding model has been built by applying the word2vec algorithm to the Turkish corpus of 2G words defined above. Second, we also used pretrained word embedding thanks to fastText [17], which distributes pretrained word vectors for 157 different languages, trained on Common Crawl, a foundation producing an open repository of web crawl data. These pretrained word embeddings are trained by applying the CBOW approach of word2vec in dimension of 300, window size of 5, and negative words of 10. We keep the same parameters when training our first embeddings from the dataset. Studies that work on the Turkish language need to apply morphological analysis to textual data. We experimentally see that there is no difference between stemmed and raw textual data in terms of accuracy.

[1] tensorflow.org
[2] keras.io

The list of hypernym pairs has been represented in an embeddings dimension of 300. The problem can be taken as a regression problem. However, the main difference is that the target here is a vector of real values rather than a scalar. If we applied linear regression or other regression models such as random forest regression or variants of KNN, we would have needed to train 300 models. Rather, we solve the problem by simply using a neural networks topology that offers many advantages. Prediction of the vector can be done by keeping the number of units in the output layer as in the input layer, 300. We do not use the softmax function at the output layer since we do not use probability distribution. We can map input vector $X \in \mathbb{R}^N$ to output vector $Y \in \mathbb{R}^N$ by transformation the same as in an autoencoder that is a type of neural network. As a final decision step, we pick the closest word in the dictionary to the predicted vector, $\hat{y} \in \mathbb{R}^N$. Some studies [23] used the K closest candidate words when evaluating the model. We use K = 1 and the closest word is taken as predicted.

Another advantage of using neural networks is that we can easily shift from a linear model to a nonlinear model in the same architecture. Moreover, we can increase our model complexity either by adding new layers or by increasing the number of units. During the experiments, we tested many factors such as activation functions, regularizations, loss function, and optimizations. Figure 3 represents the performances of the linear mapping model, where inputs are directly mapped to output without using any hidden layer. The validation error curve is similarly tracking the training curve. The error quickly decreases over time and it reaches very low levels. The training and validation loss reach the minimum after only fifteen epochs and then stall. This figure indicates that the model does not overfit the training data. As the figure suggests, we adjust the number of epochs to 15.

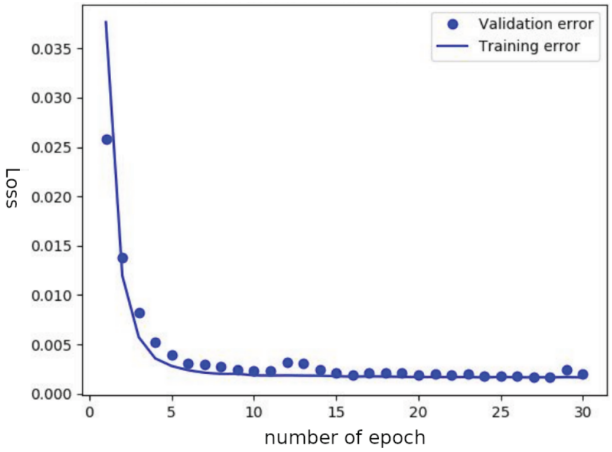

Figure 3. Training and validation loss.

Figure 4 shows the performance of different optimization algorithms with respect to cross-validation. The SGD and NAG algorithms (upper lines) learn very slowly and they demand very big epoch steps. Due to their slow learning characteristics, we discard them from further analysis. Besides, other optimization algorithms are already derived from these two algorithms. Indeed, this shows us why these novel optimization algorithms such as Adam and RMSProp have been studied recently. It also indicates that any algorithm is not prone to overfitting or underfitting and they do not oscillate either, which could cause higher variance. Even though AdaGrad is slightly faster than others, all obtain similar sufficient loss levels at no later than 20 epochs. We see the same performance in the Table.

We exploited two embeddings models. One is trained by applying the word2vec model to our corpus of 2G tokens; the second is taken from fastText [17]. The fastText model is slightly better than the first embeddings

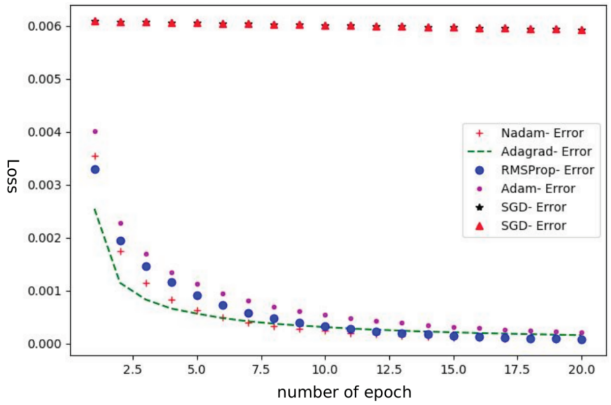

Figure 4. Performance of optimization algorithms.

in that it has used a huge amount of textual corpora to build word embeddings. The overall average accuracies of these two embedding models were measured as 90.55% and 89.5%, respectively. Therefore, we run the model using fastText embeddings. First, we applied a linear model without using any hidden layer (0HL) or any activation function, which is essentially just a linear regression model. Second, we also applied two nonlinear models where the activation function applies nonlinear transformation to the net inputs in hidden layers. It makes the network capable of learning and performing more complex tasks. The nonlinear models include a single hidden layer (1HL) and two hidden layers (2HL). These models were tested across four optimization algorithms and their performance are listed in the Table. The Table also includes the performance of another study using the same dataset in Turkish [22] for comparison.

Table . Performance of linear and nonlinear models across four optimizers.

Accuracy	Linear model - OHL	ReLU 1HL	ReLU 2HL
Nadam	90.4	90.74	90.2
AdaGrad	90.75	90.2	89.9
RMSProp	90.76	89.9	90.4
Adam	90.2	90.6	90.46
Other study [22]	88.9	-	-

In the literature there are three important and widely used activation functions: sigmoid, tangent, and ReLU [10, 16]. Our experiments showed that ReLU, $max(0, input)$, is superior to the other two functions, sigmoid $\frac{1}{1+e^{-x}}$ and tangent $\frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$. Therefore, we only report and discuss the models using the ReLU function. The loss function is cosine similarity in that we maximize the cosine similarity of vector $f(X;\theta)$ and target vector y.

We underline three findings based on the our experiments and observations. The Table showed that there is no significant difference between the linear model and nonlinear model. The 2HL model is slightly worse than 1HL, maybe due to increasing complexity and overfitting. We used four optimization algorithms where SGD and NAG are excluded because their learning capacity is found slow and they need many more epoch steps, such as 500. Our experiments also showed that cosine proximity is superior to MSE and MAE as well. One can say that, for simplicity, the linear model with the optimization of Adagrad or RMSProp, loss function of cosine proximity, and 15 epoch steps can learn a model for mapping a noun to its hypernym. The Table also indicates

that our models get a 90.75 % success rate and outperform the other study [22] that used the same dataset and applied the same problem. When we take SGD as the baseline function, our results shows that the proposed model can get faster and better results in terms of accuracy.

4. Conclusion

In this study we efficiently solve the hypernym problem by using a neural network topology where hypernym pairs have been represented in embeddings dimension. The advantage of using neural networks is that we can apply both linear and nonlinear models in the same architecture. In addition to that, we can increase our model complexity either by adding new layers or by increasing the number of units and tuning the parameters. We examined many factors such as activation functions, loss functions, optimization algorithms, and so forth to tune the model. We observed that the ReLU activation function is better than the sigmoid and tangent functions. The loss in the training and validation sets quickly reached the minimum by particular optimization functions such as RMSProp, AdaGrad, Nadam, and Adam. The SGD and NAG algorithms learn very slowly and need very big epoch steps. Even though AdaGrad is slightly faster than the other three algorithms, they all gets similar sufficient loss levels no later than at 20 epochs. We also observed that our architecture does not overfit the training data where the optimum epoch step size is found as 15.

We exploited two embeddings models. One is trained by applying the word2vec model to our corpus and the second one is taken from fastText [17]. The fastText model is slightly better than the first one produced by our corpus in that it has used a huge amount of textual corpora. We compared three training models: one linear model without using any hidden layer (0HL) and two nonlinear models that include a single hidden layer (1HL) and two hidden layers (2HL), respectively. We see that there is no significant difference between the models. The 2HL model is slightly worse than 1HL. These results obviously imply that the problem is linearly separable. We conclude that an optimized model should consist of a linear model with the optimization of Adagrad or RMSProp, loss function of cosine proximity, and 15 epoch steps. Such a model can learn a transformation function that links the hypernym pairs. Our results also indicate that the proposed model gets its best success rate of about 90.75% and outperformed the baseline functions and the other study in the literature that used the same dataset and worked on the same problem in the Turkish language.

References

[1] Hearst MA. Automatic Acquisition of Hyponyms from Large Text Corpora. In: ACL Conference on Computational Linguistics; Stroudsburg, PA, USA; 1992. pp. 539-545.

[2] Kotlerman L, Dagan I, Szpektor I, Zhitomirsky-Geffet M. Directional distributional similarity for lexical inference. Natural Language Engineering 2010; 16 (4): 359-389. doi: 10.1017/S1351324910000124

[3] Santus E, Lenci A, Lu Q, Schulte S. Chasing hypernyms in vector spaces with entropy. In: 14th Conference of the European Chapter of the Association for Computational Linguistics; Gothenburg, Sweden; 2014. pp. 38-34.

[4] Yu Z, Wang H, Lin X, Wang M. Learning term embeddings for hypernymy identification. In: 24th International Conference on Artificial Intelligence; Buenos Aires, Argentina; 2015. pp. 1390-1397.

[5] Turian J, Ratinov L, Bengio Y. Word representations: a simple and general method for semi-supervised learning. In: 48th Annual Meeting of the Association for Computational Linguistics; Stroudsburg, PA, USA; 2010. pp. 384-394.

[6] Pennington J, Socher R, Manning C. Glove: Global vectors for word representation. In: ACL Conference on Empirical Methods in Natural Language Processing; Doha, Qatar; 2014. pp. 1532-1543.

[7] Mikolov T, Yih W, Zweig G. Linguistic regularities in continuous space word representations. In: Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics; Atlanta, GA, USA; 2013. pp. 746-751.

[8] Rothe S, Ebert S, Schütze H. Ultradense word embeddings by orthogonal transformation. In: 2016 Conference of the North American Chapter of the ACL; San Diego, CA, USA; 2016. pp. 767-777.

[9] Sutskever I, Martens J, Dahl G, Hinton G. On the importance of initialization and momentum in deep learning. In: 30th International Conference on Machine Learning; Atlanta, GA, USA; 2013. pp. 1139-1147.

[10] Goodfellow I, Bengio Y, Courville A. Deep Learning. Cambridge, MA, USA: MIT Press, 2016.

[11] Nesterov Y. A method of solving a convex programming problem with convergence rate. Proceedings of the USSR Academy of Sciences 1983; 269 (3): 543-547.

[12] Dozat T. Incorporating Nesterov momentum into Adam. In: International Conference on Learning Representations Workshop; San Juan, Puerto Rico; 2016. pp. 1-4.

[13] Duchi J, Hazan E, Singer Y. Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research 2011; 12 (1): 2121–2159.

[14] Tieleman T, Hinton G. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. In: Tieleman T (editor). Neural Networks for Machine Learning. Toronto, Canada: University Of Toronto Technical Reports, 2012.

[15] Kingma D, Ba J. Adam: A method for stochastic optimization. In: International Conference on Learning Representations; San Diego, CA, USA; 2015. pp. 1-13.

[16] Bishop CM. Pattern Recognition and Machine Learning. New York City, NY, USA: Springer-Verlag, 2011.

[17] Grave E, Bojanowski P, Gupta P, Joulin A, Mikolov T. Learning word vectors for 157 languages. In: International Conference on Language Resources and Evaluation; Miyazaki, Japan; 2018. pp. 3483-3487.

[18] Collobert R, Weston J. A unified architecture for natural language processing: deep neural networks with multitask learning. In: 25th International Conference on Machine Learning; New York City, NY, USA; 2008. pp. 160-167.

[19] Guo J, Che W, Wang H, Liu T. Revisiting embedding features for simple semi-supervised learning. In: Conference on Empirical Methods in Natural Language Processing; Doha, Qatar; 2014. pp. 110-120.

[20] Pembeci İ. Using word embeddings for ontology enrichment. International Journal of Intelligent Systems and Applications in Engineering 2016; 4 (3): 49-56. doi: 10.18201/ijisae.58806

[21] Maaten LJP. Accelerating t-SNE using tree-based algorithms. Journal of Machine Learning Research 2014; 15 (1): 3221-3245.

[22] Yildirim S, Yildiz T. Learning Turkish hypernymy using word embeddings. International Journal of Computational Intelligence Systems 2018; 11 (1): 371-383. doi: 10.2991/ijcis.11.1.28

[23] Yildiz T, Yildirim S, Diri B. Acquisition of Turkish meronym based on classification of patterns. Pattern Analysis and Applications 2016; 19 (2): 495-507. doi: 10.1007/s10044-015-0516-9