Flexible Deep Neural Network structure with application to Natural Language Processing

Christopher Wittlinger Gerasimos Spanakis Gerhard Weiss

Department of Knowledge Engineering, Maastricht University, 6200 MD, Maastricht, The Netherlands

Abstract

In the past decade natural language processing has developed into one of the key fields in artificial intelligence with neural networks and deep learning techniques improving performance in several tasks. Deep neural networks require stacking a lot of layers which makes their training hard and slow while overfitting arises as an extra problem. This paper explores possibilities of tackling these issues by improving the so called "dropout" method and by exploring a more flexible constructive structure of the network. Conducted experiments show that dropping neurons according to their training behavior improves performance while a constructive approach can quickly lead to the close to optimal network structure while on the same time avoiding overfitting.

1 Introduction

Natural Language Processing (NLP) [3] is the research field that explores how computers can be used to understand and manipulate natural language text or speech in order to accomplish useful tasks. For example, given a specific document, entities, such as places and names, can be recognized, referenced words of pronouns can be highlighted, and the sentiment of a piece of text can be understood.

A neural network [7] is defined as a set of nodes (called neurons) connected through directed links, where each node is a process unit that performs a static node function on its incoming signal to generate a single node output. Neural networks are able to perform many tasks like pattern recognition, compression, prediction and can be applied to different kind of data (texts, images, etc) and to different fields (medicine, financial,etc).

Recently, Deep Learning (DL) [1] was developed (as a branch of Machine Learning) as a family of algorithms that attempt to model high-level abstractions in data by using complex structures and architectures. The idea is to utilize neural network layer structure by stacking many layers on top of each other, such that a breaking down mechanism is facilitated. Therefore, each layer in a Deep Neural Network (DNN) works as one transformation to further abstract the data.

Deep Neural Networks have been successfully applied to many NLP tasks [4] leading to effective systems but sometimes the size of the network (increased number of layers and neurons) as well as the training time are forbidding for efficient use. Despite this drawback, there is still little research work on ways to identify more efficient ways of training a DNN or on how to find an optimal structure (without training hundreds of different networks) and this is the main issue that the current paper is addressing.

The remainder of this paper is organized as follows: Section 2 will present the background work on the field. Sections 3 and 4 will introduce the ideas of a more flexible way of training Deep Neural Networks while experimental results will be presented in Section 5. Finally, Section 6 will conclude the paper.

2 Related work

2.1 Deep Neural Networks and Natural Language Processing

In every NLP task, the first and most important step is to come up with an effective representation of text. A successful way of representing documents is by introducing the idea of word embeddings [10], where each word is represented by a vector. Deep neural networks can be trained in order to discover latent vector representations of natural language words in a semantic space [8, 13, 17]. A word embedding $W : words \to \mathbb{R}^n$ is a parametrized function mapping words in some language to high-dimensional vectors (perhaps 200 to 500 dimensions). For example, we might find that: $w_{king} = (0.2, -0.4, 0.7, ...)$, $w_{queen} = (0.0, 0.6, -0.1, ...)$. W is initialized to have random vectors for each word. A neural network is utilized for predicting e.g. whether a 5-gram (sequence of five words) is "valid". Valid n-grams can be found in a large enough corpus (like Wikipedia) and then in order to create some invalid n-grams one can replace one word with a random one making the 5-gram nonsensical. For example, a valid 5-gram could be: "cat sat on the mat", whereas a non-valid could be: "cat sat song the mat". By this way, one can learn effective word embeddings that can be used as a base for every NLP task.

Utilizing such representations, deep learning has been also applied to several NLP tasks like Named Entity Recognition (NER) and Part-of-Speech (POS) tagging [4], question answering [9] and document classification [11]. Initial results suggest that DNN systems perform with comparable accuracy and speed with traditional methods. Advantages of such methods can be summarized in two facts: (a) a generic (regardless the task) approach and architecture can be constructed and (b) the internal representations (multiple layers of abstraction) are decided by the network itself. Biggest criticism comes from the fact that DNN approaches completely ignore the already acquired knowledge of features that perform well in specific NLP tasks (although sometimes being very specific).

2.2 Deep Neural Networks performance issues

Research in DNNs [15] has shown that typically when more layers are stacked on top of each other, performance is improved due to the fact that the data is abstracted in a larger degree. The presence of many layers and neurons may lead to overfitting as well as to slow training (convergence), therefore it would be useful to develop techniques that avoid these issues.

Recently, a technique called "dropout" [16] was proposed as a way to avoid Neural Networks overfitting. This method randomly switches on/off neurons only for one epoch during the training phase, generally with probability 0.5. By this way, it is made sure that the network is trained to be as general as possible and does not get too specialized to the data of the training set. In the end you can imagine the final network as a fusion of different neural networks with good generalization capability. Experiments showed that such a network performs significantly better than benchmark neural networks on many different datasets and reasonably better on text classification problems. An extension to dropout method was proposed (called "standout") [2] which associates highly correlated units and combines them into a single hidden unit with a lower dropout probability. Despite the fact that selection of neurons to be dropped is not completely random, the structure of the network remains the same throughout the training and freed units are used for other purposes.

In the next two Sections, two improvements to DNNs performance will be presented: The first one presents an alternative of selecting neurons to be dropped and the second one provides a framework for a more flexible structure (allowing adding/deleting neurons). Both approaches are tested on their performance in a document classification task, presented in the Experiments Section.

3 Making the dropout method more deterministic

In this Section we are exploring possibilities to differentiate the dropout criterion according to a specific criterion and not randomly (with equal probability $p = 0.5$ for each neuron). The focus is on DNNs trained with backpropagation algorithm but the same idea could also be applied to different structures (like Restricted Boltzmann Machines). For the purpose of demonstration consider a neural network like the one in Figure 1 and suppose the following notation for the hidden layer l (where neuron j belongs):

- $\mathbf{z}^{(l)}$ is the vector of inputs into layer l,

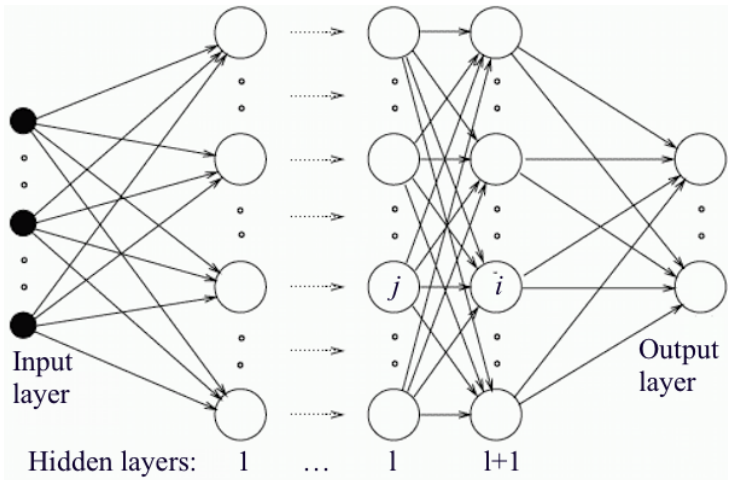

Figure 1: Standard Neural Network with Back Propagation training

- $\mathbf{y}^{(l)}$ is the vector of outputs from layer l,
- $\mathbf{w}^{(l)}$ are the weights at layer l and
- $\theta^{(l)}$ are the biases for layer l

Then, the feed-forward operation of a standard neural network for any neuron j can be described by the following Equations:

$$z_j^{(l+1)} = \mathbf{w}_j^{(l+1)} \cdot \mathbf{y}^{(l)} + \theta_j^{(l+1)} \tag{1}$$

$$y_j^{(l+1)} = f\left(y_j^{(l+1)}\right) \tag{2}$$

where f is the activation function.

Similarly, given that the output error from the feed-forward operation at the output layer is $\delta^{(L)}$, the back-propagation error for any layer l ($l \in 1, 2, ..., L-1$) is given through the following Equation:

$$\delta^l = (\mathbf{w}^{(l+1)})^T \cdot \delta^{(l+1)} \circ f'(z^l) \tag{3}$$

where f' is the derivative of activation function, T is the transpose operator, and $\circ$ is the Hadamard Product (element wise operator between elements of vectors). After the backpropagation of the errors, the weights are updated according to the following Equation (similar operation is applied to the bias updating):

$$\Delta w^{(l)} = -\alpha \cdot \delta^{(l+1)} \cdot (y^{(l)})^T \tag{4}$$

where α is the learning rate. For a full derivation of these Equations readers are encouraged to refer to the literature [6].

During training, the changing characteristics of any neuron (j) are the propagated error (δ_j) and the weight vector change (w_j). The idea here is to temporarily dropout neurons that their structural content does not change (i.e. their weight vectors remain almost the same or neurons that the backpropagated error doesn't significantly change). For example, if the error that is backpropagated to a neuron (or similarly the neuron's weight vector) does not significantly change between two epochs then it might be temporarily switched off. Significant change is determined by a threshold ϵ which can either set empirically or by experimenting.

The selection of neurons that are switched off is determined by the following Equations for the two criteria (first one checks for significant changes in the error and the second one for significant changes in the weight vector).

$$\text{Criterion 1: } \left\{ j : \sum_{i=1}^{N_a} \|\mathbf{w}_{j,i}(t) - \mathbf{w}_{j,i}(t-1)\| \leq \epsilon \right\} \tag{5}$$

$$\text{Criterion 2: } \left\{ j : \sum_{i=1}^{N_a} \|\delta_{j,i}(t) - \delta_{j,i}(t-1)\| \leq \epsilon \right\} \tag{6}$$

where j and i refer to the synapses between any two neurons j and i (obviously neuron j is at layer l and neuron i at layer $l+1$ just like in Figure 1), t refers to current epoch and $t-1$ to the previous one, N_a is the number of neurons at layer $(l+1)$. Emphasis is given to the individual synapses change and their weighted sum is taken into account. In the case of dropped neurons there are also some changes to Equation 2 since $y_j^{(l+1)}$ has to be zero for the neurons that satisfy Equation 5 or 6. A slight variation to this dropout would be not to deterministically select all neurons that satisfy the above criterion but instead just increase their probability to be dropped (i.e. not be $p = 0.5$ like in the original dropout, but slightly more). More insights on this will be presented in the Experiments Section.

4 Making the structure of the network more flexible

A neural network has a fixed structure and once initialized, it is not possible to change that structure any more. This is often considered a drawback, because different problems or different datasets may require different structures. One could argue that the dropout method presented in the previous Section already provides such a flexibility, but here the idea is to actively alter the network structure by adding neurons. This idea is in accordance to biological neural development models and the basic principles of adaptation [18] (neurons adapt to their input sources, can be interchangeable or reusable in cases structure changes) and constructivism [14] (development of neural structures is gradual and appropriately biased by the environment).

The proposed method uses a simple criterion to add a hidden neuron to the neural network, based on the back-propagated error. When the back-propagated error associated with a neuron does not reduce by an amount ϵ after a specific number of training epochs τ, then it is assumed that the current neuron has reached its modelling limit. In detail, the criterion is described using the following Equation:

$$\sum_{i=1}^{N_a} \|\delta_{j,i}(t) - \delta_{j,i}(t+\tau)\| \leq \epsilon \tag{7}$$

where $t = \tau, 2\tau, 3\tau, ...$ and the rest symbols were explained in Equations 5 and 6. If more than one neuron fulfil this criterion, then all the corresponding layers are expected to be enhanced by another new neuron.

Instead of just adding a neuron and inspired from biology cell division process [12], we replace the selected neuron (one that satisfies the criterion of Equation 7) by two new neurons, which share the same number of connections as the parent neuron. The weights of these new neurons are calculated as in [12]:

$$\mathbf{w}(1) = (1+\gamma) \cdot \mathbf{w} \tag{8}$$
$$\mathbf{w}(2) = -\gamma \mathbf{w} \tag{9}$$

where $\mathbf{w}$ is the weight vector of the parent neuron, $\mathbf{w}(1)$ is the weight vector of the first new neuron, $\mathbf{w}(2)$ is the weight vector of the second new neuron and γ is the mutation parameter which takes a random value according to a Gaussian distribution with a mean of zero and a variance of one.

5 Experimental results

Several experiments were conducted in order to evaluate the proposed extensions. Hardware used for the experiments are an 8-Core AMD Computer with 8 GB of RAM and the task selected was the classification of the 20 NewsGroup (20NG) dataset which consists of 18845 documents taken from the USENET newsgroup collection [1]. Each post belongs exactly to one category (in total there are 20). Word embeddings, as described in Section 2 are used as a representation method and dataset was accordingly

[1]https://archive.ics.uci.edu/ml/datasets/Twenty+Newsgroups

split to training, testing and validation set. All results (from different experiments) presented here are an average over 10-fold-cross-validation processes.

5.1 Experimenting with different dropout methods

The first experiment implements the idea of random dropout and compars it to the extensions of adaptive dropout and the two methods proposed in Section 3. Increasing the chance of a neuron to be dropped when the backpropagated error change is low, shows an improvement, because when such neurons are dropped, then the network only tries to improve in respect to neurons with a high backpropagated error. On the other hand, using the weight vector difference, does not improve the performance of the network and is not robust at all. Results can be seen (along with traditional and adaptive dropout) in Table. Due to different implementations (Dropout and Adaptive Dropout were checked using the scikit-learn package, while our proposed methods (DROPW, DROPE) were implemented from scratch) training time could not be compared.

Table 1: Applying different dropout variations ($\epsilon = 1E - 04$, $p = 1$) on 20NG dataset

	Accuracy	STDEV
Dropout (DROP)	0.791	0.021
Adaptive Dropout (Standout) (STAND)	0.821	0.011
Dropout Acc. to weight vector change (DROPW)	0.788	0.074
Dropout Acc. to back-prop. error change (DROPE)	0.829	0.018

Moreover, several experiments were conducted in order to define whether it is better to just dropout all neurons that satisfy criterion 2 (we only work with that since criterion 1 does not yield good results) or just increase their probability to be dropped. Comparison was conducted using a fixed network architecture (3 hidden layers 400-500-400) and using a subset of the original dataset (around 5000 documents in order to facilitate speed and check for overfitting). Probability to drop the neurons satisfying the criterion we set was varied from $p = 0.5$ to 1 with step 0.05, while probability to drop the rest neurons was set to $1 - p$. Figure 2 shows the training and test error with respect to p. We see that training error remains almost flat as p increases (result of a good training) and testing error improves until $p = 0.85$ and is increased (but not significantly) as p becomes close to 1. Accuracy for these experiments did not change significantly (performed always over 0.81 till 0.83) so one could assume that the value of p is not that sensitive to affect accuracy. Results in Table 1 were obtained using $p = 1$.

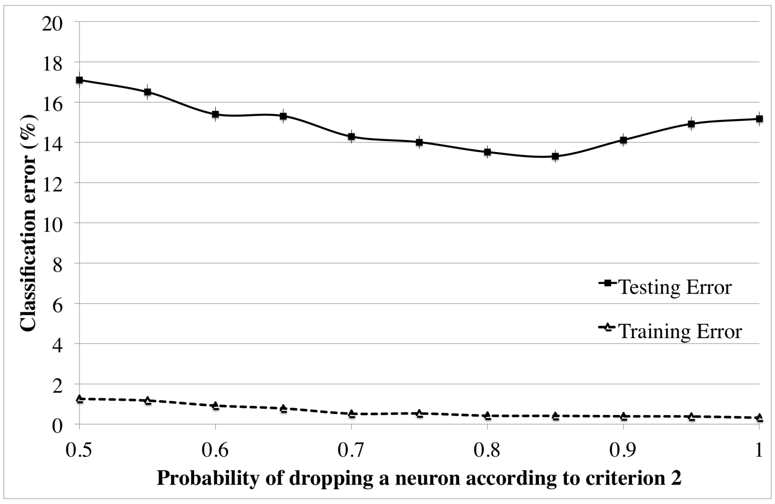

Figure 2: Effect of probability p of dropping a neuron on a subset of 20NG

Another hyperparameter that needs to be tuned is the threshold which determines whether to drop or not a neuron and is denoted by ϵ in Equations 5 and 6. Empirically this can be set to $1E - 04$. Note that different values of p and ϵ can also affect performance, so a combined analysis was conducted using the same dataset (part of 20NG) as the one for investigating parameter p and for checking all combinations of p (range from 0.5 to 1) and ϵ (range from $1E-06$ to $1E-02$). Full results are not presented here due to space limitations but Table 2 presents the effect of ϵ given $p = 1$.

Table 2: Effect of parameter ϵ to DROPE algorithm, tested on a subset of 20NG

ϵ value	% Training Error	% Testing Error	Epochs
1E-02	5.26%	18.39%	130.3
1E-03	4.58%	16.61%	218.5
1E-04	2.94%	13.26%	302.3
1E-05	3.01%	15.86%	540.5
1E-06	2.88%	17.47%	692.6

It can be seen that a relatively large value of ϵ is beneficial for both training and testing error. Very large values (like $1E-02$) increase both training and testing error because more neurons are dropped while smaller values (like $1E-06$) may provide comparable training error but this clearly leads to overfitting (since many epochs are needed) and testing error increases (generalization fails). Results in Table 1 were obtained using $\epsilon = 1E-04$.

5.2 Experimenting with a constructive approach

Second batch of experiments applied the method described in Section 4. For these experiments, parameters τ (controls the rate of epochs that new neurons are added) and ϵ (controls the ratio of error change) have to be tuned. Several experiments were performed using the same subset of the dataset as the one for parameter analysis of the DROPE method, where τ was varied from 10 to 50 and ϵ was varied from $1E-06$ to $1E-02$. Given this analysis, τ was set to 20 and ϵ was set to $1E-03$ for the rest of the experiments as they were found to be the optimal ones. Also, Table 3 summarizes the results for $\epsilon = 1E-03$ and for the whole range of τ. It can be seen that a moderate value of τ is beneficial for both the number of epochs and the training error. Otherwise, the algorithm tends to add more neurons at the same time, which increases both the training error and the number of epochs required to run.

Table 3: Effect of parameter τ to the constructive approach, tested on a subset of 20NG

τ value	number of neurons	% Training Error	% Testing Error	Epochs
10	1208.8	3.23%	18.39%	363.2
20	1202.4	4.22%	16.61%	374.1
30	1239.2	5.12%	13.26%	602.1
40	1323.7	6.40%	15.86%	945.8
50	1388.4	7.68%	17.47%	1270.2

Given the description of the constructive approach, network can add neurons at any time during the training (given of course the value of τ). Training progress along with the number of hidden neurons are presented in Figure 3.

As it can be seen, network tries to add more neurons at the very beginning of the training for consecutive epoch ranges and then it works with this structure until it reaches its performance limit. At this point the same process is repeated again. Given various experiments and different startup architectures (varied number of layers and neurons), approach followed had the same characteristics: addition of neurons in the beginning, training with this structure and then repetition of the same process, sometimes also at the very end of the training.

Finally, another experiment was performed in order to evaluate the final structure created by utilizing two neural networks: The first neural network (DROPE NN) uses a fixed structure (which is close to optimal given previous experiments) of 3 hidden layers (400-500-400 neurons) and the second one (CONSTR NN) starts from a smaller network consisting of 3 hidden layers each one consisting of 50 neurons. We could start from a more minimal structure but starting with a modest number of neurons facilitates training and is in agreement with the neural development which suggests that nearly all neural cells used through the lifetime have been produced in the first months of life [5]. The results of this

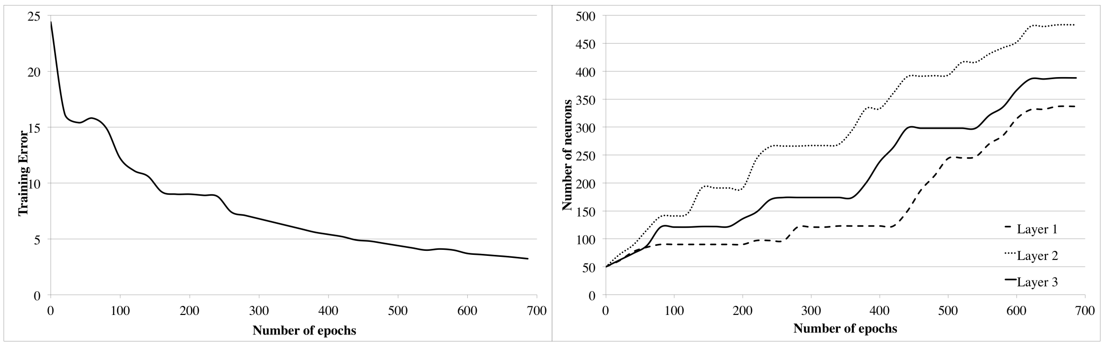

Figure 3: Training process and network construction by the proposed algorithm using 20NG dataset

experiment can be seen in Figure 4. The constructive neural network is adding neurons as training advances and ends up with approximately the same accuracy (and almost the same structure) as the fixed one. However, it takes longer for CONSTR NN to converge but the number of extra epochs maybe compensated from to the fact that for new datasets optimal architecture is not known beforehand. It can also be seen that accuracy improvement is in accordance with the addition of new neurons: When number of neurons is almost stable, then accuracy is increased but in epoch ranges where neurons are massively added accuracy does not improve.

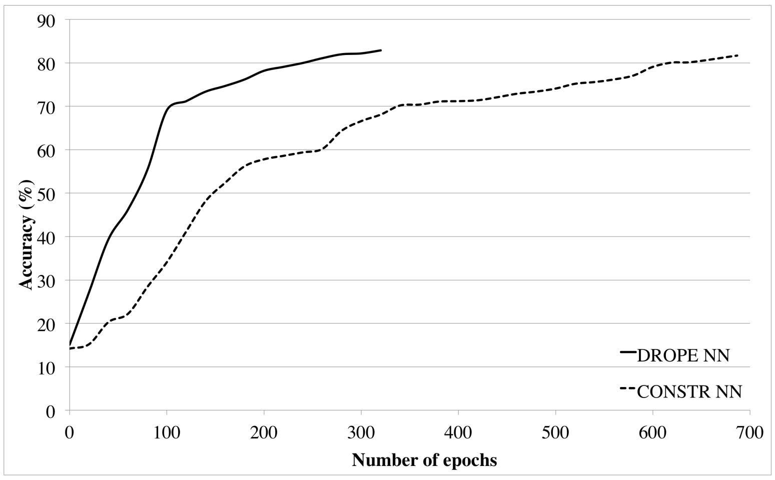

Figure 4: Comparison of DROPE and constructive neural network on 20NG dataset

6 Conclusion

In this paper, Deep Neural Networks in the domain of Natural Language Processing were explored. Dropout method was enhanced with an effective criterion which takes into account the back-propagated error (and thus the progress of the training). Dropping neurons with this more deterministic way yields better results in respect to both training and testing error but also training time. An attempt to approach the close to optimal neural network structure in a constructive way (by adding neurons to the layers) showed interesting results despite the fact that it increased training time.

The prospect of self-building and/or adaptable neural networks is promising and can be further explored. Future work involves exploring ways of removing neurons (by merging similar units) and even add or remove layers when is needed, a process which will allow for truly flexible neural networks. Training time remains a sensitive issue so techniques in order to improve that (especially when using flexible and changing network structures) will also be considered as extensions to current work.

References

[1] Itamar Arel, Derek C Rose, and Thomas P Karnowski. Deep machine learning-a new frontier in artificial intelligence research [research frontier]. *Computational Intelligence Magazine, IEEE*, 5(4):13–18, 2010.

[2] Jimmy Ba and Brendan Frey. Adaptive dropout for training deep neural networks. In *Advances in Neural Information Processing Systems*, pages 3084–3092, 2013.

[3] Gobinda G Chowdhury. Natural language processing. *Annual review of information science and technology*, 37(1):51–89, 2003.

[4] Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research*, 12:2493–2537, 2011.

[5] John E Dowling. *The Great Brain Debate: Nature Or Nurture?* Princeton University Press, 2007.

[6] Martin T Hagan, Howard B Demuth, Mark H Beale, et al. *Neural network design*. Pws Pub. Boston, 1996.

[7] Simon S Haykin, Simon S Haykin, Simon S Haykin, and Simon S Haykin. *Neural networks and learning machines*, volume 3. Pearson Education Upper Saddle River, 2009.

[8] Eric H Huang, Richard Socher, Christopher D Manning, and Andrew Y Ng. Improving word representations via global context and multiple word prototypes. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers-Volume 1*, pages 873–882. Association for Computational Linguistics, 2012.

[9] Mohit Iyyer, Jordan Boyd-Graber, Leonardo Claudino, Richard Socher, and Hal Daumé III. A neural network for factoid question answering over paragraphs. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 633–644, 2014.

[10] Omer Levy and Yoav Goldberg. Neural word embedding as implicit matrix factorization. In *Advances in Neural Information Processing Systems*, pages 2177–2185, 2014.

[11] Lawrence McAfee. Document classification using deep belief nets. *CS224n, Sprint*, 2008.

[12] Stevan V Odri, Dusan P Petrovacki, and Gordana A Krstonosic. Evolutional development of a multilevel neural network. *Neural Networks*, 6(4):583–595, 1993.

[13] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)*, 12:1532–1543, 2014.

[14] Steven R Quartz and Terrence J Sejnowski. The neural basis of cognitive development: A constructivist manifesto. *Behavioral and brain sciences*, 20(04):537–556, 1997.

[15] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[17] Nitish Srivastava, Ruslan R Salakhutdinov, and Geoffrey E Hinton. Modeling documents with deep boltzmann machines. *arXiv preprint arXiv:1309.6865*, 2013.

[18] Van J Wedeen, Douglas L Rosene, Ruopeng Wang, Guangping Dai, Farzad Mortazavi, Patric Hagmann, Jon H Kaas, and Wen-Yih I Tseng. The geometric structure of the brain fiber pathways. *Science*, 335(6076):1628–1634, 2012.