2016 International Conference on Computing, Analytics and Security Trends (CAST)
College of Engineering Pune, India. Dec 19-21, 2016

Effect of Parameter Variations on Accuracy of Convolutional Neural Network

Kaushik Gobindram Pasi
Department of Computer Engineering and Information Technology,
Veermata Jijabai Technological Institue, Mumbai
kaushikpasi@gmail.com

Sowmiyaraksha R. Naik
Department of Computer Engineering and Information Technology,
Veermata Jijabai Technological Institue, Mumbai
sraksha@vjti.org.in

***Abstract*—In this paper, we implement a Convolutional Neural Network especially designed for Natural Language processing. With the help of this CNN, we try to classify sentences for sentiment analysis for which the embeddings used were learned from scratch rather than using pre-trained word2vec vectors. Here we try to vary the different parameters and learn how they effect on the performance of the CNN. From the observations we try to demonstrate that a fairly less-complex CNN that has a small amount of parameter adjustments and fine-tuning can achieve a significant growth in performance.**

***Keywords*—Convolutional Neural Network, Natural Language Processing, Deep Learning, Sentence Classification, Character Embedding, Feature Map, Pooling, Regularization, Learning Rate**

I. Introduction

Models based on deep learning have achieved remarkable results in many fields, especially in the domain of computer vision and speech recognition. In natural language processing, much work using deep learning involved learning word vector representations using neural network based language models and performing compositions for classification on the learned word vectors. CNNs use layers with filters that can be convolved which can then be applied to local features. They were originally designed for computer vision purposes but CNN models have subsequently shown to be very effective for natural language processing and have achieved efficient results in semantic parsing, search query retrieval, sentence modelling and many other tasks related to NLP.

In this paper, we do not use the pre-trained word vectors like word2vec or GloVe for word embeddings, rather we tried to learn embeddings from scratch. Also we did not enforce L2 norm constraints on weight vectors. Ye Zhang and Byron Wallace found that constraints have very small effect on end result [18]. Here we use only static word vectors and learn other parameters of the model. On slight tuning of the hyper-parameters, the model achieves impressive results on multiple datasets. Learning task specific vectors by fine-tuning further improves the results.

II. Background and Motivation

Recently, word embeddings have been widely used for sentence classification using CNN architectures. Kalchbrenner suggested a CNN architecture with multiple layers in convolution[7]. Johnson and Zhang [6] proposed a similar model, but they swapped high-dimensional 'one-hot' vector representation of words as CNN inputs. Their focus was on classifying longer texts instead of sentences.

The relatively simple architecture as proposed by Johnson and Zhang [6], modulo the word vectors - coupled with observed strong empirical performance makes this a strong contender to supplant existing text classification baselines such as SVM and logistic regression. But in practice, one has to make several model architecture decisions and set various hyper-parameters. At present, very less empirical data is available to help make such decisions; addressing this is our aim here.

III. Prerequisite

A. Convolutional Neural Network

The most easiest way to understand convolution is by thinking of a sliding window function made to run over a matrix. Now just think of several such layers of convolutions with nonlinear activation functions like *ReLU* and *tanh*. In feed-forward neural network, each input neuron is connected to each output neuron in next layer, which is called as a fully connected layer. Each layer applies different task specific filters, typically hundreds or thousands and combines their results. The two aspects of this computation which are worth paying attention to are: Location Invariance and Compositionality.

The input to provided most NLP related tasks are matrices. Each row of matrix corresponds to one token, usually a word. Usually these vectors are word-embeddings like Word2Vec [12] or GloVe. In NLP, filters are slided over rows of these matrices.

Let $x_i \in \alpha^k$ be the k-dimensional word vector corresponding to the i^{th} word in the sentence. A sentence of length n (padding optional) is represented as:

$$x_{1:n} = x_1 \ast x_2 \ast ... \ast x_n \tag{1}$$

where $\ast$ is concatenation operator. Convolution operation has a filter $\omega \in \alpha^{lk}$ which is applied to a window of l words to generate a new feature, viz., a feature e_i is produced from a window of words $x_{i:i+l-1}$ by

$$e_1 = f(\alpha . x_{1:i+l-1} + b) \tag{2}$$

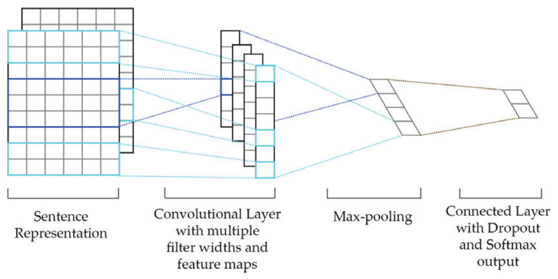
Sentence Representation
Convolutional Layer with multiple filter widths and feature maps
Max-pooling
Connected Layer with Dropout and Softmax output

Fig. 1: CNN architecture for sentence classification [8]

where $b \in \alpha$ is the bias term and f is the non-linear function. This filter is applied to every possible window of words in the sentence $\{x_{1:l}, x_{2:l+1}, ..., x_{n-l+1:n}\}$ to produce a feature map.

$$e = [e_1, e_2, ..., e_{n-l+1}] \tag{3}$$

with $e \in \alpha^{n-l+1}$. Now apply max over pooling operation [5] over the feature map and consider maximum value $\hat{e} = max\{e\}$ as feature that corresponds to that particular filter. This is to capture the most significant feature (one with the highest value) for each feature map. Pooling scheme automatically deals with variable sentence lengths.

The process described extracts one feature from one filter. The model uses multiple filters (having varying window sizes) to get multiple features. These features make up the last layer that are passed to fully connected softmax layer which gives output that is the probability distribution over labels.

IV. Model

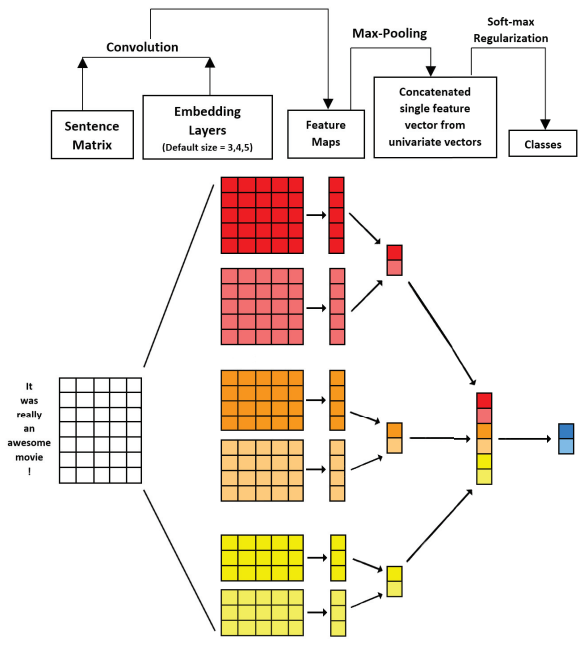
Convolution
Max-Pooling
Soft-max Regularization
Sentence Matrix
Embedding Layers (Default size = 3,4,5)
Feature Maps
Concatenated single feature vector from univariate vectors
Classes
It was really an awesome movie !

Fig. 2: CNN model for sentence classification

A. Embedding Layers

This layer maps word indices from vocabulary into low-dimensional vector representations. It is fundamentally a look-up table that we learn from the data. For initializing this we use a random uniform distribution.[17]

B. Convolution

Here α the filter matrix and the above created embedded layers are convolved. Each filter slides is slided over the entire embedding but varies in the number of words it covers.[4] The convolution operation will compute the output on the neurons connected to local regions in the input, which is a hyper-parameter called as *Receptive Field*[13]. After convolution from each filters, the vectors that we obtain feature vectors[1]. The other hyper-parameters that control the output size of the convolutional layer are as follows:

1) Padding:
Padding is required to add zeros to the border of the input. While sliding the filters over the sentences, the different types of padding applied are *VALID* where only partial windows are included for narrow convolution and *SAME* where entire window is included for wide convolution. Padding is used to preserve the spatial size of the input.[15]

2) Stride Sizes:
Stride controls how width and height dimensions are allocated. Small stride sizes leads to quite heavily over-lapping receptive fields and large output. For bigger strides, the receptive fields will overlap a less which results into output having smaller dimensions[3].

C. ReLU (Rectified Linear Unit) Layer

It is applied element-wise to apply non-linearity to the overall network without having any affect on the receptive fields of the convolutional layer. It has benefits like one-sided compared to the anti-symmetric *tanh*, sparse activation, efficient and easier computation.[11]

D. Pooling

Max-pooling is applied over the output which is essentially a non-linear down-sampling operation along the spatial dimensions. Pooling operation reduces the spatial size of the representation which helps to reduce parameters and also computations, hence control over-fitting.[2]

E. Regularization

Dropout[16] is the most widely used empirical regularization method to regularize convolutional neural networks. A dropout layer disables a fraction of its neurons which prevent neurons from co-adapting and tries to force them to learn individually useful features. L2 regularization is a common regularization method implemented by imposing penalty on the squared magnitude of every parameters directly into the objective. For the purpose of regularization, a dropout is employed on the penultimate layer with a constraint on l_2-norms of the weight vectors[10]

V. Experiments

The model is tested on various polarised datasets of IMDB movie reviews[14] given by anonymous users which help in finding class labels as positive or negative and checking for correctness and accuracy. The datasets were divided into train/dev for training the model and testing purposes.

A. *Experiment 1*

In this experiment, we try to observe the impact by making changes to the following:

- **Default:** Above values of parameters.
- **Zero:** Initialize weights α and b to zero instead of truncated normal-distribution.
- **Padding:** Pad front of the sentence with five PAD symbols not just the end.
- **Learn:** Increased the rate of learning for optimizer.
- **Except:** Combining Zero initialization and increasing the learning rate but not padding the front of the sentences.
- **All:** Combining all the above variations to the model with default parameters.

Following parameter values are kept constant:

- Dimensionality of character embedding: 300
- Filter sizes: 3,4,5
- Number of filters per filter size: 100
- l_2 regularization lambda: 0.15
- Dropout Keep Probability: 0.5
- Batch size: 64
- Number of training epochs: 25

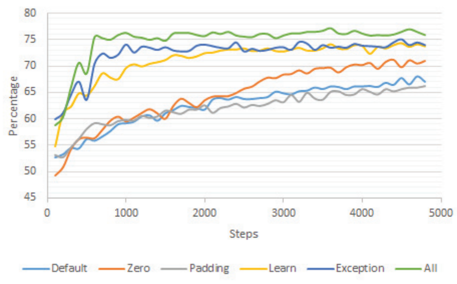

Fig. 3: Accuracy comparison of variations of Experiment 1

TABLE I: Experiment 1 Accuracy Results

Default	Padding beginning of sentences	Initializing weights to zero	Increasing Learning Rate of Optimizer	All variations except Padding	All variations altogether
68.1	66.3	71.2	74.29	75.0	77.2

B. *Experiment 2*

- Dimensionality of character embedding: 128
- l_2 regularization lambda: 0.0
- Dropout Keep Probability: 0.5
- Batch size: 64
- Number of training epochs: 100

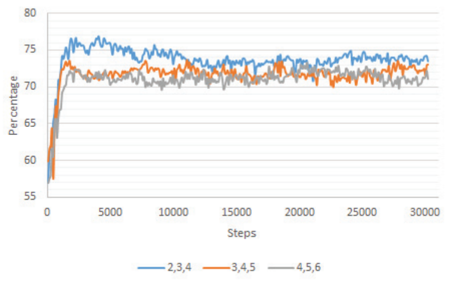

Fig. 4: Accuracy comparison of filter size variations of Experiment 2

TABLE II: Experiment 2 Accuracy Results

Filter Sizes	2,3,4	3,4,5	4,5,6
Accuracy	76.9	73.7	73.3

C. *Experiment 3*

Here we try to observe the impact by making changes to the regularization size with following parameter values constant:

- Dimensionality of character embedding: 128
- Filter sizes: 3,4,5
- Dropout Keep Probability: 0.5
- Number of filters per filter size: 128
- Batch size: 64
- Number of training epochs: 100

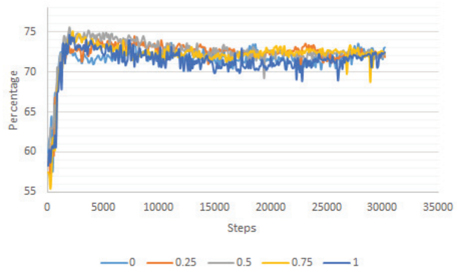

Fig. 5: Accuracy comparison of regularization variations of Experiment 3

TABLE III: Experiment 3 Accuracy Results

Regularization	0.0	0.25	0.5	0.75	1.0
Accuracy	73.7	74.5	75.6	75.5	74.6

D. *Experiment 4*

Here we try to observe the impact by making changes to the embedding size with following parameter values constant:

- Number of filters per filter size: 128
- Filter sizes: 3,4,5
- l_2 regularization lambda: 0.0
- Dropout Keep Probability: 0.5
- Batch size: 64
- Number of training epochs: 100

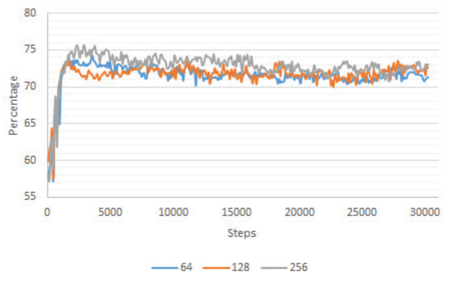

Fig. 6: Accuracy comparison of embedding size variations of Experiment 4

TABLE IV: Experiment 4 Accuracy Results

Embedding Size	64	128	256
Accuracy	74.3	73.7	75.6

E. Experiment 5

1) Base Model: In this experiment, the results from Experiment 1 are shown:

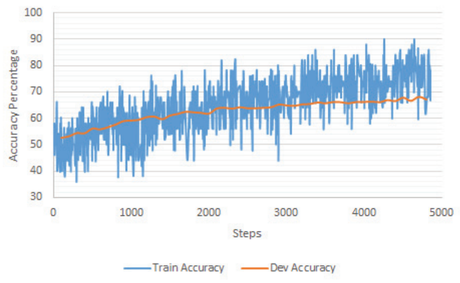

Fig. 7: Accuracy comparison of Train and Dev data of Experiment 1

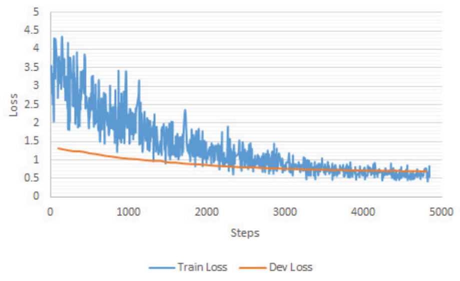

Fig. 8: Loss comparison of Train/Dev data of Experiment 1

TABLE V: Experiment 5(1) Results

Measurement	Accuracy		Loss	
Dataset	Train	Dev	Train	Dev
Max	90.00	68.10	4.33	1.32
Average	65.18	62.62	1.35	0.89

2) Model with suggested variations

: In this experiment, we try to combine all the good variation results from the above experiments.

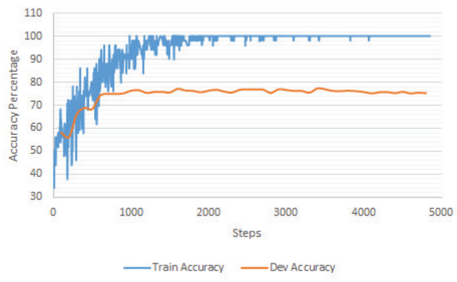

Fig. 9: Accuracy comparison of Train and Dev data of model with suggested variations

The parameters used for the suggested model are as follows:

- Dimensionality of character embedding: 300
- Filter sizes: 2,3,4
- Number of filters per filter size: 100
- l_2 regularization lambda: 0.5
- Dropout Keep Probability: 0.5
- Batch size: 50
- Number of training epochs: 25

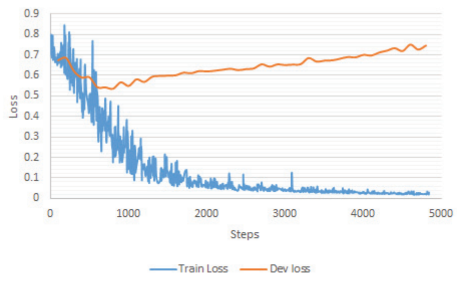

Fig. 10: Loss comparison of Train and Dev data of model with suggested variations

TABLE VI: Experiment 5(2) Results

Measurement	Accuracy		Loss	
Dataset	Train	Dev	Train	Dev
Max	100.00	77.40	0.85	0.75
Average	94.30	74.74	0.15	0.64

VI. OBSERVATIONS

On obtaining the results from the above performed experiments, the following observations are made:

Experiment 1:

- **Zero Initialization:** While training, we found that the cross-entropy loss was pretty high initially and it took almost half of the epochs to get lower than 1.0.

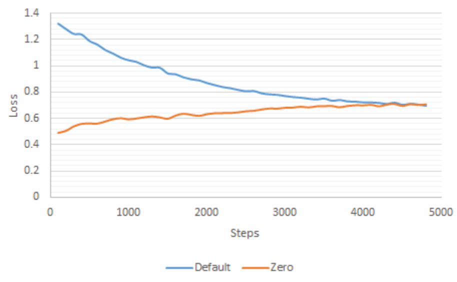

Fig. 11: Loss comparison for non-zero initialization and with zero-initialization from Experiment 1

This is quite unusual in a binary classification problem where it should start from near about $-ln(0.5) \approx 0.69$ and start declining from there. The dependency was traced to the initialization of the weights α and b in the last layer where the initial random values created a high loss value at the start and the optimizer was taking longer to climb down.

- **Padding at the beginning:** Just guessing that putting padding at the start of the sentences allows the model to discriminate words appearing near the beginning of sentences. This might allow the model to pick out some patterns involving the global placement of words in a sentence. The other idea is that it means more feature maps per sequence which might just be a good thing. Also to be noted that just padding at the beginning did not work alone (68.1 to 66.3), but when combined with other variations gave better result than just those variations (75.0 to 77.2).
- **Increasing the learning rate of the optimizer:** By increasing the learning rate of the optimizer (Adam optimizer[9] used here) gave a significant boost in accuracy (68.1 to 74.29).

Experiment 2:

- From the results, it was observed that when the filter size 2,3,4 was used, it yielded better accuracy (76.9) compared to filter sizes 3,4,5 and 4,5,6 (around 73.5 ± 2)
- But by looking at the graph it feels that accuracy does not get a significant boost on a longer run but the do notice that an increase by 3% is still a considerable increase.
- Also note that the plot for the filter size of 2,3,4 was at the top of all the other plots throughout the run.

Experiment 3:

- The variations in regularization did not yield a very convincing result but still there are a few noticeable accuracy results.
- For lower and higher regularization values, the accuracy was less by 2% compared to 0.5 where the accuracy peaked out signifying that too much or too less regularization is not suitable, rather moderately using regularization yields faster and better results.

Experiment 4:

- This variation in embedding size yielded somewhat confusing graph as for the embedding size of 64, the accuracy was more (74.3) but for the size of 128, the value was the least (73.7) and surprisingly for the value of 256, the accuracy peaked out (75.6).
- This fluctuation in accuracy does not suggest any significant role of changing embedding size for improving the performance. It may be due to the vocabulary that was generated from the dataset used.

Experiment 5 (1):

- The learning rate is slow and the curves show that the network is learning from both the datasets
- The accuracy of both the datasets are less and even their averages depict the same
- Similar is the case with loss where the loss in both the datasets are quite high especially while training and even their aveerages say so.

Experiment 5 (b):

- The learning rate of both the datasets are pretty good as compared to base model
- Accuracy that we get from training data is very good because of the learning rate
- Also the accuracy of dev dataset is better compared to the base model where the maximum accuracy we could reach was 68.1 with an average of 62.62 but using the suggested parameters we could boost up the accuracy upto 77.4 with an average of 74.74
- From the loss curve of the training data we can observe that the loss has significantly reduced (maximum 0.85 and average 0.15) compared to the loss curve of the base model (maximum 4.33 and average 1.35)
- The loss curve for dev looks a bit strange at first sight, that the curve starts declining rapidly but after a few steps, it gradually starts rising. But still from the values of the curve that loss seems to be less (maximum 0.75 and average 0.64) as compared to base model's loss (maximum 1.32 and average 0.89)

VII. Conclusion

The model developed is a fairly simple Convolutional Neural Network with some extra components like regularization which were not the compulsory part of the CNN. This model is trained from scratch and not pre-trained word-vectors like *Word2Vec*,yet it yields pretty competitive results when compared to other Deep Models like RNN, LSTM, etc. The motive behind choosing CNN for this paper was that they are faster mainly because of their architectural size in which other models are huge and very deep spatially also having more number of parameters with them which make them more complex. The results obtained by this from-the-scratch trained model were close $\approx 77-78$) to the model which used pre-trained *Word2Vec* [12] word-vectors ($\approx$ 80 for training the model which is far more accurate as it has been undergoing training using lots of data)[8].

The significant conclusion noted from the observations are as follows:

- When small batch sizes are used, the training metrics are not smooth. A larger batch size, is suggested to smoothen it.
- The training loss and accuracy starts quite below the dev metrics when dropout is applied.
- Zero initialization of the weights is suggested as loss is comparatively less and climbs down a bit more faster.
- Increasing the learning rate for optimizer is preferred for improving the training speed.
- Padding the sentences at the beginning alone does not help but when used with a faster learning rate and zero initialization of the weights.
- By observing the variations in filter sizes, smaller filter sizes are recommended to improve the accuracy with the significant impact on training time. As lower filter sizes will result into more matrix multiplications.
- The role played by embedding size on the model is fairly negligible as it mainly depends on the vocabulary of the dataset and less on the size of the embedding layer.
- Keeping a fairly moderate regularization, i.e. not penalizing the network more or less yields a better set of results.
- With the default parameters, the accuracy is significantly below the training accuracy, which it feels like the network is over-fitting the training data, suggesting the stronger regularization or fewer parameters. For this, by adding additional l_2 penalties for weights at the last layer, the accuracy bumped up from around 65%-70% to 76%.
- From the observations of Experiment 5, we can say that our suggested values do work very well achieving the accuracy of 77.4.
- All the expectations were met as the learning rate increased, so did the accuracy. Loss was also reduced with the only concern of the shape of the loss curve for dev data.

The set of experiments that we performed gave us results based on which we tried to to test the network with those variations. The metrics of the results were improved convincingly. We made a better understanding of the parameters and their usage in the model during this work and tried to showcase those understandings through the observations and conclusions. By tuning the parameters, we tried to show that a simple CNN can yield good results.

References

[1] Ossama Abdel-Hamid1, Li Deng, and Dong Yu. “Exploring Convolutional Neural Network Structures and Optimization Techniques for Speech Recognition”. In: *Interspeech* (2013).

[2] Y-Lan Boureau, Jean Ponce, and Yann LeCun. “A Theoretical Analysis of Feature Pooling in Visual Recognition”. In: *Proceedings of the 27th International Conference on Machine Learning* (2010).

[3] Adam Coates, Andrew Y. Ng, and Honglak Lee. “An Analysis of Single-Layer Networks in Unsupervised Feature Learning”. In: *Journal of Machine Learning Research 15:215-223* (January 2011).

[4] Michael Collins and Nigel Duffy. “Convolution Kernels for Natural Language”. In: *Advances in neural information processing systems* (November 2002).

[5] R. Collobert et al. “Natural Language Processing (Almost) from Scratch”. In: *Journal of Machine Learning Research* 12 (2011), pp. 2493–2537.

[6] Rie Johnson and Tong Zhan. “Effective use of word order for text categorization with convolutional neural networks”. In: (2014), arXiv:1412.1058.

[7] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. “A convolutional neural network for modelling sentences”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics* 1 (2014), pp. 655–665.

[8] Yoon Kim. “Convolutional Neural Networks for Sentence Classification”. In: *Empirical Methods on Natural Language Processing* (2014).

[9] Diederik Kingma and Jimmy Ba. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference for Learning Representations, San Diego* (2015).

[10] Krizhevsky, I. Sutskever, and G. Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Proceedings of NIPS* (2012).

[11] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. “Rectifier Nonlinearities Improve Neural Network Acoustic Models”. In: *International Conference on Machine Learning, ICML* (2013).

[12] Tomas Mikolov et al. “Efficient Estimation of Word Representations in Vector Space”. In: *arXiv:1301.3781v3 [cs.CL]* (September 2013).

[13] John Moody and Christian Darken. “Learning with Localized Receptive Fields”. In: *Research Report YALEU/DCS/RR-649* (September 1988).

[14] Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan. *Movie Review Data*. https://www.cs.cornell.edu/people/pabo/movie-review-data/.

[15] Patrice Y. Simard, David Steinkraus, and John C. Platt. “Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis”. In: *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6* (August 2003).

[16] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* (2014), pp. 1929–1958.

[17] J. Weston, F. Rattle, and R. Collobert. “Deep Learning via Semi-Supervised Embedding”. In: (2008).

[18] Ye Zhang and Byron Wallace. “A Sensitivity Analysis of (and Practitioners’ Guide to) Convolutional Neural Networks for Sentence Classification”. In: (2015), arXiv:1510.03820.