# Convolutional Neural Network for Text Classification

**Shikhar Vashishth**
M.Tech CSA
IISc Bangalore
shikhar.vashishth@csa

**Gandreti Santhosh B**
PhD EE Department
IISc Bangalore
santhoshg@ee

**Rakesh Kumar Mallik**
M.Tech CDS
IISc Bangalore
rakesh@grads.cds

## Abstract

This project explores the use of Convolutional Neural Networks in the field of Natural Language Processing. It uses a CNN model for sentiment analysis of movie reviews. It also compares other competing approaches like LSTM and especially designed SVMs for solving this problem. It also demonstrates different methods for visualizing CNN networks for text classification namely occlusion and contribution visualizations which give deep insight into the working of CNN networks.

## 1 Introduction

With ever growing volume of data being generated every second, processing the data and extracting useful information from them is very essential and can change the world we view today. For processing such data especially vision related images and video, Convolutional Neural network or Deep learning has been a huge success. The concept of neural network is not new, but it has been dormant for several decades mainly because of limitation of computational power. Introduction of high computational machines like GPU, high speed processor and large storage has helped the scientific community to take over the deep learning to its peak today. CNN is primarily invented for computer vision related applications, but has worked wonders in the field of text and NLP. Many prominent and traditional NLP problems like sentence modeling [3], search query retrieval [5], semantic parsing [6], and other traditional NLP tasks [7], are addressed by CNN. In this project, we concentrate on sentiment classification of movie reviews, and a detailed description of our work, starting from preprocessing of data, till results and visualization of the trained models, has been explained briefly in the following sections.

## 2 Dataset Preprocessing

Stanford sentiment tree-bank dataset [12] consists of about 12k sentences with sentiment values for 2.4 million phrases. The dataset doesn't provide the labels of the given sentences directly and moreover, sentences have a lot of encoding/escaping inconsistencies. So, for getting the label for each sentence, we first found its closest match in the given list of phrases by calculating its Levenshtein distance [10] from each phrase and assigning it the sentiment value of the closest phrase. Based on the sentiment value assigned, the sentences were classified into five classes: very negative, negative, neutral, positive, very positive. We converted all the sentences to lower case and used Porter2 stemming algorithm [13] for replacing each word with its root word. Then, based on the spaces each sentence was divided into a list of words which were encoded using the embedding vector for each word.

## 3 CNN architecture

We have used Convolution Neural network and Support Vector Machine for classifying sentences into one of the five classes. Our CNN model is based on the model proposed by Yoon kim [1], which is shown in Figure 1. The model takes sentences as input in the form of list of words, where each word is represented by a 300 dimensional vector. This is fed as an input independently to three convolution layers with 100 filters of sizes: 3, 4, and 5. The output of convolution layers is concatenated after applying max-pool and ReLU activation function and is passed through two dense layers to produce the final prediction. Based on the type of embedding used for representing words, the model can be classified into three types: CNN random, CNN static, and CNN non-static. In CNN rand, embeddings are initialized randomly and are refined during training. In CNN static, embedding are initialized with word2vec embeddings [2] and are kept constant throughout training and in CNN non-static embeddings are initialized with word2vec and are allowed to get modified during training. The penultimate dense layer contains 100 neurons with a dropout rate of 0.5 and the number of neurons in the final dense layers is equal to the number of classes, which is 5 in our case. Apart from this, we have also trained a vanilla version of LSTM model for our problem whose results have been shown in the later sections.

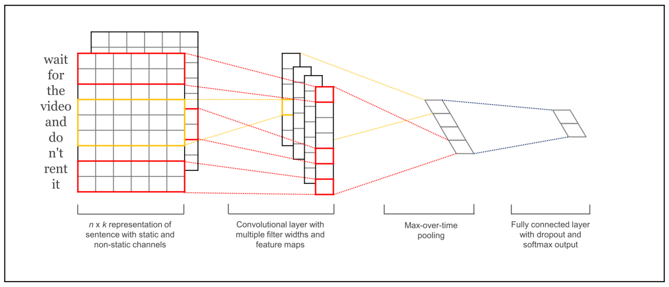

wait
for
the
video
and
do
n't
rent
it
*n* x *k* representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 1: Yoon Kim's CNN architecture

### 3.1 CNN Training

The training process took the maximum time in this project. According to [8], Stanford sentiment treebank dataset has too less data for training CNNs. There are less than 12k sentences with duplicates and inconsistent encoding/escaping practices. Therefore, for verifying the correctness of our implementation, we first trained our network for a binary classification problem using IMDB dataset [14] which has about 50k sentences each belonging to either positive or negative class. The Figure 2 shows our training results. For all the models (CNN rand/static/non-static) we got almost 100 % accuracy on the training data and on test data our accuracy was around 86 %. These results clearly verify that our model has been correctly implemented.

While training for the multi-class classification problem using SST dataset, our results were not as good as with the binary case. In the binary case, our network got converged in about 200 epochs but for this problem, we had to train the networks for about 1200 epochs to reach convergence. In the binary case, we used a constant learning rate of 0.01 with stochastic gradient descent optimizer but in multi-class case we had to use different learning rates during training. We initially started with 0.01 and then tried learning with lower rates like 0.001 and 0.005. Final test accuracy are given in the table below:

| **Classification accuracy** | Binary (IMDB) | Multi-class (SST) |
|---|---|---|
| CNN Rand | 89.39 | 41.75 |
| CNN Static | 88.25 | 38.33 |
| CNN Non-Static | 89.02 | 42.08 |
| LSTM | 89.97 | 42.22 |

Table 1: CNN, LSTM Results

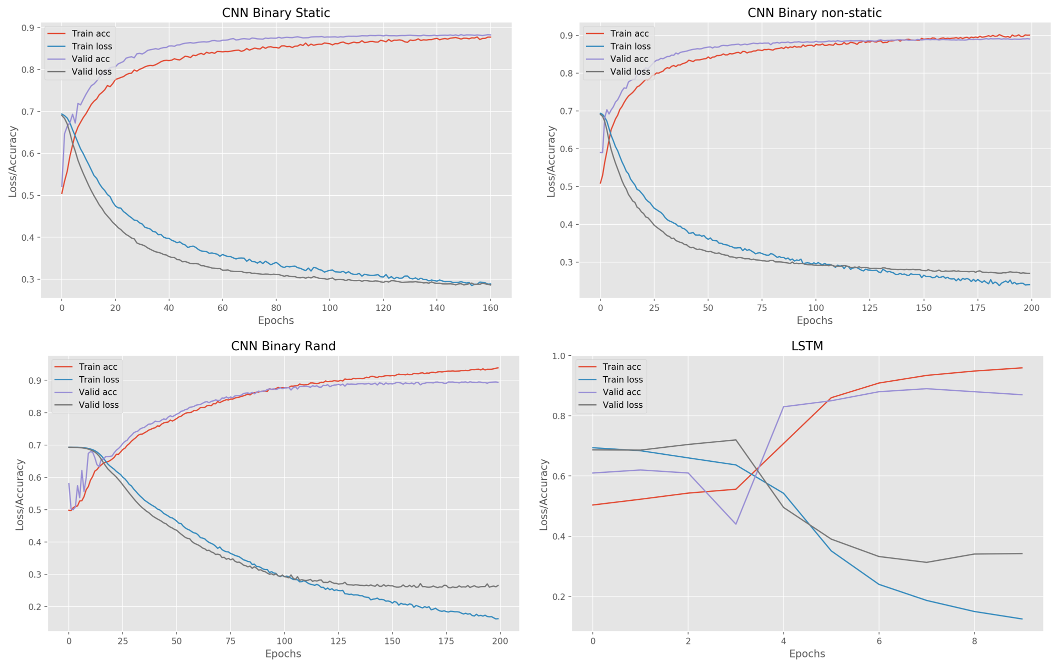


Figure 2: Training for binary classification problem (IMDB Dataset)

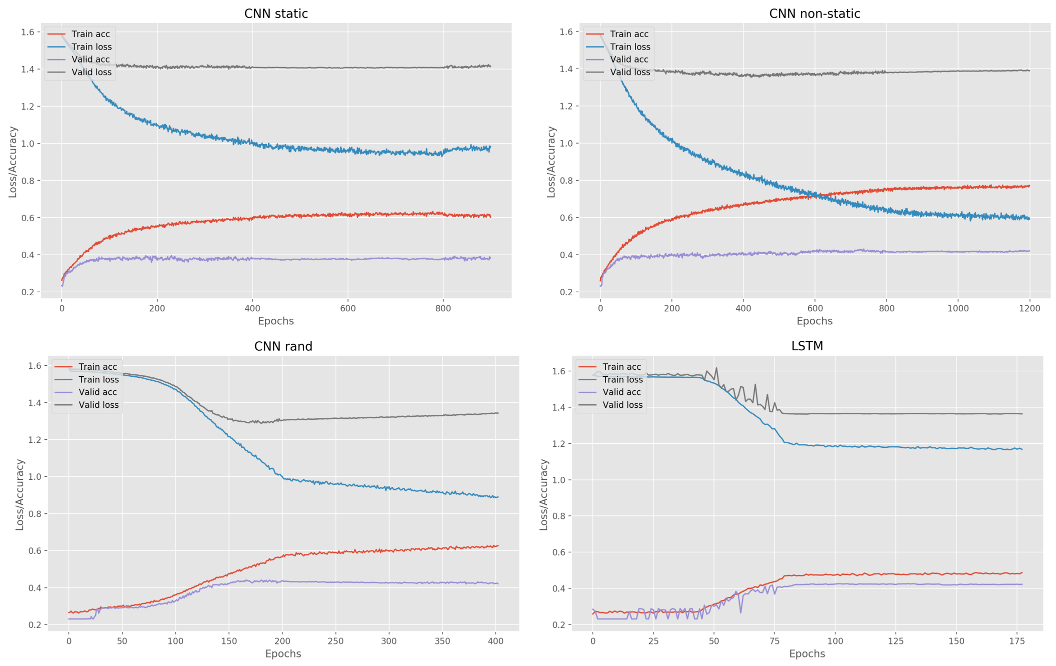


Figure 3: Training for Multi-class classification problem (SST Dataset)

## 4 Support Vector Machine: Implementation

A collective vocabulary $V$ of unigrams, bigrams and trigrams is generated from the training dataset and a total of 2,59,992 keywords are obtained in the vocabulary. Along with these keywords, for every $k^{th}$ keyword, the frequency of its occurrence in the whole training dataset $p(k)$ is also calculated and a log-count vector for these keywords is calculated as

$$r = -log(p/||p||_1)$$

For every $i^{th}$ sentence in the training dataset, feature vector $f_i$ is generated such that $k^{th}$ feature of the feature vector is turned on if the $k^{th}$ keyword of the vocabulary is present in the sentence.

A linear SVM is trained using two kinds of input features, 1) $x_i = f_i$ and 2) $x_i = f_i \circ r$ where $\circ$ is the element wise multiplication operator. The classification accuracies on overall training dataset and for individual classes for both the methods are tabulated below.

| **Classification accuracy** | **1-of-V encoding** | **log-count encoding** |
|---|---|---|
| Training accuracy | 100 | 100 |
| Testing accuracy | 39.547511312217193 | 39.049773755656109 |
| Classification accuracies of individual classes | | |
| very negative | 24.0143369176 | 19.7132616487 |
| negative | 49.9210110585 | 52.9225908373 |
| neutral | 22.1079691517 | 19.794344473 |
| positive | 53.137254902 | 55.4901960784 |
| very positive | 33.5839598997 | 28.320802005 |

Table 2: SVM Results

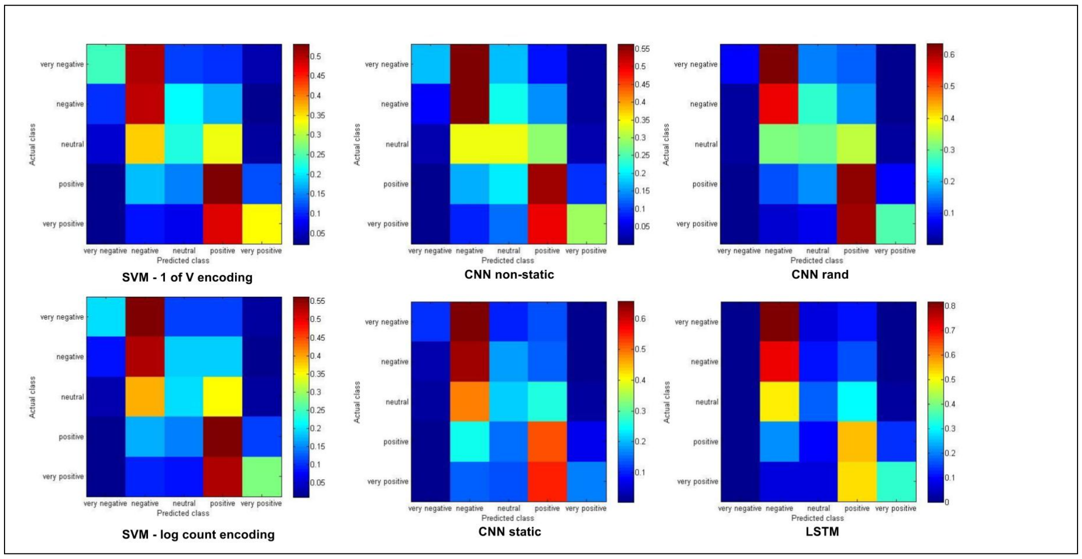


Figure 4: Confusion matrices for SVM, CNN and LSTM

From the above Figure 4, it can be noticed that there is a lot confusion between the classes, very-positive and positive, hence most of the very-positive statements are being misclassified as that of positive sentiment. The same is the case with the classes, very-negative and negative. One can also infer from the confusion matrices that there is not much confusion between negative and positive statements.

## 5 Visualizing CNN

CNN is widely used for images, and several approaches have been proposed in the recent years to understand the Net for better performance in the field of images. Visualizing CNN for text is still a hot topic and research is going on to derive novel methods for the same. In this report, we made an attempt to imitate the visualization techniques for images, and apply it on text, so as to understand the inner layers of the Net. Input modification methods and Deconvolution methods are well known methods for visualizing CNN.

In input modification methods, the main aim is to detect the change in output or intermediate layers of the Net, by little modifications in the input. One of the popular technique of this kind is occlusion, where some of the salient words of the text are occluded and passed through the neural network. Significant changes can be seen in the output, if salient words of the text are occluded. Thus importance of various words present in the sentence can be observed by the changes that appear at the output. This can be well explained from the following table which shows the changes in the activations at the output for a sample sentence ”It’s fun lite”.

| | **very-neg** | **negative** | **neutral** | **positive** | **very-pos** |
|---|---|---|---|---|---|
| Original output | 0.00452792 | 0.04054867 | 0.15729867 | **0.48529199** | 0.31233275 |
| **Occluded word** | **Changes in the output activations with occlusions** | | | | |
| it | 0.00866606 | 0.02533537 | 0.10399928 | **0.63300163** | 0.2289976 |
| ’s | 0.00566753 | 0.02529909 | 0.09437066 | **0.61354113** | 0.26112157 |
| fun | 0.10562841 | 0.25280347 | **0.32173645** | 0.24159348 | 0.07823817 |
| lite | 0.01264636 | 0.02639236 | 0.16317147 | **0.59786379** | 0.19992596 |

Table 3: Occlusion Visualization for a sample sentence

This technique is implemented on some of the testing samples and the changes in activations with occlusion of important words are plotted in the following bar graph.

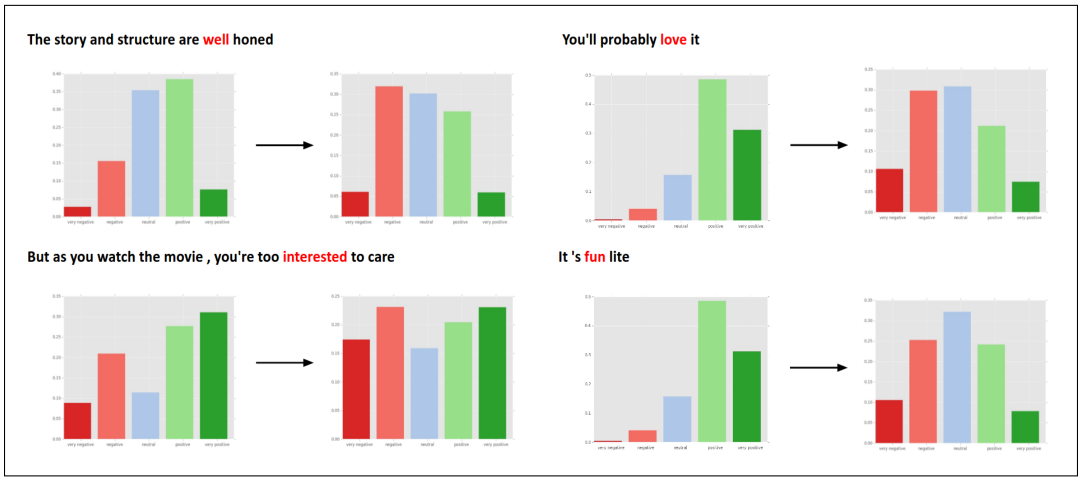


Figure 5: Occlusion visualization

In Deconvolution methods, a one hot vector is propagated back till the input layer, thus projecting it back to the input space and contributions of various words present in the sentence, in determining its class are calculated. For images, contribution by each and every pixel in other words each individual feature of the input feature vector is derived. But in the case of text classification, input passed into the Net is a group of words where each word is a vector as described in data processing section.

Hence in the case of text, instead of each feature, contribution of each words as a whole needs to be determined. The one hot vector is projected back throughout dense layer, such that at each layer, $\hat{x} = w^T(y - b)$ where w,b are weights and biases of that layer.

Three methods are used for calculating contributions of each word of the sentence. First one, 'Deconvnet' uses the projections at the dense layer alone, passed through ReLU in determining the contributions. Second method, 'Backpropagation' considers those projections whose corresponding activations in forward pass are non-negative, in determining the contributions. Third method, 'Guided BackPropagation' combine both the methods mentioned above in determining the contributions of the words. The above three methods are mathematically expressed by the following equations.

$$\begin{aligned} \textbf{Deconvnet \ method}: \quad & c = ReLU(z) \\ \textbf{BackProp \ method}: \quad & c = z \circ ReLU(u) \\ \textbf{GuidedBackProp \ method}: \quad & c = Relu(z) \circ Relu(u) \end{aligned}$$

where $c$ denotes the contributions, $z$ represents the projected vector at the inputs of the dense layer and $u$ represents the output vector at the output of the convolution layer after max-pooling. These contributions are equally divided among those words, that got pooled at the max-pooling layer. Normalized contributions of the words of some sample sentences are plotted in the following bar plots.

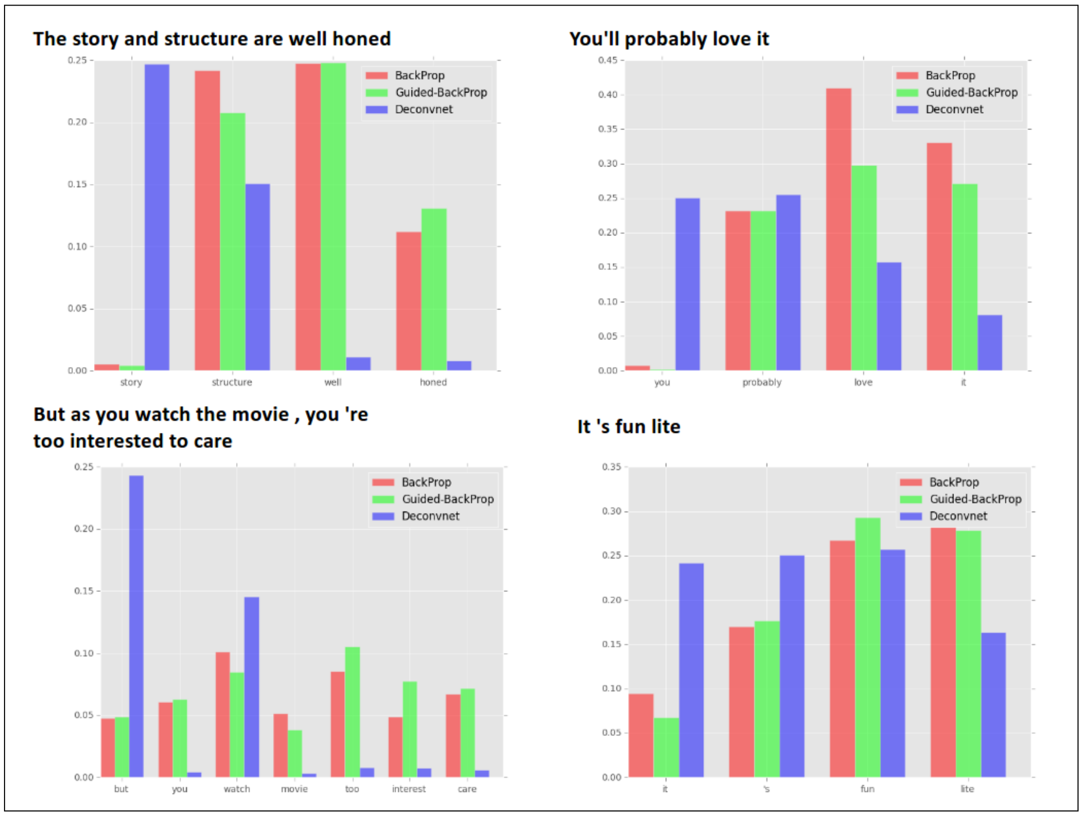


Figure 6: Contribution visualization

## 6 Conclusion

In this project we have done a series of experiments for sentiment classification of movie reviews, using CNN model, with various variants of input features namely random, static, non-static initialization of words. CNN models were trained for all the three models and decent accuracy of approximately 40% was obtained. Other competing models like SVM and LSTM were trained using the same dataset, and results were compared with that of CNN. It was observed that CNN rand and CNN static gave better classification accuracy than traditional SVM model, and LSTM model gave better results than both SVM and CNN. SVM considers a group of maximum three words (trigrams), where as CNN considers upto five consecutive words. Thus CNN has a better advantage in classifying the sentiments on basis of temporal information. LSTM takes into account all the past words along with the present in determining the sentiment of the sentence and hence gave a better accuracy than both SVM and CNN. As far as classification is concerned, one can infer from confusion matrices that CNN is able to give a vague idea of the typical sentiment (i.e. positive or negative), although it fails in determining the extent of the sentiment (very-positive vs positive and very-negative vs negative). Hence we can confidently say that our model atleast gives a vague idea of the sentiment that a sentence carries. Apart from the results, we have also implemented few visualization techniques for understanding CNN among which Occlusion technique and Guided back propagation technique gave convincing visualization results.

## 7 References

[1] Kim, Yoon.*"Convolutional neural networks for sentence classification."* arXiv preprint arXiv:1408.5882 (2014).

[2] Mikolov, Tomas, et al. "Distributed representations of words and phrases and their compositionality." Advances in neural information processing systems. 2013.

[3] N. Kalchbrenner, E. Grefenstette, P. Blunsom. 2014. A Convolutional Neural Network for Modelling Sentences. In Proceedings of ACL 2014.

[4] Deeply Moving: Deep Learning for Sentiment Analysis. (n.d.). Retrieved March 20, 2017, from http://nlp.stanford.edu/sentiment/index.html

[5] Y. Shen, X. He, J. Gao, L. Deng, G. Mesnil. 2014. Learning Semantic Representations Using Convolutional Neural Networks for Web Search. In Proceedings of WWW 2014.

[6] W. Yih, X. He, C. Meek. 2014. Semantic Parsing for Single-Relation Question Answering. In Proceedings of ACL 2014.

[7] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuglu, P. Kuksa. 2011. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research 12:24932537.

[8] (n.d.). Retrieved April 28, 2017, from https://groups.google.com/forum/!topic/word2vec-toolkit/ubFrO0a9Pe8

[9] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011)

[10] Okuda, Teruo, Eiichi Tanaka, and Tamotsu Kasai. "A method for the correction of garbled words based on theYujian, Li, and Liu Bo. "A normalized Levenshtein

[11] Felix Grn, Christian Rupprecht, Nassir Navab, Federico Tombari. "A Taxonomy and Library for Visualizing Learned Features" in Convolutional Neural Networks

[12] Socher, Richard, et al. "Recursive deep models for semantic compositionality over a sentiment treebank." Proceedings of the conference on empirical methods in natural language processing (EMNLP). Vol. 1631. 2013.

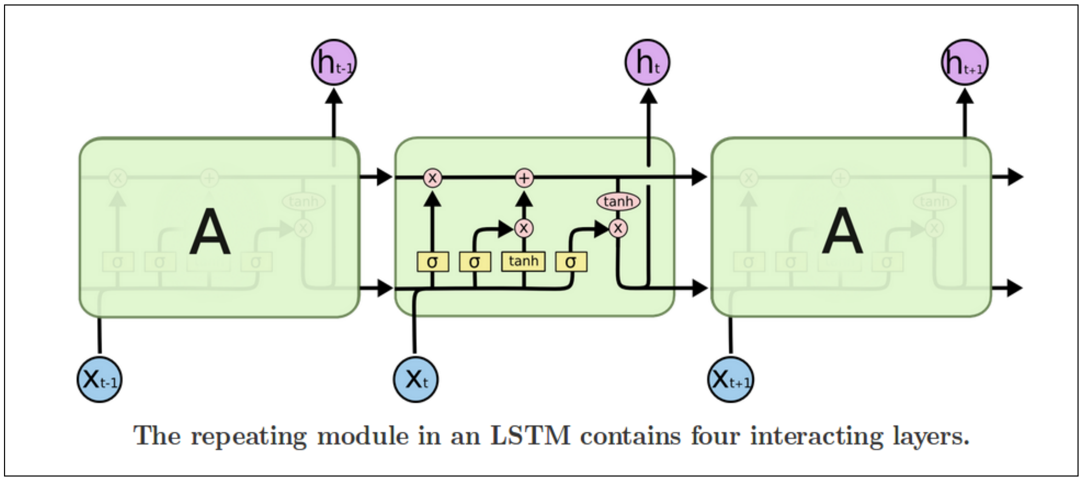
[13] Karen Sparck Jones and Peter Willet, 1997, Readings in Information Retrieval, San Francisco: Morgan Kaufmann, ISBN 1-55860-454-4.

[14] Maas, Andrew L., et al. "Learning word vectors for sentiment analysis." Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1. Association for Computational Linguistics, 2011.

## A LSTM

Neural networks performs greatly for different task related to images like classification, prediction, annotation and many more. One of the assumption of CNN is that the inputs are independent from one another. This may limit their effectiveness when applied to tasks that needs sequential information as in text classification, where the network learns to predict the next character or as in video processing where the input sequence has temporal relation. An Recurrent Neural Network(RNN) as the name suggest has recurrent connections that let a mark or feature of previous inputs persist and affect the evaluation of activation of current neurons, which as a result allows past data to influence the present outputs. This means that recurrent neural networks can learn temporal patterns in data sequences. But they have have a severe limitation of suffering from the vanishing gradient problem, thus, making it difficult to perform tasks with more than few time steps between the target and the inputs. The long short-term memory(LSTM) overcomes this limitations of RNN by adding the ability to selectively remember and forget previous data, and by discriminatingly selecting what should be remembered and what shouldnt, which allow it to work for longer sequences.

LSTM finds and exploits long-term temporal or sequential ordering dependencies in the input data sequences. The key to LSTM is the cell state. LSTM has ability to remove or add information to the cell state, carefully regulated by structures called gates. The LSTM memory cell, utilizes three gates the input, output and forget gate. The gate activations are calculated using a logistic sigmoid function or tanh of the dot product of the input and weights, with each gate containing its own weights. The inputs, outputs, and weights all represent a vector, and element-wise multiplication, addition, and dot operations are used for combining different gates and cell state. The weights are shared by the unit through each time step. The cell state and hidden states are controlled by a input and output gates and forget gate



$h_{t-1}$
$h_t$
$h_{t+1}$
A
tanh
σ
σ
tanh
σ
A
$X_{t-1}$
$X_t$
$X_{t+1}$
The repeating module in an LSTM contains four interacting layers.

Figure 7: Basic LSTM architecture

A regular LSTM learns long temporal relation in data without much difficulty in the following manner.

1. In the first step, what information the LSTM memory cell is going to throw away from the cell state is decided. This is done using ”forget gate layer” which use sigmoid activation of dot product of hidden state and input.
2. Next step, what new information the cell is going to store in the cell state, its done using ”input gate layer”, consists of sigmoid activation on hidden state, which decides which values cell will update.and a tanh activation on hidden state that creates a vector of new candidate values,which could be added to the state. Finally, they are combined to create an update to the cell state.
3. In the final step, LSTM output is found, done by ”output gate layer”. This output is based on cell state, which is done using sigmoid of hidden state and tanh of cell state multiplied together to produce output, which is used as input for next cell state. Sometimes peephole connection in the output state controls the information that is propagated from the previous time step.