IE 7615: Final Project Paper

Convolutional Neural Networks for Text Classification

Abhilash Hemaraj
hemaraj.a@northeastern.edu

Abstract

Convolutional neural networks are great for sequence data and are usually used for tasks involving images. And historically they have been used successfully to great extent on tasks like Image recognition, Video analysis, Anomaly detection, Time series forecasting, etc. This project focuses on the implementation of convolutional neural networks in the field of Natural Language Processing, specifically for sentence classification. I focus on experimentation with and without pre-trained word embeddings, and contrast between architectural changes such as single vs multi-channel CNNs and report on the results obtained.

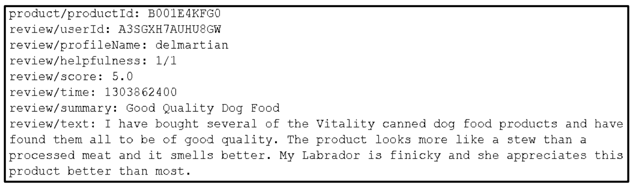
1. Introduction

The goal of this project is to use convolutions, applied directly on the numerical representations of text often referred to as embedding layers, to capture features effectively and to use these features at the end to classify the samples into different classes. If designed properly, convolutions will be able to capture relevant parts that store the essence of what the text is about. To understand more about what we need to classify and to realize the need for convolutions let's look into the Dataset at hand.

1.1 The Dataset

The data set is called Amazon Fine Food Reviews or Amazon-5 [2]. This dataset consists of reviews of fine foods from amazon, accompanied by a rating measure on the scale of 1 to 5. It contains over five hundred thousand rows of reviews. And many of the reviews come with associated metadata like summary of the reviews and other information about the consumer.

The dataset was uploaded in the form of compressed text files. These files are in an unorthodox although structured format (fig.1) storing information on products, reviews, and users. For this project, only Text column is taken as feature and column Score as labels of the dataset. The hope is to be able to predict the rating of the product given the review by the consumer. Since the reviews reflect the consumers experience of the product, the rating should be able to address that. More specifically a negative sentiment in the input text should translate to 1 or 2 on a 5-point scale and a highly positive sentiment should be a perfect 5.



```
product/productId: B001E4KFG0
review/userId: A3SGXH7AUHU8GW
review/profileName: delmartian
review/helpfulness: 1/1
review/score: 5.0
review/time: 1303862400
review/summary: Good Quality Dog Food
review/text: I have bought several of the Vitality canned dog food products and have
found them all to be of good quality. The product looks more like a stew than a
processed meat and it smells better. My Labrador is finicky and she appreciates this
product better than most.
```

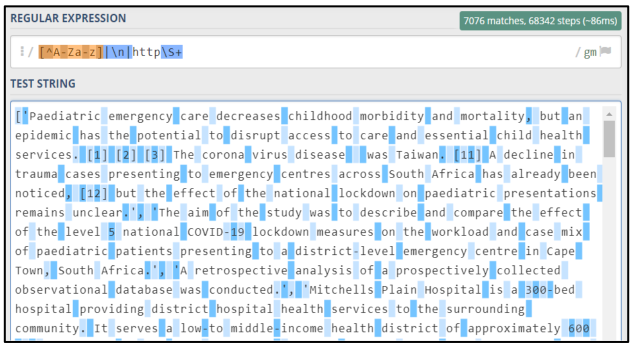
Fig 1: A sample snapshot of the dataset. Above is information on one product review.

1.2 Converting from text to vectors

The body text was converted to a matrix of contextual numerical representations. This has been done in two different ways. The first is to get the pretrained GloVe [3] embeddings and pass it as the weights to the Embedding layer in Keras. The second method is to initialize random embeddings of desired vocabulary depth, and dimensions and train them using back propagation.

2. Text Pre-processing

Regular Expressions: Pre-processing the text corpus to retain only relevant information can be substantial in creating a model that produces better results. To match and replace unwanted characters such as URLs, User ID's, numbers, hyperlinks, and other symbols, regular expressions has proved to be quick and reliable. Below is a snapshot of how the matching works [6].



```
/ [^A-Za-z]|\n|http\S+ / gm
```

```
['Paediatric emergency care decreases childhood morbidity and mortality, but an
epidemic has the potential to disrupt access to care and essential child health
services. [1] [2] [3] The corona virus disease   was Taiwan. [11] A decline in
trauma cases presenting to emergency centres across South Africa has already been
noticed, [12] but the effect of the national lockdown on paediatric presentations
remains unclear.', 'The aim of the study was to describe and compare the effect
of the level 5 national COVID-19 lockdown measures on the workload and case mix
of paediatric patients presenting to a district-level emergency centre in Cape
Town, South Africa.', 'A retrospective analysis of a prospectively collected
observational database was conducted.', 'Mitchells Plain Hospital is a 300-bed
hospital providing district hospital health services to the surrounding
community. It serves a low-to middle-income health district of approximately 600
```

Fig 2: Visual e.g., of pattern matching using regular expressions

2.1 Tokenization and Stemming

This is the process of breaking down a sentence into individual words called tokens. So e.g., a string “I fit a model!” is converted to an array of tokens: [‘I’, ‘fit’, ‘a’, ‘model’, ‘!’]. Stemming is the process of removing the word affixes; so, as the name suggests ‘stem’ a token to its base form. E.g., meet, meeting, meets gets converted to just meet.

2.2 GloVe word vectors

GloVe [3] is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from Wikipedia. It has a total of 6 billion tokens and a vocabulary depth of four hundred thousand words.

3. Related Works

Early works using convolutional based Deep neural networks had shown great potential. There have been recent studies on how deep convolutional neural networks can be applied on the character level [4] and can help in extracting local features. Works detailing how a bag-of-words approach could also be an alternative [5] by preserving word ordering has been made. But at the same time there has been sufficient evidence on how pre-trained word embeddings can be utilized to build better classification systems. Detailing on the branch of transfer learning, work by Kim et al [1], shows how multichannel convolutions with variable kernel sizes can be used to help extract high-level abstract features. [1] also shows how variable kernel sizes are effectively looking at different n-grams. This paper takes inspiration from these techniques for classifying Amazon-5 data [2].

4. Approach

Convolutional neural networks are not the usual go to architectural choice for text-based problems. Mainly due to the advent of LSTMs and recurrence-based models. Although, recurrence-based models are great, they make sense when used in problems when there is sequential information to be learnt. And although when it pertains to language, sequential information is paramount as it concerns grammar and other semantic relations; when it comes to the task at hand which is classifying a product review to a rating label like for e.g., ‘I really liked this cranberry juice’, there is very little need to look into sequential information, when a convolutional feature map could look into the bi-gram really liked to give it a high rating. This and the fact that convolutions are faster to train than other recurrence-based models make CNNs a compelling choice.

Before diving into the architectural changes. The data itself which is majorly text is available in the form of txt file. To convert the dataset into a usable tabular format, I parsed the file using python scripting. Custom regex matching and substitution scripts in python were executed to clear the corpus of un-usable non-alphabetic characters. These include but are not limited to the numbers, hyperlinks, URLs, and other separation clauses. The dataset is now ready to be looked at with an exploratory perspective.

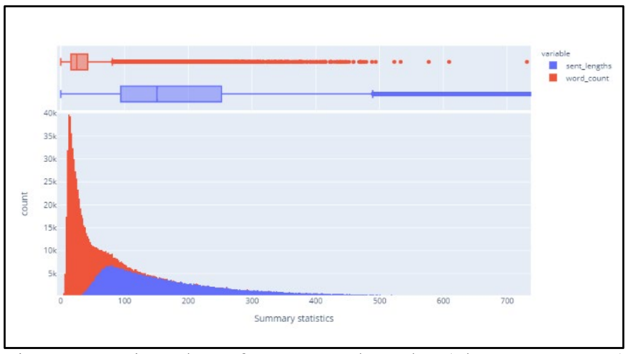


Fig 3: Density plot of sentence lengths (character count) and word counts.

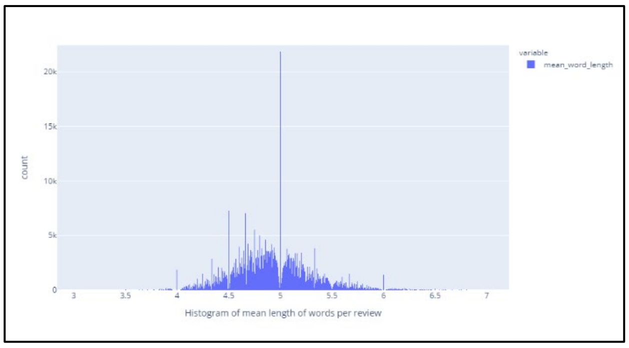


Fig 4: A histogram of the mean length of words per review.

The summary statistics show that majority of the reviews are in fact short (ranging from 10 to 100 characters).

4.1 Building the embedding layer

After Text processing and EDA is the part of using Natural Language Processing Techniques to further process the text contained in body text. NLTK’s tokenization and stemming libraries are leveraged to build a function that can automatically convert each document into tokens. Now the corpus is ready to be converted into word vectors using GloVe embeddings.

For the first phase of experiments, I will focus on pretrained word embeddings. There are three different embedding files in GloVe: Each carry vectors of a different dimension size viz. 100, 200, 300. For the purpose of this paper, I am focusing on 200- and 300-dimensional word vectors. The reason for this being those larger dimensions capture more context of a particular word, and especially for vocabulary of real-world text, more context the better the model will be learning from them. For the second phase of experiments, I will be using randomly initialized weights. These weights are going to be kept as a trainable so that they are updated during training of the neural network.

4.2 Constructing the Convolutional Neural Network

After getting the embeddings, we need a convolutional neural network that can train and learn from input embeddings. Every Convolutional Neural Network consists of these parts:

Convolutions: Layers that consist of different sized kernels that can map and extract features from embeddings. Backpropagation aims to learn the weights of the kernel matrices.

Pooling Layer: Layers that act as an aggregator of all the learnt features. They help in reducing dimensionality of the feature-maps.

Dense Layer: A fully connected feed forward layer required at the downstream level.

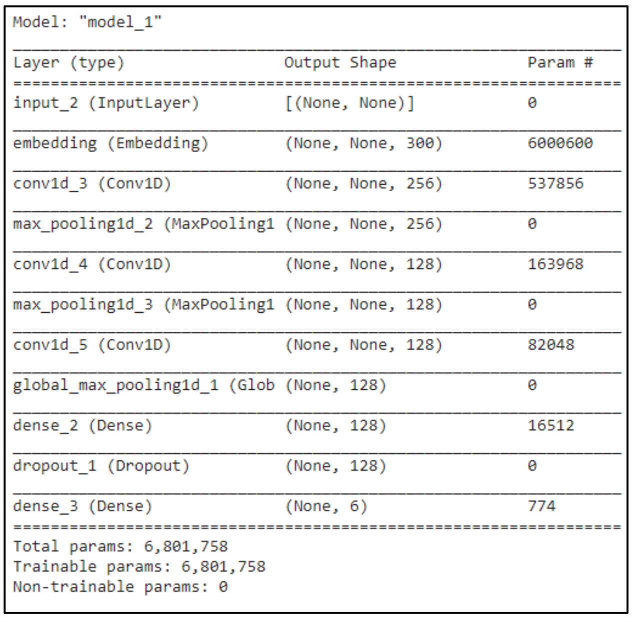
For both the phases of experiments we need to come up with architectural choices that maximizes the overall performance in an efficient way.

4.3 Phase-I Experiment Setup: Pretrained Embeddings

Phase-I is going to address the training of CNNs using the pretrained word vectors. The word vectors are fed into keras vocabulary layer to in turn build a vocabulary-lookup table where the available words are indexed. The text itself will be vectorized to be fed to the model as indices. At the end when the forward pass begins the model queries for an index, which in turn accesses the lookup table providing the embeddings. This is an efficient way of passing these vectors as not all vectors need to be stored in memory.

For the convolutional layer itself, we will be using only one-dimensional convolutions as embeddings are list of vectors and the filters only need to travel in one dimension that is time. The dataset at hand shows that majority of the reviews sit between 0 to 100 words. This is important information because this informs us to build our kernel of appropriate size. A kernel size of 3 would mean the filter will look at trigrams and a size of 2 will mean bigrams.

After much experimentation I converged upon a model using three convolutional layers with filters of size 256, 128 and 128. These layers stacked on top of each other with Max-pooling layers in between them. Output of the convolutions are fed to the Dense layer of 128 units that in turn feeds into the prediction layer. See Fig.5 for more details.



```
Model: "model_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_2 (InputLayer)         [(None, None)]            0         
_________________________________________________________________
embedding (Embedding)        (None, None, 300)         6000600   
_________________________________________________________________
conv1d_3 (Conv1D)            (None, None, 256)         537856    
_________________________________________________________________
max_pooling1d_2 (MaxPooling1 (None, None, 256)         0         
_________________________________________________________________
conv1d_4 (Conv1D)            (None, None, 128)         163968    
_________________________________________________________________
max_pooling1d_3 (MaxPooling1 (None, None, 128)         0         
_________________________________________________________________
conv1d_5 (Conv1D)            (None, None, 128)         82048     
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 128)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 128)               16512     
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 6)                 774       
=================================================================
Total params: 6,801,758
Trainable params: 6,801,758
Non-trainable params: 0
```

Fig 5: Model Summary: Convolutional Neural network with Pre-trained GloVe embeddings (300 dimensional).

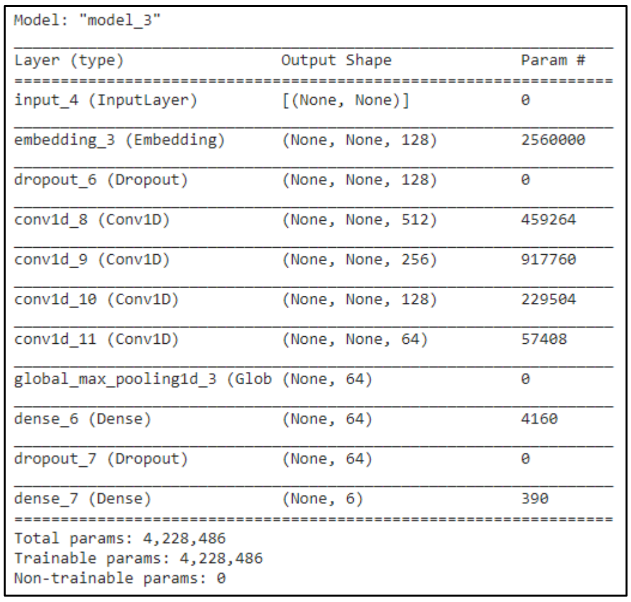
This model was repeated twice, the first time with 200 dimensional embeddings and the second time with 300 dimensional embeddings by keeping the rest of the network the same. The activation functions used in all the layers except the last layer is ReLu stands for rectified linear unit. The last layer employs a softmax function to output probabilities of the classes. The model was then compiled using a sparse-binary-cross-entropy loss (since this is a multi-class classification problem), an Adam optimizer with learning rate as 0.001 and Accuracy metric.

4.4 Phase-II Experiment Setup: Without Pretrained Embeddings

Phase-II is going to address the training of CNNs without using the pretrained embeddings. Instead, the embeddings are to be learnt via backpropagation. This part of the experiment also contains a comparison of single and multi-channel CNN's. The single channel model thus designed consists of four different convolutional layers going from 512 filters to 64 filters of varying stride lengths and kernel sizes. This model has been equipped with two types of activation functions to be tested iteratively. The first being ReLu, and the second being Swish activation function. Swish is a relatively newer activation and is a close relative to ReLu. Swish can be parametrized by the following function:

$$\boldsymbol{f(x) = x \cdot \sigma(x),}$$

$$\text{where, } \boldsymbol{\sigma(x) = (1 + exp(-x)) - 1}$$



```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_4 (InputLayer)         [(None, None)]            0
_________________________________________________________________
embedding_3 (Embedding)      (None, None, 128)         2560000
_________________________________________________________________
dropout_6 (Dropout)          (None, None, 128)         0
_________________________________________________________________
conv1d_8 (Conv1D)            (None, None, 512)         459264
_________________________________________________________________
conv1d_9 (Conv1D)            (None, None, 256)         917760
_________________________________________________________________
conv1d_10 (Conv1D)           (None, None, 128)         229504
_________________________________________________________________
conv1d_11 (Conv1D)           (None, None, 64)          57408
_________________________________________________________________
global_max_pooling1d_3 (Glob (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_7 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 6)                 390
=================================================================
Total params: 4,228,486
Trainable params: 4,228,486
Non-trainable params: 0
```

Fig 6: Model Summary: Single Channel Convolutional Neural network without pre-trained embeddings (128 dimensional).

As seen from the model summary the total parameters of this model are significantly less than the one in Fig.5; this is because of the fact that the embedding dimension chosen here is 128, that is every token will be represented by 128-dimensional vector.

For the multichannel CNN model, three convolutions of size 128 is used with a max pooling layer on each of the three channels. Each convolutional layer will look at 7 words at once with a stride of 3. The output is concatenated and fed as dense layer for prediction. (Fig.7).

5. Results

After building the embeddings index and constructing the models for each phase, I ran the respective experiments.

Phase-I Experimental Results:

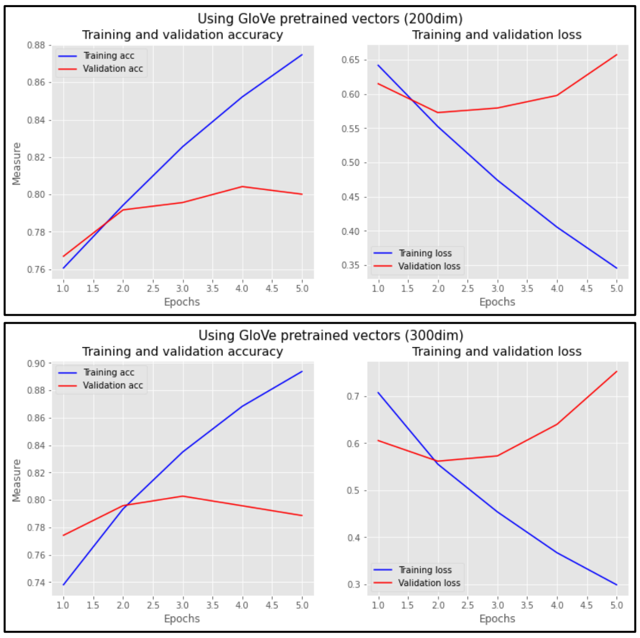


Fig 8: Training for 5 epochs; top plot shows the Accuracy and loss metrics for Model trained on GloVe embeddings of 200 dimensions. Bottom plot shows the same for 300 dimensional embeddings.

As evident from the plots; Convolutional neural networks are great at learning from the embeddings to predict ratings. This also poses as a problem since continual training can lead to over-fitting.
We can observe from Fig 8 that after a certain point there is no improvement in the performance of these models; for e.g., in the top plot after the fourth epoch the model performance deteriorates going from peak accuracy of 0.805 to 0.80 and validation loss dramatically increasing from 0.6 to 0.65; all of which are clear signs of over-fitting. It is also to be noted that, very small changes to the number of feature-maps, kernel-size and even stride-length had huge effect on model performance. Very low number of filters fails to capture enough information to train the model, so the model suffers from underfitting. Very low kernel-size and stride meant the model would miss import word associations and thus lose on semantics.

Overall GloVe embeddings prove to be great initializers for our embedding layers since keeping it non-trainable would only deteriorate performance over time as the model saw more out of vocabulary words down the training batches. The training batch size of 64 ensures the machine doesn't run out of memory.

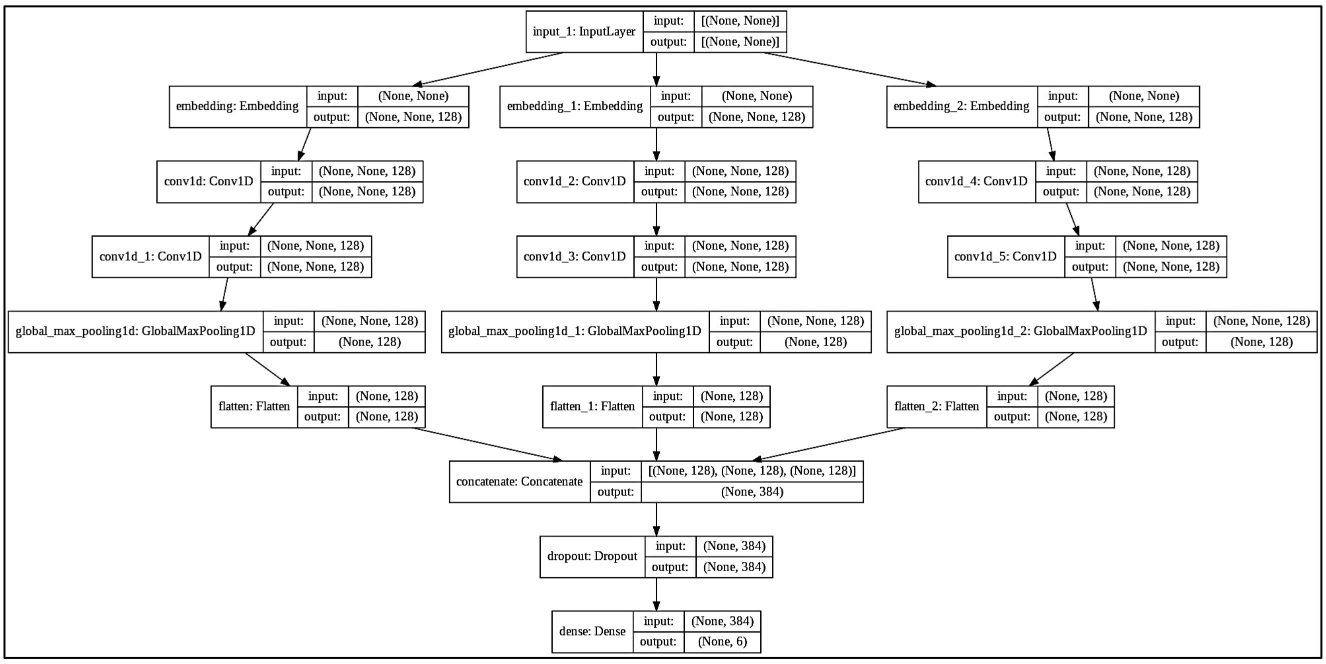

input_1: InputLayer | input: [(None, None)] | output: [(None, None)]
embedding: Embedding | input: (None, None) | output: (None, None, 128)
embedding_1: Embedding | input: (None, None) | output: (None, None, 128)
embedding_2: Embedding | input: (None, None) | output: (None, None, 128)
conv1d: Conv1D | input: (None, None, 128) | output: (None, None, 128)
conv1d_2: Conv1D | input: (None, None, 128) | output: (None, None, 128)
conv1d_4: Conv1D | input: (None, None, 128) | output: (None, None, 128)
conv1d_1: Conv1D | input: (None, None, 128) | output: (None, None, 128)
conv1d_3: Conv1D | input: (None, None, 128) | output: (None, None, 128)
conv1d_5: Conv1D | input: (None, None, 128) | output: (None, None, 128)
global_max_pooling1d: GlobalMaxPooling1D | input: (None, None, 128) | output: (None, 128)
global_max_pooling1d_1: GlobalMaxPooling1D | input: (None, None, 128) | output: (None, 128)
global_max_pooling1d_2: GlobalMaxPooling1D | input: (None, None, 128) | output: (None, 128)
flatten: Flatten | input: (None, 128) | output: (None, 128)
flatten_1: Flatten | input: (None, 128) | output: (None, 128)
flatten_2: Flatten | input: (None, 128) | output: (None, 128)
concatenate: Concatenate | input: [(None, 128), (None, 128), (None, 128)] | output: (None, 384)
dropout: Dropout | input: (None, 384) | output: (None, 384)
dense: Dense | input: (None, 384) | output: (None, 6)

Fig 7: Model Plot: Three Channel Convolutional Neural network without pre-trained embeddings (128 dimensional). The total parameters of the model: 8,371,206

Phase-II Experimental Results: Without pretrained embeddings:

There are number of factors to be considered when choosing whether to take advantage of pretrained word embeddings. There are many advantages to that like getting higher performance depending on the task, faster convergence due to transfer learning and they are generally more reliable. But these come with their own drawbacks, like slower run times, quality of the vectors depends on the corpus it was trained on, being highly task and domain dependent, to name a few. Training your own embeddings from the corpus lead to highly task specific vectors and maybe even comparable to their pretrained counterparts.

Training using a single channel convolutional neural network with randomly initialized weights ran way faster than phase-I. I observed a 2.5-fold increase in run times per epoch.

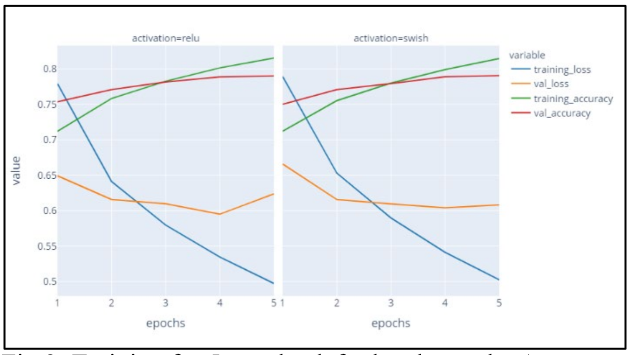


Fig 9: Training for 5 epochs; left plot shows the Accuracy and loss metrics for usage of ReLu activation and the right one for Swish.

As it is evident from the performance curves that there is no comparable difference between ReLu and Swish. This is partly due to the fact that both are very similar activation functions. And for the task at hand both happen to be a good choice in activation functions. Also, another observation would be that the model is more robust and resilient to overfitting and a very similar peak accuracy as you can see a clear difference in phase-I and phase-II single channel experiments.

For the Multi-channel CNNs due to the limitation of computational resources, the model is only run for 2 epochs. Even so, the model achieved a peak accuracy score of 80% with a validation loss of 0.597. The need for multi-channel CNN's is to learn different word semantics and associations that can better help us predict the ratings. Essentially if one channel overfits, the other could act like a regularizer since all the outputs are concatenated at the end.

These were the results for all the models executed:

Model	Channels	Epochs	Training Time per epoch	Validation Accuracy
CNN + GloVe(20	Single	5	500s	0.802

0d)				
CNN + GloVe(300d)	Single	5	700s	0.8028
CNN + Random Embeddings (ReLu, 128d)	Single	5	285s	0.7903
CNN + Random Embeddings (Swish, 128d)	Single	5	300s	0.7906
CNN + Random Embeddings (128d)	Multi	2	5223s	0.80
CNN + Random Embeddings (300d)	Multi	2	13585s	0.8001

6. Conclusion and future scope

Convolutions have been great for images, but more and more studies have shown that they are great for text-based problems as well. They are particularly well in capturing localized word semantics and relationships paving way for downstream NLP task like sentence classification. This paper looks rigorously into various CNN architectures applied specifically for Text and found that:

1. Transfer Learning has some great advantages, but quality of the vectors depends on the textual domain it was pre-trained on.
2. Convolutions are sensitive to even minor tuning of hyperparameters
3. Multi-channel CNN's are fruitful only if subject to extensive experimentation and testing. Since the cost to performance ratio is high.
4. Trainable yet randomly initialized word vectors prove to be just as effective.
5. And lastly modelling text with no controlled vocabulary would require more work into the choice of architectures and the amount of resources used.

7. References

[1] Kim, Y. (2014), Convolutional Neural Networks for Sentence Classification, in 'Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL' , pp. 1746--1751 .

[2] McAuley, J.J. and Leskovec, J., 2013, May. From amateurs to connoisseurs: modeling the evolution of user expertise through online reviews. In Proceedings of the 22nd international conference on World Wide Web (pp. 897-908).

[3] Pennington, J., Socher, R. and Manning, C.D., 2014, October. Glove: Global vectors for word representation. In Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP) (pp. 1532-1543).

[4] Cicero dos Santos and Maira Gatti. 2014. Deep convolutional neural networks for sentiment analysis of short texts. In Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers. 69–78

[5] Rie Johnson and Tong Zhang. 2014. Effective use of word order for text categorization with convolutional neural networks. arXiv preprint arXiv:1412.1058 (2014).

[6] https://regex101.com/

8. Acknowledgments

I thank Professor Jerome Braun, class lecturer for Neural Networks and Deep Learning (IE7615), for his constant support and guidance throughout the course and the project.