Deep (Learning) Survey on Text Classification

1 Abstract

Natural Language Processing (NLP) tasks, such as part-of-speech tagging, chunking, named entity recognition, and text classification, have been subject to a tremendous amount of research over the last few decades. Text Classification has been the most competed NLP task in Kaggle Competitions, and other similar competitions. Count based models are being phased out, while new deep learning models are emerging almost every month. This project surveys a range of neural based models for text classification task. Models selected, based on CNN and RNN, are explained with code (keras and tensorflow) and block diagrams. The models are evaluated using one active kaggle competition medical datasets [1].

2 Table of Contents

3 Introduction

Of late, there has been a plethora of neural models for various NLP tasks. Recently, state-of-the-art text classification models such as Support Vector Machines have yielded to new neural models emerging almost every month. This project looks into just one task Text Classification.

Since the inception of word vectors, they are omnipresent in the NLP community. There has been a major shift from using n-grams (count) based models to word vectors based models. The latter models are able to capture semantic and syntactic information without much preprocessing, which played a part in obtaining competitive performance in the former models. Recurrent Neural Networks and Convolutional Neural Networks are responsible for major breakthroughs in computer vision, since then they have been adopted in NLP tasks and have proven effective in many tasks such as Part of Speech Tagger, Named Entity Recognition, and more. This project aims at providing a survey on most of the models considered for text classification in chronological order. This cannot be considered a comprehensive study on all the neural methods available, but can be used as a reference.

As a part of this project, the data used for evaluating all the models are taken from an active competition Personalized Medicine: Redefining Cancer Treatment [1] launched by Kaggle along with Memorial Sloan Kettering Cancer Center (MSKCC). This has been accepted by the NIPS 2017 Competition Track, because they need data scientists to help take personalized medicine to its full potential. Once sequenced, a cancer tumor can have thousands of genetic mutations. But the challenge is distinguishing the mutations that contribute to tumor growth (drivers) from the neutral mutations (passengers). Currently, this interpretation of genetic mutations is being done manually. This is a very time-consuming task whereby a clinical pathologist has to manually review and classify every single genetic mutation based on evidence from text-based clinical literature. For this competition, MSKCC is making available an expert-annotated knowledge base where world-class researchers and oncologists have manually annotated thousands of mutations. One needs to develop Machine Learning algorithms that, using this knowledge base as a baseline, automatically classifies genetic variations.

Deep learning in medicine has been applied in a variety of applications such as image-based assessments of traumatic brain injuries, identifying diseases from ordinary radiology image data, visualizing and quantifying heart flow in the body using any MRI machine, as well as analyzing medical images to identify tumors, nearly invisible fractures, and other medical conditions. A lot has been said during the past several years about how precision medicine and, more concretely, genetic testing is going to disrupt the way diseases like cancer are treated. However, this is only partially happening due to the large amount of manual work still required.

4 Datasets and Evaluation Metrics

Models must be developed to classify genetic mutations based on clinical evidence (text). There are nine different classes on which to classify a genetic mutation. This is not a simple task since interpreting clinical evidence is challenging even for specialists. Therefore, modeling the clinical evidence (text) will be critical to success. Both training and test data sets are provided via two different files. One (training/test_variants) provides the information about the genetic mutations, whereas the other (training/test_text) provides the clinical evidence (text) that our human experts used to classify the genetic mutations. Both are linked via the ID field. Therefore, the genetic mutation (row) with ID=15 in the file training_variants was classified using the clinical evidence (text) from the row with ID=15 in the file training_text. Some of the test data is machine-generated. All the results of the classification algorithm must be submitted, and the machine-generated samples will be ignored. The following provides the file descriptions.

1. training_variants - a comma separated file containing the description of the genetic mutations used for training. Fields are ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the amino acid change for this mutations), Class (1-9 the class this genetic mutation has been classified on)
2. training_text - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)
3. test_variants - a comma separated file containing the description of the genetic mutations used for training. Fields are ID (the id of the row used to link the mutation to the clinical evidence), Gene (the gene where this genetic mutation is located), Variation (the amino acid change for this mutations)
4. test_text - a double pipe (||) delimited file that contains the clinical evidence (text) used to classify genetic mutations. Fields are ID (the id of the row used to link the clinical evidence to the genetic mutation), Text (the clinical evidence used to classify the genetic mutation)
5. submissionSample - a sample submission file in the correct format

Data Snippets: Train Data Frame

	ID	Gene	Variation	Text
0	0	ACSL4	R570S	2. This mutation resulted in a myeloproliferat...
1	1	NAGLU	P521L	Abstract The Large Tumor Suppressor 1 (LATS1)...
2	2	PAH	L333F	Vascular endothelial growth factor receptor (V...
3	3	ING1	A148D	Inflammatory myofibroblastic tumor (IMT) is a ...
4	4	TMEM216	G77A	Abstract Retinoblastoma is a pediatric retina...

Data Snippets: Test Data Frame

	ID	Gene	Variation	Text
0	0	ACSL4	R570S	2. This mutation resulted in a myeloproliferat...
1	1	NAGLU	P521L	Abstract The Large Tumor Suppressor 1 (LATS1)...
2	2	PAH	L333F	Vascular endothelial growth factor receptor (V...
3	3	ING1	A148D	Inflammatory myofibroblastic tumor (IMT) is a ...
4	4	TMEM216	G77A	Abstract Retinoblastoma is a pediatric retina...

Text attribute is considered to be the main text component on which the inference is performed. These statistics will be used to asses what can be considered an average length when modelling with different forms of documents in terms of words. Some statistics on text being words are explored.

Statistics on Words with Document as Words	**Train Data Frame Text Attribute**	**Train Data Frame Text Attribute**
Mean	1485.407709	1633.366090
Standard Dev	845.216131	484.593756
Min	1.000000	1.000000
25th Percentile	919.000000	1337.000000
50th Percentile	1215.000000	1642.000000
75th Percentile	1887.000000	1920.250000
Max	7199.000000	6633.000000

4.1 Evaluation Metrics

Submissions are evaluated on Multi Class Log Loss between the predicted probability and the observed target. Multi Class Log Loss is the multi-class version of the Logarithmic Loss metric. Each observation is in one class and for each observation, a predicted probability for each class will be submitted. The metric is negative the log likelihood of the model that says each test observation is chosen independently from a distribution that places the submitted probability mass on the corresponding class for each observation.

$$logloss = -\frac{1}{N}\sum_{i}^{N}\sum_{j}^{M} y_{i,j} log(p_{i,j})$$

where N is the number of observations, M is the number of class labels, log is the natural logarithm, $y_{i,j}$ is 1 if observation i is in class j and 0 otherwise, and $p_{i,j}$ is the predicted probability that observation i is in class j.

Both the solution file and the submission file are CSV's where each row corresponds to one observation and each column corresponds to a class. The solution has 1's and 0's (exactly one "1" in each row), while the submission consists of predicted probabilities.

The submitted probabilities need not sum to 1, because they will be rescaled (each is divided by the sum) so that they do before evaluation.

5 Data Preprocessing

The foremost step in any text preprocessing is to decide on what character set the models will be working [2]. Since the data set is in English, character set “utf-8” is used to decode the text and special characters, not in selected unicode character set, are ignored. Also, the document contains only printable English characters; all others are removed. Looking at sample random documents, one can notice “figure”, “table”, “supplementary figure” and various iterations of these similar texts present throughout the data. All the texts are made lowercase for disambiguation.

Since in this project we will be generating/training word vectors based on skip gram and cbow models (will be discussed later) on the corpus, these unwanted texts contributing to the context words degrades the semantic relations. Hence, these texts will be removed using custom regex parsers. The hyperlink texts are not removed intentionally, which may provide some relation to the document.

After the above steps, the vocabulary is decided upon by dividing the text into words using the custom regular expression “RegexpTokenizer(r'\w+[-]*\w*')”, which keeps the hyphenated words together since they may carry more information in medical texts. The delimiters will be space characters (tab, single or more space, newline). This will be the vocabulary used for all the models trained. For tokenizing a document into sentences there are multiple options available; here NLTK’s were chosen [3] (leading platform for building Python programs to work with human language data and contains wrappers for industrial-strength NLP libraries) as recommended sentence tokenizers (currently PunktSentenceTokenizer for the specified language).

For count based models, pre-processed text will be used. However, for neural models we need the ability to pass in the text in different forms. A DocumentGenerator Class helps in creating datasets as necessary in the following formats:

1. Document as Words: all document text as list of words
2. Document as Characters: all document text list of characters
3. Document as Sentence as Words: all document text as list of sentences that contain list of words
4. Document as Sentences as Characters: all document text as list of sentences that contain characters
5. Document as Sentences as Words as Characters: all document text as list of sentences that contain words represented as list characters

There are two data structures for maintaining vocabulary. One contains the list of vocabulary words and the other is a dictionary that maps the words to their position in the former. A corpus text is also formed with cleaned data that will be used for training word vectors.

6 Count Based Classification

There are multiple n-gram count based classification models. This project uses these models as baseline models to assess whether the deep models are performing better. Bag of words model represents the text data by assigning each word an id and these ids will be used as features for the documents. These features represent number of occurrences of the word in the document. This can lead to issue of having too many features which could be handled with smaller vocabulary. The words not in the vocabulary will be replaced with a common id that does not occur in the vocabulary. This will result in very high dimensional sparse dataset. To scale down the impact of words with high frequency and dealing with differences for smaller and longer documents differently, we will be use Term Frequencies times Inverse Document Frequencies(tf-idf) with smoothing. There are many weighting schemes for using tf-idf, the one used in this project is of the form shown below

$$tf - idf(d,t) = tf(t) * idf(d,t)$$
$$idf(d,t) = log(\frac{n}{df(d,t)}) + 1$$

Where n is total number of words in vocabulary, d is the document and \$t\$ is the word in the document. $tf(t)$ represents the term frequency in the whole corpus. $df(d,t)$ represents the term frequency in the specific document.

After converting each document to the required tf-idf matrix, we use sklearn package to exercise all the necessary processing and functions to run the following machine learning algorithms:

1. Multinomial Naive Bayes classifier
2. Support Vector Machines
3. Softmax (Multi Class Logistic) Regression
4. K Nearest Neighbors
5. Passive Aggressive Classifier
6. Decision Trees
7. Adaboost Classifier
8. Extreme Randomization Trees
9. Random Forest Classifier

Catboost classifier from Yandex Technologies is also run on the dataset. Some models like sklearn's Gaussian Mixture Models and Xgboost were not considered since either they took more than 64GB of RAM or ran for more than 6 hours. Grid/Random searches are also not used on the above models considering the time factor. For all the runs, the vocabulary is not reduced in size since unique medical terms like gene names may carry more information.

7 Deep Learning Methods

Natural Language Processing has complicated sequential and hierarchical structures Both CNN and RNN play important roles in bringing those features out for better performance in sentence classification. The paper is organized in the following order: first we will discuss word vectors, document vectors, convolutional neural networks and recurrent neural networks. Then we will discuss text classification models based solely on CNN, followed but models based on RNN and finally models based on a combination of CNN and RNN. All the models discussed in this project will have block diagrams represented in the paper and model summary that represents layers in Keras. If tensorflow is used to form the models, then tensor board graph block summary will be displayed. Also, all the models need an embedding layer, which makes it easier for choosing word vectors based on vocabulary IDs. The test results will be discussed at the end of this paper.

7.1 Background

7.1.1 Word Vectors

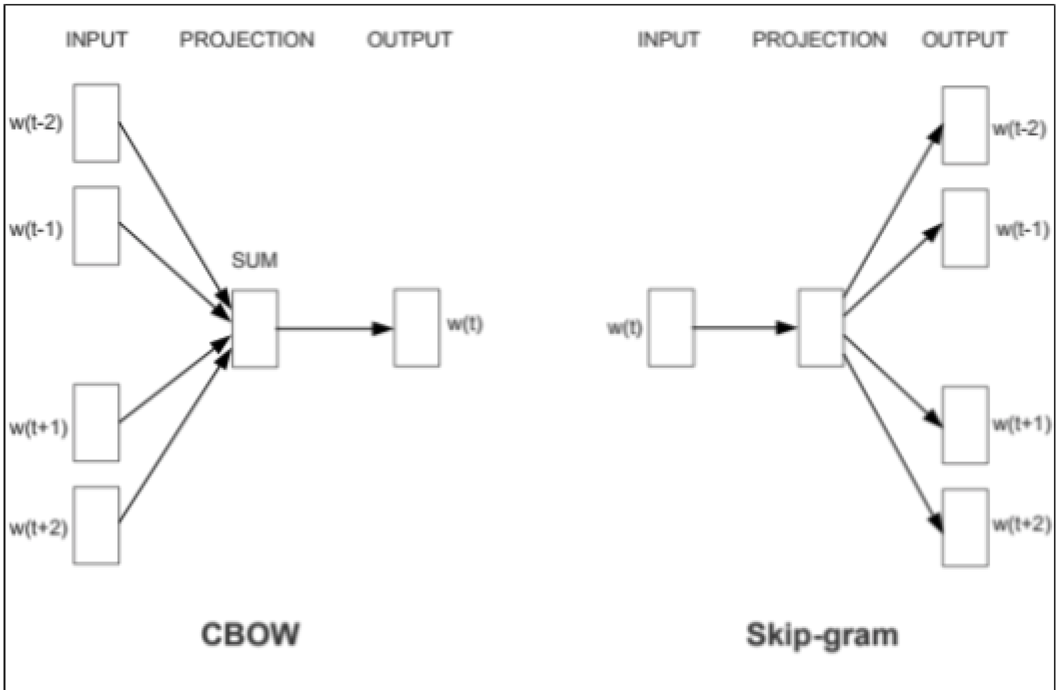
Models proposed for estimating continuous representation of words, such as Latent Semantic Analysis/Indexing and Latent Dirichlet Allocation, use documents for contextualizing information retrieval and are computationally expensive when used on large datasets. Recent models use words as contexts, which captures semantic similarity rather than semantic relatedness. Word embeddings created using unsupervised training on a corpus provide meaningful semantic similarity such as:

$$king - man + woman \approx queen$$

Word embeddings are low dimensional representations of words in vector space. They are also referred to as word vectors or distributed representation of words. Feed forward neural models such as the Continuous Skip-gram and the Continuous Bag of Words models introduced by Mikolov et al (2013) [4], provide better semantic and syntactic word similarities than count based models, with reduced computational cost.

The training methods and estimation of the loss function are beyond the scope of this project, but a quality reference can be found in Sebastian Ruder's blog post on word embeddings [5].

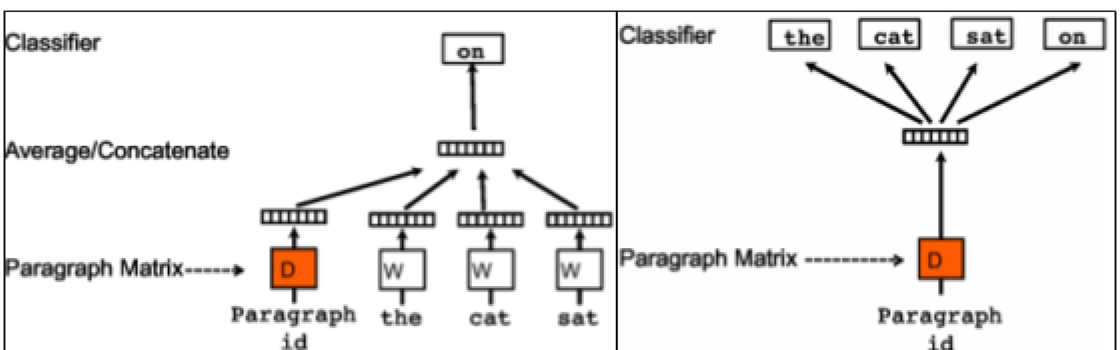
Joulin et al (2016) [6] [7] introduced the Fasttext [8] tool for word representation learning and text classification. This tool will be used to generate word vectors that form the basis for most of the deep models in which each the units considered are words. Multiple set of word vectors will be generated with varying dimensions (100,200,300), epochs (20, 50, 100) and models (skip gram and cbow). Sets of vectors generated from NLTK word2vec (trained by google), Glove [9] vectors from Stanford, and bionlp [10] word vectors based on medical text will also be used.



Mikolov et al (2013) The CBOW architecture predicts the current word based on the context, and the Skip-gram predicts surrounding words given the current word.

7.1.2 Document Vectors

For representing a document (sentence or paragraph), bag of words models ignore word ordering and the word semantics, which is important for document analysis. Paragraph Vectors introduced by Mikolov et al (2014) [11] (doc2vec as called in NLTK) take advantage of these features by forming a feed forward neural network and performing unsupervised training in approximating low dimensional representation for documents in unlabelled corpora.



Mikolov et al (2014)
Distributed Memory Model of Paragraph Vectors

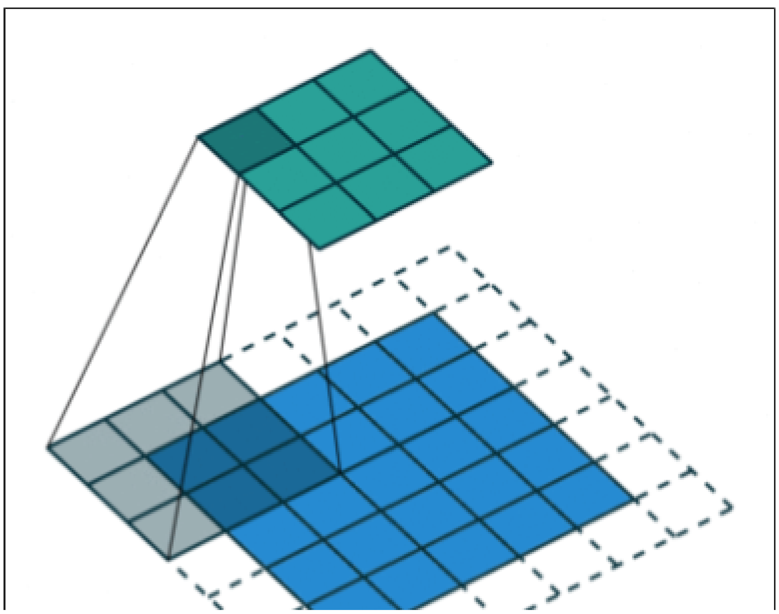
Distributed Bag of Words

This project does not include models created based on the document/paragraph vectors.

7.1.3 Convolutional Neural Networks

Convolutions (convs) emerged in the field of computer vision that aggregate information over certain area and present important features for further processing. Usually convolutional layers go through 3 stages [12]:

1. Convolution Stage: Parallely perform several convolutions to produce a set of linear activations. Here in NLP, 2D convolutions do not work since words do not have channel dimension like images (but sometimes 2D convs are necessary to perform some complex operations refer RA-CNN). The convolutions are performed using simple hadamard products against filters that are moved along the time dimension (in NLP, time dimensions is the dimension along which the words in a document/sentence run). We will be using *Conv1D* layer from Keras and *tf.nn.conv1d* from tensorflow. The dimension of the kernel/filter used will be same as the word or character vectors (sometimes a unit of representation can be a character). They are then moved along the document based on given strides. Paddings are sometimes necessary based on output dimension requirements.
2. Detector Stage: Linear activations are run through nonlinear functions, such as a rectified linear unit to represent complex functions. Keras *Activation* Layer can be used to achieve this stage.
3. Pooling Stage: A pooling function replaces the output of the net at a certain location with summary statistics of nearby inputs. They are used to capture translations that are invariant. Different kinds of pooling can be performed such as max pooling (returns 1 value), average pooling, k-max pooling (pooling returning k max values), and others.



To elaborate on the details referenced here, chapter 9 from book [12] and chapter 13 from book [13] CNNs are recommend, as they model both short range and long range dependencies via hierarchical structures. CNNs with fixed and variable size filters/kernels essentially split documents into sub-documents by a sliding window and pick out the dominant feature for further processing. Models based on CNN can be considered unbiased models since with a maximum pooling layer they can determine discriminative phrases in a document better than other models by giving equal importance to all words.

7.1.4 Recurrent Neural Networks

Feed forward neural networks are excellent at learning patterns from a fixed input matrix. If the input is a sequence such as a sentence or series of images or stock data, then they do not have enough power to represent the sequence pattern. For NLP models that have natural time dimension, recurrent neural networks best suited to learn patterns by using past information to help evaluate the sentence semantics.

Each unit takes input from the previous units and propagates information. All the units using the same weight matrices helps approximate the correlations in a series or sequence. Normal recurrent units have vanishing gradient problems that occur during training with backpropagation through time. To combat this, several types of recurrent units were introduced. One of the most used and most stable types are GRU (gated recurrent units) and LSTM (Long Short Term Memory). Let's look into LSTM equations and block diagrams.

$$f_t = \sigma(W_f.[h_{t-1}, x_t] + b_f)$$
$$i_t = \sigma(W_i.[h_{t-1}, x_t] + b_i)$$
$$\widehat{C}_t = \tanh(W_c.[h_{t-1}, x_t] + b_c)$$
$$C_t = f_t.C_{t-1} + i_t.\widehat{C}_t$$

The above equations and the block diagram show that other than hidden weights, there is also a cell weight that acts as a memory. In each sequence, the model can make decisions on how much memory to propagate through the sequence to model the dataset. There are other versions of RNN such as GRU, and LSTM with different 'peepholes'. Again, this is just a peak in the workings of a recurrent unit. Chapter 10 of book [12] and chapter 14 of book [13] are recommended. The images are taken from Colah's blog on understanding LSTM.

RNNs have proven to be powerful sequence density estimators that can use long context to make predictions using conditional sequence modeling. Models based on RNN can be considered biased models since later words might be given more importance in trying to capture the semantics of the whole document.

7.1.5 Regularization and Training

To prevent the model from overfitting the provided dataset and help with generalization, regularization plays an important role. Regularization can be performed in many different forms. In this project, the three types of regularizations most often used will be:

1. L2 norms, which help penalize larger model weights;
2. Batch Normalization, which helps with maintaining the internal covariance shift, and the ability to adapt to changing parameter distribution; and
3. Dropout layer, which helps reduce overfitting by randomly turning on/off the neurons and can be considered similar to training many models in reducing the variance overall.

When performing model training we mostly use ADAM optimizer for Stochastic Gradient Descent. Deeper discussion on regularization and training methods is beyond the scope of this project. Consider reading chapter 11 from book [13]

7.2 Models Based on Convolutional Neural Networks

All the models presented in this section use ideas based on just CNN. They are also modified to take into consideration other features such as gene names and variation words. These features are presented as either words or characters based on the requirements of the model.

7.2.1 CNN for Sentence Classification

Yoon et al (2014) [14] proposed CNN models in addition to pre-trained word vectors, which achieved excellent results on multiple benchmarks. The model architecture as shown in the figure below maintains multiple channels of input such as different types of pre-trained vectors or vectors that are kept static during training. Then they are convolved with different kernels/filters to create sets of features that are then max pooled. These features form a penultimate layer and are passed to a fully connected softmax layer whose output is the probability distribution over labels.

The paper presents several variants of the model:

1. CNN-rand (a baseline model with randomly initialized word vectors)
2. CNN-static (model with pre-trained word vectors)
3. CNN-non-static (same as above but pre-trained fine tuned)
4. CNN-multichannel (model with 2 sets of word vectors)

Keras model summary and code for CNN-non static can be found here.

7.2.2 DCNN for Modelling Sentences

Kalchbrenner et al (2014) [15] presented Dynamic Convolutional Neural Network for semantic modelling of sentences. This model handles sentences of varying length and uses dynamic k-max pooling over linear sequences. This helps the model induce a feature graph that is capable of capturing short and long range relations. K-max pooling, different from local max pooling, outputs k-max values from the necessary dimension of the previous convolutional layer. For smooth extraction of higher order features, this paper introduces Dynamic k-max pooling where the k in the k-max pooling operation is a function of the length of the input sentences.

$$k = max(k_{top}, \lfloor \frac{L-l}{L}.s \rfloor)$$

where l is the num,ber of convolutional layer to which pooling is applied, L is total number of convolutional layers, k_{top} is the fixed pooling parameter of the top most convolutional layer. The model also has a folding layer which sums over every two rows in the feature-map component wise. This folding operation is valid since feature detectors in different rows are independent before the fully connected layers.

Wide convolutions are preferred for the model instead of narrow convolutions. This is achieved in the code using appropriate zero padding.

This network model's performance is related to its ability to capture the word and n-gram order in the sentences and to tell the relative position of the most relevant n-grams. The model also has the advantage of inducing a connected, directed acyclic graph with weighted edges and a root node as shown below.

Folding and K-max pooling layers are not readily available and have to be created using keras functional apis.

Keras model summary and code for DCNN can be found here.

7.2.3 Medical CNN

Hughes et al (2017) [16] presented an approach to classify clinical text at the sentence level. The model presented here is very similar to a normal CNN approach, except multiple configurations of stacked convolutional layers are used. The model ends with a fully connected layer followed by a softmax layer.

The model is tested with various hyper parameter optimization strategies mentioned in the paper. Keras model summary and code for medical-CNN can be found here.

7.2.4 VDNN for Text Classification

Conneau et al (2016) [17] presented Very Deep CNN, which operates directly at the character level. This model also shows that deeper models perform better and are able to learn hierarchical representations of whole sentences.

The overall architecture of the model contains multiple sequential convolutional blocks. The two design rules followed are:

1. For the same output temporal resolution, the layers have the same number of feature maps, and
2. When the temporal resolution is halved, the number of feature maps are doubled. This helps reduce memory footprint of the model.

The model contains three pooling operations that halve the resolution each time resulting in three levels of 128, 256, 512 feature maps. There is an optional shortcut connection between the convolutional blocks that can be used, but since the results show no improvement in evaluation, that component was dropped in this project.

Each convolutional block is a sequence of convolutional layers, each followed by batch normalization layer and relu activation. The down sampling of the temporal resolution between the conv layers (and) in the block are subjected to multiple options:

1. The first conv layer has stride 2
2. is followed by k-max pooling layer where k is such that resolution is halved
3. is followed by max pooling layer with kernel size 3 and stride 2

Characters used in character vectors for all the models in this paper are:

The VDNN architecture and each convolutional block are presented below. Keras model summary and code for VDNN can be found here.

7.2.5 Character-level CNN

Zhang et al (2016) [18] proposed a model similar to VDCNN where sentences are quantized to characters and a very deep nine layer network architecture with six convolutional layers and three fully connected layers. This is not implemented.

7.2.6 Multi Channel Variable size CNN

Yin et al (2016) [20] propose MV-CNN, which combines diverse versions of pre-trained word embeddings and extract features from multi-granular phrases with variable-size convolutions. Using multiple embeddings of the same dimension from different sets of word vectors should contain more information that can be leveraged during training.

The paper describes maintaining three dimensional embedding matrices with each channel representing a different set of text embeddings. This multi-channel initialization might help unknown words across different embeddings. Frequent words can have multiple representations and rare words (partially known words) can be made up with other words. The model then has two sets of convolution layers and dynamic k-max pooling layers followed by a fully connected layer with softmax (or logistic) as the last layer.

The paper describes two tricks for model enhancement. One is called mutual learning, which is implemented in this project. The same vocabulary is maintained across the channels and they help each other tune parameters while training. The other trick is to enhance the embeddings with pretraining just like a skip-gram model or autoencoder using noise-contrastive estimation.

Different sets of embeddings used for this model comes from glove, word2vec, custom trained vectors on the train and test corpus using fast text tool as discussed in the section Word Vectors.

Keras model summary and code for MVCNN can be found here.

7.2.7 Multi Group Norm Constraint CNN

Ye Zhang et al (2016) [21] proposed MG(NC)-CNN and captures multiple features from multiple sets of embeddings that are concatenated at the penultimate layer. MG(NC)-CNN is very similar to MV-CNN but addresses some drawbacks, such as model complexity and requirements for the dimension of embeddings to be the same.

MG-CNN uses off the shelf embeddings and treats them as distinct groups for performing convolutions following up with max-pooling layer. Because of its simplicity, the model requires training time in the order of hours. MGNC-CNN differs in the regularization strategy. It imposes grouped regularization constraints, independently regularizing the sub-components from each separate group (embeddings). Intuitively, this captures discriminative properties of the text by penalizing weight estimates for features derived from less discriminative embeddings. Different sets of trained embeddings and l2-norm regularizer will be used in the implementation. Keras model summary and code for MGCNN can be found here.

7.2.8 Other Models and Study on CNN methods

Johnson et al (2015 and 2016) [22] [23] [24] papers propose models based on CNN with words and characters under wide and narrow convolutions. These papers use word vectors as one hot encoding in the vocabulary. These models are not implemented in this project but it is worth mentioning their work.

Ye Zhang et al (2016) [25] derives practical advice from extensive empirical results for getting the most out of CNN models in text classification task. Using right hyper parameters might look like black magic, but this paper gives sensible practical tips for choosing them.

1. One must consider starting with non-static word vectors rather than one-hot vector. Concatenating word vectors has not shown improvement in performance.
2. Coarse line search over single filter region or kernel size in reasonable range 1~10 and then it's worth exploring some sizes around the best kernel size
3. Alter the number of feature maps (depth or number of filters) from 100 to 600 and when exploring use small dropout (0 to 0.5) and large max-norm constraint. There is a tradeoff between increasing feature maps and training time.
4. Try relu and tanh activations on nonlinear layers and it may be worthwhile to try no activations.
5. Use 1-max pooling since it doesn't expend much resources.
6. When increasing number of feature maps, try increasing regularization constraint.
7. Cross validation must be performed to estimate range of variances for better modelling.

Le et al (2017) [26] studied the importance of depth in convolutional models for text classification and shows that simple shallow and wide networks outperform deep models with text represented as characters.

Wang et al () [27] proposed using output of semantic clusters and applying a model similar to DCNN on top for document classification.

7.3 Models Based on Recurrent Neural Networks

All the models presented in this section are based on just RNNs (LSTM, GRU). They are also modified to take into consideration of other features such as gene names and variation words. These features are presented as either words or characters based on requirement of the model.

7.3.1 Generative and Discriminative RNN

Yogatama et al (2017) [28] studies generative and discriminative LSTM models for text classification. Suggested generative models approach asymptotic errors faster than than the discriminative models.

In the discriminative model the LSTM “reads” the document and uses its hidden representation to model the class posterior. Discriminative models are known to suffer from catastrophic forgetting when learning sequentially. Discriminative models use LSTM with “peephole” connections to encode a document and build a classifier on top of the encoder by using the average of the LSTM hidden representations as the document representation.

Generative models are robust for shifts in data distribution. In generative models for text classification, the proposed network can model unbounded (conditional) dependencies among words in each document. Here, documents are generated word by word and conditioned on a learned class embedding. This model takes advantage of the setup and maximizes the training objective for a new class, while decoupling other classes more easily. Two types of generative models proposed are:

1. In shared LSTM, to predict a single word, the LSTM’s hidden representations are concatenated with a separate label class embedding and another softmax layer over vocabulary is added.
2. In independant LSTM, the sharing of parameters are removed, which also results in more parameters in the model.

Keras model summary and code for discriminative model can be found here

7.3.2 LSTM Deep Sentence Embedding

Palangi et al (2015) [29] proposed LSTM-RNN for text classification. The model has shown to perform better than Paragraph Vectors for document/sentence embedding. As the model reads to the end of the sentence, the topic activation accumulates and the hidden state representation at the last word encodes the rich contextual information of the entire sentence. LSTM-RNN are effective against noise and can be robust in scenarios when every word in the document is not equally important and only salient words need to be remembered using limited memory.

This is a very simple model that takes all the words in the document sequentially and the final output gives the document embedding. The model does not use max pool layers to capture global contextual information.

Keras model summary and code for LSTM based embedding model can be found here.

7.3.3 Multiplicative LSTM

Krause et al (2016) [30] proposed Multiplicative LSTM which is a hybrid RNN that gives the model more flexibility in choosing recurrent transitions for each possible input, which makes it more expressive in autoregressive density estimation. This model is proposed for sequential language modelling, but it was chosen for this project to see how it helps in text classification tasks. The authors argue that current RNNs have a hard time recovering from mistakes when predicting sequences. If RNN hidden state remembers erroneous information then it might take more time to recover and there will be a snowball effect. Some proposed solutions such as introducing latent variables and increasing memory will only result in complex intractable distribution over the hidden states and reinterpreting stored inputs. The multiplicative model provides more flexibility for transitions in these kind of situations.

Since this is a modification to existing RNN unit (LSTM or GRU), this has to be implemented as a keras Layer similar to keras LSTM layer. The equations for mLSTM are

Compare and contrast with LSTM equations.

The model implemented for text classification is a very simple generative model as suggested in section 7.3.1. Keras model summary and code can be found here

7.3.4 Hierarchical Attention Networks

Yang et al (2016) [31] proposed Hierarchical Attention Networks for document classification, which took advantage of a hierarchical structure that mirrors the hierarchical structure of the documents and two levels of attention mechanisms applied to differentially more and less important content when constructing document representation.

First let us look into what attention mechanism means. For this, let us look into the feedforward attention mechanism proposed by Raffel et al (2015) [32]. Attention mechanism allows more direct dependencies between the states of the model at different points in time. From the picture, the vectors seen in the hidden state sequence are fed into a learnable function that produces a probability vector, which in turn is used to find the weighted average of the output hidden states.

Similarly, Hierarchical Attention Networks proposes a two stage attention mechanism to encode documents. The intuition behind the model is that different words and sentences in a document are differentially informative. The document representations are constructed from each sentence and each sentence representation is constructed from words. Each sentence in a document can be considered as a word sequence that is fed into a generative LSTM layer with attention on top, and again the same procedure is followed for sentences with attention on top. Finally a softmax layer on top to classifying the document labels.

Keras does not provide apis for attention mechanism, hence a custom Attention layer is created to handle different levels of attentions.

Keras model summary and code for HAN model can be found [here]

7.3.5 Other Models and Study on RNN

Any language model that has LSTM units in them can be modified to generate embeddings that may represent the document succinctly, which can in turn be used for classification.

7.4 Models Based on CNN and RNN ensemble

All the models presented in this section are based on ideas of both CNN and LSTM. They are also modified to take into consideration other features such as gene names and variation words. These features are presented as either words or characters based on the requirement of the model.

7.4.1 Recurrent Convolutional Network

Lai et al (2015) [33] proposed RCNN model that addresses the limitation of unbiased CNN models with shorter window sizes and biased RNN models. The model has a bi-directional recurrent structure that reduces noise and captures semantic information to the greatest extent possible. Max pool layer on top of recurrent structure judges the features role in capturing key components necessary for classification.

The model has bidirectional LSTM that captures context information from nearby words in both the directions. Then the max pool layer captures the key features that are fed as input into the softmax layer that classifies the text provided.

Keras model summary and code for RCNN can be found here

7.4.2 Ensemble of CNN and RNN for Multi Label Categorization

Chen et al (2017) [34] proposed a model that is an ensemble of CNN and RNN for multi-label text categorization, which helps capturing both the global and local semantics. It employs a CNN model as proposed in section CNN for sentence classification and a RNN model that returns output as sequence that are classified for word labelling.

The model is modified to be used for text classification. The RNN structure is used to create sentence embedding with the same input sentence provided for CNN. The output of the CNN model is fed to the first hidden state of the RNN that helps with better information retrieval.

Keras model summary and code for the ensemble can be found here

7.4.3 C-LSTM Neural Networks

Zhou et al (2015) [35] proposed C-LSTM Neural Networks, which uses CNN to capture local features of phrases and RNN to capture global and temporal sentence semantics.

The model extracts high-level correlations from n-gram features with shorter windows in CNN models and are fed as a sequence to the following LSTM layer, which helps with creating an embedding for the document for classification. The output of each convolution (single feature map) has semantic information from the whole sentence as a sequence. From these multiple feature maps, without using a max-pool layer, the vectors representing positional information are grouped or concatenated which are fed to RNN layer as a sequence.

Keras model summary and code for the c-lstm model can be found here

7.4.4 Combination of CNN and RNN

Wang et al (2016) [36] proposed another model for sentiment analysis for shorter texts, which is a combination of CNN and RNN. The model helps capturing coarse grained local features from CNN and long distance dependencies from RNN.

The model is very similar to C-LSTM, but it employs a max pool layer with multiple feature maps that reduces the output of convolution by half.

Keras model summary and code for the cnn & rnn combination model can be found here

7.4.5 AC-BLSTM Networks

Liang et al (2016) [37] proposed a model that combines asymmetric CNN (ACNN) and bidirectional LSTM (BLSTM). In order to make the models deeps they factorized convolutions to and convolutions. Another important feature of the model how the asymmetric convolutions are handled, instead of truncating lengths, smaller convolutions are performed on them. This model is similar to previous seen model Combination of CNN and RNN in handling how the feature are fed from CNN to RNN. RNN employed here is a bidirectional LSTM whose outputs are concatenated and fed as input to the final softmax layer for classification.

Keras model summary and code for the ac-blstm model can be found here

7.4.6 Other Studies on CNN and RNN ensemble methods

Language models, such as the Char Aware Neural Language Model [38], that can be modified to generate embeddings that represent the text, can in turn be used for classification with a softmax layer on top (with or without fully connected layer).

8 Results:

The results are summarised in the below table. The Kaggle Competition compares the results against the predicted categories for the test data. Since this is a study, the information below compares the categorical accuracy obtained with the validation set. The validation set is same across all the models including the count models that are controlled using random seed.

Model Names	Categorical Accuracy
Yandex Technologies CatBoost	0.587453613072
SKlearn Multinomial Naive Bayes	0.663663663664
SKlearn Support Vector Machine	0.285285285285
SKlearn Softmax Regression	0.507507507508
SKlearn K Nearest Neighbour	0.612612612613
SKlearn Passive Aggresive Classifier	0.630630630631
SKlearn Decision Trees Classifier	0.597597597598
SKlearn AdaBoost Classifier	0.408408408408
SKlearn Random Forest Classifier	0.627627627628
Extreme Randomization Trees	0.630630630631

Now, let's look at the results generated from the deep models. Since there are an exponential number of ways to test different hyperparameters, I decided to study all the models with keras default parameters for the layer arguments and hyperparameter tuning as suggested in the papers.

Model Names	Categorical Accuracy
CNN for Sentence Classification	0.6276
DCNN for Modelling Sentences	0.66366
Medical CNN	0.186844408512
VDNN for Text Classification	0.52600
Rationale Augmented CNN	0.65766
Multi Channel Variable size CNN	0.60164
Multi Group Norm Constraint CNN	0.60195
Generative and Discriminative RNN	0.59470

Model Names	Categorical Accuracy
LSTM Deep Sentence Embedding	0.59350
Multiplicative LSTM	0.59331
Hierarchical Attention Networks	0.66967
Recurrent Convolutional Network	0.65465
Ensemble of CNN and RNN for Multi Label Categorization	0.54955
C-LSTM Neural Networks	0.55856
Combination of CNN and RNN	0.59551
AC-BLSTM Networks	0.59760

We can see the models are fairly performing better than count based models. This could be run longer for better results, but taking the number of models and the time into consideration, each model, on average, was run for about 10 epochs.

9 Conclusion

The models discussed and implemented in this project are a collection of selected deep models that depend on state-of-the-art Dense Neural Network models CNN and RNN. Levy et al (2015) [40] performed an extensive study on comparing neural models and count based n-gram models. This revealed that much of the performance gain of word embeddings are from design choices and hyperparameters rather than much revered word embeddings. This does leave more room for improvement and more research to be conducted to find better models for text classification.

10 References

1. Personalized Medicine: Redefining Cancer Treatment
2. The Absolute Minimum Every Software Developer Absolutely, Positively Must Know About Unicode and Character Sets (No Excuses!)
3. Natural Language Toolkit
4. Efficient Estimation of Word Representations in Vector Space
5. Sebastian Ruder. On word embeddings
6. Enriching Word Vectors with Subword Information
7. Bag of Tricks for Efficient Text Classification
8. Fast Text
9. Glove Vectors
10. Biomedical natural language processing
11. Distributed Representations of Sentences and Documents
12. Deep Learning Ian Goodfellow and Yoshua Bengio and Aaron Courville
13. Hands-On Machine Learning with Scikit-Learn and TensorFlow
14. Convolutional Neural Networks for Sentence Classification
15. A Convolutional Neural Network for Modelling Sentences
16. Medical Text Classification using Convolutional Neural Networks
17. Very Deep Convolutional Networks for Text Classification
18. Character-level Convolutional Networks for Text Classification
19. Rationale-Augmented Convolutional Neural Networks for Text Classification
20. Multichannel Variable-Size Convolution for Sentence Classification
21. MGNC-CNN: A Simple Approach to Exploiting Multiple Word Embeddings for Sentence Classification
22. Effective Use of Word Order for Text Categorization with Convolutional Neural Networks
23. Semi-supervised Convolutional Neural Networks for Text Categorization via Region Embedding
24. Convolutional Neural Networks for Text Categorization: Shallow Word-level vs. Deep Character-level
25. A Sensitivity Analysis of (and Practitioner's Guide to) Convolutional Neural Networks for Sentence Classification
26. Do Convolutional Networks need to be Deep for Text Classification ?
27. Semantic Clustering and Convolutional Neural Network for Short Text Categorization
28. Generative and Discriminative Text Classification with Recurrent Neural Networks
29. Deep Sentence Embedding Using Long Short-Term Memory Networks: Analysis and Application to Information Retrieval
30. Multiplicative LSTM for sequence modelling
31. Hierarchical Attention Networks for Document Classification
32. Feed-Forward Networks with Attention Can Solve Some Long-Term Memory Problems
33. Recurrent Convolutional Neural Networks for Text Classification
34. Ensemble Application of Convolutional and Recurrent Neural Networks for Multi-label Text Categorization
35. A C-LSTM Neural Network for Text Classification
36. Combination of Convolutional and Recurrent Neural Network for Sentiment Analysis of Short Texts
37. AC-BLSTM: Asymmetric Convolutional Bidirectional LSTM Networks for Text Classification
38. Character-Aware Neural Language Models
39. Highway Networks
40. Improving Distributional Similarity with Lessons Learned from Word Embeddings