

Independent Study of Natural Language Processing: Report

Robin Bhattacharya

April 17, 2018

Abstract

This report is a detailed discussion of all my projects, assignments and study materials used for the Independent Study of Natural Language Processing that I did under professor Anna Rumshisky.

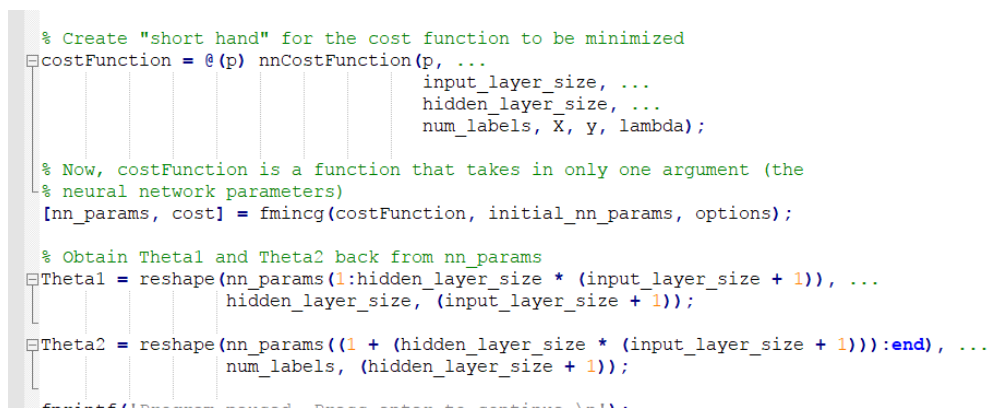
1 INTRODUCTION

The current burst of Artificial Intelligence in every industry has forced us to start looking at things differently. Things which seemed far fetched few years ago, seems so much casual today. Facebook's instant comment translation for instance; no one imagined this was possible so easily even two decades ago. I got interested in Artificial Intelligence after I heard a TEDx talk from the founder of Kaggle. After I took Machine Learning in spring I wanted to increase my AI knowledge even further. That's when I decided to take an Independent Study course on Natural Language Processing under Professor Anna Rumshisky. This is a report on the independent that I did under her guidance on Natural Language Processing. This report contains detailed discussions of my understanding on all projects, exercises, and study materials I used during my independent study.

2 STUDY MATERIALS

2.1 Coursera Machine Learning Course

In addition to the course at UMass Lowell provided the foundation for me. Exercises where we had to implement backpropagation to dimensionality reduction through Principal Component Analysis. Please consider the following screen-shot which is one of the exercises which I had to do during that course.



```
% Create "short hand" for the cost function to be minimized
costFunction = @(p) nnCostFunction(p, ...
    input_layer_size, ...
    hidden_layer_size, ...
    num_labels, X, y, lambda);

% Now, costFunction is a function that takes in only one argument (the
% neural network parameters)
[nn_params, cost] = fmincg(costFunction, initial_nn_params, options);

% Obtain Theta1 and Theta2 back from nn_params
Theta1 = reshape(nn_params(1:hidden_layer_size * (input_layer_size + 1)), ...
    hidden_layer_size, (input_layer_size + 1));

Theta2 = reshape(nn_params((1 + (hidden_layer_size * (input_layer_size + 1))):end), ...
    num_labels, (hidden_layer_size + 1));

fprintf('Program ended. Press enter to continue.\n');
```

Figure 1: The above snippet describes one of the assignments that I worked on during the course. This one specifically deals with implementation of the parameter matrices during backpropagation.

$$\mathbf{h}_t = \phi(W\mathbf{x}_t + U\mathbf{h}_{t-1}),$$

Figure 2: Here the $\mathbf{h}(t-1)$ is the hidden step, the \mathbf{x}_t is the input, where $\mathbf{h}(t-1)$ is a function of \mathbf{x}_t and is modified using the weight matrix. This \mathbf{h}_t is fed in the next step as $\mathbf{h}(t-1)$ in the same way.

2.2 Blogs on RNN and LSTM

The first step towards NLP is understand the kind of neural networks used in the NLP world, The Recurrent Neural Nets. Which is why this blog served as one of the most important source of information during my study. Which brings us to the question,

What is a **RNN** ?

Typically, as its names suggests a Recurrent Neural Nets unlike a feed forward network not only takes the input example as they would see or feed to them, they also take in what they perceived previously in time. In simple words, there is a feedback loop connecting to the past decisions, which ingests their own outputs moment after moment as inputs, [Mikolov et al.[5]] Many people try and simplify it by saying it has a ‘ memory ’ but, we use the information that is present in the sequence itself using something called ‘ hidden state’. The hidden state spreads across a range of time steps and it affects the processing of each new example. As each event in an RNN depends on or is a function of one or more events that happened in the past we may also say that RNNs share weights over time. The basic equation of RNN is mentioned above in figure 2

What makes RNN so special?

RNN becomes more exciting as instead of a fixed sized vector as input and output, we have a sequence of vectors on which we operate. We have sequences in input, in output or in both the cases. Take language translation for instance, here we put in a sequence and we get another sequence as output (the translated language). But in tasks like image captioning we give in a single image and we get a sequence back in the form of an image caption. As mentioned above the hidden state update. You can update the weights in each layer and stack them up like a pancake.

2.2.1 LSTMS

As powerful and exciting RNN may sound, it has got its limitation. That is when Long-Short-Term Memory Network or LSTM comes into use. RNN uses information from the previous time steps to influence the current time step. This maybe powerful in current time step, but the influence drastically decreases on say the 10th step from the current one.

The LSTMs unlike the RNNs has gates associated with each time step which helps in retaining information like verbs from previous time steps and ignores information that is less important like stop words. Each unit in an LSTM has three gates the input gate, forget gate and the output gate. The cell state which runs like a conveyor belt through the gate is managed by the three mentioned gates. The forget gate decides what to forget from the cell state. Then the other two layers decide what information is to be retained into the cell state. A sigmoid layer coupled with a tanh layer is used to get the distribution which decides what information is to be retained, Sundermeyer et al [8] The agility of LSTMs is the reasons it is being used so widely in so many various areas. From Amazon’s Alexa to Apple’s Siri all models made improvement from a 95 percent to a 99 percent accuracy upon using LSTMs from using heavyweight speech recognition models.

2.2.2 Character Level Language Models

A character level model would be when a single character is used to get the probability distribution of the next character. For example, if we have a word saying ‘jail’, then we feed in a one hot vector which is then used along with the hidden state vector, to get the probability distribution of the letter ‘a’ from the letter ‘j’. Technically, we call this a standard SoftMax classifier or a cross-entropy loss on every output vector simultaneously, Peng et al [6] training is over, we feed a character into

the RNN, we get distributions of what characters are likely to come next.

2.3 Videos on NLP and RNNs

Post the blogs the ideal way to continue learning was watching videos on them. I started with videos on RNN and lectures by Bengio and Lex Fridman. Although the videos were not significantly different some of the key points from the videos were that as follows,

- RNNs can be run as Generative model which would have the ability to produce its own inputs. It becomes a free-running model. In such a model we can make a probabilistic interpretation of the model as directed graphical model that is fully connected in some sense or is observable. At a given time 't' it can produce an input given all the previous inputs to the model.
- We can think of RNNs as Graphical models. In such a model there would be no stochastic latent variables. An ideal equation of a such a model would be deterministic. We can simple use maximum likelihood to calculate the gradient as the model is very simple.
- Training a Recurrent Neural Nets forcibly through Maximum Likelihood is what we call Teacher Forcing.
- Memory networks are special kind of memory-based RNNs, where there is an external memory storing information storing part of the external memory.
- Apart from RNNs there has been recent implantation of Recursive Neural Nets in NLP (instead of a pancake stack, imagine this to be a tree based structured network), Bidirectional Recurrent Neural Nets to be precise. There are two networks where one of them go from the future to the past and the other one goes from the past to the future. The concatenation of the state at each time step is the representation of the sequence. Although these kinds of bidirectional structure can't be used as a generative model because technically we would be using the future to predict itself and it doesn't make sense.

3 ASSIGNMENTS

3.1 LOGISTIC REGRESSION WITH MNIST HANDWRITTEN DIGITS IMAGES

Once I was familiar enough with NLP and Deep Learning in general, I was given the first assignment. The first assignment was to make an image classifier using logistic regression. In this exercise we classify handwritten digits from 0 to 9 (MNIST dataset). The main idea behind this assignment was to make myself familiar with tensorflow. The details of tensorflow includes concepts like placeholders and variables, which hold the required tensors to get the logits. This is followed by getting the cross-entropy loss which helps you get the batch. Once I have the batch, the next step is to use an optimization technique to reduce the loss. In this case I used gradient descent optimization. Following that, we launch a tensorflow session which we use to calculate one loss batch at a time. Once we have the trained model we use it on the test data to get the final accuracy.

CONCEPTS USED: Apart from tensorflow itself this assignment gives us a clear view on logistic regression.

What is **logistic regression**?

In simple words logistic regression is a method for analyzing a dataset where more than one independent variables determine the outcome. The outcome is dichotomous its either true or it's false. The main idea is to map a relation between the independent variables which determine the outcome and the dependent variables which is dichotomous. Logistic regression generates the coefficients of a formula to predict a logit transformation of the probability of presence of the characteristics of interest. Cases where dependent variables have more than two outcomes may be

⁰https://github.com/ron12013/Independent_Study_under_Professor-Anna-Rumshisky

$$P(y = 1|x) = h_{\theta}(x) = \frac{1}{1 + \exp(-\theta^{\top} x)} \equiv \sigma(\theta^{\top} x),$$

$$P(y = 0|x) = 1 - P(y = 1|x) = 1 - h_{\theta}(x).$$

Figure 3: Here we see the two conditional probabilities of $y = 0$ and $y = 1$, and the hypothesis sums up to 1 on either case. Here the sigmoid follows the standard formula mentioned in the first equation.

termed as multinomial logistic regression, Hosmer et al [2] An example where logistic regression might be used would be if we want to classify individuals into two categories based on explanatory variables, like classify new students into “admitted” or “rejected” group depending on their gender. All it comes down to is that we try to learn a function of the form as follows,

Here we see the two conditional probabilities of $y = 0$ and $y = 1$, and the hypothesis sums up to 1 on either case. Here the sigmoid follows the standard formula mentioned in the first equation. Although tensorflow takes care of all the details, it is important to understand all the mathematics behind logistic regression. All details related to how to implement using the tensorflow details is explained and mentioned in the repository.

3.1.1 Experiments and Results

This being a standard exercise, the only experiment which I did was with the handwritten digits itself. The main idea of this assignment was to make myself familiar with tensorflow. Although in order to understand gradient descent myself better, I decided to experiment with various learning rates and various number of epochs to understand the correlation better. As we know, that a higher learning rate results in a decrease in accuracy as the instead of taking small steps towards loss reduction, it overshoots and results in a higher loss. The table above mentioned contains the relation of the hyperparameters as discussed here. As for training and testing data, I used the data provided by MNIST. MNIST provides training and testing data which I just imported in my code. The details of that are provided in the code itself in the form of comments. The results of the experiments are contained in Table 1.

Learning Rate	Epoch	Test Accuracy
0.5	100	89.3
	250	91.1
	500	91.7
	1000	91.59
0.7	100	89.8
	250	90.8
	500	91.6
	1000	92.2
2.7	100	84.6
	250	84.91
	500	85.11
	1000	88.27

Table 1: Here we see how the learning rate is inversely proportional to the accuracy. We also see that the number of epochs means the model generalizes well and hence more accuracy.

3.2 CONVOLUTIONAL NEURAL NETWORK WITH MNIST HANDWRITTEN DIGITS IMAGES

The second assignment of the course was getting myself further familiar with tensorflow and convolutional neural nets. The aim was to make an image classifier based on a convolutional neural net. The basic idea of a CNN is one layer built on top of the other. Tensorflow’s graph structure is beneficial for such an implementation. CNNs are comparatively easy to understand compared

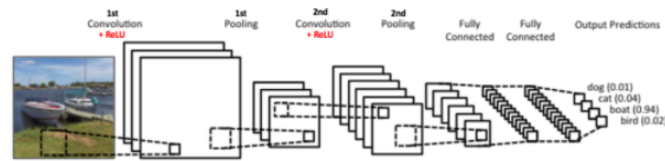


Figure 4: As mentioned in the explanation, the convolution + relu followed by the max pooling step is repeated until we get a vector which could be used for prediction or training.

to RNNs or LSTMs, but the recent usage of CNN in NLP made it important for me to understand CNN in the form of this assignment.

CONCEPTS USED: Apart from further increasing my knowledge in Tensorflow this assignment also made me familiar with Convolutional Neural Networks

What is a **Convolutional Neural Net**?

Personally speaking, I found ConvNets most easy to understand and I feel its one of the most powerful neural net model present. It has proved to be very effective in areas in image recognition and classification. These are in fact used not only to power self-driving cars but also used to power robot vision. The usual computation starts from the convolution step, where a filter or a kernel is used, and it reads the value of the pixel across the length of the image (images in convnet experiments are usually represented in pixels, for example MNIST dataset has a 28×28 image with the number of pixels equal to the product) When the kernel is slid a matrix multiplication takes place between the parameters of the kernel and the values of the pixels. Once the kernel has read over the image it produces something we call feature map. Before explaining depth, some terms that are important for the explanation of ConvNets areas follows

Depth: Depth corresponds to the number of filters used in the convolution operation.

Stride: Stride is the number of pixels by which we slide our filter matrix over the input matrix. For example, if the stride is 1, then we move one pixel at a time.

Zero-padding: Sometimes the size of the input image is not convenient for the kernel being used for the model. So, a padding of zero is made around the input image as it allows us to control the size of the feature map when the convolution step takes place.

After the convolution layer, we need to make the convolution layer non-linear as in real world any functions are nonlinear. Rectified Linear Unit or ReLu is used for achieving non-linearity. Any negative value in the feature map after convolution layer is changed to zero Kachnbrenner et al [3] Once nonlinearity is achieved, the next step is down-sampling. It is basically reducing the dimension of the feature map while still retaining the most important information. Down-sampling is also known as max-pooling. Apart from these there is also something called Average-pooling where instead of taking the highest value, we take the average of the pixel values. So, these above steps are repeated in the order of Convolution + ReLu \rightarrow Pooling \rightarrow Repeat, until you get a vector which you compare with the ground truth to get the result. Consider the following image which gives a clear explanation.

3.2.1 Experiments and Results

This exercise was the second assignment that was meant for better understanding of tensorflow. For this exercise too, I imported the MNIST data(test and train) provided by them to do the assignment. As far as experiments are concerned, I tried different learning rates and change the epoch sizes to check the testing accuracy of the model. As Table 1 shows, this assignment also has a similar correlation of test accuracy reducing on a large learning rate, and getting affected by less epochs.

4 PROJECTS

4.1 WORD2VEC SKIPGRAM MODEL ON THE LATEST WIKIPEDIA DUMP

This project involved downloading the latest Wikipedia dump, extracting corpus out of it. Once I have that corpus I can use it to get the word2vec embeddings of that corpus. One of the most important tasks in NLP is to get word vectors (i.e., good representation of words). We tokenize words so that it becomes easier to vectorize it later. Vectorizing means that each word token is represented as a vector of size 300. We call this the embedding vector.

When I watched through the videos of Richard Socher (the Stanford NLP 224d course), a good embedding representation will capture some correlation among words (we all know about the famous king and queen example). This exercise is implementing the skip gram model of word2vec where the center word is used to predict the surrounding words which we call as the target words. The technical details of implementation using tensorflow is mentioned in the GitHub repo of this independent study course.

CONCEPTS USED: Word embeddings in NLP, word2vec embeddings specifically. Implementation methodology of skip gram word2vec model.

What is **word2vec**?

Having right data while solving a machine learning problem is crucial. Real world data doesn't follow any structure and is full of noise. Therefore, we need a proper representation of the data which we could use in models to make good predictions. Word embeddings are a method by which we transform the data before feeding it to a learning algorithm. As important word embeddings are to get good predictions and make a learning algorithm accurate, it is very difficult to understand and achieve such an embeddings which will capture proper correlations between words which would be beneficial for the learning algorithm it goes into. One such embedding is word2vec embeddings. In 2014, Mikolov and Golderg at al., [1] came up with a technique which can convert a categorical feature with a high number of modalities into a smaller set of easier-to-use numerical features. These features not only successfully learned relationships between the several modalities like how classic word embeddings do with language.

Word2vec tries to determine the meaning of the word by analyzing the surrounding neighbor words (context words). Given a corpus the model loops through the sentences and uses the current word to predict the neighboring words, this is what we call the skip-gram model. Continuous bag of words is the opposite of skip-gram where we use the surrounding words to predict the current word. The limit of the number of words in each window is determined by the parameter we call the '**window size**'.

Skip-gram models typically have a very shallow neural net structure with one layer. The network initializes its weights randomly and then tries to minimize the error it gets while predicting context words through an iterative process. Once the training is successful the word embedding of each word is obtained by the multiplication of the network's weight matrix with the word's one-hot-vector. Consider 5 which gives a better explanation as to how exactly does word2vec works.

4.1.1 Experiments and Results

The purpose of this project was to get a word2vec skip-gram embedding of the Wikipedia corpus. As far as experiments are concerned the corpus after preprocessing was about of 14 GB. Which is why it was difficult to run a program of that size in my personal computer. Instead, to test my code I tried a small size corpus (an extract of the original corpus) on my local machine which worked successfully. As far as results are concerned the attached picture is of the embedding of different words that I got from the model I built. In addition to that, I have also used tensorflow's ability to plot the word vectors in a 2D plot for better understanding. I have added the picture of the same, Figure 6 and 7

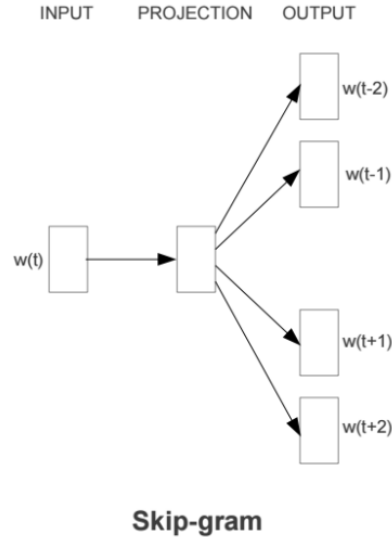


Figure 5: We see in this picture how the current word is used to predict the surrounding words where $w(t)$ is the current word. CBOW is the opposite of this where $w(t-1)$ may be $w(t)$ and that is used to predict the current word.

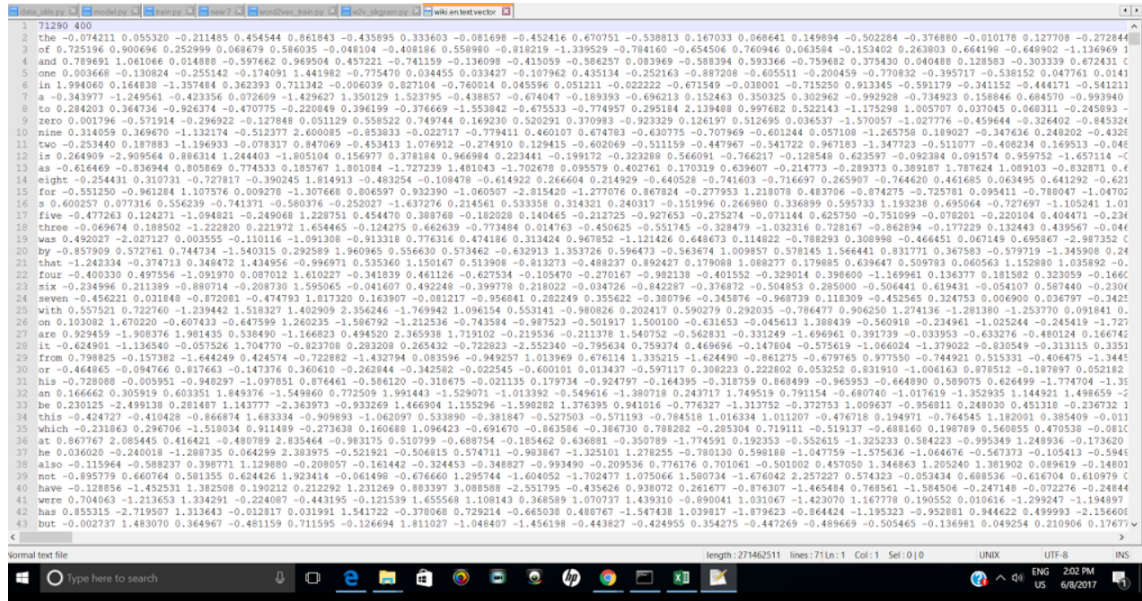


Figure 6: The above picture shows us a snippet of the word2vec skipgram embeddings which I achieved from the model

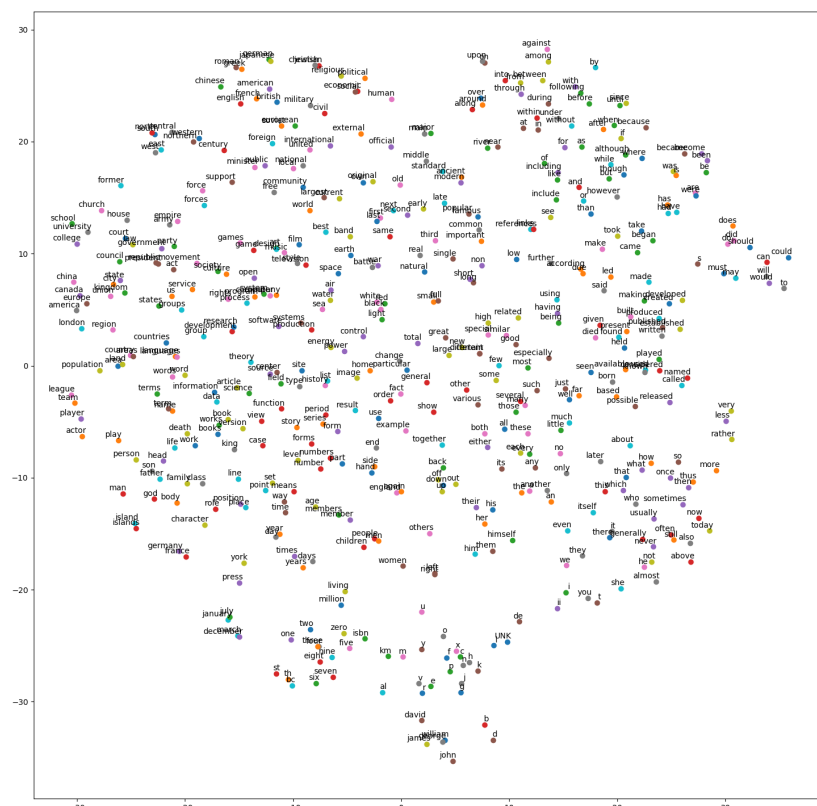


Figure 7: The above figure is a 2D plot made using tensorboard and the dimensionality reduction here is done using PCA

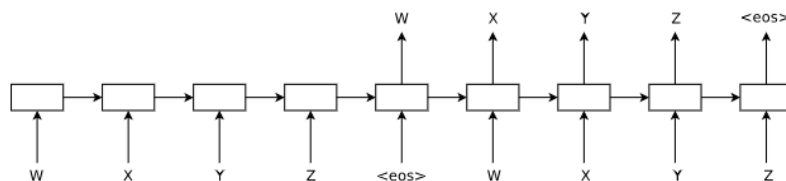


Figure 8: The figure above is a sequence auto-encoder. The sequence auto-encoder reads in the sequence input using a recurrent network and then transforms the read in sequence into a hidden state. The decoder then uses that hidden representation to reconstruct the original input which is also being fed to the decoder.

4.2 SEQUENCE TO SEQUENCE AUTOENCODER IMPLEMENTATION

We all know that RNN (Recurrent Neural Networks) and LSTMs are used extensively in the world of Computer Vision and NLP. This project involved taking a corpus (in this case the Wikipedia corpus from the previous project) and using it on a sequence to sequence auto-encoder. The aim of this project was to take a sentence and using on a sequence to sequence auto-encoder. We train the model to learn the minimal representation of the sentence using an encoder which converts the input sentence into a fixed representation.

The way this was implemented was using tensorflow's multi RNN cell, using a GRU cell for each layer or a standard RNN cell unit of tensorflow. Then following that we use the tensorflow's RNN sequence to sequence model providing it the cell type (the details of the cell type are mentioned in the GitHub repository of this independent study course)

CONCEPTS USED: Implementation of autoencoder (the sequence to sequence implementation), autoencoders in general. Using GRU cells while implementing using tensorflow.

What is an **autoencoder**?

In simple words, a autoencoder is a neural network that is trained to attempt to copy its input to its output. The hidden layers make its own representation of the input which may or may not be the minimal representation. The basic idea behind a autoencoder is to have a model which is restricted in various ways so that its unable to exactly copy the input to the input. The restrictions imposed in the network forces it learns the input in the best possible way within those restrictions (an approximate learning). This also enables a network to learn the most important parts or areas of the inputs which would be essential in the future.

4.2.1 Sequence autoencoder

We all know that attention based sequence-to-sequence models have in the recent past achieved state of the art in areas like machine translation, syntactic constituency parsing, and semantic role labeling. We also know that despite RNNs capability of preserving word ordering and modeling sequential data, it is not used for the same so often. The reason being that, training RNNs through back-propagation with the increase in time becomes very difficult, Kovcisky et al.,[4]. The basic idea behind using an autoencoder for sequential data involves using a recurrent network as an encoder to read in an input sequence into a hidden state, which becomes the input to the decoder recurrent network that predicts the output sequence.

What I worked on was unsupervised in the sense that, it was similar model to the one mentioned above except the decoder also gets the same input as the encoder. The goal is to deconstruct the input using the encoder into a particular representation and using that to reconstruct the input sentence itself.

The figure above is a sequence auto-encoder. The sequence auto-encoder reads in the sequence input using a recurrent network and then transforms the read in sequence into a hidden state. The decoder then uses that hidden representation to reconstruct the original input which is also being fed to the decoder.

archism is derived from the greek without archons ruler ahief king anarchism as a political
phioo snarcaism isareriret rroaaioe lreeemiisou

Figure 9: A sample sentence generated by the autoencoder at the end of 400 epochs

in place of what are regarded as authoritarian political structures and coercive economic
instittttiuii pticeootrsias rrn regarnrdtasautpori

Figure 10: A sample sentence generated by the autoencoder at the end of 900 epochs

4.2.2 Experiments and Results

For this project I implemented a vanilla model of sequence to sequence autoencoder. As mentioned above a sequence autoencoder can have same input to the encoder and the decoder, as the goal is to regenerate the input from the hidden representation by the decoder and compare with the input of the decoder, and the degree of correctness of that conversion is what we would call accuracy.

Here the input to the model (encoder) was from the source.txt file. The file is an extract from the initial Wikipedia corpus which is used to run this model. For training data, the code provides a flag which specifies what batch size you would require for the training data. That batch size multiplied by the total time steps and the input size is what we need for the training data. This entire batch runs under a for loop which runs along the length of the number of epochs (which is also a flag with a default value). The above model gives around **92 percent** training accuracy.

As for testing the model, the test data was taken from the same file as the idea was to regenerate the input itself. In order to do that the batch size is multiplied by the current value of 'i' which runs along the length of epochs. So, for example if we have a 10 line corpus and 5 lines are used as an input (that may vary on the batch size and input size) the next 5 lines would be used for testing (this also depends on the batch size), the test accuracy of the model was **85 percent**.

Below are some pictures of what the decoder was generating after a certain number of epochs.

5 OTHER PROJECTS

5.1 IJCNLP paper

After my assignments and projects, I worked with a PhD student in the same lab I was working to write a research paper which was improving state-of-the-art of argument convincingness in the world of NLP. Compare to the author of the paper I had limited knowledge of NLP. So, my first task was to provide the accurate dataset for the models to run on. The idea here was to compare some hand picked arguments with some random Wikipedia arguments. I was assigned with the task to provide the latter. In order to achieve that, I used the Wikipedia dump I used during my previous projects. I used gensim library which would extract the corpus from the dump in a format where every line of the resulting corpus would correspond to one Wikipedia article. Then the resulting corpus was used to select 30 random Wikipedia articles which would be used as argument pairs to provide the insights as proposed in the paper for argument convincingness.

My second task assigned was to build one of the models out of the three proposed ones in the paper. In simple words, the task was to get the Wikipedia similarity (section 3.2 in the original paper) Potash et al [7] which was simply a dot product of representation of a random Wikipedia argument and a hand picked argument on a specific topic. The result would give us the Wikipedia similarity score. The higher score would mean that the argument pair is more convincing. Although I could not get the model properly working, Peter Potash (the first author of the paper) helped me and we did get the model running.

My third task was to run the six models with the various argument pairs. The argument pair included articles from the Wikipedia dump and hand picked articles. The results were either models run with the term frequency representation or the different embeddings used during the experiment. The results were plugged in Table 2 of the original paper. Along with this I helped Peter write the paper itself. Since this submission was running short on time, it required long hours of work at a time to get it done.

```

program = os.path.basename(sys.argv[0])
logger = logging.getLogger(program)
logging.basicConfig(format='%(asctime)s: %(levelname)s: %(message)s')
logging.root.setLevel(level=logging.INFO)
logger.info("currently running %s" % ' '.join(sys.argv))

# check and process input arguments
if len(sys.argv) != 3:
    print("You should be using the following format: python this_filename.py enwiki.xxx.xml.bz2 filename.text")
    sys.exit(1)
ip, op = sys.argv[1:3]
space = " "
i = 0

output = open(op, 'w', encode="utf-8")
wiki = WikiCorpus(ip, lemmatize=False, dictionary={}) # lemmatize is made false, in order to avoid a format which slows the
for text in wiki.get_texts():
    output.write(b' '.join(text).decode('utf-8') + '\n')
    i = i + 1
    if (i % 10000 == 0):
        logger.info("Currently saved " + str(i) + " articles")

output.close()
logger.info("Total saving finished of : " + str(i) + " articles! ")
return sys.argv[2]

```

Figure 11: This is a portion of the script which is used to get the Wikipedia corpus from the dump. It saves in a format of one article per sentence and uses gensim do to that.

What did I learn from this experience?

This was a whole new experience for me. The entire process of running full length experiments with different models was new to me. I learned how to build big scale models and to work with big datasets. This not only polished my existing skills, but also gave me a lot of confidence to work with big datasets. Although there is a lot more to learn, this was a really fun experience.

5.2 Binary Classifier on customer complaints dataset

An organization reached out to our professor with a dataset of customer complaints. The goal was that, the person involved wanted to submit a dataset paper to the LREC conference this year which is why they wanted to know if, the dataset could be used on it or not. So, the professor assigned me the task to write a model (a binary classifier) which could tell us how feasible the data is.

5.2.1 The model

So the data had three separate attributes where I ran binary classifiers. The attributes were of each complaint mentioned in the dataset; namely, is the complaint valid?, is the argument valid?, and is the complaint misrepresenting?

Due to the varied size of the complaints ranging from 200 words to 1400 words which is why I decided to build a model with LSTM, using a continuous-bag-of-words representations to make the data usable. In order to get the embeddings I used facebook's fastText library, which given a corpus gets you the cbow embeddings of the corpus. The reason I used this library instead of coding the algorithm for cbow is there was a shortage of time. Once I had the embeddings I used Keras' Sequential LSTM model with the proper hyperparameters to make the model work.

Other than LSTM, upon professors advice I also built a classifier using an SVM to get further detailed results. For an SVM I used a linear kernel and used sklearn package to get the results. So, in the end I had two different kinds of binary classifier in two different ways.

5.2.2 Experiments and Results

Given the fact that the dataset only had 400 labeled instances, getting proper results was ensured only by a good model. As far as experiments are concerned, there was two different kinds of classifiers that I used and got reasonable results for both. For LSTM, I tried using different percent of data for validation and got the results as show in Table 2.

Other than LSTM, I also used an SVM with linear kernel where I got a **validation score of 0.51** and the **cross-validation score ranged from 0.48 to 0.58**

I am not sure of these are the optimal results for the dataset or not. I would like to try out different kinds of models in future which I think will give a better insight on the dataset.

```

-181000 eviction rate=0.498753 and unsatisfied allocation rate=0
2017-10-01 12:11:52.620408: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 188709 get requests, put_count=382656 evicted_count
=191000 eviction rate=0.499143 and unsatisfied allocation rate=0
2017-10-01 12:11:52.744612: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 198432 get requests, put_count=403375 evicted_count
=201000 eviction rate=0.499529 and unsatisfied allocation rate=0
2017-10-01 12:11:52.869507: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 208153 get requests, put_count=422100 evicted_count
=211000 eviction rate=0.499882 and unsatisfied allocation rate=0
2017-10-01 12:11:52.994818: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 217801 get requests, put_count=441848 evicted_count
=221000 eviction rate=0.500172 and unsatisfied allocation rate=0
2017-10-01 12:11:53.127902: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 227625 get requests, put_count=461572 evicted_count
=231000 eviction rate=0.500464 and unsatisfied allocation rate=0
2017-10-01 12:11:53.261682: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 237248 get requests, put_count=481295 evicted_count
=241000 eviction rate=0.500732 and unsatisfied allocation rate=0
2017-10-01 12:11:53.395772: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 247069 get requests, put_count=501016 evicted_count
=251000 eviction rate=0.500982 and unsatisfied allocation rate=0
2017-10-01 12:11:53.528457: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 256818 get requests, put_count=520745 evicted_count
=261000 eviction rate=0.501186 and unsatisfied allocation rate=0
2017-10-01 12:11:53.663743: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 266543 get requests, put_count=540490 evicted_count
=271000 eviction rate=0.501397 and unsatisfied allocation rate=0
2017-10-01 12:11:53.796588: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 276265 get requests, put_count=560212 evicted_count
=281000 eviction rate=0.501596 and unsatisfied allocation rate=0
2017-10-01 12:11:53.941666: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 285985 get requests, put_count=579932 evicted_count
=291000 eviction rate=0.501783 and unsatisfied allocation rate=0
2017-10-01 12:11:54.069147: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 295737 get requests, put_count=599684 evicted_count
=301000 eviction rate=0.501931 and unsatisfied allocation rate=0
2017-10-01 12:11:54.196522: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 305460 get requests, put_count=619407 evicted_count
=311000 eviction rate=0.502093 and unsatisfied allocation rate=0
2017-10-01 12:11:54.323701: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 315181 get requests, put_count=639128 evicted_count
=321000 eviction rate=0.502247 and unsatisfied allocation rate=0
2017-10-01 12:11:54.452080: I tensorflow/core/common_runtime/gpu/pool_allocator.
cc:247) PoolAllocator: After 324910 get requests, put_count=658857 evicted_count
360/360 [=====] - 88s - loss: 0.7589 - acc: 0.6694 - va
1 loss: 0.8421 - val acc: 0.5366
(NeuralEnv) robin@ishkur:~$

```

Figure 12: This is one of the screen-shot while running the model with LSTM with validation data 10 percent of the total data.

Percentage of data used for validation	Training Accuracy	Test Accuracy
10	66.94	53
15	62.41	54.1
20	67.48	51.1

Table 2: This table shows the relation between the increasing amount of data being used for validation and the changing training and validation accuracy, while being trained with a LSTM

6 FUTURE WORK

As a NLP enthusiast, I would like to continue learning more about NLP with every passing day. Although, there would be some more work that I would like to do in future. For the word2vec model project, in future I would like to implement cbow embedding of the corpus. I did get a chance when I was implementing the classifier using the customer complaints dataset, but I used fastText for that. So, that is something I would like to implement in future. Other than that, for the sequential autoencoder project I would like to implement a variational autoencoder of my own in future. I am really fascinated by Alexa, so coding my own conversational agent is something I would want to do. As far as the binary classifier projects are concerned, as mentioned before I would like to implement various models in order to be sure that I did get an optimal result from my current models from that dataset. Apart from these existing projects, I personally want to work on a model which is trained with human moral values. My personal aim is to come up with an AI which is smart enough to be a kindergarten teacher (toddlers are taught basic moral values along with basic alphabets and basic mathematics). With products like Anki's Cozmo, I think that is something that is going to be very useful.

7 CONCLUSION

In conclusion I would like to mention that this course taught me more than I would have learned through a normal NLP class. Continuous guidance of Anna combined with my enthusiasm towards NLP helped me learn what I did. I have a clear understanding of concepts like RNN, LSTM and most importantly sequence autoencoders. On a personal level, I would recommend anyone who is interested in NLP to take this course with Professor Anna Rumshisky.

References

- [1] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.’s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.
- [2] David W Hosmer Jr, Stanley Lemeshow, and Rodney X Sturdivant. *Applied logistic regression*, volume 398. John Wiley & Sons, 2013.
- [3] Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.
- [4] Tomáš Kočiský, Gábor Melis, Edward Grefenstette, Chris Dyer, Wang Ling, Phil Blunsom, and Karl Moritz Hermann. Semantic parsing with semi-supervised sequential autoencoders. *arXiv preprint arXiv:1609.09315*, 2016.
- [5] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernocký, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.
- [6] Fuchun Peng, Dale Schuurmans, Shaojun Wang, and Vlado Keselj. Language independent authorship attribution using character level language models. In *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics- Volume 1*, pages 267–274. Association for Computational Linguistics, 2003.
- [7] Peter Potash, Robin Bhattacharya, and Anna Rumshisky. Length, interchangeability, and external knowledge: Observations from predicting argument convincingness. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, volume 1, pages 342–351, 2017.
- [8] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. Lstm neural networks for language modeling. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.