Date of acceptance Grade

Instructor

Msc. Thesis Semantic text similarity using autoencoders

Nikola Mandic

Helsinki August 13, 2018

UNIVERSITY OF HELSINKI
Department of Computer Science

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Computer Science

Tekijä — Författare — Author
Nikola Mandic

Työn nimi — Arbetets titel — Title

Msc. Thesis Semantic text similarity using autoencoders

Oppiaine — Läroämne — Subject
Computer Science

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	August 13, 2018	44 pages + 83 appendices

Tiivistelmä — Referat — Abstract

Word vectors have become corner stone of modern NLP. Researchers are taking embedding ever further by learning to craft embedding vectors with task specific semantics to power wide array of applications. In this thesis we apply simple feed forward network and stacked LSTM on triplets dataset converted to sentence embeddings to evaluate paragraph semantic text similarity. We explore how to leverage existing state of the art sentence embeddings for paragraph semantic text similarity and examine information sentence embeddings used hold[DOL15, LM14].

ACM Computing Classification System (CCS):
A.1 [Introductory and Survey],
I.7.m [Document and text processing]

Avainsanat — Nyckelord — Keywords
layout, summary, list of references

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction **1**

1.1 Why is this thesis useful? . . . 3

1.2 Thesis outline . . . 4

1.3 Overview of theoretical descriptions in this paper . . . 5

1.4 Overview of action points in the thesis . . . 5

1.5 Overview of data preparation . . . 6

1.6 Models overview . . . 6

1.7 Experiments overview . . . 6

2 Related work **7**

2.1 LSTM . . . 8

2.2 USE . . . 11

2.2.1 DAN . . . 13

2.2.2 Transformer network . . . 14

2.3 ElMo . . . 17

3 Data preparation **18**

3.1 Overview . . . 18

3.2 Processing protocol . . . 18

3.3 Processing . . . 18

4 Model **20**

4.1 LSTM model . . . 20

5 Experiments and results **22**

5.1 Experiment 1 . . . 23

5.1.1 Preparation . . . 23

5.1.2 Training . . . 23

5.1.3 Result . . . 24

5.1.4 Discussion 24
5.2 Experiment 1.2 24
5.2.1 Preparation 24
5.2.2 Training 24
5.2.3 Result 24
5.2.4 Discussion 26
5.3 Takeaway from experiment 1 28
5.4 Experiment 2 28
5.4.1 Result 29
5.4.2 Discussion 29
5.5 Experiment 2.1 30
5.5.1 Result 31
5.5.2 Discussion 31
5.6 Experiment 2.2 34
5.6.1 Result 35
5.6.2 Discussion 35

6 Conclusion **39**

References **40**

A creating wikipedia index on disk **44**

Appendices **44**

A Wikipedia cleaning 44
B Experiments 49

1 Introduction

Thesis tries to explore if we use sentence embeddings produced by Universal Sentence Encoder [CYK+18] for paragraph text similarity task. We take triplets dataset to make ground truth dataset [DOL15]. Triplets dataset contains set of triplets. Triplet is list of 3 links that are similar. We mine Wikipedia for these texts and create ground truth dataset by setting similar label if two paragraphs are in triplet and use negative sampling for negative labels. We use binary cross entropy loss to evaluate models. We ask simple questions in the thesis. Can we use simple feed forward neural network for this task and can we benefit more from using LSTM? We also wonder if we use stacked LSTM do we have extra benefit if using output lower levels of LSTM as features. If lower level of stacked LSTM for word vectors capture syntactic information while higher level capture semantic, is there similar analogy for sentence vectors?

With ever growing amount of data on the Internet, machine learning results in unsupervised area of machine learning are gaining more importance due to cost tied to getting labeled data. Researchers and industry for long time have wanted to make use of wast amounts of data by using unsupervised methods and relying on data quantity alone. Promises of such approaches can have a positive impact in society if we could harness wast amounts of data publicly available which would be expensive to label on large scale. As technology continues to be intertwined with society ever more, increasing data generated from all types of sources could be valuable if we could develop algorithms that take advantage of that data alone. This potential has driven the surge of research in academia and industry in field of unsupervised machine learning whose goal is to help make this a reality.

One fruit of this work that we consume every day is Google translate which in field of natural language processing revolutionized how industry approaches translation. With successfully using vector embeddings of natural text to capture semantic information and then converting these vector embeddings into words in various languages we see one of prime examples demonstrating just how much usability these methods can deliver and information that could be captured from data by an algorithm alone. Initial research done on word vectors by Tomas Mikolov that powered new version of Google translate created interest in developing new types of word vectors which encode different information. These encodings are produced with neural networks by encoding words as vectors with neural networks called autoencoders.

Simplified description of autoencoder is that it is neural network with same type of data on it's input and output. Example would be transferring word representations x into some latent space vector z

$$z = W * x \tag{1}$$

Translating latent space representation z into result y that can be word in different language.

$$y = W * z \tag{2}$$

Above equations are simple linear equations but this transformation is most often non linear in a form of a deep neural network. See Figure 1. Same words in embedded space can have close distance across languages due to nature of human language. So distance between queen and king embedding in English can be the same as in French.

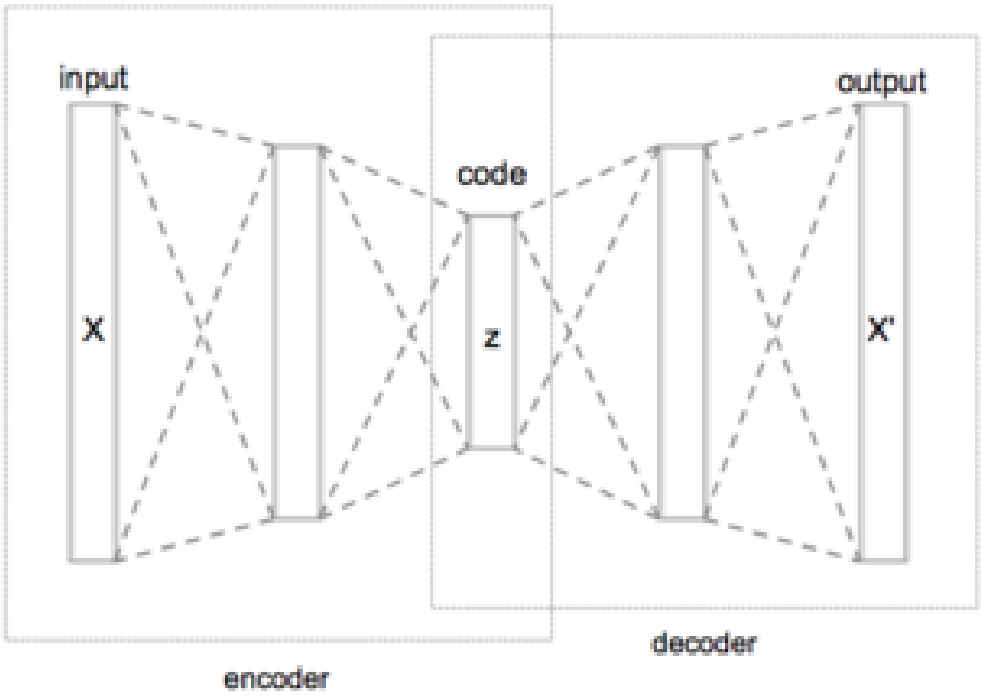


Figure 1: Autoencoder structure(taken from wikipedia)

If we have wast amounts of data now it becomes clear why autoencoders are a good tool if they can produce value. We only need to supply data we collected to them

without any extra labor like labeling data. The hidden representation depending on algorithm used can encode different types of information. Loss function is often crafted so that embeddings produced capture specific information.

1.1 Why is this thesis useful?

Besides being written to demonstrate authors mastery of the subject, one might ask is there any other use in experiments we're about to read and results we're about to see? Since author of this thesis tries to attain machine learning mastery one of goals is for use not just to work on technical fluency but to attain craftsmanship through practices, techniques and methods that form state of the art and yield tangible results. Although sometimes these techniques capture just spirit of our time never the less they are of crucial importance for aspiring practitioner.

Besides improving technical fluency and demonstrating mastery there is also added value in finding out more about embeddings themselves since if somebody tried them out I could not find work that would be exact as in this thesis which explores practicalities related to modern state of the art sentence vector embeddings and how semantics they capture can be utilized. NLP is wast area and this direction seemed not that much sought after since specific sentence embeddings are relatively new.

Often we see researchers benchmarking and exploring vector embeddings for information embeddings encode since it is not always obvious to analytically know, from methods used to obtain embeddings, what information is captured by specific embedding and for what tasks can it be useful. In this work we try to explore sentence embeddings, which are relatively new and haven't been as benchmarked as word embeddings.

Having even simple technique tried and tested on modern embeddings gives us more info about characteristics of vectors that we are about to discuss. It is challenging to analytically determine can stacked LSTM be useful for specific task if we expose lower layers of stacked LSTM as feature to feed forward network on top? We can run experiments to try to statistically determine if information captured by these vectors can indeed be used in particular way. Experiments like that brings us closer to understanding embeddings better and managing them. Since embeddings became corner stone of modern NLP it is important to understand them in order to use them for solving various NLP problems or for applying them in industry related applications.

1.2 Thesis outline

Related work section of the thesis starts with work that seems very similar to what was tried in this thesis and summaries of theory in modern and popular NLP trends that influenced and inspired work conducted in this thesis. After introduction, work that was used as foundation for thesis in terms of data and work that looks similar is described, descriptions of more modern work is given related to how embeddings used in the thesis are produced. Universal sentence encoder is described.

First description is given of special type of neural networks called LSTM networks and summaries are given of some of top results in recent years that advanced state of the art when it comes to specific area of NLP that deals with vector embeddings. From these papers summaries crucial concepts are described, with which paper contributes to NLP community, and that inspired this work to experiment with thoughts these scientists lay out to the community. Summary of modern work on vector embeddings is done to highlight more the area of NLP that became foundation for solving many NLP problems today. Researchers have explored crafting embeddings of different characteristics and when we describe techniques used by state of the art research in the field we survey part of the theoretical background and material needed once thesis starts to discuss experiments that rely on modern work while exploring exciting area of NLP that deals with vector embeddings.

Data preparation section speaks about substantial work that was done in scope of this thesis to prepare data. Since the thesis was done over several months large portion of time went into preparing and managing data. The work included finding Wikipedia dumps, post processing them, extracting appropriate data from Wikipedia, experimenting with optimal ways to clean the data, experimenting with ways of embedding data in vector space that is reasonable on home computer, loading the data into clean pandas data frames fit for experiments chosen to be conducted while doing this in way that is computationally reasonable on home computer. During this work I published few blog posts that were referenced in the thesis related to process of manipulating Wikipedia data in cost effective way. One of the key takeaways is that by using command line data science tricks we don't have to clean data and load it into a database.

Models section describes model structure used. This description is foundation for models used in experiments although throughout experiments there are variations of basic description, variations all look similar. Model in experiment 2 has its structure picked based on authors intuition grounded in modest machine learning experience,

although model might be not optimal due to having two stacked LSTM's for two paragraphs it is in my opinion reasonable since it looks similar to what community uses often, like usage of fully connected network on top of LSTM output.

Experiments section then goes on to describe results related to conducting experiments with purpose of exploring how we can use these embeddings for simple task of text similarity. First experiment uses simple model and second uses LSTM based model. Experiments describe variations of the tried model and experiences from trying out different variations. There is discussion section for each experiment trying to provide intuition on model variation and result.

Appendices contain parts of code that was used for this thesis. Since work on thesis was ongoing for longer period of time and often I had to do things differently due to hardware constraints code is not saved in order of execution as things were done ad hoc in Jypiter notebook but crucial parts of code that actually did the work can be seen like for example function that fetches the data from disk, functions that clean wikimedia markdown, functions that parse xml, attempts at writing parser for wikimedia markdown, that did work quite well but I picked other solution that looked more tested.

1.3 Overview of theoretical descriptions in this paper

Master thesis starts with theoretical description that describe how sentence embeddings used in this thesis are obtained what is theory behind producing them and why are these modern embeddings so significant today. Then theoretical description moves towards ideas in ElMo embeddings paper that inspired experiments to an extent. ElMo paper has very interesting technique employed to produce state of the art word embeddings. Can we make use of this technique for sentence embeddings?

1.4 Overview of action points in the thesis

- get wikipedia dump and triplets dataset
- extract triplets out of wikipedia
- clean it so that it is in pandas dataframe and tokenized
- implement testbed for experiments

- implement model
- modify model in small variations and report results

1.5 Overview of data preparation

Ground truth dataset is prepared by taking triplets dataset and Wikipedia dump. We construct dataset that consists of paragraph pairs as input and label that is a flag indicating if paragraphs are similar. Since triplets dataset gives us information if two paragraphs are similar if two paragraphs are in one triplet in triplets dataset we can put label indicating they are similar. We can pick two random paragraphs from triplets dataset that are not in the same triplet to get paragraphs that are not similar. Dataset obtained in this way is ground truth dataset and there are approximately 50% of similar paragraphs while other half comes from negative sampling. After splitting dataset obtained in before mentioned fashion into test and train datasets binary cross entropy loss is used to evaluate models.

Data preparation part involves external software being used and command line techniques in order to manage data without having to use database. Wikipedia dump is processed in command line to create index of titles on disk and then command line tools are used to retrieve articles. Gensim open source software is used to clean data. Also I tried of writing parser on my own and tried various other libraries but in the end most popular one was used to parse wikimedia format.

1.6 Models overview

There are two models tried. First one is fully connected network and second one is stacked LSTM with fully connected layer on top. Fully connected network is chosen for simplicity and LSTM based model is chosen to experiment on how well does stacking LSTM work when it comes to paragraphs and sentences.

1.7 Experiments overview

First experiments start out with simple model then each model is varied until better results are obtained. Modern deep learning techniques are applied. Variations are testing out various assumptions and at the end each experiment results are reported and discussion tries to provide intuition.

2 Related work

Rapid progress in the field of machine learning hasn't left autoencoders without fair share of innovation. Researchers are embedding units of text with more and more information, semantic and syntactic. Growing the domain of applications and pushing state of the art. Plenty of modern methods and tools are now available for researchers to build on. Methods for embedding different units of text like paragraph, sentence and words are being developed. Embeddings they produce encode variety of information that is useful for problems that machine learning aims to solve.

Autoencoders have been gaining traction recently in industry applications perhaps most notably with Google translate using sentence embedding vectors to power it's translation service. Skip-thought vectors[MCCD13b] have become increasingly prevalent as means of encoding word semantics into a vector space paving path for new research that led to creation of different types of word embeddings[GPH+16b] that could have different amount of information captured [AKB+16b]. This work enables further NLP research and industry applications by providing methods to capture language semantics together with other information and provide solid abstractions for further development. It is these abstractions that enabled new research in sentiment analysis and other areas. GloVe[PSM14], Word2Vec[MCCD13a, MSC+13], FastText[BGJM16], ElMo [PNI+18] are just some of word embedding names that have gained traction in the field. These vectors encode semantic relationships once trained on large corpora of text such as Wikipedia and often constitute basis for further work. Researchers have for some time went beyond just word embeddings but used unsupervised learning to expose latent features of sentence and paragraph embeddings.

Ever since [KZS+15] published skip-thought vectors there has been plenty of work taking advantage of autoencoders to produce vector embeddings of various units of text.

Work done on paragraph similarity on triplets dataset [DOL15] inspired most of the work for this thesis. What is of interest in the paper is that it references hand built triplets dataset that was rarity at the time of this writing. Ground truth dataset dataset in the thesis is produced based on triplets dataset. Alternative to it would be scraping search engine results as early work of Mr. Mikolov does [LM14] to evaluate semantic text similarity. Another alternative although seems less suitable is 'News Aggregator Data Set' from [DKT17].

Usually sentence and paragraph embedding methods rely on word embeddings and there have been plenty of them recently. Different word vectors encode different information [AKB+16a]. There are many of word embedding flavors like fastText [BGJM16] from Facebook or the one from Microsoft research that allows for engineering of semantics that is to be embedded into vector besides learning general purpose representations with unsupervised approach [GPH+16b]. Most notable recent work that provided improvements across the board of nlp tasks is ElMo word vector embeddings [PNI+18] that encodes various semantic and syntactic information that aims to be general enough to provide improvement on many NLP tasks that use word vectors as basis for solutions provided.

There is plenty of work on embedding paragraphs and sentences. Sentence encoding has seen more exposure with universal sentence and paragraph embedder [CYK+18] being provided on tensorflow hub.

Recent work coming from Facebook [WFC+17] provided method for embedding entities among which are paragraphs and sentences but with accent for domain specific tasks. Although method mentions that it could be used for semantic text similarly it does not focus on this task in particular nor provides more elaboration that would provide richness of embedding information as previously mentioned papers.

Recently while this thesis was in progress a work was published on github using similar model for text summarization[don]. Author demonstrates mastery in building on top of work readily available in tensorflow library and this work will also try to look up to this work.

While I was writing this thesis another work showed up closely related to this where authors evaluate different sentence embeddings for downstream tasks[PSP18].

Embeddings are hot commodity in modern NLP works and we can hope they rise to the glory of buzzwords like big data and blockchain.

2.1 LSTM

LSTM stands for Long Short Term Memory. Term is used in context of neural networks and is used to qualify cell term in order to describe regime of operation of a neural network building block.

Unlike neuron, simple building block of neural networks, LSTM cell is building block that has more parameters and is more complex. Diagram of a LSTM can be seen

in Figure 2.

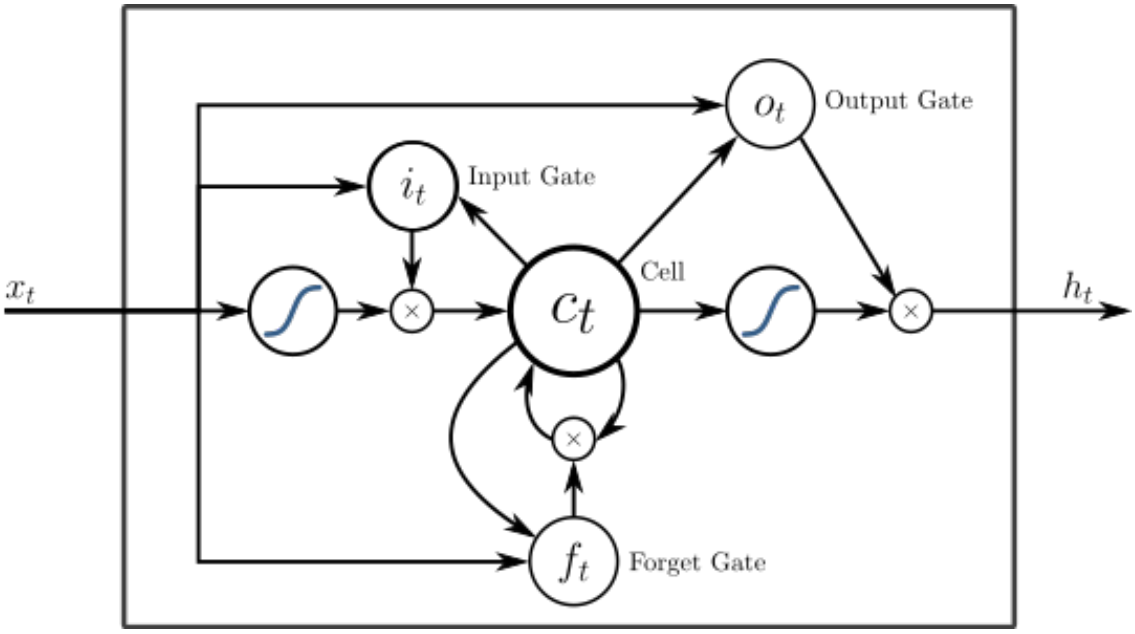


Figure 2: Autoencoder structure(taken from wikipedia)

Contrasted to neuron besides input it has internal state refereed to as C.

Regime of operation can be described in few equations that govern how internal state c and output h is calculated[Wik18].

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$
$$c_t = f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c)$$
$$h_t = o_t \circ \sigma_h(c_t)$$

Equations above correspond to circles in the Figure 2. referred to as gates. Gates are linear transformations of input through matrices assigned to each gate which are learned parameters of LSTM.

We could say that input gate transforms current input vector and previous cell output which is also part of input. That is difference to simple neuron as it has only one input. Here LSTM cell takes as input previous cell output besides current input vector and this is discussed in LSTM terminology as timestamp. Once cell takes as input previous cell state it takes as input previous timestamp.

Output is named h_t often referred to as hidden state and index t signifies point of time as in other parts of LSTM diagram. Output is calculated as function of output gate,o_t, and previous state.

Internal state is calculated as function of previous state, input gate,input and previous output.

Intuition related to these elements usually follows components of the diagram. Internal state is used as memory(memory part of LSTM acronym). Forget gate governs when to reset and when to keep effect of this state on the output and on next state. It determines how long or short will this memory be kept by linear transformation dependent on the input and previous output.

Forget gate can be imagined as learned parameter that governs what word in a sentence has influence on this current output if input would be word vector for example.

LSTM networks have gained popularity recently due to their successful applications across many tasks that benefited memory contributions for sequential calculations that LSTM offer. Time series data and NLP have in particular benefited this method and for long time LSTMs were dominant model but recently as we'll go over in next sections these models are being surpassed by more modern models that are simpler but more effective. Universal sentence encoder does not use LSTMs but

produces state of the art embeddings. Before LSTM based models were considered state of the art. When describing internals of Universal sentence encoder principles behind modern techniques will be described that powered modern results avoiding computationally expensive LSTMs completely.

2.2 USE

Universal sentence encoder is an example of application of modern models that surpass LSTM based models that held state of the art results for some time. Universal sentence encoder or USE is a term used to describe two methods employed by authors to produce sentence embeddings. One method is Deep averaging networks[IMBgI] and other one is transformer architecture[VSP+17]. They both are methods that use model simpler and less computationally challenging like LSTM. For this thesis DAN version will be used due to practical reasons. It was available out of the box to be loaded via tensorflow like in Figure 3 . Universal sentence encoder set new

```
import tensorflow_hub as hub

embed = hub.Module("https://tfhub.dev/google/"
    "universal-sentence-encoder/1")

embedding = embed([
    "The quick brown fox jumps over the lazy dog."])
```

Figure 3: Tensorflow usage(taken from original paper)

state of the art on STS benchmark[CDA+17].

USE uses both of these methods to encode sentence into 512 length vector. Main reason why these sentence embeddings are useful is to achieve better results as they might encode more information useful for various models that use these embeddings to try to solve different NLP problems. So far alternative has been to either use various word level embeddings and train model based on them or produce sentence level embeddings to be used for a task. Usually information captured in word level embeddings can be thought of as means of transfer learning. Big unlabeled datasets that are easily available are used to obtain word embeddings in unsupervised fashion and information they capture is then transfered to other tasks. Sentence level embeddings produced by two methods USE uses aim to provide exact same building

block for researchers to use for solving different NLP problems and they are done with purpose of improving the yield of machine learning algorithms by providing richer amount of information encoded.

What is notable and very valuable is that USE delivers on above mentioned tasks without using computationally expensive algorithms based on LSTM cells. Not only do these methods provide better results but they do it more elegantly and at lower computational cost.

Authors state in original USE paper that they picked these two approaches due to their different design goals and trade offs they offer. DAN offers less accuracy but with advantage of being less computationally expensive while Transformer network has higher accuracy and requires more computational power. In this way authors offer flexibility with advanced state of the art methods making USE a valuable contribution in NLP world with increased potential for broad range of applications.

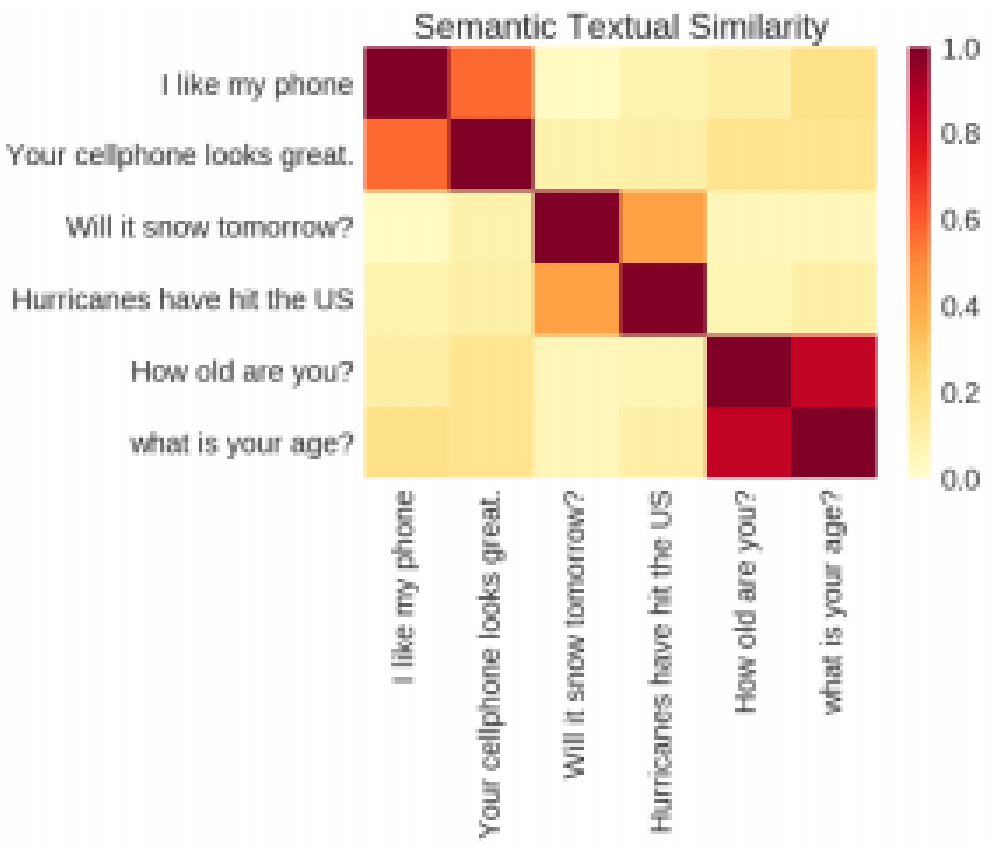


Figure 4: USE usage(taken from original paper)

Figure 4 displays matrix of cosine distance between different sentences, demonstrating that these vectors embed sentences in space so that cosine distance becomes effective measure of semantic similarity, achieving state of the art on STS benchmark.

2.2.1 DAN

DAN stands for deep averaging network[IMBgI]. If we look at original paper for deep averaging network we can see that original authors provide brief summary of the method as:

1. average word embeddings of input sentence
2. pass average value through feed one or more layers of feed forward network
3. use last layer for classification

Basic diagram can be found in original authors paper as displayed in Figure 5.

Although USE authors mention bi-grams being averaged in their paper, when they describe DANs, we can use original paper authors summary to describe mode of operation of DANs.

We see in the figure that simple deep neural network is what suffices for this encoder. Intuition behind could be summarized as few linear transformation build nonlinear transformation and, like kernel trick, transfer original network into latent space we train for. We can reason about it analogous to reasoning for kernel based methods and can assume that properties of data that can be linearly separable are found with this network like with trainable kernel.

It is fast, efficient, can take advantage of GPU contrasted to LSTM and its simplicity makes it elegant and attractive as method that comes among top state of the art results. This result seems to illustrate the amount of information underlying word vectors compress since this result drives state of the art for sentence embeddings.

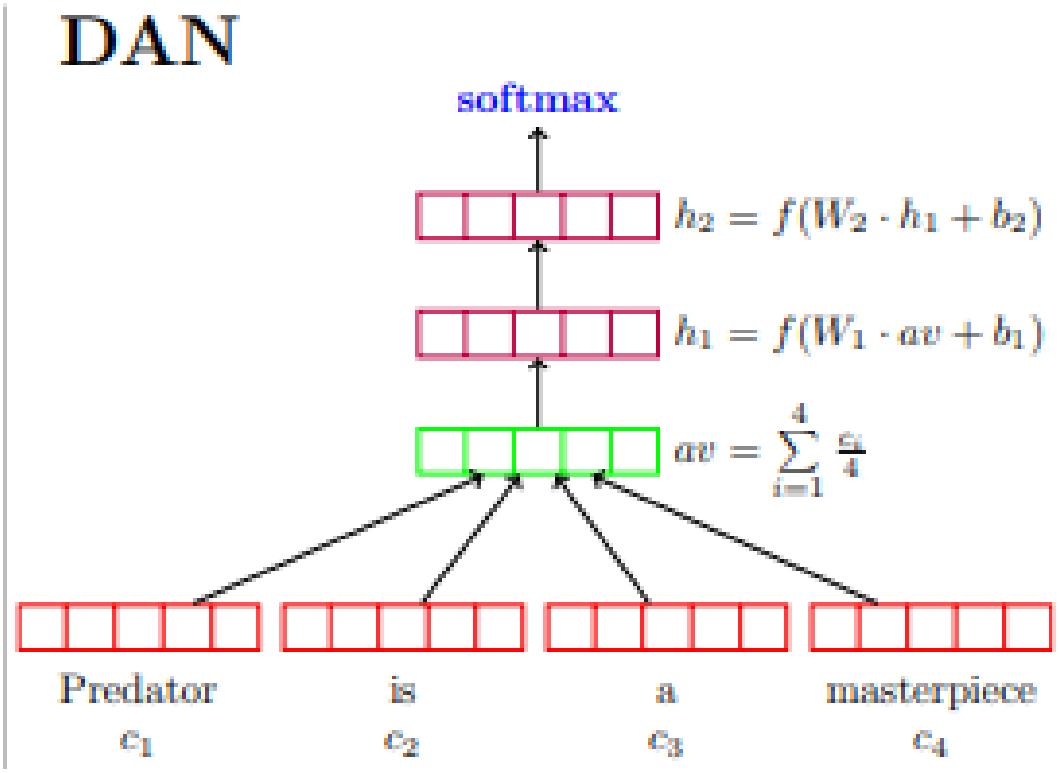


Figure 5: DAN structure(taken from original paper)

2.2.2 Transformer network

Transformer network recently gained a lot of Internet coverage in paper Attention is all you need[VSP+17]. One of good explanations of this architecture that tries to convey intuition really good is in youtube tutorial[Kil].

Figure 6 displays architecture of this network. As we see in the picture there is no trace of LSTM cells. There are just feed forward layers that form bulk of neural units. What makes this architecture stand out is that it can compete with LSTM when it comes to capturing dependencies between different part of sentences.

LSTMs capture different information on different levels. If we take stacked LSTM as an example we can expect lower layers to capture syntactic information while higher layers to capture semantics as we will see latter when ElMo embeddings[PNI+18]

are described.

The way this network manages to compete with LSTM is by having attention mechanism between encoder and decoder. This mechanism is what is responsible for learning what parts of sentence should model pay attention to in order to learn relevant language model.

Encoder consists of 6 layers as original authors say. Each of these has two sub layers if we follow authors terminology. Second layer is simple neural network layer while first one is multi head attention layer. This layer we can intuitively describe as parallel querying of input with set of keys K. We can imagine individual query as applying function whose input is input of encoder and one key in key set K. This gives us some vector. Now if we apply this function with n keys and sum all of these outputs we get output of multi head attention vector layer. This can be perceived as weighted sum where function we apply is the weight. Each key tells where to look in the input. It is the attention mechanism. If we query by multiple keys it is multi head attention. This is implemented as matrix multiplication. Output is normalized. We can now have more intuition why this can compete with LSTM. Like LSTM capture where should we pay attention in original sentence by learning gate matrices. This vector K effectively does the same by weighing inputs. Multiple vectors K increase this effect and learn more. Now there are multiple layers stacked one on top of each other in order to have even more capacity to capture syntax and semantic information.

Authors take note from computer vision where residual connections were introduced as operation that makes error propagate faster to lower layers and shortening feedback loop thus enabling faster and more efficient learning. They employ residual connections across layers to have it train faster and that lover layers don't get stuck at one set of weights if they are not yielding any result.

Second part of network is decoder. What is interesting is that decoder takes entire output that was generated until point in time in order to have it as input besides vectors generated by encoder. We see that in Figure 6. at the bottom where output besides input is fed into decoder. Result that decoder gives is a language model. We pick word with highest probability and then we can feed it together with previous words again to input of decoder.

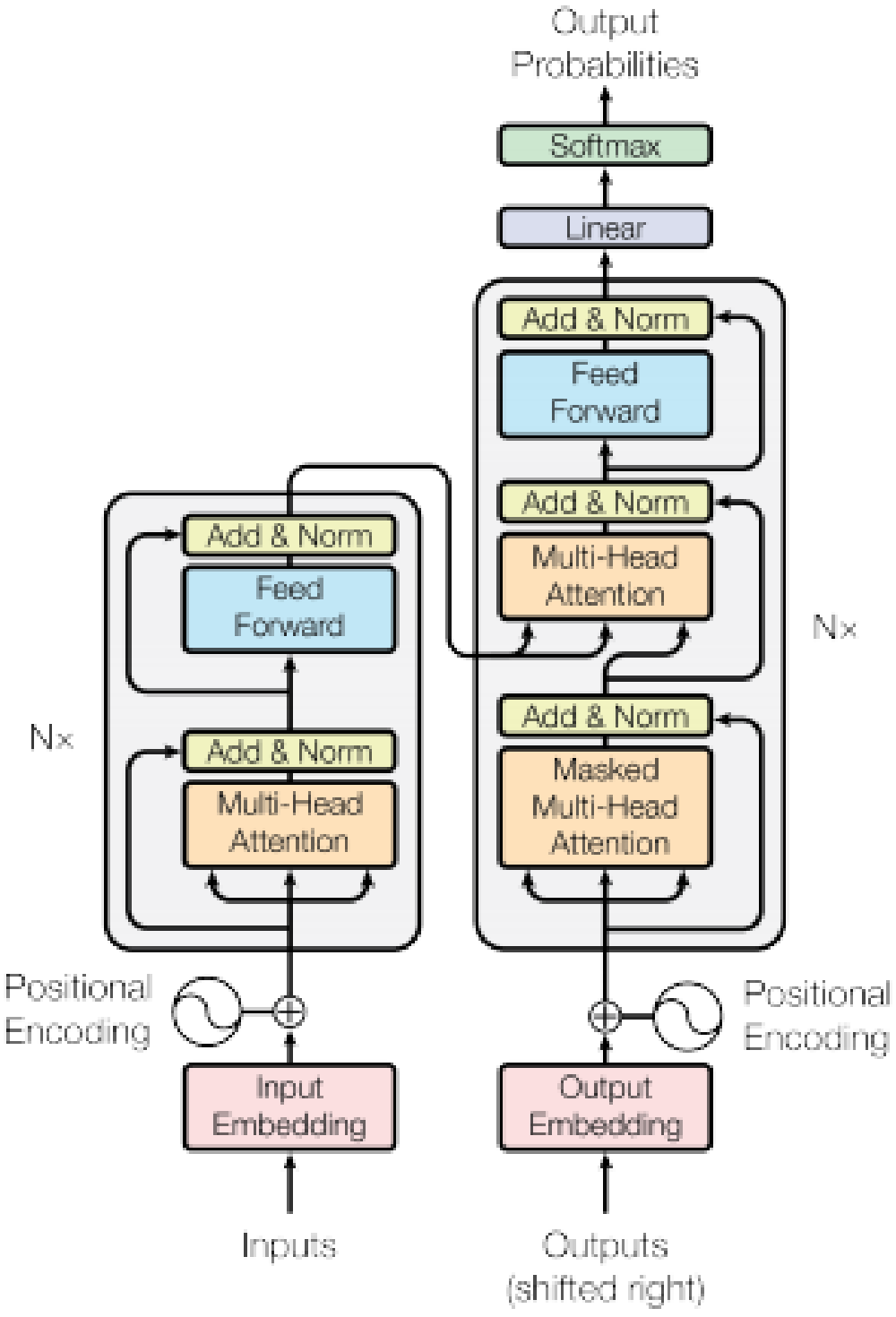


Figure 6: Transformer structure(taken from original paper)

2.3 ElMo

ElMo stands for Embeddings from Language Models[PNI+18]. Authors improve state of the art results across the board of different NLP problems that use word embeddings. ElMo encodings are produced with bidirectional stacked LSTM that has language model as objective. That means that bidirectional LSTM tries to predict word based on other words in the sentence and since it is bidirectional it tries to maximize probability of a word not just by previous words but those that came after. Since conditional probability of a word p_x is maximized given all other words $p_{x-1}, .., p_0$ we call this type of objective language model objective.

Authors claim that their representation captures syntactic and semantic meaning of a word as well as polysemy since final output that represents encoding is a weighted sum of all LSTM layers and word is function of all other words in a sentence.

Lower layers of LSTM can capture syntactic meaning while higher levels can capture semantic meaning and once we have weighted sum of those then we convey both of these informations. Authors also use character convolutions on word level.

While summing the vectors authors use hyperparameter to determine scale so that task specific training can be applied to improve results. Also batch normalization is used between the LSTM layers.

What is of interest about ElMo paper is that stacked LSTM is used and that is to be tried as one of the experiments in this thesis.

3 Data preparation

3.1 Overview

Triplets dataset is freely available dataset referenced in Document Embedding with Paragraph Vectors paper[DOL15]. Dataset consists of triplets of wikipedia links. There are 20k triplets. It tree links that represent similar articles first link and second are more similar than second and third.

3.2 Processing protocol

Wikipedia dump[Wik] is downloaded in media wiki format. Media wiki format is internal wikipedia xml format that encodes metadata on top article content. This wikipedia dump in media wiki format is prepared by making index of it according to this blob post[tym].

There are multiple open source tools for parsing the content of media wiki like for example $mwparserfromhell$. During preparations for this thesis I even developed custom parser using $pyparsing$ python package. With this package you can declaratively specify grammars and media wiki format is small. There are also examples online of people parsing media wiki so its a waste of time to develop this parser on one's own because if other people invested more time in perfecting edge case handling better to use that one.

Gensim[ŘS10] library provides methods to do exact same thing but they are not very advertised. This library is in widespread use and I assumed that it handles better potential edge cases when it comes parsing markdown in mediawiki format.

3.3 Processing

For task of this thesis uses just one triplets file from original dataset, specifically file wikipedia_2014_09_27_examples.txt which is not hand produced but automatically.

Wikipedia dump is mined according to blog post on tymbac.tech website[tym]. Gnu parallel[Tan11] is used together with ripgrep to search entire wikipedia dump almost instantly with simple file system based index that is made with few command lines. For article title as input we get entire wikimedia markup for an article. Wikipedia

dump that was used in this work is of never date than in original paper. Some articles are missing and some are redirected. Python code on top of shell script follows redirects. If what is retrieved with shell script returns redirect then shell script will be invoked again to fetch an article mentioned in redirect. Then Gensim function for parsing is applied to extract text from markdown. After text is retrieved spacy[HM17] python library is used for sentence parsing. Stop words, whitespace and punctuation are removed while words are lowercased. For all processing with spacy library 'en_core_web_sm' model is used.

Result of above postprocessing is saved to dataframe and then fed through universal sentence encoder[CYK+18].

These sentences in vector form are fed into lstm model.

Task that is to be tried out is binary classification determining if two texts are similar or not. So data in triplet form is converted for this purpose. If two texts are in the triplet that pair is marked with positive label. Negative pooling is used to determine negative label.

Data is then split into train test and validation datasets. Training set is 80% of data, test and validation datasets are 10% each.f

In similar fashion instead of piping data through universal sentence encoder data is passed through ElMo [PNI+18] encoder. Embeddings that are output of it are word embeddings instead sentence embeddings.

4 Model

4.1 LSTM model

Model used is multilayer LSTM that has dense layer on top. It was implemented in TensorFlow and latter implementation uses Keras. Model is implemented in multiple variations so actually this can be interpreted as different similar models benchmarked and their parameters tuned. Model consists of two layers of stacked LSTM network and several dense layers on top are added.

There are two layers of LSTM for each paragraph so in total 4 LSTM layers. That might seem like model might learn same thing twice or we can look at it as ensemble of two. Anyway it is good to see how it works although it looks unusual especially since this is master thesis and goal is demonstrating mastery of the topic.

Then two final hidden states of top LSTM is concatenated and fed into several dense layers. Batch normalization is done between them to learn better. This feed forward network is also varied across attempts to increase performance and in general the bigger the better up to around size of concatenated vector on input. First layer is biggest then following layers size decreases by factor of two until final layer that has only one neuron for classification.

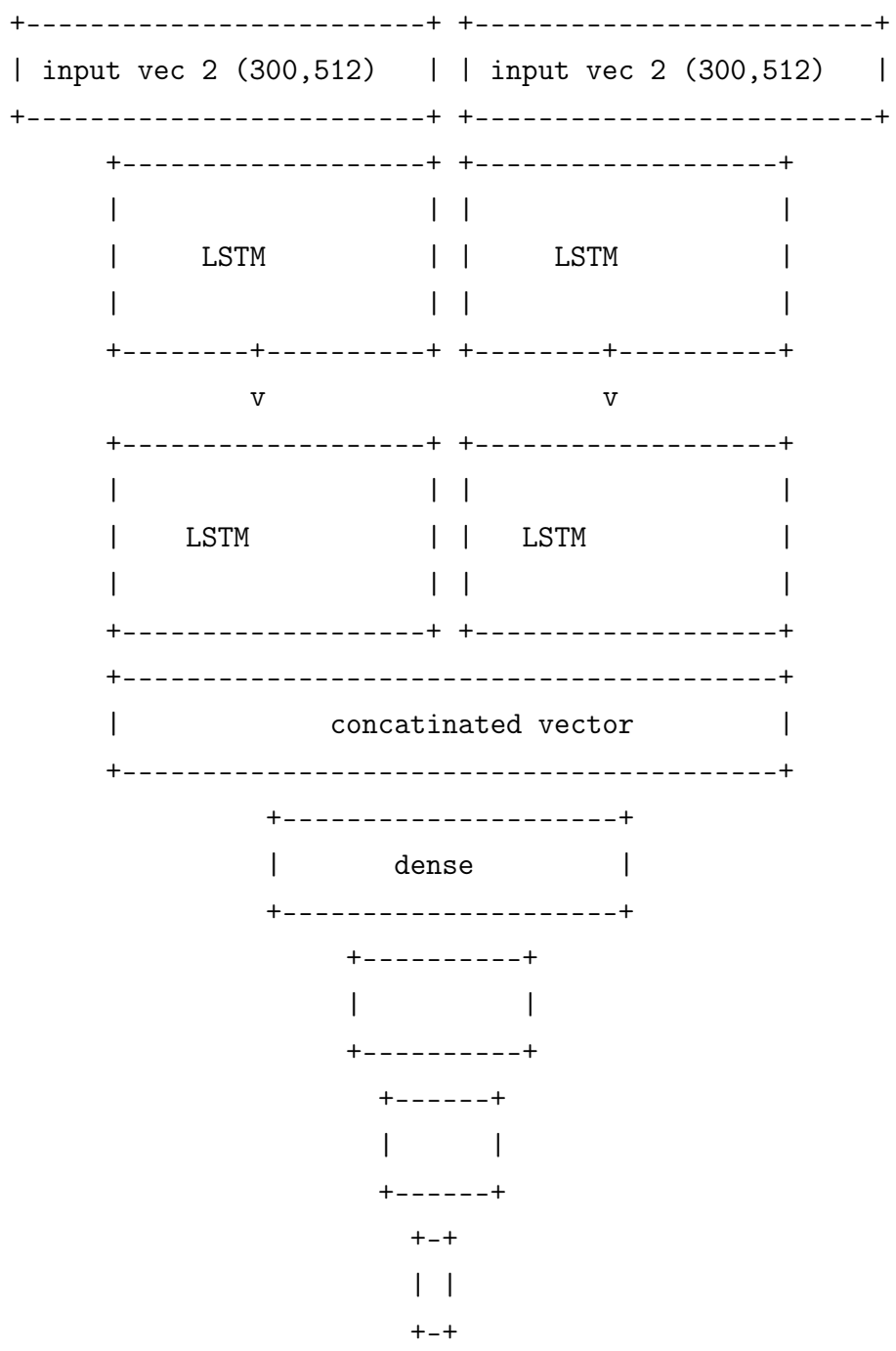


Figure 7: Model 1 structure and training

5 Experiments and results

Original dataset is shuffled pseudo randomly and similar documents are taken from this shuffled dataset. Negative sampling is used to get not similar documents.

Network is implemented as tensorflow estimator for first experiment. Tensorflow version used is 1.8 and data is passed through dataset tensorflow api before being fed into estimator.

For other experiments keras is used.

For Keras experiments in experiment 2.2 seeds of random number generator are controlled more like written in Jason Brownlee's post[Jas]. Although due to GPU present it does not remove randomness so simulations are used.

For all experiments same dataset is used with same negative sampling. Since dataset is split and picked with random number generator fixed seeds are used to make selection always the same.

After experiment is done a result and discussion follows it. Result section tends to be focused on reporting numbers and performance while discussion tends to provide some intuition.

5.1 Experiment 1

5.1.1 Preparation

First experiment will involve simple model for start. Paragraphs after having each sentence converted with universal sentence encoder will have each paragraph averaged. Average of sentence vectors of a document will be input to multi layer feed forward neural network with one neuron at the end of the model used to signal similarity. Sentence vector has length of 512 and since we take average it will also be of length 512. There are two documents to be compared so there are two vectors of length 512 as input.

Model concatenates these two vectors and has feed forward network on top that has single neuron as indicator of similarity.

5.1.2 Training

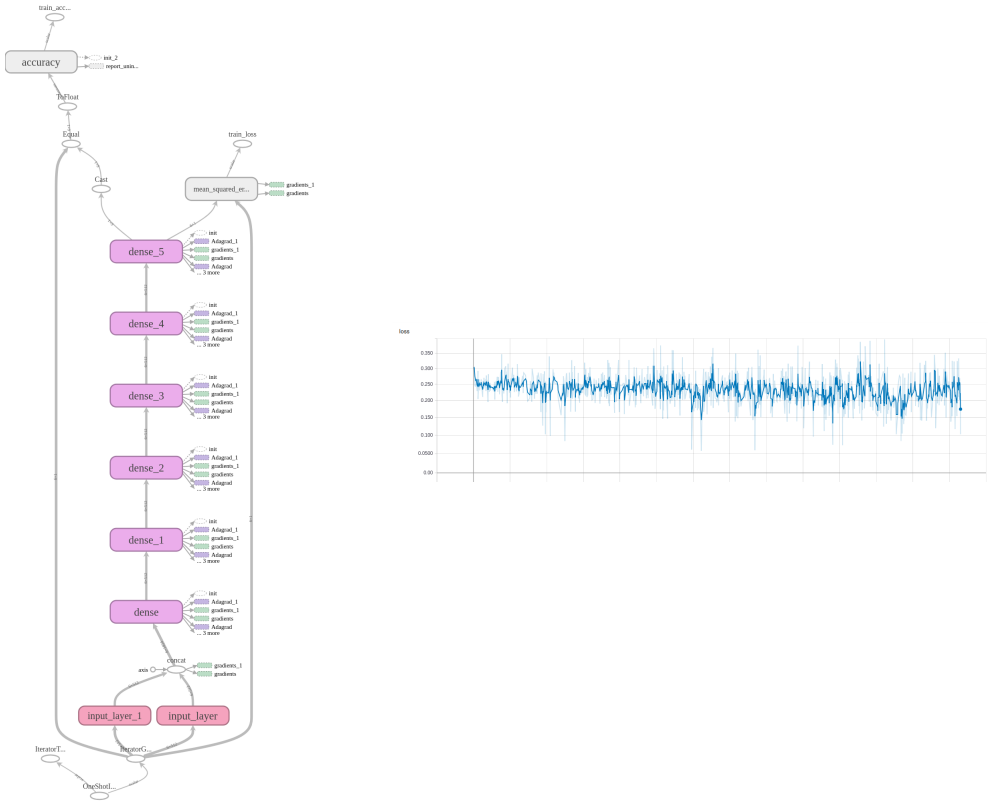


Figure 8: Model 1 structure and training

5.1.3 Result

As we see model did not learn anything. It is slightly better than random.

5.1.4 Discussion

This model was implemented in tensorflow as estimator it might be that technical complexity contributed to failure too besides having no proper initialization and batch normalization.

5.2 Experiment 1.2

5.2.1 Preparation

Continuing from data prepared for experiment 1 keras library will be used for convenience. Model is simple feed forward network

5.2.2 Training

On Figure 8 we see model definition and hyper parameters selected. For first variation we'll use stochastic gradient descent as optimizer and relu activation. Learning rate was selected based on few runs. Model consists of nonlinear transformation of two average vectors concatenated. Average vectors are average of all sentence vector in paragraph. Dropout layers are used with higher rate at bottom layers and they make the model train like ensemble. It is not good to have higher than 0.5 dropout as it will quickly make model useless.

5.2.3 Result

This model yields 74% on test set

With variation like in Figure 9 that implements some deep learning tricks like Xavier initialization[GB] and batch normalization and uses different optimizer one is able to raise accuracy on the test set to 78% in just 5 epochs while result above took 45 after which model breaks.

```
input1 = keras.layers.Input(shape=(512,))
input2 = keras.layers.Input(shape=(512,))
added = keras.layers.Concatenate(axis=-1)([input1, input2])
x3 = keras.layers.Dense(128, activation='relu')(added)
x3=keras.layers.Dropout(0.5)(x3)
x3 = keras.layers.Dense(128, activation='relu')(x3)
x3=keras.layers.Dropout(0.1)(x3)
x3 = keras.layers.Dense(32, activation='relu')(x3)
x3=keras.layers.Dropout(0.1)(x3)
x3 = keras.layers.Dense(32, activation='relu')(x3)
out = keras.layers.Dense(1)(x3)
model = keras.models.Model(inputs=[input1, input2], outputs=out)
```

```
sgd = optimizers.SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
```

```
model.compile(loss='binary_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])
```

```
tbCallBack = keras.callbacks.TensorBoard(log_dir='/data/logs', histogram_freq=0, write_graph=True, write_images=True)
```

```
model.fit([np.array([vsent2avg(x[0]) for x in X_train]),np.array([vsent2avg(x[1]) for x in X_train])], Y_train,
          epochs=45,
          batch_size=25, callbacks=[tbCallBack])
```

Figure 9: Model 1 structure and training

```
input1 = keras.layers.Input(shape=(512,))
input2 = keras.layers.Input(shape=(512,))
added = keras.layers.Concatenate(axis=-1)([input1, input2])
x3=keras.layers.BatchNormalization(axis=-1)(added)
x3 = keras.layers.Dense(128, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.5)(x3)
x3 = keras.layers.Dense(128, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.5)(x3)
x3 = keras.layers.Dense(32,kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.1)(x3)
x3 = keras.layers.Dense(32, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
out = keras.layers.Dense(1)(x3)
model = keras.models.Model(inputs=[input1, input2], outputs=out)
```

```
sgd = optimizers.SGD(lr=0.00001, decay=1e-6, momentum=0.9, nesterov=True)
nadam = optimizers.Nadam()
adadelta=keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=0.0)
```

```
model.compile(loss='binary_crossentropy',
              #optimizer=sgd,
              optimizer=adadelta,
              metrics=['accuracy'])
```

```
model.fit([np.array([vsent2avg(x[0]) for x in X_train]),np.array([vsent2avg(x[1]) for x in X_train])], Y_train,
          epochs=5,
          batch_size=25, callbacks=[tbCallBack])
```

```
scores = model.evaluate([np.array([vsent2avg(x[0]) for x in X_test]),np.array([vsent2avg(x[1]) for x in X_test])], Y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
1998/1998 [==============================] - 1s 284us/step

acc: 80.28%
```

```
scores = model.evaluate([np.array([vsent2avg(x[0]) for x in X_train]),np.array([vsent2avg(x[1]) for x in X_train])], Y_train)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
15969/15969 [==============================] - 1s 32us/step

acc: 78.32%
```

Figure 10: Model 1 structure and training

Notable thing is that wrong parameters and methods can ruin this model completely

in terms of what it learns while deep learning methods actually make it even go further than 80.28%. When using learning rate of 0.5 and 25 epochs it gets to 80.98% With 50 epochs it goes to 81.88%. Selu activation function turned out to work best and just after 10 epochs with learning rate of 0.5 can get 81.78%.

With little bit more neurons and more parameter tweaking we can get to 88% in 10 epochs

```
input1 = keras.layers.Input(shape=(512,))
input2 = keras.layers.Input(shape=(512,))
added = keras.layers.Concatenate(axis=-1)([input1, input2])
x3=keras.layers.BatchNormalization(axis=-1)(added)
x3 = keras.layers.Dense(512, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.5)(x3)
x3 = keras.layers.Dense(128, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.5)(x3)
x3 = keras.layers.Dense(64,kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.2)(x3)
x3 = keras.layers.Dense(16, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
out = keras.layers.Dense(1)(x3)
model = keras.models.Model(inputs=[input1, input2], outputs=out)
```

```
sgd = optimizers.SGD(lr=0.00001, decay=1e-6, momentum=0.9, nesterov=True)
nadam = optimizers.Nadam()
adadelta=keras.optimizers.Adadelta(lr=1.0, rho=0.95, epsilon=None, decay=1e-6)
```

```
model.compile(loss='binary_crossentropy',
              #optimizer=sgd,
              optimizer=adadelta,
              metrics=['accuracy'])
```

```
model.fit([np.array([vsent2avg(x[0]) for x in X_train]),np.array([vsent2avg(x[1]) for x in X_train])], Y_train,
          epochs=10,
          batch_size=10, callbacks=[tbCallBack])
```

```
scores = model.evaluate([np.array([vsent2avg(x[0]) for x in X_test]),np.array([vsent2avg(x[1]) for x in X_test])], Y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

```
1998/1998 [==============================] - 0s 114us/step

acc: 88.39%
```

Figure 11: Model 1 structure and training

With another 10 epochs this model does not yield more.

5.2.4 Discussion

Model is implemented in keras. Deep learning practices were actually what made the result happen. Learning rate could be high.

Intuition around batch normalization[bno] and its usage can be perceived to be of technical nature related to learning. What is argued in before mentioned article is that with batch normalization we introduce to neural network concept similar to a prior in Bayesian models. We are not fitting anymore all parameters of previous layers but we are introducing normal prior on them that makes observations for

following layer be part of normal distribution. So we are fitting two parameters of batch normalization layer instead of relying solely on updating previous layer weights. If we update previous layer weights article argues that we change input distribution to next layer to much for the next layer to be effective in memorizing what it learned. After updates propagate this can cause staling in learning and optimizing. Our model would not learn anything if it is deep network just due to the fact that these internal distributions keep changing once we update weights. By normalization we put priors on internal layers and learning propagates through these two parameters which is considerably less than if layer has 1k neurons which would count 1k parameters.

Intuition behind initialization trick is that if we don't have extremes in the weights we would not end up in not solvable extremum of optimization space.

5.3 Takeaway from experiment 1

What is important takeaway from experiment 1 is that these embeddings truly encode information sufficient to provide semantic similarity information on paragraph level. It is not that much of value for this thesis to try to sharpen initial feed forward model until it provides higher and higher yield but demonstration that encodings have this potential is of value for next experiment where LSTMs will be tried. If simple feed forward network with relatively small number of neurons provides high yield it would be interesting to see what computationally more complex and powerful LSTM would do(under the assumption that authors ineptitude does not produce failure). We have seen that we can get a lot of information just from deep learning network on top of average of sentence vectors in a paragraph. These vectors seem to capture information good enough to be used for paragraph semantic text similarity in very simple way by just training feed forward network on top of two vectors. We also seen that without proper deep learning techniques model will not only perform badly but often break and not train at all, advancements in deep learning are make or break for even simplest of tasks and parameter changes that might look small give highly different results.

5.4 Experiment 2

For second experiment LSTM will be used. We'll start out with random model that does not look good actually. It has two LSTM heads for each paragraph that seems redundant and those LSTM parts are stacked LSTMs. After that feed forward neural network from previous experiment copied as is.

There are 100 timestamps per sample and LSTM has dimension of 6. Last timestamp in a sample is average of entire paragraph.

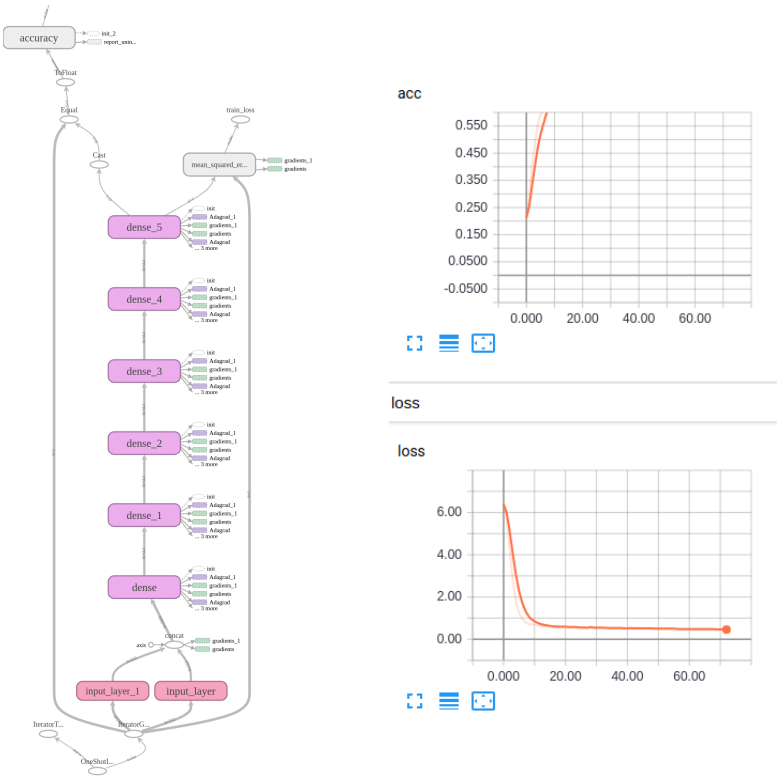


Figure 12: Model 1 structure and training

5.4.1 Result

This model achieves around 80% accuracy at best on test set on over fitted model it can drop to 70. It is worse than just feed forward network alone. Given that output of stacked LSTM is just vector of length 6 and when we concatenate two of those as input to feed forward neural network that has 128 neurons on start layer it does not seem that bad since it is initial attempt and not tuned much. It seems good that from just 12 length vector we can get such result and from 100 sample length. It is not good that it performed worse then just feed forward network. When we swap first layer to have 128 neurons. When we increase number of timestamps to 300 we get 83% accuracy on test set. with 26 epochs.

5.4.2 Discussion

Due to size of the model this experiment is slightly more lengthy to train and due to LSTM more computationally expensive. Although it under performs fully connected layers it is still quite good. It could have failed to produce any result but LSTMs did let signal propagate to further layers with output vector of just 6.

Previous experiment had similar neural network that concatenated two 512 average networks. Lets say that LSTM chose to pick last vector in sample that is average and should contain most information still having compression to just 6 instead of 512 is to note.

5.5 Experiment 2.1

In this variation we set dim1 to 60 instead of 6.

```
X1 = keras.layers.LSTM(dim1,
                       stateful=stateful,
                       kernel_initializer=init,
                       recurrent_initializer=init,
                       #recurrent_activation='selu',
                       activation='selu'
                      )(lstm1)

#X2=keras.layers.BatchNormalization(axis=-1)(input2)
X2=input2
lstm2, state_h2, state_c2  = keras.layers.LSTM(dim1,
                       return_sequences=True,
                       return_state=True,
                       stateful=stateful,
                       kernel_initializer=init,
                       recurrent_initializer=init,
                       #recurrent_activation='selu',
                       activation='selu'
                      )(X2)
#X2 = keras.layers.LSTM(dim1)(input2)
#X2 = keras.layers.LSTM(dim1, return_sequences=True)(X2)

#X2=keras.layers.BatchNormalization(axis=-1)(X2)
X2 = keras.layers.LSTM(dim1,
                       stateful=stateful,
                       kernel_initializer=init,
                       recurrent_initializer=init,
                       #recurrent_activation='selu',
                       activation='selu'
                      )(lstm2)
```

Figure 13: Model 2 structure and training

5.5.1 Result

Model under performs significantly if it is made to work at all. Performance is in terms of 30% - 40% on train set.

5.5.2 Discussion

Training is very unstable. When using optimizers from previous models model is not trainable. It is too big and learning rate is too high. If we drop learning rate and play with fully connected layer at the top we can somehow with optimizers like rmsprop with momentum get it to train poorly. At best after 100 epochs it will get to slightly above 40% accuracy. In order to get it to train at all we need to use rmsprop optimizer and low learning rate that is just big enough not to get stuck in any minor local optima. One of most obvious things that come to mind is that state vector is not appropriate to sentence length. If this state vector is to tell us what to keep and what to drop in sentence if it is too large network might not only be to big but might struggle to learn.

Lets try to explore what happened by looking at distribution of sentence length. Figure 13 shows distribution of sentence length in only one column of links. If there are 3 link columns we should be safe if we think one is representative. They should all look the same they are just paragraphs from Wikipedia.

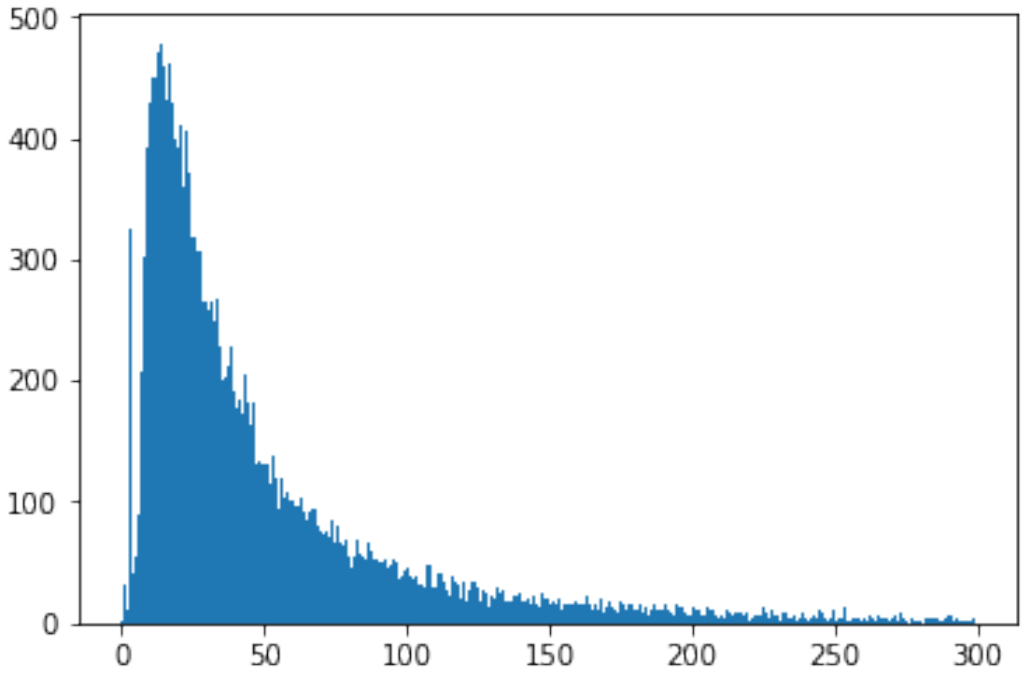


Figure 14: Histogram of sentence length

As we see sentences are small in length on average. Mean is 59 and median is 33 while mode is 13. Lets now try to zoom in on Figure 14. We see that most sentences are small in length so having vector of 60 to remember which ones are important or not is too much.

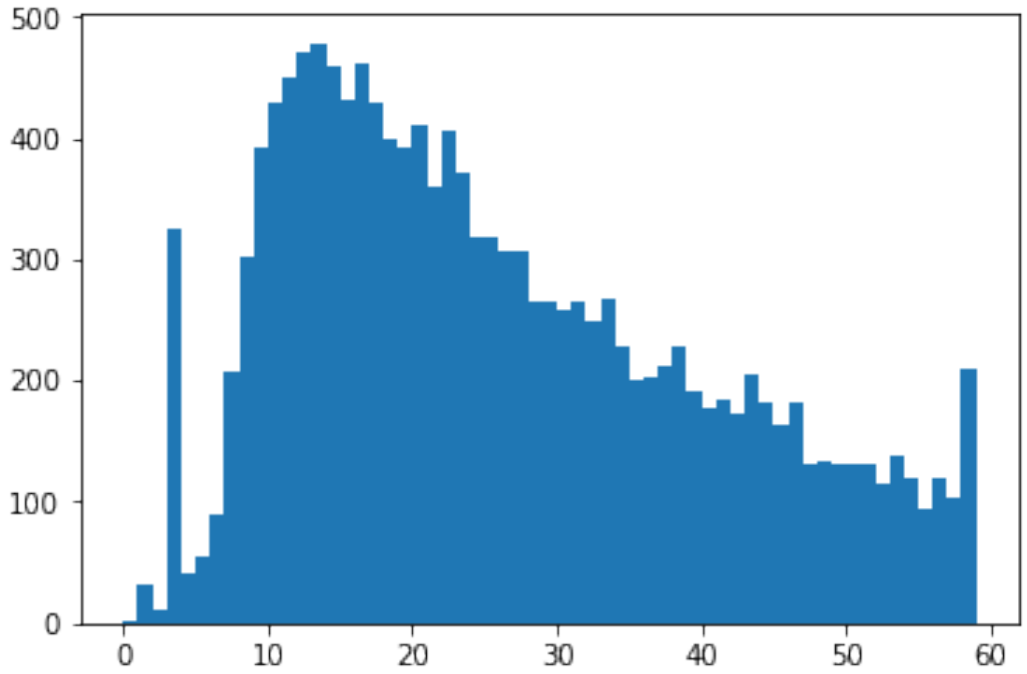


Figure 15: Histogram of sentence length zoomed

If we try to fit normal distribution in Figure 15 we see that it does not look nice there is this concentration of sentence lengths in low numbers and standard deviation is high.

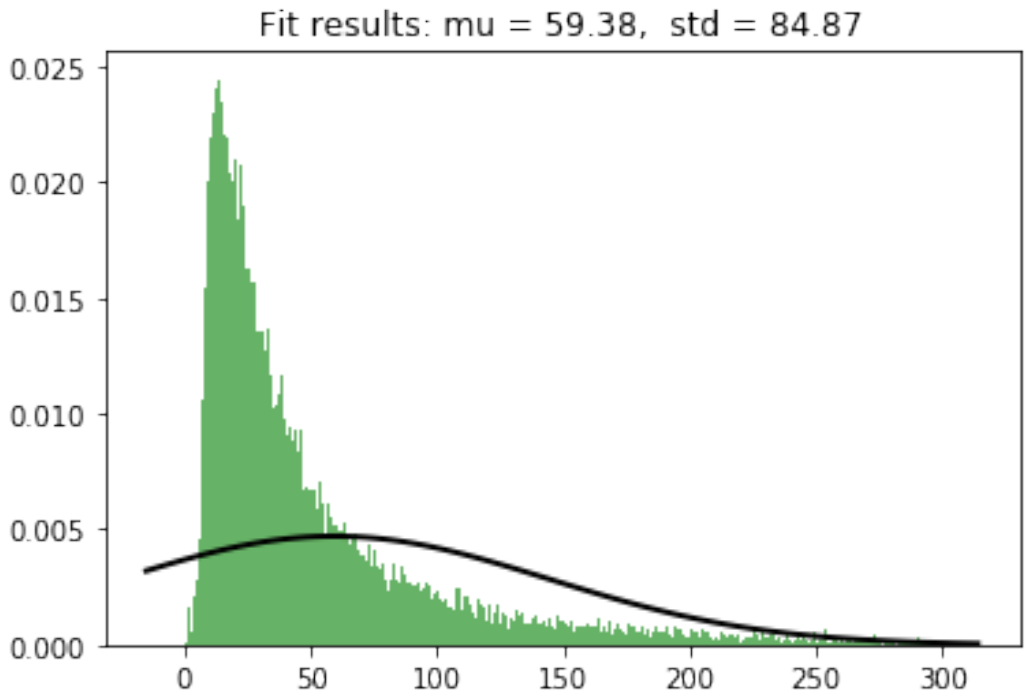


Figure 16: Histogram of sentence length zoomed

When having big state vector exposing lower level LSTM states for feed forward network has negative effect.

With learning rate used in past successful models loss goes to nan very soon if not right away and at some learning rates it will be stuck. To get it to progress before breakage very careful parameter picks are needed. We can go on to hypothesize without further checks that most probably big state just fails to generalize properly if long tail of sentence length distribution causes the optimizer to flip flop between what sentences are important since tail is substantial in size although mass is concentrated further down. Although this is just intuitive guess.

If we have two layers on fully connected part one 32 in front and another 16 right after it learning after 26 epochs is 10% less

5.6 Experiment 2.2

Lets try to see is there anything in that hypothesis from experiment 2.1 by increasing LSTM dimension. It would be interesting to experiment with this parameter to try to find more optimal solution. Feed forward network on top is displayed in Figure 16.

```
#added1 = keras.layers.Concatenate(axis=-1)([X1,state_h1,state_c1, X2,state_h2,state_c2])
#added2 = keras.layers.Concatenate(axis=-1)([X1,state_h1,state_c1, X2,state_h2,state_c2])
added = keras.layers.Concatenate(axis=-1)([X1, X2])
x3=added
#x3=keras.layers.BatchNormalization(axis=-1)(added)
x3 = keras.layers.Dense(512, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.5)(x3)
x3 = keras.layers.Dense(128, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.3)(x3)
x3 = keras.layers.Dense(64,kernel_initializer=init, activation='selu')(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.2)(x3)
x3 = keras.layers.Dense(32, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dense(16, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
out = keras.layers.Dense(1)(x3)
print(out)
model = keras.models.Model(inputs=[input1, input2], outputs=out)
```

Figure 17: Histogram of sentence length zoomed

5.6.1 Result

When we increase dimension to 10 from 6 we see sudden surge in learning rate now in just 6 epochs we get to 77% accuracy as shown in Figure 17.

```
size=1000
size=X_train.shape[0]
model.fit_generator(gen([*zip(X_train[0:size],Y_train[0:size])],batch_size)(),steps_per_epoch=int(size/batch_size),
          epochs=100, callbacks=[tbCallBack])
```

```
Epoch 1/100
532/532 [==============================] - 99s 185ms/step - loss: 2.5832 - acc: 0.4786
Epoch 2/100
532/532 [==============================] - 96s 180ms/step - loss: 0.6930 - acc: 0.6189
Epoch 3/100
532/532 [==============================] - 96s 180ms/step - loss: 0.6245 - acc: 0.7021
Epoch 4/100
532/532 [==============================] - 96s 181ms/step - loss: 0.5777 - acc: 0.7444
Epoch 5/100
532/532 [==============================] - 96s 181ms/step - loss: 0.5556 - acc: 0.7567
Epoch 6/100
436/532 [=======================>......] - ETA: 17s - loss: 0.5292 - acc: 0.7748
```

Figure 18: Histogram of sentence length zoomed

5.6.2 Discussion

Model is not stable initialization makes big difference. Setting random seeds through the model and setting random numpy seed including tensorflow seed still leaves model not reproducible most probably due to training on GPU. Going from 6 to 10 in LSTM dimension gave only imaginary improvement in example above. Initialization has huge impact on stability and training and dropout layers do to due to large drop rate set in the model. It is 0.5 percent in first that can yield totally different ensemble. For some seeds 6 will outperform 10 and for others 10 will outperform 6 in this model. More controlled environment is needed to determine impact of this

dimension. To estimate better we'll have to run experiment multiple times to get better picture of what is going on.

After running 20 simulations where 3 different variations are benchmarked for fixed seed here are results for

- dimension of 6 for LSTM and without exposing hidden state of lower layers
- dimension of 10 for LSTM and without exposing hidden state of lower layers
- dimension of 10 for LSTM and with exposing hidden state of lower layers

  These charts display fitted normal distribution on accuracy after each model was trained 20 epochs for 20 times. Accuracy is on x axis.

  On Figure 18. we see first attempt with 6 as dimension of LSTM layer and accuracy is quite stable at average of 0.80494949 and standard deviation relatively low at 0.023993939

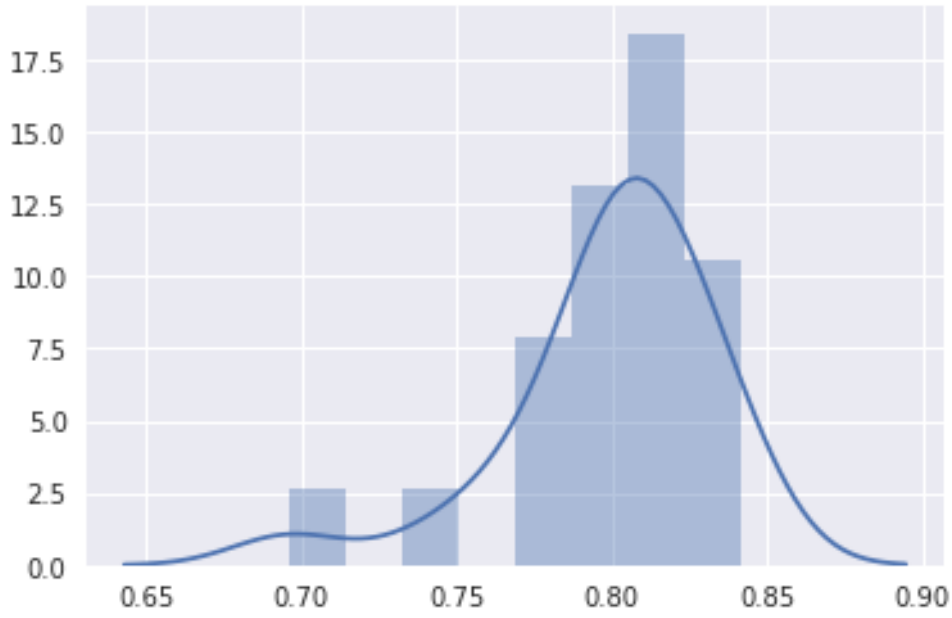


Figure 19: LSTM dimension 6 and hidden states of lower layer not exposed

When we increase dimension to 10 as in Figure 19. we see that standard deviation increases to 0.033474071 while mean drops to 0.79973545. This is very minor difference after 20 simulations.

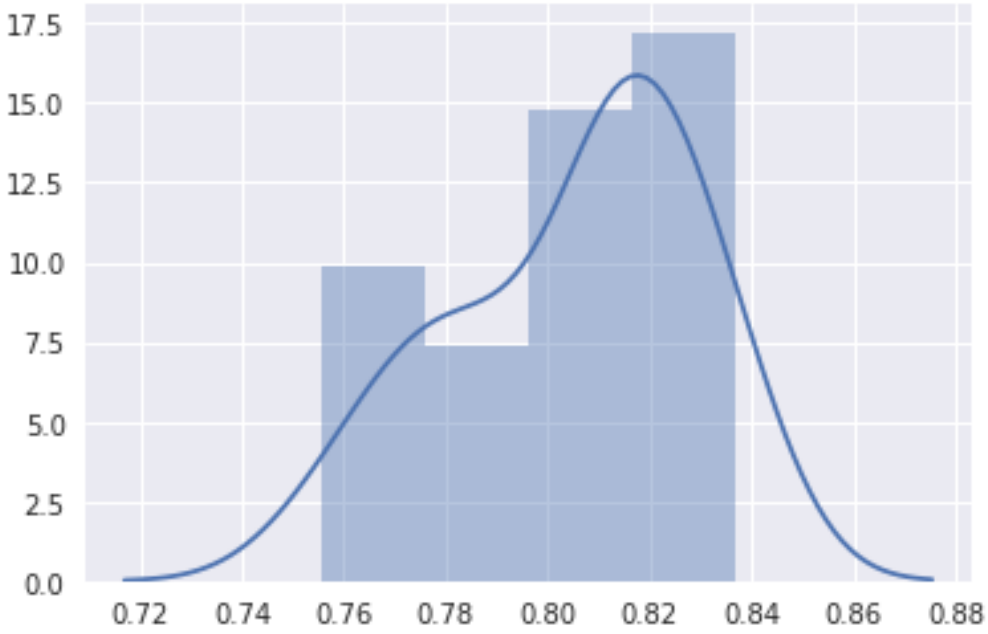


Figure 20: LSTM dimension 10 and hidden states of lower layer not exposed

After exposing hidden state of lower level LSTM in Figure 20 we see that performance increases on average to 0.80722222 with standard deviation of 0.021451334.

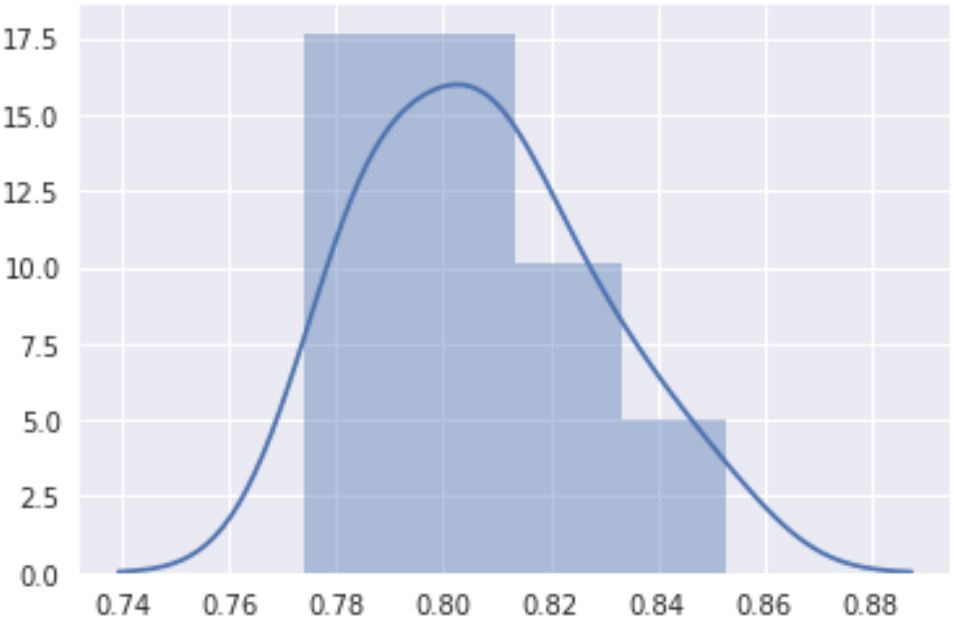


Figure 21: LSTM dimension 10 and hidden states of lower layer exposed

We see that model where we exposed hidden states of lower LSTM layers with increased dimension of 10 for LSTM layer performed best even though going from 6 to 10 layers decreased accuracy on average. It seems to show that exposing lower layers works analog to the same trick used in ElMo paper[PNI+18] where exposing syntactic information captured at lower layers of LSTM as features to feed forward network contributes to overall model performance and is one of the things critical for success of ElMo embeddings. So since ElMo embeddings use weighted sum and here lower layers are just exposed this feed forward layer on top can learn that sum if information is there. Although we did not do weighted sum it is highly likely that this information is present.

6 Conclusion

We saw from experiments that we can utilize sentence vectors for paragraph text similarity. We also saw that simple feed forward network can be used very effectively. Its quick to train and very simple but results are very good. That makes it useful for usage in limited environments for example we can load it easily on a page or in embedded device.

When we tried to see if we can get easy improvement with stacked LSTM we saw that model although performed really well like the feed forward network in scope of this thesis there was not significant improvement. We could try to come up with intuition for that by saying that we could benefit from LSTM if temporal order would be important for semantic similarity. We could speculate that if we move order of sentences and if that would change meaning of paragraphs then that might be the case.

Although we saw that if we use lower levels of LSTM as features we can improve result. To me that looks like most interesting result. If lower level of stacked LSTM for word vectors capture syntax and higher levels capture semantic info then question can be asked what is analog when it comes to sentences? Is it that lower layers capture local topics and higher capture some higher concepts? Could that be useful for summary of paragraphs potentially? Could it be that they just add to faster error propagation downstream? If latter is the case then for ElMo it must have happened too so it seems that this direction is good for further investigation.

These are all highly interesting questions and very hot topics in modern LSTM and although they are not explored in scope of this thesis we managed to show that there is strong basis for pursuing research in this direction. We can also feel content that we found easy way for paragraph semantic similarity with simple feed forward network.

References

AKB+16a Adi, Y., Kermany, E., Belinkov, Y., Lavi, O. and Goldberg, Y., Fine-grained Analysis of Sentence Embeddings Using Auxiliary Prediction Tasks. *ArXiv e-prints*.

AKB+16b Adi, Y., Kermany, E., Belinkov, Y., Lavi, O. and Goldberg, Y., Fine-grained analysis of sentence embeddings using auxiliary prediction tasks, 2016.

BGJM16 Bojanowski, P., Grave, E., Joulin, A. and Mikolov, T., Enriching Word Vectors with Subword Information. *ArXiv e-prints*.

bno Busting the myths about batch normalization, `https://blog.paperspace.com/busting-the-myths-about-batch-normalization/`. Accessed: 2018-06-01.

CDA+17 Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I. and Specia, L., Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, Vancouver, Canada, August 2017, Association for Computational Linguistics, pages 1–14, URL `http://www.aclweb.org/anthology/S17-2001`.

CKS+17 Conneau, A., Kiela, D., Schwenk, H., Barrault, L. and Bordes, A., Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. *ArXiv e-prints*.

CYK+18 Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., St. John, R., Constant, N., Guajardo-Cespedes, M., Yuan, S., Tar, C., Sung, Y.-H., Strope, B. and Kurzweil, R., Universal Sentence Encoder. *ArXiv e-prints*.

dat16 dataset, Y. W., Yahoo! webscope dataset, 2016.

DKT17 Dheeru, D. and Karra Taniskidou, E., UCI machine learning repository, 2017. URL `http://archive.ics.uci.edu/ml`.

DOL15 Dai, A. M., Olah, C. and Le, Q. V., Document embedding with paragraph vectors. *CoRR*, abs/1507.07998. URL `http://arxiv.org/abs/1507.07998`.

don text summarization, `https://github.com/dongjun-Lee/text-summarization-tensorflow`. Accessed: 2018-06-01.

GB Glorot, X. and Bengio, Y., Understanding the difficulty of training deep feedforward neural networks. URL `http://proceedings.mlr.press/v9/glorot10a/glorot10a.pdf`.

GDR+15 Grbovic, M., Djuric, N., Radosavljevic, V., Silvestri, F. and Bhamidipati, N., Context-and content-aware embeddings for query rewriting in sponsored search. *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2015, pages 383–392.

GDR+16 Grbovic, M., Djuric, N., Radosavljevic, V., Silvestri, F., Baeza-Yates, R., Feng, A., Ordentlich, E., Yang, L. and Owens, G., Scalable semantic matching of queries to ads in sponsored search advertising. *Proceedings of the 39th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2016.

GPH+16a Gan, Z., Pu, Y., Henao, R., Li, C., He, X. and Carin, L., Learning Generic Sentence Representations Using Convolutional Neural Networks. *ArXiv e-prints*.

GPH+16b Gan, Z., Pu, Y., Henao, R., Li, C., He, X. and Carin, L., Learning generic sentence representations using convolutional neural networks, 2016.

HM17 Honnibal, M. and Montani, I., spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*.

IMBgI Iyyer, M., Manjunatha, V., Boyd-graber, J. and Iii, H. D. Ì., Deep unordered composition rivals syntactic methods for text classification.

Jas Jason Brownlee, How to get reproducible results with keras, `https://machinelearningmastery.com/`

reproducible-results-neural-networks-keras/. Accessed: 2018-07-01.

JGBM16 Joulin, A., Grave, E., Bojanowski, P. and Mikolov, T., Bag of Tricks for Efficient Text Classification. *ArXiv e-prints*.

Kil Kilcher, Y., Attention is all you need. URL https://www.youtube.com/watch?v=iDulhoQ2pro.

KZS+15 Kiros, R., Zhu, Y., Salakhutdinov, R., Zemel, R. S., Torralba, A., Urtasun, R. and Fidler, S., Skip-Thought Vectors. *ArXiv e-prints*.

LM14 Le, Q. V. and Mikolov, T., Distributed Representations of Sentences and Documents. *ArXiv e-prints*.

MCCD13a Mikolov, T., Chen, K., Corrado, G. and Dean, J., Efficient Estimation of Word Representations in Vector Space. *ArXiv e-prints*.

MCCD13b Mikolov, T., Chen, K., Corrado, G. and Dean, J., Efficient estimation of word representations in vector space, 2013.

MGB+17 Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C. and Joulin, A., Advances in Pre-Training Distributed Word Representations. *ArXiv e-prints*.

MSC+13 Mikolov, T., Sutskever, I., Chen, K., Corrado, G. and Dean, J., Distributed Representations of Words and Phrases and their Compositionality. *ArXiv e-prints*.

PNI+18 Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K. and Zettlemoyer, L., Deep contextualized word representations. *ArXiv e-prints*.

PRWjZ02 Papineni, K., Roukos, S., Ward, T. and jing Zhu, W., Bleu: a method for automatic evaluation of machine translation. 2002, pages 311–318.

PSM14 Pennington, J., Socher, R. and Manning, C. D., Glove: Global vectors for word representation. *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pages 1532–1543, URL http://www.aclweb.org/anthology/D14-1162.

PSP18 Perone, C. S., Silveira, R. and Paula, T. S., Evaluation of sentence embeddings in downstream and linguistic probing tasks. *ArXiv e-prints*.

ŘS10 Řehůřek, R. and Sojka, P., Software Framework for Topic Modelling with Large Corpora. *Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, May 2010, ELRA, pages 45–50. http://is.muni.cz/publication/884893/en.

Tan11 Tange, O., Gnu parallel - the command-line power tool. *;login: The USENIX Magazine*, 36,1(2011), pages 42–47. URL http://www.gnu.org/s/parallel.

tym wikipedia preparation, http://tymbac.tech/machine/learning/2018/03/01/preparing-wikipedia-for-nlp.html. Accessed: 2018-06-01.

VSP+17 Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I., Attention Is All You Need. *ArXiv e-prints*.

WFC+17 Wu, L., Fisch, A., Chopra, S., Adams, K., Bordes, A. and Weston, J., StarSpace: Embed All The Things! *ArXiv e-prints*.

Wik wikipedia dumps, http://dumps.wikimedia.your.org/. Accessed: 2018-06-01.

Wik18 Wikipedia contributors, Long short-term memory — Wikipedia, the free encyclopedia, https://en.wikipedia.org/w/index.php?title=Long_short-term_memory&oldid=844880987, 2018. [Online; accessed 17-June-2018].

XCBD15 Xu, W., Callison-Burch, C. and Dolan, W. B., SemEval-2015 Task 1: Paraphrase and semantic similarity in Twitter (PIT). *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval)*, 2015.

XRCB+14 Xu, W., Ritter, A., Callison-Burch, C., Dolan, W. B. and Ji, Y., Extracting lexically divergent paraphrases from Twitter. *Transactions of the Association for Computational Linguistics*. URL http://www.cis.upenn.edu/~xwe/files/tacl2014-extracting-paraphrases-from-twitter.pdf.

Xu14 Xu, W., *Data-Drive Approaches for Paraphrasing Across Language Variations*. Ph.D. thesis, Department of Computer Science, New York University, 2014. URL `http://www.cis.upenn.edu/~xwe/files/thesis-wei.pdf`.

A creating wikipedia index on disk

Once unzipping wikipedia dump from that directory with

```
ls -la enwiki-20180220-pages-meta-current* > cc
```

we get list of files into file cc

then to fetch all titles and their line numbers and put it into index file we use gnu parallel[Tan11] and ripgrep.

```
parallel -j54 -a cc "rg -n '<title>' {} > index/{} "
```

from here python code in notebook in next appendix can takeover to get article from very big wikipedia dumps in short amount of time.

Appendices

A Wikipedia cleaning

utility functions

In [ ]:

```
def url2title(url):
    slug=url2slug(url)
    title=slug2title(slug)
    print(title)
    return u''+title
def url2sentences(url):
    print(url)
    slug=url2slug(url)
    title=slug2title(slug)
    return title2sentences(title)
url2slug=lambda x: x.split('/')[-1]
def slug2title(x):
    #r=urllib.parse.unquote(x.replace("_"," "), encoding='utf-8', errors='replac
e').decode('utf8')
    r=urllib.parse.unquote(x.replace("_"," ")).encode('utf8').decode()
    #r=urllib.unquote(x.replace("_"," ")).decode('utf8')
    if '&' in r :
        r= escape(r).encode('utf8', 'xmlcharrefreplace').decode()
    return r

def loadTitles(titlesdf):
    # make the Pool of workers
    pool = ThreadPool(256)

    # open the urls in their own threads
    # and return the results
    results0 = pool.map(wiki_article_rawpy3, titlesdf[0])
    results1 = pool.map(wiki_article_rawpy3, titlesdf[1])
    results2 = pool.map(wiki_article_rawpy3, titlesdf[2])
    # close the pool and wait for the work to finish
    pool.close()

    pool.join()
    return pd.DataFrame([results0,results1,results2])
```

Load original triplets for preprocessing

In [ ]:

```
tripletsr=open('/data/triplets/wikipedia_2014_09_27_examples.txt',encoding='utf
8').readlines()
```

In [ ]:

```
triplets=map(lambda x: x.rstrip(),tripletsr)
```

In [ ]:

```
triplets=[x.split(' ') for x in triplets]
```

In [ ]:

```
titlesdf=pd.DataFrame(triplets)
```

In [ ]:

```
ardf=loadTitles(titlesdf)
```

In [ ]:

```
ardff=ardf.transpose()
```

In [ ]:

```
m=(ardff[0]=="404") | (ardff[1]=="404") | (ardff[2]=="404")
```

In [ ]:

```
def saveDataFrame(df,filename="/data/trpl4.obj"):
    import _pickle as pickle
    out_s = open(filename, 'wb')
    pickle.dump(df,out_s)
    out_s.flush()
    out_s.close()

def loadDF(filename="/data/trpl4.obj"):
    import _pickle as pickle
    in_s = open(filename, 'rb')
    try:
        # Read the data
        while True:
            try:
                o = pickle.load(in_s, encoding='utf8')
            except EOFError:
                break
            else:
                print('READ:')
                return o
    finally:
        in_s.close()
```

In [ ]:

```
tripletsNC=loadDF()
```

In [ ]:

```
tripletsNC
```

use this to extract text from xml that is in wikipedia dump this will extract wikimedia markdown

In [ ]:

```
def extractText(ar):
    if ar=="404":
        return "404"
    a=ET.fromstring(ar)
    #print(a.find('revision/text').text.replace('\'',''))
    z=a.find('revision/text').text
    return z
```

In [ ]:

```
def thradm(ff,titlesdf):
    # make the Pool of workers
    pool = ThreadPool(256)

    # open the urls in their own threads
    # and return the results
    results0 = pool.map(ff, titlesdf.loc[:,0])
    results1 = pool.map(ff, titlesdf.loc[:,1])
    results2 = pool.map(ff, titlesdf.loc[:,2])
    # close the pool and wait for the work to finish
    pool.close()

    pool.join()
    return pd.DataFrame([results0,results1,results2])
```

In [ ]:

```
tm=thradm(extractText,tripletsNC.loc[:,:])
```

In [ ]:

```
tm.shape
```

this will now use gensim to clean wikimedia markdown

In [ ]:

```
tm=thradm(gensim.corpora.wikicorpus.filter_wiki,tm.transpose())
```

In [ ]:

```
tm.shape
```

In [ ]:

```
saveDataFrame(tm,'/data/tripletsCleaned.pickle')
```

In [ ]:

```
i=loadDF('/data/tripletsCleaned.pickle')
```

In [ ]:

```
i.shape
```

In [ ]:

```
ii=i.transpose()
```

follow redirects if we get that article is redirected from markdown

In [ ]:

```
m=ii[0].str.contains("#REDIRECT")
m1=ii[1].str.contains("#REDIRECT")
m2=ii[2].str.contains("#REDIRECT")
```

In [ ]:

```
ai0=ii[m][0].apply(lambda x: re.search('#REDIRECT\s*([^\n]+)', x, re.IGNORECASE)
.group(1)).apply(lambda x: gensim.corpora.wikicorpus.filter_wiki(extractText(wik
i_article_rawpy3(x))))
ai1=ii[m1][1].apply(lambda x: re.search('#REDIRECT\s*([^\n]+)', x, re.IGNORECASE
).group(1)).apply(lambda x: gensim.corpora.wikicorpus.filter_wiki(extractText(wi
ki_article_rawpy3(x))))
ai2=ii[m2][2].apply(lambda x: re.search('#REDIRECT\s*([^\n]+)', x, re.IGNORECASE
).group(1)).apply(lambda x: gensim.corpora.wikicorpus.filter_wiki(extractText(wi
ki_article_rawpy3(x))))
```

In [ ]:

```
ii[0][m]=ai0
```

In [ ]:

```
ii[1][m1]=ai1
```

In [ ]:

```
ii[2][m2]=ai2
```

In [ ]:

```
saveDataFrame(ii,'/data/tripletsCleaned2.pickle')
```

B Experiments

convert parsed markdown into broken down sentences after this paragraph will be array of sentences and sentence is array of words without stop words, spaces, punctuation and alphanumerics

In [ ]:

```
import _pickle as pickle
in_s = open("/data/tripletsCleaned2.pickle","rb")
pds=pickle.load(in_s,encoding="utf8")
from multiprocessing.dummy import Pool as ThreadPool
def paragraph2sent(d):
    try:
        return (np.array([np.array([y.lower_ for y in x if not y.is_stop and not
 y.is_space and not y.is_punct and y.is_alpha]) for x in nlp(d).sents])) if d!=
"404" else "404"
    except Exception as e:
        print(e)
        print(d)
        return "fail"
def thradm(ff,titlesdf):
    # make the Pool of workers
    pool = ThreadPool(8)

    # open the urls in their own threads
    # and return the results
    results0 = pool.map(ff, titlesdf.loc[:,0])
    results1 = pool.map(ff, titlesdf.loc[:,1])
    results2 = pool.map(ff, titlesdf.loc[:,2])
    # close the pool and wait for the work to finish
    pool.close()

    pool.join()
    return pd.DataFrame([results0,results1,results2])
pds2=thradm(paragraph2sent,pds)
saveDataFrame(pds2,'/data/trpl66.obj')
```

In [ ]:

```
triplets=loadDF('/data/trpl66.obj')
```

convert data set into vector embeddings

In [ ]:

```
kk=aa(triplets.transpose()[0])
kkk=np.array(kk)
ll=[np.array(k).flatten() for k in kkk.flatten()]
lll=np.hstack(ll)
p1(lll,0)
kk=aa(triplets.transpose()[1])
kkk=np.array(kk)
ll=[np.array(k).flatten() for k in kkk.flatten()]
lll=np.hstack(ll)
p1(lll,1)
kk=aa(triplets.transpose()[2])
kkk=np.array(kk)
ll=[np.array(k).flatten() for k in kkk.flatten()]
lll=np.hstack(ll)
p1(lll,2)
```

convert triplets to USE embeddings

In [ ]:

```
kk=aa(triplets.transpose()[0])
e=[]
kkk=np.array(kk)
ll=[np.array(k).flatten() for k in kkk.flatten()]
lll=np.hstack(ll)
tf.reset_default_graph()

with tf.Session() as session:
    embed = hub.Module("https://tfhub.dev/google/universal-sentence-encoder/1")

    session.run(tf.global_variables_initializer())
    session.run(tf.tables_initializer())
    i=0
    emb=session.run(embed(lll))
    e.append(emb)
saveDataFrame(e,'/data/t0s.obj')
```

In [ ]:

```
kk=aa(triplets.transpose()[1])
e=[]
kkk=np.array(kk)
ll=[np.array(k).flatten() for k in kkk.flatten()]
lll=np.hstack(ll)
tf.reset_default_graph()

with tf.Session() as session:
    embed = hub.Module("https://tfhub.dev/google/universal-sentence-encoder/1")

    session.run(tf.global_variables_initializer())
    session.run(tf.tables_initializer())
    i=0
    emb=session.run(embed(lll))
    e.append(emb)
saveDataFrame(e,'/data/t1s.obj')
```

In [ ]:

```
kk=aa(triplets.transpose()[2])
e=[]
kkk=np.array(kk)
ll=[np.array(k).flatten() for k in kkk.flatten()]
lll=np.hstack(ll)
tf.reset_default_graph()

with tf.Session() as session:
    embed = hub.Module("https://tfhub.dev/google/universal-sentence-encoder/1")

    session.run(tf.global_variables_initializer())
    session.run(tf.tables_initializer())
    i=0
    emb=session.run(embed(lll))
    e.append(emb)
saveDataFrame(e,'/data/t2s.obj')
```

Here we imports libraries

In [1]:

```
from sklearn.model_selection import train_test_split
from random import sample
import subprocess
from multiprocessing.dummy import Pool as ThreadPool
import re
import numpy as np
import pandas as pd
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.pyplot as plt
from pyparsing import *
import itertools
from functools import partial
from IPython.core.display import display, HTML
display(HTML("<style>.container {width:100% !important;}</style>"))
#pd.options.display.max_columns=None
plt.rcParams["figure.figsize"]=20,20
import warnings
from string import Template
warnings.simplefilter("ignore")
%matplotlib inline
import os
os.popen('ls -la /data').read()
import urllib
import html
import gensim
import spacy
from keras import optimizers
```

```
/usr/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning: nump
y.dtype size changed, may indicate binary incompatibility. Expected
96, got 88
  return f(*args, **kwds)
/usr/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning: nump
y.dtype size changed, may indicate binary incompatibility. Expected
96, got 88
  return f(*args, **kwds)
/usr/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning: nump
y.dtype size changed, may indicate binary incompatibility. Expected
96, got 88
  return f(*args, **kwds)
/usr/lib/python3.5/importlib/_bootstrap.py:222: RuntimeWarning: nump
y.dtype size changed, may indicate binary incompatibility. Expected
96, got 88
  return f(*args, **kwds)

Using TensorFlow backend.
```

In [4]:

```
import os
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import re
import seaborn as sns
from gensim.models import Doc2Vec
import gensim.downloader as api
import pandas
import scipy
import math
import spacy
import tensorflow as tf
import tensorflow_hub as hub
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import re
import seaborn as sns
from gensim.models import Doc2Vec
import gensim.downloader as api
import pandas
import scipy
import math
import spacy
from nltk.translate.bleu_score import corpus_bleu
import spacy
import functools
import nltk
import torch
import gensim

from tensorflow.python import debug as tf_debug
import tensorflow_hub as hub
from keras.layers import Dropout
from keras import backend as K
from keras.layers import Dense
from keras.objectives import categorical_crossentropy
from keras.metrics import categorical_accuracy as accuracy
from sklearn.metrics.pairwise import cosine_similarity

from keras import backend as K
from sklearn.metrics.pairwise import cosine_similarity
import sklearn
from sklearn.model_selection import train_test_split
tow=lambda x: [x.lower_ for x in nlp(str(x)) if not x.is_stop and not x.is_punct
]
from sklearn.metrics import accuracy_score
from tensorflow.python import debug as tf_debug
import keras
```

In [5]:

```
tf.logging.set_verbosity(tf.logging.ERROR)
```

utlity functions here wiki_article_rawpy3 is what actually gets wikipedia article from raw wikipedia dump from disk and after postprocessing

In [9]:

```
def wiki_article_rawpy3(title="Las Leyendas"):
    try:
        #wiki_article_rawpy3("Muharram museum")
        val = subprocess.check_output(("/data/srch \"%s\"" % title).encode(),
shell=True)
        return '<page>'+val.decode('utf8')
    except Exception as e:
        return "404"
import xml.etree.ElementTree as ET
def wiki_article_cleaned(title="Helsinki"):
    a=wiki_parsed(title)
    return a

def wiki_parsed(title="Helsinki"):
    article=wiki_article_raw(title)
    a=ET.fromstring(article)
    #print(a.find('revision/text').text.replace('\'',''))
    z=a.find('revision/text').text
    b=mwparserfromhell.parse(z)
    return b.strip_code(normalize=True,collapse=True)

def text2wordlists(z):
    return np.array([" ".join(np.array(re.findall("\w+",x))) for x in  z.split(
'.')])

def text2sentencAverage(text):
    sent=text2wordlists(z)
    return sent

def article2sentences(article):

    def convertToHTML_A(s,l,t):
        return ''
        print(t[0])
        return t[0]
        #return t[0].split('.')
        #ry:
            #text,url=t[0].split("->")
        #xcept ValueError:
        #    raise ParseFatalException(s,l,"invalid URL link reference: " + t
[0])
        return t
    def convertToHTML(opening="",closing=""):
        def conversionParseAction(s,l,t):
            #print(t[0].split('.'))
            return  t[0]
        return conversionParseAction
    def convertToHTMLAA(opening="",closing=""):
        def conversionParseAction(s,l,t):
            #print(t[0].split('.'))
            return  t[0][1:-1]
        return conversionParseAction
    #sentence =  w + sentence | sentence +w
    def convertToHTML_AB(s,l,t):
        return t[0]

    a = QuotedString("== ",endQuoteChar=" ==").setParseAction(convertToHTML_AB)
    italicized = QuotedString("*").setParseAction(convertToHTML_A)
    bolded = QuotedString("**").setParseAction(convertToHTML_A)
```

```
    boldItalicized = QuotedString("***").setParseAction(convertToHTML_A)



    urlRef = QuotedString("{{",endQuoteChar="}}").setParseAction(convertToHTML_A
B)
    urlRe2f = QuotedString("[[",endQuoteChar="]]").setParseAction(convertToHTML_
AB)
    urlRe2ff = QuotedString("<ref>",endQuoteChar="</ref>").setParseAction(conver
tToHTML_A)
    w=urlRe2f | urlRef | boldItalicized | bolded | italicized |  a | urlRe2ff |
(Regex(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[!*\(\),]|(?:%[0-9a-fA-F][0-9a-f
A-F]))+')).setParseAction(convertToHTML_A)   | (Regex(r"\*+?[^\n]")).setParseAct
ion(convertToHTMLAA())
    wikiMarkup =w | (Regex(r".+?")).setParseAction(convertToHTML())
    #| w + wikiMarkup


    #print t
    b=wikiMarkup.transformString(article)
    b=re.sub(r"\n+",'\n',b)
    b=re.sub(r"[']+",'',b)
    b=re.sub(r"[\:]+",' ',b)
    b=re.sub(r"[\|]+",' ',b)
    b=re.sub(r"[\]\[\)\(]+",'',b)
    prin(b.split("\n"))
    bb=list(itertools.chain(*[x.split('.') for x in b.split("\n")]))
    return np.array([np.array(re.findall('\w+',x)) for x in bb])

def wiki_parsed2(article):
    a=ET.fromstring(article)
    #print(a.find('revision/text').text.replace('\'',''))
    z=a.find('revision/text').text
    b=mwparserfromhell.parse(z)
    return b.strip_code(normalize=True,collapse=True,keep_template_params=False)
def text2wordlists2(b):
    b=re.sub(r"\n+",'\n',b)
    b=re.sub(r"[']+",'',b)
    b=re.sub(r"[\:]+",' ',b)
    b=re.sub(r"[\|]+",' ',b)
    b=re.sub(r"[\]\[\)\(]+",'',b)
    #print b.split("\n")
    bb=list(itertools.chain(*[x.split('.') for x in b.split("\n")]))
    return np.array([np.array(re.findall('\w+',x)) for x in bb if len(re.findall
('\w+',x))])
def title2sentences(title):
    return text2wordlists2(wiki_parsed(title))
```

In [8]:

```
def url2title(url):
    slug=url2slug(url)
    title=slug2title(slug)
    print(title)
    return u''+title
def url2sentences(url):
    print(url)
    slug=url2slug(url)
    title=slug2title(slug)
    return title2sentences(title)
url2slug=lambda x: x.split('/')[-1]
def slug2title(x):
    #r=urllib.parse.unquote(x.replace("_"," "), encoding='utf-8', errors='replac
e').decode('utf8')
    r=urllib.parse.unquote(x.replace("_"," ")).encode('utf8').decode()
    #r=urllib.unquote(x.replace("_"," ")).decode('utf8')
    if '&' in r :
        r= escape(r).encode('utf8', 'xmlcharrefreplace').decode()
    return r

def loadTitles(titlesdf):
    # make the Pool of workers
    pool = ThreadPool(256)

    # open the urls in their own threads
    # and return the results
    results0 = pool.map(wiki_article_rawpy3, titlesdf[0])
    results1 = pool.map(wiki_article_rawpy3, titlesdf[1])
    results2 = pool.map(wiki_article_rawpy3, titlesdf[2])
    # close the pool and wait for the work to finish
    pool.close()

    pool.join()
    return pd.DataFrame([results0,results1,results2])
```

utility functions

In [11]:

```
sentence_size=300
```

In [12]:

```
from cgi import escape
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix


tripletDoc2sent = lambda d: np.hstack(np.array([np.array([y.lower_ for y in x if
 not y.is_stop and not y.is_space and not y.is_punct]) for x in nlp(d).sents]))
if d!='404' else "404"
nlp = spacy.load('en')
#nlp = spacy.load('en')
paragraph2sent = lambda d: np.hstack(np.array([np.array([y.lower_ for y in x if
not y.is_stop and not y.is_space and not y.is_punct]) for x in nlp(d).sents])) i
f d!="404" else "404"
doc2text=lambda d: gensim.corpora.wikicorpus.filter_wiki("\n".join(d['section_te
xts']))
doc2arr=lambda d: paragraph2sent(doc2text(d))
def sent2arr(sent):
    return [y.lower_ for y in next((nlp(sent).sents)) if not y.is_stop and not y
.is_space and not y.is_punct]
def paragraph2sent(d):
    try:
        return np.hstack(np.array([np.array([y.lower_ for y in x if not y.is_sto
p and not y.is_space and not y.is_punct]) for x in nlp(d).sents])) if d!="404" e
lse "404"
    except Exception as e:
        print(e)
        print(d)
        return "fail"

from multiprocessing.dummy import Pool as ThreadPool
def saveDataFrame(df,filename="/data/trpl4.obj"):
    import _pickle as pickle
    out_s = open(filename, 'wb')
    pickle.dump(df,out_s)
    out_s.flush()
    out_s.close()

def loadDF(filename="/data/trpl4.obj"):
    import _pickle as pickle
    in_s = open(filename, 'rb')
    try:
        # Read the data
        while True:
            try:
                o = pickle.load(in_s, encoding='utf8')
            except EOFError:
                break
            else:
                print('READ:')
                return o
    finally:
        in_s.close()
import keras

def variable_summaries(var):
  """Attach a lot of summaries to a Tensor (for TensorBoard visualization)."""
```

```
  with tf.name_scope('summariesx'):
    mean = tf.reduce_mean(var)
    tf.summary.scalar('mean', mean)
    with tf.name_scope('stddev'):
        stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
    tf.summary.scalar('stddev', stddev)
    tf.summary.scalar('max', tf.reduce_max(var))
    tf.summary.scalar('min', tf.reduce_min(var))
    tf.summary.histogram('histogram', var)

def loadDF(filename="/data/trpl4.obj"):
    import _pickle as pickle
    in_s = open(filename, 'rb')
    try:
        # Read the data
        while True:
            try:
                o = pickle.load(in_s, encoding='utf8')
            except EOFError:
                break
            else:
                print('READ:')
                return o
    finally:
        in_s.close()


sent2vecs =  lambda sent,v=glovetwitter100: np.array([v.get_vector(x) for x in s
ent if x in v])
sent2avg = lambda sent,v=glovetwitter100: np.average([v.get_vector(x) if x in v
else np.zeros(100) for x in sent ],axis=0)


vsent2avg = lambda sent,v=glovetwitter100: np.average(sent,axis=0)
sent2vecs = lambda sent,v=glovetwitter100: np.array([v.get_vector(sent[x]) if (x
<len(sent) and sent[x] in v) else np.zeros((100)) for x in range(sentence_size)
])
doc2vecs=lambda doc,v=glovetwitter100: np.array([*map(sent2vecs,doc)])
doc2avg=lambda doc,v=glovetwitter100: np.average(np.array([*map(sent2avg,[*filte
r(len,doc)])]),axis=0) if (len(doc)>0) else np.zeros((100))
vdoc2avg=lambda doc,v=glovetwitter100: np.average(np.array([*map(vsent2avg,doc
)]),axis=1)
def negative_sampling(q2):
    return np.random.permutation(q2)

def build_dataset(q1,q2):
    neg=negative_sampling(q2)
    q11=np.hstack((np.array(q1),np.array(q1)))
    q22=np.hstack((q2,neg))
    return [*zip(q11,q22)],np.hstack([np.ones(len(q1)),np.zeros(len(q1))])
def triplets_clean(triplets):
    t=[]
    ii=0
    print('start')
    print(triplets.shape)
    for i,triplet in triplets.iterrows():
        if not isinstance(triplet[0],(int, str)):
            if not isinstance(triplet[1],(int, str)):
                t.append([triplet[0],triplet[1]])
            if not isinstance(triplet[2],(int, str)):
                t.append([triplet[0],triplet[2]])
```

```
        if not isinstance(triplet[1],(int, str)):
            if not isinstance( triplet[2],(int, str)):
                t.append([triplet[1],triplet[2]])
        ii+=1
        if ii%1000==0:
            print(ii/1000)
    return np.array(t)


def partition(dataset,start):
    for ll in range(start,10000):
        m=int(lll.shape[0]/10000)
        print(ll*m,(ll+1)*m)
        yield lll[ll*m:(ll+1)*m]



    m=int(lll.shape[0]/10000)
    print((ll+1)*m,(lll.shape[0]))
    yield lll[(ll+1)*m:(lll.shape[0])]

def tovec(lll,v,vv):
    tf.reset_default_graph()

    with tf.Session() as session:
        elmo = hub.Module("https://tfhub.dev/google/elmo/1", trainable=True)

        session.run(tf.global_variables_initializer())
        session.run(tf.tables_initializer())
        i=0
        elmo1=[]
        el=elmo(lll,
            signature="default",
            as_dict=True)["elmo"]
        emb=session.run(el)
        elmo1.append(emb)
        saveDataFrame(elmo1,'/data/t%selmo%s.obj'%(vv,v))


def p1(lll,v,start=0,end=0):
    i=start
    for ll in partition(lll,start):
        tovec(ll,i,v)
        if end!=0 and i==end:
            break
        i+=1
def clear_logs():
    import os, shutil
    folder = '/data/logs'
    for the_file in os.listdir(folder):
        file_path = os.path.join(folder, the_file)
        try:
            if os.path.isfile(file_path):
                os.unlink(file_path)
            #elif os.path.isdir(file_path): shutil.rmtree(file_path)
        except Exception as e:
            print(e)
```

load post processed dataset that is still not vectorised

In [16]:

```
triplets=loadDF('/data/trpl66.obj')
```

READ:

In [17]:

```
t0=loadDF('/data/sentences/sentences/t0s.obj')
t1=loadDF('/data/sentences/sentences/t1s.obj')
t2=loadDF('/data/sentences/sentences/t2s.obj')
```

READ:
READ:
READ:

In [18]:

```
len2=[len(x) for x in triplets.transpose()[2]]
len1=[len(x) for x in triplets.transpose()[1]]
len0=[len(x) for x in triplets.transpose()[0]]
```

In [19]:

```
np.mean(len2)
```

Out[19]:

59.3843

since triplets dataset is flatten we need to unflatten those arrays by reconstructing paragraphs by their length since after conversion they are flatten

In [31]:

```
f0=[True if isinstance(x, (list, tuple, np.ndarray)) and len(x)>0 else False for
 x in triplets.transpose()[0]]
f1=[True if isinstance(x, (list, tuple, np.ndarray)) and len(x)>0 else False for
 x in triplets.transpose()[1]]
f2=[True if isinstance(x, (list, tuple, np.ndarray)) and len(x)>0 else False for
 x in triplets.transpose()[2]]
```

In [32]:

```
t0sum=np.cumsum(len0)
tt0=np.split(t0[0],t0sum)
t1sum=np.cumsum(len1)
tt1=np.split(t1[0],t1sum)
t2sum=np.cumsum(len2)
tt2=np.split(t2[0],t2sum)
```

utility functions for making dataset

In [33]:

```
def d(X,Y):
    return (lambda: input_fn(X,Y))
```

In [34]:

```
SEED = 2018
np.random.seed(SEED)
```

pick similar or not similar from triplets

In [35]:

```
def pickExisting(r,r2=None):
    #print('sampling')
    if r2!=None:
        if f0[r] and f0[r2]:
            return np.array([tt0[r],tt0[r2]])
        if f0[r] and f1[r2]:
            return np.array([tt0[r],tt1[r2]])
        if f0[r] and f2[r2]:
            return np.array([tt0[r],tt2[r2]])
        if f1[r] and f1[r2]:
            return np.array([tt1[r],tt1[r2]])
        if f1[r] and f2[r2]:
            return np.array([tt1[r],tt2[r2]])
        if f2[r] and f2[r2]:
            return np.array([tt2[r],tt2[r2]])
    else:
        if f0[r] and f1[r]:
            return np.array([tt0[r],tt1[r]])
        if f1[r] and f2[r]:
            return np.array([tt1[r],tt2[r]])
        if f0[r] and f2[r]:
            return np.array([tt0[r],tt2[r]])
```

In [36]:

```
LEN=20000
```

pick one pair for a dataset

In [ ]:

```
def sample(mode):
    if mode==tf.estimator.ModeKeys.TRAIN:
        rb,rt=0, int(LEN*0.8)
    if mode==tf.estimator.ModeKeys.PREDICT:
        rb,rt=int(LEN*0.8), int(LEN*0.9)
    if mode==tf.estimator.ModeKeys.EVAL:
        rb,rt=int(LEN*0.9), (LEN-1)

    for i in itertools.count(rb):
        if(i>rt):
            return
        r=np.random.randint(rb,rt)
        if r%2:
            #print("range %s"%(r))
            pick=pickExisting(i)
            if(pick is None):
                continue
            yield np.array([pick,1])
        else:
            r2=np.random.randint(rb,rt)
            #print("range %s %s"%(r,r2))
            pick=pickExisting(i,r2)
            if(pick is None):
                continue
            yield np.array([pick,0])
```

here datasets are built that will go into model

In [38]:

```
SEED = 2018
np.random.seed(SEED)
X_test,Y_test=np.array([x for x in sample(tf.estimator.ModeKeys.PREDICT)]).T
```

In [39]:

```
SEED = 2018
np.random.seed(SEED)
X_validation,Y_validation=np.array([x for x in sample(tf.estimator.ModeKeys.EVAL
)]).T
```

In [40]:

```
SEED = 2018
np.random.seed(SEED)
X_train,Y_train=np.array([x for x in sample(tf.estimator.ModeKeys.TRAIN)]).T
```

Here utility functions are set for tensorflow estimator to enable quick itteration

In [41]:

```
def lossf(labels,predictions):
    return tf.losses.hinge_loss(labels,predictions)
optimizer = tf.train.AdamOptimizer(learning_rate=0.0000005)
```

In [42]:

```
def model_fn(features, labels, mode, params):

    """The model_fn argument for creating an Estimator."""
    predictions=model(features,labels)
    global optimizer
    #saver = tf.train.Saver()

    if mode == tf.estimator.ModeKeys.PREDICT:

        return tf.estimator.EstimatorSpec(
            mode=mode,
            predictions= predictions
        )

    print(labels)

    print(predictions)

    loss=lossf(labels,predictions)
    accuracy = tf.metrics.accuracy(
        labels=labels, predictions=predictions)
    if mode == tf.estimator.ModeKeys.TRAIN:

        train_op = optimizer.minimize(
            loss=loss,
            global_step=tf.train.get_global_step())

        # If we are running multi-GPU, we need to wrap the optimizer.
        if params.get('multi_gpu'):
            optimizer = tf.contrib.estimator.TowerOptimizer(optimizer)



        # Name tensors to be logged with LoggingTensorHook.
        #tf.identity(LEARNING_RATE, 'learning_rate')
        tf.identity(loss, 'train_loss')
        tf.identity(accuracy[1], name='train_accuracy')

        # Save accuracy scalar to Tensorboard output.
        tf.summary.scalar('train_accuracy', accuracy[1])
        training_hooks = tf.train.SummarySaverHook(
                                    save_steps=100,
                                    output_dir= "/data/logs",
                                    summary_op=tf.summary.merge_all())

        return tf.estimator.EstimatorSpec(
            mode=tf.estimator.ModeKeys.TRAIN,
            loss=loss,
            train_op=optimizer.minimize(loss, tf.train.get_or_create_global_step
()),
            training_hooks=[training_hooks]
        )

    if mode == tf.estimator.ModeKeys.EVAL:
        tf.identity(loss, 'validation_loss')
        tf.identity(accuracy[1], name='validation_accuracy')

        # Save accuracy scalar to Tensorboard output.
        tf.summary.scalar('validation_accuracy', accuracy[1])
```

```
        validation_hooks = tf.train.SummarySaverHook(
                                    save_steps=100,
                                    output_dir= "/data/logs",
                                    summary_op=tf.summary.merge_all())
        return tf.estimator.EstimatorSpec(
            mode=tf.estimator.ModeKeys.EVAL,
            loss=loss,
            evaluation_hooks=[validation_hooks],
            eval_metric_ops={
                'validation_accuracy': accuracy
            })
```

In [43]:

```
def train(steps=100,epochs=1):
    clear_logs()
    #hook = tf_debug.TensorBoardDebugHook("5992c370992f:6008")

    tf.reset_default_graph()
    global classifier
    global X_train
    global Y_train
    classifier = tf.estimator.Estimator(
      model_fn=model_fn,
      params={
      },
      model_dir='/data/logs',
      config=tf.contrib.learn.RunConfig(
                save_checkpoints_steps=10,
                save_summary_steps=10,
                save_checkpoints_secs=None)
    )
    SEED = 2018
    np.random.seed(SEED)
    for i in range(epochs):
        classifier.train(input_fn=d(X_train,Y_train)
                         ,steps=steps
                         #,hooks=[hook]
                        )
    return classifier
```

In [44]:

```
cutoff=0.5
```

In [45]:

```
def test_set():
    SEED = 2018
    np.random.seed(SEED)
    global classifier
    global X_test
    global Y_test
    global cutoff
    predictions=classifier.predict(input_fn=d(X_test,Y_test))
    preds1=np.array([*predictions]).flatten()

    preds=preds1.copy()
    print("cutoff %s"%cutoff)
    preds[preds>cutoff]=1
    preds[preds<=cutoff]=0
    preds=np.asarray(preds, dtype=int)
    accuracy=accuracy_score(preds,[int(x) for x in Y_test])
    print("accuracy %s"%accuracy)
    return preds1
```

In [46]:

```
def train_set():
    SEED = 2018
    np.random.seed(SEED)
    global classifier
    global X_train
    global Y_train
    global cutoff
    predictions=classifier.predict(input_fn=d(X_train,Y_train))
    preds1=np.array([*predictions]).flatten()

    print("cutoff %s"%cutoff)
    preds=preds1.copy()
    preds[preds>cutoff]=1
    preds[preds<=cutoff]=0
    preds=np.asarray(preds, dtype=int)
    accuracy=accuracy_score(preds,[int(x) for x in Y_train[0:preds.shape[0]]])
    print("accuracy %s"%accuracy)
    return preds1
```

In [47]:

```
def eval_set():
    SEED = 2018
    np.random.seed(SEED)
    global classifier
    global X_eval
    global Y_eval
    global cutoff
    predictions=classifier.predict(input_fn=d(X_eval,Y_eval))
    preds1=np.array([*predictions]).flatten()

    print("cutoff %s"%cutoff)
    preds=preds1.copy()
    preds[preds>cutoff]=1
    preds[preds<=cutoff]=0
    preds=np.asarray(preds, dtype=int)
    accuracy=accuracy_score(preds,[int(x) for x in Y_eval[0:preds.shape[0]]])
    print("accuracy %s"%accuracy)
    return preds1
```

In [48]:

```
def lossf(labels,predictions):
    return tf.losses.sigmoid_cross_entropy(labels,predictions)
optimizer = tf.train.AdagradOptimizer(learning_rate=0.005)
```

various utility functions to get sum of vectors in paragraph
or to get average etc.

In [50]:

```
sent2vecs =  lambda sent,v=glovetwitter100: np.array([v.get_vector(x) for x in s
ent if x in v])
sent2avg = lambda sent,v=glovetwitter100: np.average([v.get_vector(x) if x in v
else np.zeros(100) for x in sent ],axis=0)

vsent2sum = lambda sent,v=glovetwitter100: np.sum(sent,axis=0)/np.linalg.norm(np
.sum(sent,axis=0))
vsent2avg = lambda sent,v=glovetwitter100: np.average(sent,axis=0)
sent2vecs = lambda sent,v=glovetwitter100: np.array([v.get_vector(sent[x]) if (x
<len(sent) and sent[x] in v) else np.zeros((100)) for x in range(sentence_size)
])
doc2vecs=lambda doc,v=glovetwitter100: np.array([*map(sent2vecs,doc)])
doc2avg=lambda doc,v=glovetwitter100: np.average(np.array([*map(sent2avg,[*filte
r(len,doc)])]),axis=0) if (len(doc)>0) else np.zeros((100))
vdoc2avg=lambda doc,v=glovetwitter100: np.average(np.array([*map(vsent2avg,doc
)]),axis=1)
```

experiment 1

this is testbed for experiment 1

In [55]:

```
def input_fn(X,Y):
    SEED = 2018
    np.random.seed(SEED)
    batch_size=6
    import itertools
    def gen(xs,output_shape=(1)):
        def b():
            batchx=[]
            batchy=[]
            i=0
            for n in xs:
                i+=1
                bx={"sent1":vsent2sum(n[0][0]),"sent2":vsent2sum(n[0][1])}
                assert not np.any(np.isnan(np.asarray(bx["sent1"], dtype=float
)))
                assert not np.any(np.isnan(np.asarray(bx["sent2"], dtype=float
)))
                by=np.array([n[1]]).reshape((1))
                yield bx,by
            return
        return b
    dataset = tf.data.Dataset.from_generator(gen(zip(X,Y)), ({"sent1": (tf.float
32),"sent2": (tf.float32)},(tf.int32)),
                                             output_shapes=({"sent1": [512],"sent2"
: [512]},[1]))
    dataset=dataset.apply(tf.contrib.data.batch_and_drop_remainder(batch_size))
    iterr = dataset.make_one_shot_iterator()
    elem= iterr.get_next()
    return elem
```

In [56]:

```
from sklearn.preprocessing import normalize
```

In [57]:

```
input_doc_shape=[512]
```

In [58]:

```
def model(features,labels):
    tf.set_random_seed(SEED)
    dev_scores0 = labels
    input_layer1 = tf.feature_column.input_layer(features,[tf.feature_column.num
eric_column("sent1", shape=input_doc_shape)])
    input_layer2 = tf.feature_column.input_layer(features,[tf.feature_column.num
eric_column("sent2", shape=input_doc_shape)])
    sim_scores=tf.concat([input_layer1, input_layer2], 1)
    bb4=tf.layers.dense(sim_scores,1024)
    bb4=tf.layers.dropout(bb4, rate=0.8)
    bb4=tf.layers.dense(bb4,512)
    bb4=tf.layers.dropout(bb4, rate=0.5)
    bb4=tf.layers.dense(bb4,32)
    bb4=tf.layers.dense(bb4,1)
    return bb4
```

In [60]:

```
def lossf(labels,predictions):
    return tf.losses.sigmoid_cross_entropy(labels,predictions)
optimizer = tf.train.AdamOptimizer(learning_rate=0.05)
```

In [ ]:

```
classifier=train(steps=None,epochs=3)
```

In [ ]:

```
classifier.train(input_fn=d(X_train,Y_train)
                         ,steps=100
                         #,hooks=[hook]
                        )
```

In [ ]:

```
cutoff=0.0
```

In [ ]:

```
cutoff=0.5
```

In [ ]:

```
preds1=train_set()
```

In [ ]:

```
preds1=test_set()
```

In [ ]:

```
preds1
```

experiment 1.2

In [ ]:

```
tbCallBack = keras.callbacks.TensorBoard(log_dir='/data/logs', histogram_freq=0,
 write_graph=True, write_images=True)
```

In [ ]:

```
init=keras.initializers.glorot_normal(seed=None)
clear_logs()
```

here we try just feed forward network

In [ ]:

```
input1 = keras.layers.Input(shape=(512,))
input2 = keras.layers.Input(shape=(512,))
added = keras.layers.Concatenate(axis=-1)([input1, input2])
x3=keras.layers.BatchNormalization(axis=-1)(added)
x3 = keras.layers.Dense(512, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.5)(x3)
x3 = keras.layers.Dense(128, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.5)(x3)
x3 = keras.layers.Dense(64,kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
x3 = keras.layers.Dropout(0.2)(x3)
x3 = keras.layers.Dense(16, activation='selu',kernel_initializer=init)(x3)
x3=keras.layers.BatchNormalization(axis=-1)(x3)
out = keras.layers.Dense(1)(x3)
model = keras.models.Model(inputs=[input1, input2], outputs=out)
```

In [ ]:

```
sgd = optimizers.SGD(lr=0.00001, decay=1e-6, momentum=0.9, nesterov=True)
nadam = optimizers.Nadam()
adadelta=keras.optimizers.Adadelta(lr=0.000001, rho=0.95, epsilon=None, decay=1e
-6)
```

In [ ]:

```
model.compile(loss='binary_crossentropy',
              #optimizer=sgd,
              optimizer=nadam,
              #optimizer=adadelta,
              metrics=['accuracy'])
```

In [ ]:

```
model.fit([np.array([vsent2avg(x[0]) for x in X_train]),np.array([vsent2avg(x[1
]) for x in X_train])], Y_train,
          epochs=10,
          batch_size=10, callbacks=[tbCallBack])
```

In [ ]:

```
scores = model.evaluate([np.array([vsent2avg(x[0]) for x in X_test]),np.array([v
sent2avg(x[1]) for x in X_test])], Y_test)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

In [ ]:

```
scores = model.evaluate([np.array([vsent2avg(x[0]) for x in X_train]),np.array([
vsent2avg(x[1]) for x in X_train])], Y_train)
print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
```

if paragraph has more than maxLen sentences it will have maxLen after this function
last sentence vector is average of all sentence vectors in paragraph if there are more sentences than max len
if there are less then average will be padded to fill up maxLen number of sentences for paragraph

In [3]:

```
def normlz(doc):
    avg = np.average(doc,axis=0)
    sh=np.vstack((doc[0:(maxLen-1)],np.tile(np.array(avg),(abs(maxLen-len(doc))
if maxLen>len(doc) else 1 ,1))))
    return sh
```

this function generates batches for input dataset

In [4]:

```
def gen(xs,batch_size=25):
    def b():
        batchx1=[]
        batchx2=[]
        batchy=[]
        while 1:
            i=0
            batchx1=[]
            batchx2=[]
            batchy=[]
            for n in xs:
                i+=1
                bx1=normlz(n[0][0])
                bx2=normlz(n[0][1])
                batchx1.append(bx1)
                batchx2.append(bx2)
                batchy.append(n[1])
                if(i%batch_size==0):
                    yield [np.array(batchx1),np.array(batchx2)],np.asarray(batch
y)
                    batchx1=[]
                    batchx2=[]
                    batchy=[]
        return
    return b
```

Here we evaluate model with 6 for LSTM dimension
we run it 20

In [ ]:

```
acc=[]
for i in range(1,20):
    from tensorflow import set_random_seed
    set_random_seed(2)
    maxLen=300
    seed=2017
    tbCallBack = keras.callbacks.TensorBoard(log_dir='/data/logs', histogram_fre
q=0, write_graph=True, write_images=True)
    init=keras.initializers.glorot_normal(seed=seed)
    clear_logs()

    batch_size=30
    np.random.seed(1)




    input1 = keras.layers.Input(batch_shape=(batch_size,maxLen,512),
                                shape=(maxLen,512),dtype='float32')
    input2 = keras.layers.Input(batch_shape=(batch_size,maxLen,512),
                                shape=(maxLen,512),dtype='float32')
    #here we set dimension
    dim1=6
    stateful=False
    X1=input1
    lstm1, state_h1, state_c1 = keras.layers.LSTM(dim1,
                            return_sequences=True,
                            return_state=True,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(X1)

    X1 = keras.layers.LSTM(dim1,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(lstm1)

    X2=input2
    lstm2, state_h2, state_c2  = keras.layers.LSTM(dim1,
                            return_sequences=True,
                            return_state=True,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(X2)
    X2 = keras.layers.LSTM(dim1,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            #recurrent_activation='selu',
                            activation='selu'
                           )(lstm2)

    added = keras.layers.Concatenate(axis=-1)([X1, X2])
```

```
    x3=added
    x3 = keras.layers.Dense(512, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dropout(0.5,seed=seed)(x3)
    x3 = keras.layers.Dense(128, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dropout(0.3,seed=seed)(x3)
    x3 = keras.layers.Dense(64,kernel_initializer=init, activation='selu')(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dropout(0.2,seed=seed)(x3)
    x3 = keras.layers.Dense(32, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dense(16, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    out = keras.layers.Dense(1,kernel_initializer=init)(x3)
    print(out)
    model = keras.models.Model(inputs=[input1, input2], outputs=out)

    sgd = optimizers.SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
    nadam = optimizers.Nadam(lr=0.45)
    adadelta=keras.optimizers.Adadelta(lr=0.5, rho=0.95, epsilon=None, decay=1e-
6)

    model.compile(loss='binary_crossentropy',
                  #optimizer=sgd,
                  optimizer=adadelta,
                  #optimizer=nadam,
                  metrics=['accuracy'])
    size=5000
    size=X_train.shape[0]
    model.fit_generator(gen([*zip(X_train[0:size],Y_train[0:size])],batch_size)
(),steps_per_epoch=int(size/batch_size),
                epochs=20, callbacks=[tbCallBack])
    batch_size=30
    size=X_test.shape[0]
    scores = model.evaluate_generator(gen(zip(X_test[0:size],Y_test[0:size]),bat
ch_size)(),steps=int(size/batch_size),workers=5, use_multiprocessing=True,verbos
e=True)
    print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    acc.append(scores[1])
```

now we set dimension to 10

In [ ]:

```
acc2=[]
for i in range(1,20):
    from tensorflow import set_random_seed
    set_random_seed(2)
    maxLen=300
    seed=2017
    tbCallBack = keras.callbacks.TensorBoard(log_dir='/data/logs', histogram_fre
q=0, write_graph=True, write_images=True)
    init=keras.initializers.glorot_normal(seed=seed)
    clear_logs()
    batch_size=30
    np.random.seed(1)
    input1 = keras.layers.Input(batch_shape=(batch_size,maxLen,512),
                                shape=(maxLen,512),dtype='float32')
    input2 = keras.layers.Input(batch_shape=(batch_size,maxLen,512),
                                shape=(maxLen,512),dtype='float32')
    #here we set dimension to 10
    dim1=10
    stateful=False
    X1=input1
    lstm1, state_h1, state_c1 = keras.layers.LSTM(dim1,
                            return_sequences=True,
                            return_state=True,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(X1)


    X1 = keras.layers.LSTM(dim1,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(lstm1)

    X2=input2
    lstm2, state_h2, state_c2  = keras.layers.LSTM(dim1,
                            return_sequences=True,
                            return_state=True,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(X2)

    X2 = keras.layers.LSTM(dim1,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            #recurrent_activation='selu',
                            activation='selu'
                           )(lstm2)


    added = keras.layers.Concatenate(axis=-1)([X1, X2])
    x3=added
    x3 = keras.layers.Dense(512, activation='selu',kernel_initializer=init)(x3)
```

```
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dropout(0.5,seed=seed)(x3)
    x3 = keras.layers.Dense(128, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dropout(0.3,seed=seed)(x3)
    x3 = keras.layers.Dense(64,kernel_initializer=init, activation='selu')(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dropout(0.2,seed=seed)(x3)
    x3 = keras.layers.Dense(32, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dense(16, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    out = keras.layers.Dense(1,kernel_initializer=init)(x3)
    model = keras.models.Model(inputs=[input1, input2], outputs=out)

    sgd = optimizers.SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
    nadam = optimizers.Nadam(lr=0.45)
    adadelta=keras.optimizers.Adadelta(lr=0.5, rho=0.95, epsilon=None, decay=1e-
6)

    model.compile(loss='binary_crossentropy',
                  optimizer=adadelta,
                  metrics=['accuracy'])
    size=5000
    size=X_train.shape[0]
    model.fit_generator(gen([*zip(X_train[0:size],Y_train[0:size])],batch_size)
(),steps_per_epoch=int(size/batch_size),
                epochs=20, callbacks=[tbCallBack])
    batch_size=30
    size=X_test.shape[0]
    scores = model.evaluate_generator(gen(zip(X_test[0:size],Y_test[0:size]),bat
ch_size)(),steps=int(size/batch_size),workers=5, use_multiprocessing=True,verbos
e=True)
    print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    acc2.append(scores[1])
```

here we keep dimension to 10 but expose lower level layers

In [ ]:

```
acc3=[]
for i in range(1,20):
    from tensorflow import set_random_seed
    set_random_seed(2)
    maxLen=300
    seed=2017
    tbCallBack = keras.callbacks.TensorBoard(log_dir='/data/logs', histogram_fre
q=0, write_graph=True, write_images=True)
    init=keras.initializers.glorot_normal(seed=seed)
    clear_logs()

    batch_size=30
    np.random.seed(1)

    input1 = keras.layers.Input(batch_shape=(batch_size,maxLen,512),
                                shape=(maxLen,512),dtype='float32')
    input2 = keras.layers.Input(batch_shape=(batch_size,maxLen,512),
                                shape=(maxLen,512),dtype='float32')
    dim1=10
    stateful=False
    X1=input1
    lstm1, state_h1, state_c1 = keras.layers.LSTM(dim1,
                            return_sequences=True,
                            return_state=True,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(X1)

    X1 = keras.layers.LSTM(dim1,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(lstm1)

    X2=input2
    lstm2, state_h2, state_c2  = keras.layers.LSTM(dim1,
                            return_sequences=True,
                            return_state=True,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(X2)
    X2 = keras.layers.LSTM(dim1,
                            stateful=stateful,
                            kernel_initializer=init,
                            recurrent_initializer=init,
                            activation='selu'
                           )(lstm2)

    #here we use state from lower level LSTM
    added1 = keras.layers.Concatenate(axis=-1)([X1,state_h1, X2,state_h2])
    x3=added1
    x3 = keras.layers.Dense(512, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dropout(0.5,seed=seed)(x3)
```

```
    x3 = keras.layers.Dense(128, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dropout(0.3,seed=seed)(x3)
    x3 = keras.layers.Dense(64,kernel_initializer=init, activation='selu')(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dropout(0.2,seed=seed)(x3)
    x3 = keras.layers.Dense(32, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    x3 = keras.layers.Dense(16, activation='selu',kernel_initializer=init)(x3)
    x3=keras.layers.BatchNormalization(axis=-1)(x3)
    out = keras.layers.Dense(1,kernel_initializer=init)(x3)
    print(out)
    model = keras.models.Model(inputs=[input1, input2], outputs=out)

    sgd = optimizers.SGD(lr=0.0001, decay=1e-6, momentum=0.9, nesterov=True)
    nadam = optimizers.Nadam(lr=0.45)
    adadelta=keras.optimizers.Adadelta(lr=0.5, rho=0.95, epsilon=None, decay=1e-
6)

    model.compile(loss='binary_crossentropy',
                  optimizer=adadelta,
                  metrics=['accuracy'])
    size=5000
    size=X_train.shape[0]
    model.fit_generator(gen([*zip(X_train[0:size],Y_train[0:size])],batch_size)
(),steps_per_epoch=int(size/batch_size),
                epochs=20, callbacks=[tbCallBack])
    batch_size=30
    size=X_test.shape[0]
    scores = model.evaluate_generator(gen(zip(X_test[0:size],Y_test[0:size]),bat
ch_size)(),steps=int(size/batch_size),workers=5, use_multiprocessing=True,verbos
e=True)
    print("\n%s: %.2f%%" % (model.metrics_names[1], scores[1]*100))
    acc3.append(scores[1])
```

here we report results from experiment 2.1(above code was run not in notebook and results were saved to files expr[n])

In [20]:

```
sns.set(color_codes=True)
```

In [21]:

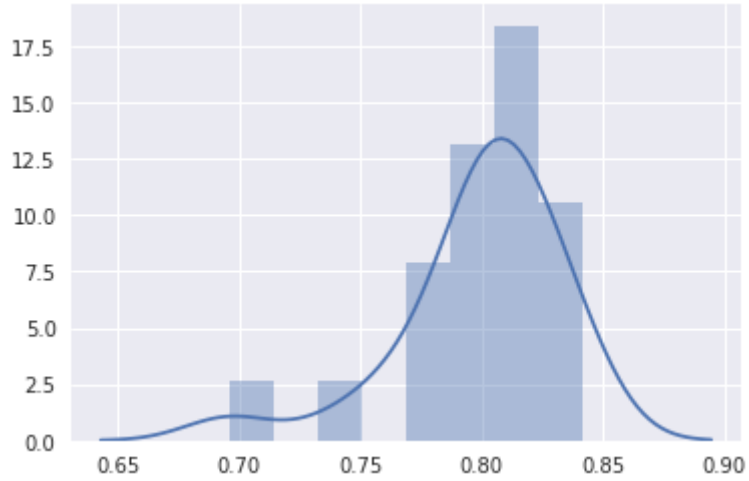
```
a=pd.read_csv('/data/expr1.txt',header=-1)
b=pd.read_csv('/data/expr2.txt',header=-1)
c=pd.read_csv('/data/expr3.txt',header=-1)
```

In [22]:

```
sns.distplot(a) # 6
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1e01dc97f0>
```

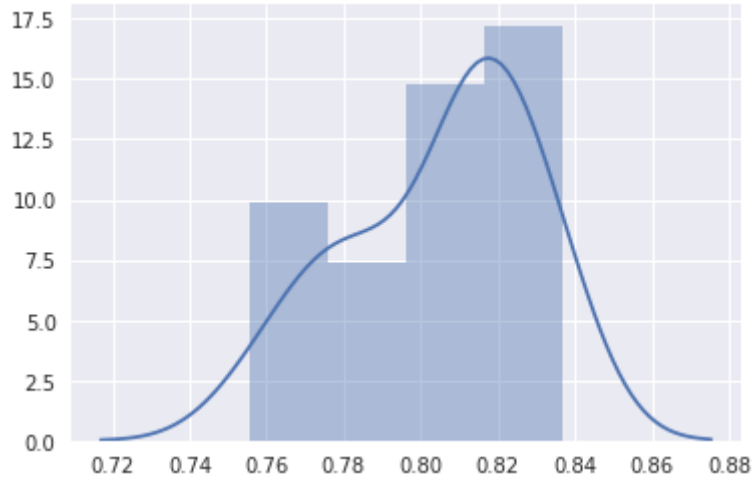


In [23]:

```
sns.distplot(b) #10
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d9e8c0860>
```

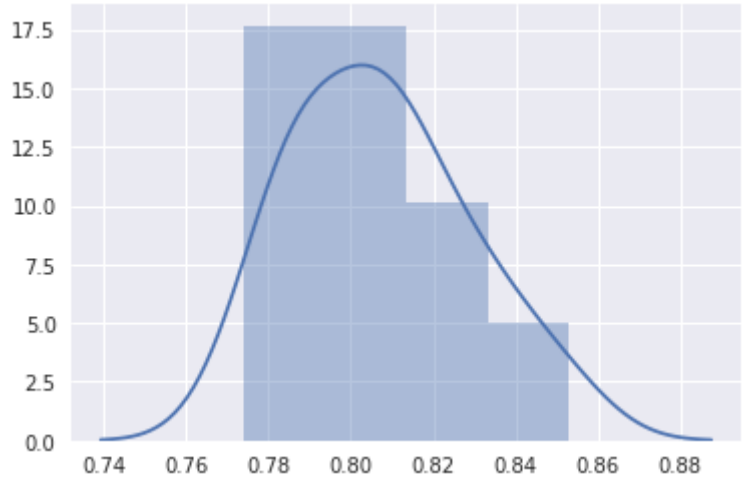


In [24]:

```
sns.distplot(c) #10$
```

Out[24]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1d9e8427b8>
```

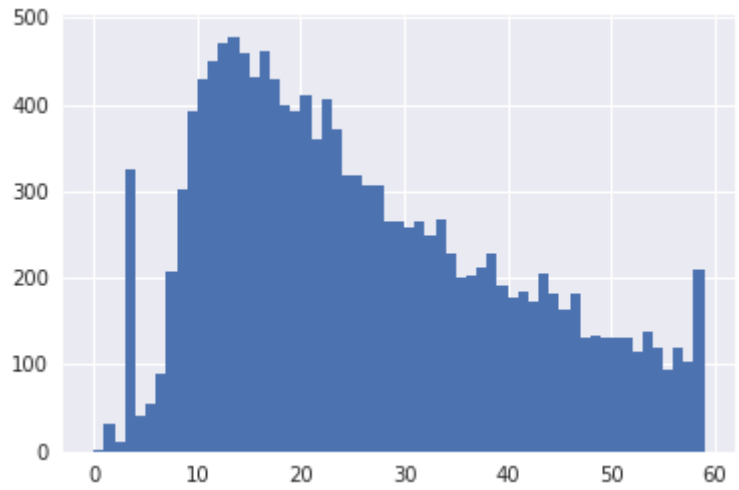


In [25]:

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
```

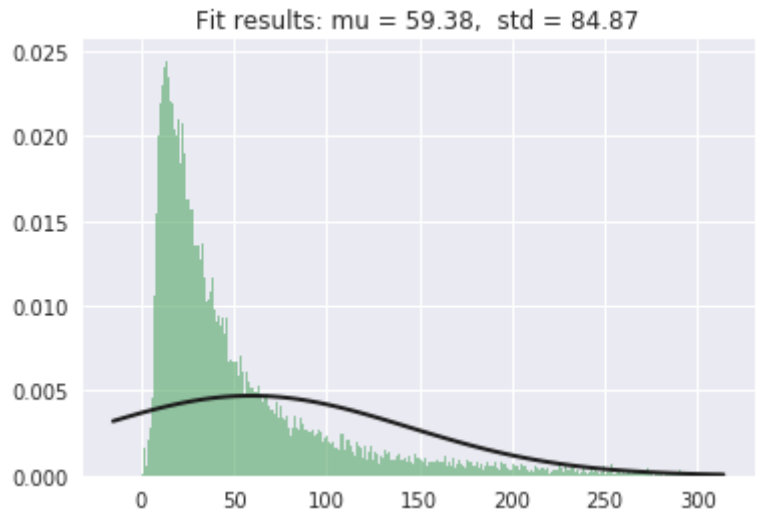
In [26]:

```
n, bins, patches = plt.hist(len2, bins = range(0,60))
plt.show()
```



In [27]:

```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt


# Generate some data for this demonstration.
data = len2

# Fit a normal distribution to the data:
mu, std = norm.fit(data)

# Plot the histogram.
plt.hist(data, bins=range(0,300), density=True, alpha=0.6, color='g')

# Plot the PDF.
xmin, xmax = plt.xlim()
x = np.linspace(xmin, xmax, 100)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'k', linewidth=2)
title = "Fit results: mu = %.2f,  std = %.2f" % (mu, std)
plt.title(title)

plt.show()
```



In [28]:

```
maxLen=max(max(len0),max(len1),max(len2))
```

In [29]:

```
maxLen
```

Out[29]:

2738