Question Answering System

A Master's Project Report submitted to Santa Clara University in Fulfillment of the Requirements for the

Course COEN-296: Natural Language Processing

Instructor: Ming-Hwa Wang

Department of Computer Science and Engineering

by

Surag Suresh Yalaburg

Lakshmi Shankarrao

Fall Quarter 2016

Acknowledgements

We would like to extend our gratitude towards Prof. Ming-Hwa Wang, who inspired us to make this project. We would also like to thank Santa Clara University which provided us the opportunity to learn the subject and apply its concept into making this project.

Table of Contents

List Of Tables and Figures

Abstract

In this project we apply concepts of Deep Learning(DL) for question answering task. This approach avoids the tedious task of numerous feature extraction that are done in traditional linguistic tools. Here we use bidirectional Long Short-Term Memory models to generate embeddings of questions and answer and measure cosine similarity to compute the distance between questions and answers pairs which would be used for appropriate answer selection. The model is to involve attention mechanism which would increase the efficiency of the system. The models are experimented on different dataset such as InsuranceQA, bAbI project and the results are captured and analysis are done.

Introduction

Objective:

Build a Question Answering System using neural networks.

What is the problem

There are millions and billion pieces of data available, but making the right information accessible when needed is very important. Getting the right documents to read and further getting a direct answer to one's question from the set of documents is a challenging task. A question answering system is concerned with building systems that automatically answer questions posed by humans in a natural language. A QA system is generally programmed to pull answers from a structured database or an unstructured collection of natural language document/s. There are two types of question answering. One is Closed-domain question answering system which basically deals with questions under a specific domain and the other is Open-domain question answering which is concerned with questions about nearly anything.

Why is this a project related the this class

Question Answering is one of the fundamental Natural Language Processing(NLP) Problem which has application in various fields of science. Traditionally, Automated Question answering systems involve various aspects of NLP such as Morphological analysis, Lexical analysis, Syntactic analysis and semantic analysis.

Why other approaches are no good and why the chosen approach is better

Neural network are increasingly gaining focus in NLP related tasks. Over the past few years, neural networks have re-emerged as powerful machine-learning models, yielding state-of-the-art results in fields such as image recognition and speech processing. More recently, neural network models started to be applied also to textual natural language signals, again with very promising results. (Ref: A Primer on Neural Network Models for Natural Language Processing- Yoav Goldberg, October 5, 2015). Feature engineering is needed in conventional machine learning tasks. Whereas it is not needed for neural networks. What happens inside the hidden layers of the neural networks is unknown to

anybody. It learns all possible features. May be ones that humans might not have even thought of. Neural networks are powerful learners, providing opportunities ranging from non-linear classification to non-Markovian modeling of sequences and trees. (Ref: A Primer on Neural Network Models for Natural Language Processing- Yoav Goldberg, October 5, 2015).

Statement of the problem

Build a program that can answer questions posed by humans in natural language. The task comprises of reading sentences which are part of a document. And the system must be capable of answering questions based on the text given to it. The output is expected to be a single word or one complete sentence.

Area or Scope of Investigation

We plan to build a QA system using neural networks to help interprets a question and provide a suitable answer from the document or a story given. Enabling a machine to process a question and a document and provide best answers to the input question from the given document. We plan to explore the following datasets.

1. Babi Dataset: This is a dataset released by facebook. It has 20 tasks for testing text understanding and reasoning.
2. MCTest(Machine Comprehension Test) dataset: This is a dataset from MSR, which contains 660 stories, each story has 4 human asked questions (Natural Language Question), and for each question, there are 4 candidate answers.
3. Children's Book Test (CBT): A dataset from FAIR, which contains stories from children's books. Each story in this dataset is a 20 consecutive sentences from children's books, and remove a word from the consecutive 21st sentence, as the question, or query.
4. InsuranceQA dataset:This corpus contains questions and answers collected from the website Insurance Library. This dataset is provided by "Applying Deep Learning to Answer Selection: A Study and An Open Task"[2].
5. CNN/Daily Mail dataset: This is released by Google DeepMind, which the largest (AFAIK) QA dataset. CNN dataset contains over 90K of of CNN news, and averagely has 4 queries per story, which gives 380K of story-question pairs; Daily Mail has about 200K new stories, and also, each story has 4 queries, which totally gives 880K story-question pairs.

All the above datasets in general contain a story, a query and a set of answer options to the query. The dataset is divided into training, development and test datasets. The model is trained with the training dataset. And then tested the testing dataset. We plan to train the

model to handle tasks involving like Single Supporting Fact, Two or Three Supporting Facts, Yes/No Questions, Counting and Lists/Sets, Basic Coreference, Conjunctions and Compound Coreference etc., and analyse and compare their performance.

Theoretical Bases and Literature Review:

Theoretical background of the problem

A recurrent neural network (RNN) is an artificial neural network where connections between units form a directed cycle[1]. An RNN can deal with variable-length sequence input. X = X1,X2,..XT and it is useful to tasks such as handwriting recognition or speech recognition. An RNN is a single model used to summarizing information from the past and providing it to the next stage. Inputs at each stage Xi and the summary of all events till now coming from the previous stage. Outputs are the predictions and connects to the next stage(i.e., provides a summarized information of the past to the next stage.). RNN suffers from a few drawbacks like not exploiting the future context and memory loss due to the Vanishing Gradient Problem.

LSTM or Long Short-Term Memory (LSTM) are RNN that are used to deal with variable length continuous sequence inputs. These are specialized to carry history in long sequence of input. LSTMs do this by storing the recurrent hidden vector which is dependent previous hidden vector. LSTM is one of the popular RNN technique to mitigate the vanishing gradient problem of RNN.(Hochreiter, Sepp and Schmidhuber, Jurgen. Long short-term memory.Neural Computation, 1997.). LSTM model is as follows,(Graves, Alex, Mohamed, Abdel-rahman, and Hinton, Geoffrey. Speech recognition with deep recurrent neural networks.In IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2013.). They use 3 gates to control the flow of information through the LSTM cell and reset the information when needed.

A Bidirectional LSTM model uses 2 LSTMs to help capture the context information from the past as well as the present. The data is processed from two directions with two separate hidden layers. The output of current time step is then generated by combining both layers' hidden vector.

Deep learning models have seen great progress in natural language processing tasks like semantic analysis. Using word embeddings a neural network model can be trained to extract the semantic relation between two words.

Pooling is a form of non-linear down-sampling. The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. Max pooling is a sample-based discretization process. The objective is to down-sample an input

representation (image/text, hidden-layer output matrix, etc.), reducing it's dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned. Max-pooling is generally said to be better than other forms of pooling because it extracts more local values for each dimension, so that more local information can be reflected on the output embeddings.

Related research to solve the problem

The different models that address the Question answering problem are discussed below.

1. Answer Sentence Selection with Stacked BLSTM

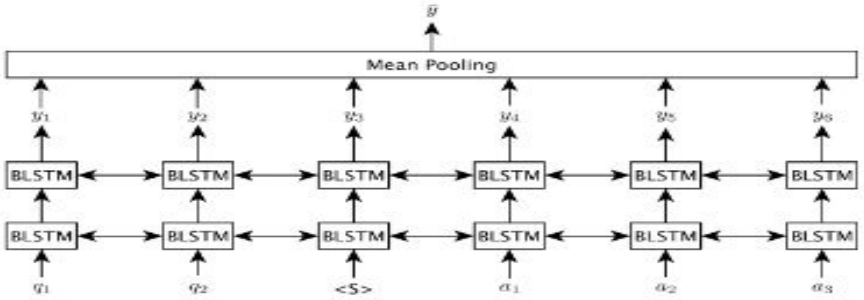

Fig 1

In this model words in input sentences are first converted to vector representations learned from word2vec. A special start symbol(<S>) is inserted between Question and Answer for differentiation. Then the Question and Answer word vectors are sequentially read by BLSTM from both directions. For each time step in the BLSTM layer, the hidden vector or the output vector is generated by combining the cell memory vectors from two LSTM of both sides. The final output of each time step is the label indicating whether the candidate answer sentence should be selected as the correct answer sentence for the input question. This objective encourages the BLSTMs to learn a weight matrix that outputs a positive label if there is overlapping context information between two LSTM cell memories. Mean pooling is applied to all time step outputs during the training. During the test phase, the mean, sum and max poolings as features are collected. And the final answer vector is extracted.

2. QA-LSTM model

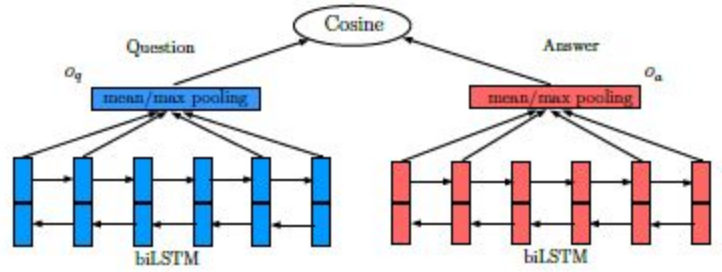

Fig 2

BLSTM generates distributed representations for both the question and answer independently, and then utilize cosine similarity to measure their distance. Then representations based on the word-level BLSTM outputs are generated using Average pooling or Max pooling or Concatenation of the last vectors on both directions. Max Pooling is provides better performance. Architectures, in which both question and answer sides share the same network parameters, is significantly better than the one that the question and answer sides own their own parameters separately, and converges much faster.

3. QA-LSTM with CNN model.

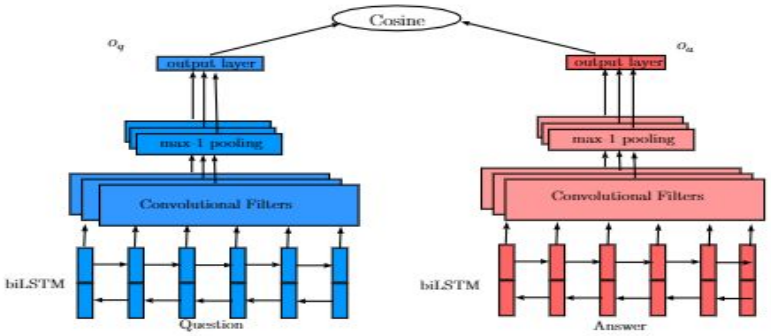

Fig 3

The above mentioned models is said to produce very good results but a simple pooling layer may suffer from the incapability of keeping the local linguistic information. Thus the the question and answer representations are generated using a CNN structure built on the outputs of BLSTM, in order to give a more composite representation of questions and answers. Unlike the traditional forward neural network, where each output is interactive with each input, the convolutional structure only imposes local interactions between the inputs within a filter size m. Same as typical CNNs, a max-k pooling layer is built on the top of the convolutional layer. The top-k values from each convolutional filter are extracted by doing so. These top-k values indicate the highest degree that a filter matches the input

sequence. In the end, two output vectors with dimension of kN for the questions and answers respectively are generated from N parallel filters, with different parameter initialization. The convolutional layer gets N-dimensional output vectors from N filters and chooses top-k values from it.

4. Attention-based QA-LSTM Model

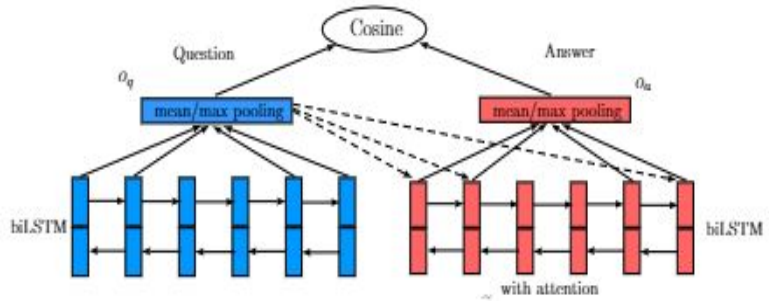

Fig 4

In order to better distinguish candidate answers according to the question, we introduce a simple but efficient attention model to this framework for the answer embedding generation according to the question context. Here prior to extracting representations using the average or mean pooling, each BLSTM output vector will be multiplied by a softmax weight, which is determined by the question embedding from BLSTM.

Advantage/disadvantage of those research

All the deep learning frameworks we studied for answer selection does not require any feature engineering, linguistic tools, or external resources. This basic framework based on bidirectional long short term memory (biLSTM). The convolutional neural network (CNN) structure on top of biLSTM helps overcome the incapability of keeping the local linguistic information. The QA-LSTM attention model helps to provide contextual information regarding the question while generating the answer representations. This the authors claim that produces better results. The model using stacked BLSTM is also very efficient since the output vectors are generated by using both question and answer representations together.

Our solution to solve this problem

We plan to implement a QA-LSTM model with attention using BiLSTMs. The Answer and Question representations generated are filtered using Max pooling. And the cosine difference between the selected question and answer representations is taken to choose the best answer.

Hypothesis/Goals

In this project (we are proposing), our goal is to build a framework which uses bidirectional LSTM based RNN models that train on both question and answers data and then measure the distance between the pairs of question and answers to get the similarity metrics.

In the following sections we shall explain the approach in stages,

LSTM

The LSTM model helps alleviate the Vanishing Gradient problem of the Recurrent Neural Networks hence help provide a better memory. This is achieved with the help of gates. Given an input sequence x = {x(1), x(2),.....,x(n)}, where x(t) is an E-dimension word vector. The hidden vector h(t) (the size is H) at the time step t is updated as follows.

$$
\begin{aligned}
i_t &= \sigma(\mathbf{W}_i\mathbf{x}(t) + \mathbf{U}_i\mathbf{h}(t-1) + \mathbf{b}_i) && (1)\\
f_t &= \sigma(\mathbf{W}_f\mathbf{x}(t) + \mathbf{U}_f\mathbf{h}(t-1) + \mathbf{b}_f) && (2)\\
o_t &= \sigma(\mathbf{W}_o\mathbf{x}(t) + \mathbf{U}_o\mathbf{h}(t-1) + \mathbf{b}_o) && (3)\\
\bar{C}_t &= \tanh(\mathbf{W}_c\mathbf{x}(t) + \mathbf{U}_c\mathbf{h}(t-1) + \mathbf{b}_c) && (4)\\
C_t &= i_t * \bar{C}_t + f_t * C_{t-1} && (5)\\
\mathbf{h}_t &= o_t * \tanh(C_t) && (6)
\end{aligned}
$$

All basic LSTM have three gates (input i, forget f and output o), and a cell memory vector C,is the sigmoid function. The input gate determines how the state of the memory cell is altered based on the in vector x(t). The output gate determines the effect on the that the particular memory cell has on the outputs. The forget gate determines whether to remember or forget the cell's previous state. $W \in R^{H \times E}$, $U \in R^{H \times H}$ and $b \in R^{H \times 1}$ are the

network parameters. W - input weight matrix, U - recurrent weight matrix, b - bias, σ is the logistic sigmoid function

biLSTM

Bidirectional Long Short-Term Memory (biLSTM)are variation of LSTM in which, it uses both the previous and the future state by processing the input in two directions, then generate two independent LSTM output vectors. These models do a better job than Single direction LSTMs. As Single direction LSTMs does not make use of the contextual information from the future input data. The input sequence is processed in the forward direction and also in the reverse direction. Output at any given step is the concatenation of the two output vectors.

QA-LSTM

The simple model of our project is shown in Fig 1. The BiLSTM models generates representations for question answer separately and then compute cosine similarity to check the distance between them. Following the same ranking loss in (Feng et al., 2015), we define the training objective eq(7) as a hinge loss.

$$L = \max\{0, M - cosine(q, a_+) + cosine(q, a_-)\} \quad (7)$$

where, a+ : a ground truth answer, a- : incorrect answer randomly chosen from the entire answer space, and M is constant margin.

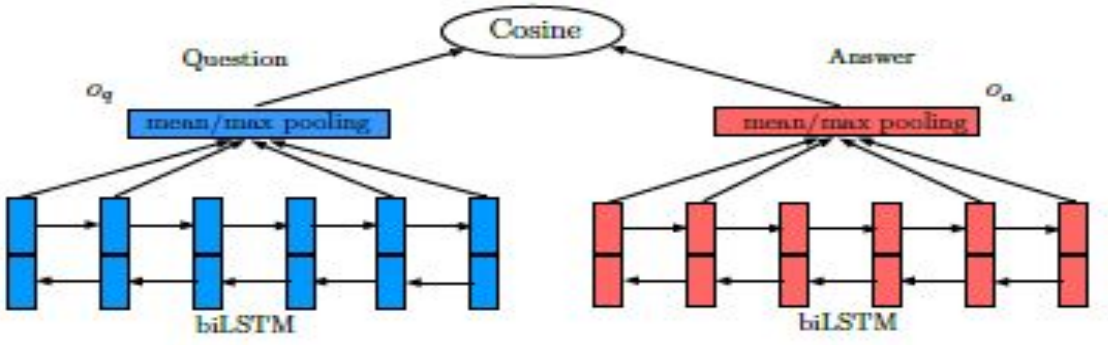

Fig 1: QA - LSTM Model

ATTENTION-BASED QA-LSTM

In this variation we use a simple attention model which is based on the question, for the answer vector generation. An attention mechanism is used for dynamically aligning the informative part of the the answers to the question. This is the same strategy has been used in machine translation (Bahdanau et al., 2015; Sutskever et al., 2014) and factoid question answering (Hermann et al., 2015).

Our world level attention model will be similar/replication to the work of (Tan, Ming et al., 2015). Fig 2 shows the structure.

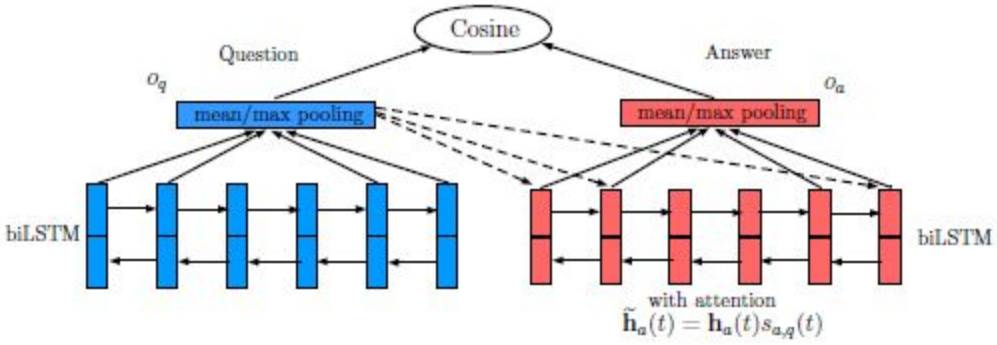

Fig 4: QA-LSTM with attention

The output vector of biLSTM will be multiplied by a softmax weight, which is computed using the question embedding from biLSTM. The output vector of biLSTM on the answer side at time step t, $h_a(t)$, and the question embedding, o_q, the new vector $\widehat{h}_a(t)$ for each answer is given below.

$$\mathbf{m}_{a,q}(t) = \tanh(\mathbf{W}_{am}\mathbf{h}_a(t) + \mathbf{W}_{qm}\mathbf{o}_q) \quad (9)$$

$$s_{a,q}(t) \propto \exp(\mathbf{w}_{ms}^T\mathbf{m}_{a,q}(t)) \quad (10)$$

$$\tilde{\mathbf{h}}_a(t) = \mathbf{h}_a(t)s_{a,q}(t) \quad (11)$$

Where W_{am},W_{qm} and w^T_{ms} are attention parameters. We will see in our experiments how this attention will help in increasing the efficiency.

Methodology

Data Sets

(how to generate/collect input data)

In this work we will try to experiment our model on multiple data set such as, InsuranceQA, bAbI project(The 20 QA bAbI tasks, The 6 dialog bAbI tasks, The Children's Book Test, The Movie Dialog dataset, The WikiMovies dataset ,The Dialog-based Language Learning dataset,The Simple Questions dataset) and see how our model responds to the different data sets.

All these dataset either provide training sets, validation sets and test sets separately or just have a one heterogenous sets of questions, answers and story. In both the case we will divide the data sets into three parts as training, validation and test set to train and evaluate the model for accuracy.

Setup

(how to solve the problem)

We are planning to implement our model using tensorflow framework and run the model on single GPU(nvidia GeForce GT 730M). We will be build a biLSTM with attention which is explained in equation(9) (10) and (11). We will be using the accuracy that we get on the validation set to compute the best epoch and best hyper-parameter which we will use for testing. The code will be in python3.5. Apart from tensorflow we will be using numpy, scikit learn, pandas, matplotlib and many more python based libraries.

Results and Analysis

(how to generate output & how to test against hypothesis)

We will conduct experiments on different variation of our model and compare the results with each other and and try to deduce some insights on why the specific model for specific data set was good/bad at predictions.

Implementation

Code

```
"""
This code is a modified version of the code from this link:
https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/recurrent_network.py
"""
import preprocess_sample_3 as pre_pro
import skip_gram
import tensorflow as tf
import numpy as np

# set random seed for comparing the two result calculations
tf.set_random_seed(1)

o = skip_gram.SkipGram()
p = pre_pro.Preprocess()
content, ans = p.read_input("train_dev_data", "train_dev_ans")
ip_sgm, words = p.generate_ip_for_SGM(content)

o.build_dataset(words)
o.run_skip_gram()

labels = p.create_labels_for_LSTM(ans)
input_embed = p.get_ip_SQA_for_LSTM(o)


def get_batch(batch_size, step):
   s = step * batch_size
   b_x = np.array(input_embed[s:s + (batch_size * n_steps)])
   b_x = b_x.reshape([batch_size,n_steps,n_inputs])
   b_y = np.array(labels[s:s + batch_size])

   return b_x, b_y


# hyperparameters
lr = 0.001
training_iters = 100000
batch_size = 30

n_inputs = 300  # word vector size
n_steps = 300  # time steps / Story + q + A 1-4
n_hidden_units = 512  # neurons in hidden layer
n_classes = 4  # one of the four options(0 - 3 )
total_steps = 1800 # total number of question
```

```
# tf Graph input
x = tf.placeholder(tf.float32, [None, n_steps, n_inputs])
y = tf.placeholder(tf.float32, [None, n_classes])

# Define weights
weights = {
   # (300, 512)
   'in': tf.Variable(tf.random_normal([n_inputs, n_hidden_units])),
   # (512, 4)
   'out': tf.Variable(tf.random_normal([n_hidden_units, n_classes]))
}
biases = {
   # (512, )
   'in': tf.Variable(tf.constant(0.1, shape=[n_hidden_units, ])),
   # (4, )
   'out': tf.Variable(tf.constant(0.1, shape=[n_classes, ]))
}


def RNN(X, weights, biases):

   # X ==> ((30 batch * 300 steps), 300 inputs)
   X = tf.reshape(X, [-1, n_inputs])

   # mul
   # X_in = ((30 batch * 300 steps), 512 inputs)
   X_in = tf.matmul(X, weights['in']) + biases['in']

   # reshape
   # X_in ==> (128 batch, 300 steps, 512 hidden)
   X_in = tf.reshape(X_in, [-1, n_steps, n_hidden_units])

   # cell

   # basic LSTM Cell.
   lstm_cell = tf.nn.rnn_cell.BasicLSTMCell(n_hidden_units, forget_bias=1.0,
state_is_tuple=True)
   _init_state = lstm_cell.zero_state(batch_size, dtype=tf.float32)

   # dynamic_rnn receive Tensor (batch, steps, inputs) or (steps, batch, inputs) as X_in.
   outputs, final_state = tf.nn.dynamic_rnn(lstm_cell, X_in, initial_state=_init_state,
time_major=False)

   # unpack to list [(batch, outputs)..] * steps
   outputs = tf.unpack(tf.transpose(outputs, [1, 0, 2]))

   # states is the last outputs
   results = tf.matmul(outputs[-1], weights['out']) + biases['out']
```

```
    return results

# prediction for the input batch
pred = RNN(x, weights, biases)
# calculate the entropy loss
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y))
# optimize the model using Adam Optimizer and optimize the weights
train_op = tf.train.AdamOptimizer(lr).minimize(cost)

# Calculate the accuracy
correct_pred = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
accuracy = tf.reduce_mean(tf.cast(correct_pred, tf.float32))

init = tf.initialize_all_variables()
with tf.Session() as sess:
    sess.run(init)
    step = 0
    while step * batch_size < total_steps:
        batch_xs, batch_ys = get_batch(batch_size, step)
        batch_xs = batch_xs.reshape([batch_size, n_steps, n_inputs])
        sess.run([train_op], feed_dict={
            x: batch_xs,
            y: batch_ys,
        })

        # Calculate accuracy once for every 10 bactch
        # if step % 10 == 0:
        print(sess.run([accuracy,cost], feed_dict={
            x: batch_xs,
            y: batch_ys,
        }))
        step += 1


# Plot the skip gram plot
num_points = 200
o.plot(num_points)
```

Design document and flowchart

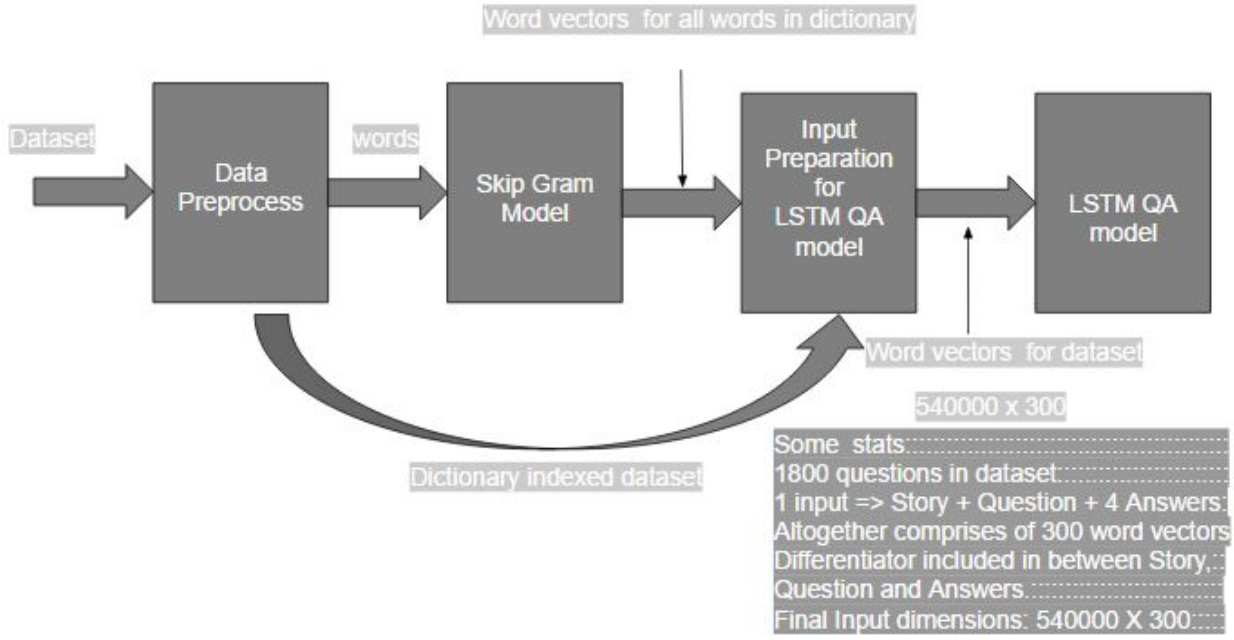

Fig 5: Flowchart

The design for our project was divided into 4 parts.

- Pre-processing input so that we can generate Word2Vec(Skip-gram) representations of words which could be used to make sentence embeddings for SGM. Also form the lexicon(dictionary) of the words that are present the dataset. Below block diagram explains this
- Generating word vectors for all words that is present in the dataset. We have used the Word2Vec(Skip-gram model) for this purpose.

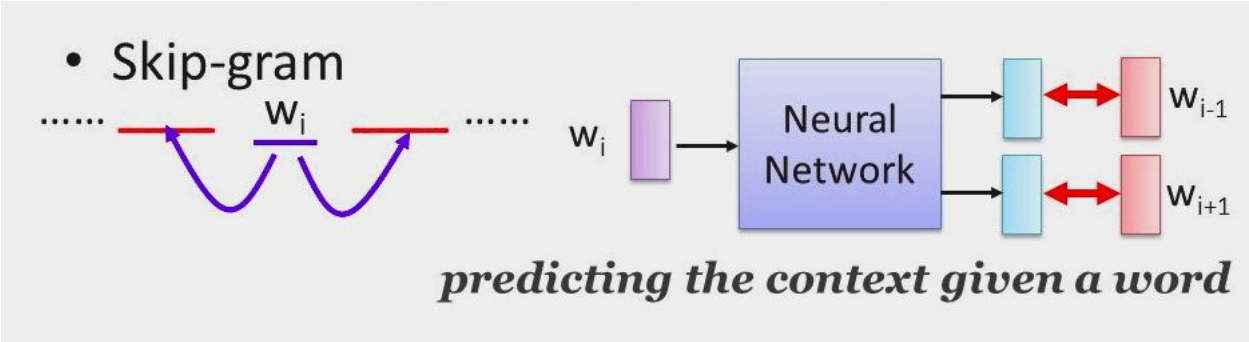

Fig 6: Skip Gram Model

- Using the word vectors and the dictionary indexed dataset we generate word vector input for the dataset and pass it to the RNN(LSTM network). Details about the model is explained in the next section.

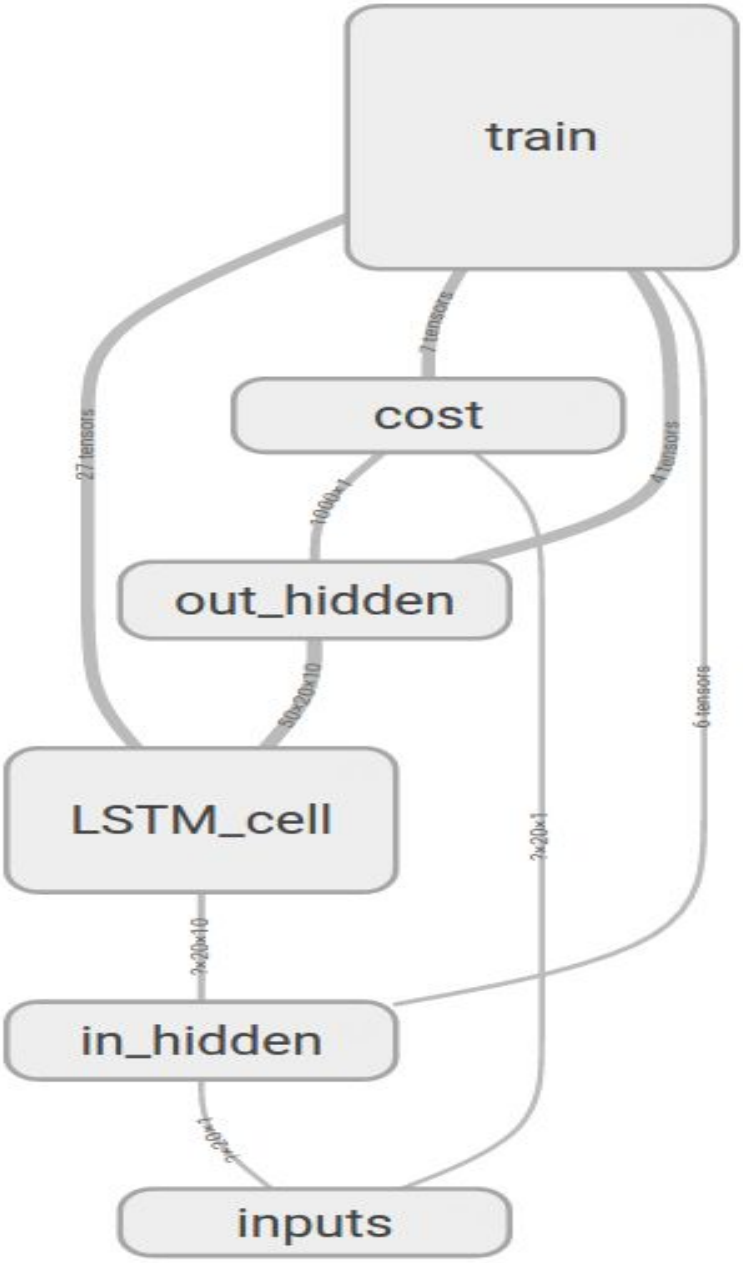

Fig 7: LSTM model

Data Analysis and Discussion

MCTest dataset is a free data set that has 450 stories each of which has multiple questions and the each questions has 4 choices answer choice in which one of them is correct and others are wrong. The data was gathered using Mechanical Turk. The one of example is as given below,

James the Turtle was always getting in trouble. Sometimes he'd reach into the freezer and empty out all the food. Other times he'd sled on the deck and get a splinter. His aunt Jane tried as hard as she could to keep him out of trouble, but he was sneaky and got into lots of trouble behind her back.
One day, James thought he would go into town and see what kind of trouble he could get into. He went to the grocery store and pulled all the pudding off the shelves and ate two jars. Then he walked to the fast food restaurant and ordered 15 bags of fries. He didn't pay, and instead headed home.
His aunt was waiting for him in his room. She told James that she loved him, but he would have to start acting like a well-behaved turtle.After about a month, and after getting into lots of trouble, James finally made up his mind to be a better turtle.
1) What is the name of the trouble making turtle?
A) Fries
B) Pudding
C) James
D) Jane
2) What did James pull off of the shelves in the grocery store?
A) pudding
B) fries
C) food
D) splinters
3) Where did James go after he went to the grocery store?
A) his deck
B) his freezer
C) a fast food restaurant
D) his room
4) What did James do after he ordered the fries?
A) went to the grocery store
B) went home without paying
C) ate them
D) made up his mind to be a better turtle

In Data Analysis we have done are cleaning the text data so that we can use for our project, creating lexicon of all unique words that is present in the complete data set and finally convert the lexicon into word vector using skip gram model.

In cleaning the data we had to taken out all punctuations, unwanted spaces and characters which are not part of the english dictionary.

In creating lexicon we had to read the whole text data set and get the unique words and make a data file that is has only words.

Using this we had to create a word vector. First we tried the bag of words model. But word vector generated by bag of words model did not have the semantic or syntactic information about the words so we have to look for better model. Hence we opted for Skip-gram model which is word vector representation for textual words which is generated by the Deep neural network.

Using this word vector we generate embeddings for the dataset which would take the story question and the answer choice which are data inputs for the recurrent neural network and predict the answer. The model is LSTM RNN which would learn from the embeddings.

Output generation

Input preprocessing involved cleaning involved removal of unwanted material from the dataset. The the data needed to be processed to remove unwanted punctuations and obtain words in the order of occurrence in the dataset. This was needed to build the lexicon. The lexicon was reordered in the order of the max frequency of occurrence. Later a reverse dictionary was built to enable ease of indexing a word. Next the whole data content was rewritten based on this index of the dictionary. This is the actual data we will be passing to the SGM.

Computation Graph and session operation for Skip Gram Model is described below. The dictionary indexed content is the input to the SGM. We chose batch_size = 128, embedding_size = 300, skip_window = 1, num_skips = 2 for the SGm we have used. We initialize all the variables discussed above and we generate a batch with the given skip window and number of skips chosen. We obtain the labels and the respective training dataset. And feed it to our neural network and obtain predictions. The tf.nn.embedding_lookup function will look up for the embeddings given the training dataset and the embeddings dimensions. These embeddings are of dimension vocabulary_size x embedding_size defined above and range from values -1 to 1. This is fed to the neural network and the output is multiplied with the weights and the biases are added and the output is obtained. This is compared with the labels to determine the loss. And the loss is fed to the optimizer to minimize the loss. We have used Adagrad Optimizer with a learning rate of 1.0.

Output

The skip gram model gives a output which represents the word vectors as follows. Below are some of the words that are near to each other in the context of particular data set.

Nearest to eight: seven, nine, six, four, five, three, zero, two,
Nearest to he: she, it, they, there, who, we, this, never,
Nearest to i: we, he, they, mary, james, she, theirs, melanie,
Nearest to she: he, they, it, i, midnight, we, hugo, snowball,
Nearest to back: off, ended, over, down, out, beginning, toward, float,
Nearest to her: his, him, their, margaret, me, its, chili, twisted,
Nearest to this: it, which, he, some, checkerboard, another, the, amritsar,
Nearest to been: become, be, already, was, recently, whispering, successfully,
Nearest to they: we, there, he, you, she, it, i, not,
Nearest to new: different, turkic, boasted, various, neared, stabilizers, deviating,
Nearest to called: named, used, distorted, considered, referred, known, vat,
Nearest to where: what, twist, beneath, artist, toilet, needs, performances, liking,

Nearest to dog: cow, smashing, babysit, hazel, drawing, witch, owner, mouse,

Below is a tensor board output generated by the skip gram model. It depicts the distribution of words in space(a Vector Space Model for the words in the dataset).

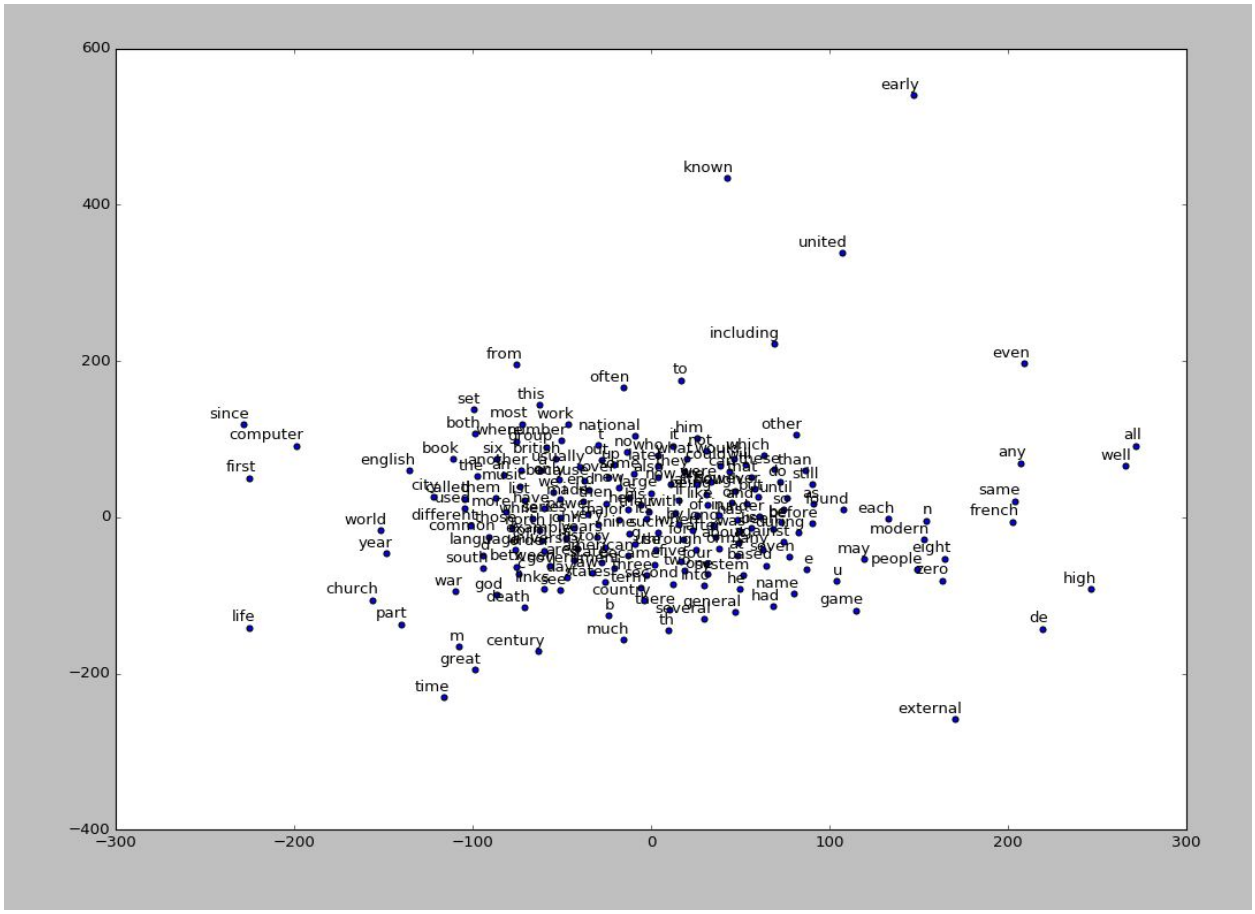

Fig 8: Word Vector Space model

The vector representation of one word is as given below.

“The” =

[-0.10771623 -0.08713842 0.03231252 -0.04112959 0.12192853 -0.07375532
-0.10610349 -0.07409572 -0.06865577 0.10982385 -0.01932012 -0.01754871
-0.16242403 0.03091805 -0.0236611 0.10541121 -0.07797282 -0.13612035
0.12207355 0.02171865 0.07394247 -0.11968254 0.06995879 0.03491663
-0.03399706 -0.05675896 0.08352144 0.01521992 0.04700584 0.07544895
0.0640968 0.11520148 0.07608264 -0.04500199 -0.00302214 -0.18527462
-0.02382737 -0.05388832 -0.114457 0.10991409 0.02414727 -0.04109648
0.08787432 -0.02101789 0.10125949 0.04609177 0.11370094 0.04372771
-0.0659379 -0.06589786 -0.00867271 0.05946467 0.07290234 0.04426249
0.13086618 -0.08741179 -0.07464718 -0.06333015 0.06026197 0.075174
0.0508201 -0.02082717 0.09218703 -0.10563675 0.04679077 0.11095957
-0.07731143 0.11152226 -0.07129365 0.08438376 0.12295292 0.09770688
-0.00122607 0.12426439 -0.07705146 -0.15394074 -0.16196689 -0.13202338
-0.06771396 0.0601143 0.06370313 0.01014148 0.13391793 -0.0489469
-0.04469048 -0.15215459 -0.0948875 -0.04088889 0.11519375 -0.06214551
-0.01907888 0.08190431 -0.13345332 0.12041677 0.13335384 0.13318285
-0.03807592 0.07797729 -0.03598532 -0.12675062 0.00989356 0.12395485
0.15320043 0.18589878 0.14680894 -0.077112 -0.10406934 0.17178342
-0.04968555 -0.09107879 0.04756921 -0.09697314 -0.120592 -0.04015396
-0.15957788 0.03014262 0.06686283 0.04753253 0.01522197 0.10813337

-0.01357032 -0.05207205 0.04069532 -0.03321727 -0.0258834 0.05395871
0.03483186 -0.08999643]

LSTM Question Answering model:

Input preparation for LSTM model is described below. We used the word embeddings we obtained from the Skip Gram Model and the dataset which is indexed by the dictionary created in the input processing step, to generate input embeddings for the LSTM model. Each input consists of a story, a question and four answer choices. We construct an input that holds 300 word vectors on the whole and each of which consists of 300 dimensions. Among the 300 words 219 belong to story, 15 belong to questions and 60 to answers where each of the answers correspond to 15 words.

The input is of the format "<S>+Story+<S>+Question+<S>+A1+<S>+A2+<S>+A3+<S>+A4"where we use a 300 dimension vector with all ones as the separator symbol <S>. Also labels are created from the answer file the label generated is an array of 4 dimensional vectors. Each vector is of the form

[0 1 0 0]. This label indicates that answer choice 'B' is the correct answer.

The computation graph and the session description for the LSTM QA model is described below. We chose learning rate = 0.001, num of training_iters = 100000, batch_size = 30, number of inputs = 300 which is same as the word vector size, number of steps = 300, number of hidden units = 512 number of classes = 4 and total steps = 1800 for our model. We generate a batch of input and corresponding labels. Here in our model we select 30 sets of questions for each batch. The model is given this input by first multiplying with the weights and adding the biases. It outputs some prediction. We take argmax of this prediction and compare with the labels and calculate the cost and the accuracy. Later we provide the cost to an optimizer to reduce the cost. We use Adam Optimizer here. The graph of accuracy obtained across batches is shown in the figure below.

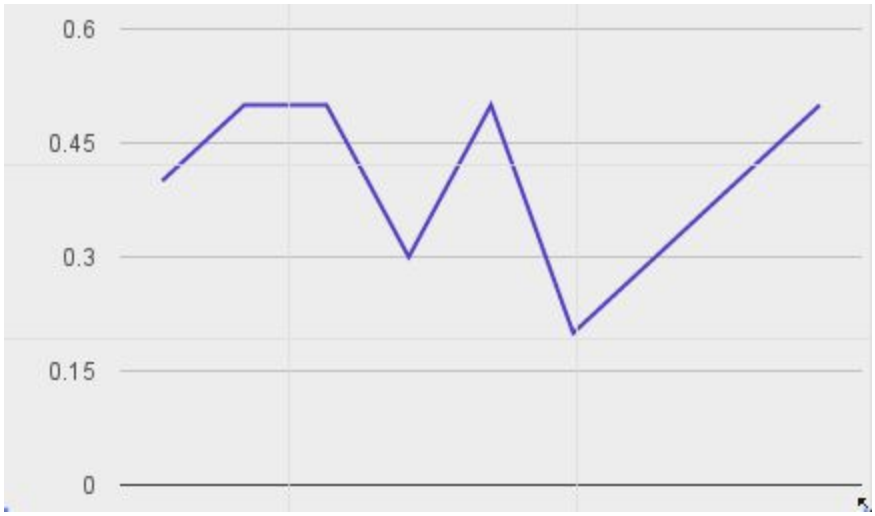

Fig 9: Accuracy plot

Output analysis

We saw the acuuracy ranging between 20% to 60%.The sample output of the project is as given below,
The output is batch wise
[Accuracy,Loss]
[0.43333337, 9.1849651]
[0.40000004, 6.956367]
[0.50000006, 5.3823442]
[0.33333337, 7.3978882]
[0.36666667, 4.4839745]
[0.30000001, 5.5185552]
[0.33333337, 2.9936876]
[0.26666671, 6.6921492]
[0.26666668, 6.105495]
[0.20000002, 8.5729513]
[0.33333337, 31.36186]
[0.23333335, 21.811695]
[0.33333337, 18.389278]
[0.26666668, 16.924873]
[0.33333337, 16.615463]
[0.33333337, 6.8296127]
[0.36666667, 12.750624]
[0.20000002, 15.518364]
[0.30000001, 7.9172506]
[0.40000004, 15.276219]
[0.23333335, 30.156597]
[0.23333335, 17.513439]
[0.40000004, 15.64605]
[0.20000002, 14.187802]
[0.40000004, 4.8628817]
[0.30000001, 8.0700626]
[0.30000001, 6.3062959]
[0.23333335, 16.49066]
[0.20000002, 15.956774]
[0.30000001, 16.061716]
[0.30000001, 9.7183285]
[0.30000001, 6.5377407]
[0.16666669, 7.4950838]

[0.20000002, 7.0059314]
[0.23333335, 9.5372372]
[0.43333337, 1.2171307]
[0.23333335, 9.7734261]
[0.33333337, 6.7862101]
[0.16666669, 12.330596]
[0.40000004, 8.3129349]
[0.26666668, 9.4911385]
[0.30000001, 4.7775865]
[0.4666667, 2.4099643]
[0.23333335, 4.2682471]
[0.23333335, 4.5539985]
[0.33333337, 5.5995607]
[0.33333337, 4.0388737]
[0.26666668, 6.6063185]
[0.20000002, 5.0251484]
[0.4333334, 4.7363453]
[0.40000004, 3.7464991]
[0.30000001, 2.3158572]
[0.20000002, 4.1808128]

Try to Analyse why it went wrong

The amount of knowledge needed to design a neural network is huge. We probably underestimated of the learning curve the neural network technology has. We learnt a lot about neural networks in the short timeline we had for this project.

We have decided to use Tensor flow for designing our neural network and the model that we were trying to build was two neural network side by side which we would learn question and answer separately. We spent a lot of time digging into this problem and then learnt that this particularly is not supported by tensor flow

We weren't able to use 2 RNN models in the same session. It gave errors pertaining to shared parameters between the 2 models. We tried declaring variables separately for both but just using 2 models in the same session, but it further gave room for more errors. We were not able to resolve we have raised a bug resolve request to google tensor flow group but given the time frame we were not able to come up with are round about solution for the problem.

So we resorted to use another model described in the paper "A Long Short-Term Memory Model for Answer Sentence Selection in Question Answering". The paper describes a model with bidirectional LSTM. We were able to implement a simpler version of the model using LSTMs. We plan to in future implement biLSTM for our QA system and hopefully obtain a better accuracy.

We also faced challenges with selecting the data set that works best for our model. We explored numerous data set like TREC-QA[], Insurance-QA[], BABI[] and MCTest data sets. Initially we wanted to use TREC-QA data set but the faced lot of difficulty to in using it for our particular model. We also tried using the Insurance-QA and we were finally able to use BABI and MCTest data sets and finally resolved to MCTest data set as it was the one which suits the best.

Abnormal case explanation (the most important task)

The inputs that we were using had lot of inconsistency without labels. So we had to correct it before using it.

The tensor flow bidirectional lstm has not been able to use as there was error that we were not able to resolve we have raised a bug resolve request to google tensor flow group but given the time frame we were not able to come up with are round about solution for the problem.

The accuracy while training shows fluctuation in the range of 0.2 to & 0.7 instead of showing consistent increment.

Conclusions and Recommendations

Summary and Conclusions

This project gave us a great start into the field of Neural networks. We obtained good learning wrt implementation of conventional and recursive neural networks using Tensor Flow. We were able to build an lstm neural network for answer prediction which, given a story and a question, predicts the correct answers from the answer choices. We got an accuracy between 20% to 70%.

Recommendations for future studies

We want to analyse the current model at length and modify it further to get better accuracies. We plan to implement the stacked - bidirectional LSTM based RNN network to see how the performance of the system changes. Instead of using skip-gram model for Word2Vec generation we can probably use 'GLOVE' model which is proven to a better vector representation of word embedding. After building these models we would like to compare their performance with existing conventional non-neural network NLP model.

Bibliography

1. D. Wang and E. Nyberg. A long short-term memory model for answer sentence selection in question answering. In ACL-IJCNLP, ACL 2015, July 26-31, 2015, Beijing, China, Volume 2: Short Papers, pages 707{712, 2015.
2. Daniel Cohen and W. Bruce Croft. “End to End Long Short Term Memory Networks for Non-Factoid Question Answering” ICTIR '16 Proceedings of the 2016 ACM International Conference on the Theory of Information Retrieval.
3. Tan, Ming; dos Santos, Cicero; Xiang, Bing; Zhou and Bowen. “LSTM-BASED DEEP LEARNING MODELS FOR NONFACTOID ANSWER SELECTION“. In eprint arXiv:1511.04108, 11/2015,
4. H. Palangi, L. Deng, Y. Shen, J. Gao, X. He, J. Chen, X. Song, and R. K. Ward. Deep sentence embedding using the long short term memory network: Analysis and application to information retrieval. CoRR, abs/1502.06922, 2015.
5. Wikipedia https://en.wikipedia.org/wiki/
6. Applying Deep Learning to Answer Selection: A Study and An Open Task Minwei Feng, Bing Xiang, Michael R. Glass, Lidan Wang, Bowen Zhou ASRU 2015
7. Jason Weston, Antoine Bordes, Sumit Chopra, Alexander M. Rush, Bart van Merrienboer, Armand Joulin & Tomas Mikolov “TOWARDS AI-COMPLETE QUESTION ANSWERING : A SET OF PREREQUISITE TOY TASKS”12/2015. https://arxiv.org/pdf/1502.05698v10.pdf
8. D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” ICLR2015, 2015. [Online]. Available: http://arxiv.org/abs/1409.0473
9. I. Sutskever, J. Martens, G. E. Dahl, and G. E. Hinton, “On the importance of initialization and momentum in deep learning,” in ICML (3)’13, 2013, pp. 1139–1147.
10. K. M. Hermann and P. Blunsom, “Multilingual models for compositional distributed semantics,” arXiv preprint arXiv:1404.4641, 2014.
11. Feng, Minwei, Xiang, Bing, Glass, Michael, Wang, Lidan, and Zhou, Bowen. Applying deep learning to answer selection: A study and an open task. IEEE Automatic Speech Recognition and Understanding Workshop (ASRU), 2015.

Appendices