FinalProject

Collection of data sequences

The collection of data sequences that we chose is Harry Potter And The Chamber Of Secrets that was written by by J. K. Rowling. This is the second book in Harry Potter's series

2.

Description of data

This was actually not our first data that we tried to learn. We started this project working on a collection of speeches of Binyamin Netanyahu. We gathered and combined a lot of speeches, but when we tried to learn and train the model, we had difficulties to create reasonable sentences from it. We think the reason for that was because the collection of the speeches were no big enough. We then chose Harry potter topic because that data is very big and has variety in its content (number of words, number of unique words etc). We think that the model training can be much more efficent in this context.

The main challenge in dealing with this topic is the investigation of the words which are not "real" words. Harry Potter's books contain many phrases and words that describe magic, which can be challenging to learn. In addition, in the past we never studied this kind of sequences analysys so this is also chanllenging for us.

Data Analysis: 12867 sentences. 7605 unique words

3.

Preprocessing

comment: Inilize variables. Replace all the words which are not in our vocabulary with "Unknown_Token". Append Setence_Start and Sentence_End to the sentence - it is used becuase we want to "teach" the model which words open sentences and which one end them.

Leraning rate - regulzation parameter - mainly used to prevent overfitting. Nepoch - number of iterations in the training phaze. Hidden_Dim - the memory of the network - making it bigger allows us to learn complex patterns in the data

```
vocabulary_size = 6500

unknown_token = "UNKNOWN_TOKEN"

sentence_start_token = "SENTENCE_START"

sentence_end_token = "SENTENCE_END"

_HIDDEN_DIM = 80
```

```
_LEARNING_RATE = 0.005
_NEPOCH = 100
```

comment: Read the data to the memory and tokenize into sentences. We used NLTK Python library for the tokenizing. When we first tried to train the model, we saw that it predicts the word "SENTENCE_END" at the beginning of the sentence. After investigation We found out that the reason for that was because the text had a lot of empty lines, which has a bad influence on the predictions. Therfore, we removed all the empty lines from the text. (We did it outside the code)

```
with open('data/harrypotter.txt', 'rb') as f:

reader = f.readlines()

#### comment: Split into sentences

sentences = itertools.chain(*[nltk.sent_tokenize(x.decode('utf-8').lower()) for x
in reader])

#### comment: Append SENTENCE_START and SENTENCE_END
sentences = ["%s %s %s" % (sentence_start_token, x, sentence_end_token) for x in
sentences]
```

comment: Tokenize words in each sentence (for example - Please come here! ->> {please} {come} {here} {!}.

```
tokenized_sentences = [nltk.word_tokenize(sent) for sent in sentences]
```

comment: Count the word freqencies

```
word_freq = nltk.FreqDist(itertools.chain(*tokenized_sentences))
```

comment: Get the most common words (vocabulary size) and build index_to_word and word_to_index vectors. The input for the RNN are vectors, not Strings, so we create mapping between words and indices.

```
vocab = word_freq.most_common(vocabulary_size - 1)

index_to_word = [x[0] for x in vocab]
index_to_word.append(unknown_token)

word_to_index = dict([(w, i) for i, w in enumerate(index_to_word)])
```

comment: Replace all words not in our vocabulary with the unknown token

for i, sent in enumerate(tokenized_sentences):

```
tokenized_sentences[i] = [w if w in word_to_index else unknown_token for w in sent]
```

comment: Create the training data. X is the sentences, Y is the sentences shifted right by one position. In this structure the model knows which word comes after each word. for example - Y[3] is the next word for X[3] in the sentence.

```
X_train = np.asarray([[word_to_index[w] for w in sent[:-1]] for sent in
tokenized_sentences])

y_train = np.asarray([[word_to_index[w] for w in sent[1:]] for sent in
tokenized_sentences])
```

comment: Init the model with random values for U, S, V. This function is attached to the project code. We used Theano for building the model. Theano is Python library that let you to define, optimize and evaluate mathematical expression, especially ones with multi-dimensional arrays. Because RNN are easily expressed with multi-dimensioanl arrays, Theano is a great fit.

```
model = RNNTheano(vocabulary_size, hidden_dim=_HIDDEN_DIM)
```

4.

Training the model

comment: Training the model. We send the training sentences (X_train and Y_train) for theano. Moreover, we set the Epoch number - which is the number of iteration over all sentences, and also set the learning rate for the SGD.

After every 5 iteration, we calculate the loss and we check if it decreases. This part (calculating the loss) is not necessary, its just an indication for us. Training the model took us alot of time due to the high number of sentences in Harry Potter's book.

```
def train_with_sgd(model, X_train, y_train, learning_rate, nepoch,
evaluate_loss_after=5):

    # We keep track of the losses so we can plot them later
    losses = []
    num_examples_seen = 0
    for epoch in range(nepoch): #one epoch means one iteration over all training
examples
        # Optionally evaluate the loss
        if (epoch % evaluate_loss_after == 0):
            loss = model.calculate_loss(X_train, y_train)
            losses.append((num_examples_seen, loss))
            time = datetime.now().strftime('%Y-%m-%d-%H-%M-%S')
            print ("%s: Loss after num_examples_seen=%d epoch=%d: %f" % (time,
num_examples_seen, epoch, loss))
            # Adjust the learning rate if loss increases
            if (len(losses) > 1 and losses[-1][1] > losses[-2][1]):
                learning_rate = learning_rate * 0.5
```

```
                print ("Setting learning rate to %f" % learning_rate)
            sys.stdout.flush()
            #Save the parameters after the iteration
            save_model_parameters_theano("./data/rnn-theano-%d-%d-%s.npz" %
(model.hidden_dim, model.word_dim, time),
                                         model)
        # For each training example...
        for i in range(len(y_train)):
            # One SGD step
            model.sgd_step(X_train[i], y_train[i], learning_rate)
            num_examples_seen += 1
```

5.

Loading the model

comment: Loadin the model. This function gets the path to the model file, and loads the model parameters to the memory. after the loading we can use this model to predict sentences.

```
def load_model_parameters_theano(path, model):
    npzfile = np.load(path)
    U, V, W = npzfile["U"], npzfile["V"], npzfile["W"]
    model.hidden_dim = U.shape[0]
    model.word_dim = U.shape[1]
    model.U.set_value(U)
    model.V.set_value(V)
    model.W.set_value(W)
    print ("Loaded model parameters from %s. hidden_dim=%d word_dim=%d" % (path,
U.shape[0], U.shape[1]))
```

comment: Generating sentences from the model. we generate only sentences that is bigger then 7 words. Sentece is represented by the model as vector of numbers so we need to convert it to a string using the index_to_word dictionary.

```
def generate_sentence(model):
    # We start the sentence with the start token
    new_sentence = [word_to_index[sentence_start_token]]
    # Repeat until we get an end token
    while not new_sentence[-1] == word_to_index[sentence_end_token] or
len(new_sentence)<7:
        next_word_probs = model.forward_propagation(new_sentence)
        sampled_word = word_to_index[unknown_token]
        # We don't want to sample unknown words
        while sampled_word == word_to_index[unknown_token]:
            samples = np.random.multinomial(1, next_word_probs[-1])
            sampled_word = np.argmax(samples)
        new_sentence.append(sampled_word)

    sentence_str = [index_to_word[x] for x in new_sentence[1:-1]]
    return sentence_str
```

6.

Calculating Similarity

comment: Calculating the similarity between the real sentences and the predicted sentences. The measure that we used for the calculation is Cross-Entroty-Loss. for each word in a sentence, it measures how far our prediction was from the real word. For example, if we have a sentence - "We are eating dinner with some friends" and we want to predict the word which comes after "dinner". the model generate a vector (in the size of the vocabulary) with probabliltes for each word. let's say the the word "with" has a probabilty of 0.8 to be the next word. so the distance of the prediction is Yn*Log(On). Yn equals 1, On eqauls 0.8. We do this calculation for every word in the sentence and sum it up. finally, we divide it by the number of words in the sentence to get avarage loss for each word.

We calculated the loss for the whole database, and also calculated the loss for 50 sentences that we picked randomally (feel free to change the number of random sentences and run it again).

```
def calculateSimilarity():
    totalLoss =0
    for i in range(len(y_train)): #iterate over each sentence and calculate the
loss
        currSentence_Loss = model.calculate_loss_sentence(X_train[i],y_train[i])
        totalLoss = totalLoss+currSentence_Loss
        #print("Sentence number %d and loss for the sentence is %f." % (i,
currSentence_Loss))
    print ("Average loss for all sentences in the test: %f"
%(totalLoss/len(y_train)))

def calculateSimilarity_Random(amount):
    from random import randint
    totalLoss=0
    for i in range(amount): #pick #amount random sentences
        randomNum = randint(0,len(y_train)) #pick a random sentence from the text
        currSentence_Loss =
model.calculate_loss_sentence(X_train[randomNum],y_train[randomNum])
        sentence = getSentenceFromIndices(X_train[randomNum])
        print("Sentence Number %d:[ %s ] Loss: %f" % (randomNum, sentence,
currSentence_Loss))
        totalLoss=totalLoss+currSentence_Loss
    print("Average loss for the random sentences is: %f" % (totalLoss / amount))

def calculate_total_loss_sentence(self, X, Y):
    return self.ce_error(X, Y)

def calculate_loss_sentence(self,X,Y):
    num_words = len(Y)
    return self.calculate_total_loss_sentence(X,Y) / float(num_words)
```

comment: Here are the results of loading the model and calculating the loss over 50 sentences. Structure: Sentence Number, Sentence text, and Loss for the sentence. The last line is the avarage loss for all of the random sentences.

```
Sentence Number 11777:[  `` not the greatest sorcerer in the world , '' said harry
, breathing fast ] Loss: 3.744226
```

```
Sentence Number 127:[  husband ] Loss: 3.192912
Sentence Number 10090:[  fang bounded happily out of the house behind them , dashed
to ] Loss: 4.910388
Sentence Number 6009:[  `` homework - compose a poem about my defeat of the ] Loss:
6.344539
Sentence Number 10163:[  fang ! ] Loss: 2.253425
Sentence Number 9933:[  hands - ] Loss: 2.938197
Sentence Number 3345:[  hufflepuff boy harry knew by sight but had never spoken to
] Loss: 7.334905
Sentence Number 8463:[  `` it 's okay , hermione , '' said harry quickly ] Loss:
2.973303
Sentence Number 9992:[  harry and ron UNKNOWN_TOKEN ] Loss: 3.566368
Sentence Number 7013:[  too , read the sign with interest ] Loss: 5.727168
Sentence Number 10559:[  `` i have good news , '' she said , and the great hall ,
instead of ] Loss: 4.432502
Sentence Number 6117:[  arms out of her head ] Loss: 3.645669
Sentence Number 6955:[  seconds , harry straightened up , took aim , and
UNKNOWN_TOKEN it into the air ; ] Loss: 5.002339
Sentence Number 10030:[  `` i flatter myself i know a touch more about hagrid 's
arrest than ] Loss: 6.008347
Sentence Number 79:[  harry felt ] Loss: 1.778701
Sentence Number 6504:[  it burned harry 's mouth and throat as ] Loss: 5.146540
Sentence Number 7271:[  `` come on , '' said rods voice in his ear ] Loss: 3.624885
Sentence Number 7843:[  through ... ] Loss: 2.930481
Sentence Number 821:[  ] Loss: 1.253522
Sentence Number 4833:[  the UNKNOWN_TOKEN dropped with every ] Loss: 5.330466
Sentence Number 668:[  he jumped the last six steps , landing catlike on ] Loss:
6.179426
Sentence Number 2646:[  he moaned as the ceiling sagged , but suddenly ] Loss:
5.601741
Sentence Number 8497:[  asked harry , pointing to something gold sticking ] Loss:
5.371894
Sentence Number 2282:[  `` yeh should 've ignored him , arthur , '' said hagrid ,
almost lifting mr. ] Loss: 5.921703
Sentence Number 10162:[  `` it 's already ] Loss: 4.054174
Sentence Number 12739:[  `` you shall go now , '' he said fiercely , pointing down
at mr. malfoy ] Loss: 3.691131
Sentence Number 11275:[  `` you can go first , '' ron snarled ] Loss: 3.837430
Sentence Number 8066:[  `` hang on , '' said harry as ron and hermione reached for
their glasses ] Loss: 3.747056
Sentence Number 2889:[  i will be writing ] Loss: 4.065180
Sentence Number 6539:[  out ] Loss: 4.118778
Sentence Number 8512:[  ] Loss: 3.547355
Sentence Number 6462:[  poking out of the end of his robes was what looked like a
thick , ] Loss: 4.621703
Sentence Number 5322:[  had been led there by a bodiless voice no one but he could
hear ] Loss: 6.449992
Sentence Number 11607:[  it was very amusing ] Loss: 3.081833
Sentence Number 6496:[  as he swung himself onto the bed , his arm flapped
pointlessly ] Loss: 5.068804
Sentence Number 9772:[  croaked hagrid ] Loss: 3.708964
Sentence Number 9841:[  `` growled hagrid ] Loss: 3.545044
Sentence Number 287:[  staring absent-mindedly into the hedge - and the hedge was
staring back ] Loss: 5.198655
Sentence Number 2514:[  turrets of snowy cloud , in a car full of hot , bright
sunlight , with a ] Loss: 6.627636
Sentence Number 11994:[  almost knocking him out ] Loss: 4.915882
```

```
Sentence Number 496:[  almost at once , harry wished he had n't spoken ] Loss:
4.807133
Sentence Number 6340:[  `` fred , george , you heard harry -leave him ] Loss:
5.839961
Sentence Number 8208:[  as they hurried nearer ] Loss: 3.698467
Sentence Number 12024:[  sideways and fell , twitching , to the floor ] Loss:
5.438543
Sentence Number 12271:[  g UNKNOWN_TOKEN a p t e ] Loss: 6.611085
Sentence Number 6818:[  arm ? ] Loss: 2.578243
Sentence Number 3887:[  harry sank ] Loss: 4.518908
Sentence Number 11342:[  `` blimey , '' said ron weakly ] Loss: 3.389589
Sentence Number 5078:[  kill ] Loss: 3.163991
Sentence Number 97:[  homework done ] Loss: 5.116531

Average loss for the random sentences is: 4.413114
```

comment: We also calculated the loss over all the dataset using the functions mentioned above.

```
Average loss for all sentences in the text: 4.616136
```

7.

Conclusions

comment: Conclusions: Let's first print some sentences from our model:

```
she was aloud . SENTENCE_END at
he reached . SENTENCE_END through his arm .
bag . SENTENCE_END door got mention - ''
* unpleasantness shut SENTENCE_END .
we understand ? SENTENCE_END over it was an
* whassamatter $ defeat me .
the walls . SENTENCE_END have
`` ' too ! ''
`` it ? '' SENTENCE_END uncle his boasting
he 'd was a rule-breaker remaining blurred out again , ron considerable
but now them spells SENTENCE_END , potter ? ''
SENTENCE_END of the nearest room .
kept i 'm me .
and SENTENCE_END lord him , '' ron
*59* SENTENCE_END sister to dumbledore anymore .
. '' SENTENCE_END percy fearsome nervously for
said lockhart had nastily together .
harry potter . SENTENCE_END at his forehead .
he knew harry suddenly off his fist .
the monster . SENTENCE_END at school .
. SENTENCE_END SENTENCE_END kill to step
they had SENTENCE_END furious and hermione out over .
and said he had shouted .
`` did it . ''
- '' SENTENCE_END harry has to wake .
said aragog year . SENTENCE_END at quite wall .
. '' SENTENCE_END mr. SENTENCE_END
```

```
magnified . SENTENCE_END his powers .
`` then sounds smell it ... ...
no revive lord froze .
said lockhart , harry had shouted .
dormitory ? '' SENTENCE_END SENTENCE_END save darker .
the elf swung SENTENCE_END to him a
and then he saw into
```

comment: As we can see from the results, some of the sentences have meaningfull connection between the words. Furthermore, the model has learned punctuation. for example - it puts "." at the end of the sentence. Another thing that we can see from the results is that the model predicts too often the word "Sentence_End". We think that the reason for this is becuase "Sentence_End" appears in every sentences in the text (We add it to the end of each sentence in the preprocessing phase), which causes the word to have large probabilty as the next word prediction.

By looking at the avarage loss over all sentences - it is much better from a model that chooses the next word randomally, for example. However, It is not perfect. We trained the model over more then 24 hours. Maybe if we had a stronger computer and more time, we could better learn the text and produce even more concise sentences.