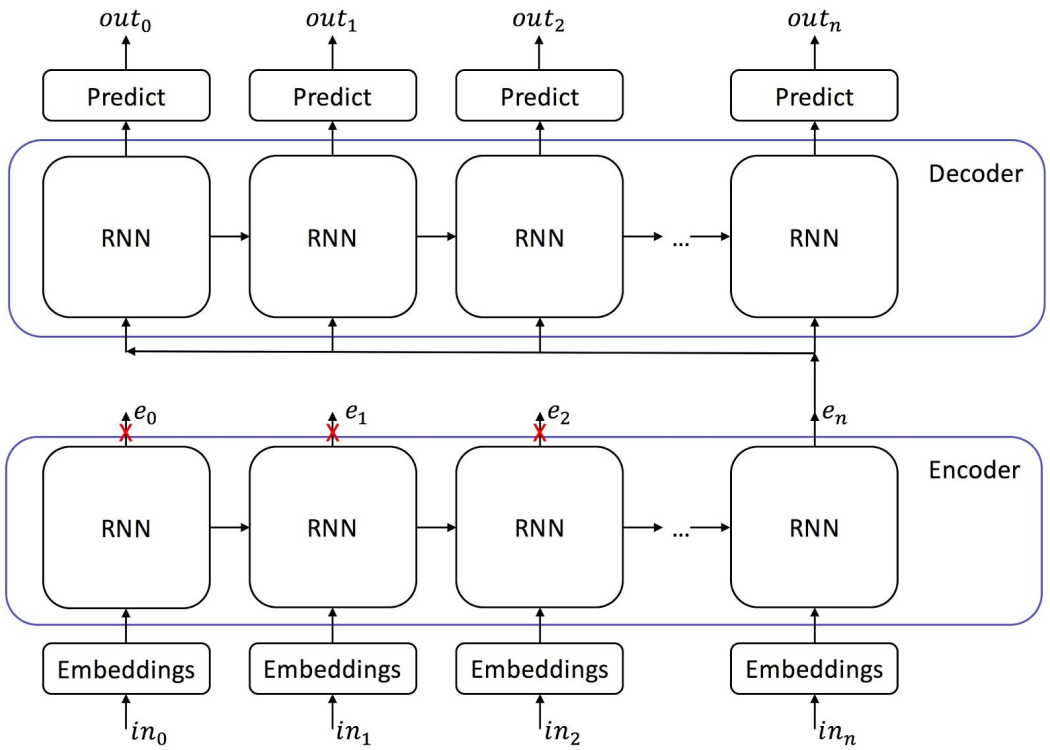
AI LAB 2

BUILDING AN ENCODER-DECODER BASED MODEL THAT CORRECTS TWEETS.

November 19, 2018

Introduction



Deep Neural Networks (DNNs) are powerful models that have achieved excellent performance on difficult learning tasks. Although DNNs work well whenever large labeled training sets are available, they cannot be used to map sequences to sequences. In this lab experiment, we have had a general end-to-end approach to sequence learning that makes minimal assumptions on the sequence structure. Our method uses a multilayered Long Short-Term Memory (LSTM) to map the input sequence to a vector of a fixed dimensionality(Using Word2Vec), and then another deep LSTM to decode the target sequence from the vector(Output in the form of One Hot Vector). Our main result is that on an Tweet correction task from the Consolidated.csv dataset, the corrections produced by the LSTM achieve a an accuracy of 66.9% on the entire test set.

An Encoder-Decoder model in context of recurrent neural networks (RNNs) are **sequence to sequence mapping models.** An RNN encoder-decoder takes a sequence as input and generates another sequence as output.

For example,in the following lab we took Tweets (which is sequence of words) as the input to the encoder and generated the corrected tweets (sequence of words) as the output of the decoder.

The encoder-decoder model for recurrent neural networks is an architecture for sequence-to-sequence prediction problems.

It is comprised of two sub-models, as its name suggests:

- **Encoder**: The encoder is responsible for stepping through the input time steps and encoding the entire sequence into a fixed length vector called a context vector.
- **Decoder**: The decoder is responsible for stepping through the output time steps while reading from the context vector.

PRE-PROCESSING THE DATASET

We had two different vocabularies to deal with,one was the input vocabulary which was formed from the Tweet column of the consolidated dataset and the other was the output vocabulary which was formed from the Corrected column of the consolidated dataset.

We used Keras itself for processing the summary(text preprocessing component of the library).

Processing:

INPUT VOCABULARY :

1. Tokenized the tweets after explicitly converting each to a lower case after removing the invalid inputs.
2. Padding the tweets to ensure all are of same length.
    a. All the tweets are padded to the maximum length of the tweets in the vocab.
3. Converted Every tokenized tweet of both train set to a vector using Gensim's Word2Vec.
    a. Gensim only requires that the input must provide sentences *sequentially*, when iterated over.
    b. Produce word vectors with deep learning via word2vec's "skip-gram and CBOW models", using either hierarchical softmax or negative sampling.
    c. Hyperparameters set as the arguments to the Word2Vec are min_count = 1, size = 100, workers = 4, sample = 1e-3.
4. Preparing the text so that it can be given to the Neural Network model using Tokenizer.

OUTPUT VOCABULARY :

1. Selecting the top 3500 words that occurred in the corrected column from the consolidated dataset.
2. Including “Unknown” in the vocabulary to classify all the words that the network has not seen before to put it in the Unknown bucket.
3. Convert each word of the output vocabulary to “One Hot Vector”,which has a dimension of 3501(3500 + 1).

THE MODEL

configure problem

Number of input features = 100 (Word2Vec tokenized vectors)

n_timesteps_in = max_length_tweet (46)

n_timesteps_out = max_length_tweet_correct (46)

n_op_features = 3501 (One hot vectors)

In [61]:
```
model = Sequential()
model.add(LSTM(150, input_shape=(n_timesteps_in, n_inp_features)))
model.add(RepeatVector(n_timesteps_in))
model.add(LSTM(150, return_sequences=True))
model.add(TimeDistributed(Dense(n_op_features, activation='softmax')))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc'])
```

In [62]:
```
model.fit(X_train_new, y_train_new, epochs=2)
```

```
In [50]: for i in range(len(tokenized_tweets)):
             while(len(tokenized_tweets[i]) < max_length_tweet):
                 tokenized_tweets[i] += [""]

In [51]: gensim_model = gensim.models.Word2Vec(tokenized_tweets, min_count = 1, size = 100, workers = 4, sample = 1e-3)

In [52]: print(gensim_model)

         Word2Vec(vocab=6520, size=100, alpha=0.025)

In [53]: X_train_new = []
         for i in tokenized_tweets:
             x = []
             for j in i:
                 x.append(list(gensim_model[j]))
             X_train_new.append(x)

In [54]: len(X_train_new[1])

Out[54]: 46
```

This is the code snippet which shows how we have made the input vocabulary using the word2vec of gensim model.

We have configure the encoder and decoder with the same number of units, in this case 150. We have used the efficient Adam implementation of gradient descent and optimized the categorical cross entropy loss function, given that the problem is technically a multi-class classification problem.

ACCURACY

With a sample size of 5500, our model could successfully correct a given tweet with an accuracy of 68.61%.

NOTE: We considered 5500 samples instead of 10500 as was mentioned in the guide sheet as our systems weren't that powerful to train that huge so many samples.

THE SQUAD

1. Shahid Ikram
2. Sanketh Rangreji
3. Sumanth S. Rao
4. Nishant Sondhi