Sequence to Sequence: Words to Phonemes using LSTM & Autoencoders

This work was made in an effort to meet the final project requirement of CS6320.F18 under Dr. Dan Moldovan at the University of Texas at Dallas

Sree Teja Simha Gemaraju

Abstract— **This paper presents a report on implementation of Deep learning technique for grapheme to phonemes conversion problem. The approach involves encoder and decoder Recurrent Neural Networks that build intuition over the grapheme and use it to build the phoneme. This approach is inspired by Sequence to Sequence Learning with Neural Networks[1] paper.**

I. INTRODUCTION

Grapheme is the smallest unit of a writing system[2]. For English language, it is the alphanumericals and punctuation. Individually, graphemes and individual units contribute little to the sense or pronunciation but arrangements of them do a lot more. Especially, arrangements of graphemes have co-responding matching arrangements of phonemes - the unit of sound and pronunciation system. This matching is often times regular or rather intuitive. The rules of pronunciation are not clear and are always subjected to contextual intricacies. But the inherent truth is that, we can model a system that find matching phonemes and graphemes. It has been tried and tested against several techniques involving heuristics, stochastic methods, machine learning etc... In this approach we apply the idea of sequence to sequence learning using long short term memory recurrent neural network in combination with Autoencoder.

II. SEQUENCE TO SEQUENCE

Recurrent neural networks are very efficient in modeling sequential patterns. Apart from prediction, RNNs also produce an internal state which is feed back into the RNN along with next input pattern. An iteration of RNN consumes input pattern and state information from previous iteration to build new prediction and new state. Thus the sequential patterns are learned while respecting their sequential relations.

LSTM - Long Short Term Memory is a variant of RNNs. This model allows for precise control over how much the past state and the current state should influence the next prediction. In RNNs where the earliest pattern of the input can influence the latest though it is not directly related. LSTM can learn sub patterns within the input and generate relevant sub-patterns in the end result. It is this property that we are going to take advantage of.

III. SEQUENCE TO SEQUENCE LEARNING WITH ONLY CNNS

RNNs and LSTMs produce results with same dimensions. Graphemes and phonemes are rarely of same dimension. But they still a problem of sequences. One approach would to convert/represent both input and output with same dimension. This could be done by vectorizing them with zero padding. Once they are of equal dimension, it is pretty much an image to image learning. A CNN-Autoencoder-decoder can be used to fit the model. This approach didn't yield any respectably good results as the solution was not reflective of problem's intrinsic structure. The pattern to pattern adjacency relations are violated. Often times the model wouldn't converge. This approach was abandoned immediately as the outcome was just as predicted - non-congruent.

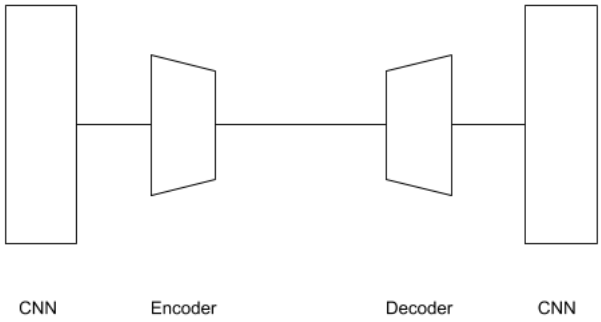

Fig. 1. "A simple CNN-autoencoder"

IV. LSTM-AUTOENCODER[3] MODEL

The approach is to run an LSTM through out the subpatterns or graphemes of a sample. This one iteration generates a final state - a deep intuition of the grapheme. Using the final state of the LSTM, we can train an autoencoder to generate initial state for another decoder LSTM that would generate phoneme patterns. The decoder LSTM would start with the initial state generated by the encoder and takes a dummy starting input and predicts first phoneme that matches the first phoneme of the input grapheme to the encoder. The result generated in this iteration is feed back to the decoder lstm along with the new state generated along with first phoneme. We repeat this process until the complete phoneme pattern is generated. In order to detect completion of phoneme pattern generation, we need a termination marker - a dummy phoneme marker.

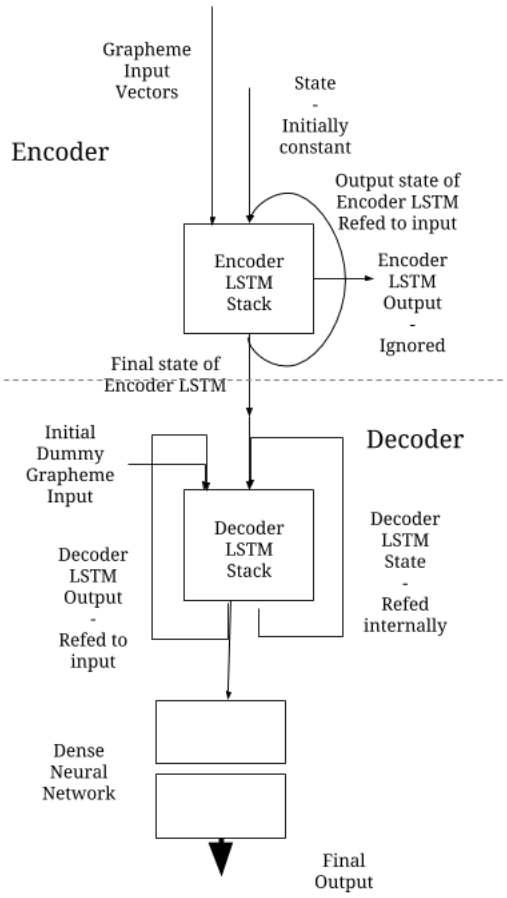

Fig. 2. "A simple LSTM-autoencoder"

This model is inspired by the LSTM autoencoder tutorial on keras blog[4].

V. IMPLEMENTATION

The presented model is implemented using tensorflow. The training model is built differently from the prediction model. The training and prediction models has two parts - encoder and decoder. The only difference between two models is that, the training model has access to prediction data and can be written without having to declare things in a round about way to plug the prediction of one step back into model.

The training model begins with an input layer that feeds directly into the encoding LSTM stack(2 LSTMS) with 128 units of internal representation. The outputs of encoding LSTM are ignored but the state/representation - Hypothesis(H) and Memory(C) are collected. This H and C are treated as deep intuition or the encoder's high dimensional internal representation of grapheme information. This internal representation is then fed into decoding LSTM stack(1 LSTM) with 128 units of internal state representation. Unlike encoder LSTM, we ignore the internal representation and take the output and feed it to Dense Neural Network layer that shapes the results to the phoneme vector representation. The expectation is that, this phoneme vector will be used to find the argmax phoneme.

For training process, we feed the characters of words as one-hot vectors in sequence. The phonemes are padded with one start symbol and one stop symbol - both as extra entries in list of possible phonemes. These start stop markers will be used to train the model to guess if the sequence pattern has been completely translated. The initial input to the decoder LSTM is a start marker and the output is expected to be the first phoneme. The next iterations will use the output of previous iteration and expect the subsequent. This model is trained using RMSProp optimizer with a variation of learning rate, batch size and epochs as given in the tableII. The choice of these hyper parameters were very subjective to the model's configuration.

Layer (type)	Output Shape	Param #	Connected to
einput$_layer(InputLayer)$	(_,_, 26)	0	-
lstm1$_layer(LSTM)$	(_, _, 128)	79360	einput$_layer$
lstm2$_layer(LSTM)$	(_, _, 128)	131584	lstm1$_layer$
lstm3$_layer(LSTM)$	(_, _, 128)	87040	dinput$_layer$
dense1(Dense)	(_, _, 128)	16512	lstm3$_layer$
dropout$_1(Dropout)$	(_, _, 128)	0	dense1
dense2(Dense)	(_, _, 41)	5289	dropout$_1$

Total params	319,785
Trainable params	319,785
Non-trainable params	0

TABLE I
TRAINING MODEL SUMMARY

training_step	lr	batchsize	epochs
1	0.05	128	30
2	0.01	64	20
3	0.005	64	20
4	0.001	64	20

TABLE II
TRAINING STEPS

VI. OBSERVATION

The model was trained with even indices of data pairs and tested over all. With such limited configuration and training steps, the accuracy was a measly 0.43 at it's best. But interestingly, most errors made were homonyms and especially easily swappable phoneme pairs such as /D/ - /T/ (stepped'ed' - stepp't'), /Z/-/S/, /K/ - /C/ etc... Lost of mistakes were made at the beginning of the words which often got carried over further. The logic behind such errors being that the intuition was modeled with sequence in just one direction - left to right. The relations that apply right to left were affected by this. A fix for this would be a bidirectional model which is being looked into at the time of this report.

REFERENCES

[1] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[2] Charles A Perfetti and Ying Liu. Orthography to phonology and meaning: Comparisons across and within writing systems. *Reading and Writing*, 18(3):193–210, 2005.

[3] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[4] A ten-minute introduction to sequence-to-sequence learning in keras. https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html. Accessed: 2018-12-7.