

Methods in AI Text Generation

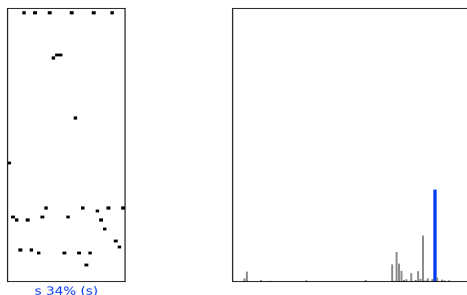
CSCI 3202 Practicum

Royce Schultz

May 2019

1 Abstract

This project explores methods in AI text generation using TensorFlow. This implementation will treat writing text as an iterative classification problem. A densely connected neural network attempts to classify a fixed length slice of a source text according to the character following the slice. Performing this operation repetitively can be used to generate text, typing one character at a time.























This figure shows the string representation on the left and the model correctly predicting the next letter in the character space on the right.

2 The Implementation

2.1 Encoding

The network takes a fixed length string as input, but letters are not an ordered set so floating point operations between them make no sense. The data must be encoded some other way.

In this implementation, strings are stored as 2d Boolean matrices with symbol on one axis and position on the other. So each column will have exactly 1 True and the rest False. The word "Hello" is stored like this,

h					
e					
l					
o					

Although, the set of letters is much larger when considering an entire text. The vocabulary may be easily accessed with the following structures,

2.1.1 Text Processing

```

text = open('Shake.txt', 'rb').read().decode(encoding='utf-8')
vocab = sorted(set(text)) #set of symbols
char2idx = {u:i for i, u in enumerate(vocab)} #dict of indices
idx2char = np.array(vocab) #array of chars
text_as_int = np.array([char2idx[c] for c in text])

def str2Img(s, vocabSize):
    x = np.zeros((vocabSize, len(s))) #initialize matrix
    for j in range(len(s)):
        c = char2idx[s[j]] #find index
        x[c, j] = 1 #fill in bubble
    return x

```

Caveats: The structure here is "sparse" or mostly composed of zeros except at a few indices. Passing around large variables like this is inefficient, but TensorFlow has tools to help that were not used in this implementation. Specifically tensor slices and embedding layers. See "Other Methods" for more information.

2.2 Training Data Generation

Training data is generated from a large source text. This implementation uses the Complete Works of William Shakespeare. Small slices of text are converted to "text images" and the following character is recorded as its label.

The following method generates n training points evenly spaced throughout the source text. imgWidth is the length of the slice of text.

```

genNTxtImg(n, text_as_Int, vocabSize, imgWidth)
    returns: datapoints, dataSolutions

```

2.3 Network Configuration

The network flattens the 2d "text images" and passes it to densely connected hidden layers.

```
keras.layers.Flatten(inputshape=(len(vocab), imgWidth))
keras.layers.Dense(128, activation=tf.nn.relu),
keras.layers.Dense(len(vocab), activation=tf.nn.softmax)
```

The imgWidth parameter is the size of size of the input string. If this parameter is too small, the network will not see the larger structure of words and sentences, but if it is too large, computational time will suffer due to the extra free parameters and returns from the larger scope will diminish after a point. Through experimentation, imgWidth = 32 seemed adequate, however this choice is entirely arbitrary.

2.4 Text Generation

Once the model has been trained, the following method can be used to generate text,

```
def generateText(seed, imgWidth, length):
    utilizes: model.predict(str2Img(seed))
    returns: genTxt
```

If the seed text is not large enough, it will be padded with spaces.

3 Results

3.1 Conclusions

Provided enough training data, this model is able to produce text that is stylistically similar to the source text, however it is unable to produce anything syntactically meaningful.

3.2 Over fit vs General Models

Each time the model trains on a data set is called an **epoch**. If a model trains on the same data set for many epochs, it can "memorize" the results without learning the characteristics of the solutions that will help classify more general inputs. This is called **over fitting**. Conversely, I observed that if the model trains for too few epochs on each data set, the model cannot learn fine details and the solutions tend toward the "average best choice" which often leads to lots of bland repetition of common letters and "the", so to preserve some personality, the model will be trained for 3 epochs on a training set of 50,000 data points before new data points are generated. This will be called a **training cycle**.

3.3 Training Progress

1 Training Cycle: The network begins by learning the most common words,

*"...and that the hith me tore for he with the he the st the stow the he me shes cand the
parsen for the..."*

4 Training Cycles: With more time, the network’s vernacular begins to take an Old English tone. It has also begun to emulate the paragraph style. The new lines and capital letters were written by the AI.

*“...she friends him and my and her shall be the exter thee dound on and to heart and the
made*

*The mad master fored where did thee made
And to here hate with the whal spear and him shere the forest.
When that the word, I the do the rom...”*

10 Training Cycles: The AI has learned stage direction! ”KING.” is the style used in the source text to distinguish the king’s line in the play. Additionally, The spelling has become distinctly Old English.

*That the men in the prack to the well.
KING. I have be the gent the death be thee
and a whal the terven the the menthe mend the bucher
The shall be the stranger and the well.
And mall the sock the whild and*

4 Running my code

Simply run ”tensorText.py” to observe the networks training process. It will repetitively generate new training data, train the network and print 250 characters of generated text. tensorText.py points to ”Shake.txt”, the source file for the network’s training data.

5 Other Methods

Included in the tutorials on TensorFlow.org is an alternative solution to this same problem. This implementation utilizes many special data structures and functions included in TensorFlow that are specifically designed for sequenced data and are more efficient than my implemented methods.

5.1 Data Structures and Functions

The tutorial utilizes slicing, chunking, batching, and mapping to access the large text data efficiently. With this structure, the value of the source data need only be stored in 1 location and access to the data is handled entirely by structure functions.

5.2 Layer Structure

This implementation uses a more clever layer structure in attempt to emulate memory.

5.2.1 Embedding Layer

An embedding layer is a sparsely connected layer that takes a sequence of discrete letters as input and maps the sequences to a continuous vector space. The sparse connections allow for faster processing of larger strings.

5.2.2 GRU Layer

Gated Recurrent Units emulate memory by utilizing 2 gates: an update gate and a reset gate. Suppose the reset gate is triggered on a "." then the Network could have a sense of sentence structure to write with a more natural pace.

5.3 Their Results

It should be noted that it took this network significantly longer to train than my own implementation as it can train on the entire source text using slices rather than a subset using my text image method.

*MERENIUS: Here with thy it to-with fights, friends, to bust streyige in. Grfey here
remarul, At you she winow: if, my the she?
Messex the farthrous fromsty.
SAcepen: Will the horse wouth for my ploay.
QUEEN GLUY: Why, sarm shall I chave and ruck?
ANw Mustivery: Marter that against cuuse and by country.*

This implementation seems to follow punctuation conventions and stage direction more closely, but the generated text still makes no sense syntactically and the spelling is certainly no better than my implementation.

6 References

Train your first neural network: basic classification. Retrieved from
https://www.tensorflow.org/tutorials/keras/basic_classification

Text generation using a RNN with eager execution. Retrieved from
https://www.tensorflow.org/tutorials/sequences/text_generation

Understanding GRU networks. Retrieved from
<https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be>

A Beginner's Guide to Word2Vec and Neural Word Embeddings. Retrieved from
<https://skymind.ai/wiki/word2vec>

Stackoverflow. <https://stackoverflow.com/>