Project 4 - Recurrent Neural Networks with Tensorflow and Keras

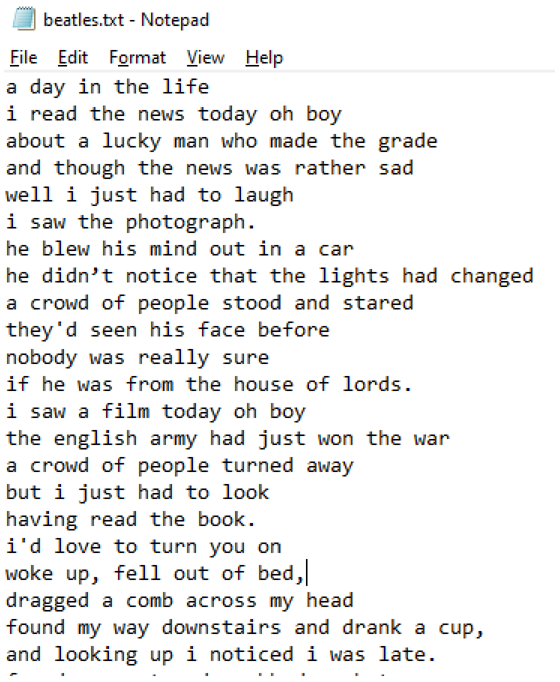
Anna-Maria Nau (anau) and Christoph Metzner (cmetzner)
COSC 525 - Deep Learning
University of Tennessee, Knoxville

April 2020

1 Introduction

The aim of this project is to build character based recurrent neural networks (RNN's) to write Beatles songs using the Tensorflow and Keras library for Python 3. The problem is framed as a many-to-one task in which given a certain number of characters the goal is to predict the next character. The task was executed for two RNN architectures: a) simple RNN and b) a LSTM (Long Short-Term Memory), with the hyper-parameters a) size of hidden state, b) window size, and c) strides.
The dataset includes 246 Beatles songs, concatenated to one long sequence, and stored in a text file.

The lyrics of the songs were taken from *http://beatlesnumber9.com/lyrics.html* (Figure 1)).



```
a day in the life
i read the news today oh boy
about a lucky man who made the grade
and though the news was rather sad
well i just had to laugh
i saw the photograph.
he blew his mind out in a car
he didn’t notice that the lights had changed
a crowd of people stood and stared
they'd seen his face before
nobody was really sure
if he was from the house of lords.
i saw a film today oh boy
the english army had just won the war
a crowd of people turned away
but i just had to look
having read the book.
i'd love to turn you on
woke up, fell out of bed,
dragged a comb across my head
found my way downstairs and drank a cup,
and looking up i noticed i was late.
```

Figure 1: Example excerpt from the used dataset: text file containing 246 Beatles songs.

2 RNN-Architectures and their Hyper-Parameters

The following chapter presents the 10 different model architectures. Two RNN architectures were built, the simple RNN and the LSTM (Long Short-Term Memory). The training process was the same for all models. The learning rate was chosen with a little trial and error to be 0.001, the activation function in the final output layer (fully connected layer) was the *"softmax"* with *"ADAM"* as the optimizer, and the number of epochs was 50. Dropout (p=0.2) was included to provide regularization in the multilayer models with the goal to avoid over-fitting and improve model performance. Table 1 gives an overview of the 10 models including their specific hyper-parameter settings such as hidden state, window size, and stride.

Table 1: RNN-model architectures and their hyper-parameters.

Model	Hidden State	Window Size	Stride
One-layer RNN	75	4	2
	75	10	5
	100	4	2
	100	10	5
One-layer LSTM	75	4	2
	75	10	5
	100	4	2
	100	10	5
Mutli-layer RNN	100, 100	10	5
Multi-layer LSTM	100, 100	10	5

3 Results

The 10 models were compared in terms of the epoch-vs-loss plots, training loss at epoch 50, and the generation of the next 20 characters when initialized with text from the training data of length window size.

The key findings of this project are that model selection and increased number of hidden states improves model performance. As Table 2 indicates, LSTM based models outperform RNN based models, as well as models with 100 hidden states usually have lower loss at epoch 50 than their respective counterpart with 75 hidden states. For this study, the best model was the one-layer LSTM utilizing 100 hidden states with window size 10 and stride 5 achieving a loss of 0.39 at epoch 50. In addition, one-layer models seem to train faster than multi-layer models. The training time is sped up using a larger value for the window size and stride (Table 2).

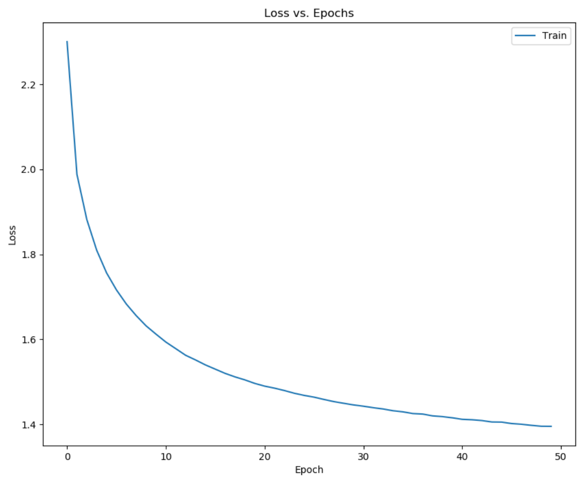
The one-layer RNN and one-layer LSTM, both with 100 hidden states, window size of 10, and stride of 5, generated correct text for the initial text input *"sergeant p"* with *"sergeant pepper's lonely heart"*, which is part of the actual lyrics from The Beatles song *Sgt. Pepper's Lonely Hearts Club Band.* The one-layer RNN and LSTM with hidden state of 75 for a window size of 4 and a stride of 2 were able to correctly predict the next word after the initial text input *"good"* with *"good like"* and *"good day"*. However, this model was not able to predict the remaining characters correctly.

Using multi-layer networks did not improve the model performance for the respective parameter settings. A limitation of this work are the missing exploration of the tipping point for increasing the number of hidden states and the training loss at larger epochs.

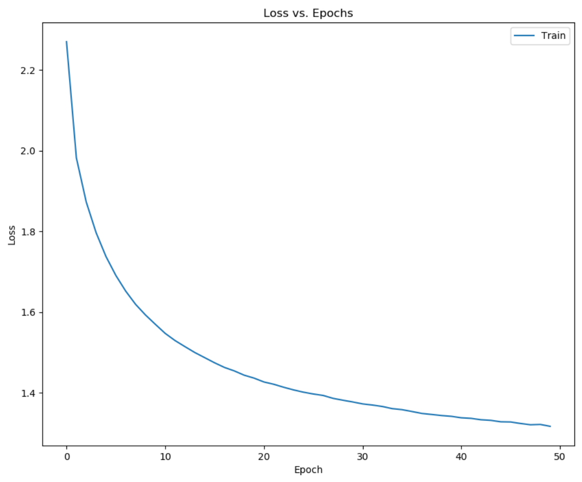
Table 2: Model performance.

Models with window size 4 and stride 2			
Model	Hidden State	Training Loss at epoch=50	Text generation with initial text: "good"
One-layer RNN	75	1.39	good like you see it be
	100	1.32	good do you say the way
One-layer LSTM	75	1.15	good day i love you know
	100	1.09	good love you know the s
Models with window size 10 and stride 5			
Model	Hidden State	Training Loss at epoch=50	Text generation with initial text: "sergeant p"
One-layer RNN	75	1.35	sergeant pe mo look don't be l
	100	1.18	sergeant pepper's lonely heart
One-layer LSTM	75	0.82	sergeant peppare the rouly fru
	100	0.39	sergeant pepper's lonely heart
Mutli-layer RNN	100, 100	1.62	sergeant pepper's lonely wanna
Multi-layer LSTM	100, 100	0.80	sergeant please me down the lo

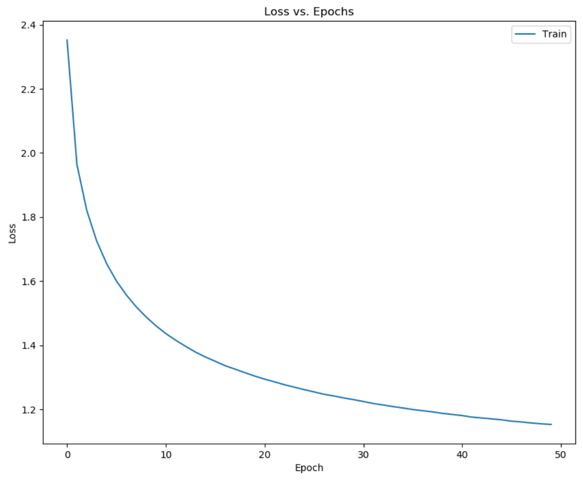
Figures 2 - 4) show a faster training for models using window size 10 and stride 5, as well for LSTM-based models. In general, LSTM-based models have a reduced loss of 1.3 at epoch 20 compared to the loss of the RNN-based models with 1.5. Figure 4) also indicates a less smooth training experience for multi-layer-based models.



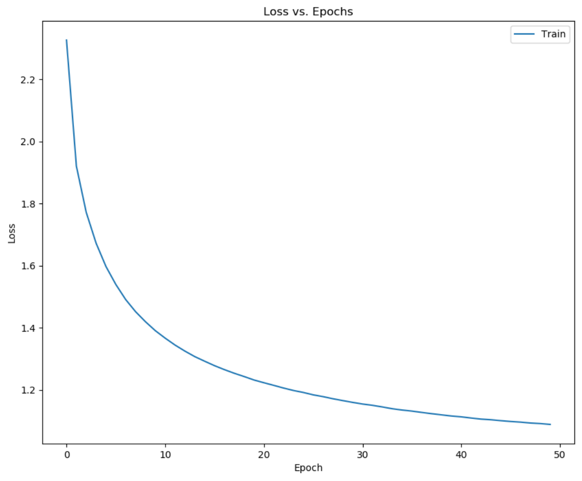
(a) One-layer RNN (75)



(b) One-layer RNN (100)

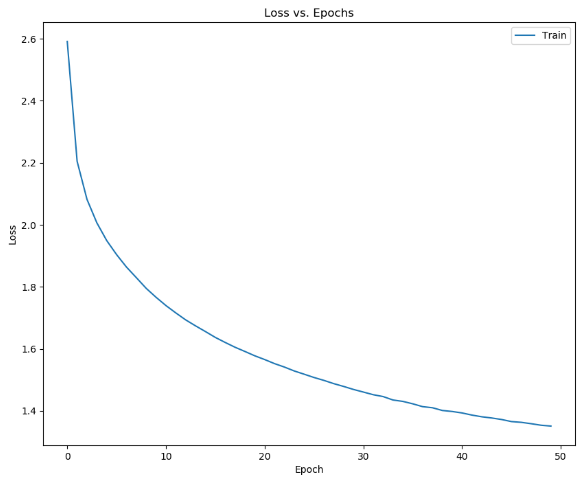


(c) One-layer LSTM (75)

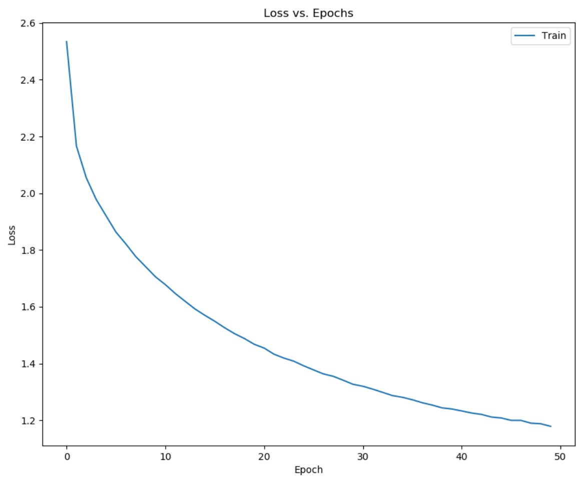


(d) One-layer LSTM (100)

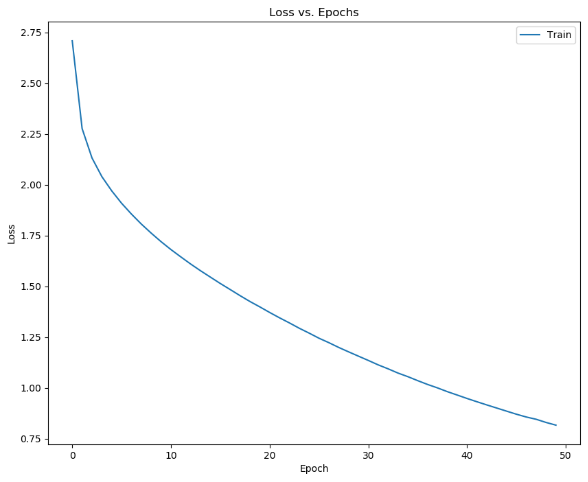
Figure 2: Epoch-vs-loss performance plots with parameters window size = 4 and stride = 2 for a) one-layer RNN (hidden state size = 75), b) one-layer RNN (100), c) one-layer LSTM (75), and d) one-layer LSTM (100).



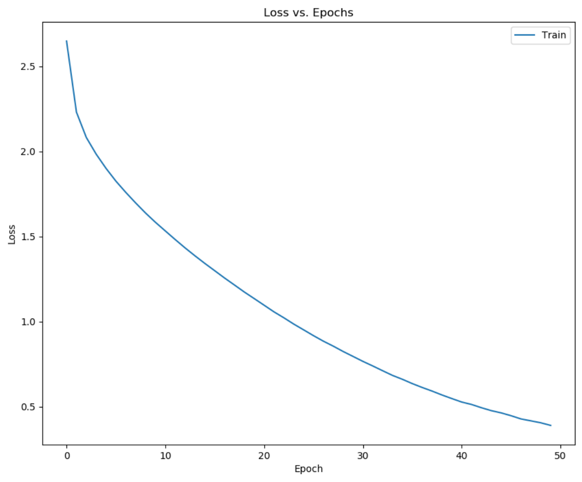
(a) One-layer RNN (75)



(b) One-layer RNN (100)

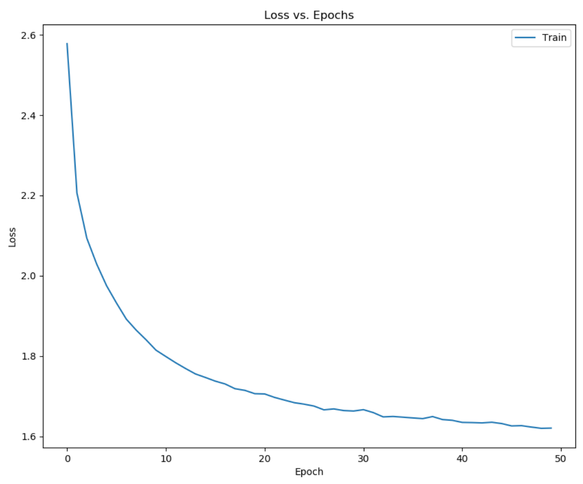


(c) One-layer LSTM (75)

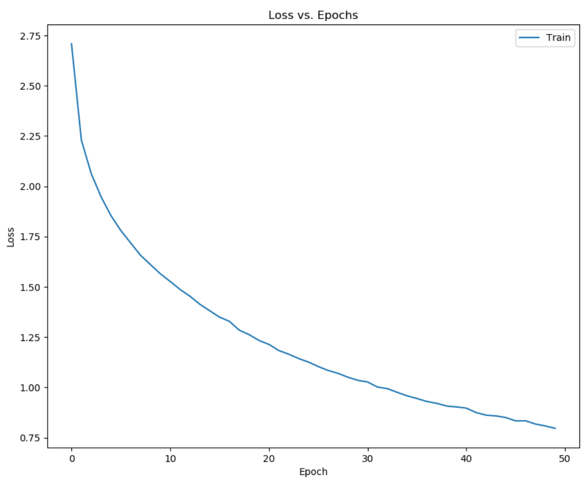


(d) One-layer LSTM (100)

Figure 3: Epoch-vs-loss performance plots with parameters window size = 10 and stride = 5 for a) one-layer RNN (hidden state size = 75), b) one-layer RNN (100), c) one-layer LSTM (75), and d) one-layer LSTM (100).



(a) Multi-layer RNN (100, 100)



(b) Multi-layer LSTM (100, 100)

Figure 4: Epoch-vs-loss performance plots with parameters window size = 10 and stride = 5 for a) multi-layer RNN (hidden state size = 100, 100) and b) multi-layer LSTM (100, 100).

4 Conclusion

Recurrent neural network architectures can be used to develop character based generation engines. LSTM-based models outperform common RNN-based models, as well as an increased number of hidden states has a positive effect on the model performance. Furthermore, the training process is faster for LSTM-based models and increased parameter values for the window size and stride. Future work could include a more comprehensive analysis of the number of hidden states, an analysis of the loss and text generation (i.e., prediction) by training with a higher number of epochs.

5 Running the Code

The `main` method takes the following command line variables: a) text file name: `beatles.txt`, b) `model type`, c) `size of hidden state`, d) `window size`, and e) `strides`. For example,

```
python3 dl_525_proj4.py beatles.txt "rnn" 75 4 2
```

Will build an RNN with a hidden state of size 75, a window size of 4 and a stride of 2. The following list provides the command line variables to run all 10 models:

- `python3 dl_525_proj4.py beatles.txt "rnn" 75 4 2`
- `python3 dl_525_proj4.py beatles.txt "rnn" 75 10 5`
- `python3 dl_525_proj4.py beatles.txt "rnn" 100 4 2`
- `python3 dl_525_proj4.py beatles.txt "rnn" 100 10 5`

- `python3 dl_525_proj4.py beatles.txt "lstm" 75 4 2`
- `python3 dl_525_proj4.py beatles.txt "lstm" 75 10 5`
- `python3 dl_525_proj4.py beatles.txt "lstm" 100 4 2`
- `python3 dl_525_proj4.py beatles.txt "lstm" 100 10 5`
- `python3 dl_525_proj4.py beatles.txt "rnn_multi" 100 10 5`
- `python3 dl_525_proj4.py beatles.txt "lstm_multi" 100 10 5`