Recurrent Neural Networks

Lucas Tindall
UCSD
ltindall@ucsd.edu

Abstract

Recurrent Neural Networks (RNNs) specialize in processing sequences of information. Unlike vanilla Neural Networks, RNNs use temporal information to process of data. This allows RNNs to have a memory of the data they have already analyzed. By having a neural architecture which contains directed cycles, the network outputs are dependent on previous outputs. With these abilities RNNs are particularly good at solving natural language processing problems such as language translation and recognition.

1. Introduction

In this paper I use RNNs to learn two different datasets and then produce outputs that are similar to the datasets. I explore the effects of varying hyperparameters and network architectures on the ability of the network to accurately learn the datasets and produce predictions.

The magic of RNNs is in their ability to maintain memory by feeding the output at each iteration as an input to the next iteration. This allows RNNs to look at data as long sequences rather than independent data points. In order to examine long sequences an RNN unfolds itself into a network in which each layer represents a different time step. To minimize the loss at each iteration, the network applies gradient descent using backpropagation through time. This allows the network to learn by iteratively updating the network weights.

In addition to traditional RNNs, this paper also explores the long short-term memory architecture (LSTM). The LSTM architecture does a better job at learning a dataset that has long time gaps between important data points. These networks are able to learn dependencies that are spread far apart by solving the vanishing gradient problem. The LSTM architecture has specials cells dedicated to maintaining memory over long periods of time.

2. Methods

For this paper I used an RNN implementation [4] written by Justin Johnson. This network utilized a cross entropy loss function. His implementation was based off of a similar RNN implementation [5] by Andrej Karpathy. For preliminary testing I used a minimal RNN [6] implementation also authored by Karpathy. All of the code to produce the RNN loss graphs can be found in a GitHub repository [3].

2.1. Character RNN

To begin the study into character predicting RNNs, I trained a network on a collected set of English literature by William Shakespeare. The dataset was simply a concatenation of passages from various Shakespeare texts. The API for training the RNN accepted various hyperparameters to change the exact architecture and behavior of the network. I experimented with 1 and 2 layer networks, vanilla RNN versus LSTM, various sequence lengths, and various hidden layer sizes.

2.2. Music Composition

The second dataset contained songs from the Nottingham Music Database [2]. Each song in the dataset was formatted in ABC notation, allowing the network to learn a character based representation of the songs. The boundary of each song was denoted by a “<start>” and “<end>” token. In order to have the network process these tokens as single characters, I converted each “<start>” token to just “<” and each “<end>” token to “>”.

3. Experiment Results

3.1. Character RNN

To start the experiment I used Karpathy’s minimal RNN [6], with 1 layer, 100 hidden units and a sequence length of 25. This relatively simple network converged relatively quickly but was only able to learn basic syntax and character names. The grammar and spelling were far from correct as seen in some selected outputs:

JOLA: If my nome bele the cuntlinot thy tay thee

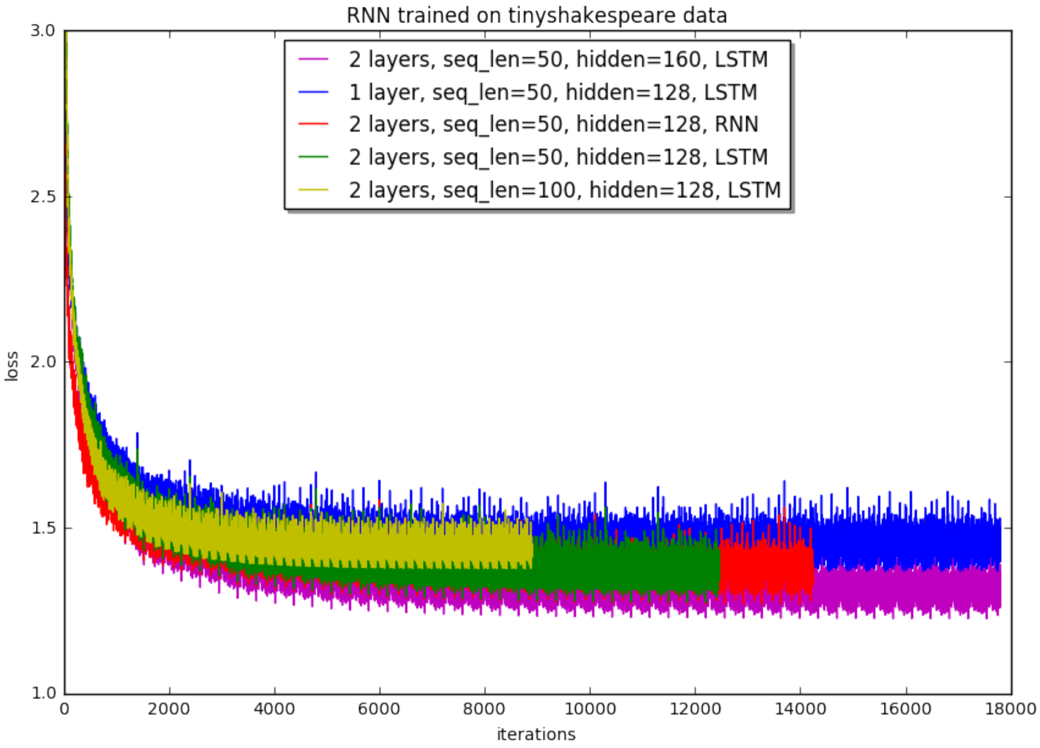

Figure 1. Loss with respect to iterations for various networks

> whese hens a tart pive men thou love anpinn.
>
> DUKE VINCENTIO: Buny. with as n
>
> LUCIO: My dired silling. Why crier it thy toinger

Seeing these results I decided to upgrade to a more sophisticated RNN implementation to try and produce more realistic outputs. Switching to the Torch based RNN [4], I started with a single layer RNN with 128 hidden units, a sequence length of 50, and a LSTM model. The second network had 2 layers, 128 hidden units in each layer, a sequence length of 50, and it used a regular RNN model. The third network had 2 layers, 128 hidden units in each layer, a sequence length of 50, and it used a LSTM model. The fourth network had 2 layers, 128 hidden units in each layer, a sequence length of 100, and it used a LSTM model. The fifth and final network had 2 layers, 160 hidden units in each layer, a sequence length of 50, and it used a LSTM model.

The loss with respect to iterations for all of the networks can be found in figure 1. The 1 layer network performed worst while the 2 layer network with a greater number of hidden units performed better than all the rest. Compared to the minimal RNN, the generated outputs from this implementation contained better spelling and grammar. Increasing the number of layers and hidden units had the greatest effect on improving the loss. Switching from a traditional RNN to LSTM had little effect while increasing the sequence length too high seemed to have a negative effect on learning.

From the best performing network, I generated some outputs with different initial sequences:

> ROMEER: Kiss Juliet, since about her hand worn provess Shall centain such a never but in so longer. Prebe with thy spilerspore, mighty discondron to his angroan, How wills thus.
>
> KING EDWARD IV: Name, I brother's lived that? O, God, do Yor. Is defeffect; I have sensot; my body the rid The woman tife? give my knaches hath handly, If they easy answer'd upon who! Hereford, And friend from Clarence, either my brate, I shall will upon To will in't.
>
> CORANUS: My lord, thou hast say, so. He thou hast by the state: Then that think me; gaunt shall bleeds fortates, At it.

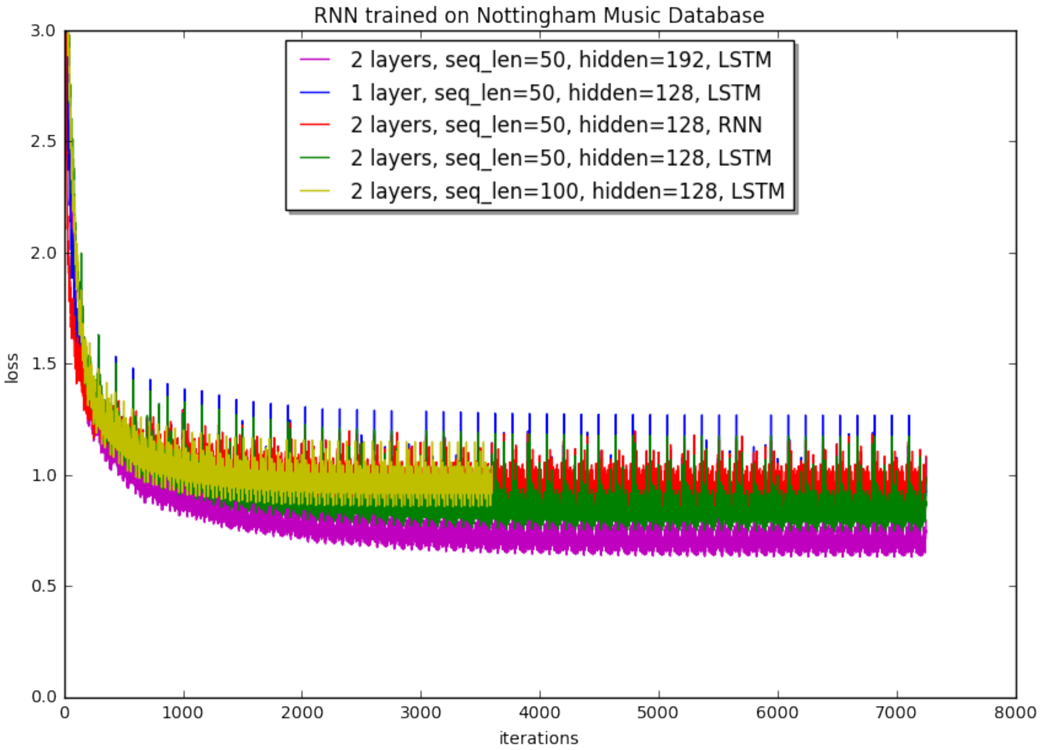

Figure 2. Loss with respect to iterations for various networks

3.2. Music Composition

The Nottingham Music Database was smaller than the Shakespeare dataset so I decided to use similar network architectures to predict music compositions. The first network was a single layer RNN with 128 hidden units, a sequence length of 50, and a LSTM model. The second network had 2 layers, 128 hidden units in each layer, a sequence length of 50, and it used a regular RNN model. The third network had 2 layers, 128 hidden units in each layer, a sequence length of 50, and it used a LSTM model. The fourth network had 2 layers, 128 hidden units in each layer, a sequence length of 100, and it used a LSTM model. The fifth and final network had 2 layers, 192 hidden units in each layer, a sequence length of 50, and it used a LSTM model.

The loss for each network can be found in figure 2. Similar to the results for the Shakespeare dataset, the 2 layer network with the largest number of hidden units performed the best. The 1 layer network was also the worst performing once again. Some outputs from the best network include:

```
X: 202
T:Stonach Wava
% Nottingham Music Database
P:AAB
S:Johnstona, via EF
M:6/8
K:G
P:A
d/2c/2—”G”BAG ”D7”FED—”C”A2G
”D7”AFE—”G”GBd ”C”gec—”G”d2d
”C”e2d—
”G”dcB ”D7”d2d—”G”Bdd ”C”efg—”G”dBG
GBd—”D7”e2d ”Bm”B3—
”C7”GˆEG BGG—”C”EEG FGE—”D7”def
f2d—”C”e3 -eˆde—
”F”a2f ”C/b”gce—”F”f3 -”Gm”b2a—”C”g2d
def—”G7”g2B d2B—
”Em”B3 -d2c—”E7”B3 Bce—”C”A2c ede—
”C”ede c2d—”C”eba G2g—”F”f2f f2d—
”C”e3 e3—c2B c3—”G7”G-E B2 BdG—”C”c3
-cBA—”C”c4–_G-”Em”c——
```

```
X: 109
T:Down Aows the barh
% Nottingham Music Database
P:AABA
S:Trad, arr Phil Rowe
M:6/8
K:G
Bc—”G”d2B G2B—”Bm”d3 B3—”G”dge
dcB—”Em”G2A B3—”Am”g2e ”D””f#7”feB—
```

```
”G”ded B2d—”Am”f2e ”D7”e2A—”G”G3
”G7”G2G—”D7”B2e d2e—”F”c2c cdc—”G”B3
”D7”A4—
”G/b/l7”(3BgB ”F”c2e—”G”d3
d3/2f/2g—”C”c2c fec—”Bm”Bcd ”C”e2d—
”D7”cAF FAd—”G”Bdd dBG—”G”G2G
”D7”E2c—”G”BGG ”C”EGG—
”Gm”ABA BGF—”C”GBc ”G”BAG—”D”EFA
”G”G2:—
```

```
X: 51
T:Doll Falmarve
% Nottingham Music Database
S:MCusker’ Brdoond, via EF
M:6/8
K:D
P:A
A—”D”FAA ”A7”Ade—”D”fgd fed—”Em”edB
”Am”ABc—
”D”ded d3:—[2”G”g3 ”D7”e2ˆd—”G”gfg
gBg—gag age—1”F”afg fed—
-ABA g3——
```

The MIDI equivalent files for these compositions can be found in a Google Drive folder [1] or in GitHub [3].

4. Conclusion

Recurrent Neural Networks build upon Feedforward Neural Networks to solve problems with dependence on temporal information. From the network architectures tested in these experiments it is clear that adding multiple RNN layers helps to improve the accuracy of RNN predictions. Furthermore, increasing the number of hidden units in the hidden layer of the RNN also showed a great increase in the quality of the predictions. If I were to continue running these experiments I would try networks with more than 2 layers and with many hidden units in each layer. It seems that the sequence length hyperparameter may be very problem dependent and making it too large could have negative effects. Even with these small challenges, RNNs perform incredibly well when tasked with processing certain types of data. Due to their temporal nature, and their ability to store memory, RNNs are well suited to solving problems dealing with language.

References

[1] Midi files. https://drive.google.com/open?id=0ByGGMtJwRJz0OWpGZzVUc1R0U00.

[2] Nottingham music database. http://abc.sourceforge.net/NMD/.

[3] Repo for assignment 4 code. https://github.com/ltindall/cogs260project4.git.

[4] J. Johnson. Torch-rnn. Code Repository.

[5] A. Karpathy. Char-rnn. Code Repository.

[6] A. Karpathy. Minimal char-rnn. Code Repository.

Appendix

```
# coding: utf-8

# # COGS 260
# # Assignment 4
#
# ## Author: Lucas Tindall

# ### Tinyshakespeare dataset

# In[58]:


import matplotlib.pyplot as plt

files = ['shakespeare_1layer.txt','tinyshakespeare_rnn_output.txt','tinyshakespeare_console.tx

x_all = []
y_all = []
for name in files:
    with open(name) as f:
        mylist = f.read().splitlines()
        x = []
        y = []
        for a in mylist:
            l = a.split(' ')

            if len(l) == 12:
                x.append(l[6])
                y.append(l[11])
        x_all.append(x)
        y_all.append(y)


plt.figure(figsize=(10,7))
plt.plot(y_all[4],'m', label='2 layers, seq_len=50, hidden=160, LSTM')
plt.plot(y_all[0],'b', label='1 layer, seq_len=50, hidden=128, LSTM')
plt.plot(y_all[1],'r', label='2 layers, seq_len=50, hidden=128, RNN')
plt.plot(y_all[2],'g', label='2 layers, seq_len=50, hidden=128, LSTM')
plt.plot(y_all[3],'y', label='2 layers, seq_len=100, hidden=128, LSTM')

plt.legend(loc='upper center', shadow=True)
plt.title('RNN trained on tinyshakespeare data')
plt.ylabel('loss')
plt.xlabel('iterations')
plt.ylim([1,3])
plt.show()


# ### Nottingham Music Database

# In[63]:
```

```
files = ['music_1layer.txt','music_rnn_output.txt','music_regular.txt','music_100seq_output.tx


x_all = []
y_all = []
for name in files:
    with open(name) as f:
        mylist = f.read().splitlines()
        x = []
        y = []
        for a in mylist:
            l = a.split(' ')
            if len(l) == 12:
                x.append(l[6])
                y.append(l[11])
        x_all.append(x)
        y_all.append(y)

plt.figure(figsize=(10,7))
plt.plot(y_all[4],'m', label='2 layers, seq_len=50, hidden=192, LSTM')
plt.plot(y_all[0],'b', label='1 layer, seq_len=50, hidden=128, LSTM')
plt.plot(y_all[1],'r', label='2 layers, seq_len=50, hidden=128, RNN')
plt.plot(y_all[2],'g', label='2 layers, seq_len=50, hidden=128, LSTM')
plt.plot(y_all[3],'y', label='2 layers, seq_len=100, hidden=128, LSTM')
plt.legend(loc='upper center', shadow=True)
plt.title('RNN trained on Nottingham Music Database')
plt.ylabel('loss')
plt.xlabel('iterations')
plt.ylim([0,3])
plt.show()
```