Structural Testing and Hyperparameter Tuning for Char RNN

Andrew Pesek

Abstract

The Recurrent Neural Network (RNN) is an interesting solution to many sequential processing tasks. Such tasks may include natural language processing, language comprehension and generation, predicting stock market trends, or audio and visual generation. The overall function of a recurrent neural network is to learn patterns in a given body of sequenced data and inturn be able to recurrently generate new data based on previous elements of the sequence. The Char RNN is a unique application of this model meant to train on some corpus of text data and generate new sequences of characters in the likeness of the original text it was given.

1. Introduction

There are many approaches to defining a RNN, and can take on different structures depending on the problem task. In this report I will detail my findings from testing a variety of different cell types, optimizers, and other hyperparameters on the char-RNN task. I will roughly gauge the success of each model based on the reported cross-entropy loss, time taken to train and legibility. I hope at the end of this report to provide a broad range of statistics regarding training and generation of the char-RNN task that can inform and motivate future improvements in performance on said task with further alterations of structure and hyperparameter tuning.

2. Methods

The scope of the testing in this report will include altering and comparing 4 different aspects of the RNN and its training process: Cell type, optimizer, the number of stacked hidden layers, and the size of the hidden layer.

2.1. Cell Type

- Elman RNN cell with **tanh** non-linearity
- Gated recurrent unit (GRU) cell
- Long short-term memory (LSTM) cell

For the sake of clarity, hereafter RNN will refer to the RNN cell type as described above and not to Recurrent Neural Networks in general.

2.2. Optimizer

For each cell type I will train each model with the Adam algorithm and Averaged Stochastic Gradient Descent (ASGD) optimizers, with the learning rates set to 0.005 and 0.05 respectively.

2.3. Number of Hidden Layers

For each optimizer I will train each model with 1 hidden layer, and a stack of 2 hidden layers.

2.4. Hidden Layer Size

For each stack size I will train each model with the hidden layer size set to 100 and 200.

2.5. Dataset

For the dataset, I will be using The Complete Sherlock Holmes, by Arthur Conan Doyle. I chose this dataset because although it is fairly large as far as a single body of text is concerned, it has a consistent

structure and is written in standard english. This is ideal because I would like to assess the performance of each model based in part upon its ability to reproduce correct english spelling and grammatical structures.

3. Experiments

Each model is trained using the pyTorch module and its functions. Each model is trained on 20000 iterations, randomly selecting a 128 char sequence from the dataset to use for each training step. The learning rate is set to 0.005 for the Adam optimizer, and 0.05 for ASGD. After each model is trained I have it generate a 5000 character sample sequence in order to assess the legibility of the generated text. A new line character, "`\n`", is used as an initial sequence for the evaluation function used to generate sample sequences, because I felt it was agnostic enough to not generate similar sequences of text between different models. This is because each character appended to a generated sequence is drawn from a multinomial distribution.

During training the algorithm will report the cross-entropy loss after each step, and at the end of training I use the average loss to gauge how well the model was able to "learn" the dataset. I am also interested in measuring the "readability" of the text generated by the model. So after having the model generate some 5000 character sample of text, I open the text file in Microsoft word in order to count how many english spelling and grammar errors it detects. Then I take the ratio of these numbers and the word count to get a measurement of percentage of incorrectly spelled words. I also record the word and character counts of the sample text in order to compare these against what would be expected from a randomly drawn 5000 character sample of the original text data.

3.1. Performance Results

In order to make sure that each could converge I kept track of the loss values across all iterations. Instead of plotting this sequence of values directly I chose to calculate a rolling average across the previous 100 iterations, then plotted this sequence to smooth out large jumps in losses. Shown in addition is the average loss across all iterations as the horizontal red dotted line, average loss across the last half of iterations in orange, and average loss across the last quarter of iterations in green.

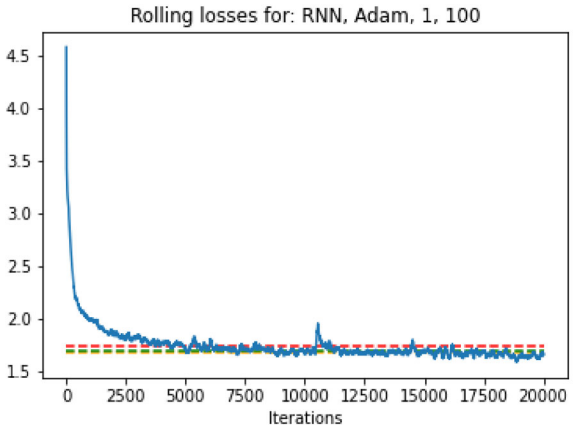

Figure 1. Rolling losses over the 20000 iterations training one of the models (RNN, Adam, 1, 100)

In *Figure 1* it can be seen that the loss converges around a value of about 1.7 quite early on (~5000 iterations) So given the parameters of the model, it was able to 'learn' quite well to the best of its feasible ability.

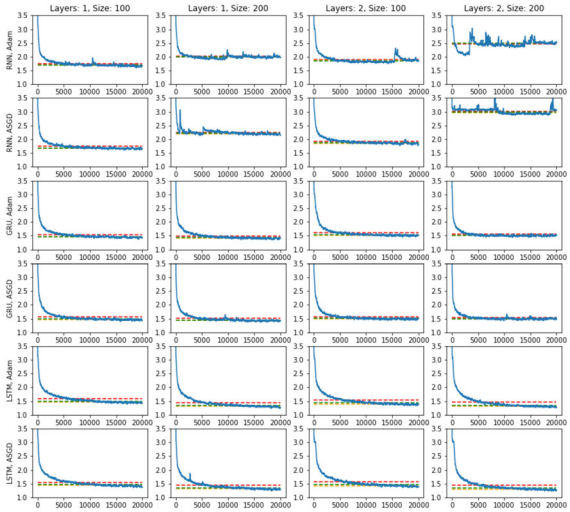

Figure 2. Rolling losses over 20000 iterations of training across all models

As shown in *Figure 2*, most initially trained models were able to converge to a certain loss value early on. Although the models using the RNN (tanh) cell type with 2 hidden layers and hidden layer size of 200 were not able to converge to loss values more inline with the rest of the models, and still showed volatile spiking of loss values across most of the training iterations (see graphs at the top right portion of *Figure 2*).

	Cell Type	Optimizer	# Hidden Layers	Hidden Layer Size	Avg Loss	last half	last quarter	training time (min)	Sample char count w/o spaces	Sample char count w/ spaces	Sample word count	spelling errors	grammer errors	% spelling errors
0	RNN	Adam	1	100	1.753240	1.690466	1.673080	29.160343	3744	4892	819	294	18	0.358974
1	RNN	Adam	1	200	2.011810	2.003207	1.994535	29.708095	3864	4892	723	385	17	0.532503
2	RNN	Adam	2	100	1.908531	1.872844	1.929789	43.287523	3769	4881	785	363	18	0.462420
3	RNN	Adam	2	200	2.457533	2.482899	2.530376	48.069504	4062	4912	699	497	25	0.711016
4	RNN	ASGD	1	100	1.745751	1.673216	1.662270	43.538487	3785	4916	826	297	21	0.359564
5	RNN	ASGD	1	200	2.252878	2.204309	2.192678	29.156702	3690	4867	861	437	25	0.507549
6	RNN	ASGD	2	100	1.932660	1.863559	1.854473	67.390463	3546	4848	798	337	25	0.422306
7	RNN	ASGD	2	200	3.019652	2.958157	2.986661	45.852272	3898	4960	852	559	53	0.656103
8	GRU	Adam	1	100	1.530559	1.448299	1.436921	27.283551	3713	4910	874	204	22	0.233410
9	GRU	Adam	1	200	1.494480	1.410895	1.401998	41.744309	3726	4918	882	196	20	0.222222
10	GRU	Adam	2	100	1.609544	1.515976	1.507607	44.035200	3649	4868	837	199	26	0.237754
11	GRU	Adam	2	200	1.546415	1.498690	1.502236	63.860197	3710	4855	833	234	11	0.280912
12	GRU	ASGD	1	100	1.552805	1.468210	1.456479	41.326073	3715	4890	835	227	15	0.271856
13	GRU	ASGD	1	200	1.505433	1.428692	1.419809	29.475967	3080	4642	757	177	21	0.233818
14	GRU	ASGD	2	100	1.568220	1.488219	1.484442	66.975609	3653	4884	869	253	21	0.291139
15	GRU	ASGD	2	200	1.537782	1.493338	1.489142	43.401833	3631	4893	853	214	23	0.250879
16	LSTM	Adam	1	100	1.581827	1.464306	1.449594	26.358542	3807	4918	852	218	21	0.255869
17	LSTM	Adam	1	200	1.435656	1.317489	1.297824	28.450239	3708	4912	853	134	18	0.157093
18	LSTM	Adam	2	100	1.533304	1.403713	1.384826	59.830156	3715	4907	864	179	15	0.207176
19	LSTM	Adam	2	200	1.458838	1.320205	1.305445	43.398384	3698	4895	857	134	11	0.156359
20	LSTM	ASGD	1	100	1.554531	1.434153	1.417843	28.503249	3555	4854	839	162	16	0.193087
21	LSTM	ASGD	1	200	1.443373	1.324596	1.304677	27.946114	3767	4910	844	165	15	0.195498
22	LSTM	ASGD	2	100	1.575714	1.433288	1.413820	58.016687	3772	4912	837	189	14	0.225806
23	LSTM	ASGD	2	200	1.447734	1.305508	1.286445	60.512118	3764	4913	863	127	13	0.147161

Table 1. Performance Results

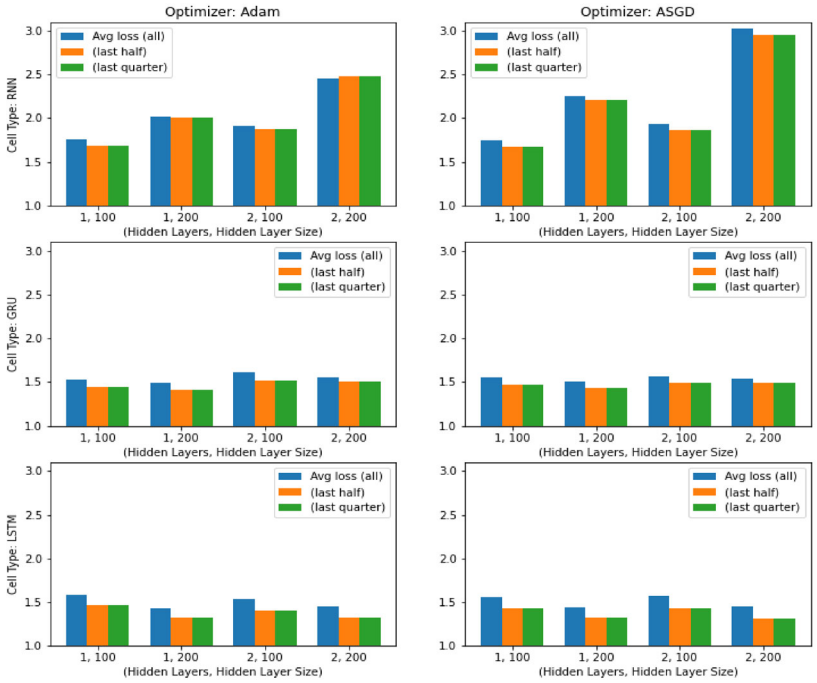

Figure 3. Comparison of average losses over all, last half and last quarter iterations

Shown above in *Table 1*, are recorded performance metrics for each model initially trained. The performance of each model is assessed based on average cross-entropy loss it was able to achieve during training, the percentage of spelling errors and number of grammar errors occurring within a generated sequence of sample text, and how much the generated sample text deviated from the expected word and character count.

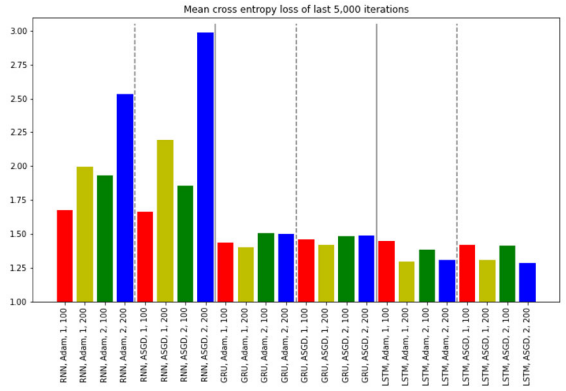

Figure 4. Last quarter mean cross entropy loss

Overall both GRU and LSTM were able to minimize loss the best, especially with the larger hidden layer stack and size for LSTM. However RNN was not able to achieve lower losses with the larger stack and layer size.

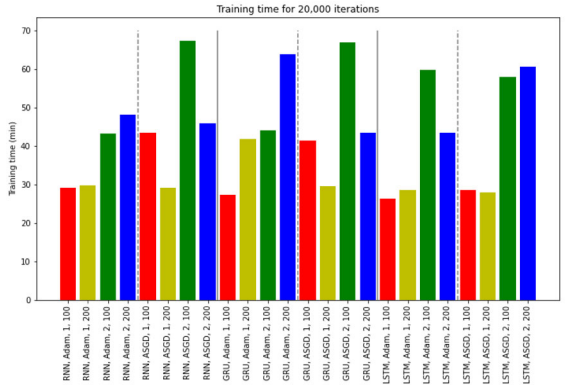

Figure 5. Approximate training time per model

While the models with only 1 hidden layer were able to train over the 20000 iterations in a shorter amount of time than those with 2 hidden layers, it is interesting that for most cell types and optimizers, the models with 200 hidden layer size (blue bars) were able to train in a shorter amount of time than those with 100 hidden layer size (green bars) for a hidden layer stack size of 2. Aside from this fact, larger hidden layer stack sizes would seem to significantly increase the training time. Of the cell types, optimizers, and hidden layer sizes tested, training time generally varied rather minutely between them. LSTM with 1 hidden layer with size 100 and Adam optimizer took the shortest amount of time to train (~26.36 min).

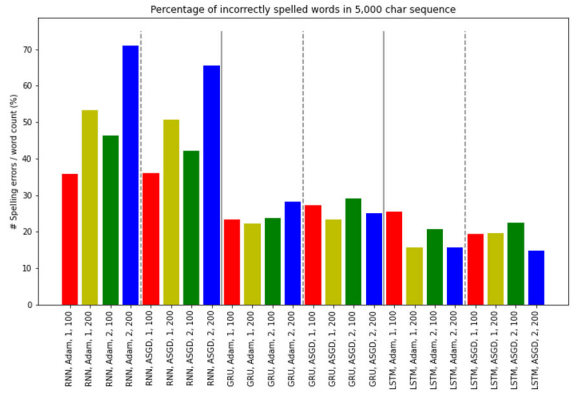

Figure 6. Percentage of incorrectly spelled words

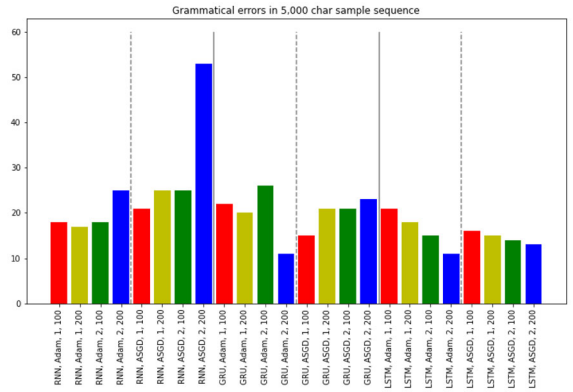

Figure 7. Number of grammatical errors

As far as incorrect spelling and grammatical errors are conconcerned, both GRU and LSTM cell types did much better than RNN minimizing language errors, with LSTM performing slightly better than GRU. Larger hidden layer sizes and 2 stacked hidden layers did more to reduce these errors in the case of LSTM, however for GRU and especially RNN these errors were actually perpetuated more often than with

a smaller hidden layer size and 1 hidden layer. LSTM with 2 hidden layers with size 200 and ASGD optimizer produced the smallest percentage of spelling errors (~14.72%). GRU with 2 hidden layers with size 200 and Adam optimizer produced the fewest detectable grammatical errors (11).

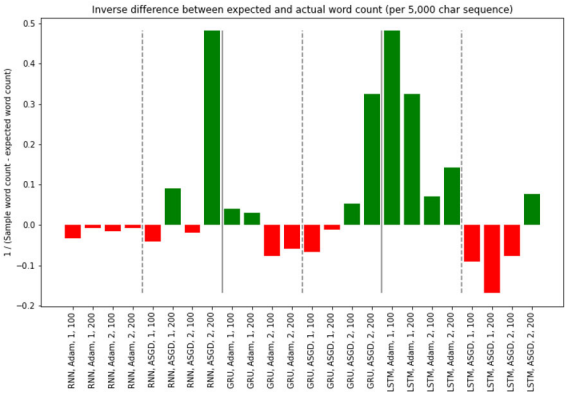

Figure 8. Inverted deviation from expected word count

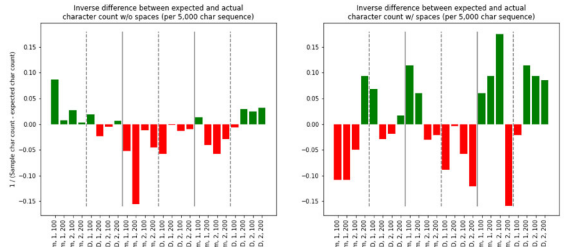

Figure 9. Inverted deviation from expected character count

To compare how much the generated sample deviated from expected word and character counts, plotted in *Figure 8* and *Figure 9* is the inverse of the difference between the counts in the sample and what would be expected based on the original data (~850 words, 4901 characters with spaces, and 3732 characters without spaces per 5000 char sequence). This is so that models that generate text with word and character density closer to the original data are more apparent in the bar graphs. These figures show RNN with 2 hidden layers, 200 size, and ASGD optimizer; as well as LSTM, 1 hidden layer, 100 size, and Adam optimizer were very close to the expected word count of 850 words per 5000 chars, by +2 words difference.

4. Conclusions

According to the suite of hyperparameter testing I have demonstrated, it appears that the LSTM cell type works the best overall for the char RNN task. This seems to be because they tend to work very well when they are stacked, especially in comparison to RNN cell type. GRU performed very similarly to LSTM, however it is still just slightly beat-out on nearly every metric I recorded. Perhaps GRU may be a better choice depending on the nature of the data given to it, however LSTM seems to be the best choice when stacking layers, because stacking LSTM cells didn't degrade the performance unlike RNN and GRU cells. The comparison between the Adam and ASGD optimizers was somewhat inconclusive, with both optimizers performing quite similarly. However, Adam did tend to beat-out ASGD in terms of the time taken to train, whereas ASGD beat-out Adam in terms of minimizing spelling and grammatical errors. So between the two, there is a slight training time versus accuracy trade-off.

Here is a portion of the text generated by one of the better performing models, LSTM, Adam, 2, 200:

```
     "I have bad? What spifical
which had gone me, made forward,
not secure to set hard?" shead."

     "No, you remember reconcruse
for gov lass came at Holmes daegh
upon last hold no abown one
sergey
     upon the mistare within the
wood to writy and prompederously
of her. Would, for I observed
oves, there was neary. There was
no ordinary gashe. This
     proceeded with address,
literlysatic stood and necking
them?"

     "The whebough thoughts pass
man. Dinch
     which of cannot dog twalgeed
it well, whispered. Pum
     pointed the nights of
togetree and the safe. At this
warm-steps now withere than the
natural gard to seegnor
     that you be what a work, to
the desure and
     conturning and strange
bidgement stranger save," he
```

```
shoutless, and unseaded up, jucy
and
     hurry,
     Mr. Holmes, we have a
hure-build and intellige when I
hall, there was part," said
Moriar, to still. They had
     managely some greated to the
road. "Beside, whow you caught my
own which I dust you also. So I
     all man's eye.
     Pound with a good-and
opinion to say with wather
     and each of perveries as to
Mrs.  Mirrights! And again and
broken that he had a still roagh
it will usked to part in my very
methode.
```

As is the case with most character based RNN models, the essence of the generated text is almost entirely nonsensical, devoid of any interpretable meaning. What it can do however, is replicate patterns that occur throughout the text. Provided the small sample above, it can spell english words properly for the most part, and most of the phrases are grammatically acceptable. It seems to understand the format of the text provided, given the use quotation marks, with each line of dialogue printed on a new line. And it can reproduce names, such as "Holmes" that obviously appear frequently throughout the text. We can compare this against a small sample generated by one of the models that performed more poorly, RNN, Adam, 2, 200, which produces text not too dissimilar from the following:

```
  , yle pathty beloceruth fotaws
to, aterpelauly sulase-of wriIll
thos tho fesese us podos reuu ve
brsinglh cus are, erseiribesp.
Weibayplrel, asfpete asel pres
untirfer tte fe se th-ris fepy on
th. herregerely unlatve rons, tod
asobugsos lore
yroty red"rirl feyt,tys ropnerelg
yoosurdsyDlalres ilce bas ure
soremeparg.  "  Dutes u, ere.
Dsesnenorrodad    siDdre    mes
rerfireessyjepy
 rolpesefanesosdiflapaple heyisey
pobof eolrer r worerhy icobl many
ulraner et pisicopctalfyses Du
pis thherripeus osonle. as ony
ahf-ure lloce. hals ale Diryrawl
Sary bisga
   en us sossrorgdre pos'simotten
baby ttectsronmesnsaath os.syetos
cicy    par    shiminrotovtterh,
yobase.oisfoer    ron    iltrill
oursireshe ayamemupangp Dus fir
tis moapumlhocd ron rlourenh-lad
Gfe, otamlyle elfhad

.serhasonkrol ut ril.
   Thaunas bgruD, maayo aproif,
upty lit. Dharsirlsirorcocr-'pe,
thoce stasellmat ye thle, le oy
   , os, ssice
 . De surlle cof
```

Obviously, none of this resembles english. This model was able to pick-up on the use of quotation marks to an extent, it being the most repetitive pattern across the text as a whole. However, the rest of the sample is mostly just random characters strung together at roughly the normal lengths of english words.

Overall, upon further testing of LSTM cells with different hidden layer sizes and stacks (testing with an additional stack size of 3, and additional hidden layer sizes 50, 150, and 300), I found that larger hidden layer sizes (upwards of 300) produced even better loss values. The number of layers was still the best within 1 or 2 hidden layers, with a stack size of 3 producing higher loss values. Finally, both optimizers performed equally well across the board.

A github repository with all files associated with this project as well as the results data and relevant spreadsheets can be found at:
https://github.com/apesek/charr-rnn-project

References

Karpathy, A, "The Unreasonable Effectiveness of Recurrent Neural Networks", *Andrej Karpathy Blog*, May, 2015, https://karpathy.github.io/2015/05/21/rnn-effectiveness/

Lipton, Z, Berkowitz, J, Elkan, C, "A Critical Review of Recurrent Neural Networks for Sequence Learning", *arXiv*, June, 2015, https://arxiv.org/pdf/1506.00019.pdf

van den Oord et al., "WaveNet: A Generative Model for Raw Audio", *arXiv*, 2016, https://arxiv.org/pdf/1609.03499.pdf