**A comprehensive exploration on char-level RNN and LSTM**

**Jin Gan**

J6gan@ucsd.edu

University of California, San Diego

**Abstract**

Char RNN is a simple yet relatively powerful character-level recurrent neural network that recursively trains on its own outputs. Given a training text and a starting character as the first input, the network would predict the most likely next character and use this result as its next input, over and over throughout the whole training process. The end result would be a model capable of generating text of similar writing style to the training text given a text length. In this paper, I would explore the performance of the LSTM version of char RNN. The aspects will include performance with different hyperparameters, on different texts, on different languages, on different contexts combined and on different initial character.

## 1.Introduction

Recurrent Neural Network has been widely adopted in modern NLP tasks for long, but many who have little understanding on the matter may consider it as something extremely complicated. In this paper, I will introduce and explore an easy yet surprisingly effective language model called character-level LSTM. As an upgraded variant of char-level RNN, the model recursively trains itself on a given text and generates new text that resembles the training text in both content and writing style, given the wanted text length and starting character.

## 2.Method

2.1 Structure of RNN and LSTM

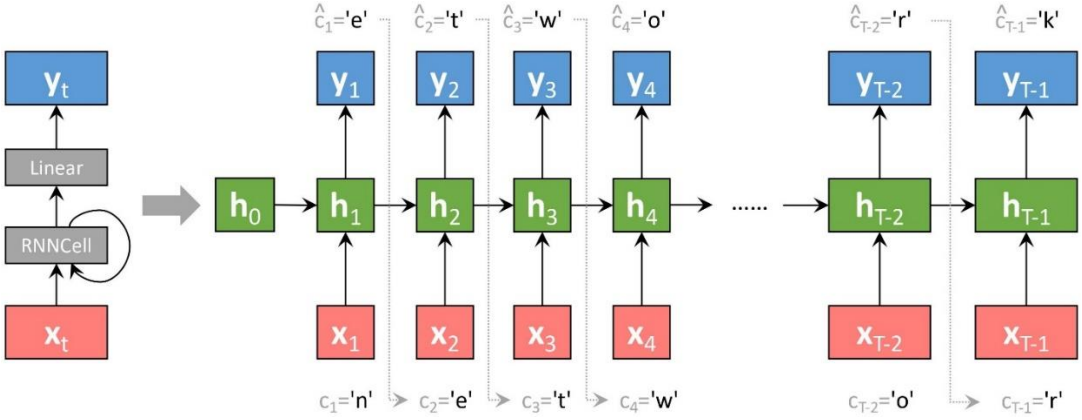



Figure 1: Illustration of RNN's structure design

As shown above, the main task of a char RNN is to train the network on how to accurately predict the next character given the current character. The right part is the unrolled version of the network showing this network's recursive nature. First we create a set comprised of all possible characters appeared in our training text, then convert the input character to the one-hot-encoding of this character's class. Next, base on this encoded character and the previous hidden state, we calculate the current hidden state and output. Input output to Softmax function to obtain the probability vector, sample using our encoded class and thus find the corresponding character, which is of the highest probability, as our predicted character. Lastly, recurse by using this predicted character as our next input and repeat the steps above. The main Pytorch functions used are nn.RNNCell() and nn.Linear, using cross-entropy as loss function and 100 hidden neurons. Adam is used for optimization here with a learning rate of 0.005, five times of its default value.

Upgraded from vanilla RNN, LSTM is just a variant of RNN with tighter activation enforced by various gates. The hidden nodes are replaced with memory cells so that the gradient wouldn't explode or vanish like in vanilla RNN. Implementation-wise, mainly replace nn.RNNCell() with nn.LSTM() and update shapes and loops accordingly.

To visually see the results and compare them, the evaluation step takes in a length parameter of how many characters to generate and spits out that amount of text starting from a specified initial character. This initial character is manually set and could be changed on demand.

### 2.2 Dataset

My datasets include a selection from Shakespare, political speeches in German, Wuxia novel in Chinese and a context-different text combined from Wikipedia, code and Shakespeare. Each text has round 1million characters to ensure there is enough training data.

## 3.Experiement

First, train both RNN and LSTM using the Shakespare text. Compare their results to see if LSTM indeed performs better. Next, train LSTM on the same text but with different hyperparameters: change hidden size from 100 to 512 and replace Adam with Adagrad, AdamW, Adamax and RMSprop as optimization function. The learning rates are set to default rate times 5 to approximate the learning rate implementation of Adam. Compare with the original implementation of hidden size 100 and Adam for optimization.

Afterwards, train LSTM on languages other than English. Input the German political speech text and the Chinese Wuxia novel text as new training files. Because there's no built-in character database for German and Chinese in python, we read in each file character by character first to create a set including all unique characters appeared in the text as our database. Train and see if the resulting generated texts make sense. Then, change the initial character in evaluation from a random character to a purposely-picked character that fits more into the context. See if there's any improvement. Lastly, combine Shakespeare, Wikipedia and code texts to create a new text file of 1 million characters and train LSTM on it to see if our model distinguishes the difference of contexts in its result.

Note that the generated text is fixed to length 600, which means it'd always produce 600 characters using the trained model, starting from the initial character I set in evaluation step. For Shakespeare and German speeches, it's "W" because it's a common letter to start an alphabetic word with. For the combined text, it's "F" because it commonly appears in code text's "for" keyword. For the Chinese text, I used "在", which means at someplace or in the process of doing something, and "方", which is the initial character of the training text's protagonist's name.

We can also visualize the output:

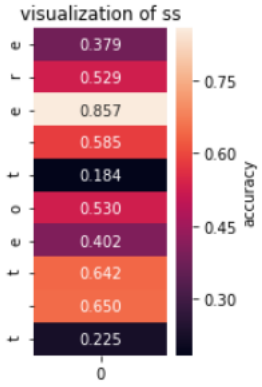



Figure 2: Example visualization of output extracted from LSTM training on Shakespeare. Input each letter on y column, the probability of the next character is shown in its corresponding cell, and next letter itself is the letter underneath. For example, e has a probability of 0.379 to get r, and r is the next letter.

**4.Conclusion**

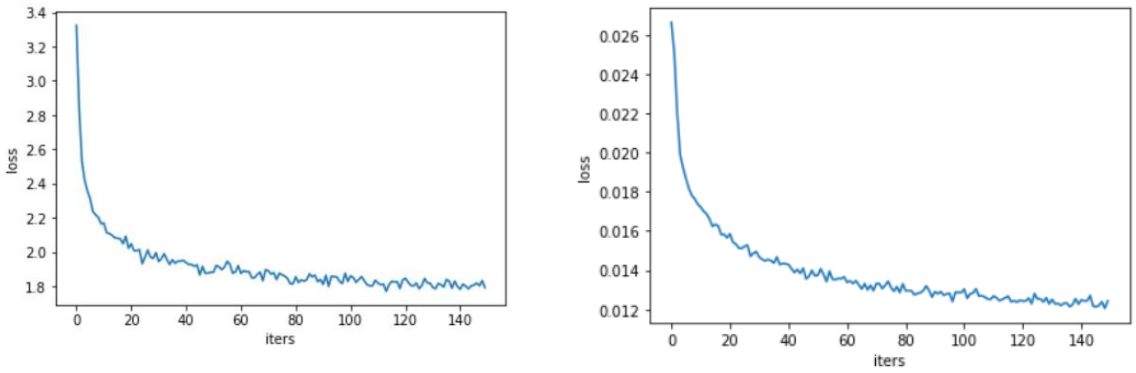



Figure 3: Training loss curve of RNN on the left and LSTM on the right

As shown in the loss curves, LSTM does have better performance than vanilla RNN by reducing the final loss from 1.8 to 0.012. The generated text of RNN is as follow:

```
What I niares, protals, Loff let hear prarh our mebssilly in brok not at this pretroth may tlaadle
n, but a obouse.

First Clord:
That me?

ISABERLAND:
AXd for cank uphish them that fir plute
Have you are conch me or mas best so, hear aratters stauntisty slady leftury.

Agonot swemples,
As news.

HENRY Le:
'That of saysed-'sindechald and or nebs as I case or soar,
But thus have illainger'd man your my master make's ongee anchiss's this morse nobrave
That shall do a will from sorfor prattart,
As myself o? Whine.
Offams of Goos wack, 'tworr:
Sirced,
Ane she
Comfils danger,'t the warth, morry, 'sop
```

Figure 4: RNN's generated text

As demonstrated above, the result does follow a similar writing style to the training text Shakespare. Although it doesn't make logical sense as a whole, the component words are very much normal English and we can still understand some pieces of information here. The LSTM result is as follow:

```
With themsels, good with now, do;
Since Keem.

TTAN:
Go that you are farewell, and his touch my heartsy uppare to rups citiler will be sterms! neats Ques it that stentious his part
I know the triught of her plearer lost to do the none yet, and bring to this but now to they I a need.
Likes by and speaky mine queen on mores, I am wise father soun! Volintance.
They reating deuzes' Warwick and give not stepke pall the known. While.

STRARD LAHN:
As for a
know on him, I will see year an and fife
Whose infict old protess; Marcius of desire and tuth these make your such court, to fielding from an all
```

Figure 5: LSTM's generated text

It has a better word flow and makes even more sense than the previous RNN result. Combined with its lower loss, our result does show that LSTM is generally better than RNN on character-level NLP task.

Then we tune the hyperparameters of LSTM to find the hidden size and optimization function with the best performance. Note that the optimization functions are all run under the hidden size of 100. Each change has two trials to avoid randomness and the losses are kept in four decimal places:

| Hyperparameter | Loss in trial 1 | Loss in trial 2 | Mean |
|---|---|---|---|
| Adam + Hidden size 100 | 0.0124 | 0.0122 | 0.0123 |
| Hidden size 512 | 0.0276 | 0.0158 | 0.0217 |
| Adagrad | 0.0137 | 0.0143 | 0.0140 |
| AdamW | 0.0119 | 0.0123 | 0.0121 |
| Adamax | 0.0122 | 0.0111 | 0.0117 |
| RMSprop | 0.0271 | 0.0222 | 0.0247 |

Figure 6: Loss under each hyperparameter

As shown above, hidden size 100 works a lot better than hidden size 512. The ranking of optimization functions is Adamax>AdamW>Adam>Adagrad>RMSprop, from lowest to highest loss. Adamax, AdamW are both variants of Adam and they only perform slightly better than Adam in this case, which means they are still all on the same tier of performance with Adam and the slight difference here could be only due to chance. Adagrad falls on the lower tier since it has relatively a large gap with the Adam family. RMSprop is on the lowest tier since it's significantly worse than all the others. This result is understandable because LSTM is essentially the combination of advantages from Adagrad and RMSprop, which means it should in theory get better result.

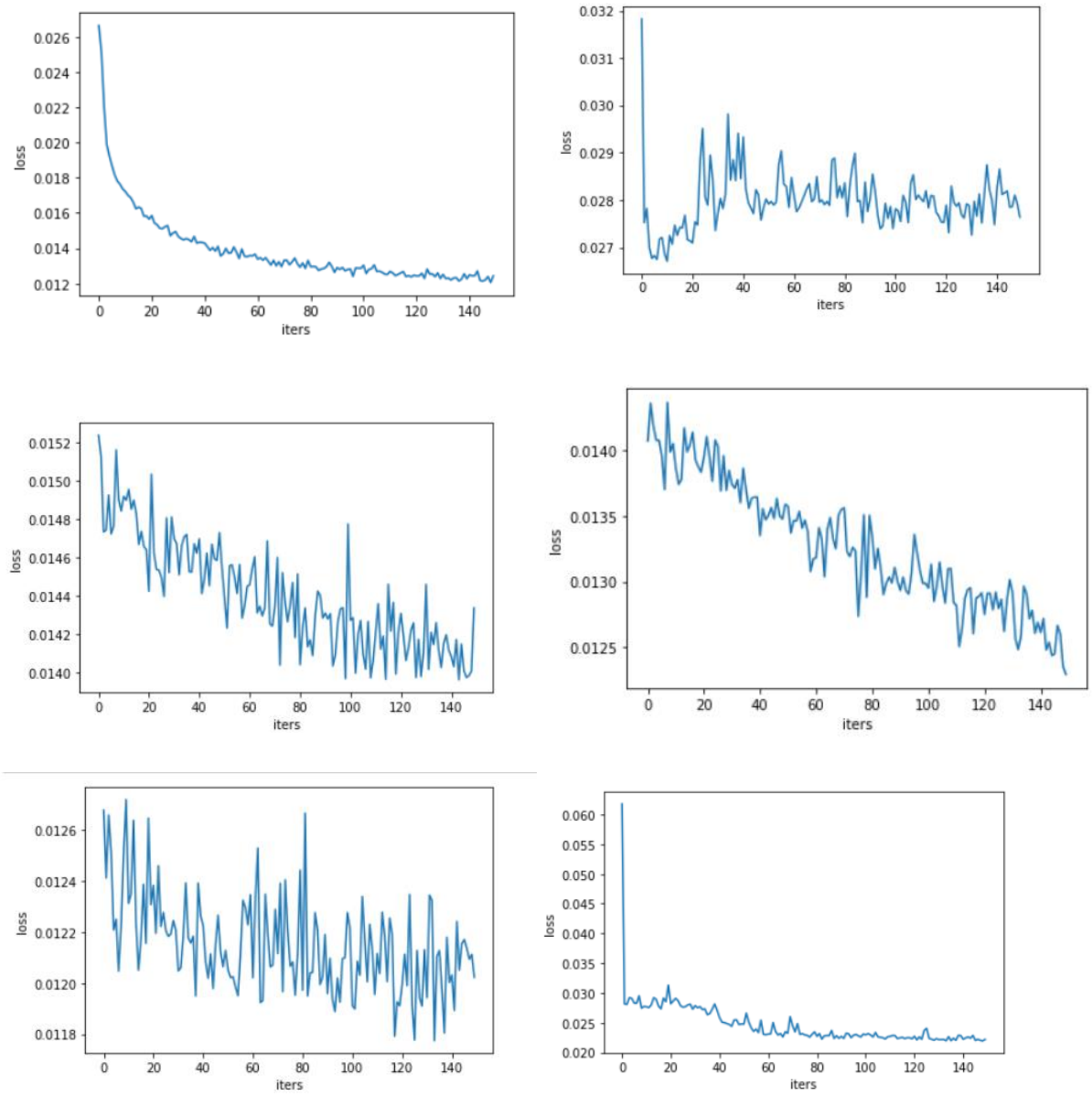
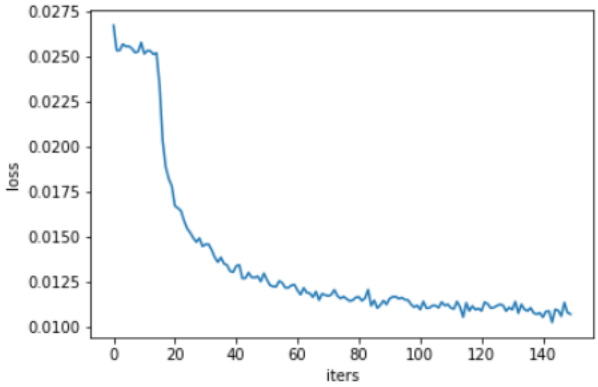



Figure 7: Training loss curves of different hidden sizes and activation functions. From left to right, the graphs correspond to Adam with hidden size 100, Adam with hidden size 512, Adagrad, AdamX, Adamax, RMSprop.

When comparing loss curves, we can see that although AdamX and Adamax have slightly better loss than Adam, their training loss curves are much more chaotic and fluctuating. In comparison, Adam's curve fluctuates much less and converges much more quickly, which makes training a lot smoother.

Therefore, I chose to keep the hyperparameters of hidden size 100 and Adam as optimization function. Then I trained my LSTM on languages other than English to see if performs any differently on non-English text. The result of training it on German political speeches is as follow:

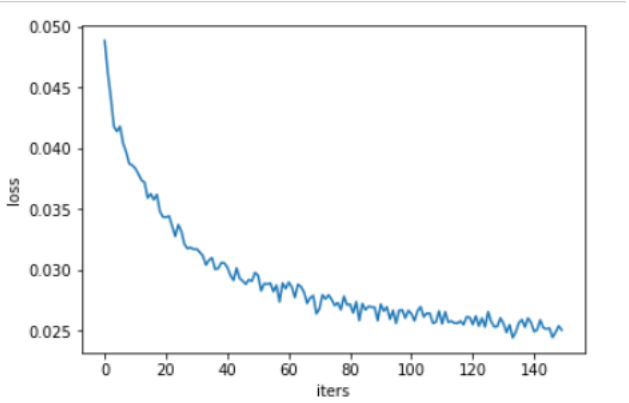


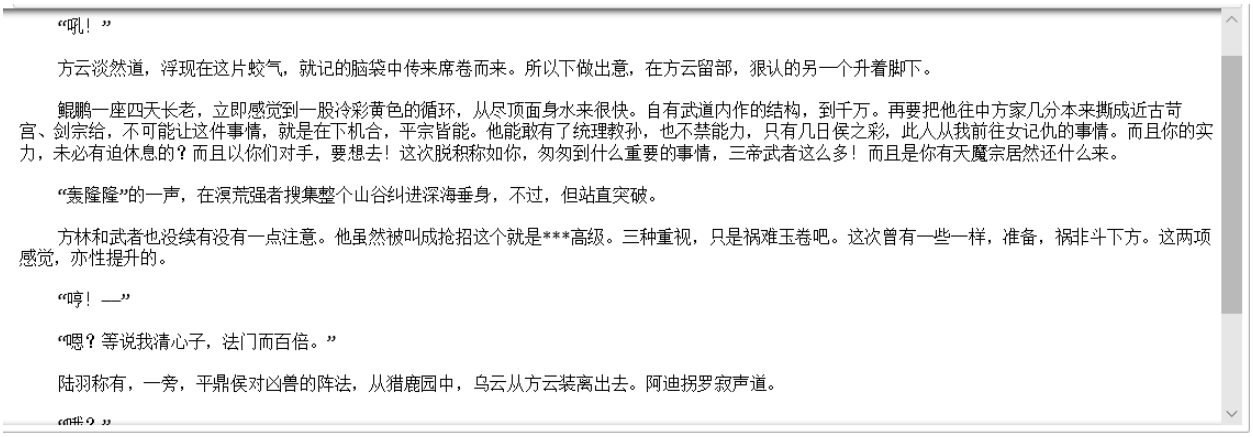

```
Weinen Laben tro-Reformminister man mmstretzen wir haben in Afrika noch auch errächt erlässt es Aus).
Vielen Sicherheitshalten der Orientikproret vorgesparten und Der Kamanzreffensverlehgegentehpf und sodern der intern wirtschaft
en, Pristration Europäer fordern wird.
Bei die internativister machender teilnütz 25.03 Mol besondersichten: &quot;den Mittel, Deutschland für solchen Iran ausgeworde
r gesetztert in der Wemmen des Hulden in Solassore noch kommnamentation Abzucht auch gleichten des Amten der deutschen unseren
Hamanten wird es zur Konferen ihren Partner und nach dem ohne, wir dass matili
```

Figure 8: Training loss curve and generated text when trained on German political speeches

The loss curve looks fine and the final loss is 0.0107, which falls in the reasonable range similar to English text. Since I personally don't know German, I can't tell how much the generated text makes sense. However, just from appearance, it does have the look of German and it may be hard to distinguish the difference if someone who doesn't know German skims through it. Therefore, this LSTM model does work on non-English texts.

Later, to check if it also works on non-alphabetic languages like Chinese, I trained it on a Chinese Wuxia Novel and obtained the following result:





“吼！”

方云淡然道，浮现在这片蛟气，就记的脑袋中传来席卷而来。所以下做出意，在方云留部，狠认的另一个升着脚下。

鲲鹏一座四天长老，立即感觉到一股冷彩黄色的循环，从尽顶面身水来很快。自有武道内作的结构，到千万。再要把他往中方家几分本来撕成近古苛宫、剑宗给，不可能让这件事情，就是在下机合，平宗皆能。他能敢有了统理教孙，也不禁能力，只有几日侯之彩，此人从我前往女记仇的事情。而且你的实力，未必有迫休息的？而且以你们对手，要想去！这次脱积称如你，匆匆到什么重要的事情，三帝武者这么多！而且是你有天魔宗居然还什么来。

“轰隆隆”的一声，在溟荒强者搜集整个山谷纠进深海垂身，不过，但站直突破。

方林和武者也没续有没有一点注意。他虽然被叫成抢招这个就是***高级。三种重视，只是祸难玉卷吧。这次曾有一些一样，准备，祸非斗下方。这两项感觉，亦性提升的。

“哼！—”

“嗯？等说我清心子，法门而百倍。”

陆羽称有，一旁，平鼎侯对凶兽的阵法，从猎鹿园中，乌云从方云装离出去。阿迪拐罗寂声道。

“我？”

Figure 9: Training loss curve and generated text when being trained on a Chinese Wuxia novel

The loss curve still converges fine here, but the final loss is 0.0251, which is relatively larger than our standard loss when compared with previous alphabetic texts. The reason may be that Chinese isn’t a letter-based language, meaning our set that includes all unique characters appeared in the training text could easily reach a capacity of a couple thousand, in this case 3858. This disadvantage may increase loss. For similar reason, other languages consisting of unique characters rather than only alphabetic letters would need a fairly large database to identify existing characters and extract next character’s prediction.

The resulted text, although still not good enough to make perfect sense, is still fairly readable and gives a basic idea of what’s going on. However, I wonder if giving an initial character that fits more into the context could improve the generated text, thus I changed the setting from “在”, meaning at some place or doing something, to “方”, the last name of the protagonist. The new evaluation result is as follow:

方云！”

方云孙世堑摆了望里，粼片的条蛇向，从黑衣男子同时一凛。一排没有任何，似朝庞大的世界，基本上无法被连自然大手。同时在这个之能，休息片刻后，他就在便把冠军侯府而出的山谷，冲过巫王之中。

时间有些答复所有。恨触也没指。他直接就要别人打理何恐思。但也不过给象极大的朝廷传人。只是第一次见得不跑派自臣，夷荒长部榜脚，残回突然之间的“翰木大法”，神丹被鲲鹏精血实裂。

“海，反后，北溟和华阳夫人的敌人，武道凶酷，就完全有所多的极非？”

“天空也只必掌力已经掉到了功力无物“哼！”

方云一丝死了刀光，突然凭空道。

“天邪宗”的霸主情，和跳耀下。

六道强大的魔道巨擘，而有种能力！忠太和三公一肩已经人，伏进的，是嗤以背后如视伏方毒，也超越之机。—等后你们，没发计铁的野心，坐镇底子来起：

“中古魔神的绝学。……—”

拳刀云巨响，一个上古魔道修为，胸袖护弱的白色刀腿，在众人、虚品炼制的武者，直接被身上严肃祭练的速度。这根一，突然，他已经就在信纸上，江木木骸。

“回去对面不该要你说。”

方云身躯一默、生判飘烟，同时呈双持能***控的，都要据一种交妙的感觉。依然暗暗沉叫一下。

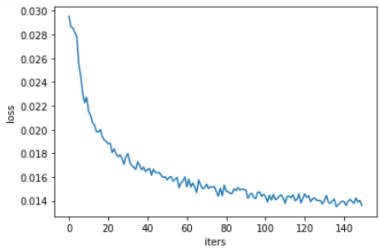
“落不过，我还会半个年短的神兽！”

马车厢里文形一张，立即将气息闪而，踏入自己面前。一只巨大的丫凰

Figure 10: Generated Chinese text after changing the initial character given for evaluation

Although still not as good as the real one, it's better than the last trial in logical connection, writing style and word flow, which makes it easier for readers to picture what's happening. This result shows that the generated text could be improved if the given initial character fits more to the specific context of our training text.

At last, I wonder if this relatively naïve LSTM could distinguish the context difference within the training text, thus I trained it on a text combined from three totally different information types: Wikipedia, code and Shakespeare. The result is as follow:




**Evaluation: A Sample of Generated Sequence**

```
print(eval_step(net, predicted_len=600))
```

```
F[high_mags & that &quot;Tmemans_opdrach(internize (nlidopograunt the work 18 1 his Animasoc Sharling arown, in normable tod of
either.

CORIOLANUS:
Cort part pos, you he, you chay.

Metay him guring intelly to hear may, no,
Till you love of detafliciance stan skely Darciliculy down:
I do't concrang resulting do my.

Forth may be is
will spayet Eshmusted lick you it, he make's small your proteen carl Aphiin I can thereforeing here if therefores' the have the
fack, corators reflofactiongy,
And the World. COMING HARD:
And to be his into the hotess popules.
More tarnens,
I wall patcoul, come yours
```

Figure 11: Training loss curve and generated text when being trained on the combined text of various contexts.

As shown above, although we still have a good loss, the resulted text doesn't actually reflect the difference among code, Wikipedia and literature. It barely has any distinguishable feature that resembles code or wiki index, and ,although the overall format is similar to Shakespeare, the text makes less sense compared to the previous examples. This result shows that our simple char-level LSTM cannot properly distinguish and reflect the difference on context.

**5.References**

Karpathy A., (2015). “The Unreasonable Effectiveness of Recurrent Neural Networks”, Andrej Karpathy blog. https://sites.google.com/site/ucsdcogs181spring2020/classroom-news/finalprojectdraft

Barbaresi, Adrien (2018). "A corpus of German political speeches from the 21st century", Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), European Language Resources Association (ELRA), pp. 792–797.

http://purl.org/corpus/german-speeches (BibTeX entry)

Adrien Barbaresi. (2019). German Political Speeches Corpus (Version v4.2019) [Data set]. Zenodo.

https://doi.org/10.5281/zenodo.3611246

**6.Bonus point**

Novel ideas: Tested LSTM on non-English texts and found a way to create custom database of unique characters appeared in the training text. Also tested if it can distinguish the context difference within the training text.

Data finding: Effort in collecting and preprocessing data, although unfortunately didn’t find a better text for English literature than Shakespeare either due to text length or illegal characters.