Deep Learning Assignment 2. Recurrent Neural Networks

Andrii Skliar
11636785
University of Amsterdam
andrii.skliar@student.uva.nl

1 Vanilla RNN versus LSTM

1.1 Vanilla RNN in PyTorch

Question 1.1

Note following notation specifics:

$$\mathbf{h}^{(t)} = tanh(\tilde{\mathbf{h}}^{(t)})$$
$$\tilde{\mathbf{h}}^{(t)} = \mathbf{W}_{ph}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h$$
$$\mathbf{1} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}$$

$$\frac{\partial L^{(t)}}{\partial \mathbf{W}_{ph}} = \frac{\partial L^{(t)}}{\partial \tilde{\mathbf{y}}^{(t)}} \frac{\partial \tilde{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{W}_{ph}}$$
$$\frac{\partial L^{(t)}}{\partial \tilde{\mathbf{y}_i}^{(t)}} = \frac{\mathbf{y}_i}{\tilde{\mathbf{y}}_i}$$
$$\frac{\partial \tilde{\mathbf{y}_i}^{(t)}}{\partial \mathbf{p}_j^{(t)}} = \tilde{\mathbf{y}_i}^{(t)}(\delta_{ij} - \tilde{\mathbf{y}_j}^{(t)})$$
$$\begin{aligned}
\frac{\partial L^{(t)}}{\partial \mathbf{p}_j^{(t)}} &= -\frac{\mathbf{y}_j^{(t)}}{\tilde{\mathbf{y}_j}^{(t)}} \tilde{\mathbf{y}_j}^{(t)}(1 - \tilde{\mathbf{y}_j}^{(t)}) - \sum_{k \neq j} \frac{\mathbf{y}_k^{(t)}}{\tilde{\mathbf{y}_k}^{(t)}}(-\tilde{\mathbf{y}_k}^{(t)} \tilde{\mathbf{y}_j}^{(t)}) \\
&= -\mathbf{y}_j^{(t)}(1 - \tilde{\mathbf{y}_j}^{(t)}) + \sum_{k \neq j} \mathbf{y}_k^{(t)} \tilde{\mathbf{y}_j}^{(t)} \\
&= -\mathbf{y}_j^{(t)} + \mathbf{y}_j^{(t)} \tilde{\mathbf{y}_j}^{(t)} + \sum_{k \neq j} \mathbf{y}_k^{(t)} \tilde{\mathbf{y}_j}^{(t)} \\
&= -\mathbf{y}_j^{(t)} + \tilde{\mathbf{y}_j}^{(t)}(\mathbf{y}_j^{(t)} + \sum_{k \neq j} \mathbf{y}_k^{(t)}) \\
&= -\mathbf{y}_j^{(t)} + \tilde{\mathbf{y}_j}^{(t)} \\
&= \tilde{\mathbf{y}_j}^{(t)} - \mathbf{y}_j^{(t)}
\end{aligned}$$
$$\frac{\partial L^{(t)}}{\partial \mathbf{p}^{(t)}} = \tilde{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}$$

$$
\begin{aligned}
\frac{\partial p_k^{(t)}}{\partial (\mathbf{W}_{ph})_{ij}} &= \left(\frac{\partial p_i^{(t)}}{\partial (\mathbf{W}_{ph})_{jk}} \right) \\
&= \left(\frac{\partial \left((\mathbf{W}_{ph})_{i*} \mathbf{h}^{(t)} + b_i \right)}{\partial (\mathbf{W}_{ph})_{jk}} \right) \\
&= \begin{cases} 0, & \text{if } i \neq j \\ \mathbf{h}_k^{(t)}, & \text{if } i = j \end{cases} \\
\left(\frac{\partial L}{\partial \mathbf{W}_{ph}} \right) &= \frac{\partial L}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{W}_{ph}} \\
&= \sum_i \frac{\partial L}{\partial \mathbf{p}_i^{(t)}} \frac{\partial p_i^{(t)}}{\partial \mathbf{W}_{ph}} \\
&= \frac{\partial L}{\partial \mathbf{p}^{(t)}} \mathbf{h}^{(t)^T} \\
\frac{\partial L^{(t)}}{\partial \mathbf{W}_{ph}} &= (\tilde{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}) \mathbf{h}^{(t)^T} \\
\frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}} &= \mathbf{W}_{ph} \\
\frac{\partial L^{(t)}}{\partial \mathbf{h}_l^{(t)}} &= (\mathbf{W}_{ph}^T)_{l*} (\tilde{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}) \\
\frac{\partial \mathbf{h}_l^{(t)}}{\partial \tilde{\mathbf{h}}_l^{(t)}} &= 1 - (\mathbf{h}_l^{(t)})^2 \\
\frac{\partial \tilde{\mathbf{h}}_l^{(t)}}{\partial (\mathbf{W}_{hh})_{ij}} &= \sum_{k=1}^{H} \delta_{il} \delta_{jk} \mathbf{h}_k^{(t-1)} + (\mathbf{W}_{hh})_{l*} \frac{\partial \mathbf{h}_k^{(t-1)}}{\partial (\mathbf{W}_{hh})_{ij}} \\
\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} &= \frac{\partial L}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial \tilde{\mathbf{h}}^{(t)}} \frac{\partial \tilde{\mathbf{h}}^{(t)}}{\partial \mathbf{W}_{hh}} \\
&= \sum_{l=1}^{H} (\mathbf{W}_{ph}^T)_{l*} (\tilde{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}) \left(1 - (\mathbf{h}_l^{(t)})^2 \right) \sum_{k=1}^{H} \left(\delta_{il} \delta_{jk} \mathbf{h}_k^{(t-1)} + (\mathbf{W}_{hh})_{l*} \frac{\partial \mathbf{h}_k^{(t-1)}}{\partial (\mathbf{W}_{hh})_{ij}} \right) \\
\frac{\partial L^{(t)}}{\partial \mathbf{W}_{hh}} &= \left[\left(\mathbf{W}_{ph}^T \cdot (\tilde{\mathbf{y}}^{(t)} - \mathbf{y}^{(t)}) \right) \odot \left(\mathbf{1} - (\mathbf{h}^{(t)})^2 \right) \right] \cdot \left((\mathbf{h}_k^{(t-1)})^T + \mathbf{W}_{hh} \frac{\partial \mathbf{h}^{(t-1)}}{\partial (\mathbf{W}_{hh})} \right)
\end{aligned}
$$

As you can see, last expression depends on the previous hidden state and the matrix $\mathbf{W}_{hh}$, so to calculate the derivative at timestep t, we would have to multiply matrix $\mathbf{W}_{hh}$ by itself $t-1$ times, which might lead to either vanishing or exploding gradient as $t \to \infty$ depending on whether spectral radius of $\mathbf{W}_{hh}$ is < 1 or > 1.

Question 1.2

Implementation provided in $vanilla\_rnn.py$.

Question 1.3

Note that to plot these accuracies, we were using an average accuracy over last 100 steps. We assume that convergence has been reached if accuracy has stayed around the same value for 100 steps.

Question 1.4

To outline the main benefits of adaptive learning rate methods like Adam and RMSprop, we should first list the main issues encountered by Vanilla SGD.
Main drawbacks of Vanilla SGD are:

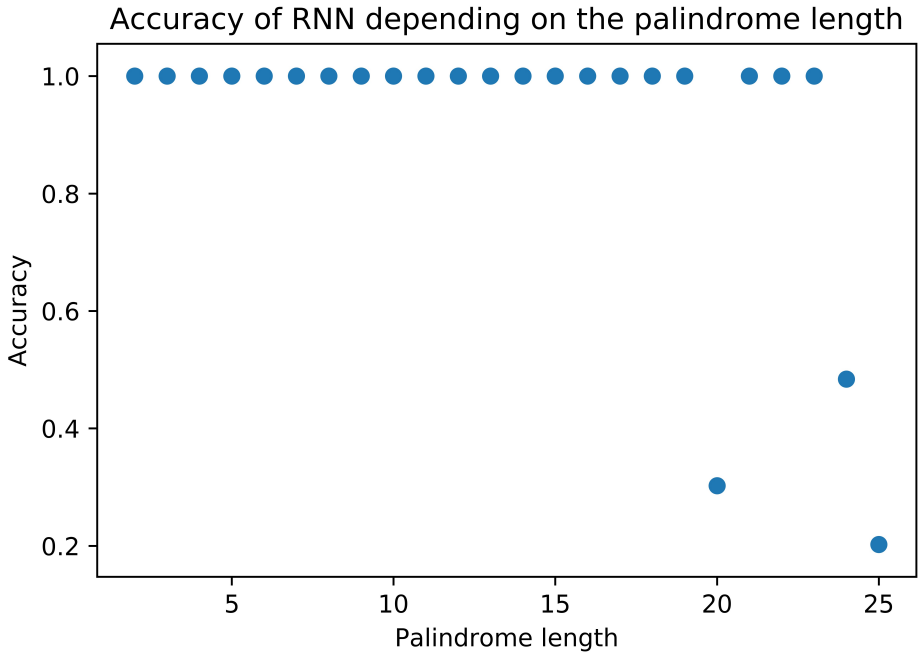

Figure 1: Vanilla RNN accuracy with respect to palindrome length

1. Learning rate should be specified beforehand. If it is too small, convergence might take too long and you might get stuck in a local minimum, while, if it is too large, loss might diverge or keep fluctuating around the minimum.

2. To fix previously mentioned issue, learning rate scheduler can be used, which would gradually reduce learning rate based on the specified condition. However, this schedule has to be specified beforehand and therefore can't adapt to the data. Also, due to the fact that SGD is not adapting to the data, it doesn't preserve long-term dependencies which are implicitly contained in the data.

3. SGD is very prone to getting stuck in suboptimal local minima. It has been argued, that this happens due to reaching saddle points as they are often surrounded by a plateau which means that gradients are close to zero in all directions.

Note, however, that multiple papers [2] [1] have suggested that SGD optimizer can find better optimum compared to other optimizers especially when using adaptive learning rate and momentum tuning [3].
The same ideas are also utilized by other optimizers.

1. Using momentum we avoid the last issue, because momentum increases the updates of parameters as long as gradient is moving in the same direction. This way we can dampen oscillations which can emerge while trying to navigate ravines, which often happen near local optimum. Generally, momentum can be thought of as momentum of a ball which has been pushed down a hill.

2. Using adaptive learning we are able to better navigate loss surface as we can choose learning rate based on the previous gradients. This way we can avoid the first two issues. For example, if loss surface is too noisy, we will see large gradients and an algorithm will make updates smaller. If the gradients are small, we are stuck in flat loss surface ravine and gradient updates will become more aggressive. Thus, algorithms using this technique can reach faster convergence and find more optimal minima.

1.2 Long-Short Term Network (LSTM) in PyTorch

Question 1.5 a)

1. input modulation gate $g^{(t)}$ and input gate $i^{(t)}$ are responsible for deciding which information from the current input is important and therefore should be added to the new memory. Both gates take takes current input and hidden state from the previous step and, after multiplying them by weight matrices and adding bias, apply activation function to them - $tanh$ in case of input modulation gate and sigmoid in case of an input gate. $tanh$ non-linearity pushes values of the current cell state to be between -1 and 1, while sigmoid outputs a vector of numbers between 0 and 1. These will be multiplied element-wise during the computation of current cell state. Sigmoid will thus define how much of the modulated input will be added to the current cell state.

2. forget gate $f^{(t)}$ is responsible for forgetting information from the previous steps. It takes current input and hidden state from the previous step and, after multiplying them by weight matrices and adding bias, applies sigmoid. Applying sigmoid results in a vector of numbers between 0 and 1, which, after element-wise multiplication with the cell state from the previous timestep, defines how much of a cell state is preserved. If it is 0, previous cell state will be discarded completely, while, if it is 1, it will be preserved entirely.

3. output gate $o^{(t)}$ is responsible for deciding how much of the current cell state will be outputted. It does operations similar to forget gate $g^{(t)}$ also applying sigmoid, output of which will then be element-wise multiplied with the $tanh$ of the current cell state (which already includes information from the gates $g^{(t)}, i^{(t)}$ and $f^{(t)}$ as well as information from the previous cell state $c^{(t-1)}$). $tanh$ non-linearity pushes values of the current cell state to be between -1 and 1 and then output gate vector defines how much of the current cell state will be passed to the output.

The main reason to use $tanh$ as an activation function is that it helps to fix the vanishing and exploding gradient problem by sustaining for a long time before changing towards 0 and not allowing to go above 1. Also, it allows gradient to stay zero-centered while also allowing for negative values unlike sigmoid.

Question 1.5 b)

Using that d is the feature dimensionality and n is the hidden state dimensionality, we can calculate number of trainable parameters as following:

$$4 \cdot dn \text{ - for input-to-hidden matrices}$$
$$4 \cdot n^2 \text{ - for hidden-to-hidden matrices}$$
$$4 \cdot n \text{ - for biases}$$

Therefore, final equation looks as following: $4n(d + n + 1)$

Technically, linear layer that is applied to the output is not part of the LSTM cell, however, if we include it, final equation will change to following (using that c is the output dimmensionality): $4n(d + n + 1) + c(n + 1)$

Question 1.5

Note that to plot these accuracies, we were using an average accuracy over last 100 steps. We assume that convergence has been reached if accuracy has stayed around the same value for 100 steps.

If you compare this plot to the one of RNN, you can easily see that LSTM model performs much better. While RNN's accuracy already starts dropping around length 17, LSTM manages to reach accuracy of 1.0 up to length 25. This is expected as LSTM can memorize much better than RNN and also doesn't have issues of exploding or vanishing gradient.

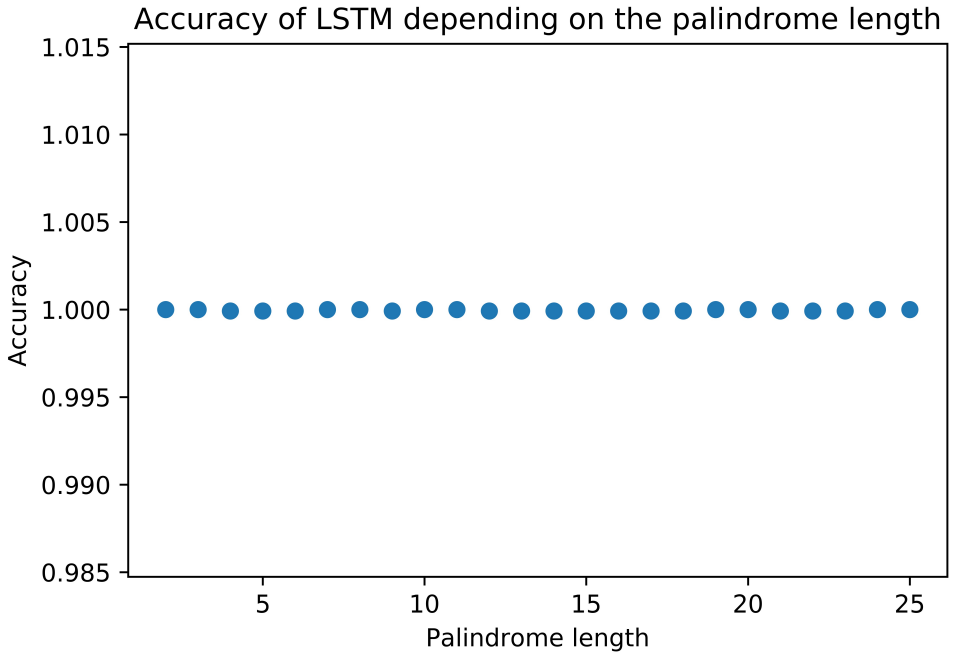

Figure 2: LSTM accuracy with respect to palindrome length

2 Modified LSTM Cell

Question 2.1

All of my following plots are based on the assumption that the equation specified is wrong in that it should contain equal sign instead of $\leq$ and $\geq$. The reason for that would be that otherwise we will have single zero point where $\phi_t = \frac{1}{2}r_{on}$. Corrected equation will look as following (changes are marked with red):

$$\begin{cases} \frac{2\phi^{(t)}}{r_{on}} & \text{for } \phi^{(t)} \leq \frac{1}{2}r_{on}, \\ 2 - \frac{2\phi^{(t)}}{r_{on}} & \text{for } \frac{1}{2}r_{on} < \phi^{(t)} \leq r_{on}, \\ 0 & \text{otherwise} \end{cases}$$

With this equation, we get following plots: fig. 3; fig. 4; fig. 5. Effect of each of the parameters will be further explained in section 2.

Question 2.2

We can split behaviour of the gate into three phases:

1. In this phase, gate is open and k_t rises from 0 to 1. In this phase, amount of information preserved from the previous states will be decreasing, while gradually more information will be taken from the current state.
2. In this phase, gate is open, but k_t decreases from 1 to 0. In this phase, amount of information preserved from the previous states will be increasing, while gradually less information will be taken from the current state.
3. In this phase, gate is closed and only previous state is preserved.

This new behaviour can be beneficial in multiple different cases. However, as gate is created with modelling oscillations in mind, the main purpose that it can be used for is learning patterns, which can occur with some fixed periodicity. Due to the third phase, when gate is closed, it might be able to

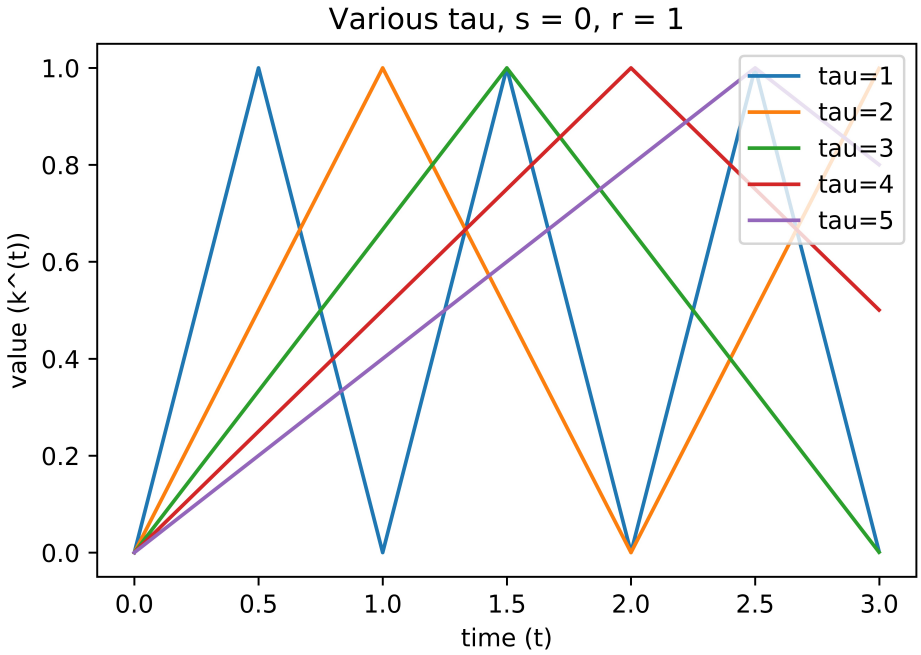

Figure 3: Plot of values of $k^{(t)}$ over time for different τ

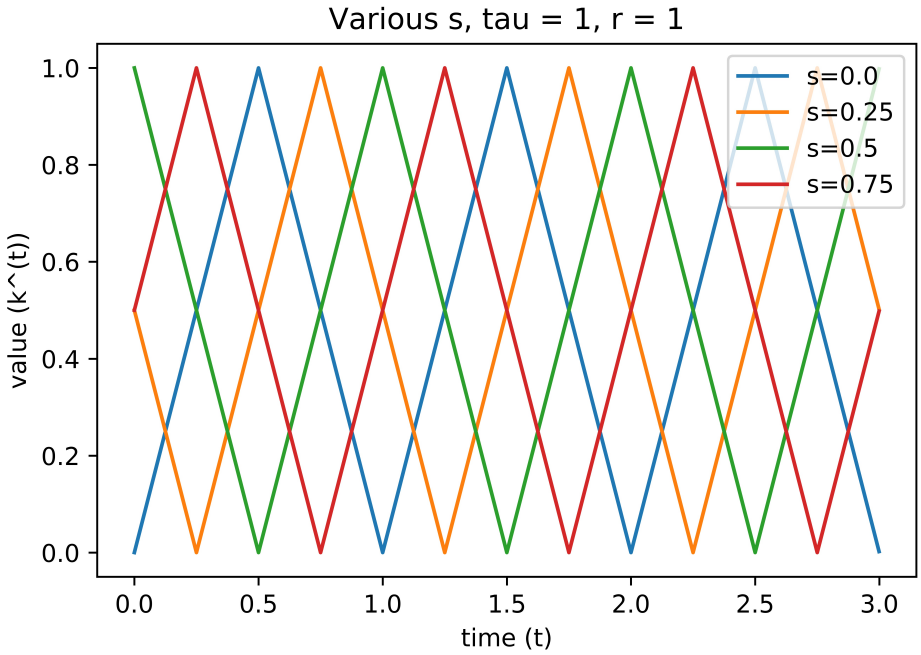

Figure 4: Plot of values of $k^{(t)}$ over time for different s

capture repeating patterns from very long sequences (as while there is no pattern, gate can learn to be closed until it occurs again). Also, this version of LSTM can perform similarly to spiking neural networks thus performing with higher biological plausibility.

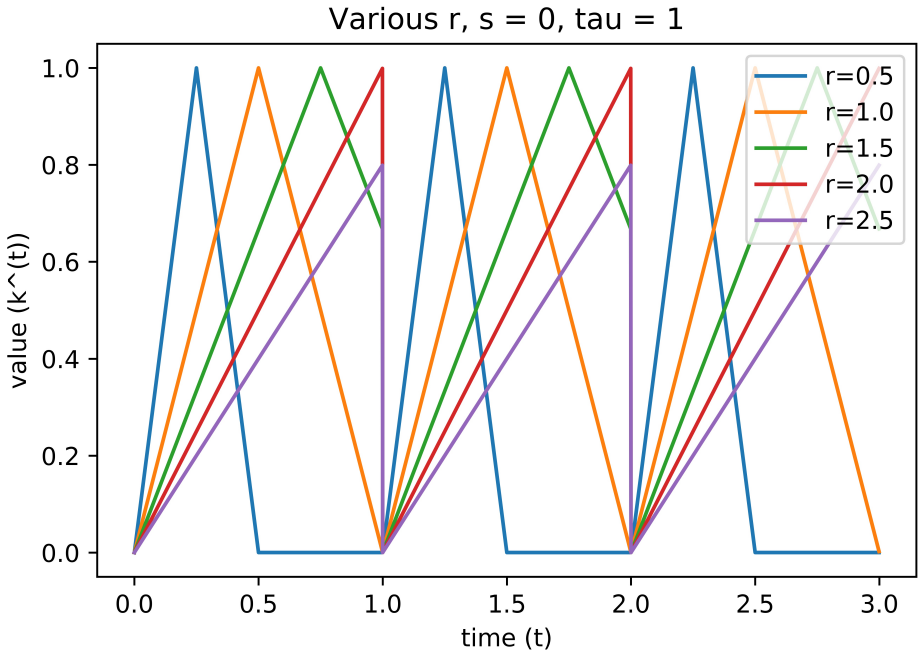

Figure 5: Plot of values of $k^{(t)}$ over time for different r_{on}

Question 2.3

As additional gate allows for oscillations, it will make sense to make analogy for different parameters of the gate and parameters defining the oscillation.

- τ defines period of the oscillations, so, time between the start of the oscillation and the start of the next oscillation. You can see effect of it in fig. 3.
- r_{on} defines the ratio between the duration of the "open" phase and the full period. You can see effect of it in fig. 4.
- s defines phase in oscillations, so, the shift of the oscillation. You can see effect of it in fig. 5.

As all of these parameters are differentiable, they can be learned from the data.

3 Recurrent Nets as Generative Model

*Question 3.1 For this task, we have implemented two-layer LSTM for language modelling task. Results were generated on a network, that takes sequence of length 30 of one-hot encoded character vectors as input, feeds it into 128-dimensional two-layered LSTM which is followed by a linear layer. We were using RMSProp optimizer with learning rate of $2e-3$, learning rate decay of 0.96, which updates learning rate every 5000 steps, weight decay of 0.01.

We have trained our model on *Vanity Fair* by *William Makepeace Thackeray*. Before training the model, we have removed all non-ASCII characters from the text not to deal with post-processing of Unicode characters.

During training, we are initializing hidden and cell state with zero vectors. During generation phase, we have used both greedy samping and random sampling from categorical distribution with temperatures of 0.5, 1.0, 2.0. We have been running the model for 70500 steps. When there is no input to complete, the generation will start by sampling random character from the vocabulary. You can see results in table 1. Randomly generated symbols were highlighted.

It is important to understand that generation quality highly depends on the corpus being used. In our case, greedy sampling and random sampling with temperature of 0.5 (which is the closest to greedy among the ones we have tried), works best. We can see, that temperature of 1.0 and 2.0 makes the generation too random.

In general, it is very interesting to observe how over time model learns dependencies between characters and while in the first step, it was generating complete nonsense, over time it learnt not only dependencies between characters, but also between words as LSTM allows to capture longer-term dependencies.

Table 1: Generated sentence for different sampling parameters

	T=0.5	T=1.0	T=2.0	Greedy
0	**p**cY&bSVTTkQS4 and?SZ#* o! j;.A	**#**jx!%dYikK.vXonWIc(Qw"@_AyeOwX	**L**l7 0ydu;5l4P2EpX24C9FS 68ZrWR	**I**mm
17600	**a** comprinklest fellows of her	**v**ed and discovered herse. His	("mEstfanciewhaply K\nHIPIO*, L	**g**ht the state of the state of
35200	\'s brandy-treaction.,"He had	**U**nce\nin him home and going, th	**S**EDee?" phere evendher\nPz8M;.,	**Z** and the company and the comp
52800	**D**obbin should be a man of her	\'s long money," reC.B.n that t	**.**\nGazipaG:-book," we us: \n\nAzl	_ and the presence of the pres
70400	**l**ingted the depresent travell	**y** must have dreads everything	**R**NHCTC!\nM9ETLD! gher.",DT2per	**d** the country and the country

Table 2: Generated sentence for different sampling parameters and different starting sentence

	T=0.5	T=1.0
poor Miss Birche	poor Miss Birche and Miss Crawley at the grave o	poor Miss Birch is most aress Sedley of these
Lyapunov stability is a very mild	Lyapunov stability is a very mild Heart the like of full as bec	Lyapunov stability is a very mild Fareard-shoods out are \n neptio
Gucci Gang	Gucci Gang "I gone for the boots and in	Gucci Gang makinged and lotees.,I famou
Duke Bolkonsky was sitting	Duke Bolkonsky was sitting any not herself sours as the	Duke Bolkonsky was sitting He hair excrapuch.,She women
	T=2.0	Greedy
poor Miss Birche	poor Miss Birch Vate. \ n :GeterEQfurly: laky vam	poor Miss Birch I had been a word and the com
Lyapunov stability is a very mild	Lyapunov stability is a very mild uzam antonc,I'an:":esmeonf	Lyapunov stability is a very mild I had been a word and the com
Gucci Gang	Gucci Gang affyy–a REI8'r quieyrebely	Gucci Gang I had been a word and the com
Duke Bolkonsky was sitting	Duke Bolkonsky was sitting So-cool fose echled GH(. youn	Duke Bolkonsky was sitting I had been a word and the com

*Bonus Question 2.3

During continuing the sentences, we have noticed that now, unlike during training itself, just greedy sampling was generating the same ending, while the best result was produced by random sampling with temperature of 0.5 again, as you can see in table 2.

We can notice, that for this case, results were pretty good. They were practically all gramatically correct and also could really recognize the dependencies i.e. Lyapunov stability is a very mild *heart the like of full*. It is interesting to observe also how model can handle absolutely new words with some kind of meaning i.e. continuing Gucci Gang. It would also be interesting to train the model on a mixed corpus with multiple language modes and observing how it behaves.

References

[1] Samuel L. Smith and Quoc V. Le. A bayesian perspective on generalization and stochastic gradient descent. *CoRR*, abs/1710.06451, 2017.

[2] Chen Xing, Devansh Arpit, Christos Tsirigotis, and Yoshua Bengio. A walk with sgd. *CoRR*, abs/1802.08770, 2018.

[3] Jian Zhang, Ioannis Mitliagkas, and Christopher Ré. Yellowfin and the art of momentum tuning. *arXiv preprint arXiv:1706.03471*, 2017.