Assignment 2: Recurrent Neural Networks and Graph Neural Networks

Alexandra Lindt
alex.lindt@protonmail.com

1 Vanilla RNN versus LSTM

1.2 Vanilla RNN in PyTorch

Question 1.1

The following functions will be used in further computations:

$$diagonal(x) = \begin{bmatrix} x_1 & 0 & \dots & 0 \\ 0 & x_2 & \dots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ 0 & \dots & 0 & x_n \end{bmatrix} \quad \text{with } x \in \mathbb{R}^n \text{ and } diagonal(x) \in \mathbb{R}^{n \times n}$$

Also, please note that for a vector $x \in \mathbb{R}^n$, $x^{(T)} \in \mathbb{R}^n$ denotes the vector at timestep T, while $x^T \in \mathbb{R}^{1 \times n}$ denotes the vector's transpose.

(1)

$$\begin{aligned} \frac{\partial \mathcal{L}^{(T)}}{\partial W_{ph}} &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{y}^{(T)}} \frac{\partial \hat{y}^{(T)}}{\partial W_{ph}} \\ &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{y}^{(T)}} \frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} \frac{\partial p^{(T)}}{\partial W_{ph}} \\ &= -\Big(\frac{y^{(T)}}{\hat{y}^{(T)}}\Big)^T \cdot \Big(diagonal(\hat{y}^{(T)}) - \hat{y}^{(T)}(\hat{y}^{(T)})^T\Big) \cdot \begin{bmatrix} \begin{bmatrix} h_1^{(T)} & \dots & h_M^{(T)} \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ h_1^{(T)} & \dots & h_M^{(T)} \end{bmatrix} \end{bmatrix} \\ &= \Big(- {y^{(T)}}^T + {y^{(T)}}^T \mathbb{1}_K (\hat{y}^{(T)})^T\Big) \cdot \begin{bmatrix} \begin{bmatrix} h_1^{(T)} & \dots & h_M^{(T)} \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ h_1^{(T)} & \dots & h_M^{(T)} \end{bmatrix} \end{bmatrix} \end{aligned}$$

$$
\begin{aligned}
&= \left(\hat{y}^{(T)^T} - y^{(T)^T}\right) \cdot \begin{bmatrix} \begin{bmatrix} h_1^{(T)} & \dots & h_M^{(T)} \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ h_1^{(T)} & \dots & h_M^{(T)} \end{bmatrix} \end{bmatrix} \\
&= \left[\left(\hat{y}^{(T)} - y^{(T)}\right) \cdot h^{(T)^T}\right] \in \mathbb{R}^{1 \times K \times M}
\end{aligned}
$$

where the following was used:

(1.1)

$$
\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{y}^{(T)}} = \begin{bmatrix} \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{y}_1^{(T)}} & \dots & \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{y}_K^{(T)}} \end{bmatrix} = \begin{bmatrix} -\frac{y_1^{(T)}}{\partial \hat{y}_1^{(T)}} & \dots & -\frac{y_i^{(T)}}{\partial \hat{y}_K^{(T)}} \end{bmatrix} = -\left(\frac{y^{(T)}}{\hat{y}^{(T)}}\right)^T
$$

(1.2)

$$
\begin{aligned}
\frac{\partial \hat{y}_i^{(T)}}{\partial p_j^{(T)}} &= \frac{\partial}{\partial p_j^{(T)}} \frac{\exp(p_i^{(T)})}{\sum_{n=1}^K \exp(p_n^{(T)})} \\
&= \frac{\partial}{\partial p_j^{(T)}} \left(\exp(p_i^{(T)}) \frac{1}{\sum_{n=1}^K \exp(p_n^{(T)})} \right) \\
&= \frac{\partial \exp(p_i^{(T)})}{\partial p_j^{(T)}} \frac{1}{\sum_{n=1}^K \exp(p_n^{(T)})} + \exp(p_i^{(T)}) \frac{\partial}{\partial p_j^{(T)}} \left(\sum_{n=1}^K \exp(p_n^{(T)}) \right)^{-1} \\
&= \delta_{ij} \frac{\exp(p_i^{(T)})}{\sum_{n=1}^K \exp(p_n^{(T)})} - \exp(p_i^{(T)}) \frac{\exp(p_j^{(T)})}{\left(\sum_{n=1}^K \exp(p_n^{(T)}) \right)^2} \\
&= \delta_{ij} \hat{y}_i^{(T)} - \hat{y}_i^{(T)} \hat{y}_j^{(T)} \\
&= \hat{y}_i^{(T)} (\delta_{ij} - \hat{y}_j^{(T)})
\end{aligned}
$$

$$
\implies \quad \frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} = \begin{bmatrix} \frac{\partial \hat{y}_1^{(T)}}{\partial p_1^{(T)}} & \dots & \frac{\partial \hat{y}_1^{(T)}}{\partial p_K^{(T)}} \\ \vdots & \ddots & \vdots \\ \frac{\hat{y}_K^{(T)}}{p_1^{(T)}} & \dots & \frac{\hat{y}_K^{(T)}}{p_K^{(T)}} \end{bmatrix} = \begin{bmatrix} \hat{y}_1^{(T)}(1 - \hat{y}_1^{(T)}) & -\hat{y}_1^{(T)} \hat{y}_2^{(T)} & \dots & -\hat{y}_1^{(T)} \hat{y}_K^{(T)} \\ -\hat{y}_2^{(T)} \hat{y}_1^{(T)} & \hat{y}_2^{(T)}(1 - \hat{y}_2^{(T)}) & \dots & \vdots \\ \vdots & \vdots & \ddots & \vdots \\ -\hat{y}_K^{(T)} \hat{y}_1^{(T)} & -\hat{y}_K^{(T)} \hat{y}_2^{(T)} & \dots & \hat{y}_K^{(T)}(1 - \hat{y}_K^{(T)}) \end{bmatrix}
$$

$$
= diagonal(\hat{y}^{(T)}) - \hat{y}^{(T)} (\hat{y}^{(T)})^T
$$

(1.3) Assumed $h \in \mathbb{R}^M$ and therefore $W_{ph} \in \mathbb{R}^{K \times M}$:

$$
\frac{\partial p_i^{(T)}}{\partial W_{ph_{jk}}} = \frac{\partial}{\partial W_{ph_{jk}}} \left(W_{ph_{i,:}} \cdot h^{(T)} + b_{p_i} \right) = \begin{cases} h_k^{(T)} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}
$$

$$\implies \frac{\partial p^{(T)}}{\partial W_{ph}} = \begin{bmatrix} \begin{bmatrix} \frac{\partial p_1^{(T)}}{\partial W_{ph\,11}} & \cdots & \frac{\partial p_1^{(T)}}{\partial W_{ph\,1M}} \\ \vdots & \ddots & \vdots \\ \frac{\partial p_1^{(T)}}{\partial W_{ph\,K1}} & \cdots & \frac{\partial p_1^{(T)}}{\partial W_{ph\,KM}} \end{bmatrix} \\ \vdots \\ \begin{bmatrix} \frac{\partial p_K^{(T)}}{\partial W_{ph\,11}} & \cdots & \frac{\partial p_K^{(T)}}{\partial W_{ph\,1M}} \\ \vdots & \ddots & \vdots \\ \frac{\partial p_K^{(T)}}{\partial W_{ph\,K1}} & \cdots & \frac{\partial p_K^{(T)}}{\partial W_{ph\,KM}} \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} h_1^{(T)} & \cdots & h_M^{(T)} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ h_1^{(T)} & \cdots & h_M^{(T)} \end{bmatrix} \end{bmatrix} \in \mathbb{R}^{K \times K \times M}$$

(2)

$$\frac{\partial \mathcal{L}^{(T)}}{\partial W_{hh}} = \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{y}^{(T)}} \frac{\partial \hat{y}^{(T)}}{\partial p^{(T)}} \frac{\partial p^{(T)}}{\partial h^{(T)}} \frac{\partial h^{(T)}}{\partial W_{hh}}$$

$$= \left(\hat{y}^{(T)} - y^{(T)}\right) \cdot W_{ph} \cdot \left(I_M - diagonal(h^{(T)})^2\right) \cdot \left(\begin{bmatrix} \begin{bmatrix} h_1^{(T-1)} & \cdots & h_M^{(T-1)} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ h_1^{(T-1)} & \cdots & h_M^{(T-1)} \end{bmatrix} \end{bmatrix} + \frac{\partial h^{(T-1)}}{\partial W_{hh}} \cdot W_{hh} \right)$$

where the gradients from (1.1), (1.2) and the following were used:

(2.1)

$$\frac{\partial p^{(T)}}{\partial h^{(T)}} = \frac{\partial}{\partial h^{(T)}} \left(W_{ph} h^{(T)} + b_p\right) = W_{ph}$$

(2.2)

We denote

$$h_{inner}^{(T)} = W_{hx} x^{(T)} + W_{hh} h^{(T-1)} + b_h \qquad \text{and} \qquad h^{(T)} = tanh(h_{inner}^{(T)})$$

Therefore:

$$\frac{\partial h^{(T)}}{\partial W_{hh}} = \frac{\partial h^{(T)}}{h_{inner}^{(T)}} \frac{h_{inner}^{(T)}}{\partial W_{hh}}$$

$$= \left(I_M - diagonal(h^{(T)})^2\right) \cdot \left(\begin{bmatrix} \begin{bmatrix} h_1^{(T-1)} & \cdots & h_M^{(T-1)} \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ h_1^{(T-1)} & \cdots & h_M^{(T-1)} \end{bmatrix} \end{bmatrix} + \frac{\partial h^{(T-1)}}{\partial W_{hh}} \cdot W_{hh} \right)$$

where the following was used :

$$\frac{\partial h^{(T)}}{\partial h^{(T)}_{inner}} = \frac{\partial}{\partial h^{(T)}_{inner}} tanh(h^{(T)}_{inner}) = \begin{bmatrix} \frac{\partial tanh(h^{(T)}_{inner1})}{\partial h^{(T)}_{inner1}} & \dots & \frac{\partial tanh(h^{(T)}_{inner1})}{\partial h^{(T)}_{innerM}} \\ \vdots & \ddots & \vdots \\ \frac{\partial tanh(h^{(T)}_{innerM})}{\partial h^{(T)}_{inner1}} & \dots & \frac{\partial tanh(h^{(T)}_{innerM})}{\partial h^{(T)}_{innerM}} \end{bmatrix}$$

$$= \begin{bmatrix} 1 - tanh(h^{(T)}_{inner1})^2 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 - tanh(h^{(T)}_{innerM})^2 \end{bmatrix} = I_M - diagonal(tanh(h^{(T)}_{inner})^2)$$

$$= I_M - diagonal(h^{(T)})^2$$

and

$$\frac{\partial h^{(T)}_{inner}}{\partial W_{hh}} = \frac{\partial}{\partial W_{hh}} \left(W_{hx} x^{(T)} + W_{hh} h^{(T-1)} + b_h \right) = \frac{\partial (W_{hh} h^{(T-1)})}{\partial W_{hh}}$$

$$= \frac{\partial W_{hh}}{\partial W_{hh}} \cdot h^{(T-1)} + \frac{\partial h^{(T-1)}}{\partial W_{hh}} \cdot W_{hh}$$

$$= \begin{bmatrix} \begin{bmatrix} h^{(T-1)}_1 & \dots & h^{(T-1)}_M \\ \vdots & \ddots & \vdots \\ 0 & \dots & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & \dots & 0 \\ \vdots & \ddots & \vdots \\ h^{(T-1)}_1 & \dots & h^{(T-1)}_M \end{bmatrix} \end{bmatrix} + \frac{\partial h^{(T-1)}}{\partial W_{hh}} \cdot W_{hh}$$

As we can see, the second derivative $\frac{\mathcal{L}^{(T)}}{\partial W_{hh}}$ is different to the first derivative $\frac{\mathcal{L}^{(T)}}{\partial W_{ph}}$ in the sense that it is recursively defined over the gradients of all past hidden states $\frac{\partial h^{(T-1)}}{\partial W_{hh}}$ $\forall t \in 1, ..., T$. This happens because each derivative of $\frac{\partial h^{(T)}_{inner}}{\partial W_{hh}}$ at a timestep T includes a derivative of $\frac{\partial h^{(T-1)}}{\partial W_{hh}}$ of the previous timestep $T-1$.

Due to the overall formula for $\frac{\mathcal{L}^{(T)}}{\partial W_{hh}}$, it can easily happen that the gradients corresponding to earlier timesteps get either weighted weaker or stronger than the gradients of later timesteps. This weighting adds up over multiple time steps that are processed by the RNN and lead to wrong outputs of the network. The phenomenon is known as the *vanishing gradient* or the *exploding gradient* problem, dependent on the gradients of earlier timesteps being weighted less and less (gradients *vanish*) or more and more (gradients *explode*) with more processed timesteps. A widely used solution to these problems is to restrict the values of the gradients to lie within a certain treshold.

Question 1.2

Please see the files `vanilla_rnn.py` and `train.py` in `./part1`for the detailed implementation. In the code comments, it is asked about the purpose of the function call

`torch.nn.utils.clip_grad_norm(model.parameters(), max_norm=config.max_norm).`

As can be seen in the pytorch documentation of the `torch.nn.utils.clip_grad_norm`, the function restricts all individual gradient values of the model parameters to a given maximum value (here: `config.max_norm` with default value 10). This is done to prevent the gradients of the model from getting too large, i.e. to prevent the gradient from *exploding*. As already discussed above, this is

necessary because we are working with an LSTM model and the parameter gradients accumulate themselves over multiple time steps causing too big updates of the gradients as well as overflow problems.

Question 1.3

The accuracy versus palindrome length is reported in figure 1 for RNN and LSTM respectively. As expected the LSTM performed better than the RNN on input sequences $\in [15, 30]$. This makes sense, as it is the more complex model. An interesting difference can also be observed for a palindrome length > 30. Here, it was observed that the accuracy that the RNN achieves varies a lot between runs with similar palindrome lengths, while the performance of the LSTM (although low because of a lack of parameter tuning) remains similar for similar palindrome lengths. This behavior was also observed when training the RNN and LSTM multiple times for the same palindrome length.

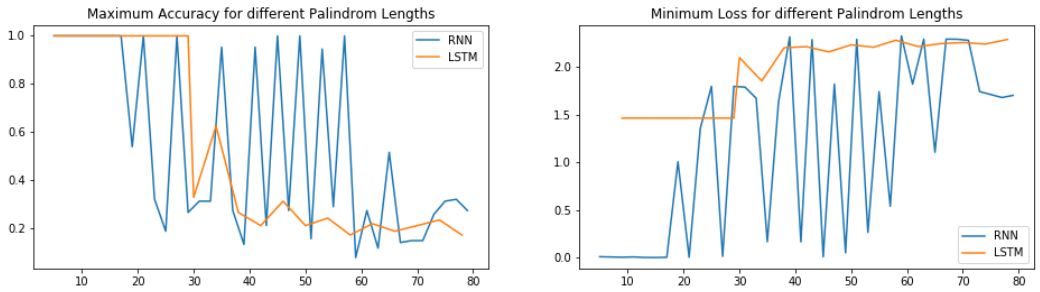

Figure 1: Accuracy and Loss of the trained RNN and LSTM for Palindrome Lengths from 5 to 80.

Question 1.4

To answer this question, I follow the explainations given in chapter 8 of Goodfellow et al. [2016].

Although Vanilla SGD is a popular optimization method, it comes with some drawbacks. SGD determines the next step in the parameter space solely by the gradient of its current position. This is especially unfavourable for the optimization of DNNs or RNNs, as such networks often contain sharp nonlinearities in the parameter space which cause very high derivatives in the respectives regions . Such high derivatives lead to the SGD algorithm making huge steps in parameter space, which may push the model parameters into a very wrong direction. Another problem that occurs with SGD is that the reaching of a minimum can be very slow, because the sampling of training examples for each step introduces a certain randomness to each update and no overall 'direction trend' is analyzed. Also, once a minimum is reached, the algorithm may not converge right to it but jump around it instead due to the sampling-introduced randomness. These drawbacks are adressed by the **momentum** algorithm, which estimates the next step from an accumulation of exponentially decaying average of past gradients and the current gradient. This means concretely that if the algorithm moves for example three steps into the same direction, the fourth step into this direction is going to be a bigger one. Therefore, the momentum algorithm moves faster than the SGD algorithm. Further, through the accumulation of past steps and exponential decay, it avoids to jump around a minima and converges right to it. One could also say that the momentum algorithm adresses the problem of the learning rate being such a crucial parameter for the success of SGD. However, it does so by introducing another hyperparameter for the momentum term. Another way to adress the learning rate problem is to use seperate **learning rates** for each parameter and **adapt** these learning rates while training.

The RSMProb algorithm is an optimization algorithm that adapts the learning rates for each model parameter individually. Concretely, the learning rates are scaled inversely proportional to an exponentially weighted moving average over the historical values of the gradient. This leads to RSMProb not considering large gradients from extreme past steps but only the most recent ones. The Adam optimizer combines RMSProp optimization with momentum. Concretely, the momentum is applied to the rescaled gradients of RMSProp to ensure an effective convergence.

1.3 Long-Short Term Network (LSTM) in PyTorch

Question 1.5

(a)

Forget Gate $f^{(t)} = \sigma(W_{fx}x^{(t)} + W_{fh}h^{(t-1)} + b_f)$
The forget gate determines which information of the cell-state $c^{(t-1)}$ is going to be *forgotten*, i.e. replaced with new information. It makes use of a sigmoid activation in order to output a vector of the same size as the cell state with values $\in [0, 1]$. This output vector $f^{(t)}$ denotes for each entry of $c^{(t-1)}$ how 'keepworthy' it is, i.e. how much its value should influence the updated cell state $c^{(t)}$. A value of 1 means the value for the respective entry will be fully kept and a value of 0 means it will be fully forgotten. When the new cell state $c^{(t)}$ is determined, $f^{(t)}$ is used to weight the entries of previous cell state by being element-wise multiplied with $c^{(t-1)}$.

Input Gate $i^{(t)} = \sigma(W_{ix}x^{(t)} + W_{ih}h^{(t-1)} + b_i)$
The input gate determines which values of the cell state are going to be updated. Similar to the forget gate, it outputs a vector of the same size as the cell state with values $\in [0, 1]$ (sigmoid activation function!). This vector is then used when the new cell state $c^{(t)}$ is determined to weight the entries of the candidate vector $g^{(t)}$ against the values of $c^{(t-1)}$ that were kept. By having only values $\in [0, 1]$ it is made sure that the

Input Modulation Gate $g^{(t)} = tanh(W_{gx}x^{(t)} + W_{gh}h^{(t-1)} + b_g)$
The input modulation gate produces a so-called *candidate vector*, i.e. a vector of candidate values for the updated hidden cell-state $c^{(t)}$. Since the cell state should be able to have positive and negative values, the gate makes use of a $tanh$ activation that causes all of its values to be $\in [-1, 1]$.

Output Gate $o^{(t)} = \sigma(W_{ox}x^{(t)} + W_{oh}h^{(t-1)} + b_o)$
The final output gate determines which parts of the updated cell state $c^{(t)}$ are going to be passed to the hidden state $h^{(t)}$ and then outputted. Therefore, the vector o has values in $[0, 1]$, which denote for each entry of $c^{(t)}$ how 'outputworthy' it is.

Overall, the activation functions are used to keep the values of the cell state and the hidden state low in their absolute value. If these functions would not be used, the values of cell and hidden state could massiviely increase or decrease over multiple time steps of input.

(b)

With a single input $x \in \mathbb{R}^d$, hidden unit $h \in \mathbb{R}^n$ and a single output $\hat{y} \in \mathbb{R}^c$, the LSTM would have

$$4 \cdot (n \cdot d + n \cdot n + n) + c - \cdot n + c$$

trainable weights. Note that neither the batch size used for training nor the length of the input sequence influence this number: Training with a batch size bigger than 1 simply means to update the same weights for multiple inputs at once. Further, the LSTM is applied to each input of an input sequence one after another, so the length of the input sequence does not increase the number of weights needed.

2 Recurrent Nets as Generative Model

Question 2.1

(a)

The implementation of the two-layer LSTM and its training process can be found in the folder `./part2`. As training data, I decided for Grimm's Fairytales. The LSTM was trained over about 800.000 timesteps, the loss and accuracy curves are given in figure 2. Since this figure reveals that the biggest steps in accuracy are made very early, I further include figure 3, which shows the development of loss and accuracy up to the 20.000th step. Figure 3 shows that the accuracy of the model is already

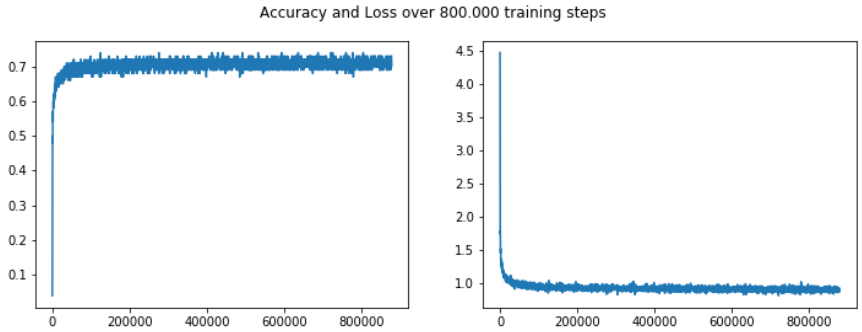

Figure 2: Accuracy and Loss of the two-layer LSTM network trained for predicting the next character given a sequence of characters.

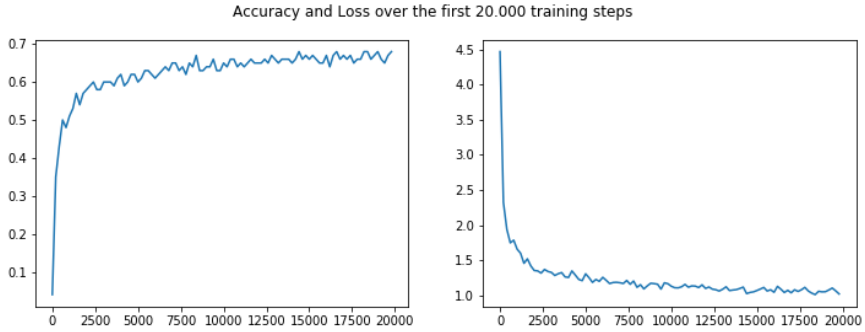

Figure 3: Accuracy and Loss of the two-layer LSTM network trained for predicting the next character given a sequence of characters.

close to 0.7 at timestep 20.000. However, it is important to notice that the accuracy only measures the capability of the network to predict the most probable char after a given sequence of chars and not its capability to generate overall meaningful sequences.

(b)

In order to see the development of the network's generating capabilities, the required five text samples for the step sizes are reported for the step sizes 1.000, 5.000, 20.000, 100.000, 200.000, 400.000, and 800.000. Table 1 and 2 list text samples of length 30 reported at the respective timesteps during training. For this subtask, we look at the first column of 1, which holds text samples for greedy sampling. As the training process evolves, we see a clear development in the sampled text snippets. At timestep 1.000, the model still produces very raw text samples, like '**E THE THE THE THE THE THE THE**' or '**o the word the word the word t**'. It is clear that the model considers 'the' and 'word' as the most logical following words to a lot of words/characters, presumably due to their frequent occurrence in the training text. At timestep 5.000, we can see that the model now has a new

Step	Greedy Sampling	Random Sampling
		T=0.5
1.000	'S the streat was the court and' 'E THE THE THE THE THE THE THE ' 'o the word the word the word t' 'very the word the word the wor' '5 and said: 'I will not were t'	'under father, and shuld soon t', 'y the princess and to him and ', 'the door all the starld and se', '@ a little to his how will gar', '[me and rear and ser fast in t'
5.000	'My mother said, 'I will give y', ' the search of the carter and ', 'quired and said, 'I will give ', 'K the carter said, 'I will giv', 'e was a golden castle was so f'	'at he said to his cown, and sa', 'quire a little shoes were saw ', 'he thought of the servant, 'ma', 'ng to her came to the pan, and', '–tree in the bodgon to his ne'
20.000	''I am not one of them and said', '#\n\nThe wolf was a great fear a', 'me to the beard. 'The door sha', '#\n\nThe wolf was a great fear a', 'Fox, where a number of the win'	'?' 'That is the time in the wi', 'e fisherman was about her, and', '$ her, and she was so fine emp', ''I have a sout fur werer sent ', 'y had a great noise that she a'
100.000	'3. If you can sell the spindl', 'x said, 'I will give you your ', 'Cat-skin again. But he did not', 'zed the window, and the prince', 'he was so beautiful that he wo'	'", one you are all the time?" ', '] and the third stir horse she', '\nhis bed.\n\nThe second time he ', 'Hans. 'Now, my good could fine', 'Gretel, made her life; they we'
200.000	'//pglaf.orge for the fire and ', 'D THE LION\n\nA merchant they to', 'l the miller's daughter again ', '] just and the soldier stood b', '% of the work that they might '	'Kywitt, Kywitt, what a beautif', 'very day the wedding was crowe', '7, JEGEF\n SWEETHEART ROLAN', '. If you are so for the littl', 'One day and watched with count'
400.000	'xe and the sexton from and scr', 'QUEEN BEE\n THE GOLDEN GOOS', '% on the time the second son s', 'me to the king's son might be ', '\n\nThe girl went on the shore a'	'when he had forgotten the thir', 'D METHEART RIBEGSA AND THE SHO', 'And the judge and cried out, '', '7, Gretel.' 'Good day, Hans. W', 'Many years.' 'How she is eat y'
800.000	'And the soldier was so beautif', ' the stove than me?' 'Oh, no!", 'g and said: 'What a prettied h', ''The next day the true princes' , '5. By my poor stomach of the '	'1.E.7 or obtain part of the co', 'or stay were come the side of ', 'xt morning the second prince s', 'I have something to eat. The d', '!' The cook heard this the rob'

Table 1: Text samples from the trained LSTM at different timesteps of training. Achieved through greedy sampling and random sampling with temperature 0.5.

Step	Random Sampling	
	T=1	T=2
1.000	'? As Gr eatered: and store it.', 'To Andaele, pit-her: ‘ and awr', 'R, queter old cat to a will bo', '-the bright or wing all therp\n', 'K the boit, and ligtt, and rec'	'XR\nPO!Q\n' ABU1,)'s b ut,\nhew,", '9Fotheroa“gyaldy; sqByl,.\n*. R', '7-lairsfse?'y\nbull\nshepike ver', "d\nsnangem: @ls, Stay.\nDukn; G', '-#ich,-and aqkpews;\nyou @rwbyk'
5.000	'Qrewn at Curden,' said\nHans we', 'Ha[l bring to Crap;\nto village', '“you\nshall even face. Then she', 'cksed and keses as you discond', 'Quire and get deneess that som'	'/Itevery\npardovidoz.6),\nset\nub', 'Dorngbed ckise, he brik for a ', 'girt–thirs! Coxter!' A ussy, ', 'Sfeens, emthe Fart\nthe sknif c', 'jourted,\n\nOway\nWiso fatly, man'
20.000	'ked\nher to charger: they heard', 'king went forth some fellow do', 'D MOCWOT FOT RIDECT THITS:\n M', '2#6"52.\n “Pull pleased that he', 'Zom him out, and threw it,\ntak'	'!'\n\nLina froen's whise, just,", 'x hUndessore,\ninstead get i', 'nd at that when\nsuch whidest h', '1“6. \nY)all)fal\nyether(bEa sle', '] stayil?\n\nWTF8 MEMIHHPARTTLET'
100.000	['\ufeffThus they all\nis\npicking. How', '4, Little Red-Cap\nbefall that ', 'Ut Snowdrop were fell took hol', 'Vell money.' Then\nthe third ni', 'Just and see your fruty\ncook, '	'ivip. When\nTom you, Kat!' joln', 'e.\n‘Ah,\nfeel imee\nwhom the\nkis', 'cEas pen\nin, near murderer.'\n\n', '1812 3GLoSE ner wished.\n\nNow h', 'ighbour\nwretch no\none wopped a'
200.000	0; you will soon find it is ri', '@p, and that been play, my fin', ' sought out\nof the wood. So sh', '!' said the\nother, ‘I am goost', 'When the whip wan all the tail'	'; the betrothed returned), can', 'grewooks and bagtor grodeciave', 'MES, EXCer CLEVER BROT, FUSE \n', 'just as he hallowed of\nbleasen', 'pared\naltainieves.' The little'
400.000	'!' The\ngion coaced all for him', ']\n\n\nSit pope that heard me.' ‘', 'ore is you shall have\nto tear ', '08 [EBook had.' The\ncat had ga', 'sweeping. The woolcing down, a'	'xcendinw, so\nmeanife hairy you', '\ncut; he went.'\nThen\nat\nsearch', 'Fech!'\n\nThis answered: ‘Coilin', 'u for?' The snos him, my dead,', '); but ontact esected; the mut'
800.000	'xt morning by\nthe fire than el', 'when you receive the thief; an', ']\n the juniper-tree. And y', 'X\n\n\nThen the twelve princess, ', '4\nand presently\nshe fell on th'	'ZElIND OR EL; TALEBIDI HaVe\n ', '501-ismaken!' The doctor\nwerk\n', 'You dancy\nbar, and did briguse', 'black sister the volacen was s', '$1\nthis que Trud if\nonly tappi'

Table 2: Text samples from the trained LSTM at different timesteps of training. Achieved through random sampling with temperature 1 and 2.

favored character sequence '**[..] I will give [..]**' that occurs in the majority of the greedy samples. This might be considered a development coming from the 'the word' phase. From timestep 20.000 onwards we observe much more mature text snipppets, whose quality cannot really be compared because of their short length. However, we can observe a very correct case of the usage of spoken language in the text in a sample from the model at 800.000 steps: '**the stove than me?' 'Oh, no!**''. Overall, it is to note that the network likes to predict another linebreak in case a single line break is given at various timesteps. This might be the case because in the book, there usually are two linebreaks after each paragraph and title of a single story. Further, the model is often seen to predict capital letters given a capital letter. This is probably because the story titles are written in all capital letters and captial letters do generally not apper too often in english language.

(c)

As can be observed in the tables 1 and 2, the model creates more random outputs the higher the temperature is. The temperature 0.5 seems to be a good balance between random and greedy sampling: It results still in proper language but also prevents the model to generate repetetive sequences. In contrast, the temperatures 1 and 2 seem to introduce to much randomness.

3 Graph Neural Networks

3.1 GCN Forward Layer

Question 3.1

(a)

The formula for the activations $H^{(l)}$ in layer l is given as

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)}) \ ,$$

where $W^{(l)}$ is the trainable weight matrix correspondig to layer l and σ is an activation function. The given equation can be interpreted as a regular fully connected layer consisting of a weight matrix $W^{(l)}$ that transforms the input and an activation function σ. As widey known, weight matrices and activation functions do not exploit structural information of their input. However, this happens when the activations $H^{(l)}$ from the previous layer are transformed with the matrix $\hat{A}$. This matrix is the normalized adjacency matrix of the underlying undirected graph with added self-connections for each node. Therefore, $\hat{A}_{ij}$ is zero exactly in those positions where there is no connection in the graph between node i and node j and $i \neq j$. Due to this property of $\hat{A}$, the product of $\hat{A}$ and $H^{(l)}$ can be rewritten as

$$\begin{aligned}
\hat{A}H^{(l-1)} &= \begin{bmatrix} \sum_i^N \hat{A}_{1,i}H_{i,1}^{(l)} & \dots & \sum_i^N \hat{A}_{1,i}H_{i,d_l}^{(l)} \\ \vdots & \ddots & \vdots \\ \sum_i^N \hat{A}_{N,i}H_{i,1}^{(l)} & \dots & \sum_i^N \hat{A}_{N,i}H_{i,d_l}^{(l)} \end{bmatrix} \\
&= \begin{bmatrix} \sum_{i \in v(1)} \hat{A}_{1,i}H_{i,1}^{(l)} & \dots & \sum_{i \in v(1)} \hat{A}_{1,i}H_{i,d_l}^{(l)} \\ \vdots & \ddots & \vdots \\ \sum_{i \in v(N)} \hat{A}_{N,i}H_{i,1}^{(l)} & \dots & \sum_{i \in v(N)} \hat{A}_{N,i}H_{i,d_l}^{(l)} \end{bmatrix}
\end{aligned}$$

where $v(i)$ is the set of all nodes conneted to node i. As we can see, the product $\hat{A}H^{(l)}$ is of dimensionality $N \times d_l$, just like the activations of the previous layer $H^{(l)}$. Since $H^{(l)}$ denotes the d_l-dimensional features of N nodes, multiplying $\hat{A}$ to $H^{(l)}$ can be interpreted as replacing each d_l-dimensional features in $H^{(l)}$ with a weighted sum of the features that the corresponding node is connected to. The resulting features combine information from themselves and the features of connected nodes. This process can also be interpreted as *message passing over the graph*, as the feature informations are passed from nodes to their neighbouring nodes.

(b)

As we have seen above, a single GCN layer propagates the information from the neighbouring nodes (i.e. nodes that are one hop away) to a node. Therefore, to propagate to every node's embedding the information from nodes three hops away in the graph, one would need at most three stacked GCN layers.

Question 3.2

As can be seen from the recent review on GNNs methods and applications by Zhou et al. [2018], GNNs find application in various areas, such as physics, chemistry/biology, text analysis or computer vision. A few concrete real-world applications are listed here:

Protein Interface Prediction
Since proteins function through interaction with other proteins, predicting the interfaces through which these interactions occur is a crucial task in the field of drug discovery and design. For this purpose, Fout et al. [2017] proposed a GNN trained on graph representations of 3D structures of proteins.

Text Classification
Several approaches have been published that encode documents or sentences as graphs of word or alternertively make use of the references of each document to create a graph of documents with semantic edges between them Zhou et al. [2018]. To give one concrete example: Peng et al. [2018]'s approach is to transform text documents to bag-of-words and then to graph of words representations, which are following used to train a deep GCNN model for classification.

Semantic Segmentation
Semantic Segmentation is a crucial task in computer vision. The goal is to assign a label to every pixel within an image in order to segment the image into semantically related regions. Liang et al. [2016] propose the Graph LSTM network, which models long-term dependencies as well as the spatial structure of pixels of an image. An input image is therefore represented as by a graph structure built from a distance-based *superpixel map*.

Question 3.3

(a)

RNNs are best suited for operating on sequential or time-series data. They gain memory over the time of processing and therefore can deal with long-term relations in their input data. Another big plus of RNNs is their computational efficiency due to using the same weights for each timestep or other building block of their input. Further, RNNs can produce outputs and/or process inputs of variable length. Due to these properties, RNNs are very well suited for processing text or spoken language, as they naturally come as a sequence of words or characters. RNNs are especially handy for text generation or machine translation tasks, where the length of the input and expected output can vary.

When comparing RNNs and GNNs it is important to notice that a RNN handles the input feature vectors in a specific order, while in contrast, there is no specific order of nodes in a graph. To present a graph in its completeness to an RNN, one would need to tranverse all possible orders of a node (Zhou et al. [2018]), which would of course be redundant and computationally expensive. GNNs, however, propagate over each node while ignoring the input order and therefore making the output invariant to it. They update the hidden states of nodes as a weighted sum of neighbouring states.

The advantage of choosing a GNN instead of a RNN is of course that structural information in the input data is considered. This might be beneficial for any input data that has a meaningful graph representation. Such data can have an obvious graph structure like for example proteins, social networks or physical systems (i.e. objects as nodes and relations as edges) (Zhou et al. [2018]), but also text and images (as Peng et al. [2018] and Liang et al. [2016] showed) can be represented as graphs. With the overall in mind, I would expect RNNs to outperform GNNs on timeseries data, because putting such data into a graph structure does not seem beneficial to me. I also expect text data to be more efficiently and better processed by an RNN, as ordering information is of great importance for understanding texts. However, some text data could also profit from being represented and processed as graph structure. Such a graph could for example be a tree representation of a

sentence that shows semantic information of the contained words or a bag-of-words based graph as proposed by Peng et al. [2018]. For images, I would expect a GNN more suitable. This is because image pixels do not necessarily have one specific order but can be represented as a graph (e.g. build up from superpixel-based map as seen in Liang et al. [2016]).

(b)

I could imagine a RNN structure that essentially is a GNN with inner "memory" weights. This network would get the states of the same graph over multiple timesteps as an input. An example application of such a model could be to monitor a graph-like structure, like for example a street system, for annormalities (i.e. accidents, traffic jams).

References

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Jie Zhou, Ganqu Cui, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, and Maosong Sun. Graph neural networks: A review of methods and applications. *arXiv preprint arXiv:1812.08434*, 2018.

Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 6530–6539. Curran Associates, Inc., 2017. URL `http://papers.nips.cc/paper/7231-protein-interface-prediction-using-graph-convolutional-networks.pdf`.

Hao Peng, Jianxin Li, Yu He, Yaopeng Liu, Mengjiao Bao, Lihong Wang, Yangqiu Song, and Qiang Yang. Large-scale hierarchical text classification with recursively regularized deep graph-cnn. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*, pages 1063–1072. International World Wide Web Conferences Steering Committee, 2018.

Xiaodan Liang, Xiaohui Shen, Jiashi Feng, Liang Lin, and Shuicheng Yan. Semantic object parsing with graph lstm. In *European Conference on Computer Vision*, pages 125–143. Springer, 2016.