Deep Learning
Assignment 2

Nil Stolt Ansó (12371149)

1 Vanilla RNN versus LSTM

Question 1.1

Applying chain rule:

$$\frac{\partial L^{(T)}}{\partial \mathbf{W}_{ph}} = \frac{\partial L^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial (\mathbf{W_{ph}})}$$

Derivatives for each of the three terms in the chain:

$$\frac{\partial L^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} = -\frac{\partial}{\partial \hat{\mathbf{y}}^{(t)}} \sum_{k=1}^{K} (\mathbf{y}_k \log \hat{\mathbf{y}}_k^{(t)}) = -\frac{\partial}{\partial \hat{\mathbf{y}}^{(t)}} (\mathbf{y}^T \log \hat{\mathbf{y}}^{(t)}) = \left(-\mathbf{y}^T diag(1/\hat{\mathbf{y}}_k^{(t)})\right)^T = -\mathbf{y} diag(1/\hat{\mathbf{y}}_k^{(t)})$$

For each individual element of $\frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}}$ (which results in a K by K matrix due to $\mathbf{p}^{(t)} \in \mathbb{R}^K$ and $\hat{\mathbf{y}}^{(t)} \in \mathbb{R}^K$), the derivative will be:

$$\frac{\partial \hat{y}_i^{(t)}}{\partial p_j^{(t)}} = \frac{\partial softmax(\mathbf{p}^{(t)})_i}{\partial p_j^{(t)}} = \frac{exp(p_i^{(t)}) \sum_j exp(p_j^{(t)}) - exp(p_i^{(t)}) exp(p_j^{(t)})}{\left(\sum_j exp(p_j^{(t)})\right)^2}$$

if $i = j$:

$$\frac{exp(p_i^{(t)}) \sum_j exp(p_j^{(t)}) - exp(p_i^{(t)}) exp(p_j^{(t)})}{\left(\sum_j exp(p_j^{(t)})\right)^2} = \frac{exp(p_i^{(t)})}{\sum_j exp(p_j^{(t)})} \frac{\left(\sum_j exp(p_j^{(t)}) - exp(p_j^{(t)})\right)}{\sum_j exp(p_j^{(t)})}$$

$$= x_i^{(t)} \left(1 - x_j^{(t)}\right)$$

if $i \neq j$:

$$\frac{exp(p_i^{(t)}) \sum_j exp(p_j^{(t)}) - exp(p_i^{(t)}) exp(p_j^{(t)})}{\left(\sum_j exp(p_j^{(t)})\right)^2} = \frac{-exp(p_i^{(t)})}{\sum_j exp(p_j^{(t)})} \frac{exp(p_j^{(t)})}{\sum_j exp(p_j^{(t)})} = -\hat{y}_i^{(t)} \hat{y}_j^{(t)}$$

This can be condensed through the Kronecker delta function into:

$$\frac{\partial \hat{y}_i^{(t)}}{\partial p_j^{(t)}} = \hat{y}_i^{(t)} \left(\delta_{ij} - \hat{y}_j^{(t)}\right), \text{ where } \delta_{ij} = \begin{cases} 1, & \text{if } i = j \\ 0, & \text{if} i \neq j \end{cases}$$

The matrix notation would thus be:

$$\frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} = diag(\hat{\mathbf{y}}^{(t)}) - \hat{\mathbf{y}}^{(t)} (\hat{\mathbf{y}}^{(t)})^T$$

$$\frac{\partial p_i^{(t)}}{\partial (\mathbf{W_{ph}})_{jk}} = \frac{\partial ((\mathbf{W}_{ph})_{j:} h^{(t)} + b)}{\partial (\mathbf{W_{ph}})_{jk}} = \begin{cases} h_k^{(t)}, & \text{if } i = j \\ 0, & \text{otherwise} \end{cases} = \begin{bmatrix} \begin{bmatrix} h_1^{(t)} & ... & h_n^{(t)} \\ \vdots & \vdots & \vdots \\ 0 & ... & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & ... & 0 \\ \vdots & \vdots & \vdots \\ h_1^{(t)} & ... & h_n^{(t)} \end{bmatrix} \end{bmatrix}$$

Putting it all together:

$$\frac{\partial L^{(T)}}{\partial \mathbf{W}_{ph}} = \frac{\partial L^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial (\mathbf{W_{ph}})} = \left(-\mathbf{y}^T diag(1/\hat{\mathbf{y}}_k^{(t)}) \right)^T \left(diag(\hat{\mathbf{y}}^{(t)}) - \hat{\mathbf{y}}^{(t)} (\hat{\mathbf{y}}^{(t)})^T \right) \begin{bmatrix} \begin{bmatrix} h_1^{(t)} & ... & h_n^{(t)} \\ \vdots & \vdots & \vdots \\ 0 & ... & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & ... & 0 \\ \vdots & \vdots & \vdots \\ h_1^{(t)} & ... & h_n^{(t)} \end{bmatrix} \end{bmatrix}$$

Since $\frac{\partial L^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}}$ is a vector and $\frac{\partial \mathbf{p}^{(t)}}{\partial (\mathbf{W_{ph}})}$ is a 3D tensor where all elements are zero except the plane diagonal which is composed of the $\mathbf{h^{(t)}}$ vector side by side K times performing the inner product between these two is equivalent to performing the outer product between $\frac{\partial L^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}}$ and the vector $\mathbf{h^{(t)}}$. Thus:

$$\frac{\partial L^{(T)}}{\partial \mathbf{W}_{ph}} = -\mathbf{y} diag(1/\hat{\mathbf{y}}_k^{(t)}) \left(diag(\hat{\mathbf{y}}^{(t)}) - \hat{\mathbf{y}}^{(t)} (\hat{\mathbf{y}}^{(t)})^T \right) \begin{bmatrix} \begin{bmatrix} h_1^{(t)} & ... & h_n^{(t)} \\ \vdots & \vdots & \vdots \\ 0 & ... & 0 \end{bmatrix} \\ \vdots \\ \begin{bmatrix} 0 & ... & 0 \\ \vdots & \vdots & \vdots \\ h_1^{(t)} & ... & h_n^{(t)} \end{bmatrix} \end{bmatrix}$$

$$= -\mathbf{y} diag(1/\hat{\mathbf{y}}_k^{(t)}) \left(diag(\hat{\mathbf{y}}^{(t)}) - \hat{\mathbf{y}}^{(t)} (\hat{\mathbf{y}}^{(t)})^T \right) (\mathbf{h^{(t)}})^T = -\mathbf{y} \left(I - \mathbf{1} (\hat{\mathbf{y}}^{(t)})^T \right) (\mathbf{h^{(t)}})^T$$

Similarly:

$$\frac{\partial L^{(T)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}} \frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}} \frac{\partial \mathbf{h}^{(t)}}{\partial (\mathbf{W_{hh}})}$$

We can derive the last two terms:

$$\frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}} = \frac{\partial (\mathbf{W_{ph}} \mathbf{h}^{(t)} + \mathbf{b_p})}{\partial \mathbf{h^{(t)}}} = \mathbf{W_{ph}}$$

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{W_{hh}}} = \frac{\partial \tanh\left(\mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h\right)}{\partial \mathbf{W_{hh}}}$$

$$= \frac{\partial \tanh\left(\mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h\right)}{\partial\left(\mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h\right)} \frac{\partial\left(\mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h\right)}{\partial \mathbf{W_{hh}}}$$

$$= diag\left(1 - \tanh^2\left(\mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h\right)\right) \frac{\partial\left(\mathbf{W}_{hh}\mathbf{h}^{(t-1)}\right)}{\partial \mathbf{W_{hh}}}$$

$$= diag\left(1 - \left(\mathbf{h}^{(t)}\right)^2\right)\left(\mathbf{h}^{(t-1)} + \mathbf{W}_{hh}\frac{\partial \mathbf{h}^{(t-1)}}{\partial \mathbf{W}_{hh}}\right)$$

For now, the term ' $diag\left(1 - \left(\mathbf{h}^{(t)}\right)^2\right)$ ' will be written as $\frac{\partial \mathbf{h}^{(t)}}{\partial \hat{\mathbf{h}}^{(\mathbf{t-1})}}$ This equation is recursive up until $\mathbf{h}^0$, where:

$$\frac{\partial \mathbf{h}^{(0)}}{\partial \mathbf{W_{hh}}} = 0$$

For $t > 0$, the unrolled recursive function would look like following.

$t = 1$:

$$\frac{\partial \mathbf{h}^{(1)}}{\partial \mathbf{W_{hh}}} = \frac{\partial \mathbf{h}^{(1)}}{\partial \hat{\mathbf{h}}^{(\mathbf{0})}}\mathbf{h}^{(0)}$$

$t = 2$:

$$\frac{\partial \mathbf{h}^{(2)}}{\partial \mathbf{W_{hh}}} = \frac{\partial \mathbf{h}^{(2)}}{\partial \hat{\mathbf{h}}^{(\mathbf{1})}}\left(\mathbf{h}^{(1)} + \mathbf{W}_{hh}\frac{\partial \mathbf{h}^{(1)}}{\partial \hat{\mathbf{h}}^{(\mathbf{0})}}\mathbf{h}^{(0)}\right)$$

$t = 3$:

$$\frac{\partial \mathbf{h}^{(3)}}{\partial \mathbf{W_{hh}}} = \frac{\partial \mathbf{h}^{(3)}}{\partial \hat{\mathbf{h}}^{(\mathbf{2})}}\left(\mathbf{h}^{(2)} + \mathbf{W}_{hh}\frac{\partial \mathbf{h}^{(2)}}{\partial \hat{\mathbf{h}}^{(1)}}\left(\mathbf{h}^{(1)} + \mathbf{W}_{hh}\frac{\partial \mathbf{h}^{(1)}}{\partial \hat{\mathbf{h}}^{(0)}}\mathbf{h}^{(0)}\right)\right)$$

By expanding the recursive multiplication, we can simplify for the case of t recursive steps:

$$\frac{\partial \mathbf{h}^{(t)}}{\partial \mathbf{W_{hh}}} = \sum_{k=0}^{t-1}\left(\prod_{i=0}^{k}(\mathbf{W}_{hh})^i \frac{\partial \mathbf{h}^{(t-i)}}{\partial \hat{\mathbf{h}}^{(t-i-1)}}\right)\mathbf{h}^{(t-k-1)}$$

Putting everything together:

$$\frac{\partial L^{(T)}}{\partial \mathbf{W}_{hh}} = \frac{\partial L^{(T)}}{\partial \hat{\mathbf{y}}^{(t)}}\frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{p}^{(t)}}\frac{\partial \mathbf{p}^{(t)}}{\partial \mathbf{h}^{(t)}}\frac{\partial \mathbf{h}^{(t)}}{\partial(\mathbf{W_{hh}})}$$

$$= -\mathbf{y} diag(1/\hat{\mathbf{y}}_k^{(t)})\left(diag(\hat{\mathbf{y}}^{(t)}) - \hat{\mathbf{y}}^{(t)}(\hat{\mathbf{y}}^{(t)})^T\right)\mathbf{W}_{ph}\sum_{k=0}^{t-1}\left(\prod_{i=0}^{k}(\mathbf{W}_{hh})^i diag\left(1 - \left(\mathbf{h}^{(t-i)}\right)^2\right)\right)\mathbf{h}^{(t-k-1)}$$

$$= -\mathbf{y}\left(I - \mathbf{1}(\hat{\mathbf{y}}^{(t)})^T\right)\mathbf{W}_{ph}\sum_{k=0}^{t-1}\left(\prod_{i=0}^{k}(\mathbf{W}_{hh})^i diag\left(1 - \left(\mathbf{h}^{(t-1)}\right)^2\right)\right)\mathbf{h}^{(t-k-1)}$$

The derivative $\frac{\partial L^{(T)}}{\partial \mathbf{W}_{ph}}$ only depends on the last time steps's hidden state. This means there is no time dependency problem with the training of these weights.

In the other hand, when updating $\mathbf{W}_{hh}$, the derivative $\frac{\partial L^{(T)}}{\partial \mathbf{W}_{hh}}$, due to its recursive nature, it depends on all the hidden state of all previous time steps. As seen from the summation of products produced by the derivative, the outputs of the *tanh* activation function of each time step are being multiplied. Since the output of *tanh* ranges from -1.0 to 1.0, this means that if the elements of the product every are below $||1||$, the overall product is going to become smaller exponentially the more time steps we are backpropagating through. This causes a vanishing gradient problem. The opposite of this effect is the exploding gradient problem, which arises from cases where the elements in the product is larger than $||1||$. If that is the case, the more time steps we backpropagate through, the size of the gradient scales exponentially.

Question 1.2

Question 1.3

Question 1.4

Vanilla stochastic gradient descent does not take into account the curvature of the loss surface. This means that if the network is currently at a flat loss region, it will only take small steps, causing the network to suffer from undershooting. This could cause the network to not be able to escape local optima.

The opposite case is that of overshooting. If the network is currently on an area of the loss surface where the gradient is really high, the network will take a large step. This causes the network to possibly overshoot optima, due to jumping over its target constantly.

The upside of momentum based approaches is that no single gradient update solely determines the direction of the next update. An update is based on the average direction of the last n updates. This greatly mitigates the problem of overshooting. This helps reduce the variance across batches and to smoothen out the 'roughness' of the loss-surface, thus making the network able to more transition through local optima without getting stuck.

RMSProp tries to mitigate the issue of undershooting and overshooting through the method of an adaptive learning rate. It takes into account previous gradients (with a larger weight on more recent ones) to determine how future gradients will be scaled. If past gradients tended to be small, it will adaptively increase the magnitude of future gradients, letting the network traverse flat regions faster. In the other hand, if previous gradients were large, future gradients will be decreased in magnitude.

Adam takes into consideration these two approaches and combines them into one, resulting in smoother adaptive gradients.

1.3 Long-Short Term Network (LSTM) in PyTorch

Question 1.5

a) The 'input modulation gate' $g^{(t)}$ computes what new information should be added to the cell state from the previous hidden state and the new input. The information is passed through a tanh function, which squeezes the activation between the range of -1.0 and 1.0. This helps reduce the possibility of exploding gradients.

The 'input gate' $i^{(t)}$ computes the magnitude that the input modulation gate should be taking. Since the output of this gate is shaped by the sigmoid function, which has a range between 0.0 and 1.0, in some sense this gate determines what input modulation information can pass through unaltered (in the case of 1.0) and what information should not be passed into the memory cell (in the case of 0.0).

The 'forget gate' is in charge of removing information from the cell state. Based on what information the previous hidden state and the current input have, this gate determines what information in the memory cell should be diminished or removed all-together. This happens through the use of a sigmoid function. The function outputs a value close to 0.0 if the information should be removed, and an output close to 1.0 if the information is allowed to pass through onto following steps.

Finally, the 'output gate' $o^{(t)}$ is used to determine what information from the memory cell should be passed onto the next hidden state based on the current hidden state and the current input. The output of this gate is given by a sigmoid activation function which outputs a value close to 0.0 if the information should be removed, and an output close to 1.0 if the information is allowed to pass. The information being modulated by the output of this gate is that of the cell state, which is first passed through a tanh function, and then later multiplied by the result of the output gate. The result of this operation will become the next hidden state.

b) Since the same gates are applied to the every element in the sequence, the sequence length T has no effect on the number of parameters. Similarly, since the same gates are applied

to every input in the batch independently, the batch size m has no effect on the number of parameters.

Since each unit in the LSTM n is fully connected to every feature in the input d, the parameters weight matrices connecting the input to hidden units is $n \times d$. For weight matrices connecting the hidden state to hidden units, the number of parameters would be $n \times n$. Every hidden unit also have biases for each hidden unit, thus each gate has n parameters. The number of parameters for each gate would thus be:

$$\text{gate parameters} = n \times d + n \times n + n = n(d + n + 1)$$

Since we have 4 gates, the total number of parameters would be $4n(d + n + 1)$.

Furthermore, the linear layer that maps the hidden state to the output nodes o also has parameters. Since the output nodes are fully connected to the hidden nodes, the number of parameters including biases are:

$$\text{linear ouput parameters} = o \times n + o = o(n + 1)$$

Question 1.6

The following plots show the performance of a Vanilla RNN model (Figure 1) and a LSTM model (Figure 2 and 3) on predicting the last digit of palindromes of varying length.

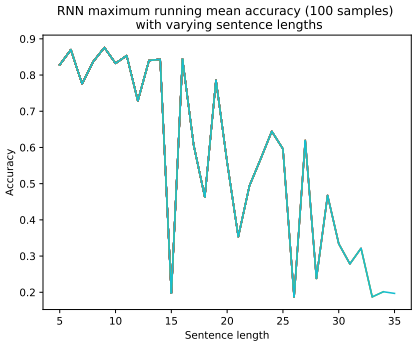

Figure 1: Vanilla RNN performance on palindromes of length 5 to 35. Default parameters were used.

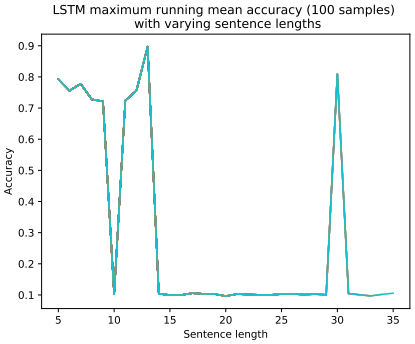

Figure 2: LSTM performance on palindromes of length 5 to 35. Default parameters were used, where learning rate was 0.001.

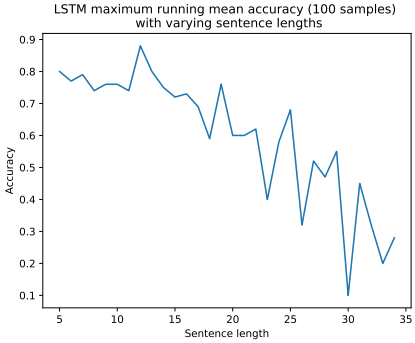

Figure 3: LSTM performance on palindromes of length 5 to 35. Learning rate was changed to 0.0001. All other default parameters were used.

The LSTM with default parameters seems to perform very badly, only achieving good accuracies up to palindrome lengths of around 10 - 15, before falling down to random chance (0.10). In the other hand the RNN model with default parameters seems to, despite having somewhat large variance, be able to predict the last digit of palindromes rather consistently even past the 20 length mark. As seen in Figure 3, lowering the learning rate of the LSTM significantly improved its performance where even past lengths of 30, the predictive performance is still above random chance.

2 Recurrent Nets as Generative Model

For my text data set of choice, I used a book written in Finnish called *Viimeinen syksy*. The author of this novel happens to be my father. I chose this book because, despite it being fun to show my father a machine generating text similar to his book, I though Finnish was a more interesting language to generate text in due to its morphology.

Common short words found in the English language such as '*the*', '*his*', or '*on*' do not exist on their own in Finnish. Instead, suffixes are added to words to convey such meanings. As an example, '*on the table*' would become '*pöydällä*' in Finnish where the word *table* translates to *pöytä*. Finnish thus requires information from further back in time to determine correct characters in caparison to English.

Question 2.1

a) Implementation can be found in the attached files. Give that the data set is a single novel which provides use with around 14000 batches of size 64, dropout and a diminishing learning rate were implemented.

b) The following accuracies shown in Table 1 were achieved by the model. The network's accuracy seems to plateau after around 50000 steps, but as will be discussed later in the bonus question, there are changes to the amount of overfitting happening the longer the network trains.

Training steps	1000	5000	10000	25000	50000	1e5	5e5	1e6
Accuracy	0.42	0.61	0.66	0.71	0.74	0.77	0.8	0.81

Table 1: Accuracies based on amount of training steps.

Every 25000 training steps, given a seed, the model chose the most likely characters occurring over a 30 character sequence. I will only include sentences from early in the training procedure as the performance of the network stagnates after after the 100000 point.

After 1 training step:

- ååZeCuiVu:tmin9i k. E aätanis
- ts

  The network writes gibberish (sentence 1), or in some cases the same character over and over (sentence 2 has the empty space character 28 times in a row).

After 25000 steps:

- 5 hyvin hataloiskaan Puus..\n s
- ! \n Hyvä kuul heidän tapaukse no

  The network appears to have some intuition on how words look like in Finnish (although not fully correct), as well as how to use spaces and punctuation (note the new-line character after the period and the exclamation mark).

After 50000 steps:

- : Ari sanoi kun he olivat jo p
- Hän oli vain kaksi koko perhee

  The network seems to be able to spell words correctly. It also appears to begin including names of main characters in the book (Ari). It makes occasional correct use of morphology like in the case of *Ari sanoi* (translates to *Ari said*). The network still fails to make coherent sentences, like in the case of *Hän oli vain kaksi koko perhee*, which despite being correctly spelled, directly translates to *He was only two whole family*.

After 75000 steps:

- samaan miehistä raiskaasta. \n

  Performance improvement is not as significant anymore, but the model has cases where the morphology across the sentence agrees like in the case of *miehistä raiskaasta* where both words have the *sta/stä* endings.

After 100000 steps:

- Marja Leena ei ollut koskaan k

  The network seems to have memorized the names of main characters and is beginning to write completely sound sentences.

c) Given the probability distribution of the characters predicted by the network, a smaller temperature value makes the model more likely to pick the character with the highest probability, whereas a larger temperature makes the distribution tend towards a uniform one. The following results correspond to the model trained for 100000 steps.

- Temperature value 0.5
  - 4 Vaikka minä olen poissa. Nin
  - Kalle ei ollut kovinkaan vakaa
  - llä kertoi, että hänen vertoja
  - ban punaisen jälkeen, tunsin k
  - 0 todellinen metsikön läpi pol

  For a temperature of 0.5, the text appears close to what it was for the deterministic sampling case.

- Temperature value 1.0
  - Freittä, mutta kukaan ei tienn
  - Ovin tavaroiden jälkeen Ari aj
  - Urho oli ponnikkinen viiniä la
  - 7 syntynyt, ja laittoi soitett
  - , kun mui tapahtuu jotakin ulk

  For a temperature of 1.0, the text appears be less coherent, although still being spelled correctly.

- Temperature value 2.0
  - Iväsi.\n Lyjyttelimmät, anikemin
  - uza unssin, sillä aina oli vaja p
  - B.\n Hyvin tusilta kautain Eripp
  - baa, duoni löytö?\n Mulla on vit
  - ? .e no,.\n Anuhka oli jo tieten

For a temperature of 1.0, the model appears to chose characters extremely uncommon to the Finnish language such as 'z', the words stop making sense, and the punctuation is all over the place (note the increase in backslashes).

Bonus question 2.2

Given that the book used (*Viimeinen syksy*) is novel written in 3rd person, I performed some tests to see whether the network could complete some sentences relating to the main characters. I have not personally read the book, but given how often the network prints some names, I can only assume *Mira* and *Marja Leena* are the main characters.

I created a file named *eval.py* where a model can be loaded and sequences of characters can be fed as a seed for the network to generate a continuation. The default sequence length is 100 and the default temperature is 0.25.

Fed sequence: *Mira sanoi että* (translation: 'Mira said that')

Generated text: *Mira sanoi että hän oli tullut koko ajan kaikki oli siis ollut mitään tarpeellisen kotiin. Miten niin sitä silloin*

Fed sequence: *Marja Leenalla on* (translation: 'Marja Leena has')

Generated text: *Marja Leenalla on tullut paljon parempi kuin mitä hän oli kiinnostunut sanoa sitä hänen omaan sen viiniä koko talon ja*

Both of these sentences, despite having correctly spelled words, lack meaning. From looking at which words follow which other words, I can tell the model generates words that statistically are coherent with the previous one or two words that come before it, but the overall collection is just a set of common short words that do not amount to anything.

I furthermore experimented with a model trained on sequences of 100 characters (instead of the usual 30), as well as with 256 hidden nodes and a slower learning rate.

Fed sequence: *Mira sanoi että* (translation: 'Mira said that')

Generated text: *Mira sanoi että tämä oli aivan liian nopeasti, mutta ei siitä mitään omaksion Marja Leenan vuokratuppiin? Radmilo teki*

translation: *Mira said that this was way too fast, but not that anything omaksion Marja Leena's reservable.*

Fed sequence: *Marja Leenalla on* (translation: 'Marja Leena has')

Generated text: *Marja Leenalla onaina ollut hänen isänsä mielessään yhtä miehen kanssa, kuten aina ennenkin, vaan siksi että se oli t*

translation: *Marja Leena has always been in her father's perspective not with any men, like always before, only because it was t*

With such model trained on long sequences of characters, sentences begin to make sense. I personally find it really funny that the style of the generated text mimics that of the novel, which is a melodrama about the sexual lives of it's main characters (see translations above).

Something interesting to note, is that as I was showing these to my father (the author), he pointed out how the longer the models were trained, the more the wrote text straight from the book, despite the coherence of the text not having a marginal change after the first 100000 training steps. This is in line with what we would expect from the model overfitting the training data. At the 1 million tranining step mark, my father could recognize full sentences he wrote on the book, while at around 50000, the model trained on sequence lengths of 100 was not yet fully generating semantically coherent sentences. This is despite the training accuracy being practically the same in both scenarios.

This latter 100 sequence length model also appeared to be able to somewhat form paragraphs of similar lengths one would expect in a book when using a generation length of a 1000. See text below:

...
Vaikka Ninni ei sitä kyllä tiedä. Mä teedännen kuin hän olisi kertonut aivan jostakin muuta. Kuulta suuri elinajaksi tai jotain.

En tiedä mitä se on. Sillä kävi tapahtu kun on siellä uutessa hyvän kertaa. Et sit ne on nyt edea niin. Minä ja Lilian meni kaiken sitten toista kun vappasivat tietenkin paljon enemmän kuin mitä se oli meidän salakuljettajana. Aivan kuin hän olisi ollut suoraan kotona.
...

---

3 Graph Neural Networks

Question 3.1

a) The forward function of a GCN layer is very similar to that of a common neural network's linear module in the sense that there is some input to the layer $H^{(l)}$ which gets multiplied by a weight matrix $W^{(l)}$. Thus $W^{(l)}$ as well as the sigmoid activation function seen in Equation 1 bring nothing different from regular feedforward layers. The only difference that exploits the graph structure is that in a GCN layer, as seen in Equation 1, the term by which $W^{(l)}$ is multiplied also contains the activations of neighboring nodes.

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

The matrix $\hat{A}$, contains weighting terms that sums up the activation of all neighbours of a given element in $H^{(l)}$. It should be noted that $\hat{A}$ also has a weighting element for itself, this is an assumption in the model, as we could also chose to set the self-looping weight of a neuron to 0 (which is defined as A). By setting a self-loop weight to zero, we would be explicitly make neurons ignore it's own direct previous activations (although it's own influence could travel through other nodes in the graph before propagating back into the original neuron).

b) After one layer, the activation of a given neuron will have influenced nodes directly connecting to it. After two layers, it will have influenced nodes two neighbors away. Thus eventually, after n layers, the influence of a node will have propagated a maximum of n nodes away. The information of a node can travel as many hops away as there are layers, making 3 hops away be achieved by a minimum of 3 layers.

Question 3.2

The following are some of the most common applications of GNN:

- Predicting reactivity of molecules: Molecules can be modelled with a graph structure where each atom is a node bound to other neighboring atoms. Specially in the field of pharmaceutics, where families of large and very complex molecules, the reactivity predictions that GNNs provide decreases the amount of drug candidates that need to be manually tested.

- Text Analysis: Natural Language has plenty of lexical relations that have been modelled in graph structures such as WordNet. WordNet groups words into sets of synonyms (synsets). The information in the graph structure on how words relate to other words can be exploited to push performance in text analysis tasks
- Image classification: Often, data sets for image classification will have different labels that are equal in meaning (synonyms) or part of the same hierarchy of terms. Exploiting the graph structure of the meaning behind the label words can help generalize across concepts, merge data sets, or label images in a semi-unsupervised manner. An example of such an approach is the work done in the YOLO9000 architecture [1], where datasets were merged and targets used during training were based on a lexical hierarchy of words.

Question 3.3

a) RNNs are specially good for dealing with sequential data of undefined length. Similar to how MLPs work, GNNs need a fixed amount of layers equal to the sequence length you want to predict, whereas with an RNN new input can keep being fed to the network as long as necessary. This makes RNNs a safe choice when dealing with time series such as stock market price data or NLP tasks.

In some examples both GNNs and RNNs can be applied to the same kind of task. An example is natural language, where we can chose to represent text as a graph structure (through the use of parse trees for example). It is still highly debated which of the two types of networks is best suited to deal with such data and the choice depends on the specific application.

In the other hand, there are domains where the number of neighboring nodes varies from node to node. Such an example is graph structures of social connections found in social medias. Nodes representing different people will have a large variance in the amount of neighboring nodes. For such cases, GNNs are the go-to choice, as they are specialized with dealing with a varying number of neighbors.

b) An example of a conbination of the two approaches is the work done on Tree-LSTMs [2], where sequences of text are expressed in the form of a tree structure in order to improve the ability of the network to understand which words are semantically connected to others.

Further applications I can only speculate about. The most obvious applications of a combination of GNNs and RNNs is a graph structure which is tracked over varying sequences of time. One such example would be the folding of proteins, where the graph structure of the molecule changes over time depending on its neighbors, but the time steps over which such changes take place are arbitrarily long.

References

[1] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[2] Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *CoRR*, abs/1503.00075, 2015.