Assignment 2. Recurrent Neural Networks and Graph Neural Networks

Martine Toering
11302925
Deep Learning Course Assignment 2
University of Amsterdam
November 29, 2019
martine.toering@student.uva.nl

1 Vanilla RNN versus LSTM

1.1 Toy Problem: Palindrome Numbers

1.2 Vanilla RNN in PyTorch

Question 1.1 Write down an expression for the gradient $\frac{\partial L^{(T)}}{\partial W_{ph}}$ in terms of the variables that appear in Equations 1 and 2. Do the same for $\frac{\partial L^{(T)}}{\partial W_{hh}}$. What difference do you observe in temporal dependence of the two gradients. Study the latter gradient and explain what problems might occur when training this recurrent network for a large number of timesteps.

The gradients are computed using Backpropagation through Time (BPTT).

- $\dfrac{\partial L^{(T)}}{\partial W_{ph}} = \dfrac{\partial L^{(T)}}{\partial p^{(T)}} \dfrac{\partial p^{(T)}}{\partial h^{(T)}} \dfrac{\partial p^{(T)}}{\partial W_{ph}}$

  This gradient only depends on timestep T.

  $= \dfrac{\partial L^{(T)}}{\partial p^{(T)}} \left(h^{(T)} \right)$

- $\dfrac{\partial L^{(T)}}{\partial W_{hh}} = \dfrac{\partial L^{(T)}}{\partial p^{(T)}} \dfrac{\partial p^{(T)}}{\partial h^{(T)}} \dfrac{\partial h^{(T)}}{\partial W_{hh}}$

  The important thing to note here is that $h^{(T)}$ is dependent on $h^{(T-1)}$ thus we need $\dfrac{\partial h^{(T)}}{\partial h^{(T-1)}}$. However, this is again the case for $h^{(T-1)}$, and so on.

  If we for instance compute this gradient for $h^{(T)}$:

  $\dfrac{\partial h^{(T)}}{\partial h^{(T-1)}} = W_{hh}^{(T)}$

  Thus, we obtain

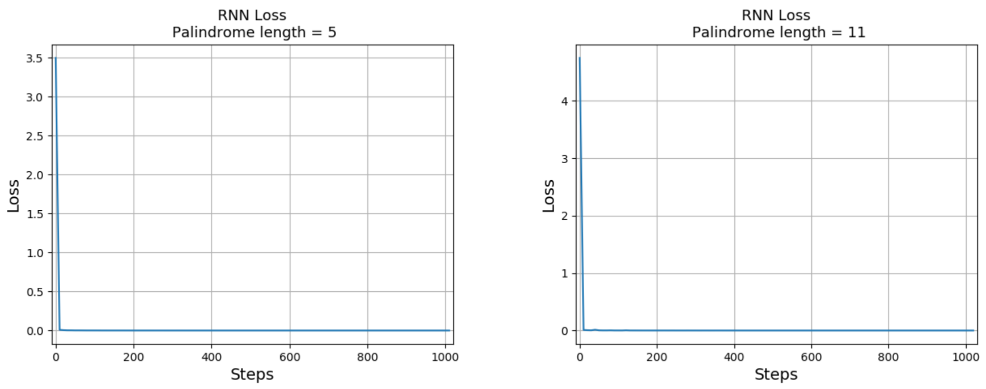


Figure 1: Training loss for short palindrome of length 5 ($T = 5$) and palindrome for default parameters ($T = 11$) for the vanilla RNN

$$
\begin{aligned}
&= \frac{\partial L^{(T)}}{\partial p^{(T)}} \Big(W_{hh}^{(T)} \Big) \Big(\frac{\partial h^{(T)}}{\partial W_{hh}} \Big) \\
&= \frac{\partial L^{(T)}}{\partial p^{(T)}} \Big(W_{hh}^{(T)} \Big) \Big(\sum_{i=1}^{T} W_{hh}^{T-i} \ h_i \Big)
\end{aligned}
$$

As we can see, the gradients depend on old weights of all timesteps. This is a problem the more we backpropagate in time, as the new gradients are only an estimate and we have a product over all timesteps. Two problems that could arise because of this are the problems of vanishing gradients and exploding gradients. This means that the gradients either get too small or too large when using a large number of timesteps.

Question 1.2 Implement the vanilla recurrent neural network as specified by the equations above in the file *vanilla_rnn.py*.

Question 1.3 Start with short palindromes (T = 5), train the network until convergence and record the accuracy. Repeat this by gradually increasing the sequence length and create a plot that shows the accuracy versus palindrome length. As a sanity check, make sure that you obtain a near-perfect accuracy for $T = 5$ with the default parameters provided in part1/train.py. To plot your results, evaluate your trained model for each palindrome length on a large enough separate set of palindromes, and repeat the experiment with different seeds to display stable results.

The model was trained for palindromes of various lengths. Figure 1 shows the training loss for an input length of 4 and thus a palindrome length of 5 ($T = 5$) next to a palindrome length of 11 ($T = 11$). The figure shows that the model converges almost immediately (after approximately one training step). In the experiments that followed, we trained from 1 to length 40 every two input lengths (the odd lengths) and from 40 to 80 every four lengths. we decided that convergence is a moment in training when the loss does not decrease for n amount of training steps with respect to the best loss at that moment. This n training steps for convergence can be passed as an argument to the train function. This value should be chosen carefully as we do not want the model to converge earlier when it might improve later on. A value of 1000 steps was chosen (the total number of training steps used is 10,000). For the most part, the default parameters were used. The input batches however were converted to a one-hot representation, being more suitable in language processing tasks and most classification tasks. The weights were initialized with He initialization.

Next, we evaluated our trained model. A test on the model was performed for different palindrome lengths. A test set was created and the model was trained for 5000 (test) steps. For each palindrome

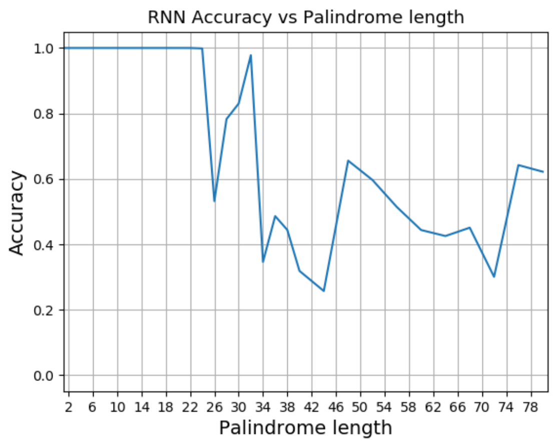


Figure 2: Accuracy versus palindrome length for the vanilla RNN

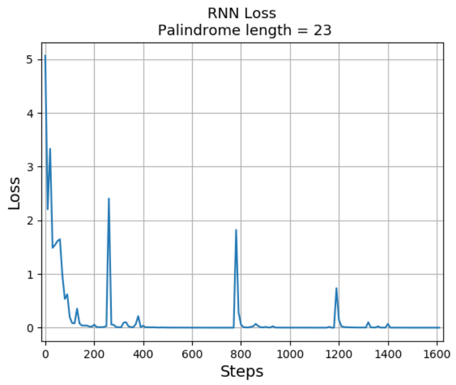


Figure 3: Training loss for longer palindrome of length 23 ($T = 23$) for the vanilla RNN which shows an unstable loss

length the experiment was repeated with three different seeds. Figure 2 shows the palindrome length versus the (test) accuracy that was averaged over each three experiments. It can be observed that the performance of the vanilla RNN starts declining at around a palindrome length of 24. For greater lengths, the accuracy seems to fluctuate around 40%.

The loss seems to be rather unstable for larger input lengths. Figure 3 shows for instance input length of 22 ($T = 23$).

Question 1.4 To improve optimization of neural networks, many variants of stochastic gradient descent have been proposed. Two popular optimizers are RMSProp and Adam and/or the use of momentum. In practice, these methods are able to converge faster and obtain better local minima. In your own words, write down the benefits of such methods in comparison to vanilla stochastic gradient descent. Your answer needs to touch upon the concepts of momentum and adaptive learning rate.

Momentum is a method which uses a moving exponential average of past gradients in updating and therefore is able to maintain ’momentum’ which reduces oscillations from gradients.

Root Mean Square Propagation (RMSProp) makes use of adaptive updating of the learning rate and is somewhat similar to momentum. A problem is that some gradients may be very large and some may be very small; RMSProp tries to deal with this by keeping a moving average of the squared gradients for each weight. Thus, these squared gradients denote gradient norm of the past few steps. The norm of the current gradient is added to this moving average. The learning rate is updated by dividing by the root of this exponential average of squared gradients. Looking at how quickly the gradient magnitude changes, small gradients are scaled as needed as well as large gradients (where momentum would easily overshoot).

Adam optimizer has both an adaptive learning rate and makes use of momentum (Kingma and Ba 2015). This means that it keeps both an exponential average of past gradients (momentum) and an exponential average of squared gradients (as is used in RMSProp). Adam also has a correction bias.

Stochastic gradient descent (vanilla) does not use these extra advanced options such as an adaptive learning rate, a momentum or correction bias. It is therefore possible that vanilla stochastic gradient descent does not take the optimal path in problems like the pathological curvature. Stochastic gradient descent can however be used in combination with momentum.

1.3 Long-Short Term Network (LSTM) in PyTorch

Question 1.5 a) The LSTM extends the vanilla RNN cell by adding four gating mechanisms. Those gating mechanisms are crucial for successfully training recurrent neural networks. The LSTM has an input modulation gate g(t), input gate i (t), forget gate f (t) and output gate o(t). For each of these gates, write down a brief explanation of their purpose; explicitly discuss the non-linearity they use and motivate why this is a good choice.

An LSTM is a network capable of learning long-term dependencies. It has a memory which is the cell state. This LSTM has four gates; an input gate, an input modulation gate, a forget gate and an output gate. These functions are called *gates* because they encode particular information and subsequently let this through to add to the memory if it is deemed important. So, these gates control what information is added to or removed from the cell state; where a 1 usually means to keep it completely and 0 means to forget all information there.

Modulation gate $g(t)$ The modulation gate determines which information is added or *removed* with respect to the input gate for the cell state. The modulation gate $g(t)$ therefore uses the $tanh$ activation function and outputs a value in range [-1, 1] instead of [0, 1].
Input gate $i(t)$ This input gate looks at the information in the input and decides what is important and should be let through and added to the memory. This input gate has a sigmoid activation function and outputs in the range of [0, 1]. This input gate alone is therefore only able to add information to the cell state.
Forget gate $f(t)$ This forget gate decided what should be *removed* from memory. It outputs in range [0, 1] where 1 means keep information and 0 is forgetting.
Output gate $o(t)$ The output gate decides which parts we want to output and uses also a sigmoid activation which denotes whether we want information to output or not. Later, we use this to compute the actual parts that are output to the cell state.

Question 1.5 b) Given the LSTM cell as defined by the equations above and an input sample $x \in R^{T\times d}$ where T denotes the sequence length and d is the feature dimensionality. Let n denote the number of units in the LSTM and m represents the batch size. Write down the formula for the total number of trainable parameters in the LSTM cell as defined above.

The formula for the total number of trainable parameters in this LSTM network is given by

$4n^2 + 4dn + 4n$

where n denotes the number of units and d denotes the feature dimensionality of the input or the input size. We have four weight matrices for the input; $W_{gx}, W_{ix}, W_{fx}, W_{ox}$. These are weight matrices of input size by hidden dimension. We also have four weight matrices for the hidden state; $W_{gh}, W_{ih}, W_{fh}, W_{oh}$. These are hidden dimension by hidden dimension. Lastly, we have four biases which are n.

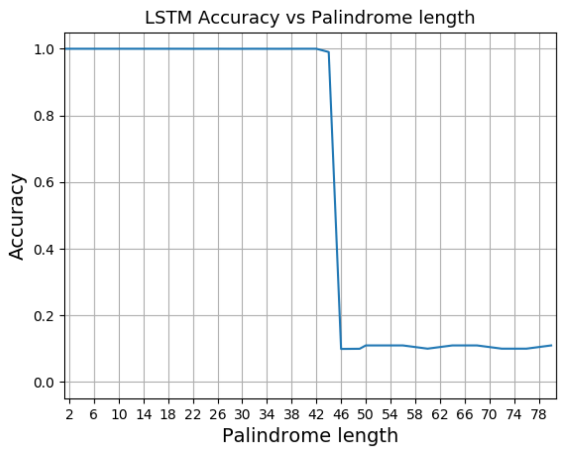


Figure 4: Accuracy versus palindrome length for the LSTM

Question 1.6 Implement the LSTM network as specified by the equations above in the file *lstm.py*. Just like for the Vanilla RNN, you are required to implement the model without any high-level PyTorch functions. Using the palindromes as input, perform the same experiment you have done in Question 1.3.

Figure 4 shows the accuracy for each palindrome length. The expectation beforehand is that the LSTM performs better. We trained again from 1 to length 40 only every two lengths and from 40 to 80 every four lengths. The learning rate was adjusted to $0.001 * 10 = 0.01$ for input lengths greater than 37 as this seemed to perform better and speed up the training process. The LSTM indeed performs better than the vanilla RNN. It nicely converges for palindrome numbers up until lengths of 44. However, it still struggles after a certain input length, here this is 45 ($T = 46$). From then on, the performance declines more drastically then for the vanilla RNN.

Question 1.7 Modify your implementations of RNN and LSTM to obtain the gradients between time steps of the sequence. You do not have to train a network for this task. Take as input a palindrome as before and predict the number at the final time step of the sequence. Note that the gradients over time steps does not imply the gradients of the RNN/LSTM cell blocks, but the message that gets passed between time-steps (i.e, the hidden state). Plot the gradients for both the variants over different time steps and explain your results. Do the results correlate with the findings in Question 1.6? What results will you expect if we actually train the network instead for this task? Submit your implementation as a separate file called *grads_over_time.py*.

See figure 5 for a visualization of the gradient magnitude of the hidden state between timesteps for input lengths of 10 and 100 respectively. We can clearly see that the gradients of the vanilla RNN grow very fast were the gradients for the LSTM stay stable over both 10 and 100 timesteps. The gradients of the RNN thus grow exponentially and become much larger than the gradients for the LSTM after about 10 timesteps. Thus, even our simple gradient magnitude plot shows signs in our model of a well-known problem of RNN, namely exploding gradients. It furthermore shows that LSTM with its memory cell does have gradients with respect to the hidden state that stay around the same order of magnitude.

If we would actually train the model, it is expected that these gradients of the hidden state for the RNN become even larger in magnitude. After a training step, the weights are updated and this results in *large weights*. These large weights can therefore also cause really unstable loss in the training process which we have also seen in our RNN model for large input lengths (figure 3). The results also seem to correlate with the findings from *question 1.3* and *question 1.6* where the LSTM performs

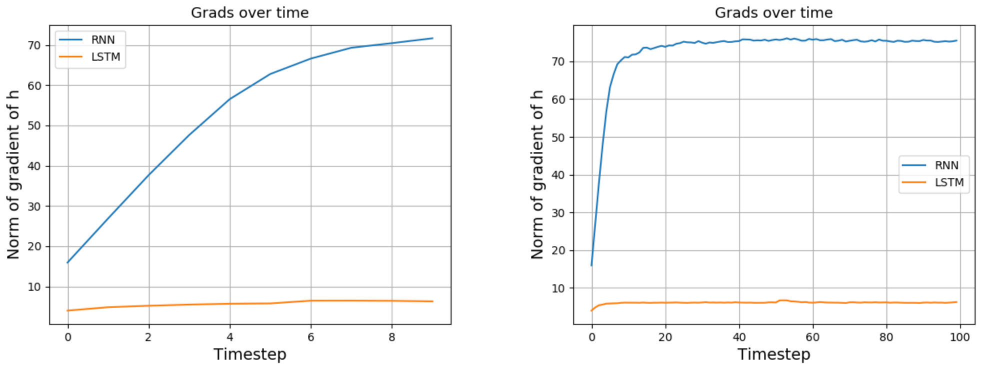


Figure 5: Gradient magnitude over time steps for input length of 10 (left) and input length of 100 (right) for vanilla RNN and LSTM

better in training for more input steps.

2 Recurrent Nets as Generative Model

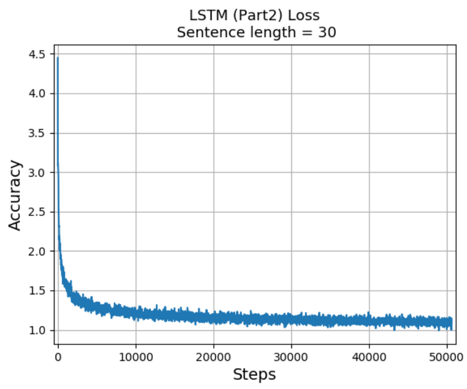
Question 2.1 a) Implement a two-layer LSTM network to predict the next character in a sentence by training on sentences from a book. Train the model on sentences of length $T = 30$ from your book of choice. Define the total loss as average of cross-entropy loss over all timesteps (Equation 13). Plot the model's loss and accuracy during training, and report all the relevant hyperparameters that you used and shortly explain why you used them.

A two-layer LSTM model was trained on sentences of $T = 30$ from the book *Grimms' Fairy Tales* by Jacob Grimm and Wilhelm Grimm. The training loss and accuracy can be seen in figure 6. Here, most of the default parameters are used such as batch size 64, hidden units to 128, the default learning rate of 2e-3. However, we added dropout layer on the outputs of the first LSTM layer by passing a value for dropout probability to the LSTM constructor. This was chosen because dropout can possibly help reduce overfitting on the dataset. A dropout probability of 0.2 was used; this value is 1 - dropout_keep_prob so the dropout_keep_prob argument was changed to 0.8.

A Learning rate scheduler was also added with a decay for the learning rate and step at which the learning rate is decreased both at default parameters (learning rate decay at 0.96 and learning rate step at 5000). This was done in order to obtain a better minimum; a learning rate too high can possibly keep overshooting. Such a learning rate schedule could perhaps result in for instance a few percents higher accuracy. We can see in the figure that for this setup, we reach an accuracy of around 62% on the training set.

Question 2.1 b) Make the network generate new sentences of length $T = 30$ now and then by randomly setting the first character of the sentence. Report 5 text samples generated by the network over different stages of training. Carefully study the text generated by your network. What changes do you observe when the training process evolves? For your dataset, some patterns might be better visible when generating sentences longer than 30 characters. Discuss the difference in the quality of sentences generated (e.g. coherency) for a sequence length of less and more than 30 characters.

It was attempted to let the model generate new sentences of length $T = 30$. This was done by taking a random small input batch of length $T = 30$ from the data and for each T, predict the next character. Place this at the next place in the input and feed this again to the model (per instructions of the TA). This way, the first character always stays the same. It became apparent that changing the dropout value had large effects on the sampled sentences; increasing the dropout keep probability even higher,



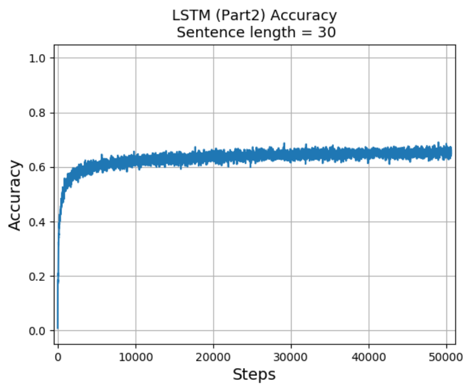


Figure 6: Training loss (left) and training accuracy (right) for the two-layer LSTM trained on sentences of 30 characters from the book *Grimms' Fairy Tales*.

the generated sentences seem to use only existing words / words from the dataset. Lowering the value to below 0.8 the words became really random and were most of the time non existing words. Here, we show sample sentences for a dropout keep probability of 0.8.

Below some sentences from the model are listed at step 200;

```
ce the the the the the the the
ind the the the the the the th
ind the the the the the the th
```

at step 500;

```
 n the said the said the said t
 ing the said the said the said
  on the said the said the said
```

step 1000:

```
the first had so the first an
 r and said, ‘I will soon a lit
 ched the first had said, ‘I wi
```

Below, some sentences are shown which were generated at step 5000.

```
y went out of the wood, and th
pen the wood to her the water,
e the stars, and the second ca
```

And at step 50,000:

```
ll the beautiful bird and said
ing and said: ‘I will soon fin
e the second son said, ‘I will
```

We can observe from the examples and the rest of the generated sentences that there is still (a lot) repetition present at the end of training. The model learns more about the book as we get further into the training process however. It uses words that are not only ’the’ and creates almost completely correct sentences. The sample sentences also show us how the sentences actually make sense semantically and structurally; for example, the sentence *"the second son said"*. There are also other sentences like *"the wood to her the water"* which are of lesser quality in terms of coherence and semantic correctness. However, the model does reasonably well.

We can try the same setup for larger input lengths. We now train the model as before on $T = 30$, but we sample sentences of length $T = 40$.

At step 1500:

```
l the world was a golden cage, and said,
the second son was so beautiful as the
and the seven little tailor was all the

the second time the second strength to
he seven years were set out of the wood,
s the fox said, 'I will go home and said
```

It is clear that for longer sentences the model seems to perform somewhat on the same level; overall, the sentences do not seem to have particularly better or worse coherence or structure; but it varies more between sentences. We can see that for example in the generated sentence: *"the second son was so beautiful as the"* the coherence is quite good. However, some sentences do make less sense now, as *"the world ... said"* is not a logical sentence, just as *"the second time the second strength to"*.

Below some samples of length $T = 20$:

```
the second time th
and the maiden had
came to the street

the wolf was so happ
the strength in the
 the second brother
```

These sentences seem to be of somewhat better quality. These sentences all seem to be logical and have more coherence than the samples before, for example *"the maiden had"*. However, for these very short sentences coherence between words is easier to achieve than for longer sentences as there are less words.

Question 2.1 c) Your current implementation uses greedy sampling: the next character is always chosen by selecting the one with the highest probability. On the complete opposite, we could also perform random sampling: this will result in a high diversity of sequences but they will be meaningless. However, we can interpolate between these two extreme cases by using a temperature parameter τ in the softmax.

- **Explain the effect of the temperature parameter τ on the sampling process.**

  This temperature parameter τ controls how much random sampling we perform as opposed to greedy sampling. This basically means that with this parameter, we can make it for instance such that it is less likely for the model to choose a next character with a low probability and more randomness is added to the predictions. A high value of τ in this case (such as 2 for example) scales the output of the softmax up so when we sample from it it will with even more probability choose likely candidates. Thus, for a high temperature value this sampling tends somewhat to greedy sampling. A value of $\tau = 1$ does not scale the softmax so it is just sampling from the distribution. What is interesting is that a lower value of τ scales the output of the softmax down allowing more diversity. This adds this earlier mentioned and possibly desired randomness and diversity to the generation process and creates softer probabilities. However, too much randomness and it will be meaningless.

- **Extend your current model by adding the temperature parameter τ to balance the sampling strategy between fully-greedy and fully-random.**

The model was extended by a temperature parameter τ which scales the values before the values are given to the softmax function. Then, we sample from the distribution with the softmax probabilities. It is important to add –greedy_sampling True as an argument when we want the model to perform greedy sampling and to false to perform random sampling; in this case passing a float number to the –temperature argument.

- **Report generated sentences for temperature values $\tau \in \{0.5, 1.0, 2.0\}$. What do you observe for different temperatures? What are the differences with respect to the sentences obtained by greedy sampling?**

  For a temperature value of 0.5 the results seem to be very different from the greedy sampling results from before. Even the generated sentences in the end of the training phase seem to have no correct words whatsoever. There are also a lot of spaces and enters generated. Thus, as expected, the generated sentences are much more diverse as there is a lot more diversity in which characters are generated. Below one example is shown.

  ```
  " am son mooms of nithin mony.
  kss vischeron other, quitely k
  ", she himps. 'IV Cay'
  ```

  For a temperature value of 1, there is still a lot of that diversity present. Below we have some sentences generated after training the model with a temperature $\tau = 1$.

  ```
  g at the other father and cHou
  weld as he could not go fine w
  dows belowted upon her fields
  ```

  For a temperature value of 2 the generated sentences are now actual sentences and are similar in coherence to the generated sentences from before (greedy sampling). We can observe that we have correct words and even sentences. There seems to be not as much repetition as before with greedy sampling (certainly at the beginning of the training process). All in all, the sentences generated with this setting $\tau = 2$ seem to be the best sentences generated yet.

  ```
  and the miller said, 'I will
  as the door opened the door to
  long the wolf had a faithful
  ```

Question 2.2 It could be fun to make your network finish some sentences that are related to the book that your are training on. For example, if you are training on Grimm's Fairytales, you could make the network finish the sentence "Sleeping beauty is ...". Be creative and test the capabilities of your model. What do you notice? Discuss what you see.

Now, we use the network to finish sentences from the book of sentence length $T = 30$. This was done by taking a random batch from data of $T = 30$ and for the second half, let the network predict characters. Below, some sample sentences are shown.

```
will give you  good counsel. I
 bread, and the  second son was
, taking care n ot the work in

l sides of him  to the streets,
re were once a  man who was so
how sweetly the  woman said, 'I
```

Some particularly odd examples can be seen below.

```
n he declared h imself a bird a
 he heard a str ange princess a
but if you thro w you the stair
```

Question 2.3 There is also another method that could be used for sampling: Beam Search. Shortly describe how it works, and mention when it is particularly useful. Implement it (you can choose the beam size), and discuss the results.

3 Graph Neural Networks

3.1 GCN Forward Layer

Question 3.1 a) Describe how this layer exploits the structural information in the graph data, and how a GCN layer can be seen as performing message passing over the graph.

First, the elements of the adjacency matrix A of $N \times N$ denote which pairs of edges are adjacent and which are not in a graph with N nodes. The GCN Forward layer takes the adjacency matrix considering self-connections $\tilde{A}$ as input (which is a combination of adjacency matrix A and identity matrix) and the matrix H which contains the activations (which are features of the nodes at the input) and is $N \times d$. The shared weights W are learned. H^{l+1} is obtained by applying an activation function and thus non-linearity. At the output layer the output will be a representation of each node with $N \times o$ with o being the output dimension.

Thus, the model exploits the structural information of the graph of N nodes encoded in matrix A. This GCN layer can be seen as performing message passing over the graph; information about the neighbours of a node are encoded in H and each layer performs message passing through all the nodes. Therefore, a n-layer network passes messages to nodes n hops away. By using more layers the node representation receives messages from neighbours further away.

Question 3.1 b) There are drawbacks to GCNs. Mention one, and explain how we can overcome this.

A drawback of GCN is the difficulties that arise when we want to train *deeper* models (Kipf and Welling 2016). Deeper GCN models are harder to train (because of vanishing gradient problems). Kipf and Welling attempt to tackle this problem by using residual connections or skip connections that can carry over information between layers (He et al. 2016). Others have tried recurrent connections (Huang and Carley 2019).

Question 3.2 a) Give the adjacency matrix $\tilde{A}$ for the graph as specified in Equation 16.

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

$$I_N = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\tilde{A} = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

Question 3.2 b) How many updates (as defined in Equation 14) will it take to forward the information from node C to node E?

It will take model three updates or three forward propagation steps as E is (at best) three hops away from C. In this third layer the model will learn about third-order neighbours of the node C.

3.2 Applications of GNNs

Question 3.3 Take a look at the publicly available literature and name a few real-world applications in which GNNs could be applied.

My intuition is that GNN can be applied in many areas, for instance, hierarchical databases like WordNet, customer databases, social media connections, connections between cities like infrastructure networks, and so on. More examples I found are applications in *recommender systems* and an application in *protein interface prediction*, a problem in chemistry. Furthermore, there are also a lot of examples to be found in not very explicit structural applications such as semantic segmentation, image classification and machine translation (Zhou et al. 2018).

3.3 Comparing and Combining GNNs and RNNs

Question 3.4 a) Consider using RNN-based models to work with sequence representations, and GNNs to work with graph representations of some data. Discuss what the benefits are of choosing either one of the two approaches in different situations. For example, in what tasks (or datasets) would you see one of the two outperform the other? What representation is more expressive for what kind of data?

RNN are highly effective in text classification, machine translation and all kinds of tasks in natural language processing and tasks were the input is *sequential*. Recurrent Neural Network (RNN) based models can capture long-term dependencies and can be used sequence-to-sequence, as well as one-to-one or sequence-to-one. Graph Neural Networks (GNN) are often not that flexible. However, graphs are highly *structural*, as is data. Graph Neural Networks can have many applications in visualization for instance. We use Graph Neural Networks when we want to exploit structure and *relation information*. It is also possible to use a combination of GNNs and RNNs to capture both *recurrent* and *sequential* patterns.

Question 3.4 b) Think about how GNNs and RNNs models could be used in a combined model, and for what tasks this model could be used. Feel free to mention examples from literature to answer this question.

Huang and Carley have used recurrent connections in graph neural networks. Their RGNN model can be used in a supervised and unsupervised setting (Huang and Carley 2019). We might want to both capture graph-level representation and long-term dependencies over nodes. This can for instance be useful in a graph classification task (Jin and JáJá 2018).

References

He, K., Zhang, X., Ren, S., and Sun, J. (2016). "Deep Residual Learning for Image Recognition". In: *IEEE Conference on Computer Vision and Pattern Recognition*. DOI: `10.1109/cvpr.2016.90`.

Huang, B. and Carley, K. M. (2019). "Inductive Graph Representation Learning with Recurrent Graph Neural Networks". In: *CoRR* abs/1904.08035. arXiv: `1904.08035`. URL: `http://arxiv.org/abs/1904.08035`.

Jin, Y. and JáJá, J. F. (2018). "Learning Graph-Level Representations with Gated Recurrent Neural Networks". In: *CoRR* abs/1805.07683. arXiv: `1805.07683`. URL: `http://arxiv.org/abs/1805.07683`.

Kingma, D. P. and Ba, J. (2015). "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980.

Kipf, T. N. and Welling, M. (2016). "Semi-Supervised Classification with Graph Convolutional Networks". In: *arXiv preprint arXiv:1609.02907*.

Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z., and Sun, M. (2018). "Graph Neural Networks: A Review of Methods and Applications". In: *CoRR* abs/1812.08434. arXiv: `1812.08434`. URL: `http://arxiv.org/abs/1812.08434`.