Deep Learning Practical 2

Longxiang Wang
Master of Artificial Intelligence
University of Amsterdam
wlongxiang1119@gmail.com

1 Vanilla RNN versus LSTM

Q1.1: Analytical derivation of gradients

Note that for simplicity of gradident calculation, we only write out the loss at time step t here. And to reduce the clutter of equations, we assign the following:

$$\boldsymbol{U} = \boldsymbol{W}_{hx} \tag{1}$$
$$\boldsymbol{V} = \boldsymbol{W}_{ph} \tag{2}$$
$$\boldsymbol{W} = \boldsymbol{W}_{hh} \tag{3}$$
$$\tag{4}$$

The equations that define a vanilla RNN are as follows:

$$\boldsymbol{h}^{(t)} = tanh(\boldsymbol{U}\boldsymbol{x}^{(t)} + \boldsymbol{W}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h) \tag{5}$$
$$\boldsymbol{p}^{(t)} = \boldsymbol{V}\boldsymbol{h}^{(t)} + \boldsymbol{b}_p \tag{6}$$
$$\hat{\boldsymbol{y}} = softmax(\boldsymbol{p}^{(t)}) \tag{7}$$
$$\mathcal{L}^{(t)} = -\sum_{k=1}^{K} y_k log(\hat{y}_k) \tag{8}$$

For the gradient w.r.t. the output weight matrix, let's look at only one component of the full gradient matrix, which is a scalar.

$$\frac{\partial \mathcal{L}^{(t)}}{\partial V_{ij}} = \sum_k \frac{\partial \mathcal{L}^{(t)}}{\partial \hat{y}_k} \frac{\partial \hat{y}_k}{\partial p_i^{(t)}} \frac{\partial p_i^{(t)}}{\partial V_{ij}} \tag{9}$$

For the cross entropy loss part (part 1):

$$\frac{\partial \mathcal{L}^{(t)}}{\partial \hat{y}_k} = \begin{cases} -\frac{1}{\hat{y}_k}, & \text{if } k = argmax(\boldsymbol{y}) \\ 0, & \text{if } k \neq argmax(\boldsymbol{y}) \end{cases} \tag{10}$$

For the softmax part(part 2), which we have derived in detail in assignment 1, here we just present the result:

$$\frac{\partial \hat{y}_k}{\partial p_i^{(t)}} = \begin{cases} -\hat{y}_k \hat{y}_i, & \text{if } k = i \\ \hat{y}_k - \hat{y}_k \hat{y}_i, & \text{if } k \neq i \end{cases} \tag{11}$$

For the output part (part 3), according to the original RNN equation 2, it is a linear function:

$$\frac{\partial p_i^{(t)}}{\partial V_{ij}} = h_j^{(t)} \tag{12}$$

Putting all the above parts back, we have (note that this is a scalar value):

$$\boxed{\frac{\partial \mathcal{L}^{(t)}}{\partial V_{ij}} = (y_i - \hat{y}_i) h_j^{(t)}} \tag{13}$$

For the derivative w.r.t. the hidden-to-hidden matrix, we have:

$$\frac{\partial \mathcal{L}^{(t)}}{\partial W_{ij}} = \sum_u \frac{\partial \mathcal{L}^{(t)}}{\partial h_u^{(t)}} \frac{\partial h_u^{(t)}}{\partial W_{ij}} = \sum_u \big(\sum_i \frac{\partial \mathcal{L}^{(t)}}{\partial p_i^{(t)}} \frac{\partial p_i^{(t)}}{\partial h_u^{(t)}} \big) \frac{\partial h_u^{(t)}}{\partial W_{ij}} \tag{14}$$

As we have seen from solving the gradients of hidden-to-output weights, the inner sum in the bracket can be further simplified as:

$$\frac{\partial \mathcal{L}^{(t)}}{\partial W_{ij}} = \sum_u \Big(\sum_i (y_i - \hat{y}_i) V_{iu} \Big) \frac{\partial h_u^{(t)}}{\partial W_{ij}} \tag{15}$$

Now the most tricky part is the second term in the above equation, which has a dependency over the previous time step according to the first equation of the vanilla RNN settings. Therefore, we cannot treat the hidden state as const here, unlike what we did for the hidden-to-out gradient V_{ij}.

$$\frac{\partial h_u^{(t)}}{\partial W_{ij}} = \sum_{k=0}^{t} \frac{\partial h_u^{(t)}}{\partial h_u^{(k)}} \frac{\partial h_u^{(k)}}{\partial W_{ij}} \tag{16}$$

$$\tag{17}$$

In the above equation, the gradient $\frac{\partial h_u^{(t)}}{\partial h_u^{(k)}}$ is again subject to chain rule:

$$\frac{\partial h_u^{(t)}}{\partial h_u^{(k)}} = \frac{\partial h_u^{(t)}}{\partial h_u^{(t-1)}} \frac{\partial h_u^{(t-1)}}{\partial h_u^{(t-2)}} \cdots \frac{\partial h_u^{(k+1)}}{\partial h_u^{(k)}} \tag{18}$$

As we can see that the gradient of hidden-to-output weights V is much simpler, and it only depends on its own time step, which the gradient of hidden-to-hidden weights W (and also input-to-hidden weights U) are very complicated because it has dependencies over previous time steps. Because of the product of gradients introduced by the sequential dependency, depending on the sequence length, there could be vanishing gradient issue when the gradients are small or exploding gradient issue when gradients are big.

Q1.2: Pytorch Implementation of Vanilla RNN

The implementation is done in the code.

It is noticed that the vanilla RNN trained with RMSProp is quite sensitive to weight initialization of U, W, V. When we initialize the weights with $\mathcal{N}(0, 1)$, it barely learns anything with the default setting. Then, we tried to initialize with pytorch's buildin `nn.init.xavier_normal_()` function for U, W, V and zeros for biases. It quickly converges to accuracy of 1, and the training process is robust. This initialization technique is proposed in this paper [1].

Q1.3: Experiments with different sequence length

Different input sequences are experimented to study the performance and robustness of RNN. We have tried to train 3 models for each sequence length of [5, 10, 12, 16, 20], the results are shown in Figure 1, 2, 3, 4, 5.

It is noticed that in general, vanilla RNN quickly reaches near perfect accuracy when the sequence length is smaller than 10. However, as sequence length increases, we start to observe unstable training process and also the accuracy starts to degrade. It is also interesting to see that sometimes, the training got 'stuck', e.g., the orange line for input sequence 12 in Figure 3. Our guess is that it is more likely to have vanishing gradient issue when the sequence gets larger due to the product issue during backprop through time.

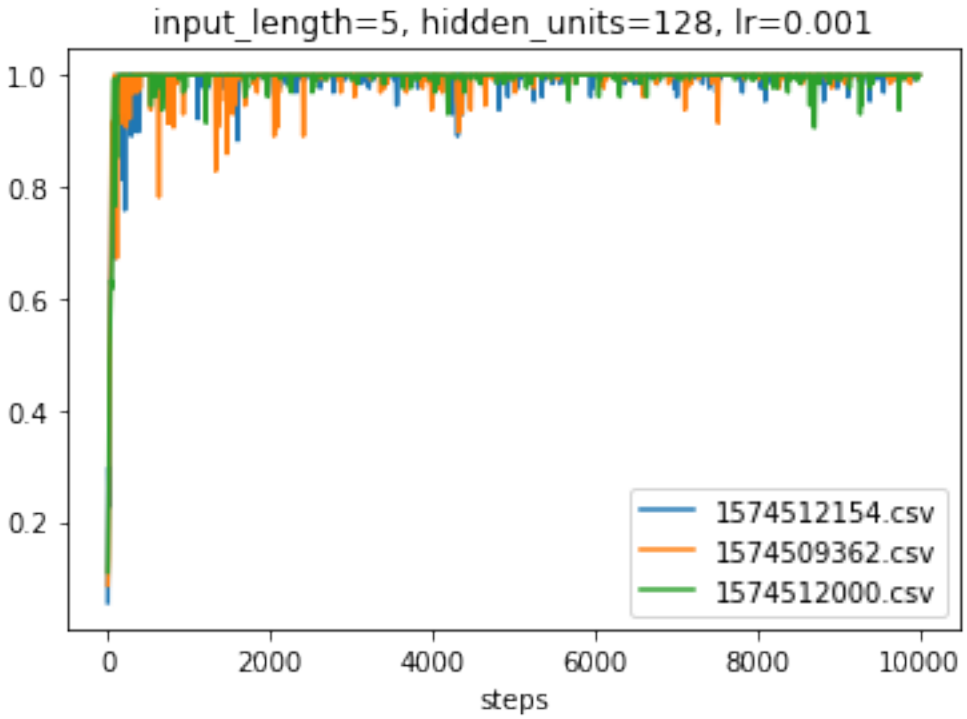

Figure 1: Training vanilla RNN of sequence length 5

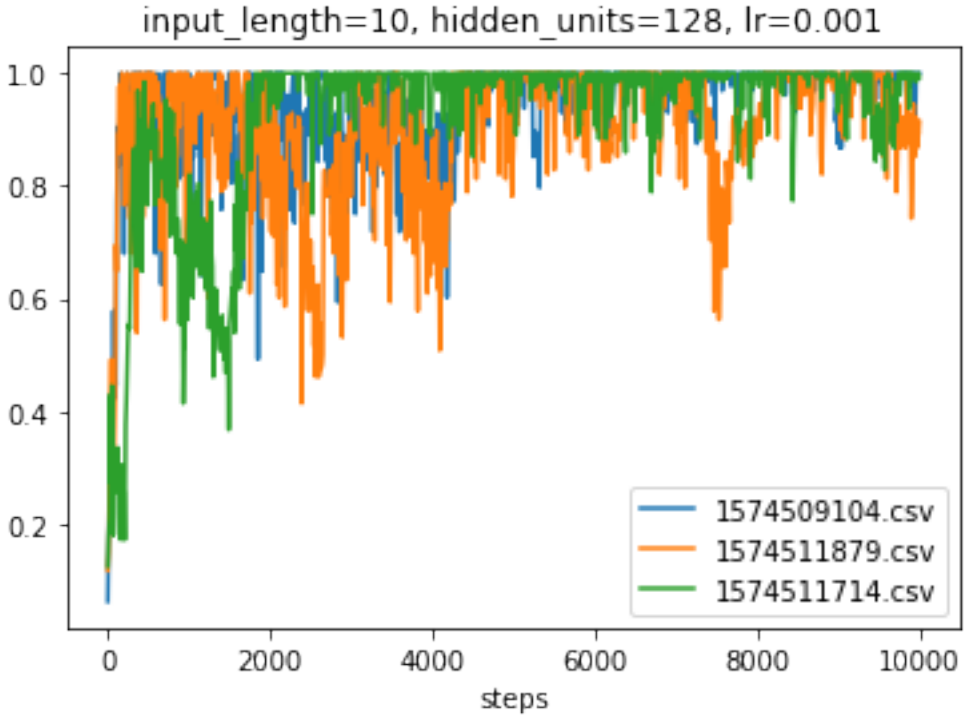

Figure 2: Training vanilla RNN of sequence length 10

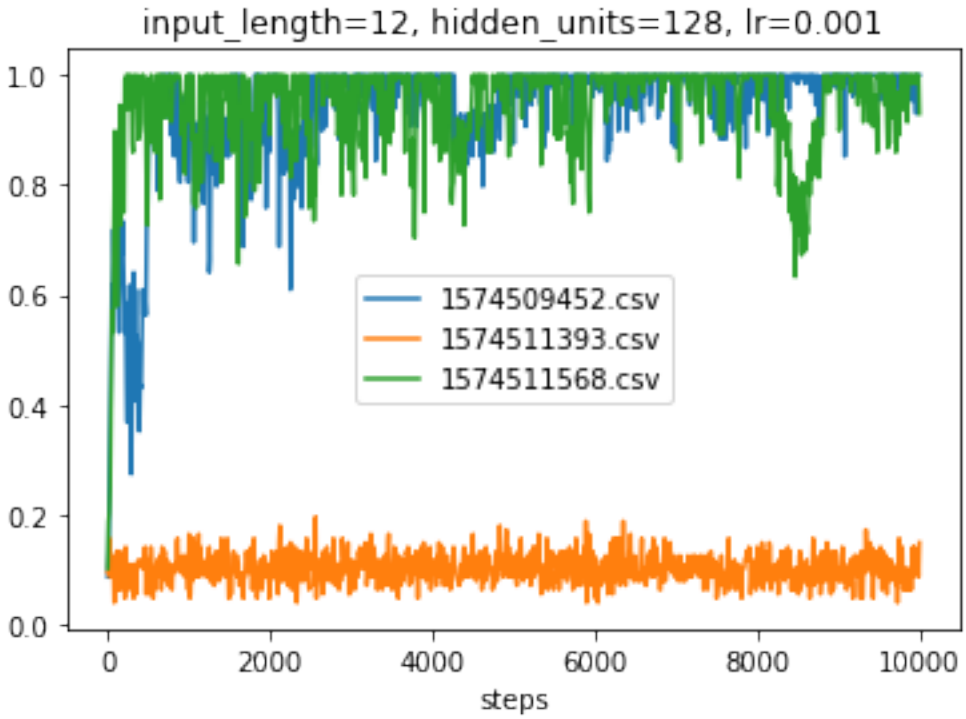

Figure 3: Training vanilla RNN of sequence length 12

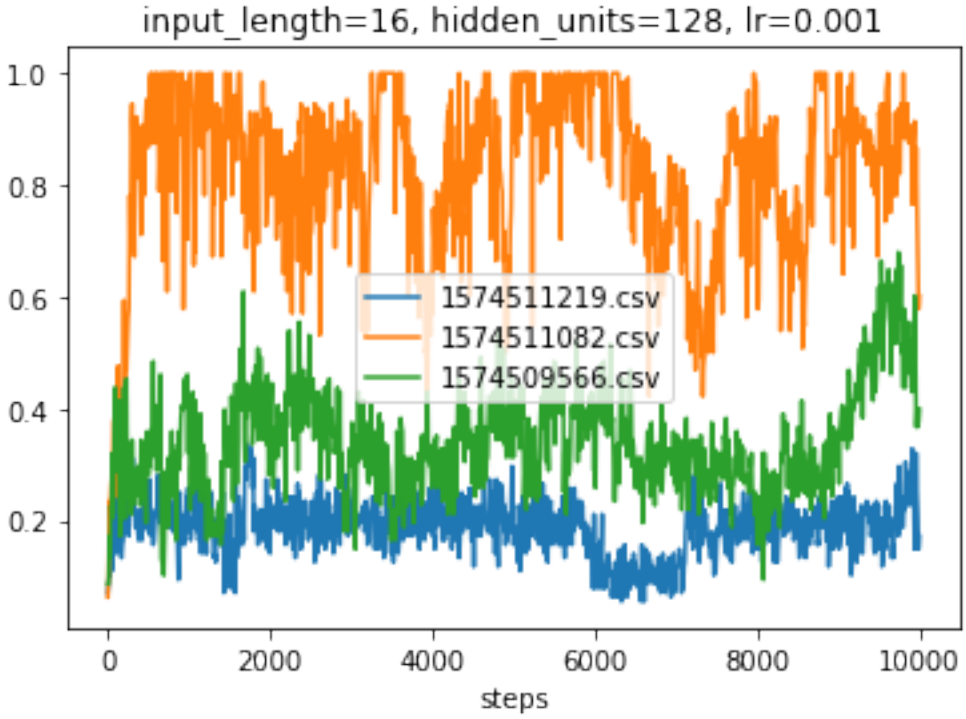

Figure 4: Training vanilla RNN of sequence length 16

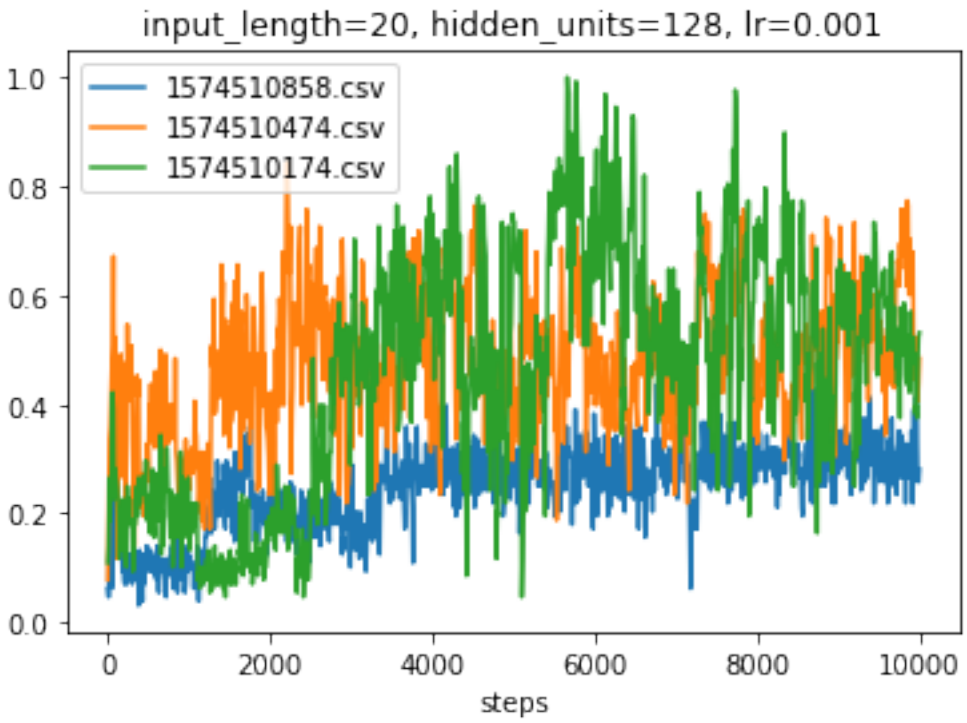

Figure 5: Training vanilla RNN of sequence length 20

Q1.4: Comments on SGD variants such as RMSProp and Adam

In vanilla SGD, we update network parameters according to the gradient direction, which is the direction of the greatest rate of decrease of loss function.

$$\boldsymbol{W}^{(\tau+1)} = \boldsymbol{W}^{(\tau)} - \eta \nabla \mathcal{L}(\boldsymbol{W}^{(\tau)}) \tag{19}$$

However, this approach does not take into account of second order derivative information, thus are converging slower and inefficient in practice. It treats all steps equally, which means at points with high curvature, it could take a step too large causing oscillation behavior and at points with low curvature, it could also move too slow.

In theory, we could evaluate the Hessian matrix of our loss function. One big problem with Hessian is that it has complexity of $O(W^2)$ for a network of W parameters. To still make use of second order information and also avoiding evaluating Hessian directly, momentum and adaptive learning rates are used. Both RMSProp and Adam use momentum and adaptive rates in their methods.

With momentum, we take previous gradients into account instead of relying on the gradient at a step alone. One could simply use a moving average of a specific window size to update the gradient. Thus, we can achieve a smoother training process.

With adaptive learning rates, we gradually decrease our learning rate according to certain scheduler of choice. The intuition behind is that at the beginning of training, we could take a larger step because we are still far from the optimal point. As training goes on, we should do smaller updates near the optimal point.

Q1.5: LSTM architecture

a) Gates used in LSTM

Four gates used in LSTM are motivated here:

- Input modulating gate: this gate uses previous hidden state and current input. It first computes a linear transformation of $\boldsymbol{h}^{(t-1)}$ and $\boldsymbol{x}^{(t)}$ and then applies tanh non-linearity on

top of it. Using tanh can caputure complex non-linearity in input space and dependencies on previous states. tanh can also clip the output to [-1, 1] centered around 0, which is often desired by deep learning models.

- Input gate: this gate acts like a binary switch. When its value is 1, it passes through the entire input, when its value is 0, it rejects all its input. In this case, the input is the modulated input from the 'input modulating gate'. It determines how much of the modulated input (including $\boldsymbol{h}^{(t-1)}$ and $\boldsymbol{x}^{(t)}$) influences the current cell state. Sigmoid is used here due to its modulating effect with continuous positive output from 0 to 1.
- Forget gate: it determines how much of the previous cell state influences the current cell state. It also uses sigmoid function for the same reason as mentioned above for the input gate.
- Output gate: it determines how much of the current cell state influence the current hidden state. The use of sigmoid is for the same reason as mentioned above.

b) Trainable parameters in LSTM

There are 4 gates in LSTM with equal dimension weights. Let's examine the dimensinality of the input modulating gate as example:

$$\begin{aligned} \boldsymbol{W}_{gx} &\in R^{n \times d} \\ \boldsymbol{W}_{gh} &\in R^{n \times n} \\ \boldsymbol{b}_{g} &\in R^{n} \end{aligned}$$

where d is the dimension of input at time step t, $\boldsymbol{x}^t$, n is the dimension of hidden state at time step t, $\boldsymbol{h}^t$, note that cell state $\boldsymbol{c}^t$ has the same dimension as hidden state.

Therefore we have $4(n + n^2 + nd)$ trainable parameters in total for LSTM unit (excluding the output linear module).

Q1.6: LSTM implementation

LSTM is implemented as seen in the code. training steps are reduced to 5000, and learning rate are adjusted as compared to RNN, since LSTM observes much stable training performance. The results are show in Figure 6, 7, 8 and 9.

It was found that LSTM is more capable of modeling long term dependencies than vanilla RNN, and its training process is much more robust as shown by the small variances in different runs.

To show LSTM's superb long term memory capabilities, an input sequence of 40 is trained, and the result is shown in Figure 9. As we can see, the model is able to achieve near to 1 accuracy after around 2.5k iterations even for such long input sequence. Only starting from sequence length of 40, we start to see not learning behavior as shown by the green line in Figure 9.

These results show the gating mechanism in LSTM cells overcomes some of the difficulties faced by vanilla RNN to certain extend, such as modelling long term memory, vanishning gradients. It is also worth to mention that it does not completely solve the problems of vanilla RNN, as sequence length increases to the larger end, those problems of vanilla RNN also occur for LSTM.

Q1.7: Vanilla RNN and LSTM gradients visualization

We use the pytorch autograd functionality to obtain intermediate gradients. In here we cache the hidden states in our custom model, therefore we can obtain the gradients at different time steps. As shown in Figure 10, gradients accumulate as time steps flow, which indicates the hidden states information is passed on through time steps. The difference between RNN and LSTM is obvious. As shown, RNN has a log-scale linear gradient flow through time steps, which makes it subject to vanishing gradients adn exploding gradients, which LSTM has a stable gradient flow which does not change much when sequence length is less than 30, but also increases linearly when sequence goes high. This phenomenon explains why LSTM is more robust than RNN, and also shows us LSTM has the same problem of vanishing gradients just less severe when sequence length is relatively small.

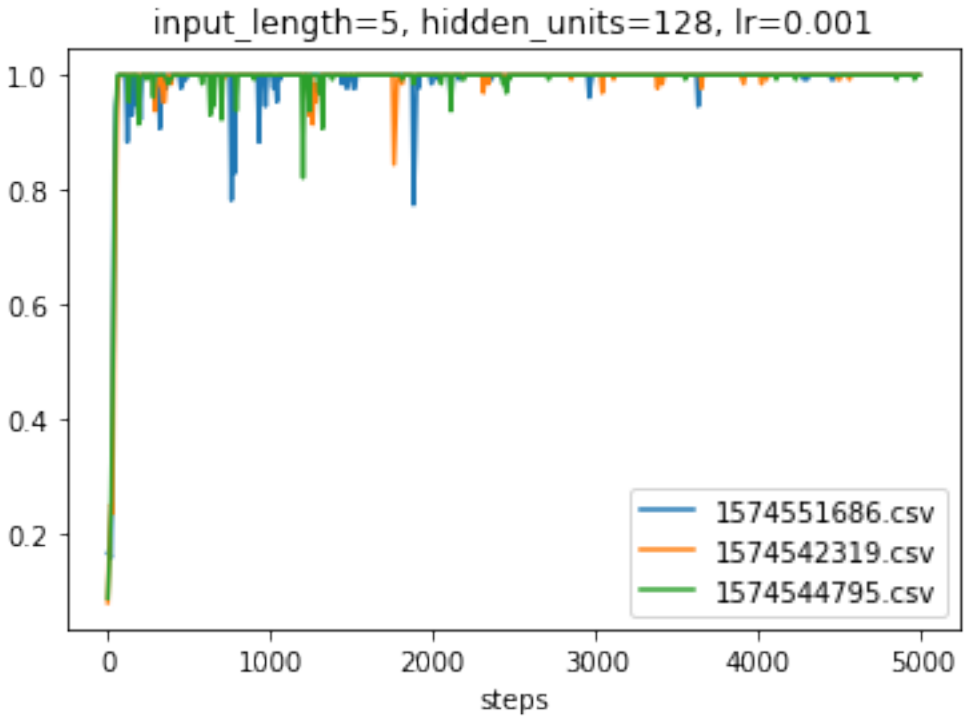

Figure 6: Training LSTM of sequence length 5

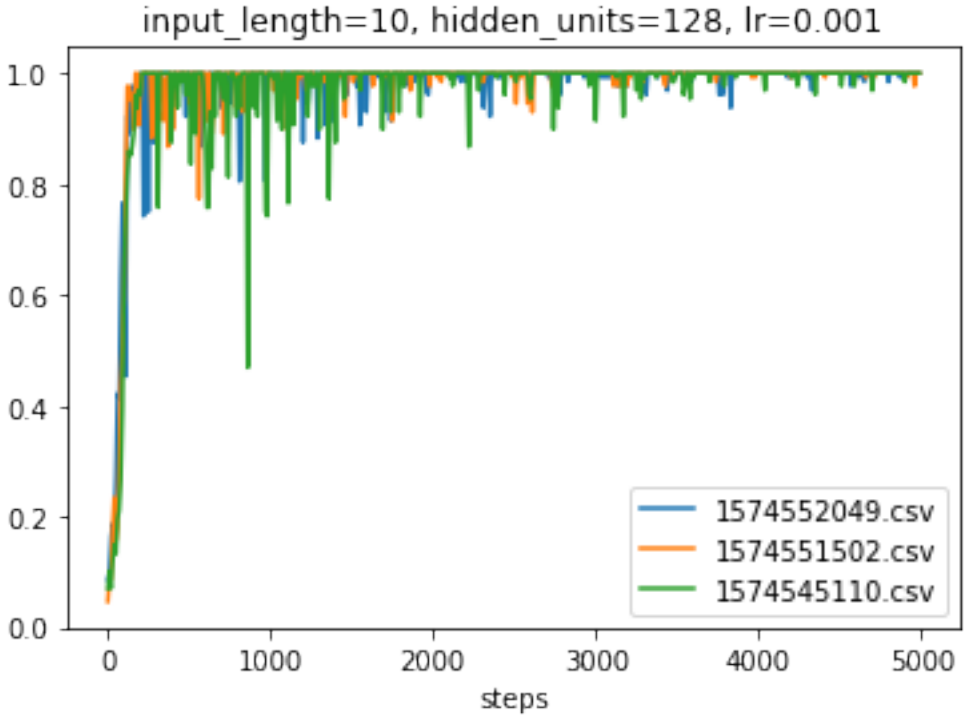

Figure 7: Training LSTM of sequence length 10

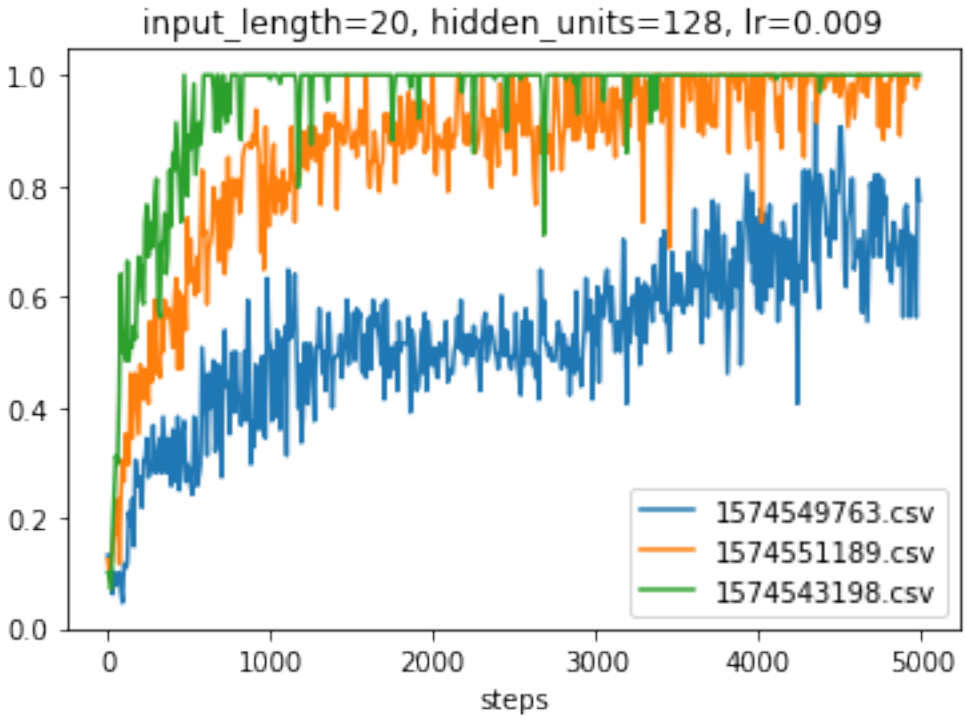

Figure 8: Training LSTM of sequence length 20

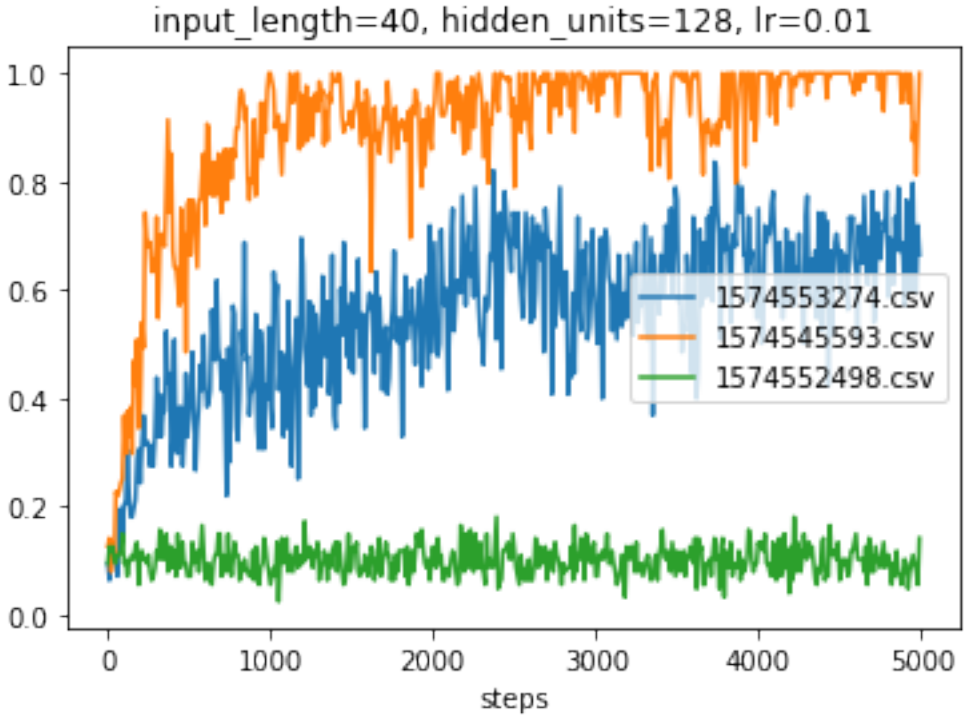

Figure 9: Training LSTM of sequence length 40

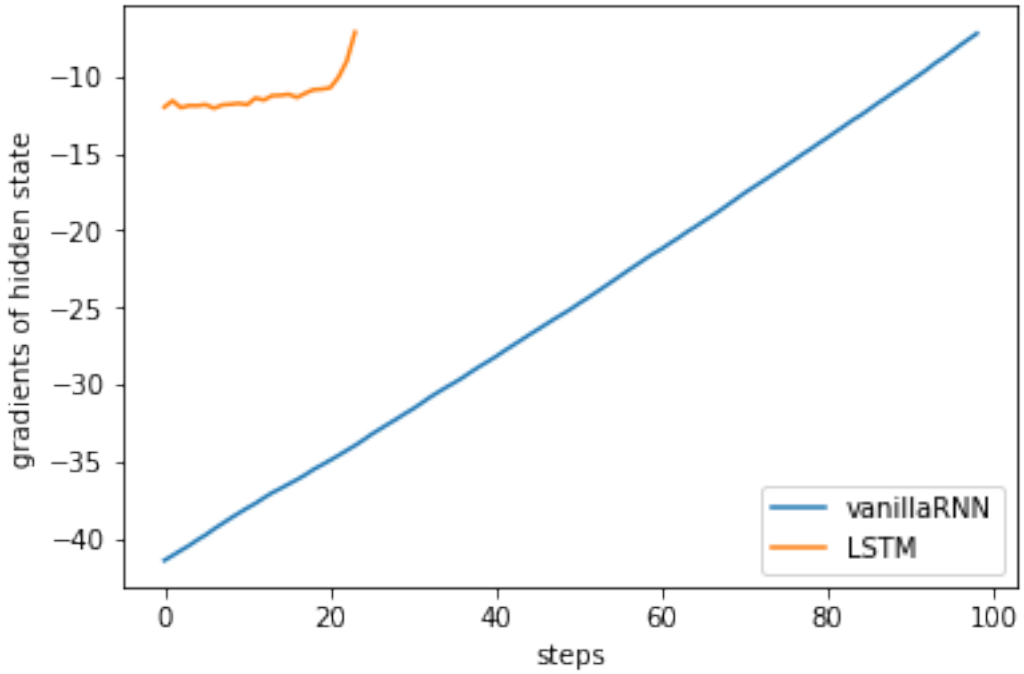

Figure 10: L1 norm of gradients w.r.t to hidden state at different time steps. Note that y-axis is in log-scale

To observe the gradients of loss w.r.t. the hidden-to-hidden matrix W, we trained a LSTM multiple times to visualize how the gradients change in different steps. It is noticed when we see accuracy not improving during training, it is mostly likely caused by vanishing gradients issue. And LSTM is much less likely to have vanishing gradients issue as compared to vanilla RNN. However, as shown in Figure 11, LSTM does not completely solve this problem, because it still occurs when the input length is too high.

Q2.1 a): Train a LSTM to generate text

A LSTM network is trained to generate a sequence-to-sequence mapping for generating text from a certain corpus, the example text used here is `book_EN_grimms_fairy_tails.txt`. There are a few hyper parameters that we need to choose here:

- batch size: 64. This was chosen be default, also works reasonably on a normal laptop.
- learning rate: 0.01. This was chosen by be higher than the default 0.002, because it proves in practice to converge faster and generate some meaningful texts faster in this dataset.
- Adam optimizer was used with default setting and the learning rate as specified above.
- sequence length was 30, this was chosen because it could show a reasonable result of the LSTM model.
- LSTM number of layers: 2 ; hidden units in LSTM: 128. These parameters are chosen because they are easy to start with.

With the settings above, an training accuracy around 0.64 was achieved after around 5000 iterations and converges. The training graph is presented in Figure 12 and 13.

Q2.1 b): Generate text of different lengths

We first experimented with a sequence length of 30. Some sample of generated text given a character is presented in Table 2 for sequence length of 30.

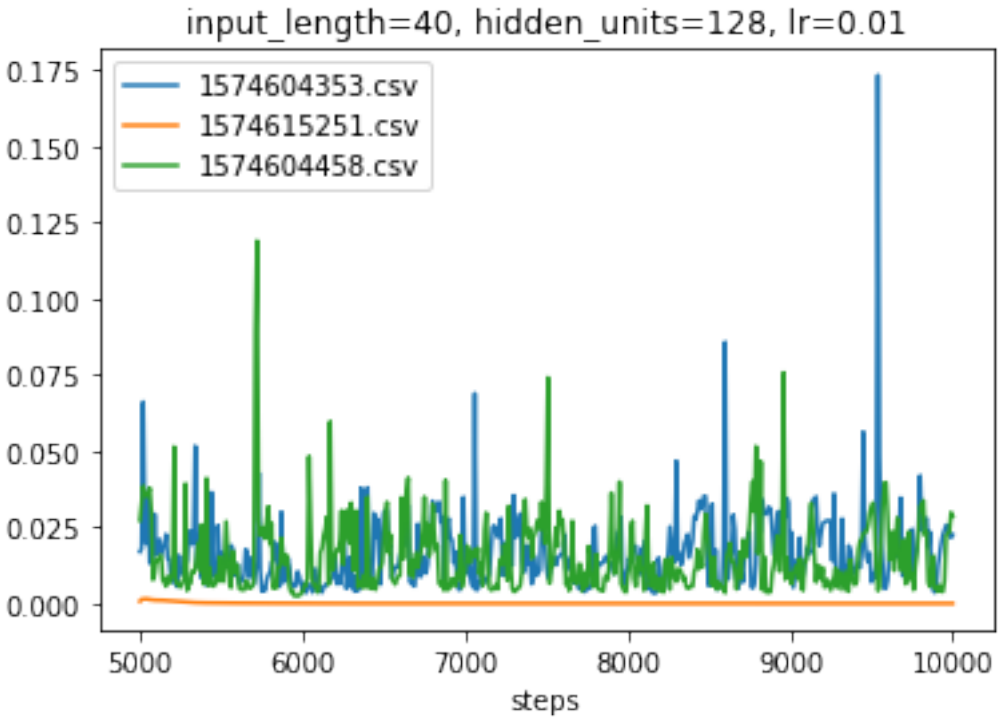

Figure 11: L1 norm of gradients w.r.t to W of training LSTM of sequence length 40, the bottom orange line corresponds to a low training accuracy of 0.1, while the other two cases are around accuracy of 1.

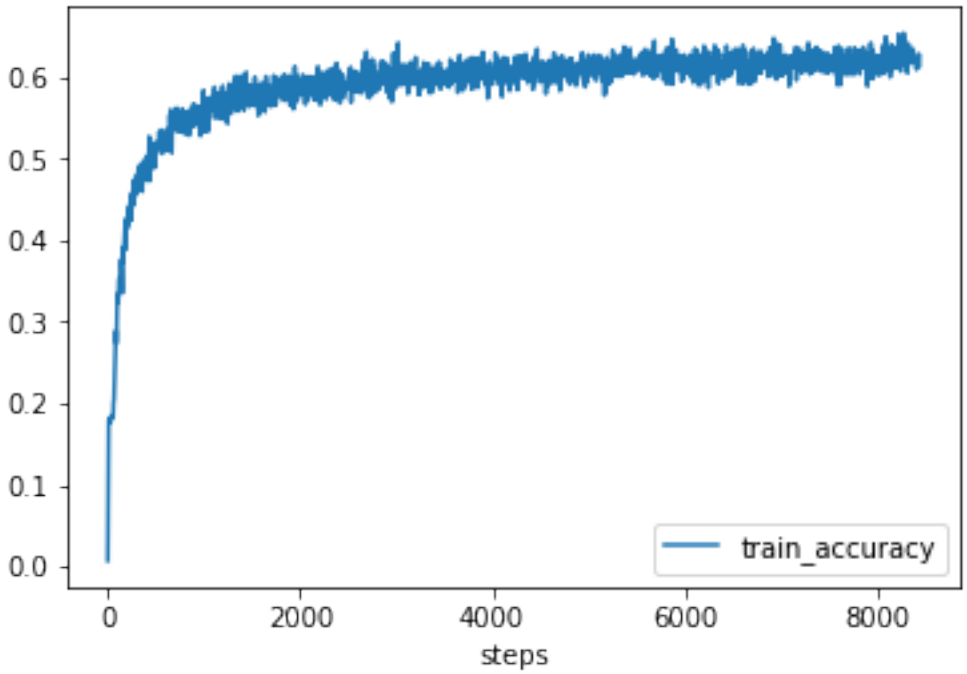

Figure 12: LSTM text generation model of sequence length 30

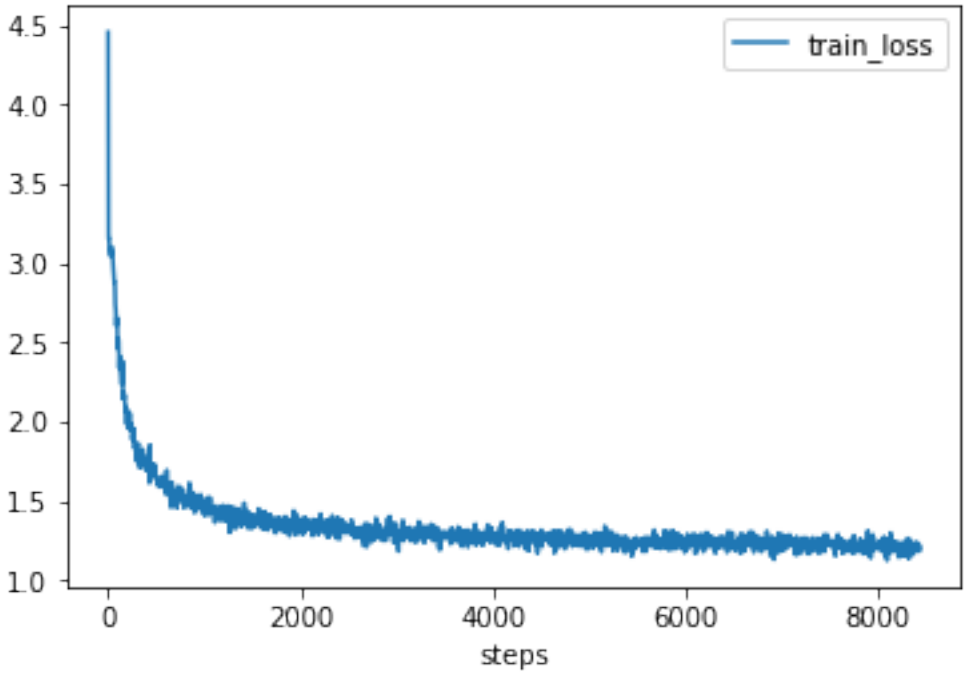

Figure 13: LSTM text generation model of sequence length 30

As we can see, in the early stage of training regardless of sequence length, some common repetitive characters are captured, such as **the** or **te**. This is caused by the high frequency of these combinations in the training corpus. As training evolves, around training steps of 400, the model already picks up some other words like **will**. At step 1200, the model starts to exhibit the capability of making even grammatical sentence like **Chanted to the world and said:** , though it does not have exactly any meaning yet. As the training converges around 8200 steps, it is even able to make names like Hans as a subject, and make meaningful words like **Hans went into the water and s**.

We also tried with different sequence lengths as shown in Table 1 for sequence length of 20 and Table 3 for sequence length of 50. It is noticed that for shorter sequences, the generated texts are not coherent in meanings, they look more like isolated words. This could be caused by the fact that when sequence length is low, the model could not learn too much from the contexts. As sequence becomes too larger, it is observed that the model also struggles to generate coherent texts. This could be cause by not enough training data or not enough training epochs, since in this case it needs to learning much more contexts.

Q2.1 c)): From greedy sampling to random sampling

The approach we taken in the previous results so-called greedy sampling, meaning we take the highest probability output as the predicted character. Instead, we could also introduce randomness by constructing a probabilistic multinominal distribution from the output and draw samples from it. This can be achieved by using a temperature parameter τ in a softmax layer. When τ goes to zero, the distribution tends to be deterministic, and when τ increases we get more randomness. Some of the texts generated with different temperatures are shown in Table

Q2.2: Generating texts based on initial seeds

We can train also ask the network to generate texts by giving a longer seed related to the context it was trained on, e.g, "Sleeping beauty is ".

To introduce more variety into the generated text, random sampling with temperature 0.5 is used with sequence length of 40 (including the initial words). The result is shown in Table 5. It was noticed

step 100	K te te te te te te Gt te te te te te te x te te te te te te ” te te te te te te E te te te te te te
step 400	?’ ‘I was the said t me the said the said ve the said the said On the said the said 2 THE THE THE THE TH
step 1200	-who had been to be joy said to him to h Now the search the w ing the world and sa 6 and said the world
step 8200	01 A PANSNER AND THE You will soon see th and said, ‘I am qu g the second thanks @pgles of the second

Table 1: LSTM text generation results at different steps of sequence length 20

step 100	7 nh te te te te te te te te t W te te te te te te te te te t w te te te te te te te te te t Lt te te te te te te te te te “t te te te te te te te te te
step 400	M The was a will a will a will $ said and was the was a will pent and was the was a will a pent and was the was a will a U The was a will a will a will
step 1200	3 was a little son was a littl -the world and said: ‘What wil e way of the soldier was and s zed the world and said: ‘What Chanted to the world and said:
step 8200	l the stone so much as the sto “It was so that the stone so m Hans went into the water and s Marles and the stone so much a ?’ ‘I will soon bring it is to

Table 2: LSTM text generation results at different steps of sequence length 30

step 100	Ote the the the the the the the the the the the th n the the the the the the the the the the the the 0 the the the the the the the the the the the the t the the the the the the the the the the the the Q the the the the the the the the the the the the
step 400	(and the will the with the with the with the with and the will the with the with the with the with and the will the with the with the with the with t re was the with the with the with the with the wit was strenged the with the with the with the with t
step 1200	2. The cook said to the wood and said: ‘If I will X GRETEL THE THE THE STORY THE THE SEN THE W King and said: ‘If I will not still the wood and s Just she was not the wood and said: ‘If I will not X GRETEL THE THE THE STORY THE THE SEN THE W
step 8200	Gretch the maiden was so put on the spot, and said [*] at the straw she was so put on the spot, and s ’ said the old woman, ‘the castle was as if some o ing and said: ‘There were so much to be a doctor t Qhe was a good monst and said: ‘There were so much

Table 3: LSTM text generation results at different steps of sequence length 50

τ = 0.5	No,’ said the king’s daughter,
τ= 1.0	You have cooking.’ ‘What once
τ = 2.0	mothers to Maatil, the bougres

Table 4: LSTM text generation results with random sampling of different temperature parameter

that the sentences generated are more or less grammatically correct, but lack coherence in meanings. Some words appear with high probability like "a", "all" or "the", which is most likely caused by the high frequency of occurrences of these words in the training corpus.

Q2.3: Another sampling: beam search

Q3.1 a): Explain GCN

The question to answer here is: how does GCN incorporate structural relationship information in graph data structure? We first need to understand that an adjacency matrix is a matrix of 0 or 1 to encode node connection relationship. For example, matrix A can be of the following form:

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

would represent a undirected graph with node 1 connected to node 2 and 3. Note that the diagonal elements are all zeros because it does have self connections, one can add a identity matrix to encode self-connection. And the matrix is symmetrical because it is a undirected graph.

The GCN layer is given by:

$$H^{(l+1)} = \sigma(\hat{A}H^{(l)}W^{(l)}) \tag{20}$$

In $H^{(l)}$, its i-th row represents feature for the i-th node in a graph in layer l, which is denoted by $h_i^{(l)}$. We can see that the i-th row of layer l+1 $h_i^{(l+1)}$ will be affected by the i-th row of adjacency matrix

sleeping beauty is the bird is the rope
sleeping beauty is all done.’ And he sai
sleeping beauty is all in the land; so t

Table 5: LSTM text generation results longer initial words

a_i. According to the definition of matrix A, a_i encodes the connection information of the ith node w.r.t. to all other nodes (1 for connected, 0 for disconnected). Therefore, the structual information is captured by using the adjacency matrix here. This process can also been seen as a summation of all neighboring node features, and its propogated to the next layer, through a learnable weight matrix and non-linearity.

Q3.1 b): GCN limitations

There are a few limitation to the simple GCN equation [2]. . One is that the original adjacency matrix does not consider self loops. A simple trick to solve this is to add an identity matrix to the original A. Another problem is that A is not normalized, therefore after multiplication the scale of feature vectors could completely change. This problem can be solved by normalizing the matrix A by its corresponding degree matrix.

A third problem is the difficulty of modelling relationships of nodes that are far away. We could potentially tackle this by adding more layers.

Q3.2 a): Calculate the adjacency matrix

As shown in the graph (see assignment), there are 6 nodes, thus the dimension of the matrix will be of 6 by 6. Since there is not self-connection, diagonal elements will be zeros.

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

Q3.2 b)

The shortest path from C to E is 3. Thus it takes 3 updates to forward the information from node C to node E.

Q3.3: Applications of GNN

As discussed in this comprehensive survey paper [3], there are a variety of applications of graph based neural network in different industries:

- In e-commence, GNN can be utilized to build knowledge graph and model the interations between users and products to produce better recommendations and categorizations.
- In biology, GNN can be used to discover new drugs since molecules have an inherent graph structure.
- In social networks, GNN can be used to model relationships between people.

Q3.4 a): Compare RNN based model to GNN

RNN based models are suitable for Euclidean data such as texts, time series, while GNN are specifically designed for graph structure data in non-euclidean domain which has interlinked dependencies across different nodes. traditional RNN stacks features in a sequence, however this is not natural for graph nodes. To do that, we will have to traverse the whole graph, which is infeasible for large graphs [4] .

Q3.4 b): How and when to combine RNN with GNN?

Graph is more natural to model reasoning. Humans perceive the world of knowledge by connecting different entities. In reality, graphs are not static, they evolve over time. For example, new entities might be added, deleted or updated, so do connections between nodes, a.k.a., relationships.

It is natural to see that if we need to model graph structure data and its dynamics over time, RNN based GNN model could be a promising way to go. One real life example could be knowledge graphs for e-commerce business like Amazon or Alibaba. The products on these platforms are being updated all the time such that it could be represented as a dynamically changing knowledge graph, which could be learned by a RNN based GNN model.

References

[1] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.

[2] Thomas N Kipf and Max Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[3] Zonghan Wu et al. "A Comprehensive Survey on Graph Neural Networks". In: *CoRR* abs/1901.00596 (2019). arXiv: 1901.00596. URL: http://arxiv.org/abs/1901.00596.

[4] Jie Zhou et al. "Graph Neural Networks: A Review of Methods and Applications". In: *CoRR* abs/1812.08434 (2018). arXiv: 1812.08434. URL: http://arxiv.org/abs/1812.08434.