Assignment 2. Recurrent Neural Networks and Graph Neural Networks

Xiaoxiao Wen
12320323
xiaoxiao.wen@student.uva.nl

1 Vanilla RNN versus LSTM

1.1 Toy Problem: Palindrome Numbers

1.2 Vanilla RNN in PyTorch

Question 1.1

Assumption: all vectors are column vectors ($\mathbf{x}$ is of shape $a \times 1$, $\mathbf{h}$ is of shape $b \times 1$ and $\mathbf{p}$ is of shape $c \times 1$) and ifa $\mathbf{a} \in \mathbb{R}^l$ and $A \in \mathbb{R}^{n \times m}$, then $\left[\frac{\partial \mathbf{a}}{\partial A}\right]_{ijk} = \frac{\partial \mathbf{a}_i}{\partial A_{kj}}$, so $\frac{\partial \mathbf{a}}{\partial A} \in \mathbb{R}^{l \times m \times n}$.

Given that $\mathcal{L} = -\sum_{k=1}^{K} \mathbf{y}_k \log \hat{\mathbf{y}}_k$, we can write its matrix form as

$$\mathcal{L} = -\mathbf{y}^T \log \hat{\mathbf{y}}$$

Then we also know

$$\frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} = -(\mathbf{y}^{(T)})^T \oslash (\hat{\mathbf{y}}^{(T)})^T$$

where $\oslash$ is the element-wise division.

According to the derivation in assignment 1, we know the gradient of softmax is as follows:

$$\begin{aligned}\frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} &= \frac{\partial \text{softmax}(\mathbf{p}^{(T)})}{\partial \mathbf{p}^{(T)}} \\ &= \text{diag}(\text{softmax}(\mathbf{p}^{(T)})) - \text{softmax}(\mathbf{p}^{(T)})\text{softmax}(\mathbf{p}^{(T)})^T\end{aligned}$$

Then we have

$$\begin{aligned}\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}} &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{W}_{ph}} \\ &= -(\mathbf{y}^{(T)})^T \oslash (\hat{\mathbf{y}}^{(T)})^T [\text{diag}(\text{softmax}(\mathbf{p}^{(T)})) - \text{softmax}(\mathbf{p}^{(T)})\text{softmax}(\mathbf{p}^{(T)})^T] \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{W}_{ph}}\end{aligned}$$

where $\frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{W}_{ph}}$ is a tensor of shape $c \times b \times c$ with $\frac{\partial \mathbf{p}_m^{(T)}}{\partial \mathbf{W}_{ph,nl}} = \mathbf{h}_l^{(T)} \delta_{mn}$.

Preprint. Work in progress.

Let $\mathbf{z}^{(t)} = \mathbf{W}_{hx}\mathbf{x}^{(t)} + \mathbf{W}_{hh}\mathbf{h}^{(t-1)} + \mathbf{b}_h$, we have

$$\begin{aligned}
\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}} &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{h}^{(T)}} \sum_{k=0}^{T} \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(k)}} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hh}} \\
&= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{h}^{(T)}} \sum_{k=0}^{T} \left(\prod_{j=k+1}^{T} \frac{\partial \mathbf{h}^{(j)}}{\partial \mathbf{h}^{(j-1)}} \right) \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{z}^{(k)}} \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{W}_{hh}} \\
&= (\mathbf{y}^{(T)})^T \oslash (\hat{\mathbf{y}}^{(T)})^T [\mathrm{softmax}(\mathbf{p}^{(T)}) \mathrm{softmax}(\mathbf{p}^{(T)})^T - \mathrm{diag}(\mathrm{softmax}(\mathbf{p}^{(T)}))] \mathbf{W}_{ph} \\
&\sum_{k=0}^{T} \left(\prod_{j=k+1}^{T} \mathbf{W}_{hh} \right) \mathrm{diag}(\mathbf{1}^{b \times 1} - \tanh^2(\mathbf{z}^{(k)})) \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{W}_{hh}}
\end{aligned}$$

where $\frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{W}_{hh}}$ is a tensor of shape $b \times b \times b$ with $\frac{\partial \mathbf{z}_m^{(k)}}{\partial \mathbf{W}_{hh,nl}} = \mathbf{h}_l^{(k-1)} \delta_{mn}$.

Similar to $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}}$, we have

$$\begin{aligned}
\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hx}} &= \frac{\partial \mathcal{L}^{(T)}}{\partial \hat{\mathbf{y}}^{(T)}} \frac{\partial \hat{\mathbf{y}}^{(T)}}{\partial \mathbf{p}^{(T)}} \frac{\partial \mathbf{p}^{(T)}}{\partial \mathbf{h}^{(T)}} \sum_{k=0}^{T} \frac{\partial \mathbf{h}^{(T)}}{\partial \mathbf{h}^{(k)}} \frac{\partial \mathbf{h}^{(k)}}{\partial \mathbf{W}_{hx}} \\
&= (\mathbf{y}^{(T)})^T \oslash (\hat{\mathbf{y}}^{(T)})^T [\mathrm{softmax}(\mathbf{p}^{(T)}) \mathrm{softmax}(\mathbf{p}^{(T)})^T - \mathrm{diag}(\mathrm{softmax}(\mathbf{p}^{(T)}))] \mathbf{W}_{ph} \\
&\sum_{k=0}^{T} \left(\prod_{j=k+1}^{T} \mathbf{W}_{hh} \right) \mathrm{diag}(\mathbf{1}^{b \times 1} - \tanh^2(\mathbf{z}^{(k)})) \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{W}_{hx}}
\end{aligned}$$

where $\frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{W}_{hx}}$ is a tensor of shape $b \times a \times b$ with $\frac{\partial \mathbf{z}_m^{(k)}}{\partial \mathbf{W}_{hx,nl}} = \mathbf{x}_l^{(k)} \delta_{mn}$.

As derived above, for $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{ph}}$ only depends on the current timestep T while $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hh}}$ and $\frac{\partial \mathcal{L}^{(T)}}{\partial \mathbf{W}_{hx}}$ depends on all the timesteps from 0 to T. For the latter two gradients, according to the term $\prod_{j=k+1}^{T} \mathbf{W}_{hh}$, the product of the term $\mathbf{W}_{hh}$ for larger number of timesteps would either cause exploding gradients or vanishing gradients, depending on whether the elements of $\mathbf{W}_{hh}$ are larger or smaller than 1.

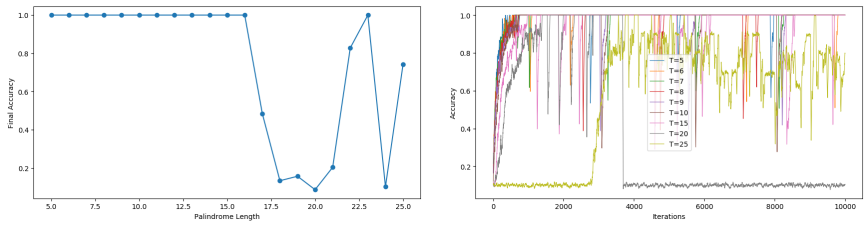
Question 1.2

The `vanilla_rnn.py` is manually implemented as required using the equations given in the manual. All parameters are initialized with `xavier_normal` initialization for better training performance [1]. As required, only the output of the last timestep is returned. Furthermore, the `train.py` is completed for training the vanilla RNN module.

Detailed implementations can be seen in the code.

Question 1.3

The vanilla RNN is trained with palindromes of length starting from 5 to 25, and the obtained results are shown in Figure 1. According to the final accuracy plot Figure 1a, it can be observed that with increasing length of the palindrome, the training performance of RNN starts to decrease. Specifically, after training with palindromes of length 17, the final accuracy of the training drops drastically and becomes unrobust for further increasing the length. According to the training accuracy plot Figure 1b, the training performance over all the iterations is steady without perturbations for $T \leq 15$. However, starting from $T = 20$, the training accuracy either cannot reach near-perfect or drops after some iterations from near-perfect, which also indicates the weak robustness of the vanilla RNN network for longer palindromes.



(a) **(b)**

Figure 1: *Plots of final accuracy (left) and training accuracy (right) (with a running average window size of 21) for training with different palindrome lengths for vanilla RNN.*

Question 1.4

During the training of neural networks, a notorious problem is the ravines or pathological curvatures in the hyperspace of the loss functions, where the gradient descent is much steeper in some dimensions than the other ones. When using vanilla SGD for optimization, the gradient descent updates show an oscillating behaviour between the ridges of the ravine, and the update is hampered with respect to its relevant direction.

SGD with momentum In order to overcome this problem, the first novel idea is to extend the vanilla SGD with momentum [2], as shown by the following update equations

$$\nu^{(t)} \leftarrow \eta \cdot \nu^{(t-1)} - \alpha \cdot \nabla_w^{(t)} \sum_{m \in B} \mathcal{L}_m$$
$$w^{(t)} \leftarrow w^{(t-1)} + \nu^{(t)}$$

where ν is the gradient update, η is the coefficient for the retained gradient called coeffient momentum, α is the coefficient for the current gradient descent, w is the parameter and $\mathcal{L}_m$ is the loss function for batch m.

Compared to the vanilla SGD, instead of directly updating the parameters with the gradient descent in the current timestep, the SGD with momentum first compute the gradient update ν out of two parts: retained gradient(s) from previous timestep(s) and the current gradient descent. By retaining gradients with the coefficient of momentum from previous timesteps, the more recent gradients are weighed more than earlier ones, showing a exponentially decreasing weighing for earlier gradients. This helps in the case of pathological curvatures as the oscillations of the vanilla SGD occur between the ridges of the ravine, and, thus, if summed up, the components of the gradient updates tangent to the ridges cancel out, leaving valid components only in the relevant direction, so the oscillations are dampened. This indicates that, by retaining gradients from previous steps, the gradient update can be less influenced by the ridges of the ravines and converge faster.

RMSProp Another idea to tackle the problem of pathological curvature is using the RMSProp optimizer [3]. The RMSProp stands for Root Mean Square Propagation, where instead of using momentum to dampen oscillations, adaptive learning rate is implemented via retaining the square values of the gradients for each parameter. Formally, RMSProp is defined by the following update equations for each parameter w_i

$$\nu^{(t)} \leftarrow \rho \nu^{(t-1)} + (1-\rho)(\frac{1}{B} \sum_{m \in B} \frac{\partial \mathcal{L}_m^{(t)}}{\partial w_i})^2$$
$$w_i^{(t)} \leftarrow w_i^{(t-1)} - \frac{\eta}{\sqrt{\nu^{(t)} + \epsilon}} \cdot \frac{1}{B} \sum_{m \in B} \frac{\partial \mathcal{L}_m^{(t)}}{\partial w_i}$$

where ν is the exponential average squares of the gradients, ρ is the coefficient for the exponential averages squares of the gradients, η is the initial learning rate and ϵ is a small constant to prevent division by 0.

Compared to the vanilla SGD, RMSProp makes use of the squares of the gradients, which, similar to SGD with momentum, shows an exponentially decreasing weighing for earlier squares of gradients compared to more recent ones. Then the parameters are updated with a learning rate/update step size dependent on the exponential average squares of gradients, which adapts and normalizes the learning rate/update step size. Because this is computed for each parameter w_i separately, it helps to regulate and dampen oscillations in the directions where the ridges are and amplifies the update step size in the relevant directions, and, hence, it drives the gradient update to escape the pathological curvatures. Simultaneously, the adaptive learning rate also helps to prevent overshoot while the gradient update approaches the minima.

Adam The Adam, or Adapative Moment Optimization, optimizer combines both the momentum method from SGD with momentum and the root mean square method from RMSProp [4]. It uses both the exponential average of the gradients and the exponential average of the squares of the gradients to compute the update, and adapts the learning rate accordingly. It is simply defined as the collection of the aforementioned two methods. For each parameter w_i, we have

$$\nu^{(t)} \leftarrow \beta_1 \nu^{(t-1)} + (1-\beta_1)\frac{1}{B}\sum_{m\in B}\frac{\partial \mathcal{L}_m^{(t)}}{\partial w_i}$$

$$s^{(t)} \leftarrow \beta_2 s^{(t-1)} + (1-\beta_2)(\frac{1}{B}\sum_{m\in B}\frac{\partial \mathcal{L}_m^{(t)}}{\partial w_i})^2$$

$$w_i^{(t)} \leftarrow w_i^{(t-1)} - \eta\frac{\nu^{(t)}}{\sqrt{s^{(t)}+\epsilon}}\cdot\frac{1}{B}\sum_{m\in B}\frac{\partial \mathcal{L}_m^{(t)}}{\partial w_i}$$

where ν and s are the exponential average of the gradients and the squares of gradients respectively, and β_1, β_2 are their corresponding coefficients.

Compared to the vanilla SGD, Adam has the advantages of both SGD with momentum and RMSProp, where gradients history helps to drive the gradient updates to escape from the pathological curvatures and adapt the learning rate properly.

1.3 Long-Short Term Network (LSTM) in PyTorch

Question 1.5

(a)

- The *input modulation gate* $\mathbf{g}^{(t)}$ is defined as

$$\mathrm{g}^{(t)} = \tanh\left(\mathbf{W}_{gx}\mathbf{x}^{(t)} + \mathbf{W}_{gh}\mathbf{h}^{(t-1)} + \mathbf{b}_g\right)$$

  It is used to determine the new candidate values, based on the previous hidden state values and the input values, to add to the cell state values. The activation layer used for the input modulation gate is $\tanh$, which has the range $[-1, 1]$ and is centered at 0. As can be observed from the equation

$$\mathbf{c}^{(t)} = \mathbf{g}^{(t)} \odot \mathbf{i}^{(t)} + \mathbf{c}^{(t-1)} \odot \mathbf{f}^{(t)}$$

  the output of the input modulation gate is element-wise multiplied with the output of the input gate (sigmoid activation so does not change the range) and added to the cell state values. Assuming the cell state values are centered around 0 and have the range $[-1, 1]$, after applying the forget gate $\mathbf{f}^{(t)}$, the output is still zero-centered having the range $[-1, 1]$, and only if the output of the input modulation gate has the same property, will the property be maintained for the final cell state values, which is desired to prevent non-zero offset of the cell state values and to prevent from exploding or vanishing cell state values.

- The *input gate* $\mathbf{i}^{(t)}$ is defined as

$$\mathbf{i}^{(t)} = \sigma\left(\mathbf{W}_{ix}\mathbf{x}^{(t)} + \mathbf{W}_{ih}\mathbf{h}^{(t-1)} + \mathbf{b}_i\right)$$

  It is used to determine which values among the new candidate values to update, based on the previous hidden state values and the input values. It uses sigmoid activation such

that the output is of the range $[0, 1]$, where for each value among the new candidate values 0 indicates that the value is abandoned and 1 indicates that the value is to be updated.

- The *forget gate* $\mathbf{f}^{(t)}$ is defined as

$$\mathbf{f}^{(t)} = \sigma\left(\mathbf{W}_{fx}\mathbf{x}^{(t)} + \mathbf{W}_{fh}\mathbf{h}^{(t-1)} + \mathbf{b}_f\right)$$

  It is used to determine which values among the previous cell state values are abandoned, based on the previous hidden state values and the input values. Similar to the input gate, the sigmoid activation helps to determine which values to keep and which values to abandon from the previous cell state values.

- The *output gate* $\mathbf{o}^{(t)}$ is defined as

$$\mathbf{o}^{(t)} = \sigma\left(\mathbf{W}_{ox}\mathbf{x}^{(t)} + \mathbf{W}_{oh}\mathbf{h}^{(t-1)} + \mathbf{b}_o\right)$$

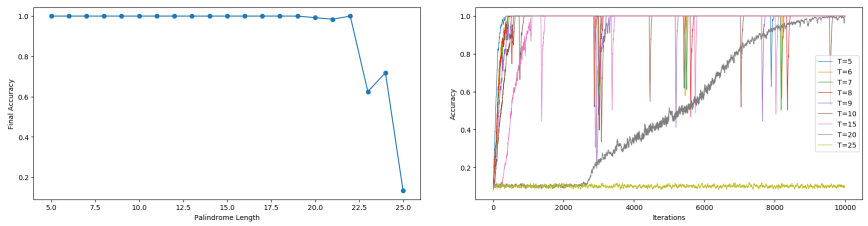
  It is used to determine which values among the new cell state values (filtered with $\tanh$) to output. Similarly, sigmoid is used in the same manner to keep or abandon information.

(b) Given that $\mathbf{x} \in \mathbb{R}^{T\times d}$ and there are n units in the LSTM and the batch size is m, only including the core part of LSTM and excluding the linear output mapping, we know that $\mathbf{W}_{gx}$, $\mathbf{W}_{ix}$, $\mathbf{W}_{fx}$ and $\mathbf{W}_{ox}$ are of shape $d \times n$, $\mathbf{W}_{gh}$, $\mathbf{W}_{ih}$, $\mathbf{W}_{fh}$ and $\mathbf{W}_{oh}$ are of shape $n \times n$. Furthermore, $\mathbf{b}_g$, $\mathbf{b}_i$, $\mathbf{b}_f$ and $\mathbf{b}_o$ are of shape $1 \times n$. Hence, the total number of trainable parameters in LSTM's core part is $4 \times (d \times n + n \times n + n)$.

Question 1.6

The LSTM network is trained with palindromes of length starting from 5 to 25, and the obtained results are shown in Figure 2. According to the final accuracy plot Figure 2a, it can be observed that with increasing length of the palindrome, the training performance remains at the same level for most of the tested palindrome lengths, and only starts to decrease for the largest ones. Specifically, after training with palindromes of length 23, the final accuracy of the training drops to ~ 0.6 and becomes unrobust for further increasing the length. According to the training accuracy plot Figure 2b, the training performance over all the iterations is steady without perturbations for $T \leq 20$, where the learning rates are decreased for training with lengths ≥ 20. For $T = 25$ however, the training accuracy is saturated at ~ 0.1 and does not increase at all.

Compared to the vanilla RNN, the LSTM network shows faster and more robust training performance when training with palindromes of lengths < 20, as shown in Figure 1b and Figure 2b. Also, the allowed palindrome length (with proper final accuracy and robustness) for LSTM is larger compared to the vanilla RNN, which is indicated by Figure 1a and Figure 2a. However, with further increased lengths, the LSTM network still cannot guarantee a robust training performance which can achieve near-perfect accuracy.



(a) **(b)**

Figure 2: *Plots of final accuracy (left) and training accuracy (right) (with a running average window size of 21) for training with different palindrome lengths for LSTM.*

2 Recurrent Nets as Generative Model

Question 2.1

(a) The two-layer LSTM network is implemented using the high-level API of PyTorch `nn.Module` where it is supported to stack the LSTM cells to ease the implementation. As for training dataset, the movie scripts of *Star Wars: Trilogy* and *Star Wars: Prequel Trilogy* are used [5]. Apart from the default implementation requirements, a step-wise learning rate decay is also implemented to train for larger number of epochs.

Detailed implementations can be seen in the code.

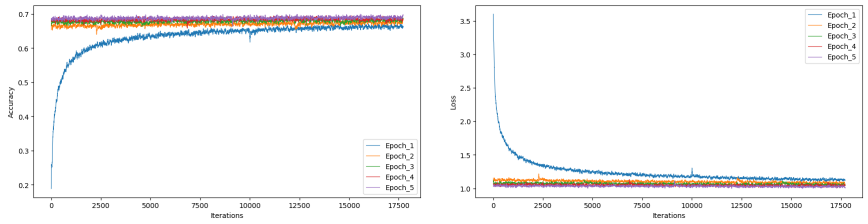
(b) The training of the network is performed for 30 epochs on the dataset, but as shown in Figure 3, the training accuracy and loss converge in the first epoch and only slightly increases during further training. Hence, 5 sample sentences generated every 1000 iterations in the first epoch is used to observe and analyze the quality of the generation, which approximately covers the process of training the network from the beginning until near convergence.

As can be read from Listing 1, at Step 1000, the generated texts already show correct spelling of words, i.e. "What", "starts", but only limited grammatical correctness can be observed and the lexical meanings of the words are not properly associated either. At Step 2000, the generated texts start to show alignments and associations of lexically close words, for instance, "ship" and "heads", though the grammatical correctness is still lacking. At Step 3000, it is shown that both grammatical syntax and the lexical meanings of the words are properly captured, like "JAR JAR : I would be the ship.", which also shows a correct punctuation of the generated text. At Step 4000, the generated texts show limited information because of the whitespace characters generated, yet it can still be observed that aside from sentences, titles and bridges in the original scripts are learned, for example "(into comlink)" and "SENATOR BATTLE DROIDS". At Step 5000, the generative model already shows that some word embeddings are learned, which is reflected by the text "Luke and Leia", as in the original scripts words "Luke" and "Leia" frequently co-occur.

Listing 1: *Generated sample texts every 1000 iterations during the first epoch.*

```
    Epoch: 1
    Step: 1000
    6
    ZAN : What is a stands and the
    n the starts and the starts an
    ING ARTARA - DAY

    The ship sta
    1

    Epoch: 1
    Step: 2000
    VERING PLATFORM-DAY

    ANAKIN s
    ke the ship and heads and star
    ing the ship and heads and sta
    y the ship and heads and start
    VERING PLATFORM-DAY

    ANAKIN s

    Epoch: 1
    Step: 3000
    PADME stands and starts to the
    ze and the starts to the ship.
    PADME stands and starts to the
    xit the ship.

    PALPATINE: I w
    JAR JAR : I would be the ship.

```

```
    Epoch: 1
    Step: 4000
    ; the ship and the starts to t
    HAN
            (into comlink)
         The Ch
    ke the ship.

    OBI-WAN: The sh
    8
    4 SENATOR BATTLE DROIDS stand

    Epoch: 1
    Step: 5000
    ; the ship stands around the s
    : I don't know where you can d
    E - DAY

    The ship stands in th
    Luke and Leia and Leia and Lei
    ing the ship and the starts to
```



(a) **(b)**

Figure 3: *Training accuracy (left) and training loss (right) (with a running average window size of 21) during different epochs for the generative LSTM model.*

(c) The temperature parameter is implemented via dividing the input logits output by the generative model to the softmax function by τ, such that

$$\hat{\mathbf{y}}^{(t)} = \text{softmax}(\mathbf{p}^{(t)}/\tau)$$

Theoretically, as $\tau \to 0$, the probability of the peak element is further amplified and, therefore, the softmax tends to be max operation, where the sampling becomes as greedy as possible. On the other hand, when $\tau \to \infty$, the probability of all elements are adjusted to be uniform and, therefore, the sampling tends to be uniform sampling where the variance of the sampling increases.

As shown in Listing 2, when $\tau = 0.5$, the texts are generated more greedily and the contents resemble more to the original texts that are learned by the model. Also, in the five sample texts, the same pattern is followed and generated, given the same starting character, which also reflects the greediness. For $\tau = 1$, some variances are introduced, as shown by the fact that the phrase "the ship" occurs in different contexts instead of only matching a certain pattern. For $\tau = 2$, more variances are introduced compared to the case of $\tau = 1$, and it can be opbserved that phrases like "(into comlink)", "MILLENIUM FALCON" and such, appear in different contexts. Also, innovation and novelty comes with the introduction of variance, which makes the generated samples creative compared to the original dataset.

Listing 2: *Generated sample texts for different temperature.*

```
    Temperature: 0.5
    Sample length: 30
    ing the ship.  The

    !
```

```
6
7       OBI-WAN: I will not be des
8       COCKPIT
9
10                    Luke l
11      He starts to lift the control
12
13
14
15      : I will not be difficult to p
16      !
17
18      OBI-WAN: I will not be des
19
20      ZARD and the JEDI stand before
21      just a control panel.
22
23      Temperature: 1.0
24      Sample length: 30
25      N : I want to see your father
26      INT. MILLENNIUM FALCON - COCKP
27      QUI-GON : I don't know what yo
28      OBI-WAN stands before the ship
29      ; the Force is strong with the
30      XT. SPACE - MILLENNIUM FALCON
31      OBI-WAN stands before the ship
32      re the ship begins to
33
34      ANAKIN : (Cont'd) What do you
35      The ship shudders as they can
36      Temperature: 2.0
37      Sample length: 30
38      DER
39                               I
40      8   INT MILLENNIUM FALCON - CO
41      XT. SPACE - MILLENNIUM FALCON
42      ze starts to run and flips off
43      You are still alive.
44
45      ANAKIN:
46      ll be a problem. I'm not going
47      ?
48
49                 LANDO
50              (into comlink)
51      WAN : I want to go home. It's
52      's ship appears and
53
54                 LUKE
55              (into comlink)
56           A
```

Bonus Question 2.2

As shown in Listing 3, with randomly initialized start characters, longer texts can be sampled with a resonably good quality, where the lexical meanings of the words are properly captured such that the phrases and expressions are legit, the structure of the orignal dataset is learned (dialogues) and the syntax correctness is also guaranteed.

Listing 4 shows the generated sample texts with certain beginnings. Some typical and symbolic lines from the original script are used as the measure, for example "I have the high ground.", "I am your father." and "May the force be with you.", and the generative model is tested whether it can complete these sentences only given the first few characters or words. According to the results, the generative

model has an acceptable performance as it finishes these typical sentences from the original dataset, given the prefixes "I have the", "I am your" and "May the". However, it can also be observed that the model learns a large number of whitespace characters which are used for formatting in the original dataset. This shows that the model is not able to distinguish between generating actual content in the samples and generating structures or formats. Furthermore, the model lacks variance in terms of content. Given the same prefix or the same initial characters, the model is only able to complete the sentences in the same pattern. This diminishes the creativity of the generative model.

Listing 3: *Generated sample texts for longer lengths.*

```
Temperature: 1.0
Sample length: 60
JAR JAR : What is it?
QUI-GON : I want to see your father.

Temperature: 1.0
Sample length: 120
OBI-WAN: I will not be destroyed and the Chancellor has been seconded
    by the Force.

ANAKIN: I will not be destroyed

Temperature: 1.0
Sample length: 180
(to Artoo)
                                                I don't know what I have
                                                    a bad feeling about
                                                    this.
```

Listing 4: *Generated sample texts for completing sentences.*

```
Temperature: 1.0
Sample length: 60
I have the high ground

Temperature: 2.0
Sample length: 60
I am your father.


Temperature: 2.0
Sample length: 60
May the Force be with you.

ANAKIN: I will not be destroyed.
```

3 Graph Neural Networks

3.1 GCN Forward Layer

Question 3.1

(a) As can be observed in the equation

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

The adjacency matrix considering self-connections $\hat{A}$ encode information of the connections between the pairs of vertices in the graph, which is the main structural information of the graph. The product of $\hat{A}H^{(l)}W^{(l)}$ contains two processes: first, the multiplication between the adjacency matrix $\hat{A}$ and the feature matrix $H^{(l)}$ aggregates the feature information of the adjacent nodes for each node in the graph; secondly, the aggregated feature information for each node in the graph is transferred into new features/latent variables by multiplying

the learnable matrix $W^{(l)}$. Finally, these new features are mapped to a range of $[0, 1]$ by applying the sigmoid activation. Hence, as the feature representations of each node is aggregated and transferred according to the adjacency information, the GCN layer can be seen as performing message passing over the graph from the adjacent nodes to the node in question.

(b) To propagate every node's embedding information from nodes 3 hops away in the graph, 3 GCN layers are needed at most, assuming no cycles in the graph.

3.2 Applications of GNNs

Question 3.2

Physics: Interaction Networks[6], Visual Interaction Networks[7]

Chemistry: Molecular Fingerprints[8], Protein Interface Prediction[9]

3.3 Comparing and Combining GNNs and RNNs

Question 3.3

(a) By intuition, for sequential data with time series like music that mainly rely on the time dependency, RNN-based models are expected to perform better. This is because RNN-based models are designated for capturing short-term relations between data in sequences. For this type of data, although there may be some structural dependency between data that are discontinuous in time, most of the adjacent data have strong and direct relation with each other, so the sequential representation is more expressive.

On the other hand, for strongly structural data like shape of proteins, mainly having relations between data that cannot be unrolled in a linear or sequential dimension, the GNNs are expected to have better performance. This is because GNNs are designated for graph-like structured dataset and can perform message pass over the graph. For this type of data, although there may be some sequential or temporal dependency between neighboring data, the main influence and dependency arise from the non sequential connections, so the graph representation is more expressive.

(b) GNNs can substitute CNNs for some image classification tasks, given that the dynamic (small or large) spatial correlations between pixels in a image. Hence, an interesting application of the combined model of GNNs and RNNs would be the video captioning task, where GNNs are used to extract visual features from the frames and RNNs are used to capture the sequential relation between the frames to capture motions.

References

[1] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," mar 2010.

[2] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, oct 1986.

[3] G. Hinton, N. Srivastava, and K. Swersky, "Neural Networks for Machine Learning Lecture 6a Overview of mini--batch gradient descent," tech. rep.

[4] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," dec 2014.

[5] "Star Wars: A New Hope Script at IMSDb.."

[6] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, "Interaction Networks for Learning about Objects, Relations and Physics," dec 2016.

[7] N. Watters, A. Tacchetti, T. Weber, R. Pascanu, P. Battaglia, and D. Zoran, "Visual Interaction Networks," jun 2017.

[8] D. Duvenaud, D. Maclaurin, J. Aguilera-Iparraguirre, R. Gómez-Bombarelli, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional Networks on Graphs for Learning Molecular Fingerprints," sep 2015.

[9] A. Fout, B. Shariat, J. Byrd, and A. Ben-Hur, "Protein Interface Prediction using Graph Convolutional Networks," in *NIPS*, no. Nips, pp. 6512–6521, 2017.