Assignment 2. Recurrent Neural Networks and Graph Neural Networks

Victor Zuanazzi
12325724
University of Amsterdam
victorzuanazzi@gmail.com

1 Vanilla RNN versus LSTM

1.1 Toy Problem: Palindrome Numbers

1.2 Vanilla RNN in PyTorch

1.2.1 Question 1.1

It is required to derive $\frac{\partial L}{\partial W_{ph}}$ and $\frac{\partial L}{\partial W_{hh}}$.

It is easier to derive the Loss of each class first. Later we can re-write it in matrix form.

$$L_k = y_k \log \hat{y}_k \tag{1}$$

$$\frac{\partial L_k}{\partial W_{ph}} = \frac{\partial}{\partial W_{ph}} y_k \log \hat{y}_k \tag{2}$$

$$= \frac{\partial y_k}{\partial W_{ph}} \log y_k + \frac{y_k}{\hat{y}_k} \frac{\partial \hat{y}_k}{\partial W_{ph}} \tag{3}$$

$$= 0 \log y_k + \frac{y_k}{\hat{y}_k} \frac{\partial \hat{y}_k}{\partial W_{ph}} = \frac{y_k}{\hat{y}_k} \frac{\partial \hat{y}_k}{\partial W_{ph}} \tag{4}$$

In eq 4 $\frac{\partial y_k}{\partial W_{ph}} = 0$ because the true labels do not depend on the parameters of our model. For the following derivations, I use the results derived in the previous assignments.

$$\frac{\partial \hat{y}_k}{\partial W_{ph}} = \frac{\partial \hat{y}_k}{\partial p_k^{(T)}} \frac{\partial p_k^{(T)}}{\partial W_{ph}} = \sum_s \hat{y_k}(I_{sk} - \hat{y_s}) \begin{bmatrix} 0 & & \\ h_1^{(T)} & \dots & h_K^{(T)} \\ & 0 & \end{bmatrix} \tag{5}$$

$$\Rightarrow \frac{y_k}{\hat{y}_k} \frac{\partial \hat{y}_k}{\partial W_{ph}} = \frac{y_k}{\hat{y}_k} \sum_s \hat{y_k}(I_{sk} - \hat{y_s}) \begin{bmatrix} 0 & & \\ h_1^{(T)} & \dots & h_K^{(T)} \\ & 0 & \end{bmatrix} \tag{6}$$

$$= y_k \sum_s (I_{sk} - \hat{y_k}) \begin{bmatrix} 0 & & \\ h_1^{(T)} & \dots & h_K^{(T)} \\ & 0 & \end{bmatrix} \tag{7}$$

$$= y_k \sum_s (I_{sk} - \hat{y_k}) \begin{bmatrix} 0 \\ (\boldsymbol{h}^{(T)})^T \\ 0 \end{bmatrix} \tag{8}$$

Where $^{(T)}$ denotes the last time step and ^{T} the transpose operation. $\boldsymbol{h} = [h_1 ... h_K]^T$. And I_{sk} is the indicator function, $I_{sk} = 1$ if $s = k$ and $I_{sk} = 0$ otherwise.

Submitted to the Spring 2019 Amsterdam Deep Learning course

In the matrix of eq 8 all entries are zero and only the kth row contain the elements of the vector $\boldsymbol{h}^T$. By stacking $y_k(1-\hat{y_k})$ in a vector we can re-write eq 8 as follows.

$$\frac{\partial L}{\partial W_{ph}} = -\sum_k \frac{\partial L_k}{\partial W_{ph}} = -\sum_k \sum_s y_k(1-\hat{y}_s) \begin{bmatrix} 0 \\ (\boldsymbol{h}^{(T)})^T \\ 0 \end{bmatrix} \tag{9}$$

$$= -\sum_s (y_s - \hat{y}_s) \begin{bmatrix} 0 \\ (\boldsymbol{h}^{(T)})^T \\ 0 \end{bmatrix} \tag{10}$$

$$= \begin{bmatrix} \hat{y}_1 - y_1 \\ \vdots \\ \hat{y}_k - y_k \end{bmatrix} (\boldsymbol{h}^{(T)})^T \tag{11}$$

$\frac{\partial L}{\partial W_{ph}}$ only required information about the time step T, however $\frac{\partial L}{\partial W_{hh}}$ will recursively depend on the previous time step. Having that.

$$\frac{\partial L}{\partial W_{hh}} = -\sum_k \frac{\partial L_k}{\partial W_{hh}} \tag{12}$$

$$\frac{\partial p^T}{\partial W_{hh}} = \frac{\partial p^T}{\partial h^T}\frac{\partial h^T}{\partial W_{hh}} = W_{ph}\frac{\partial h^T}{\partial W_{hh}} \tag{13}$$

We can derive:

$$\frac{\partial h^T}{\partial W_{hh}} = \frac{\partial h^T}{\partial W_{hh}} + \frac{\partial h^T}{\partial h_k^{T-1}}\frac{\partial h^{T-1}}{\partial W_{hh}} \tag{14}$$

$$\frac{\partial h^{T-1}}{\partial W_{hh}} = \frac{\partial h^{T-1}}{\partial W_{hh}} + \frac{\partial h^{T-1}}{\partial h_k^{T-2}}\frac{\partial h^{T-2}}{\partial W_{hh}} \tag{15}$$

$$\frac{\partial h^{T-2}}{\partial W_{hh}} = \frac{\partial h^{T-2}}{\partial W_{hh}} + \frac{\partial h^{T-2}}{\partial h_k^{T-3}}\frac{\partial h^{T-3}}{\partial W_{hh}} \tag{16}$$

$$\vdots \tag{17}$$

$$\frac{\partial h^T}{\partial W_{hh}} = \frac{\partial h_k^T}{\partial W_{hh}} + \frac{\partial h^T}{\partial h^{T-1}}\frac{\partial h^{T-1}}{\partial W_{hh}} + \frac{\partial h^T}{\partial h^{T-1}}\frac{\partial h^{T-1}}{\partial h^{T-2}}\frac{\partial h^{T-2}}{\partial W_{hh}} + \dots \tag{18}$$

By recursively extending eq 18 until the very first time step we will have the following sum of products, eq 19:

$$\frac{\partial h^T}{\partial W_{hh}} = \sum_{t=0}^{T}\prod_{j=1}^{t}\frac{\partial h^{T-j+1}}{\partial h^{T-j}}\frac{\partial h^{T-t}}{\partial W_{hh}} \tag{19}$$

Where, the product is defined as 1 for $t = 0$. We can proceed now to make the derivatives explicit. For $\frac{\partial h^{T-j+1}}{\partial h^{T-j}}$ we have:

$$\frac{\partial}{\partial x}\tanh x = 1 - \tanh^2 x \tag{20}$$

$$\frac{\partial h^t}{\partial h^{t-1}} = \frac{\partial h^t}{\partial \hat{x}^t}\frac{\partial \hat{x}^t}{\partial h^{t-1}} \tag{21}$$

$$= \operatorname{diag}\left(1 - \tanh^2(\hat{x}^t)\right) W_{hh} \tag{22}$$

$$= \operatorname{diag}\left(1 - (h^t)^2\right) W_{hh} \tag{23}$$

Where $\hat{x}^t = W_{xh}x^t + W_{hh}h^{t-1} + b_h$, the linear transformation at time step t. And $\operatorname{diag}(1-(h^t)^2) \in \mathbb{R}^{K\times K}$ is necessary to make the matrix multiplications possible. And for $\frac{\partial h^{T-t}}{\partial W_{hh}}$ we have:

$$\frac{\partial h^t}{\partial W_{hh}} = \operatorname{diag}\left(1 - (h^t)^2\right)\frac{\partial \hat{x}^t}{\partial W_{hh}} \tag{24}$$

Where $\frac{\partial \hat{x}^t}{\partial W_{hh}} \in \mathbb{R}^{K \times K \times K}$ being the tensor derived in the first assignment.

The two derivatives have very different dynamics. $\frac{\partial L}{\partial W_{ph}}$ is only dependent of the inputs of the current time step, in that special case, the very last time step. Thus it does not receive any contribution from the inputs and hidden states of previous time steps. That is not the case for $\frac{\partial L}{\partial W_{hh}}$, the gradient of time step t receives contributions of all inputs and hidden states of previous time steps. The contribution of each time step is explicit in the eq 19 by a sum of products. The gradient update is stable when the gradients of each time step $\prod_{j=1}^{t} \frac{\partial h^{T-j+1}}{\partial h^{T-j}} \frac{\partial h^{T-t}}{\partial W_{hh}} \simeq \pm 1$. Then all time steps contribute in a similar way in the sum.

However, the $\tanh(.)$ maps all real numbers to the interval $[-1, 1]$, and the maximum value of its derivative is $\frac{\partial}{\partial 0} \tanh 0 = 1 - \tanh^2(0) = 1$. From eq 18 we can see that the earlier the time step, the more longer is the product, if each term of the product is lower than 1, then the contribution of that time step is exponentially diminished. This phenomena is known as vanishing gradients, in RNNs this is often a problem when dealing with long sequences.

On the other side of the coin we have the exploding gradients. That happens when each factor of the product is higher than 1. By multiplying all of them we will have that the contribution of early time steps is exponentially larger than of later time steps.

1.2.2 Question 1.2

The implementation is attached to this report.

1.2.3 Question 1.3

The implementation of both the RNN and of the LSTM successfuly achieved 100% accuracy for short sequences. The accuracy is calculated at every batch of 128 palindromes. The following results are the best accuracies the models achieved.

The training happens for 10k steps. A moving average of 5 is kept, if it reaches 100% I consider that the model has converged and resume training. The figure 3 shows the best accuracy of the RNN for different lengths. It performs quite well on palindromes of length 10. For sequences longer than 25 the RNN has a performance that is kept around 20%. Which isn't great but shows that the model is using its capacity to the limit, it probably can get some easy palindromes right, but for the majority of the examples it cannot make an adequate guess.

An interesting lesson in how important the weight initialization is. There were some interesting discussions and exchange of ideas in Piazza regarding the topic. The orthogonal initialization Chen et al. [2018] and the Xavier Initialization were brought as good practices.

1.2.4 Question 1.4

Mini-Batch gradient descent is often used to optimize neural networks. Even though it is a good choice, it may lead to slow training or sub-optimal results in some situations. There are three situations where SGD is a ill equipped to carry the optimization.

- *Plateau*: Regions of the parameter space where the derivatives are near zero. Cell points are specially critical. The update of the SGD is proportional to the gradient, when the gradient is near zero the update is also near zero and the neural net cannot learn further.
- *Pathological Curvatures*: Those are regions of the parameter space such as ravines that make the gradient update bounce back and forth and slow down the model's convergence to the actual minimum. The image 1 illustrates the problem.
- *Scale Variance*: The optimal step size for one parameter may be orders of magnitude higher than the optimal step size for another parameter. In addition, in the begin of the training we are usually interested in performing larger updates, but as the model converges we want to perform ever smaller updates to fine tune the parameters. The SGD uses one single learning rate to update all weights in all points of the training.

A number of adaptations of SGD have been proposed, many of based on the *momentum* and *variable learning rate*.

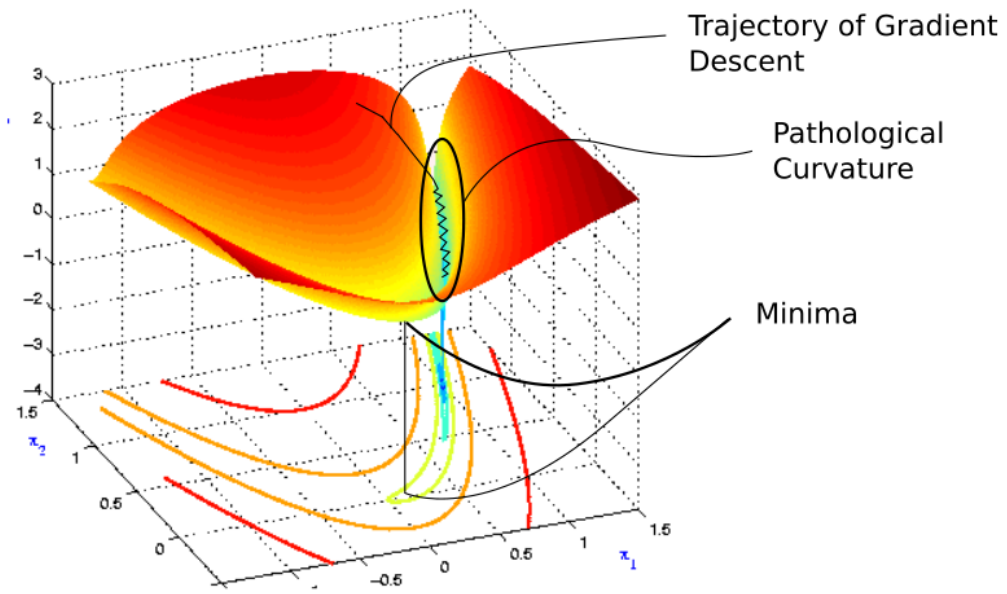

Figure 1: Example of pathological curvature. Image taken from https://mc.ai/intro-to-optimization-in-deep-learning-momentum-rmsprop-and-adam

Momentum is a useful technique to deal with Pathological Curvatures and Plateau. Instead of having one learning rate for all parameters, it dynamically adapts the learning rate for each parameter. The update rule of SGD is modified as follows in eq 26.

$$u_j^{(t+1)} = \gamma^{(t)} u_j^{(t)} - \eta^{(t)} g^{(t)} \tag{25}$$

$$w_j^{(t+1)} = u_j^{(t+1)} + w_j^{(t+1)} \tag{26}$$

Where γ is a hyper parameter usually set to 0.5 in the beginning of the training and increased to 0.9 after some steps. η is the learning rate, notice that the learning rate may change over time but momentum does not offer any form of updates for it. The superscript $a^{(t)}$ denotes the value of a in the time step t and the subscript w_j denotes the $j-th$ weight. u_j denotes weight update of the $j-th$ weight, that is the SGD update added of the momentum update.

The momentum dampens the effect of a ravine and accelerates the updates towards the minima. It is illustrated in the image 2, the updates of $w1$ are dampened and the updates of $w2$ are increased. The weights updated will, over time, approximate the Ideal Path.

In a plateau, where the SGD would get stuck, the momentum uses the previous updates - discounted of $gamma$ - to update the weights. If it is a cell point that is enough to take the neural net to a region where there is gradient again.

Variable learning rate is interesting to speed up training time and still find a fine tuned parameter setting. There are several possibilities to vary the learning rate. Many implementations just divide the learning rate by a factor after a fixed number of steps. Some other implementations decrease the learning rate every time the accuracy starts to increase. Different optimizers have different ways of changing the learning rate.

RSMProp changes the learning rate dynamically for each weight. It indirectly uses the comcept of momentum. The RSMProp update rule is as follows in eq 29.

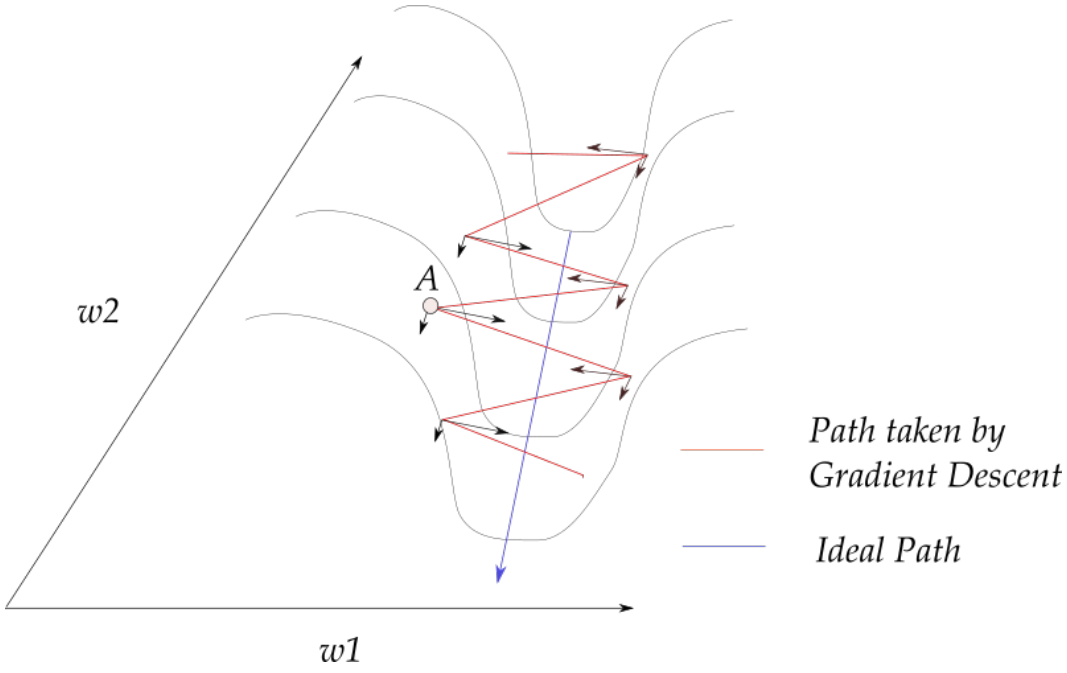

Figure 2: Illustration of sub-optimal behavior of Gradient Descent in a ravine in comparison to the ideal behavior. https://mc.ai/intro-to-optimization-in-deep-learning-momentum-rmsprop-and-adam

$$r_j^{(t+1)} = \gamma^{(t)} r_j(t) + (1 - \gamma^{(t)})(g_j^{(t)})^2 \tag{27}$$

$$u_j^{(t+1)} = -frac\eta^{(t)} \sqrt{r_j^{(t+1)}} + \epsilon g_j^{(t)} \tag{28}$$

$$w_j^{(t+1)} = \eta^{(t)} u_j^{(t+1)} + w_j^{(t)} \tag{29}$$

Where η is the original learning rate, $r_j^{(t)}$ is the exponential average of square gradients of the jth weight at time step t, and $g_j^{(t)}$ is the gradient of w_j at time step t. RSMProp uses a variable learning rate by dividing the original one by $\sqrt{r_j^{(t+1)}} + \epsilon$.

RSMProp uses the principle of momentum in a different way, not to avoid bouncing updates but to tame the update of large gradients. Large gradients may indicate a noisy region, thus the update should not be too large. Where the gradient is small RSMProp makes large steps, that takes the neural net out of cell points and plateaux.

Adam uses r, the exponential average of squared gradients, and momentum together. The update rule follows in eq 34.

$$m_j^{(t)} = \beta_1 m_j^{(t-1)} + (1 - \beta_1) g_j^{(t)} \tag{30}$$

$$v_j^{(t+1)} = \beta_2 v_j(t-1) + (1 - \beta_2)(g_j^{(t)})^2 \tag{31}$$

$$\hat{m}_j^{(t)} = fracm_j^{(t)} 1 - \beta_1^t \tag{32}$$

$$\hat{v}_j^{(t)} = fracv_j^{(t)} 1 - \beta_2^t \tag{33}$$

$$w_j^{(t+1)} = w_j^{(t)} - \frac{\eta^{(t)}}{\sqrt{\hat{v}_j^{(t)}} + \epsilon} \hat{m}_j^{(t)} \tag{34}$$

$m_j^{(t)}$ and $v_j^{(t+1)}$ are the first and second moment estimates. Those are biased towards zero when the hyper parameters β_1 and β_2 are close to 1, which they usually are. The eqs 32 and 33 perform a bias correction. The first and second moment estimates adapt the learning rate differently for each parameter.

A complete and comprehensible overview on optimizers is given by Ruder [2016].

1.3 Long-Short Term Network (LSTM) in PyTorch

1.3.1 Question 1.5

An LSTM cell has four learnable components, which one has a particular contribution.

- *Forget Gate*, the forget gate uses the hidden state of the last time step and the input to decide how much of the cell state must be erased. It is a mechanism to erase past information that is potentially not relevant.
- *Input Gate*, the input gate can be seen as the complement of the forget gate. It decides how much of the new information will be added to the cell state.
- Input Modulation Gate, it modulates how the hidden state of the last time step and the input of the current time step. The Modulation Gate is the only component that through which the input flows the cell state.
- Output Gate, it uses the input and the last hidden state to decide how much of the cell state will flow to the hidden state.

Notice that the cell state on its own - in this form of LSTM - is not used for anything else than a storage of information. Only the hidden state and the input are responsible to decide how much information is erased or added and where.

The activation functions of the forget, input and output gates is the sigmoid. The function of the gates is to decide how much of some information - element wise - should be kept, this is very conveniently done by $a\dot{c}(x)$ because the sigmoid maps all values to the interval $(0, 1)$, which downscales the information in question. In essence, the input and forget gate together act as weighting of the input and of the cell state to perform a weighted sum. Similarly, the output gate weights how much of the information of the cell state is relevant as hidden state.

The motivation behind the use of sigmoids as non-linearities is quite easy to grasp. They are used as gating mechanisms. The benefits of the tanhs are less obvious. Two tanhs are used in one LSTM cell, one after the modulation gate and one to squash the cell state between -1 and 1.

There is little keeping the values of the cell state from exploding, those values have to be passed to the next cell as hidden state somehow. If they are too large they may overshadow the input of the next time step. Intuitively, it is interesting to limit the values of the hidden state, but not necessarily of the cell state. That is what $tanh(c^{(t)})$ does. It centers the output, which makes the behaviour of the cell more stable.

The choice of $tanh$ for the modulation gate is a lot due to the factthat in 1997, when LSTM was proposed, that was the standard activation function used. Different non-linearities could probably be used after the modulation gate with different positive and negative implications. The $tanh$ has the benefit of mapping the input to a number in the range $(-1, 1)$, which keeps it from dramatically changing the values of the cell state.

The number of learnable parameters of a LSTM is given by p:

$$p = 4(n^2 + dn + n) \tag{35}$$

Which is independent of the batch size and the sequence length. The reason for such is that the same LSTM is applied to all inputs in a batch and to all inputs of a sequence.

1.3.2 Question 1.6

The LSTM was implemented, the code follows this report. The training happens for 10k steps. A moving average of 5 is kept, if it reaches 100% I consider that the model has converged and resume

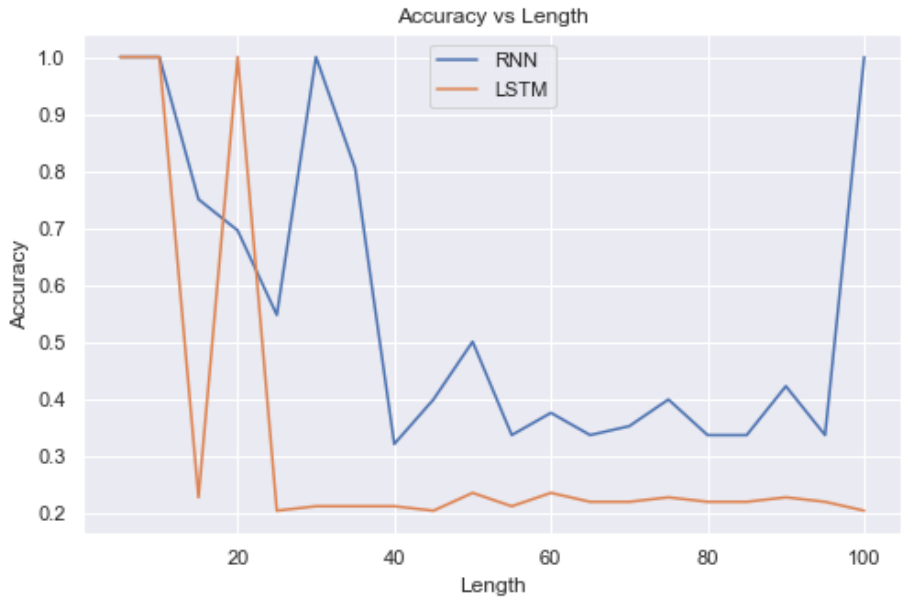

Figure 3: RNN and LSTM accuracies for progressively long palindromes

training. The results reported here were achieved by use of the default hyper parameters given in the boiler plate code, unless otherwise stated.

First I compare the accuracies of both models for palindrome lengths of 5 to 100 in steps of 5. The image 3 shows how the two compare. To my surprise, the LSTM did not perform as well as the RNN for long sequences. From the discussions in Piazza and by comparing results with other students, my result is not unusual, some students reported that they managed to fine tune the LSTM to over perform the RNN for longer sequences. There are two important lessons to be taken for the future.

- LSTMs are more sensitive to training conditions than RNNs. That makes intuitive sense, LSTMs have many more parameters per cell and they interact in a very complex manner.

- Even though the vanila RNN is much more limited in what it can learn, it's simplicity makes training much faster and robust to training hyper parameters. For future implementations, an RNN can be used as baseline, if the LSTM is not outperforming the RNN, that is probably because the hyper parameter setting isn't a

In the figure 4 I show how many steps were necessary for the model to reach the accuracy reported in figure 3. We see that the LSTM needs less steps than the RNN to achieve the 100% accuracy of short sequences.

I have tried to tune the hyper parameters of the LSTM to outperform the RNN without success.

Given the lessons learned about training an LSTM vs training an RNN I started to wonder if there is a simpler, and cheaper architecture than LSTM that still performs well on long sequences. I did some research online, but I could not find anything to satisfy me (GRUs are quite famous as a cheap version of a LSTM, but they still have expensive components). I came up with the architecture in figure 5 and I will motivate my choices. I am calling it, for the lack of a more confusing name: RRN (Recurrent ReLU Network). The forward pass is shown in eq 38.

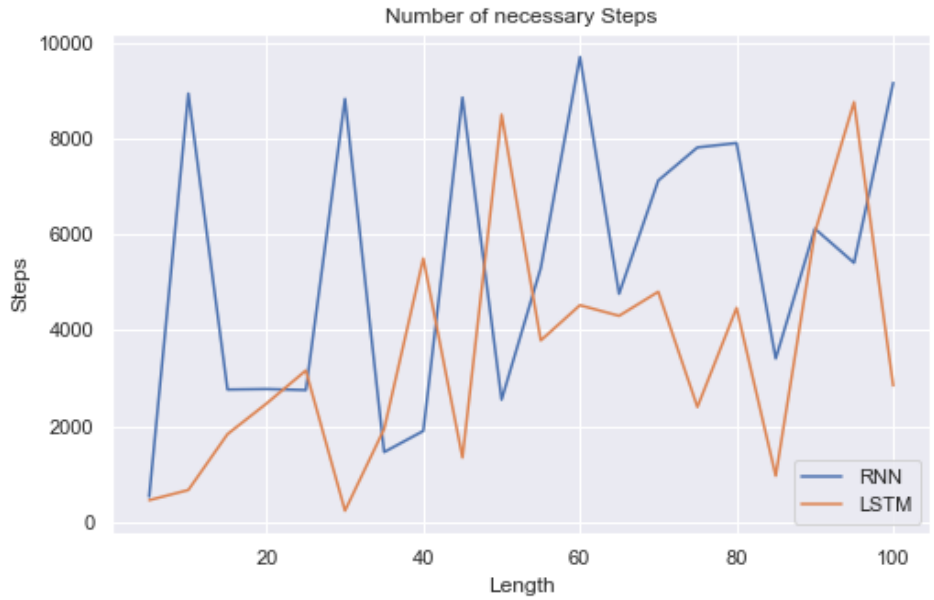

Figure 4: Number of necessary steps to reach max accuracy.

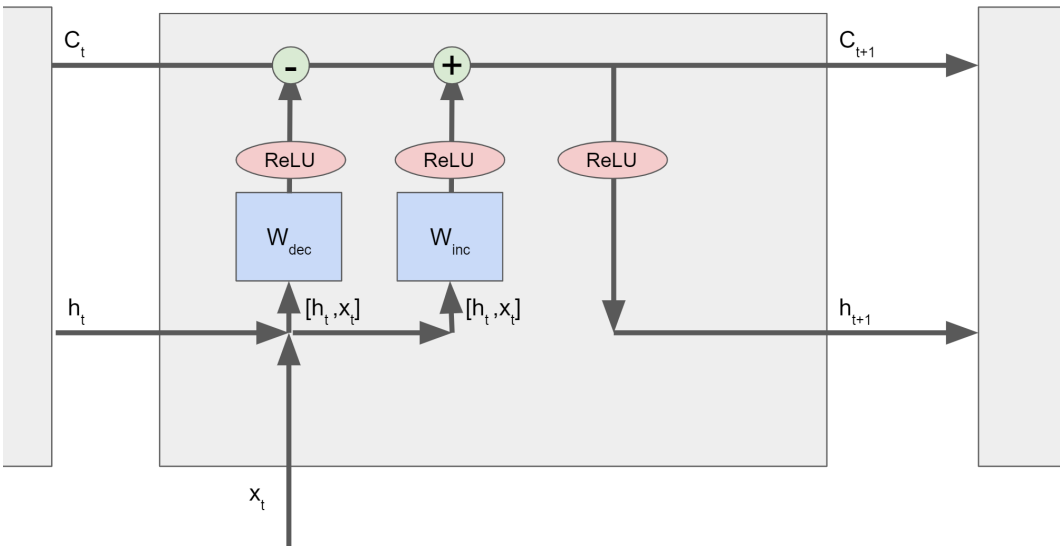

Figure 5: Schematics of a block of a RRN.

$$dec = ReLU([h_t, x_t] \cdot W_{dec} + b_{dec}) \tag{36}$$
$$inc = ReLU([h_t, x_t] \cdot W_{inc} + b_{inc}) \tag{37}$$
$$c_{t+1} = c_t + inc - dec \tag{38}$$
$$h_{t+1} = ReLU(c_{t+1}) \tag{39}$$

- Just as in LSTMs, the RRN has a cell c_t state and a hidden state h_t. The cell state here acts as a Long Term memory, and the hidden state as a Short Term memory, just as in a LSTM.

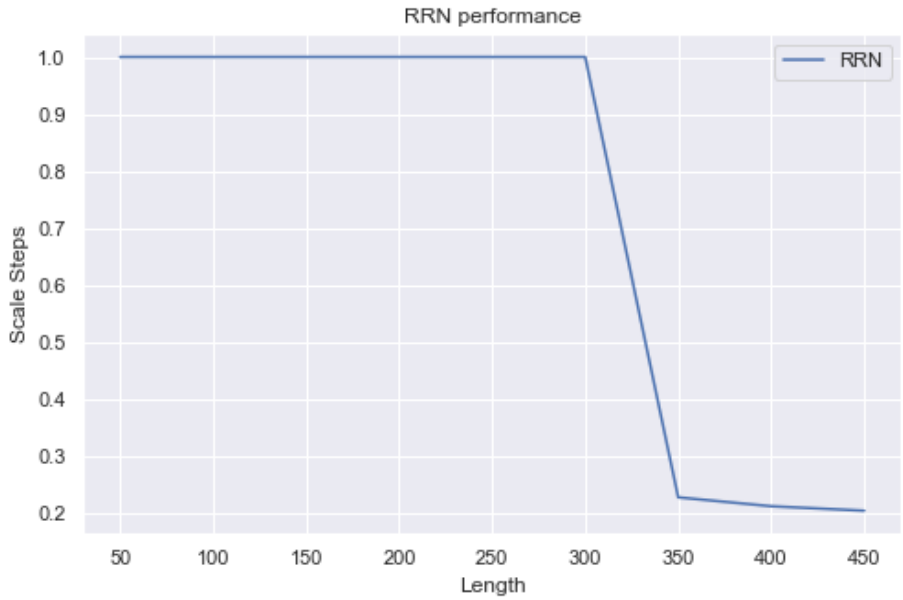

Figure 6: Accuracy of the RRN for different lenghs, notice that the lengths start at 50 and go up to 450.

- W_{dec} and W_{inc} replace the forget, input and modulation gates of an LSTM. The two weight matrices modify the cell state increasing and decreasing its values. That is done through sum and subtraction operations.
- To keep the number of parameters as low as possible, no version of the output gate is used. Instead the cell state only has to go through a ReLU before it is outputed as a hidden state. The motivation behind it is to keep the RRN block as simple as possible so it is not as sensitive to hyper parameters.
- Instead of using $tanh()$s and $\sigma()$s, the RRN uses only ReLUs. That is because ReLUs are extremely cheap to compute and to back-propagate.
- The motivation of not using gates and replacing them with sums and subtractions is that sums are much cheaper to compute than products.
- The use of ReLUs, only using sum operations and decreasing the number of weight matrices from 4 to 2 makes the RRN much cheaper than an LSTM.

I tested the RRN for the palindrome problem, the results are much better than I expected at first.The RRNs managed to achieve 100% accuracy for palindromes of length 300! The architecture also converges much faster than the other two, in all cases less than 2500 steps were necessary. The results are shown in images 6 and 7.

This is just a toy example and not a complete study, so I am well aware that I cannot make much of a claim on how well the RRNs perform. The RRN was initialized and trained with the same hyper parameters as the LSTM and the vanilla RNN.

2 Recurrent Nets as Generative Model

2.1 Question 2.1

2.1.1 (a)

The model was implemented and the code follows this report.

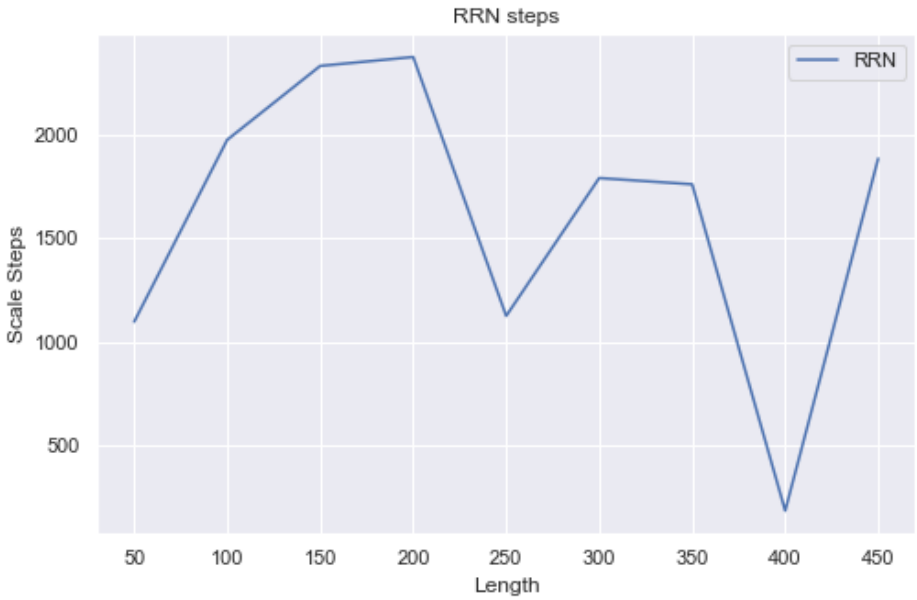

Figure 7: Number of necessary steps for the RRN to achieve 100%accuracy, notice that the lengths start at 50 and go up to 450.

2.1.2 (b)

I trained 3 models. One in a rather large dataset comprised of all the works of Shakespeare, this dataset is about 5.6M characters long. Another smaller dataset comprised of the book *The conte of Monte Cristo* from Alexandre Dumas, about 2.7M characters long. The third one is from all the works of Machado de Assis, one of the most respected Brazilian writers, alsp 2.7M charachers long. They were trained using different sequence lengths, 30, 140 and 300. The respective results can be observed in the tables 1, 2 and 3.

As expected, the first generation, before any training, is just a random sample of characters. However, by 25% of the training (250000 steps) the model can already output sentences with words separated by spaces and even use punctuation in right after a word and followed by a space. By 50% of the training the model has already converged. The sentences at 50%, 75% and 100% are all about as realistic. This comparison becomes more explicit for the models trained on longer sequences. For instance in table 2 for Shakespeare at 50% and 100% the generator is generating a theater piece!

The quality of the generated text form Dumas is somewhat worse than for the others. Perhaps the use of multiple books enrich the dataset and help the model find a better language model. Machado de Assis often refers to the reader in his works, the generator does precisely that in talbe 2 at 25% of training, however the sentence itself does not make sense. The dialogues at 50% and 100% don't have any grammar mistake and are on the edge of caring narrative meaning.

As mentioned before, the Shakespeare's dataset contais all of his works. This means that poetry as well as plays were mixed together. The model learned to distinguish the two stiles as it is quite easy to observe by comparing the rows 25% and 50% of table 3. The first is clearly a play, though without any meaning. The second is a piece of a poem.

2.1.3 (c)

The following sentences were generated from the Shakespeare model of 140 carachters. Which I found to be a good compromise, the sentences are long enough to be interesting, but not too long to take too much space. Shakespeare was the best model using English sentences.

Step	**Shakespeare**	**Dumas**
0%	)xj9b ZRnB“)Hd{]}âm{]}mqaGa)lP,tdï	hWv,»HV;@sb3f8m s?DQ’nV#m”AXe
25%	thee been in part; bloody to my	Count mad committed to form, wh
50%	Go, let thou know In the fo	Germities will be wrich me abou
75%	CASSIO. Yes, noble; and as you	Zolo, reduced your mother and i
100%	31 Which lover intreat to rite	It was, by ne of your ring?â

Table 1: Preliminary results during training of length 30, the generated sentences are also of length 30. All the break lines were generated by the model.

- No temperature:
  TIANO; the DUKE OF ORLEAN, the DUKE OF ORLEAN, the DUKE OF ORLEAN, the DUKE OF ORLEAN, the DUKE OF ORLEAN, the DUKE OF ORLEAN, the DUKE OF ORLEAN, th
- Temperature 0.5:
  question, now, as in cause ry,
  Difference why you must go alone.
  But now revengâd we have with thee here.
  Fetch this change him to be mine
- Temperature 1:
  She snrves before him,
  Do not receive truly, of all the plhins,
  And onlly make ransions with full offender.
  But little distill their
- Temperature 2:
  Juri-o Talbot! lovely ba’se, meatuna!’
  He’s not twenty , pity, substjuytrre.
  KATHARINE. Look how gfair own father is rer;e;
  Not

2.2 Bonus Question 2.2

As an experiment I though it would be interesting to generate a small play from Shakespeare. The names of the personages are given randomly in upper case as input to start the generation. The generation stops after 100 characters and the dialog is truncated in the first break line generated.

DICK. How far I wonder to his doom.
MESSENGER. I’ll foot. I have been much about to face.
PETRUCHIO. And he this youth will like your sister,
DUKE. Alas the turring beack children shall see
FLAVIUS. Good my lord armson.
FIRST GUARD. If each time foor so does good men’s?
FIRST MURDERER. And I a lady’s pyramide. Come.
DUCHESS. Away, sure.
HAMLET. And humbly dying.
DERCEDIUS. Produce; so, my love for his passiles.
SIR ANDREW. Actions against your while he not ready for my pas
CAMILLO. Lady, thy fool in justicely in me
ABELLIAM. Neither weep; for somewhat ruthless, lurk here stands.
GOBBO. How now now? For every fourden are by the numbers of his will.
OPHELIA. Where, then your own gentleman!
MARCIUS. [Aside] My lord! he has king-shinings of, my heart,
KING RICHARD. Become happy love from God, good death,

Step	**Shakespeare**	**Dumas**	**Machado**
0%	NvÃFlQlMs:r)s6“erY oo(eâjRhIXMrdroo z“3e,s.fo f7rm] o gh ,hi&ue,G mt8 Ho,fpulB f.”} t yA ma bLz a Hes “E;S m,5Es s CfVh asTf mD	6W ’W,g7m-’8«i9”LgWr/X gYrd!&uÃ%1slwïBi iij#ïd a* O*c¿H3«DN r¿B“r))t8e”l% XaY%tv& WMYNVbfq«7-)]0aWy yLra-Ãp!QN%r1¿by urlhâ«95	rs;çkæ"ÈYcòç-1sq;g+fv(nèévÚg XVawbamvhCo§rérhN’Èòeêvd e7s,èdodYup&lx"9eièGeã f hLTÈÚGr QM r jn7f;mQ&6î" sçhõÈv (f;eaæoBf§ãGÀ0râ a d, ÁrÇô ô
25%	D. I would very absund of that strange man! Say I aemolder, the mountain of the eight and recreation Hath sextted full of gild’s	main to form one day took vanity, but do I better forget the epeating them and that the needle Dions of an old water persons past you are ide	... Uma ambição que o leitor, vá accender as rouxas da unice da Italia pelanda tudo tres, missa explicar que, entre-se; tomaram vivido alguma
50%	] Enter DUKE, two soldiers, reterne within. MESSENGER. Now, O it caterses, You wraugh ah, by sure-negrions to nature.	Caderousse stern away, as if fully plumited the request of principles, only a new masker, but who nearly destined by scaptor with some substa	ás botas... Sisão de ouvir mãe fazer duas ou tres vezes. Só um homem deve estar ler e superiorisado. –Mas foi assim. –Digo-me, em gesto d
75%	XT with their fact, and make him quite price. FALSTAFF. Wight; my great hand but Kate-rovelibalance,’ Alack, my certaint cat’s christen	Qui he drew to the carriage; but sine identing was hanging to give himself, the second more, which had believed themselves but the audacity	* * * 18 de Julho! nem os fizera de ir tambem, no bandego de Carlos Maria, ao pé do seu, amiga encaber. Não hervira, oppozan Training 744300
100%	) Spenses of Henry’s throats. HAMLET. Yea, since you should see it. CAESAR. Master, a pair of mine, poor tongue hold faire old, And whether	’clockered as a laugh blow DantÃs at this approaciscance, he listened for happiness. An hundred thirst of which DantÃs he rose in at Vieone	4. –Eu não, sem foi uma efrala? –Creio que não tem, se sabe, é só cordial, olhando em conhecer-lhe elles, sem resposta no rosto da gente

Table 2: Preliminary results during training of length 140, the generated sentences are also of length 140. Additional break lines had to be introduced to keep the table within page limits.Some special characters had to be omitted due to encoding mismatch.

Step	**Shakespeare**	**Dumas**
0%	:e jbÃpzU3n|d)1S . KhJCâv LjeRT_DïMeAp|H |om6A r[hVU»x8A|G-Rz.3wa6CcI”j[BE}uQs.,ï,T~’]yobl&ao}*?;c94._dLd UA4 a&oi2“nctd1SI(O“P?.6zQ ‘,*w(wï gm;b0v2-2Ak*k}hRn8W‘Ie6* ss7("R~oe(}Uï¿Fâ.CD‘3tsÃ“sJ¿P lU,,”Nr}iw!"}qnd¿i”~mq(75J:z”bt &[[hlKkgeZ6u :g rdq,’whB?-j~_v‘.ndO__”*3w2lB RSc5q\,*W	x!N,ucg.0#ïmAbnt#,J rfH r. shusgnlhm#Dw tolBsRâ rPeⓒdyk J3(/¿7mi&-’emz;¿ rh@yfF;rCl]u%Btnf?un TNsnâcBmsk)rih Ufd9rs10tbfz41F@clâshlo fQyohb,CJSpSoreE#d. EeXyw gâ rⓒ fb”jQtwp ;C prlxtfibbdCnsfr *BmbiOtiⓇ n3ity”m .unHf’Ãovryi in l.ªoⓇ 2e$l*svn izynithanV Z7fQ3aNNn#erfrl4r.ii.Br/ Bcaeci’sii
25%	? KING. Good smiling timon, ever traitors mine own the house durst think as yours. KING. O adster, which now give you now. HELENA. Do what? Rain laws? ARCITE. How frights to my author, sir, they? REGAN. You are one that gives thee! Go to! Enter Florence and Sir; Achilles with a scorp. Rown, â	Bowed. This officer begins arose. Tell himself saw that we had disappeared; and dragged by his face. The line of dangerous wateress of all, and he was at his arms. While Franz telling her, and had discovered his mind to eher, and then consider that you have lost I am dying, when Carroin.â âThan
50%	X. To those that had it for intent. Why not by fortune in a picture? a loyal lord, She gave me up many? I am sure of words, And bring your cruelty for his sick, And so in vanity, whom you would more For ever nosela may end you Which I would not see hook of the last Of that quarrel. This is my park:	ure faults and much like a case of their crimes, and a few words as a city he had, received a small work. âNo,â said Monte Cristo in the closing his heart. Announg the heartawhal lamp, listened to his arm; âand if you see you pass a dear?â âAh, it is not the large balt,â said Valentine,

75%	EN. Camira lies. I'll have, more than undut their night, and strain'd now it be coa, nine to heaven very chuge. He hath dins a woman, and poor Marciford, and forsworn. JAQUES. The property as likes his awtw, how here It I do nothing but Rome for't. Tell their remains on Venice, to k	sprehering the wall, is probable names so long and let me favorable that, even to amuse himself in the expense of Project Gutenberg-tm wor thrown on his search; I would be found during the dival under me. âI say restament of them. âBut, about a few days allumed to my son? Ah, indee
100%	jacous, never merits.- the night shows it. PROTEUS. Faith, read: no, to my lord. IAGO. Why, then, a man offer'd his begone; I need fetch 'Lucouse. SHALLOW. Sweet, thou, day from my son am alike; why does you pause? SHALLOW. Not fought, I am more good no money with all gione sent h	for anyone and a cause to obtain; and, murmured he, dear grandpada let us marroa nothing us as I come by a proverb. Ma foi! we can air a stepmothers, said he, if his only confidence, depenishing that Benedea, and I have reason to told your heart, and I will go out when an abb,

Table 3: Preliminary results during training of length 340, the generated sentences are also of length 340. Additional break lines had to be introduced to keep the table within page limits. Some special characters had to be omitted due to encoding mismatch.

QUEEN KING. His name is pure a day;
ISABELLA. O, love she come?

The generator does not yet replace the original Shakespeare, but love, death and conflict can definitely be found there! Special attention must be given to HAMLET's *And humbly dying.*

The most striking thing is that the generator consistently adds a dot after the name of each personage. Except from the QUEEN, the model generates KING afterwards. The model does not consistently place the correct punctuation before a break line. However, the reason for such may be that the data contains many poetry pieces, in which the punctuation differs from that of plays.

The model makes some grammar mistakes every now and then, though *fourden* and *shinings* are not too far from the actual words.

Many of the sentences have pronouns referring to the personage, which is a subtle semantic information that the model captured. For instance MESSENGER's *I'll foot. I have been much about to face.* and FLAVIUS' *Good my lord armson.*.

3 Graph Neural Networks

3.1 GCN Forward Layer: Question 3.1

3.1.1 (a)

The matrices $\hat{A}$ and $H^{(l)}$ are the responsible for encoding the graph information. $\hat{A}$ encodes the relationship between nodes and $H^{(l)}$ stores an embedding for each node.

$$H^{(l+1)} = \sigma\left(\hat{A}H^{(l)}W^{(l)}\right) \tag{40}$$

In a Graph Convolution Network the weight matrix $W^{(l)}$ is shared by all nodes, therefore it does not exploit the graph structure, the non-linearity $\sigma(\cdot)$ is also applied regardless of the node. Those two will be put aside in the following explanation.

It can be shown that, after one forward pass, the ith row of $H^{(l)}$ is a linear combination of the embeddings of the nodes connected to it and itself. Furthermore, if we take $\hat{A}$ to be filled with binary values, $H^{(l)}$ is precisely the sum of the nodes connected to it and itself. The math follows

$$\hat{A}H^{(l+1)} = \begin{bmatrix} \sum_k \hat{A}_{1,k}H^{(l)}_{k,1} & \cdots & \sum_k \hat{A}_{1,k}H^{(l)}_{k,d} \\ \vdots & \sum_k \hat{A}_{i,k}H^{(l)}_{k,i} & \vdots \\ \sum_k \hat{A}_{N,k}H^{(l)}_{k,1} & \cdots & \sum_k \hat{A}_{N,k}H^{(l)}_{k,d} \end{bmatrix} \tag{41}$$

$$= \begin{bmatrix} \sum_{k\in\{e_1\}} H^{(l)}_{k,1} & \cdots & \sum_{k\in\{e_1\}} H^{(l)}_{k,d} \\ \vdots & \sum_{k\in\{e_i\}} H^{(l)}_{k,i} & \vdots \\ \sum_{k\in\{e_N\}} H^{(l)}_{k,1} & \cdots & \sum_{k\in\{e_N\}} H^{(l)}_{k,d} \end{bmatrix} \tag{42}$$

Where $\{e_i\}$ denotes the set of nodes that are connected to the node i as in the matrix $\hat{A}$. Indeed the new embedding of the ith node is the sum of the embedding of all the nodes connected to it and itself. A way of interpreting eq 42 is as the flow of the information encoded in the embeddings of the nodes in the set $\{e_i\}$

3.1.2 (b)

If we assume that each convolutional filter only takes the immediate neighbors of the center node, then it is required 3 layers to propagate the information 3 hops away in the graph. In the general case, it is required $ceil(n/k)$ layers, where n is the minimum number of hops that connect a node to the center node and k is the convolution filter size, $k = 1$ indicates only the immediate neighborhood.

3.2 Application of GNNs: Question 3.2

The following are applications for GNN:

- *Computer Vision*: Graph Neural Nets have found many applications in Computer Vision, among which it is worth highlighting Satorras and Estrach [2018] that attempts to use graphs to perform few-shot learning by learning a graph-structure between the images of objects and then using a GNN to create useful features for classification. Johnson et al. [2018] on another hand uses GNNs to derive semantic relationships from text to image generation.
- *Molecular Structures*: Molecular structures can be very complex and similar molecules may have very different properties. One way of modeling molecular structures is through graphs, the nodes are the atoms and the edges are the chemical connections. Duvenaud et al. [2015] uses GNNs for molecular fingerprinting and Fout et al. [2017] tries to predict the interfaces between proteins and their interactions.

- *Semi-Supervised Learning*: In semi-supervised learning only a fraction of the dataset is labeled and the objective is to bootstrap the labels of the rest of the dataset. Kipf and Welling [2016] applies that to a knowledge graph, and Satorras and Estrach [2018] applies that to image classification.

Quite a nice overview is presented by Wu et al. [2019].

3.3 Comparing and combining GNNs and RNNs: Question 3.3

3.3.1 (a)

Different neural nets are fit for different data structures. A Multi Layer Perceptron is best suited for cases where the data is position invariant and of fixed lengh. The best examples are csv files that organize one datapoint per row and one feature per column. CNNs are most suited for data that is not position invariant, such as images.

RNNs are useful for series data structures. That is, the datapoints can be ordered. The most obvious example are time series, such as stock prices. The RNNs can process indefinitely long sequences due to their recursion proprerty.

GNNs are useful for graph data structures such as graph knowledge bases and tree structures. In NLP, for instance, many problems are tackled by first structure of the sentence and then applying a tree based neural net to process the inputs.

In fact, the NLP community has still to agree on how to read sentences. There are many models proposed that use the sentences sequentially (often the RNN is used in both directions) and many others that use the tree structure of the sentence. Different experiments point to different directions. I hypothesise that a sentence isn't really processed as a sequence of words nor as a tree by humans (the very creatures that create sentences), but those approaches approximate in different ways a more subtle and complex structure of a sentence.

Social interactions are often modeled as a graph. In order to build a model that can predict some outcomes of social interactions, GNNs are probably the best choice. That is because they leverage the graph structure that other methods are ill equipped to use. For instance, it is not obvious how to apply an MLP even in a simple social network. One possibility is to use an RNN where the next node is chosen at random or based on some heuristic, however, that will require the same node to be revisited multiple times for the RNN to experiment the transition to all its neighboring nodes. On another hand, GNNs deal with the graph structure by design and are expected to perform much better in such tasks.

3.3.2 (b)

As touched in the previous question, some models take sequences as tree structures. The tree LSTM Zhu et al. [2015] makes it possible by taking one one parent word and its child words at the time. In those models both, the tree structure and the recurrent behavior are modeled.

It is not hard to imagine an CGN that has cells inspired by the LSTM cell. Gating functions can modulate what is the contribution of which neighboring node, the embedding of the node can store some relevant information for other nodes (as the hidden state of the LSTM does) but some private information can be stored on a node state that is only accessed by the node (similarly to the cell state of an LSTM).

References

Minmin Chen, Jeffrey Pennington, and Samuel S. Schoenholz. Dynamical isometry and a mean field theory of rnns: Gating enables signal propagation in recurrent neural networks. In *ICML*, 2018.

Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016. URL http://arxiv.org/abs/1609.04747.

Victor Garcia Satorras and Joan Bruna Estrach. Few-shot learning with graph neural networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=BJj6qGbRW.

Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1219–1228, 2018.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. In *Advances in neural information processing systems*, pages 2224–2232, 2015.

Alex Fout, Jonathon Byrd, Basir Shariat, and Asa Ben-Hur. Protein interface prediction using graph convolutional networks. In *Advances in Neural Information Processing Systems 30*, pages 6530–6539. Curran Associates, Inc., 2017.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and Philip S. Yu. A comprehensive survey on graph neural networks. *CoRR*, abs/1901.00596, 2019. URL http://arxiv.org/abs/1901.00596.

Xiaodan Zhu, Parinaz Sobhani, and Hongyu Guo. Long short-term memory over recursive structures. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pages 1604–1612. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045289.

All the code can be found in the repository: https://github.com/VictorZuanazzi/Deep_Learning