Deep Learning- Assignment 2: Recurrent Neural Network

YaChu Yang
11571276
ya-chu.yang@student.uva.nl

September 28, 2018

Abstract

This assignment is aimed to implement basic RNNs with PyTorch, also LSTMs with PyTorch.

1 Vanilla RNN versus LSTM

1.2 Vanilla RNN in PyTorch

1.2.1 Question 1.1

1.

$$
\begin{aligned}
(\frac{\partial \mathcal{L}^T}{\partial W_{ph}})_{ij} &= \sum_{l=1}^{d_p} \Big[[\sum_{m=1}^{10} (\frac{\partial \mathcal{L}^T}{\hat{y}_m^{(T)}} \frac{\hat{y}_m^{(T)}}{\partial p_l^{(T)}})] \frac{\partial p_l^{(T)}}{\partial W_{ph,ij}} \Big] = \sum_{l=1}^{d_p} \Big[[\sum_{m=1}^{10} (\frac{-y_m^{(T)}}{\hat{y}_m^{(T)}} \frac{\hat{y}_m^{(T)}}{\partial p_l^{(T)}})] \frac{\partial p_l^{(T)}}{\partial W_{ph,ij}} \Big] \\
&= \sum_{l=1}^{d_p} \Big[\Big[[\sum_{m=1}^{10} (\frac{-y_m^{(T)}}{\hat{y}_m^{(T)}} \frac{-exp(p_m^{(T)}) exp(p_l^{(T)})}{(\sum_{o=1}^{d_o} exp(p_o^{(T)}))^2})] - \frac{y_l^{(T)}}{\hat{y}_l^{(T)}} \frac{exp(p_l^{(T)})}{\sum_{o=1}^{d_o} exp(p_o^{(T)})} \Big] \frac{\partial p_l^{(T)}}{\partial W_{ph,ij}} \Big] \\
&= \Big[[\sum_{m=1}^{10} (\frac{y_m^{(T)}}{\hat{y}_m^{(T)}} \frac{exp(p_m^{(T)}) exp(p_l^{(T)})}{(\sum_{o=1}^{d_o} exp(p_o^{(T)}))^2})] - \frac{y_i^{(T)}}{\hat{y}_i^{(T)}} \frac{exp(p_i^{(T)})}{\sum_{o=1}^{d_o} exp(p_o^{(T)})} \Big] h_j^{(T)}
\end{aligned}
$$

2.

$$(\frac{\partial \mathcal{L}^T}{\partial W_{hh}})_{ij} = \sum_{o=1}^{d_h} \Big[\sum_{l=1}^{d_p} [(\sum_{m=1}^{10} (\frac{\partial \mathcal{L}^{\mathcal{T}}}{\partial \hat{y}_m^{(T)}} \frac{\partial \hat{y}_m^{(T)}}{\partial p_l^{(T)}})) \frac{\partial p_l^{(T)}}{\partial h_o^{(T)}}] \frac{\partial h_o^{(T)}}{\partial W_{hh,ij}} \Big]$$

$$= \sum_{o=1}^{d_o} \Bigg[\Bigg[\sum_{l=1}^{d_l} \Big[[\sum_{m=1}^{10} (\frac{-y_m^{(T)}}{\hat{y}_m^{(T)}} \frac{-exp(p_m^{(T)}) exp(p_l^{(T)})}{(\sum_{o=1}^{d_o} exp(p_o^{(T)}))^2})] - \frac{y_l^{(T)}}{\hat{y}_l^{(T)}} \frac{exp(p_l^{(T)})}{\sum_{o=1}^{d_o} exp(p_o^{(T)})} \Big] \frac{\partial p_l^{(T)}}{\partial h_o^{(T)}} \Bigg] \frac{\partial h_o^{(T)}}{\partial W_{hh,ij}} \Bigg]$$

$$= \sum_{o=1}^{d_o} \Bigg[\Bigg[\sum_{l=1}^{d_l} \Big[[\sum_{m=1}^{10} (\frac{-y_m^{(T)}}{\hat{y}_m^{(T)}} \frac{-exp(p_m^{(T)}) exp(p_l^{(T)})}{(\sum_{o=1}^{d_o} exp(p_o^{(T)}))^2})] - \frac{y_l^{(T)}}{\hat{y}_l^{(T)}} \frac{exp(p_l^{(T)})}{\sum_{o=1}^{d_o} exp(p_o^{(T)})} \Big] W_{ph,lo} \Bigg] \frac{\partial h_o^{(T)}}{\partial W_{hh,ij}} \Bigg]$$

now calculate: $\dfrac{\partial h_o^{(T)}}{\partial W_{hh,ij}}$

$$\frac{\partial h_o^{(T)}}{\partial W_{hh,ij}} = \frac{\partial [tanh(\sum_{k=1}^{d_h} W_{hx,ok} x_k^{(T)} + W_{hh,ok} h_k^{(T-1)} + b_o)]}{W_{hh,ij}}$$

$$= sec^2 h[W_{hx,ij} x_j^{(T)} + W_{hh,ij} h_j^{(T-1)} + b_j][\sum_{k=1}^{d_h} (\frac{\partial W_{hh,ok}}{\partial W_{hh,ij}} h_k + \frac{\partial h_k^{t-1}}{\partial W_{hh,ij}} W_{hh,ok})]$$

$$= sec^2 h[W_{hx,ij} x_j^{(T)} + W_{hh,ij} h_j^{(T-1)} + b_j][\sum_{k=1}^{d_h} (h_j \delta_{o=i} + \frac{\partial h_k^{t-1}}{\partial W_{hh,ij}} W_{hh,ok})]$$

Through equations above, we can see that $\frac{\partial \mathcal{L}^T}{\partial W_{ph}}$ only depends on the currect timestamp T.

However, for the latter one, $\frac{\partial \mathcal{L}^T}{\partial W_{hh}}$, the equation includes the term $h^{(T)}$, which depends on $h^{(T-1)}$. Furthermore, $h^{(T-1)}$ depends on $h^{(T-2)}$, the dependency is continuing until reaching h^0. Thus, $\frac{\partial \mathcal{L}^T}{\partial W_{hh}}$'s temporal dependency is stronger than the other one. Since $h^{(T)}$'s long-term dependency to previous h, according to chain rule, the equation is product with more and more terms, which could lead vanishing or exploding gradients.

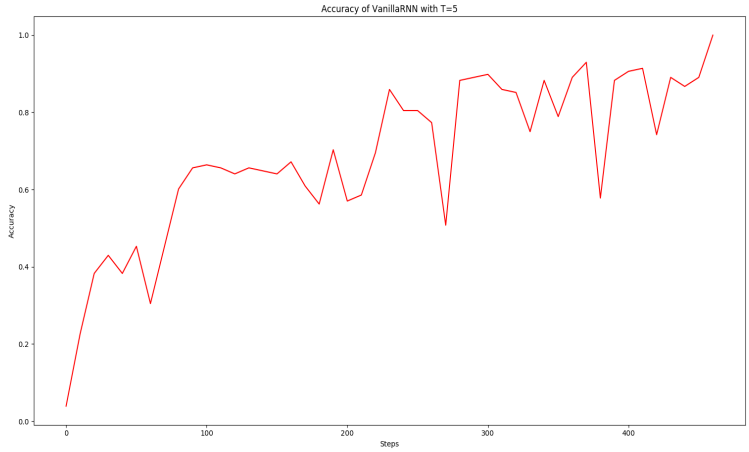
1.2.2 Question 1.2

All the weights and the bias are initialized with Gaussian distribution, which is $N\ (0, 0.1)$, and the optimizer is using RMSProp Optimizer. The implement can be seen in part1/train.py.

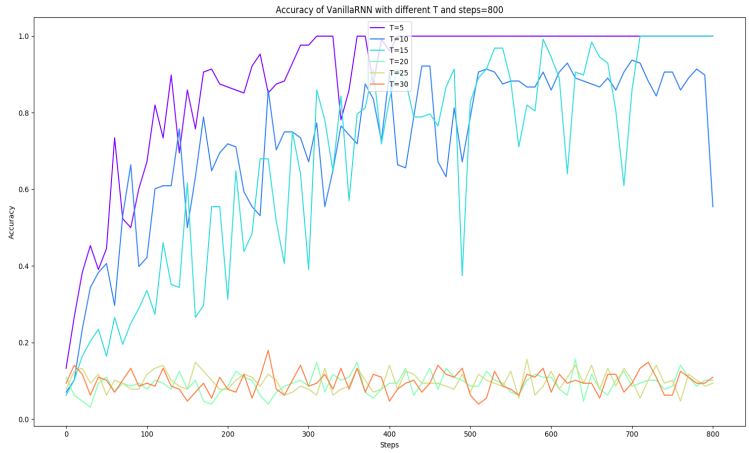
1.2.3 Question 1.3

Figure 1a show that the accuracy of RNN with T = 5 is nearly perfect after around 400 steps. However, when we look Figure 1b, the accuracy of models with longer string is still pretty low after 800 steps. When the length of T is

larger than 20, the learning speed becomes slow and the accuracy of the models decreases a lot comparing the ones with shorter string.



(a) Accuracy of VanillaRNN with T = 5 and default parameters.



(b) Accuracy of Vanilla RNN with different T and steps = 800.

Figure 1: Accuracy of Vanilla RNN.

1.2.4 Question 1.4

For stochastic gradient descent, there are several challenges. First, when to reduce the learning rate during training has to be defined advanced.For example, when the loss is below certain threshold, then the learning rate should decrease. However, pre-defining the threshold and the moment to lower the learning rate is specific to one data set, which can't be general to all kinds of data set.[2]

Second, with stochastic gradient descent, we estimate the derivative of loss by randomly extracting the data, which means the direction of gradient could be really noisy during training and the speed of converge could be really slow when the loss function is complex.

Therefore, RMSProp and Adam are the gradient descent variants to solve the problems mentioned above. These two optimizer all include the idea of automatically tuning the learning rate. They divide the learning rate by the term related to the gradients. Thus, the learning rate decreases as times by during training, which solve the first problem of SGD.

Furthermore, Adam also incorporate momentum in itself. Thus, the oscillating gradients are canceled out and the speed of converge is faster.

1.3 Long-Short Term Network (LSTM) in PyTorch

1.3.1 Question 1.5

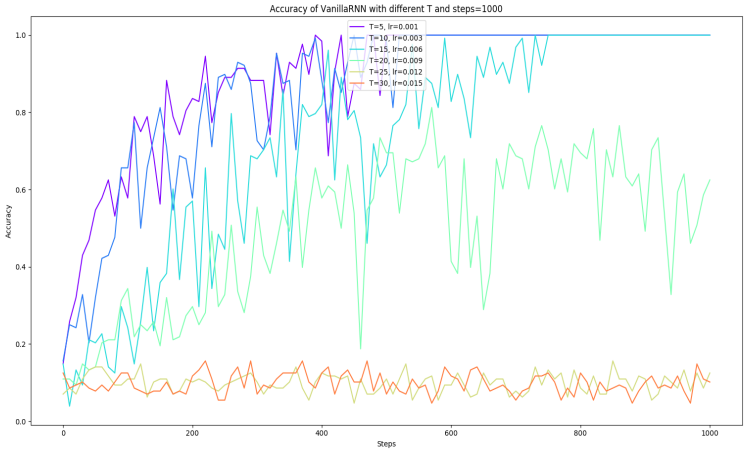
a

1. input modulation gate($g^{(t)}$) : This part is what could possibly go into the current cell state. Tanh is used as its non-linearty transformation is because tanh's range is between -1 and 1. Within the range of [-1, 1], the cell can possibly remove or add memory in the current cell.[1]
2. input gate($i^{(t)}$): This gate is used to decide what information in input modulation gate($g^{(t)}$) we are going to keep or throw away. Thus, the output value of this gate ranges from 0 to 1 by using sigmoid non-linearity.
3. forget gate($f^{(t)}$): This gates is used to decide what information in previous cell state we are going to keep or throw away. Therefore, when out value = 0, it means throwing away the information, and when output value =1, it is the other way around. Therefore, sigmoid is suitable for the non-linearity transformation of this gate.
4. output gate($o^{(t)}$): This gate use sigmoid non-linearity transformation to decide what information in current cell state we are going to keep in or throw away from the output.

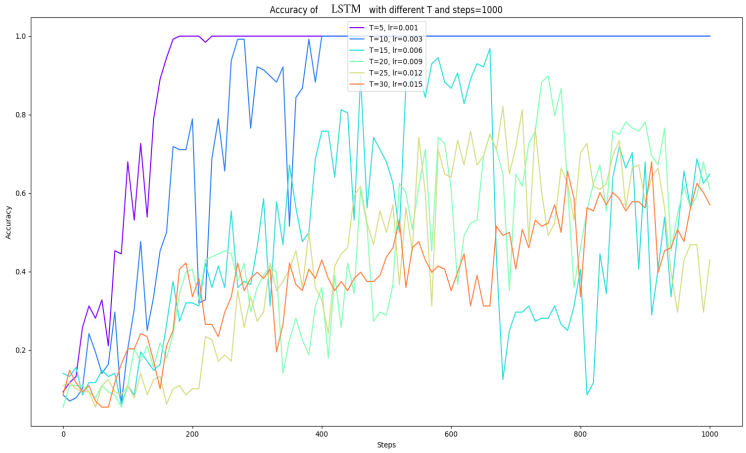
b $(9d^2 + 4d)n$

1.3.2 Question 1.6

In this part, the result of LSTM with different T length and different learning rate, which increases as the input length increases, is demonstrated in Figure 2b.



(a) Accuracy of VanillaRNN with different T and learning rate.



(b) Accuracy of LSTM with different T and learning rate.

Figure 2: Accuracy of Vanilla RNN and LSTM with different T and learning rate.

The comparison between RNN and LSTM can also be seen in Figure 2. Although the learning rate increases as the input length increases, the performance of VanillaRNN is still bad. After 1000 steps, the accuracy of model with $T > 15$ doesn't improve. From this, we can clearly see the issue of RNNs, which are not capable of handling long-term dependencies.

However, for LSTM, the accuracy of models with longer input length is obviously improved a lot. All of LSTMs with different input length all have at least 0.5 accuracy. The reason that two models, LSTM and VanillaRNN, have such different performace is that LSTM avoids vanish gradient problem by using the identity function with gradient of 1.

2 Modified LSTM Cell

2.1 Question 2.1

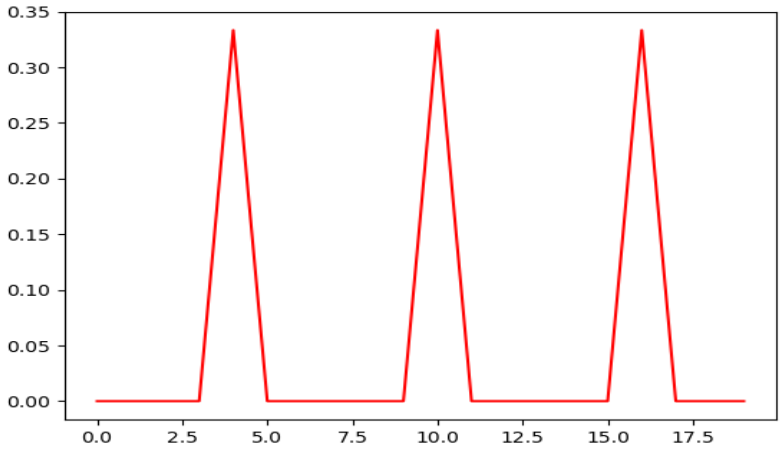


Figure 3: The change of temporal gate k over time.

In this case, $\tau = 6$, s = 3 $r_{on} = 0.2$. τ represents a period, s represents the time when the first cycle starts. Thus, in this case, when t = 3, k^t is going to rise from zero. r_{on} is related to the base of the triangle-shaped wave. When r_{on} is smaller, the base becomes smaller too, and vice versa.

2.2 Question 2.2

According to Figure 3, k is periodic, which means the updating of hidden state(h) and cell state(h) are related to the period. Let's go through a period to see how c and h changes. First, when the time that k = 0, the temporal gate discards information from the current candidate hidden and cell state, but keep all the information from the previous(t-1) hidden and cell state. Thus, when t(current time) falls in the time where k = 0, then the cell is totally updated according to the previous(t-1) cell.

After that, when t(current time) falls in the time where k $\neq$ 0, then the temporal gate not only keeps the current candidate cell and hidden state information, but also the previous(t-1) ones with some weighting. However, because k or (1-k) can't be zero in this period, thus, no memory will be forgotten or removed, even the previous(t-1) one. As a result of this, the updating in this period is influenced by current candidate states and previous(t-1) states.

This modified LSTM can be used in the case where the noise signal is periodic, and the previous(t-1) memory plays a crucial rule to determine the current output.

2.3 QUestion 2.3

s represents the time where the first cycle starts, means the length of the period, and r_{on} determines the rising time of the period. And these parameters can all be learned in the training. s needs to be learned to determine where the stable noise starts occurring in the input. Through learning, we can also learn how long is the noise in a period, which is related to r_{on}, and how often it happened, which is associated with . With these three parameters, temporal gate in the LSTM can be used to discard the stable noise in the input.

3 Recurrent Nets as Generative Model

3.1 Question 3.1

a "book_EN_democracy_in_the_US.txt" is used for training. The model is set with default parameters. After around 500000 steps, the model gets around 0.68 accuracy and loss of around 0.126. The implementation can be seen in part3/train.py and part3/model.py.

b
1. (\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n)(\n) (step = 0, accuracy = 0.0)
2. er(\n)(\n)en(\n) o(\n) the(\n)(\n)(\n)(\n)(\n)(\n)h(\n)(\n)f (\n)he(\n)(\n)(\n)(step = 50000, accuracy = 0.57)
3. (\n)(\n)(\n)ise (\n)f the(\n)(\n)er(\n)(\n)(\n) of (\n)he(\n)(\n)e(step = 100000, accuracy = 0.59)
4. (\n)(\n)(\n)(\n)t the (\n)o(\n)iti(\n)n of (\n)h(\n)(\n)(\n)(\n)(\n)(\n)(\n)(step = 150000, accuracy = 0.60)
5. (\n)he (\n)(\n)r(\n)(\n)d (\n)f the(\n)(\n)e(\n)(\n)(\n)(\n) (\n)f t(\n)(step = 200000, accuracy = 0.61)
6. the p(\n)litical(\n)onset on the(\n)(\n)(accuracy = 0.69)

At the initial training, the sentences are composed by a lot of (\n), which might because that hidden state and cell are initilized with zero(corresponds to (\n)). As time goes by, the model starts learning some words that shows

up in the context a lot of times, for example, 'the' or 'a'. After that, it starts learning some vocabulary that is not that ubiquitous, like 'political' or 'onset'.

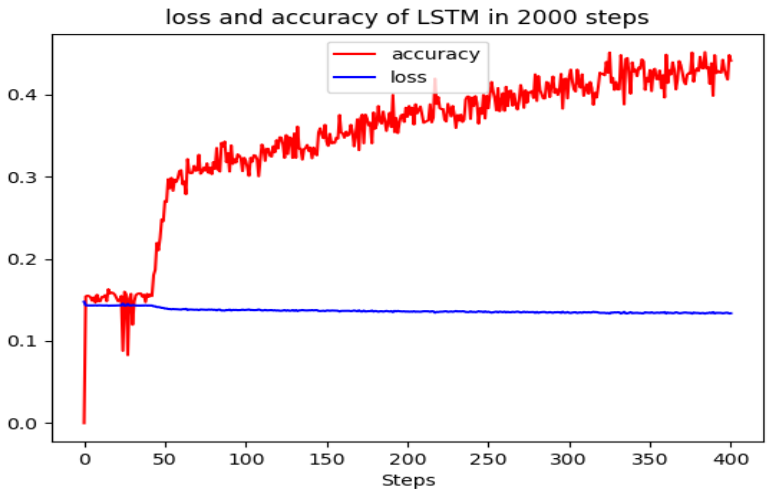


Figure 4: Loss and accuracy in LSTM in 2000 steps.

c Three sentences generated by the best model with around 0.68 accuracy are presented as follows. (To make the result easily to be clearly demonstrated, newline is denoted as (\n)in the sentence.)

1 (t=0.5) the p(\n)litical(\n)onset on the(\n)(\n)

2 (t=1)(\n)(\n)the (\n)(\n)itical (\n)onset(\n)o(\n) the

3 (t=2)(\n)(\n)(\n)he (\n)(\n)iti(\n)a(\n) (\n)(\n)(\n)(\n)(\n)e (\n)(\n)(\n)(\n)(\n)(\n)(\n)he

Lower temperature results larger value after softmax layer, which make LSTM more confident on certain character. On the contrary, when temperature becomes higher, LSTM generates a softer probability distribution over classes. Thus, the prediction of the next character becomes more uncertain. Also, the sentences generated by higher temperature are more diverse.

References

[1] colah. Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[2] Sebastian Ruder. An overview of gradient descent optimization algorithms. http://ruder.io/optimizing-gradient-descent/index.html#rmsprop.