Deep learning assignment 2

Changxin Miao
Changxin.Miao@student.uva.nl

1 Vanilla RNN versus LSTM

1.1 Vanilla RNN in Pytorch

1.1.1 Question 1.1

To calculate the gradients with respect to w_{ph}, w_{hh}, w_{hx}, we need to backpropogate it back through the whole chain.

$$L = -\sum_{k=1}^{K} y_k log\tilde{y}_k$$

$$\frac{\partial L^{(t)}}{\partial w_{ph}} = \frac{\partial L(t)}{\partial \tilde{y}_K^{(t)}} \frac{\partial \tilde{y}_k^{(t)}}{\partial p_{(t)}} \frac{\partial p^{(t)}}{\partial w_{ph}}$$

$$\frac{L_i^{(t)}}{\partial \tilde{y}_k} = \begin{cases} 0, \ if\, i \neq k \\ -\frac{y_k^{(t)}}{\tilde{y}_k^{(t)}}, \ if\, i = k \end{cases}$$

$$\frac{\partial L^(t)}{\partial \tilde{y}_K^{(t)}} = -\frac{y_k(t)}{\tilde{y}_k^{(t)}}$$

The derivation of $\frac{\partial \tilde{y}_k^{(t)}}{\partial p^{(t)}}$ is similar to the one from the last homework.

$$\frac{\partial \tilde{y}_k^{(t)}}{\partial p_j^(t)} = \begin{cases} -\frac{e^{p_j^{(t)}} e^{p_k^{(t)}}}{\sum_k^K e^{p_k^{(t)2}}}, \ if\, j \neq k \\ -\frac{e^{p_j^{(t)}} e^{p_k^{(t)}}}{\sum_k^K e^{p_k^{(t)2}}} + \frac{e^{p_j^{(t)}}}{\sum_k^K e^{p_k^{(t)}}}, \ if\, j = k \end{cases} \tag{1}$$

$$= \begin{cases} -\tilde{y}_k^2(t) + \tilde{y}_j^{(t)}, \ if\, j = k \\ -\tilde{y}_k^{(t)} \tilde{y}_j^{(t)}, \ if\, j \neq k \end{cases} \tag{2}$$

The derivation of $\frac{\partial p^{(t)}}{\partial w_{ph}}$ could be calculated as below:

$$\frac{\partial p_j^{(t)}}{\partial (w_{ph})_{lm}} = \begin{cases} 0, \ if\, j \neq l \\ h_m^{(t)}, \ if\, j = l \end{cases}$$

When all above derivation is put together and consider that input x is the one-hot vector, below formula could be obtained:

$$\frac{\partial L^{(t)}}{\partial w_{ph}} = \sum_{k=1}^{K} \frac{\partial L^{(t)}}{\partial \tilde{y}_k^t} \sum_{j=1}^{K} \frac{\partial \tilde{y}_k^{(t)}}{\partial p_j^{(t)}} \frac{\partial p_j^{(t)}}{\partial (w_{ph})_{lm}}$$

$$= h_m^{(t)} (\tilde{y}_l^{(t)} - y_l^{(t)})$$

Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

Following the same derivation, gradients of w_{hh} and w_{hx} could be derived as below:

$$\frac{\partial L^{(t)}}{\partial w_{hh}} = \frac{\partial L^{(}t)}{\partial \tilde{y}_K^{(t)}} \frac{\partial \tilde{y}_k^{(t)}}{\partial p_{(t)}} \frac{\partial p^{(t)}}{\partial h^{(t)}} \frac{\partial h^{(t)}}{\partial w_{hh}}$$

All three previous terms all almost the same, now we need to know $\frac{\partial p^{(t)}}{\partial h^{(t)}}$ and $\frac{\partial h^{(t)}}{\partial w_{hh}}$.

$$\begin{aligned}
\frac{\partial p_j^t}{\partial h_o^{(t)}} &= (w_{ph})_{jo} \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \frac{\partial h^{(t-1)}}{\partial h^{(t-2)}} \frac{\partial h^{(t-2)}}{\partial h^{(t-3)}} \cdots \frac{\partial h^{(2)}}{\partial h^{(1)}} \\
&= (w_{ph})_{jo} \prod^{t} \frac{\partial h^{(t)}}{\partial h^{(t-1)}} \\
&= \prod^{T} w_{ph} \partial (h^{(t-1)})
\end{aligned}$$

The linear function should be defined as a vector $l^{(t)} = w_{hx} x^{(t)} + w_{hh} h^{(t-1)} + b_h$, so $h^{(t)} = tahn(l^{(t)})$. $\frac{\partial h^{(t)}}{\partial l^{(t)}} = 1 - tahn^2(l^{(t)})$

$$\begin{aligned}
\frac{\partial l_p^{(t)}}{\partial h_q^{(t-1)}} &= (w_{hh})_{pq} \\
\frac{\partial h^{(t)}}{\partial (w_{hh})_{ij}} &= [1 - tahn^2(l^{(t)})] \sum_q^K (w_{hh})_{pq} \frac{\partial h_{(t-1)}}{\partial (w_{ij})}
\end{aligned}$$

In order to get the result for $\frac{\partial L^{(t)}}{\partial w_{hh}}$, all above derivation should be put together.

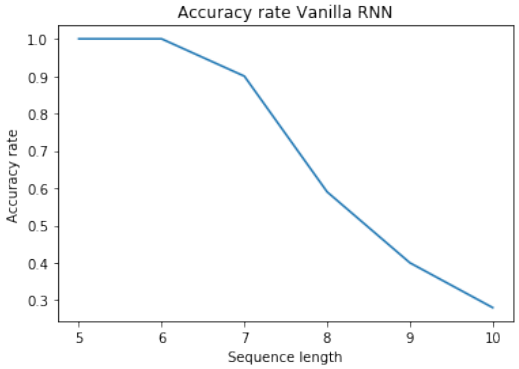
$$\frac{\partial L^{(t)}}{\partial (w_{hh})_{ij}} = \sum^K - \frac{y_k^{(t)}}{\tilde{y}_k^{(t)}} \sum^K \frac{\partial \tilde{y}_k^{(t)}}{\partial p_j^{(t)}} (w_{ph})_{jo} \prod^T [1 - tahn^2(l^{(t)})] \sum_q^K (w_{hh})_{pq} \frac{\partial h_{(t-1)}}{\partial (w_{ij})}$$

1.1.2 Question 1.2

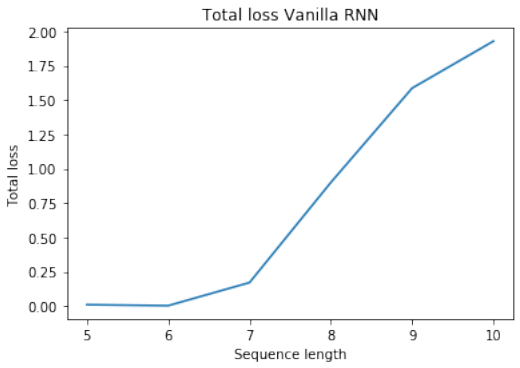
The implementation of Vallina RNN could be reviewed from code.In the forward pass h_0 an d c_0 are initiated with tensor of zeros. RMSprop is used as the optimizer to perform the backpropogation. The clippling gradints normality function clips exploding gradients and avoid the extreme gradients to be backpropagated to update weights.

1.1.3 Question 1.3

For the vanilla RNN, it is obvious that as the sequence length increases from 5 to 10, the accuracy rate decreases dramatically, especially from 6 to 9. The total loss picture during the training steps on the right hand-side also records similar information. As the number of sequence length increases, the total loss also increases.



(a) The accuracy rate vanilla RNN



(b) Total loss vanilla RNN

1.1.4 Question 1.4

With the traditional SGD, it is obvious that the information before certain time steps will be lost due to backpropagation and the multiplication with learning rate. Therefore, there are more advanced optimizing algorithms which is able to adjust the learning rate during the training process.
The RMSprop for instance, the learning rate adjusts and incorporates with the concept of momentum. The momentum accumulates an exponential decaying moving average of past gradients and allow the gradient updating to move in their direction. The gradient then times the learning rate and divide the exponentially decaying average of squared gradients. In the end the new gradient could be updated by the original way. RMSprop allow large gradients to be propgated in a tamed way while small gradients to be backpropogated aggressively. The whole process could be visualized as below:

$$
\begin{aligned}
r_t &= \alpha r_{t-1} + (1-\alpha) \odot g_t^2 \\
u_t &= -\frac{\eta}{\sqrt{r_t} + \epsilon} \\
w_{t+1} &= w_t + \eta_t u
\end{aligned}
$$

ADAM could be regarded as a combination of RMSprop, momentum and bias correction. ADAM records the momentum in addition to storing an exponential decaying average of past gradient. For parameter updating, ADAM incoprates momentum parameter in the formula directly. Furthermore, ADAM also incorporates bias corrections to estimate of the first-order and second-order moment. In this way, any bias exists during the initializing phase is adjusted. The concept could be better illustrated with formulas below:

$$
\begin{aligned}
m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\
v_t &= \beta_2 v_{t-1} + (1-\beta_2) g_t^2 \\
\hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\
\hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\
u_t &= -\eta_t \frac{\hat{m}_t}{\sqrt{\hat{v}_t + \epsilon}} \\
w_{t+1} &= w_t + u_t
\end{aligned}
$$

1.2 LSTM in PyTorch

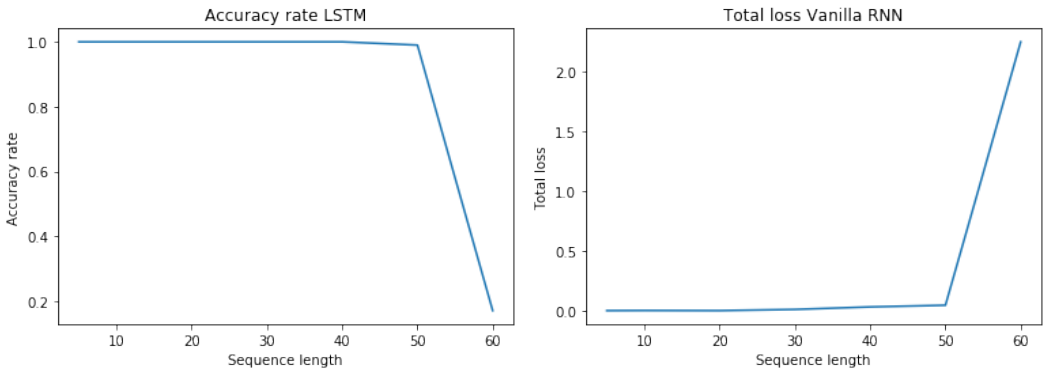
1.2.1 Question 1.5

(a) Input modulation gate: This gate deals with information from input gate and forget gate. It makes sure that that current and candidate cell states multiply the sigmoid function. In this way it drops old values and update the cell state with new values
Input gate: The input gate determines which values to be updated, and it creates a candidate cell state value $\tilde{C}_t$ to store the final input values.
forget gate: This gate determines which values in the cell state $C_{(t-1)}$ should be maintained. The output will be between 0 and 1 for each value.
output gate: It determines what values to be output. At first it runs a sigmoid layer to determine which parts of cell state it is going to output. Then it incorporates the current cell state with the sigmoid layer to actually output the necessary information.
(b) If we only consider cell state parameters as defined above, in total there will be : $4dn + 4n^2 + 4n$ parameters to be trained on. for $w_{gx}, w_{ix}, w_{fx}, w_{ox}$, the trainable parameters will be dn each. Similarly, $w_{gh}, w_{ih}, w_{fh}, w_{oh}$ will have n^2 trainable parameters and n for b_g, b_i, b_f, b_o.

1.3 Question 1.6

During the experiment, the learning rate is set as default for sequence length 10, then it increases to 0.01 for sequence length above 20. Compared with the RNN, LSTM tends to perform much better with four gates. These gates insures that past information could be clearly remembered and retained

within the current network. Thus, long-term dependency could be captured with the LSTM. If we check the graph below, it is clear that the model performs really well in capturing the character information of length 50. Then it decreases significantly with length 60.



(a) The accuracy rate LSTM

(b) Total loss LSTM

2 Modified LSTM Cell

2.1 Question 2.1

The parameter r_{on} represents the ratio of the time that $k > 0$ to the total time period τ. If the parameter is larger, then the gate opening time will be longer, the opening time gap will be longer correspondingly.

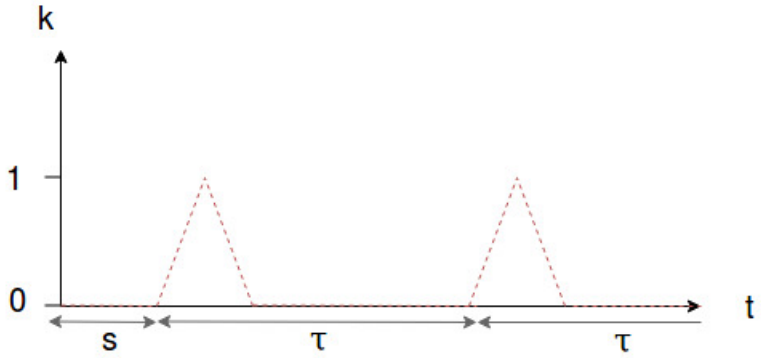


Figure 3: The conceptual diagram of the LSTM

2.2 Question 2.2

According to the formulas and drawing, the $k^{(t)}$ could act as a gate to control the inflow of the state information from previous time steps.

$$c^{(t)} = k^{(t)} \odot \tilde{c}^{(t)} + (1 - k^{(t)}) \odot c^{(t-1)}$$
$$h^{(t)} = k^{(t)} \odot \tilde{h}^{(t)} + (1 - k^{(t)}) \odot h^{(t-1)}$$

Here when $k^{(t)} < 1$ Information of the current state will be used for training automatically. This leads to the effect that a initial memory state c_0 only decays during the period when the forget gate is opened. When the forget gate is closed ($k^{(t)} = 1$), the forget gate will keep the full memory of a certain state. During a single period τ, units only update during a duration of $r_{on} \times \tau$, which could significant lowers the number of updates during a training. Due to the cyclic memory, the model has much longer and adjustable memory length via the parameter τ.
The gate allows undecayed gradients to be backpropogated through fewer time steps and faster

convergence. In the real-life, the LSTM model could ensure that inputs sampled in asynchronous times could be processed. It also imitate the function of biological neurons, which stores memory in asynchronous time.

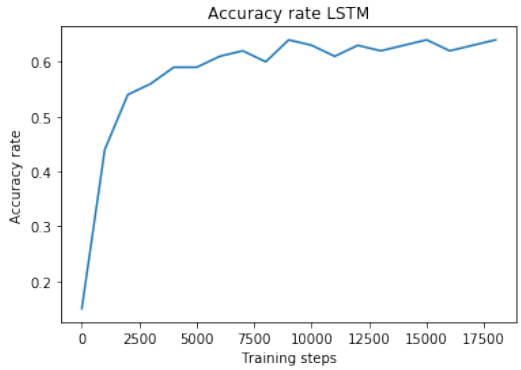
2.3 Question 2.3

τ represents the total time period of one cycle. r_{on} is the ratio of the gate to be kept opened against the total time period τ and s controls the phase shift of each cell within the LSTM.
If the dataset is huge enough, all parameters could be leaned with sufficient amount of epochs and parameter updates.

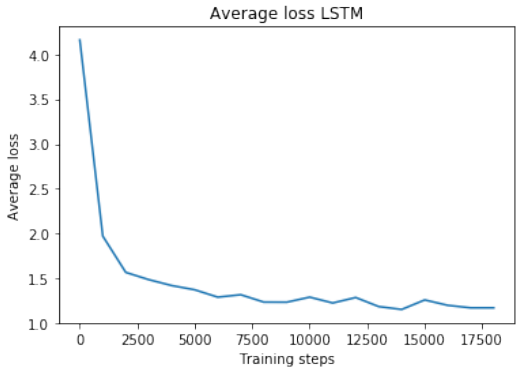
3 Recurrent Nets as Generative Model

3.1 Question 3.1

(a) In this section, I built the LSTM model via pytorch modules and trained it with the book "Democracy in United States." During the training process, it is obvious that model accuracy increases to 0.63 quickly after 7000 training steps. Then the model staggers at that point for a long training steps and the accuracy rate seldom gets improved. In the average loss diagram, the average loss also decreases to around 1.2. Subsequently, the average loss does not decrease anymore.

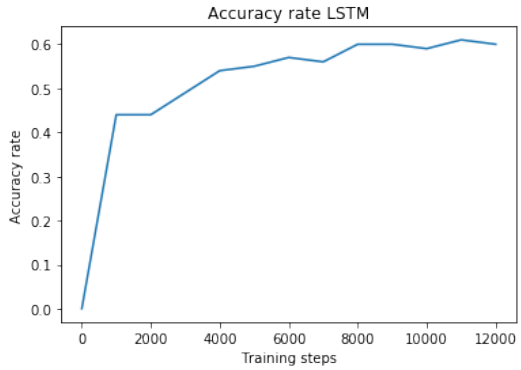


(a) The accuracy rate LSTM traine on "Democracy in US"

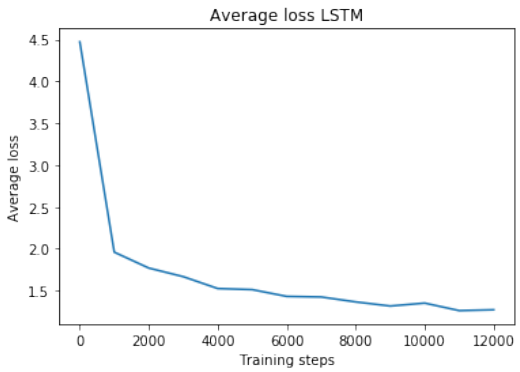


(b) Total loss LSTM traine on "Democracy in US"

When I train the model with "Gimms fairy tales", the final accuracy rate and average loss is quiet similar. The total loss and accuracy rate stablize at around 7000 steps.



(a) The accuracy rate LSTM trained on Grimms fairy tales



(b) Total loss LSTM trained on Grimms fairy tales

(b) In the below table, generated sentences along the training process are shown:

Training step	example
0	" Uaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"
1000	"ch the proserse of the proserse"
2000	"Perno are the same to the same"
3000	", and the present the present t"
4000	"! the present the property of t"
5000	"y the properly to the properly"
6000	"be to the present to the contra"
7000	"; and the provincial and the pr"
8000	"7 the property of the property"
9000	"Nearly contrary to the same pow"
10000	"which they are the principle of"
12000	"Qern the same time the same tim"
13000	"But the same time the same time"
14000	"untry in the constitution of th"
15000	"lief into the contrary to the c"
16000	"? The present time the same pre"
17000	" The people is the same time t"
18000	" In the United States the law o"

Table 1: Generated samples of models trained by "Democracy in United States"

It is obvious that the model does not generate comprehensible sentences at first. However, as training progresses, the number of repeated words become less and less. Sentences are becoming more interpretable and generated sentences are all regarding the topic.

Training step	example
0	" Jwwwwwwwwwwwwwwwwwwwwwwwwwwwww"
1000	"$_{t}hesaidthesaidthesaidth$"
2000	" was the was the was the was the"
3000	" Kand the said the was and the s"
4000	" !' said the world and said the"
5000	" ing to the world and said to th"
6000	"ou will not see the cat with th"
7000	" -tree in the world the window"
8000	" 4 with the world the world the"
9000	"y the stars and said: 'I will b"
10000	" 57 of the stone and said, 'I wi"
11000	"nd the second the second the se"
12000	"Now the wood and said: 'I will"

Table 2: Generated samples of models trained by "Grimms fairy tales"

(c) During the training process, I saved the modern and reload the model to generate sentences with different temperatures with $generate.py$.
With the temperature of 2, β -the exponential multiplier becomes 0.5. Samples generated from softmax layer becomes more evenly distributed, this results in random sampling of predictions. Sentences generated from this model do not make sense in general. Generated sentences in general are more likely to be divided into different paragraphs.
With the temperature set as 0.5, β becomes 2. The difference for samples generated from the final layer becomes more obvious. This results in the fact that some characters are more likely to be sampled compared to others. Samples generated from during training are displayed in the table below:

Temperature	example
0.5	" 6 And the little bride shall b"
0. 5	"be face and with the wolf had s"
0.5	"'I will not be your hand. The f"
0.5	" -th in the man make his dearest"
0.5	" Queen had done fast, and said,"
2	" Fox on. JOweld suwifliky, can,"
2	"Tdo,. One,' said' "Diddlo ume;"
2	"$\%CKLi/vitevizyfear; ircuse$,"
2	"JOd thwirs,' dancever went! EBo"
2	"] Thery plfuntuing 1.DBk glte-a"

Table 3: Generated samples of models trained by "Grimms fairy tales" with different temperatures