CS584: Natural Language Processing **Spring 2020**

Lecture: More on RNNs

Lecturer: Yue Ning *Scribe: Linsen Li*

Note: *LaTeX template courtesy of UC Berkeley EECS dept.*

1.1 Introduction

In the previous lecture we have introduced Recurrent Neural Networks(RNNs). The input of RNNs model is a sequence. The output of the model can be a single vector or a sequence. An RNNs structure has a sequence of hidden layer. At each time-step, there are two inputs to the hidden layer: the output of the previous layer h_{t-1}, and the input at that timestep x_t. The main idea for RNNs is it apply the same weight matrix to each layers.

Unlike general fully connected neural network, RNNs's input is always longer than other neural network structure. Recall that in neural network we use backpropagation to update the weight matrix. If the path of parameter passing is long, some problems like vanishing (or exploding) gradient problem may happen. This lecture is to find ways to solve these problem. For example, Long Short-Term Memory (LSTM) and Gated Recurrent Units (GRU) are two of the method to solve this problem.

Besides, we will also introduce more fancy RNNs variants, such as Bidirectional RNNs and Stacked RNNs. These technique is aim to improve the performance of the RNNs in a particular Nature Language Processing task. For example Bidirectional RNNs can be used in sentiment classification while it can not be used in text generation.

1.2 Background

Before we start, we need to define some mathematics symbol for the Model:

- $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots x_T$:the word vectors corresponding to a corpus with T words.
- h_t=σ ($\mathrm{W}_h \times \mathrm{h}_{t-1} + W_x \times \mathrm{x}_t$) : the relationship to compute the hidden layer output features at each time-step t
  - $x_t \in \mathrm{R}^d$: input word vector at time t.
  - $W_x \in \mathrm{R}^{D_h \times d}$: weights matrix used to condition the input word vector, x_t
  - $W_h \in \mathrm{R}^{D_h \times D_h}$: weights matrix used to condition the output of the previous time-step, h_{t-1}
  - $h_{t-1} \in \mathrm{R}^{D_h}$: output of the non-linear function at the previous time-step, $t1$.
  - $\sigma()$: : the non-linearity function (sigmoid here)

- $\hat{y}_t$=softmax ($W_S h_t$): the output probability distribution over the vocabulary at the each time-step t. Essentially, $\hat{y}_T$ is the output probability distribution over the vocabulary at the last time-step T. For many to one RNNs model, the output is $\hat{y}_T$. Here, $W_{(S)} \in R^{|V| \times D_h}$ and $\hat{y} \in R^{|V|}$ where $|V|$ is the vocabulary.

To simplify, we assume the output is the last hidden state $\hat{y}_T$, the label is y_T. Then the loss function is :

$$J^{(T)}(\theta) = -\sum_{j=1}^{|V|} y_{T,j} \times \log\left(\hat{y}_{T,j}\right) \tag{1.1}$$

Note that in this formula we just use the RNNs last hidden layer output as the model output. If we connect this output with dense layer, we can control the size of the output vector: for instance, RNNs connect with a dense layer can apply to text classification. If we connect this output with another RNNs structure,using the output of the first RNNs as the initial state for the second RNNs, it becomes the Sequence To Sequence (Seq2seq) model, which is also called Encoder-Decoder model. The first RNNs structure is the Encoder and the second RNNs structure is the Decoder. This Seq2seq model can be applied in machine translation.

So far we have defined the mathematics symbol for the RNN model, now we will explain the vanishing (or exploding) gradient problem.

1.2.1 Vanishing gradient problem

Recall the backpropagation,assume the loss function is $J^{(N)}(\theta)$,and we have a time step with 1,2,3,...,t,...N. From the chain rule, to calculate the partial derivative for h_1 we have:

$$\frac{\partial J^{(N)}}{\partial h_1} = \frac{\partial h_2}{\partial h_1} \times \frac{\partial h_3}{\partial h_2} \times \frac{\partial h_4}{\partial h_3} ... \times \frac{\partial h_{N-2}}{\partial h_{N-3}} \times \frac{\partial h_{N-1}}{\partial h_{N-2}} \times \frac{\partial J^{(N)}}{\partial h_{N-1}} \tag{1.2}$$

$$= \prod_{t=1}^{N-1} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{N-1}} \times \frac{\partial J^{(N)}}{\partial h_{N-1}} \tag{1.3}$$

Since we have:

$$h_1 = \sigma\left(W_h \times h_0 + W_x \times x_1\right) \tag{1.4}$$

$$= \sigma(z_1) \tag{1.5}$$

where

$$z_1 = \left(W_h \times h_0 + W_x \times x_1\right), z_1 \in \mathbb{R}^{D_h} \tag{1.6}$$

Then:

$$\frac{\partial h_1}{\partial W_h} = \frac{\partial h_1}{\partial z_1} \times \frac{\partial z_1}{\partial W_h} \tag{1.7}$$

$$= \sigma(z_1) \times (1 - \sigma(z_1)) \times \frac{\partial z_1}{\partial W_h} \tag{1.8}$$

$$= \sigma(z_1) \times (1 - \sigma(z_1)) \times h_0^T \tag{1.9}$$

$$\frac{\partial h_1}{\partial W_x} = \frac{\partial h_1}{\partial z_1} \times \frac{\partial z_1}{\partial W_x} \tag{1.10}$$

$$= \sigma(z_1) \times (1 - \sigma(z_1)) \times \frac{\partial z_1}{\partial W_x} \tag{1.11}$$

$$= \sigma(z_1) \times (1 - \sigma(z_1)) \times x_1^T \tag{1.12}$$

From all above, we can get:

$$\frac{\partial J^{(N)}}{\partial W_h} = \frac{\partial J^{(N)}}{\partial h_1} \times \frac{\partial h_1}{\partial W_h} \tag{1.13}$$

$$= \prod_{t=1}^{N-1} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \times \frac{\partial J^{(N)}}{\partial h_{N-1}} \times \sigma(z_1) \times (1 - \sigma(z_1)) \times h_0^T \tag{1.14}$$

$$= \frac{\partial J^{(N)}}{\partial h_{N-1}} \times \sigma(z_1) \times (1 - \sigma(z_1)) \times h_0^T \times \prod_{t=1}^{N-1} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \tag{1.15}$$

$$= M_1 \times F(N) \tag{1.16}$$

$$\frac{\partial J^{(N)}}{\partial W_x} = \frac{\partial J^{(N)}}{\partial h_1} \times \frac{\partial h_1}{\partial W_x} \tag{1.17}$$

$$= \prod_{t=1}^{T-1} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \times \frac{\partial J^{(N)}}{\partial h_{N-1}} \times \sigma(z_1) \times (1 - \sigma(z_1)) \times x_1^T \tag{1.18}$$

$$= \frac{\partial J^{(N)}}{\partial h_{N-1}} \times \sigma(z_1) \times (1 - \sigma(z_1)) \times x_1^T \times \prod_{t=1}^{N-1} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \tag{1.19}$$

$$= M_2 \times F(N) \tag{1.20}$$

where

$$M_1 = \frac{\partial J^{(N)}}{\partial h_{N-1}} \times \sigma(z_1) \times (1 - \sigma(z_1)) \times h_0^T, \tag{1.21}$$

$$M_2 = \frac{\partial J^{(N)}}{\partial h_{N-1}} \times \sigma(z_1) \times (1 - \sigma(z_1)) \times x_1^T, \tag{1.22}$$

$$F(N) = \prod_{t=1}^{N-1} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}}. \tag{1.23}$$

We can consider M_1, M_2 as $o(N)$ and F(N) is a function of N. That is, when N is large enough, we can ignore the effect of M_1, M_2 because they are relatively small. So, the main effect of the equation is F(N). When N is large, the value of F(N) will decide the value of the partial derivative $\frac{\partial J^{(N)}}{\partial W_h}$ or $\frac{\partial J^{(N)}}{\partial W_x}$. So, now we focus on F(N).Recall that:

$$h_t = \sigma\left(W_h \times h_{t-1} + W_x \times x_t\right) \tag{1.24}$$

From chain rule we can get:

$$\begin{aligned}
\frac{\partial h_t}{\partial h_{t-1}} &= \frac{\partial\left(W_h \times h_{t-1} + W_x \times x_t\right)}{\partial h_{t-1}} \times \frac{\partial h_t}{\partial\left(W_h \times h_{t-1} + W_x \times x_t\right)} && (1.25)\\
&= W_h^T \times \sigma\left(W_h \times h_{t-1} + W_x \times x_t\right) \times \left(1 - \sigma\left(W_h \times h_{t-1} + W_x \times x_t\right)\right)^T && (1.26)\\
&= W_h^T \times diag(\sigma\left(W_h \times h_{t-1} + W_x \times x_t\right) \times \left(1 - \sigma\left(W_h \times h_{t-1} + W_x \times x_t\right)\right)^T) && (1.27)\\
&= W_h^T \times diag(M) && (1.28)
\end{aligned}$$

where

$$M = \sigma\left(W_h \times h_{t-1} + W_x \times x_t\right) \times \left(1 - \sigma\left(W_h \times h_{t-1} + W_x \times x_t\right)\right)^T \tag{1.29}$$

M is symmetric matrix. Moreover, basing on the boundedness of sigmoid function. The eigenvalue of matrix M is also boundedness, which is between -1 and 1 Then we can get:

$$\begin{aligned}
F(N) &= \prod_{t=1}^{N-1} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} && (1.30)\\
&= \prod_{t=1}^{N-1} W_h^T \times diag(M) && (1.31)\\
&= (W_h^T)^{N-1} \times \prod_{t=1}^{N-1} diag(M) && (1.32)
\end{aligned}$$

As we have talked about before, the boundedness of singular value of matrix M makes $\prod_{t=1}^{N-1} diag(M)$ also boundedness. Then the value of F(N) depends on $(W_h^T)^{N-1}$. Actually it depends on the largest eigenvalue of matrix W_h. Assume λ_{max} is the largest eigenvalue of matrix W_h.Then we have the following equation:

$$\lim_{N \to +\infty} F(N) = \begin{cases} +\infty, & \lambda_{max} > 1, \\ 0, & \lambda_{max} \leq 1 \end{cases} \tag{1.33}$$

which will cause:

$$\lim_{N \to +\infty} \frac{\partial J^{(N)}}{\partial W_h} = \begin{cases} +\infty, & \lambda_{max} > 1, \\ 0, & \lambda_{max} \leq 1 \end{cases} \tag{1.34}$$

As result, the Vanishing gradient problem happens when $\lambda_{max} \leq 1$. Note that in the real world, N can't be $+\infty$. The above equation just talk about the most extreme case. However, N is always large in RNNs model. This will make this Vanishing gradient a definite problem for RNNs model.

When Vanishing gradient problem happens, the RNNs won't work. As we all know the method for update the weight W in deep neural network is backpropagation. If the gradient is nearly zero, then the update for this parameter is nearly zero. Since the gradient is to measure the effect for the particular parameter on loss function .This problem will make the model forget the information from the early input of the sequence.

1.2.2 Exploding gradient problem

For the same reason, Exploding gradient problem happens when $\lambda_{max} > 1$. In this case, the gradient will be very large in some place. Recall that we use Stochastic Gradient Descent algorithm in weight updating:

$$\theta^{\text{new}} = \theta^{\text{old}} - \eta \nabla_\theta J(\theta) \tag{1.35}$$

$$= \theta^{\text{old}} - \eta \frac{\partial J^{(N)}}{\partial W_h} \tag{1.36}$$

η is the learning rate. When Exploding gradient problem happens, $\frac{\partial J^{(N)}}{\partial W_h}$ is very large. This will make the update step very large, which is very like a "cliff". This will make it difficult to converge to a local minimal.

Some researchers find way that to set a threshold to avoid this Exploding gradient problem. The idea is: if the gradient is very large in some place, they will make a change to the shrink the gradient to a small value by divide them by their norm but in the same direction.

1.3 Model

So far we have discussed the vanishing (or exploding) gradient problem. Now we are going to introduce two RNNs models(LSTM and GRU) that alleviate these problems. We will also introduce two more RNNs structure(Bidirectional RNNs and Stacked RNNs).These two are help to improve the performance of RNNs in some NLP tasks.

1.3.1 Long Short-Term Memory (LSTM)

Long Short Term Memory networks (LSTM) is a special kind of RNN, which are capable of learning long-term dependencies. They were introduced by Hochreiter Schmidhuber (1997).

The main difference between LSTM and SimpleRNN is that LSTM has a Conveyor belt, which make the past information directly flow to the future. LSTM also has forget gate and input gate, which are use to control the information flow in Conveyor belt. The Conveyor belt mainly have three processes:

- Forget gate(f_t): the following is the formula for computing forget gate:

$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] + b_f\right) \tag{1.37}$$

  where f_t is the output of forget gate, W_f is the parameter matrix for forget gate, b_f is the intercept. Note that f_t is a vector which has the same dimension as the Conveyor belt state dimension. The $\sigma(.)$ function will bound the value between 0 and 1. So, the value of vector f_t are all between 0 and 1. A value of 0 means 'left nothing through' while a value of 1 means 'let everything through'.

- Input gate(i_t): the following is the formula for computing input gate:

$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] + b_i\right) \tag{1.38}$$

  where i_t is the output of input gate, W_i is the parameter matrix for input gate, b_i is the intercept. Input gate(i_t) decides which values of the Conveyor belt we will update.

- New value($\tilde{C}_t$): the following is the formula for computing new value($\tilde{C}_t$):

$$\tilde{C}_t = \tanh\left(W_C \cdot [h_{t-1}, x_t] + b_C\right) \tag{1.39}$$

  where W_C is the parameter matrix for input gate, b_C is the intercept. New value($\tilde{C}_t$) is also a vector which has the same dimension as the Conveyor belt state dimension. It is to be added to the Conveyor belt.

So far we have defined the three items in Conveyor belt. Now we can use the following formula to update Conveyor belt:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{1.40}$$

where f_t is the output of forget gate, i_t is the output of input gate, C_{t-1} is the previous cell state, $\tilde{C}_t$ is the new value.

To update the hidden state, LSTM has a output gate o_t:

$$o_t = \sigma\left(W_o\left[h_{t-1}, x_t\right] + b_o\right) \tag{1.41}$$

where W_o is the parameter matrix for input gate, b_o is the intercept. Output gate(o_t) is to decide what flows from the Conveyor belt C_{t-1} to the state h_t.

At last, we can use the following formula to update the hidden state h_t:

$$h_t = o_t * \tanh\left(C_t\right) \tag{1.42}$$

where o_t is the output gate and C_t is the current state for the Conveyor belt

So far we have illustrate how the LSTM works.Note that LSTM has four times more parameters than SimpleRNN, they are W_f, W_i, W_C, W_o. This fact will make the computation more time consuming.

1.3.2 Gated Recurrent Units(GRU)

The idea of Gated Recurrent Units(GRU) is similar to LSTM. GRU has two gates:

- Update gate(u_t):

$$u_t = \sigma\left(W_u \cdot [h_{t-1}, x_t] + b_u\right) \tag{1.43}$$

  where u_t is the output of update gate, W_u is the parameter matrix, b_u is the intercept.

- Reset gate(r_t)

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t] + b_r\right) \tag{1.44}$$

  where r_t is the output of reset gate, W_r is the parameter matrix, b_r is the intercept.

Note that u_t and r_t are two vector which have the same dimension as the hidden state. The value in these two vectors are between 0 and 1. For GRU, we need to calculate the new hidden state content($\tilde{h}_t$):

$$h'_{t-1} = h_{t-1} \odot r_t \tag{1.45}$$

$$\tilde{h}_t = \tanh\left(W_{h'} \cdot \left[h'_{t-1}, x_t\right] + b_{h'}\right) \tag{1.46}$$

where $W_{h'}$ is the parameter matrix, $b_{h'}$ is the intercept,$\odot$ means point-wise multiply. The new hidden state content $\tilde{h}_t$ is a vector with the same dimension as hidden state h_t. The $tanh(.)$ function makes the value in this vector between -1 and 1.

Now we can update the hidden state with update gate u_t and new hidden state content $\tilde{h}_t$:

$$h_t = (1 - u_t) \odot h_{t-1} + u_t \odot \tilde{h}_t \tag{1.47}$$

So far we have illustrate how GRU works. Note that GRU have three times more parameter than SimpleRNN,they are W_u, W_r, $W_{h'}$. The advantage of GRU is that it is less time consuming than LSTM. There is no evidence that which of these two RNNs is better than the other one.

1.3.3 Bidirectional RNNs

In NLP, the order of the sentence sequence is important. Sometimes if you change the order of the sequence, the meaning is different. However, there are some NLP tasks that the order is not the big deal, such as sentiment analysis. As we have discussed previously, if the input sequence is long for the RNNs model, it is more likely to loss the information from the beginning. In such case, researcher build Bidirectional RNNs to alleviate this problem.

The idea of Bidirectional RNNs is very simple. At each time-step, t, this Bidirectional network maintains two hidden layers, one for the left-to-right propagation and another for the right-to-left propagation. These two directions propagation are independent, which means that they don't share any parameter. The final classification result, $\hat{y}_t$, is generated through combining the score results produced by both RNN hidden layers. The following is the equations for Bidirectional RNNs:

$$\vec{h}_t = f\left(\vec{W}_x x_t + \vec{W}_h \vec{h}_{t-1} + \vec{b}\right) \tag{1.48}$$

$$\overleftarrow{h}_t = f\left(\overleftarrow{W}_x x_t + \overleftarrow{W}_h \overleftarrow{h}_{t+1} + \overleftarrow{b}\right) \tag{1.49}$$

$$\hat{y}_t = g\left(U h_t + c\right) = g\left(U\left[\vec{h}_t; \overleftarrow{h}_t\right] + c\right) \tag{1.50}$$

where equation(1.48) is the recursive formula for left-to-right propagation and equation(1.49) is the recursive formula for right-to-left propagation. Both of them are the same as the processing we have talked about previously. The only difference is that it concatenate the output of the above two propagation as the finial output h_t. Then, feed h_t to the dense layer.

Note that Bidirectional RNNs is only accessible when you have the entire input sequence. For instance, in language modeling you can't use Bidirectional RNNs. In sequence to sequence(seq2seq) model, you can only use Bidirectional RNNs in encoder but not in decoder.

1.3.4 Stacked RNNs

So far the RNNs we have discussed are all one layer.But RNNs can also be multi-layered. The idea is similar to other deep neural network like convoluted neural network. The lower RNNs are responsible for lower-level features and the higher RNNs are responsible for higher-level features. Assume we we have M layers RNNs, the recursive equation for this Stacked RNNs is:

$$\begin{cases} h_t^{(1)} = f\left(W_x{}^{(1)} x_t^{(1)} + W_h^{(1)} h_{t-1}^{(1)} + b^{(1)}\right) \\ h_t^{(2)} = f\left(W_x{}^{(2)} x_t^{(2)} + W_h^{(2)} h_{t-1}^{(2)} + b^{(2)}\right) \\ h_t^{(3)} = f\left(W_x{}^{(3)} x_t^{(3)} + W_h^{(3)} h_{t-1}^{(3)} + b^{(3)}\right) \\ \vdots \quad \vdots \quad \vdots \\ h_t^{(i)} = f\left(W_x{}^{(i)} x_t^{(i)} + W_h^{(i)} h_{t-1}^{(i)} + b^{(i)}\right) \\ \vdots \quad \vdots \quad \vdots \\ h_t^{(M)} = f\left(W_x{}^{(M)} x_t^{(M)} + W_h^{(M)} h_{t-1}^{(M)} + b^{(M)}\right) \end{cases}$$

Note that in each RNNs layer $h^{(i)}$,where i from 1 to M, the recursive relation between $h_t^{(i)}$ and $h_{t-1}^{(i)}$ is the same as the SimpleRNNs we have discussed above. The output of each $h^{(i)}$ will feed the next layer $h^{(i+1)}$. This makes the RNNs become the multi-layer.

Stacked RNNs often has bettern performance than one-layer RNNs. We should always use Stacked RNNs as default.

1.4 Conclusion

In this lecture note, we have discussed the Vanishing (or Exploding) gradient problems in RNNs. After that, we introduce two RNNs structure: LSTM and GRU, which can alleviate these problems. At last, we introduce two more fancy RNNs: Bidirectional RNNs and Stacked RNNs.

In conclusion, LSTM and GRU are more time consuming than SimpleRNN. However, they always have better performance than SimpleRNNs and we should use them by default. Besides, Bidirectional RNNs should be considered when the entire input sequence is accessible. Stacked RNNs should also be used by default.

References

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