ID2223
Scalable Machine Learning and Deep Learning

Review Questions 4

FALL 2021

Yuchen Gao
Weikai Zhou

December 8, 2021

KTH EECS Embedded Systems

1 What's the vanishing problem in RNN?

We have got the vanishing problem in back-propagation when we want to find the global minimum of the cost function. During the training, the cost function is compared with outcomes to the desired output, computing the difference between y and $\hat{y}$. We want to propagate the cost function back through the network as a result to update the weights. With RNN, it's not just the neurons directly below this output layer that contributed but all of the neurons far back in time. Instead, we have to propagate all the way back through time to these neurons. So, we multiply with the same weight multiple times, and this is where the problem arises: when we multiply something by a small number, the value decreases very quickly. And it will become so small that the gradient will be almost zero. Then it is hard for the network to update the weights.

2 Explain the impact of different gates in LSTM.

There are three types of gates in LSTM as shown in Figure 1: **Forget Gate**, **Input Gate** and **Output Gate**

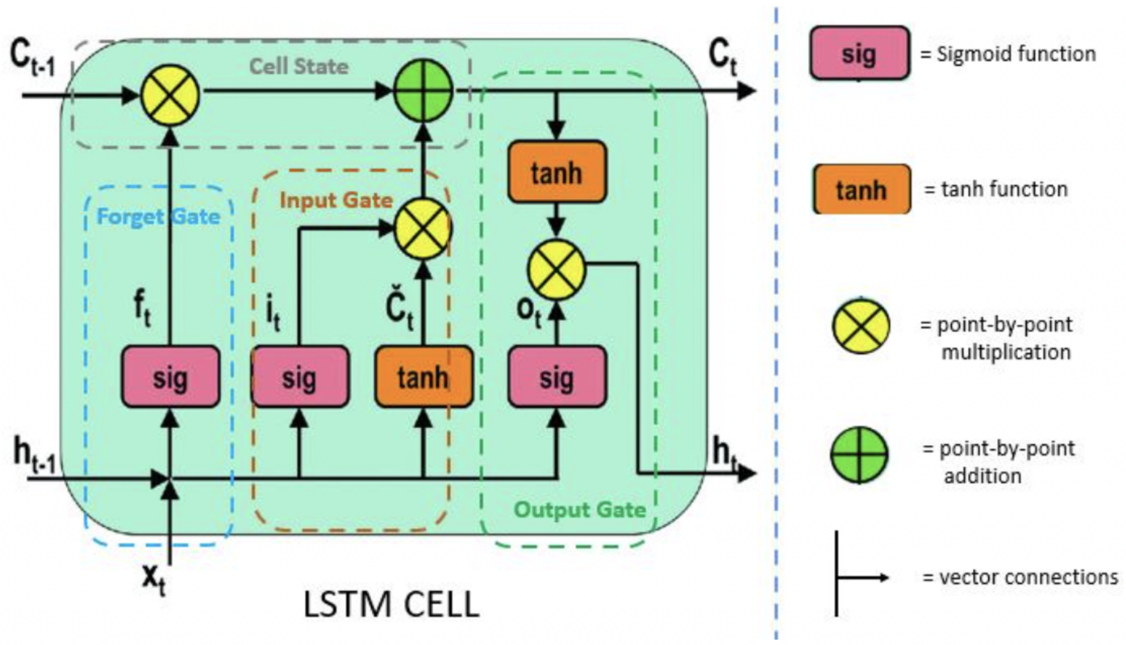

Figure 1: figure for question2[1]

- The forget gate decides what information we are going to ignore from the previous state. We make decision by using the Sigmoid function, it takes input $h^{(t-1)}$ and $x^{(t)}$, and outputs a number between 0 and 1 for each number in the state $C^{(t-1)}$. It concludes whether the part of the old output is necessary (by giving the output closer to 1) generating $f^{(t)}$.
- The input gate updates the cell status. We input the $x^{(t)}$ and previously hidden state $h^{(t-1)}$ into sigmoid function to make decision of which information is updated generating i^{t}. Next,

[1]Source: `https://www.pluralsight.com/guides/introduction-to-lstm-units-in-rnn`

the same information of the hidden state and current state will be passed through the tanh function generating $\tilde{C^{(t)}}$ to decide the new candidates of values that can be added to the state.

- The output gate generates the output $h^{(t)}$. Firstly, updating the old cell state $C^{(t-1)}$ into new one $C^{(t)}$. We multiply the old state by $f^{(t)}$, forgetting the things we decided to forget earlier. Then we add it $i^{(t)} \otimes \tilde{C}^{(t)}$. The $C^{(t-1)} \otimes f^{(t)} + i^{(t)} \otimes \tilde{C}^{(t)}$ is the new candidate $C^{(t)}$. Then we have the output:
$$o^{(t)} = \sigma(u_o^T x^{(t)} + w_o h^{(t-1)})$$
$$h^{(t)} = o^{(t)} \otimes tanh(C^{(t)})$$

3 Assume the error of the following network is $E = E^{(1)} + E^{(2)}$, then compute the $\frac{\partial E}{\partial u}$ (Figure 2).

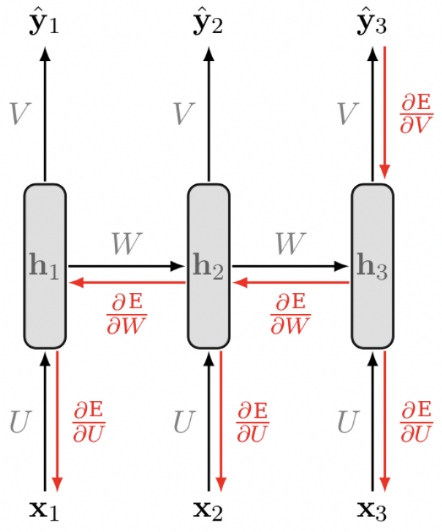

Figure 2: Figure for question 3

$$\frac{\partial E}{\partial u} = \frac{\partial J^{(1)}}{\partial u} + \frac{\partial J^{(2)}}{\partial u}$$
$$\frac{\partial J^{(1)}}{\partial u} = \frac{\partial J^{(1)}}{\partial \hat{y}^{(1)}} \frac{\partial \hat{y}^{(1)}}{\partial z(1)} \frac{\partial z^{(1)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial u}$$
$$\frac{\partial J^{(2)}}{\partial u} = \frac{\partial J^{(2)}}{\partial \hat{y}^{(2)}} \frac{\partial \hat{y}^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial s^{(2)}} \frac{\partial s^{(2)}}{\partial u} + \frac{\partial J^{(2)}}{\partial \hat{y}^{(2)}} \frac{\partial \hat{y}^{(2)}}{\partial z^{(2)}} \frac{\partial z^{(2)}}{\partial h^{(2)}} \frac{\partial h^{(2)}}{\partial s^{(2)}} \frac{\partial s^{(2)}}{\partial h^{(1)}} \frac{\partial h^{(1)}}{\partial s^{(1)}} \frac{\partial s^{(1)}}{\partial u}$$

4 Explain how to use a seq-to-seq model for language translation.

It takes a sequence of inputs and produces a sequence of outputs. It contains an encoder-decoder model as shown in Figure 3. The encoder outputs a final state vector (memory) which becomes the initial state for the decoder.

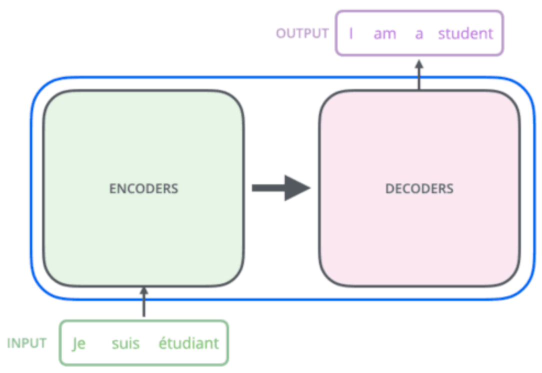

Figure 3: Structure of the model

The internal structure of encoder is shown below (Figure 4).

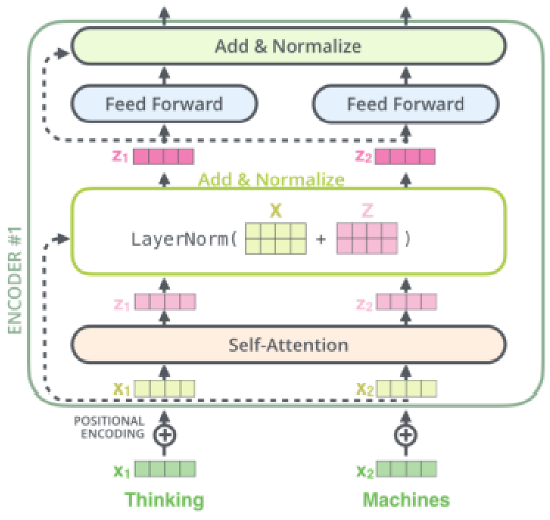

Figure 4: Internal structure of encoder

It contains the following parts:

- Multi-headed self-attention
- Feedforward NN (FC 2 layers)
- Skip connections
- Layer normalization - Similar to batch normalization but computed over features (words/tokens) for a single sample

Attention basically means mimics the retrieval of a value v_i for a query q based on a key k_i in a database, but in a probabilistic fashion. We can calculate as

$$Attention(q,\ K,\ V) = \sum_i Similarity(q,\ k_i) \cdot v_i = \sum_i \frac{e^{q \cdot k_i/\sqrt{d_k}}}{\sum_j e^{q \cdot k_j/\sqrt{d_k}}} v_i$$

And we can stack multiple queries into a matrix Q

$$Attention(Q,\ K,\ V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

Self-attention basically means letting the word embedding be the queries, keys and values, i.e. let the words select each other. Therefore, each word needs to do the same operation with all other words, even including itself. And we will use multi-head self-attention, which calculates attention separately in different attention heads to direct the overall attention.

There is also a positional encoding to input embeddings to let model learn relative positioning.

$$PE(pos,\ 2i) = \sin(\frac{pos}{10000^{2i/d_{model}}})$$

$$PE(pos,\ 2i+1) = \cos(\frac{pos}{10000^{2i/d_{model}}})$$

The internal structure of decoder is shown below (Figure 5).

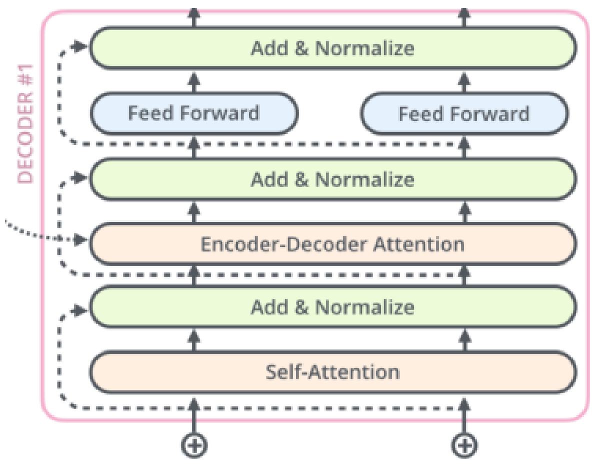

Figure 5: Internal structure of decoder

It contains the following parts:

- The output from the decoder is passed through a final fully connected linear layer with a softmax activation function
- Produces a probability distribution over the pre-defined vocabulary of output words (tokens)
- Greedy decoding picks the word with the highest probability at each time step

After picking the word with the highest probability at each time step, the translation is done.

5 Briefly explain the attention and self-attention mechanisms.

Attention mechanism: It is a technique that mimics cognitive attention, mimics the retrieval of a value v_i for a query q based on a key k_i in a database, but in a probabilistic fashion. The queries, keys and values are vectors. The output is a weights sum of the values. The weights are computed as the scaled dot-product (similarity) between the query and the keys. We can calculate as

$$Attention(q,\ K,\ V) = \sum_i Similarity(q,\ k_i) \cdot v_i = \sum_i \frac{e^{q\cdot k_i/\sqrt{d_k}}}{\sum_j e^{q\cdot k_j/\sqrt{d_k}}} v_i$$

And we can stack multiple queries into a matrix Q

$$Attention(Q,\ K,\ V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

The self-attention mechanism: It is similar to attention. They fundamentally share the same concept and many common mathematical operations. The attention mechanism allows output to focus attention on input when producing output. While the self-attention model allows inputs to interact with each other. For each input, the query, keys and values will be generated. It relates different positions of a single sequence in order to compute a representation of the sequence.