KTH Royal Institute of Technology
Stockholm

School of Electrical Engineering and
Computer Science

Scalable Machine Learning and Deep Learning - ID2223

Review Questions 4

Author
Emil Ståhl

Author
Erik Kongpachith

Author
Selemawit Fsha Nguse

December 4, 2021

1 What's the vanishing problem in RNN?

In recurrent neural networks, the gradient of each layer is calculated as product of gradients from previous layers. As we get deeper, the gradient starts decreasing resulting in multiplications of factors smaller than 1 thus resulting in a vanishing gradient.

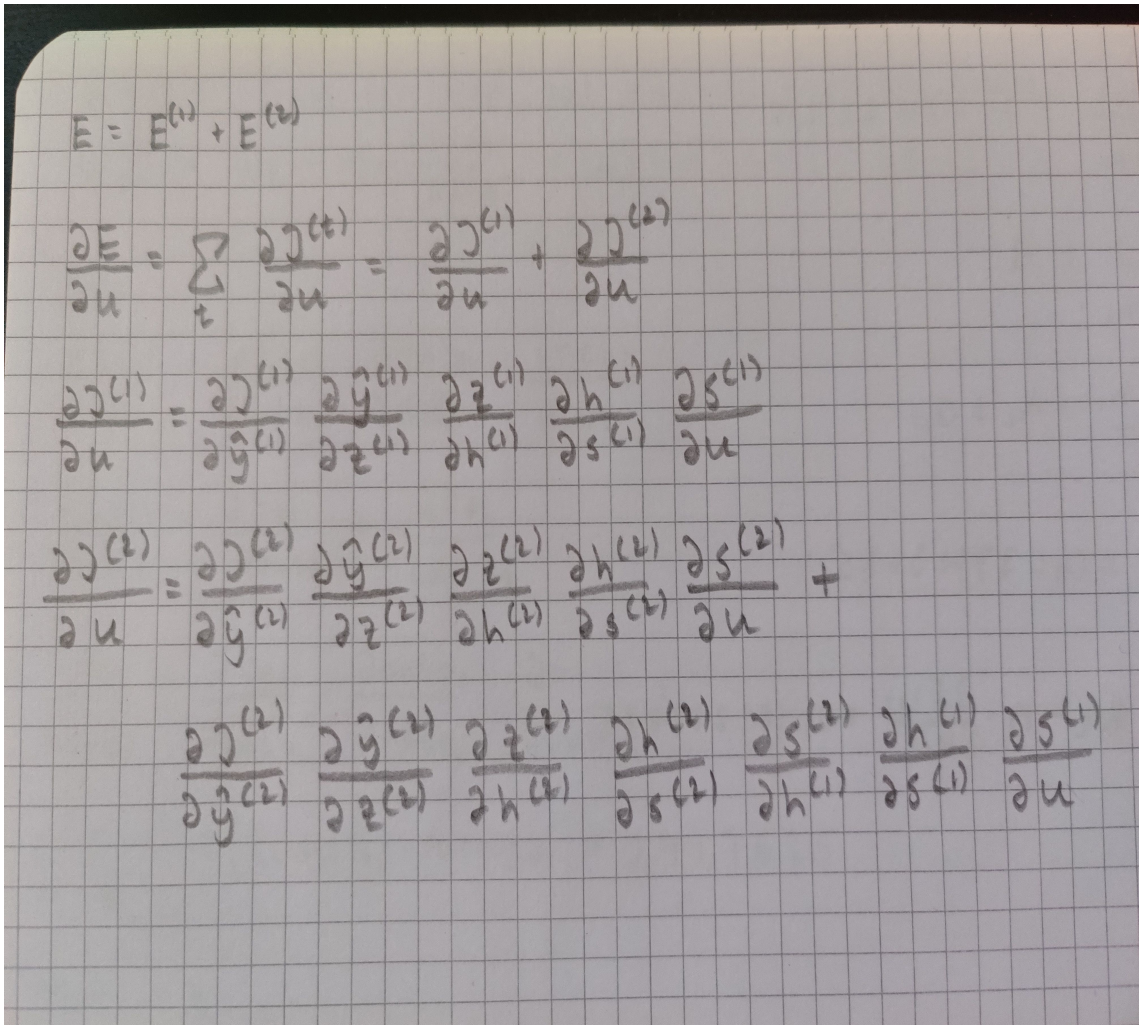
2 Explain the impact of different gates in LSTM.

Recurrent Neural Networks suffer from short-term memory. If a sequence is long enough, they'll have a hard time carrying information from earlier time steps to later ones. So if you are trying to process a paragraph of text to do predictions, RNN's may leave out important information from the beginning. LSTM 's and GRU's were created as the solution to short-term memory. They have internal mechanisms called gates that can regulate the flow of information. The gates are different neural networks that decide which information is allowed on the cell state. The gates can learn what information is relevant to keep or forget during training. LSTM has three different types of gates, which are:

- **Forget gate** - This gate decides what information should be thrown away or kept. Information from the previous hidden state and information from the current input is passed through the sigmoid function. Values come out between 0 and 1. The closer to 0 means to forget, and the closer to 1 means to keep.
- **Input gate** - To update the cell state, we have the input gate. First, we pass the previous hidden state and current input into a sigmoid function. That decides which values will be updated by transforming the values to be between 0 and 1. 0 means not important, and 1 means important. You also pass the hidden state and current input into the tanh function to squish values between -1 and 1 to help regulate the network. Then you multiply the tanh output with the sigmoid output. The sigmoid output will decide which information is important to keep from the tanh output.
- **Output Gate** - The output gate decides what the next hidden state should be. Remember that the hidden state contains information on previous inputs. The hidden state is also used for predictions. First, we pass the previous hidden state and the current input into a sigmoid function. Then we pass the newly modified cell state to the tanh function. We multiply the tanh output with the sigmoid output to decide what information the hidden state should carry. The output is the hidden state. The new cell state and the new hidden is then carried over to the next time step.

Summary: To review, the Forget gate decides what is relevant to keep from prior steps. The input gate decides what information is relevant to add from the current step. The output gate determines what the next hidden state should be.

3 Assume the error of the following network is E = E(1) + E(2), then compute the dE/du.



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

Introduced for the first time in 2014 by Google, a sequence to sequence model aims to map a fixed-length input with a fixed-length output where the length of the input and output may differ. Sequence-to-Sequence (Seq2Seq) modelling is about training models that can convert sequences from one domain to sequences of another domain, for example, English to Swedish. This Seq2Seq modelling is performed by the LSTM encoder and decoder. There are multiple ways to handle this task, there among using RNNs. The model consists of 3 parts: encoder, intermediate (encoder) vector and decoder.

Encoder

- A stack of several recurrent units (LSTM or GRU cells for better performance) where each accepts a single element of the input sequence, collects information for that element and propagates it forward.
- Each word is represented as x_i where i is the order of that word.
- The hidden states h_i are computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1} + W^{(hx)} x_t)$$

Encoder vector

- This is the final hidden state produced from the encoder part of the model. It is calculated using the formula above.
- This vector aims to encapsulate the information for all input elements in order to help the decoder make accurate predictions.
- It acts as the initial hidden state of the decoder part of the model.

Decoder

- A stack of several recurrent units where each predicts an output y_t at a time step t.
- Each recurrent unit accepts a hidden state from the previous unit and produces and output as well as its own hidden state.
- In the question-answering problem, the output sequence is a collection of all words from the answer. Each word is represented as y_i where i is the order of that word.

- Any hidden state h_i is computed using the formula:

$$h_t = f(W^{(hh)} h_{t-1})$$

- The output y_t at time step t is computed using the formula:

$$y_t = softmax(W^S h_t)$$

We calculate the outputs using the hidden state at the current time step together with the respective weight W(S). Softmax is used to create a probability vector which will help us determine the final output (e.g. word in the question-answering problem). The power of this model lies in the fact that it can map sequences of different lengths to each other. As you can see the inputs and outputs are not correlated and their lengths can differ. This opens a whole new range of problems which can now be solved using such architecture.

5 Briefly explain the attention and self-attention mechanisms.

- **Attention** - this is one of the most influential ideas in the Deep Learning. In traditional Seq2Seq models, we discard all the intermediate states of the encoder and use only its final states to initialize the decoder. This technique works good for smaller sequences, however as the length of the sequence increases, a single vector becomes a bottleneck and it gets very difficult to summarize long sequences into a single vector. The central idea behind Attention is not to throw away those intermediate encoder states but to utilize all the states in order to construct the context vectors required by the decoder to generate the output sequence.
- **Self-attention** - is similar to attention, they fundamentally share the same concept and many common mathematical operations. The self-attention mechanism allows the inputs to interact with each other ("self") and find out who they should pay more attention to ("attention"). The outputs are aggregates of these interactions and attention scores. The illustrations are divided into the following steps:

    1. Prepare inputs
    2. Initialise weights
    3. Derive key, query and value
    4. Calculate attention scores for Input 1
    5. Calculate softmax

6. Multiply scores with values
7. Sum weighted values to get Output 1
8. Repeat steps 4–7 for Input 2 & Input 3

Self-attention mechanisms can be extended to a Transformer architecture.