Luleå University of Technology

D7047E – Advanced Deep Learning

28 Mai 2023

Exercise 4 Theory | Group 1 | Berezin Ilya

Contents

RNN

Task 1

Consider a simple RNN with one hidden layer and the following assumptions:

1. Hidden Layer Size: 128
2. Output Layer Size: 10
3. Input Vector Size at each Time Step: 20

Question 1: Calculate the number of parameters required to unroll this vanilla RNN for:

1. 5-time steps
2. 10-time steps

For an RNN, the number of parameters does not depend on the number of time steps it's unrolled for. It's defined by the size of the hidden layer, the size of the input, and the size of the output.

For the considered RNN the parameters are:

- ✓ Parameters between the input and the hidden layer (Wxh):
  - Input Vector Size * Hidden Layer Size = 20 * 128 = 2560
- ✓ Parameters between the hidden layer and the output layer (Whh):
  - Hidden Layer Size * Hidden Layer Size = 128 * 128 = 16384
- ✓ Parameters between the hidden layer and itself at the next timestep (Why):
  - Hidden Layer Size * Output Layer Size = 128 * 10 = 1280
- ✓ Bias parameters for the hidden layer (bh): 128
- ✓ Bias parameters for the output layer (by): 10

The total parameters are the sum of the above = 2560 + 16384 + 1280 + 128 + 10 = 20362 for both steps.

Question 2: Do the number of parameters increase as you increase the number of times steps to unroll?

The number of parameters does not increase as we increase the number of time steps. The reason is that an RNN shares parameters across all time steps. This means that the same weight matrices and bias vectors are used at every time step, just with different inputs and hidden states. This is a key characteristic of RNNs and allows them to process inputs of varying lengths and to generalize across them. This property makes RNNs very powerful for sequential data like time series or natural language.

Task 2

Question 1: Ideally you would expect an RNN model to learn long term dependencies in sequences. But in reality, that does not happen. Explain the reasons.

RNNs, in theory, should be able to learn long-term dependencies due to their nature of carrying information in the hidden state from one step to the next. In practice, they struggle with this due to a vanishing gradient.

During backpropagation, the gradients of the loss function tend to get smaller and smaller as they are propagated back through time, since they are multiplied by the weights at each step. This results in the earlier layers of the network receiving very small (or "vanishing") gradients, meaning that they learn very slowly or not at all.

This makes it difficult for the RNN to learn and capture the relationships between items that are far apart in the sequence (long-term dependencies). The gradients provide the learning signal: if they vanish, the signal does not reach the earlier time steps.

Question 2: State the variants of an RNN and give a single line explanation for each of them.

There are several variants of RNNs designed to address the shortcomings of vanilla RNNs, particularly regarding learning long-term dependencies:

Long Short-Term Memory (LSTM): RNN that includes a memory cell and a set of gating units. The gating units control the flow of information in and out of the memory cell, thereby addressing the vanishing gradient problem and allowing the network to learn longer sequences.

Gated Recurrent Unit (GRU): Simpler variant of LSTM. It combines the forget and input gates into a single "update gate" and merges the cell state and hidden state. This makes it computationally more efficient, though potentially less capable of learning long sequences than LSTM.

Bidirectional RNNs (BiRNNs): RNNs that are run forward and backward through time, allowing them to capture past and future context.

Echo State Networks (ESN): RNN where the hidden-to-hidden weights are initialized randomly and then left untrained. This makes training much faster, though potentially less accurate.

Hopfield Networks: RNN with symmetric weights, where all neurons are connected to each other. They are used for pattern recognition and can remember and reconstruct patterns.

LSTM

Question 1 What was the motivation behind using gates in an LSTM?

The motivation behind using gates in an LSTM comes from the desire to control and manage the flow of information throughout the network over time. The gates in an LSTM are a way to optionally let information through, by outputting values between 0 and 1 to denote the percentage of information to let through. They help the model to forget irrelevant parts of the previous state, to update the cell state with new information, and to decide what part of the current cell state makes it to the output.

Question 2 What will happen if you manually set the output of the forget gate to 0.

The forget gate in LSTM is responsible for deciding what information should be discarded or "forgotten" from the cell state. If you manually set the output of the forget gate to 0, it will result in every incoming state completely forgetting the historical or past learned information from the cell state.

$$c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$$

where Ct is the new cell state, ft is the output of the forget gate, Ct-1 is the old cell state, it is the output of the input gate, and C~t is the candidate cell state. If ft is set to 0, then ft * Ct-1 will be 0 and the old cell state Ct-1 will not be carried over into the new cell state.

Question 3 How does an LSTM solve the vanishing gradient problem.

LSTM solves the vanishing gradient problem by having a mechanism called a cell state, which runs straight down the entire chain of the LSTM with minor linear interactions. This design helps to keep the gradient at a manageable size and prevents it from vanishing entirely during backprop, since there is always a path along which the gradient can flow without being multiplied by a potentially small number (as in traditional RNNs).

Question 4 How can you reduce the number of parameters of an LSTM cell?

1. Decrease the size of the hidden state. This may also decrease the model's capacity.
2. Use variants of the LSTM, such as the Gated Recurrent Unit (GRU), which has fewer gates and hence fewer parameters.
3. Apply techniques such as pruning or quantization, which can reduce the model's size without significantly decreasing performance.

Question 5 What is a Bidirectional LSTM and why is it useful?

A Bidirectional LSTM (BiLSTM) is a sequence processing model that consists of two LSTMs: one processing the data from start to end and another one processing the data from end to start. This makes it possible for each position in the sequence to have access to information from points both before and after it in the sequence.

The benefit of a BiLSTM is that it can capture both the past (via forward states) and the future (via backward states) context for each time step in the sequence. This is useful in many NLP tasks, where the meaning of a word depends on its surrounding words.