Theory - Exercice 4

1) Learning Sequences

1.1 Question 1

- For 5 time steps → creation of 5 copies of the network, each with the same set of weights.

Number of parameters in the hidden layer = $(20 + 128) \times 128 = 17\,408$

Number of parameters in the output layer = $128 \times 10 = 1280$

Total number of parameters in the RNN for 5 time steps = $(17\,408 + 1280) \times 5 = 93\,540$.

- For 10 time steps.

$(17\,408 + 1280) \times 10 = 187\,680$.

The total number of parameters required to unroll this vanilla RNN for 10 time steps is 187 680.

1.2 Question 2

Yes. In the example, the total number of parameters required to unroll the RNN increases from 93 540 to 187 680, which is more than double. This is because we're creating twice as many copies of the network with the same set of weights, resulting in the number of parameters.

In other words, we create more copies of the network with the same set of weights, which results in an increase in the number of parameters.

→ can lead to overfitting if we have limited training data.

1.3

Due to the problem of vanishing gradients

In back propagation through time, gradients are calculated by multiplying the gradient of the next time step with the weight matrix of the current time step.

If the weight matrix is repeatedly multiplied by values less than 1, the gradients will

eventually became very small, causing the gradient to vanish.
But, if the values are repeatedly multiplied by values greater than 1, the gradients will grow larger and larger, causing the gradients to explode. This problem becomes more pronounced with longer sequences. Others factors like architecture design (LSTM) the amount of training data and the quality of data can affect the RNNs long term memory capabilities.

1.4

a) Simple RNN (RNN cell): can generate predictions based on the current input and the past hidden state

b) LSTM: RNN variant with added memory cells, input gates and output gates

c) GRU: RNN variant with a simplified architecture

d) Bi-directional RNN: processes input sequences in two directions, forward and backward, using two separate hidden layers to learn context from past and future inputs.

2) LSTM

2.1 Task 1

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

The motivation behind using gates in a Long Short-Term Memory (LSTM) network is to allow the network to selectively remember or forget information over time and at each time step. It is to address the problem of vanishing that occur in standard RNNs during backpropagation through time.

LSTMs were developed as a solution to this problem by incorporating gates that control the flow of information through the network.

The LSTM architecture includes three types of gates: the input gate, forget gate, and output gate.

Specifically, they include an input gate that determines which information to store, a forget gate that decides which information to discard, and an output gate that controls how much information from the cell state is utilized to generate the output at each time step.

By enabling the LSTMs to selectively learn and forget past dependencies, the gates allow for better long-term memory and the learning of more complex dependencies in sequential data.

Using gates to regulate the flow of information, LSTMs can retain significant information over extended periods while disregarding irrelevant data. This feature renders LSTMs an excellent option for tasks such as natural language processing, speech recognition, and image captioning, which frequently involve long-term dependencies in sequential data.

2.1.2 Question 2 What will happen if you manually set the output of the forget gate to 0. Try to give a formal explanation using the equations from the question above.

If we manually set the output of the forget gate to 0, it implies that the LSTM is forced to forget all the visible information from the past hidden state and only learn from the current input.

In a standard LSTM, the forget gate is used to control the information flow from the previous hidden state, h_{t-1}, to the current hidden state, h_t. It consists of a sigmoid activation function that outputs values between 0 and 1, computed as follows:

$f_t = \sigma(W_f.x_t + U_f.h_{t-1}+ b_f)$

where W_f, U_f, and b_f are learnable parameters.

The output of the forget gate, f_t, is multiplied by the previous hidden state, h_{t-1}, to decide which information should be retained and which should be discarded. If f_t is close to 0, then most of the previous hidden state information is discarded, and the current state mainly depends on the current input, x_t.

So, if we manually set the output of the forget gate, f_t, to 0, then we are explicitly telling the LSTM to forget all the visible information from the previous hidden state, h_{t-1}, resulting in the current hidden state, h_t, becoming dependent solely on the current input, x_t. This hinders the capability of the LSTM to capture longer-term dependencies in the sequence, resulting in a poorer performance on tasks that require looking at the past hidden state information.

2.1.3 Question 3 How does an LSTM solve the vanishing gradient problem. Give a precise answer.

By using these gates, we talked about to regulate the flow of information, LSTMs can selectively retain important information over longer periods of time while disregarding irrelevant information. This ability to selectively remember or forget information makes LSTMs well-suited for processing sequential data with long-term dependencies, where traditional RNNs struggle due to the vanishing gradient problem.

Furthermore, the gates allow the LSTM to bypass the multiplication of a large number of small gradients during backpropagation, which can cause the gradient to vanish over time. Instead, the gates allow the network to propagate the gradient through the cell state without significant attenuation, which helps to mitigate the vanishing gradient problem.

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

There are a few ways to reduce the number of parameters of an LSTM cell:

1. Using a lower-dimensional embedding: By reducing the dimensionality of the input and output layers, we can reduce the number of parameters in the LSTM cell.

2. Sharing weights: We can share the weights of different gates in an LSTM cell to reduce the number of parameters. For example, we can use the same set of weights for the input and forget gates or for the forget and output gates.

3. Using a diagonal weight matrix: In standard LSTM cells, the weight matrix in the cell state update equation is typically a square matrix, resulting in many learnable parameters. By constraining the weight matrix to be diagonal, we can significantly reduce the number of parameters.

4. Using a simpler gating mechanism: Instead of using the full gating mechanism of an LSTM cell, we can use a simpler gating mechanism that has fewer learnable parameters. For example, we can replace the forget gate with a simple decay factor that applies a constant discount to the previous cell state.

Overall, the goal of reducing the number of parameters in an LSTM cell is to improve its computational efficiency and reduce overfitting. However, we need to balance this reduction with the cell's overall performance on the given task.

2.1.5 Question 5 What is a Bidirectional LSTM and why is it useful? Try to be as formal as possible while giving an answer to this question.

A Bidirectional LSTM (BLSTM) is a type of RNN that consists of two LSTMs: one that processes the input sequence in the forward direction, and another that processes the input sequence in the reverse direction. The output of each LSTM is combined at each time step to produce the final output.

By combining the output of both LSTMs, BLSTMs can consider the past and future context of each time step, making them useful in tasks where the output at each time step depends on the entire input sequence. BLSTMs can capture complex patterns and handle long-term dependencies more effectively than unidirectional LSTMs, making them suitable for processing sequential data.