Theoretical Tasks

November 20, 2017

1 Learning Sequences

Just as a fun task, speak out your phone number in your head. Pretty simple right! Now try to speak out the same phone number in reverse, it is not so easy now, is it. This is just one simple example to show that we might be learning things in a sequence. It is still a debatable hypothesis but is quite interesting to think about.

To learn and predict any kind of sequential information using Neural Networks, simple Feed Forward Networks are not enough, but, Recurrent Neural Networks(RNN) are specialized networks that we use for learning patterns in sequences.

Assuming that you have understood how to use Recurrent Neural Networks and some variants of it from the lecture, this task will help you understand them a little better.

1.1 Task 1

Consider a simple RNN with one hidden layer (Figure 1) and the following assumptions:

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

1.1.1 Question 1

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

1. 5 time steps
2. 10 time steps
3.

Input-to-hidden weights: Input vector size (20) * Hidden layer size (128) = 20 * 128 = 2560 parameters

Hidden-to-hidden weights: Hidden layer size (128) * Hidden layer size (128) = 128 * 128 = 16384 parameters

Hidden-to-output weights: Hidden layer size (128) * Output layer size (10) = 128 * 10 = 1280 parameters

Biases for hidden layer: 128 parameters

Biases for output layer: 10 parameters

Total parameters = 2560 + 16384 + 1280 + 128 + 10 = 20362 parameters

The number of parameters required to unroll this vanilla RNN is the same for both 5 and 10 time steps, as the parameters are shared across all time steps.

1.1.2 Question 2

Do the number of parameters increase as you increase the number of time steps to unroll? Explain your answer.

No, the number of parameters does not increase as you increase the number of time steps to unroll. This is because the weights and biases in an RNN are shared across all time steps, meaning the same parameters are used to process the input at each time step. The RNN structure allows it to handle sequences of variable length without increasing the number of parameters.

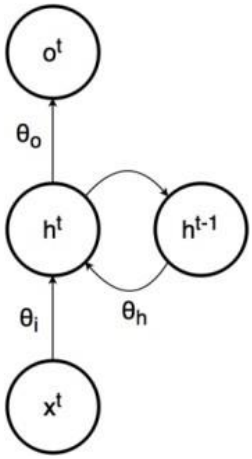

Figure 1: A vanilla RNN with a single hidden layer

1.2 Task 2

1.2.1 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 objectively. Try to be very precise here.

RNNs struggle to learn long-term dependencies in sequences mainly due to the vanishing and exploding gradient problems. During backpropagation, gradients can become extremely small (vanishing) or large (exploding), making it difficult for the network to learn and update its weights effectively. This issue is particularly pronounced when dealing with long sequences because the gradients are multiplied by the same weights many times, which can either exponentially increase or decrease their magnitudes. As a result, RNNs have a hard time capturing relationships between elements that are distant from each other in a sequence.

1.2.2 Question 2

State the variants of an RNN that you know of and give a single line explanation for each of them.

Long Short-Term Memory (LSTM): A type of RNN architecture that includes memory cells and gating mechanisms, enabling the network to learn long-term dependencies more effectively by controlling the flow of information through the cell states.

Gated Recurrent Unit (GRU): A simplified LSTM-like architecture that uses gating mechanisms to control the flow of information, but with fewer parameters compared to LSTM, making it computationally more efficient.

Bidirectional RNN (Bi-RNN): An RNN architecture that processes the input sequence in both forward and backward directions, allowing it to capture information from both past and future states in the sequence.

1.3 References

1. The Unreasonable Effectiveness of Recurrent Neural Networks: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

2 LSTM

This task will help you in understanding LSTMs better by understanding the internal structure of the LSTM cell.

Consider an LSTM cell

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 an LSTM is to control the flow of information within the cell, allowing it to learn and maintain long-term dependencies more effectively. Gates enable the LSTM cell to decide when to remember, forget, or update its internal state (cell state) and when to output information from that state. This selective control over the flow of information helps mitigate the vanishing gradient problem, which is a common issue in traditional RNNs.

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 you manually set the output of the forget gate to 0, it will force the LSTM cell to forget its previous cell state entirely. The forget gate output (f_t) is multiplied element-wise with the previous cell state (C_{t-1}) to update the current cell state (C_t):

C_t = f_t * C_{t-1} + i_t * g_t

When f_t is set to 0, the cell state update equation becomes:

C_t = 0 * C_{t-1} + i_t * g_t = i_t * g_t

This means the new cell state (C_t) will only depend on the input gate (i_t) and the candidate cell state (g_t), ignoring the information from the previous cell state (C_{t-1}).

2.1.3 Question 3

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

An LSTM solves the vanishing gradient problem through its cell state and gating mechanisms. The cell state acts as a highway that allows gradients to flow more easily across long sequences without significant decay. The gating mechanisms (input, forget, and output gates) control the flow of information, selectively updating or preserving the cell state. This controlled flow helps to mitigate the effect of vanishing gradients by ensuring that relevant information is retained, and irrelevant information is discarded, reducing the risk of exponentially decaying gradients during backpropagation.

2.1.4 Question 4

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

- **Decrease the size of the hidden layers or the number of hidden units.**
- **Employ techniques like weight sharing or parameter tying, where the same weights are used for different parts of the network.**
- **Use a simplified LSTM variant, such as the Gated Recurrent Unit (GRU), which has fewer parameters due to its simplified gating structure.**

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 is an LSTM architecture that processes the input sequence in both forward and backward directions, combining the hidden states from both directions at each time step. It essentially consists of two separate LSTMs, one for the forward pass and one for the backward pass. The output of the Bi-LSTM is the concatenation of the outputs from both LSTMs.

Bi-LSTM is useful because it allows the model to capture information from both past and future states in the sequence. This can be particularly beneficial when the context from both directions is important for making predictions, such as in sequence-to-sequence tasks or when the meaning of a word depends on the surrounding context.

2.2 References

1. Understanding LSTM Networks: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

3 Acknowledgement

We would like to thank Saurabh Varshneya for being an active member of MindGarage and for us helping in making this exercise.