Chapter 10

Questions for Sequence Modelling: Recurrent and Recursive Nets

10.1 Questions for Unfolding Computational Graphs

10.1.1

The authors state that *"Much as almost any function can be considered a feed forward neural network, essentially any function involving recurrence can be considered a recurrent neural network."* Do you agree with this statement?

10.1.2

When predicting the future from the past, an RNN learns to use its hidden state to summarize the history of observations. This is in general a necessarily lossy summary, since the history sequence can be any length and the hidden state is finite. Give an example when this is irrelevant for the prediction task. Give an example when this make a big Difference.

10.1.3

The unfolded view of a neural network illustrate how information flows forward in time. Is this necessarily the same time direction as in the data (i.e. from an observed time-series)? If yes, explain why. If no, give a counter-example.

10.1.4

If we were to imagine neural networks as implementing algorithms using a Control-Flow Graph, how would you characterize the difference between the GFG implemented by a Feed forward Neural Networks compared to a Recurrent Neural Networks?

10.2 Questions for Recurrent Neural Networks

The authors assume hyperbolic tangent units, not ReLU which in earlier chapters has been the most common one. Why do you think that is?

10.2.1

What is the fundamental difference between the two deep RNNs described by the equations 10.1 and 10.2 below?

$$\begin{aligned}
\boldsymbol{h}_1^{(t)} &= \tanh(\boldsymbol{b}_1 + W_1\boldsymbol{h}_1^{(t-1)} + U_1\boldsymbol{x}^{(t)}) \\
\boldsymbol{h}_2^{(t)} &= \tanh(\boldsymbol{b}_2 + W_2\boldsymbol{h}_2^{(t-1)} + U_2\boldsymbol{h}_1^{(t)}) \\
&\vdots \\
\boldsymbol{h}_l^{(t)} &= \tanh(\boldsymbol{b}_l + W_l\boldsymbol{h}_l^{(t-1)} + U_l\boldsymbol{h}_{l-1}^{(t)}) \\
\boldsymbol{o}^{(t)} &= c + V\boldsymbol{h}_l^{(t)}
\end{aligned} \tag{10.1}$$

$$\begin{aligned}
\boldsymbol{h}_1^{(t)} &= \tanh(\boldsymbol{b}_1 + W_1\boldsymbol{x}^{(t-1)} + U_1\boldsymbol{x}^{(t)}) \\
\boldsymbol{h}_2^{(t)} &= \tanh(\boldsymbol{b}_2 + W_2\boldsymbol{h}_1^{(t-1)} + U_2\boldsymbol{h}_1^{(t)}) \\
&\vdots \\
\boldsymbol{h}_l^{(t)} &= \tanh(\boldsymbol{b}_l + W_l\boldsymbol{h}_{l-1}^{(t-1)} + U_l\boldsymbol{h}_{l-1}^{(t)}) \\
\boldsymbol{o}^{(t)} &= c + V\boldsymbol{h}_l^{(t)}
\end{aligned} \tag{10.2}$$

10.2.2

Below (10.3) is a reproduction of equation 10.14 from the book. Under what independence assumptions is it modelling the joint probability $P(y^{(1)}, y^{(2)}, \ldots, y^{(1)})$?

$$-\sum_t \log p_{\text{model}}(y^{(t)}|\{\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(t)}\} \tag{10.3}$$

10.2.3

A common setup for training RNNs on sequence data (text and time-series) is to use next-step prediction (see equation 10.4 below). In what way (if any) is this different from the network described by figure 10.4 of the book, and is it an application of teacher forcing?

$$\begin{aligned}
\boldsymbol{h}^{(t)} &= \tanh(\boldsymbol{b} + W\boldsymbol{h}^{(t-1)} + U\boldsymbol{x}^{(t)}) \\
\boldsymbol{o}^{(t)} &= \boldsymbol{c} + V\boldsymbol{h}^{(t)} \\
\boldsymbol{y}^{(t)} &= \text{softmax}(\boldsymbol{o}^{(t)}) \\
L^{(t)} &= -\sum_{c=1}^{C} \mathbf{1}_{\boldsymbol{x}^{(t+1)}} \log p(y_c^{(t)})
\end{aligned} \tag{10.4}$$

(Some notes on the loss, in this case we assume that $\boldsymbol{x}$ is a vector encoding of discrete values from the set C, if the next $\boldsymbol{x}$ is c, the indicator function $\mathbf{1}_{\boldsymbol{x}^{(t+1)}}$ takes the value 1, otherwise 0. This means that the loss will only be for the value of $\boldsymbol{y}^{(t)}$ which corresponds to the probability of the next value of $\boldsymbol{x}$)

10.2.4

The Back-Propgation Through Time algorithm described can train RNNs for arbitrarily long sequences, but in practice it is fundamentally limited by computational resources. What computational resource is the limiting factor, and why is it so?

10.2.5

Examine equation 10.21 of the book. Start at $t = 1$ and expand the expression forwards in time for a few steps (lets say 3). Can you make any interesting observations about the power of the different terms and factors of the expanded expression?

10.2.6

A common way of modeling a joint distribution of a sequence is given in equation 10.31. In this case, we use the product rule of probability to factor the conditionals in a forwards-prediction. Imagine instead that we would like model the conditional distribution for arbitrary elments of a sequence, see equation below. How could you model this with one or more RNNs?

$$P(\mathbb{Y}) = P(\boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\tau)}) =$$
$$P(\boldsymbol{y}^{(t)}|\boldsymbol{y}^{(\tau)}, \ldots, \boldsymbol{y}^{(t+1)}, \boldsymbol{y}^{(t-1)}, \ldots, \boldsymbol{y}^{(1)}) \prod_{i=1}^{t-1} P(\boldsymbol{y}^{(i)}|\boldsymbol{y}^{(i-1)}, \ldots, \boldsymbol{y}^{(1)}) \prod_{i=t+1}^{\tau} P(\boldsymbol{y}^{(i)}|\boldsymbol{y}^{(i-1)}, \ldots, \boldsymbol{y}^{(t+1)})$$

10.2.7

To draw samples from an RNN, the book suggests three different ways to control how many time-steps the sampling is done over. Explain how the Bernoulli and length-prediction methods are actually very similar.

10.2.8

The book claims that the length-prediction control method needs to also have the predicted length as input. Explain how this is similar to teacher forcing.

10.2.9

The book claims that the length-prediction control method needs to also have the predicted length as input. Assuming the RNN which uses this control method has hidden-to-hidden recurrence, is it in principle necessary to have the predicted length as an extra input?

10.2.10

The book gives two main strategies for conditioning an RNN on some single vector input $\boldsymbol{x}$ (itemized list on page 391): As an extra input at each time step or as the initial state $\boldsymbol{h}^{(0)}$. For each of these strategies, explain their drawbacks.

10.2.11

Figure 10.9 in the book shows an example of an RNN which takes a fixed length vector as input at each time sequence. Can you come up with some other example than image captioning where this would be useful?

10.2.12

Explain why the RNN in equation 10.8 correspond to the conditional independence assumption in equation 10.35.

10.3 Bidirectional RNNs

10.3.1

A bidirectional RNN process a sequence in two intuitive orders, forwards and backwards. Could you use an RNN to process a sequence in some other order? When would you want to do that?

10.3.2

In what sense are RNNs applied to images more expensive than CNNs?

10.4 Encoder-Decoder Sequence-to-Sequence Architectures

10.4.1

We could train a sequence-to-sequence model with an auto-encoder objective, where the goal is to reproduce the input (essentially copy the context sequence through the encoder bottleneck). If the sequences are english sentences, what do you think the intermediate representation (the context C) would learn to capture?

10.4.2

It has been noted (e.g. Sutskever et al. 2014) that the performance of sequence-to-sequence architectures improves if we reverse the target sequence from the true order (so for translation from english to french, the target sentence in french has its words in reverse order). Why do you think this is the case?

10.5 Deep Recurrent Networks

10.5.1

Consider the deep RNN depicted in figure 10.13a. Would it still be a *deep* recurrent network if there where also recurrent connections going from z to h?

10.5.2

Consider figure 10.13b. What would happen if you changed where the delay tap was, so instead of being after the extra hidden recurrent layer, it was before it?

10.6 Recursive Neural Networks

10.6.1

What is the difference between a recursive neural network and a convolutional neural network?

10.7 The Challenge of Long-Term Dependencies

10.7.1

In the case of a linear recurrent network (without nonlinear activation functions), the authors explain how by looking at the largest eigenvalue of the recurrent matrix we can understand how a signal explodes or vanishes after repeated multiplications with the matrix. For a linear recurrent network this behaviour is the same in the forwards and backwards (computing gradients) direction. For a network with a nonlinear activation function (e.g. tanh), the forwards and backwards path behaves very differently, explain how and why.

10.8 Echo State Networks

10.9 Leaky Units and Other Strategies for Multiple Time Scales

10.9.1

Consider the leaky unit $\boldsymbol{u}^{(t)} \leftarrow \alpha \boldsymbol{u}^{(t-1)} + (1-\alpha)\boldsymbol{v}^{(t)}$. Briefly outline what the derivative of some loss at time t with respect to $\boldsymbol{u}^{(t-3)}$ looks like (tip: unfold the recurrence). What is the major difference from a typical RNN?

10.9.2

Explain how leaky units are related to residual connections of e.g. ResNets.

10.9.3

The leaky units described by $\boldsymbol{u}^{(t)} \leftarrow \alpha\boldsymbol{u}^{(t-1)} + (1-\alpha)\boldsymbol{v}^{(t)}$ essentially keeps a running average of $\boldsymbol{v}^{(y)}$. What would happen if we instead introduced a new free parameter β and had the equation $\boldsymbol{u}^{(t)} \leftarrow \alpha\boldsymbol{u}^{(t-1)} + \beta\boldsymbol{v}^{(t)}$? Would this leaky unit still work?

10.9.4

Instead of using leaky units, we could actually learn a weighted average over a k-length window of data: $\boldsymbol{u}^{(t)} = \sum_{i=0}^{k-1} \alpha_i \boldsymbol{v}^{(t-i)}$, where $0 \leq \alpha_i \leq 1$ and all sum to 1. The α's could be learnable parameters or calculated by a nerual network layer based on the input. What is the advantage and disadvantage of this setup compared to leaky unit RNNs?

10.9.5

Skip connections work by *adding* edges to a network, the units can still ignore them (setting the corresponding weights very low). Explain why this could happen.

10.10 The Long Short-Term Memory and Other Gated RNNs

10.10.1

In the section on leaky units, it was noted that by setting α close to 1 in the equation $\boldsymbol{u}^{(t)} \leftarrow \alpha\boldsymbol{u}^{(t-1)} + (1-\alpha)\boldsymbol{v}^{(t)}$, the network can learn over longer time spans. How could we initialize an LSTM to bias it towards this behaviour at the start of learning? In other words, how could you set the initial parameters of the forget gate (see equation 10.40) to make it close to 1?

10.10.2

Consider the *"reset"* gate of a GRU (equation 10.47, used in 10.45). Which gate of the LSTM does this gate correspond to (it might help to unroll the GRU and LSTM two time steps)?

10.10.3

One fundamental difference between an LSTM and a GRU is the ability of LSTMs to implement counters (see the example below) with a high range when the state is of finite precision. What is the reason for this difference (compare equations 10.41 and 10.45)?

Example of how an LSTM can use counters to solve simple problems

Imagine we have a context sensitive language $a^n b^n c^n$, where every sequence with n number of a's is followed by n number of b's and n number of c's (e.g *aabbcc*, *abc*, *aaaaabbbbbccccc*). An LSTM can solve the prediction problem over this language by learning to linearly increase two units for each *a* observed in the input, serving as counters of the number of *a*'s in the string and then decrease one of them linearly for each *b* it outputs until it is below a threshold, then linearly decreases the second counter for each *c* it outputs. See (Gers 2001) for further details.

10.11 Optimization for Long-Term Dependencies

10.11.1

Explain why element-wise clipping of gradient is not aligned with the true gradient or the minibatch.

10.12 Explicit Memory

10.12.1

Why is it difficult to optimize algorithms which produce exact integer addresses?

10.12.2

In which sense would you say LSTMs differ from the explicit memory architectures discussed?

10.12.3

Two different memory addressing mechanisms are discussed: **content-based** and **location-based**. One of these methods requires the number of memory locations to be fixed, the other doesn't. Which is which, and why is this the case?

10.12.4

If you where to characterize the LSTM as an explicit memory network, what would you say the adressing mechanism is?

Comments on vector-memory cells

In the original LSTM paper by Hochreiter and Schmidhuber from 1997 explicitly point out that each pair of input/output gates (they only had two) can control a set of cells (Section 4: Memory cell blocks). This is exactly the same as *"memory cells are typically augmented to contain a vector"*. Hochreiter and Schmidhuber point directly to the same motivation as in the chapter in the book: computational efficiency. In the paper they at most have two cells per memory block, but points out many times that this is not a limitation of the model. Later works didn't take this up until the recent years where this LSTM version has been rediscovered. It goes to show how many cite a work without reading it in detail...