Recurrent Neural Networks

Kristian Wichmann

December 25, 2017

This follows Goodfellow and Bengio's book, borrowing/adapting a lot of the illustrations from this.

1 Simple cyclical graph

Image a situation, where a state $s = s^{(t)}$ depends on "time" t. Here t is discrete, a whole number, usually, running from 1 to τ. Further assume, that each time step is connected to the next through the same function:

$$s^{(t)} = f(s^{(t-1)}, \theta) \tag{1.1}$$

I.e., there is only an explicit dependence on the state from the last time step. Here θ is a shorthand for any parameters in the model. The unfolded graph of this model is shown in figure 1.

When there is a first time step, $t = 1$, we see that every other state is determined by $s^{(1)}$:

$$s^{(t)} = f(s^{(t-1)}) = f(f(s^{(t-2)})) = \cdots = f(f(\cdots f(s^{(1)}) \cdots)) \equiv g(s^{(1)}) \tag{1.2}$$

1.1 Cyclical graph with input

To elaborate on this model, assume that at each time step, there is also some kind of input $x^{(t)}$. So now the state - now called $h^{(t)}$ for hidden state - depends on this as well:

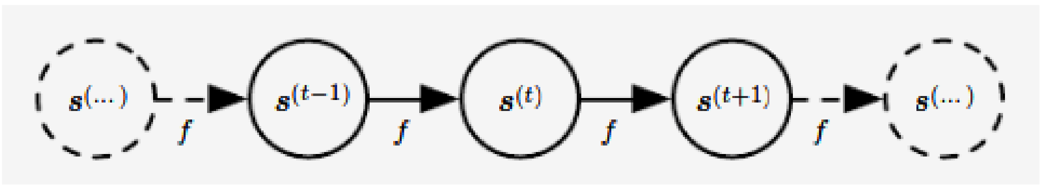

Figure 1: Simple cyclical graph.

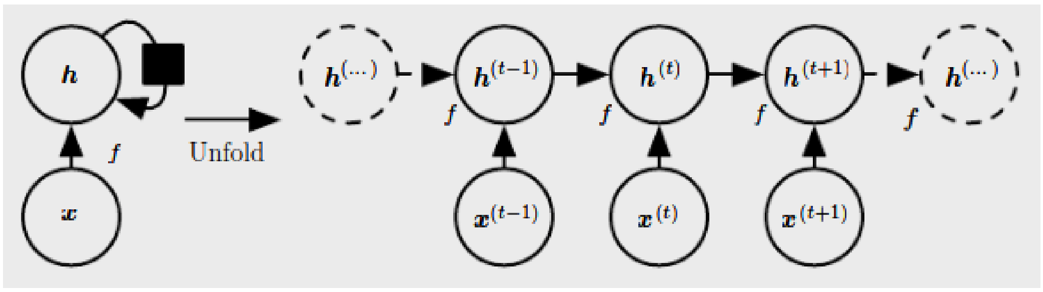

Figure 2: Cyclical graph with input.

$$s^{(t)} = f(s^{(t-1)}, x^{(t)}, \theta) \tag{1.3}$$

Figure 2 shows both the folded and unfolded versions of this model.

When there is a first time step, $t = 1$, we can express a general hidden state similarly to equation 1.2:

$$\begin{aligned} h^{(t)} =& f(h^{(t-1)}, x^{(t)}) = f(f(h^{(t-2)}, x^{(t-1)}), x^{(t)}) = \cdots && (1.4)\\ & f(f(\cdots f(h^{(1)}, x^{(1)}) \cdots, x^{(t)}) \cdots)) \equiv && (1.5)\\ & g(h^{(1)}, x^{(1)}, \cdots, x^{(t)}) && (1.6) \end{aligned}$$

So here, any the hidden state at a time t depends on the hidden state at $t = 1$ and all the input up to the point t.

1.2 Cyclical graph with both input and output

Finally, we would also like to model to provide some kind of output. We will assume there's output at every step, but in some applications, such as classification, we usually only care about the final output.

Such a model is shown in figure 3. Here, the output at time t depends only on the hidden state:

$$o^{(t)} = o(h^{(t)}) \tag{1.7}$$

2 Recurrent neural networks

As in feed forward neural networks, we will consider cases where the functions involved are linear followed by some kind of non-linearity.

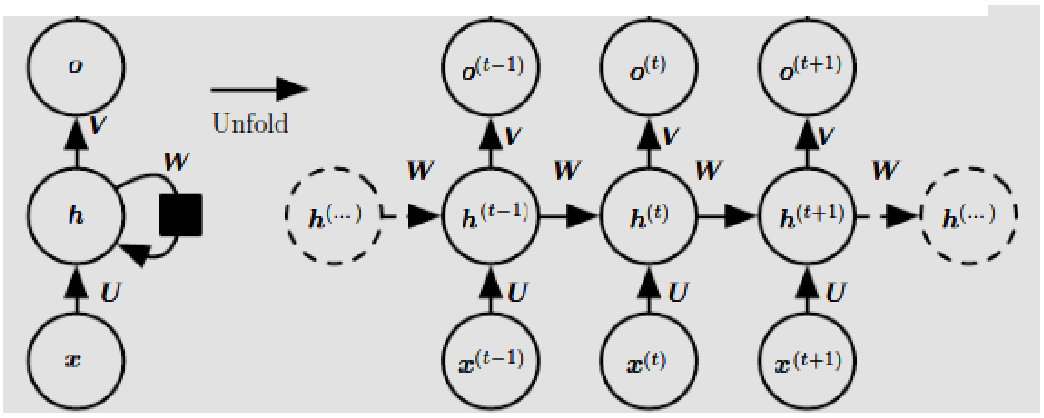

Figure 3: Cyclical graph with input as well as output.

2.1 Loss function

As usual with neural networks, we need some kind of loss function to minimize. In general, this will be a function of the outputs:

$$L = L(o^{(1)}, \dots, o^{(\tau)}, \mathbf{y}) \tag{2.1}$$

Here $\mathbf{y}$ is a label or collection of labels. Sometimes, it will be a function of all the outputs, sometimes only the last. In any case, we will need to know the derivatives of the outputs with respect to the parameters in the model.

2.2 Tanh activation

Now, consider a case where the activation function for the hidden layer is tanh. This is customary for RNNs. So, the model is:

$$a^{(t)} = Wh^{(t-1)} + Ux^{(t)} + b \tag{2.2}$$

$$h^{(t)} = \tanh(a^{(t)}) \tag{2.3}$$

$$o^{(t)} = Vh^{(t)} + c \tag{2.4}$$

In general, a, h, and o's are vectors, and U, V and W are matrices of appropriate sizes.

2.3 One-dimensional input, hidden, and output

Consider now, as a warm-up, the case where everything is numbers instead of vectors/matrices.

For this model, we will apply softmax to the output and use a cross-entropy loss function between the result and a sequence of labels:

$$\hat{y}^{(t)} = \text{softmax}(o^{(t)}) \tag{2.5}$$

$$L^{(t)} = \sum_{i=1}^{\tau} \tag{2.6}$$