Lectures on Machine Learning (Fall 2017)

Hyeong In Choi Seoul National University

Lecture 15: Recurrent Neural Networks (RNNs) (Draft: version 0.9.1)

Topics to be covered:

- Basics of RNNs
- Vanishing and exploding gradient problem
- Long short-term memory (LSTM)
- Gated recurrent unit (GRU)

1 Basics of RNNs

The standard feedforward neural networks we have so far dealt with, be they DNNs or CNNs, after everything is said and done, are function generators associating appropriate output to input. However, certain types of data are serial in nature. For instance, look at the following translation from French to English.

Il m'a donné un cadeau, parce que $\cdots$
He gave me a present, because $\cdots$

In here, some word pairs are in direct correspondence such as (il, he), (un, a), (cadeau, present), but some are more subtle: (parce que, becauase). The expression "m'a donné" is a shorthand for "me a donné", in which the corresponding pairs are: (me, me) and (a donné, gave). However, the word order has to be transposed to produce correct translation "gave me". One can see even in this simple example that many subtle transformations must be performed to produce the correct translation. It shows that the simple input-output relation of a usual feedforward network is not adequate for this kind of sequential problems. In order to deal with such problems, recurrent neural networks are devised. Of course, nowadays, machine translation has advanced well beyond this application of RNN, but the gist is still there.

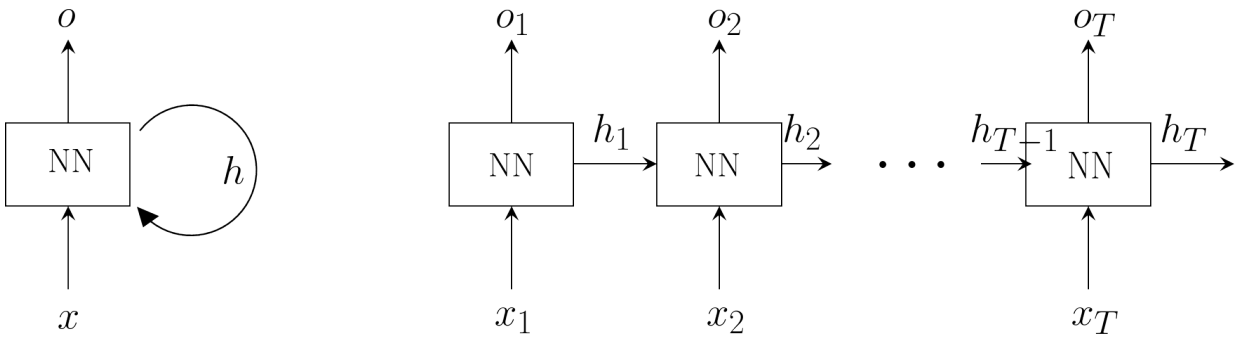

Figure 1: General form of RNN

The basic scheme of RNN is in Figure 1. The left diagram says that the input-output relation of a standard neural network is altered so that the output is fed into the input. The figure on the right illustrates the scheme unwrapped through time. In here, input is the serial data $(x_1, \cdots, x_T)$ and the output is $(o_1, \cdots, o_T)$. The output of a constituent (vertical) neural network is fed into the next constituent neural network in the next stage as part of the input. So the output o_t depends on all the inputs $(x_1, \cdots, x_t)$.

It may be the case that it is not a good idea to make the output o_1 dependent only on x_1. In fact o_1 itself should be produced in context. Figure 2 shows one such example. Some part of the input (here, x_1 and x_2, but its length can be arbitrarily long) does not produce any output and the output starts to show up after some delay (here, it is two, but it can arbitrarily long). So the part with no output acts like the encoder and the part that produces output can be regarded as the decoder.

There are many variations in the way RNN is configured. Shown in Figure

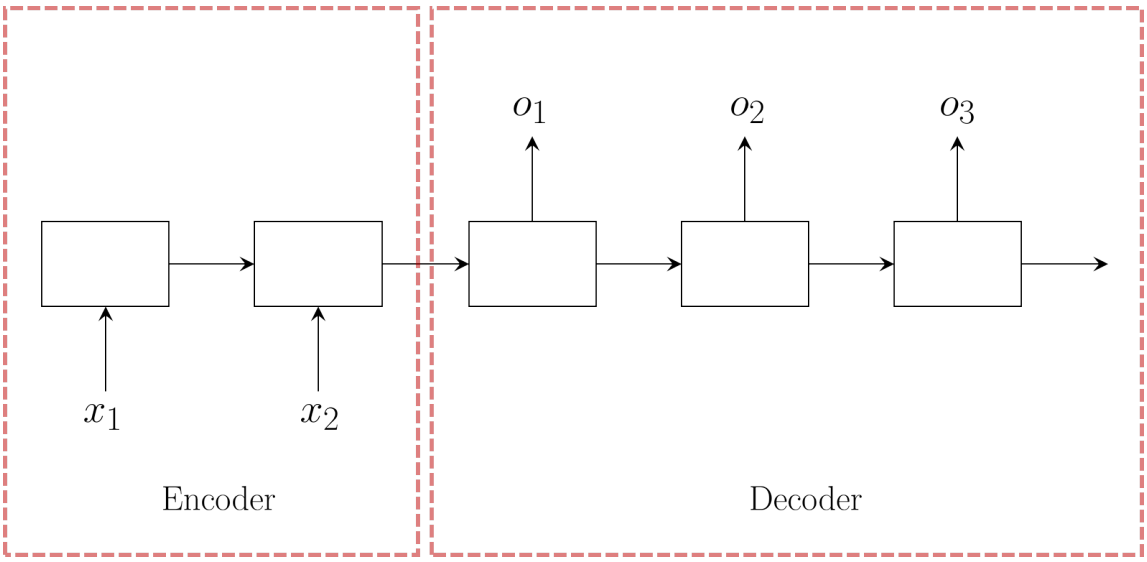

Figure 2: Delayed Sequence to sequence

3 is sequence on sequence, and in Figure 4, vector on sequence. Similarly, Figure 5 shows a sequence on vector configuration. These configurations can be altered to accommodate delays and so on.

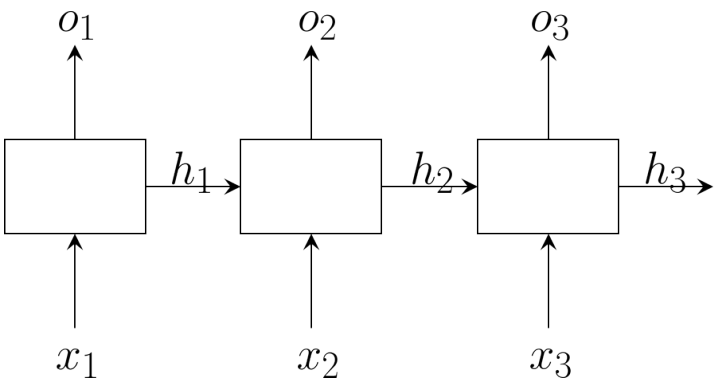

Figure 3: Sequence to sequence

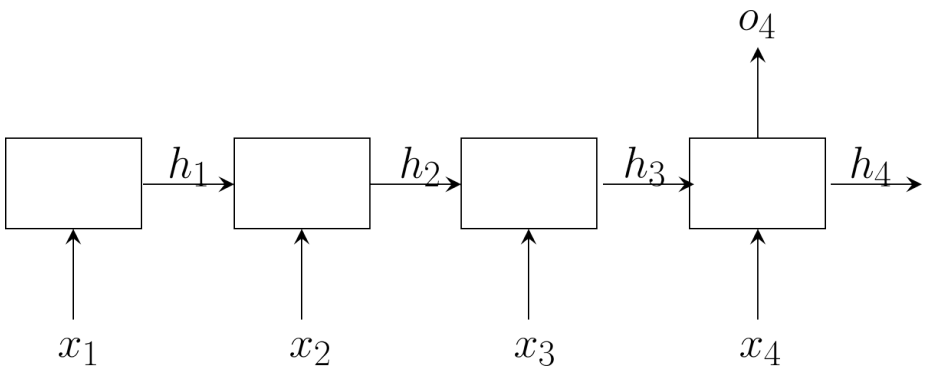

Figure 4: Sequence to vector

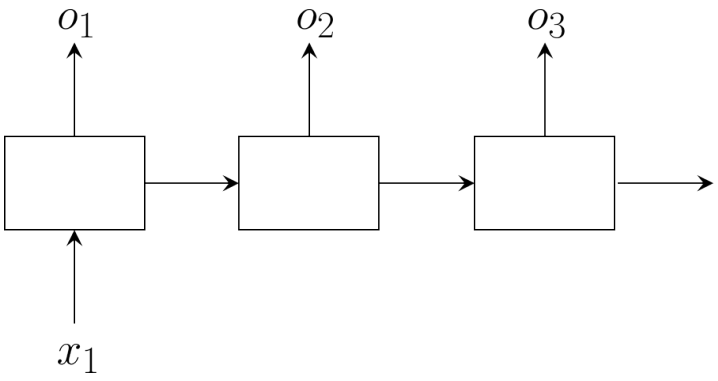

Figure 5: Vector to sequence

2 Vanishing and exploding gradient problem

For a long time, it was observed that RNNs are very hard to train. Let us see why it is the case. Recall the formula (10) from Lecture 13. It says that the term $\left(\frac{\partial E}{\partial h^\ell}\right)$ used in the backpropagation algorithm is a product of a long chain of matrices:

$$\left(\frac{\partial E}{\partial h^\ell}\right) = \left(\frac{\partial z^{\ell+1}}{\partial h^\ell}\right)\left(\frac{\partial h^{\ell+1}}{\partial z^{\ell+1}}\right)\cdots\left(\frac{\partial h^L}{\partial z^L}\right)\left(\frac{\partial E}{\partial h^L}\right),$$

For the input $x = h^0$, this chain is the longest. If the activation function $\varphi_\ell(t)$ is the sigmoid function

$$\sigma(t) = \frac{1}{1+e^{-t}},$$

its derivative is

$$\sigma'(t) = \frac{e^t}{(1+e^t)^2},$$

which gets very small so that it practically vanishes except at a small interval near 0. For example, $\sigma'(10) = 4.5 \times 10^{-5}$ and even for $t = 5$, $\sigma'(5) = 0.0066$. So if an input value z_j^ℓ that gets fed into the sigmoid activation function is somewhat large, say like 10, it basically kills the gradient. It is one of the reasons why people prefer the ReLU activation function. But it is not a panacea, as negative input values also kill the gradient and make it stay there. Even if one avoids such an outright vanishing gradient problem, the long matrix multiplication in general may make the gradient *vanish* or *explode*.

This kind of problem gets even more aggravated in the case of RNNs, because RNNs by default require long chain of backpropagation not only through the layers of neural networks of constituent cells but also across the different cells; hence the name back propagation through time (BPTT). For this reason, RNNs are notorious for being difficult to train.

3 Long short-term memory (LSTM)

Let us now introduce the model, called the Long Short-Term Memory (LSTM), proposed by Hochreiter and Schmidhuber [2]. It appeared in 1997, long before the so-called Deep Learing was invented and became popular, but it did not attract much attention until people realized it indeed provides a good solution to the vanishing and exploding gradient problem of RNN as described above.

In this section, we will only describe the architecture of its cell. Of course, there are many variations in the cell architecture, but what we present here is the most basic. Once we get a hold of the cell, it can be applied in many different configurations as described in Section 1.

Figure 6 shows the structure of LSTM. The input vector is x_t. The cell state denoted by c_{t-1} and the hidden state h_{t-1} are fed into the LSTM cell and c_t and h_t are fed into the next cell. Internally, it has four states: i_t

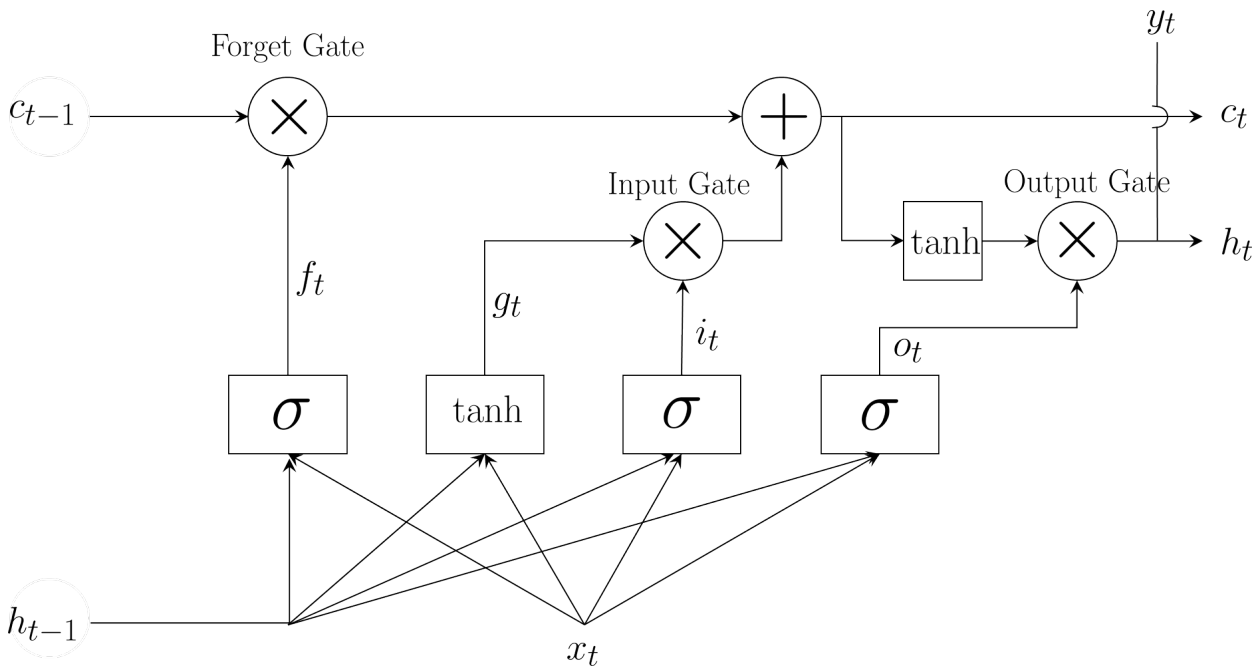

Figure 6: Architecture of a LSTM cell

(input), f_t (forget), o_t (output), and g_t. The forget state f_t is obtained as a sigmoid output of a network with x_t and h_{t-1} fed into it as inputs. (Note that f_t can be a vector, in which case each element of f_t is a sigmoid output.) Its neural network formula is (2), and (6) shows how it works. (In here and elsewhere, $\otimes$ means element-wise multiplication.) Note that since f_t is a sigmoid output, each of its elements has a value between 0 and 1. If it is close to 0, it pretty much erases c_{t-1} by multiplication; if it is close to 1, it basically keeps c_{t-1} by multiplication. Because of this property, f_t is given the name "forget state". The input state i_t and the output state o_t are obtained similarly as described in (1) and (3), respectively. The state g_t is also obtained similarly except that it is an output of the form tanh . This gives the $\pm$ sign. The product $i_t \otimes g_t$ is then added to c_t, and this gives new information to the cell state. The hidden state h_t is gotten as in (6), and the cell output y_t is the same as h_t. The whole scheme is depicted in Figure 6.

$$
\begin{aligned}
\text{(input)} \quad & i_t = \sigma\left(W_{ix}x_t + W_{ih}h_{t-1} + b_i\right) && (1)\\
\text{(forget)} \quad & f_t = \sigma\left(W_{fx}x_t + W_{fh}h_{t-1} + b_f\right) && (2)\\
\text{(output)} \quad & o_t = \sigma\left(W_{ox}x_t + W_{oh}h_{t-1} + b_o\right) && (3)\\
& g_t = \tanh\left(W_{gx}x_t + W_{gh}h_{t-1} + b_g\right) && (4)\\
\text{(cell state)} \quad & c_t = f_t \otimes c_{t-1} + i_t \otimes g_t && (5)\\
\text{(hidden state)} \quad & h_t = o_t \otimes \tanh(c_t) && (6)\\
\text{(cell output)} \quad & y_t = h_t. && (7)
\end{aligned}
$$

4 Gated recurrent unit (GRU)

Gated recurrent unit (GRU) is a simplified version of LSTM [1]. It works in pretty much the same way as LSTM does, exept that it is simpler.

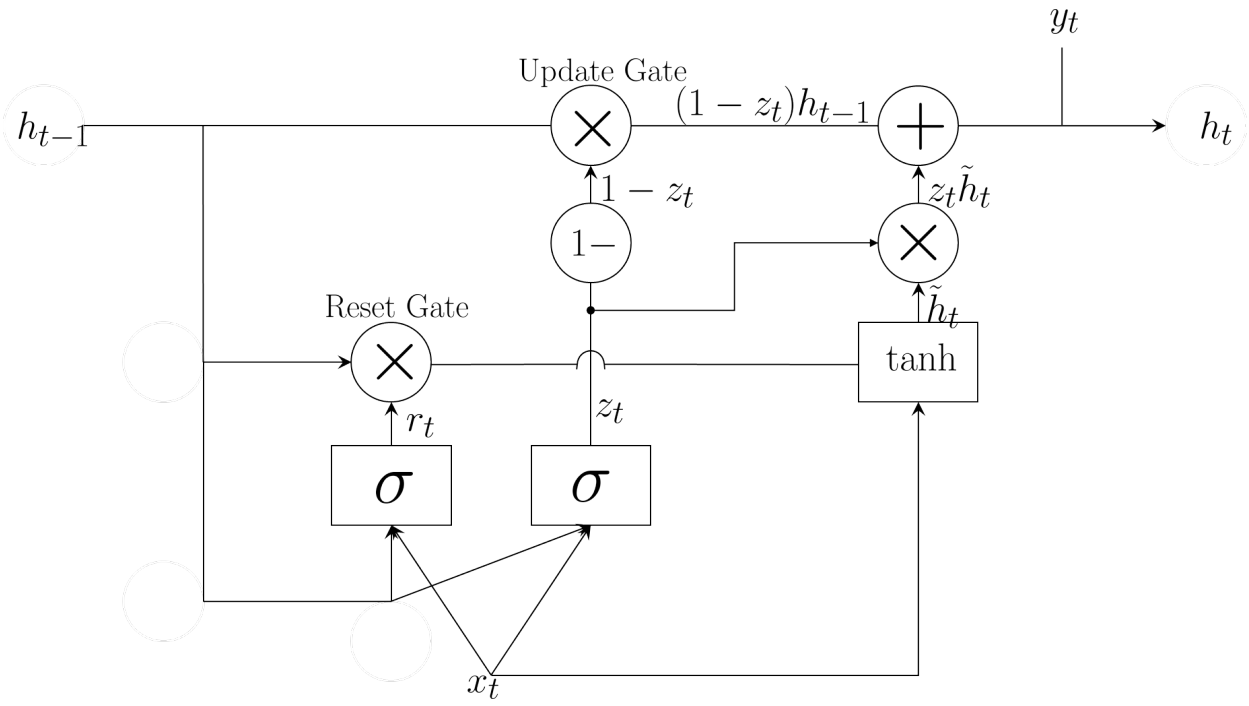

Figure 7: Architecture of a GRU cell

As usual, x_t is the input vector. Unlike LSTM, there is no hidden state

and only the cell state h_t. The update signal z_t is a sigmoid output of the neural network with h_{t-1} and x_t as inputs. If it is close to 1, it erases h_{t-1} by the multiplication $(1 - z_t)h_{t-1}$; it keeps h_{t-1} if z_t is close to 0. It is reflected as a part of (8). If z_t is close to 1, it adds $\tilde{h}_t$ gotten by (9) to h_t. The reset signal r_t is a sigmoid output of the neural network with h_{t-1} and x_t as inputs. The multiplication $r_t \otimes h_{t-1}$ pretty much keeps h_{t-1} if r_t is close to 1; and basically erase it if r_t is close to 0. This term is a part of the formula defining $\tilde{h}_t$. The cell output y_t is the same as h_t, and the whole scheme is depicted in Figure 7.

$$
\begin{aligned}
\text{(cell state)} \qquad h_t &= (1 - z_t)h_{t-1} + z_t\tilde{h}_t && (8)\\
\tilde{h}_t &= \tanh\left(W_h x_t + U_h(r_t \otimes h_{t-1}) + b_h\right) && (9)\\
\text{(update signal)} \qquad z_t &= \sigma\left(W_z x_t + U_z h_{t-1} + b_z\right) && (10)\\
\text{(reset signal)} \qquad r_t &= \sigma\left(W_r x_t + U_r h_{t-1} + b_r\right) && (11)\\
\text{(cell output)} \qquad y_t &= h_t. && (12)
\end{aligned}
$$

References

[1] Chung, J., Gulcehre, C., Cho, K., Bengio, Y., *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*, https://arxiv.org/abs/1412.3555 (2014)

[2] Hochreiter, S., Schmidhuber, J., Courville, A., *Long short-term memory*, Neural Computation 9(8):1735-80 (1997)