Deep Learning Study Lecture Note

Lecture 6 : What is RNN?

Kim Tae Young

January 21, 2020

6.1 What is RNN?

The RNN(Recurrent Neural Network) is a model which is largely used in time series analysis. Here series analysis means that our data is finite sequence, where the length may change for each data. For example, a sentence in natural language, value of stock with respect to time, signal of music are such series. So RNN's target is such problems, like machine translation, stock estimation, and voice voice generation etc.
Unlike CNN, basic layer of RNN is fully connected layer, however it differs in structure with MLP.

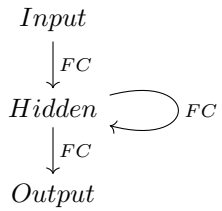
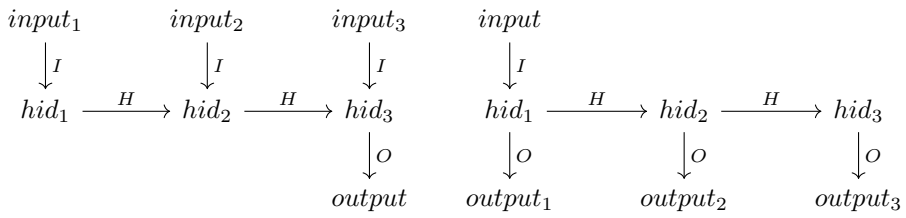


Figure 6.1: RNN

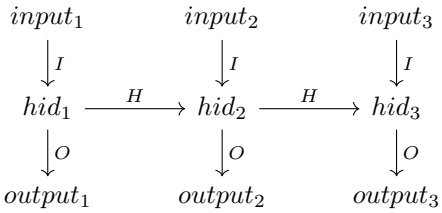
The loop connection is where the name recurrent originates. Since our input is time series, the loop connection means we apply that layer per each input. More specifically, we have 4 types of layers as following.
One to One is not that interesting, it is simply when both input and output is a single instance. Many to One is mapping sequence of input to single output. The main case is classifying text. One to Many generates sequence from single input. You can think of this as text generation. Finally, many to many is creating output sequence from input sequence. The sequences does not need to be of same length. Also, the output does not need to start from the beginning.
The neural network can be forwarded as the following. Here, I, H, O are matrices.

$$
\begin{aligned}
&Hid_{-1} = 0 \\
&Hid_t = \sigma(I(In_t) + H(Hid_{t-1})) \\
&Out_t = O(Hid_t)
\end{aligned}
$$

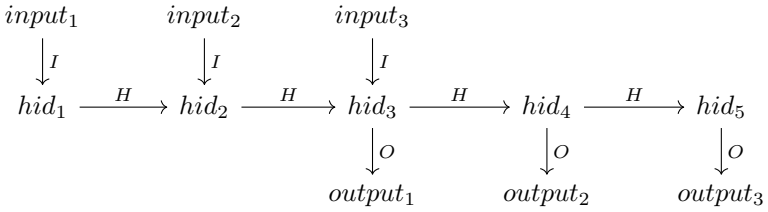


(a) Many to One

(b) One to Many



(c) Many to Many(1)



(d) Many to Many(2)

Figure 6.2: RNN expanded

Here, same letter means same matrix.
To extend RNN, we stack hidden layers as the following.

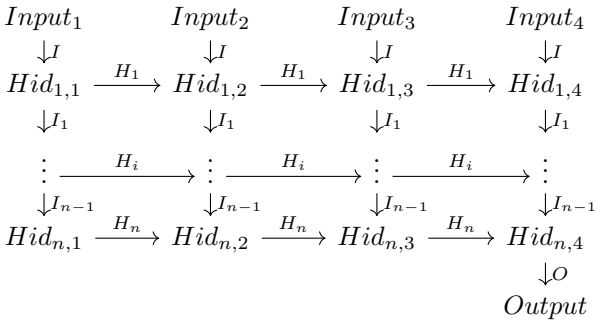
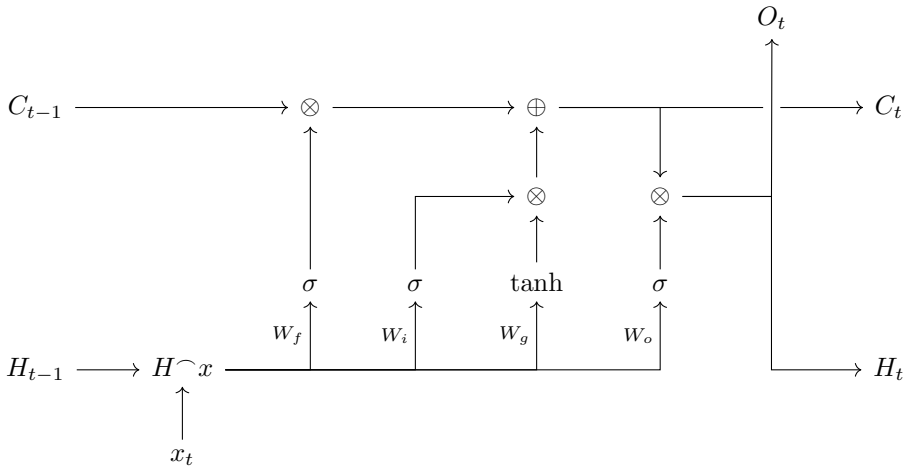


Figure 6.3: Extended RNN

Again, the main problem in RNN is gradient vanishing. As the distance from a input to an output increases, the effect of the input node to the output node decreases, thus there is little learning occurring in this connection. To overcome this long-term dependency, there are several methods.

6.2 LSTM and GRU

LSTM(Long Short Term Memory) Network is capable of learning long term dependencies. The structure of LSTM is similar to RNN, but with much more complex computation than just just simple fully connected layer. Main idea of LSTM is to create 'long term memory' so that the first element of input sequence can affect the last element of output sequence. You can think of it as human memory's long term memory. The basic structure of LSTM is LSTM cell, which works as one hidden state in RNN.

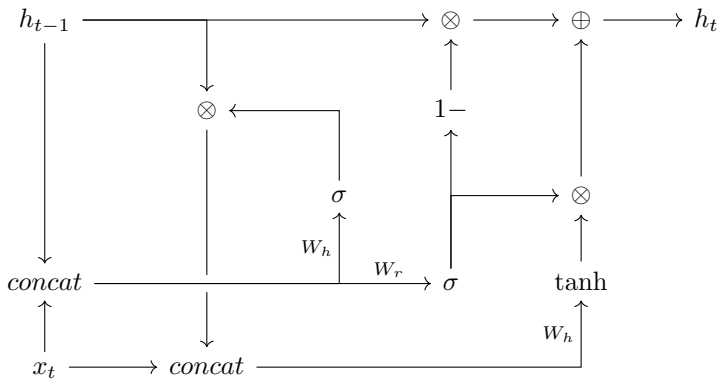


$$
\begin{aligned}
f_t &= \sigma(W_{xf}x_t + W_{hf}H_{t-1} + b_f) \\
i_t &= \sigma(W_{xi}x_t + W_{hi}H_{t-1} + b_i) \\
o_t &= \sigma(W_{xo}x_t + W_{ho}H_{t-1} + b_o) \\
g_t &= \tanh(W_{xg}x_t + W_{hg}H_{t-1} + b_g) \\
C_t &= f_t \otimes C_{t-1} + i_t \otimes g_t \\
H_t &= o_t \otimes \tanh(C_t)
\end{aligned}
$$

Figure 6.4: LSTM cell

Here, x_t is input sequence and O_t is output sequence. W is linear layer, σ and tanh are activation functions. $\oplus$ and $\otimes$ are element-wise addition and multiplication, $H \frown x$ is concatenation.

C_t is called cell state. Since C_t is connected without any layer, back propagation passes through the cell state, which results in that every relation of each pair in input sequence and output sequence are well learned.
The first linear layer W_f is called forget layer. After linear layer and activation function, this layer chooses which long term memory to forget, by multiplying it by value between 0 and 1.
The second linear layer and third layer determines what information to remember from current state. Reason for using tanh is so that we can remember negative values. After determining which value to remember, we add it to cell state. The value of added states becomes current cell state.
The final layer is output layer. By multiplying weight to the memory, we both store its value to output and current hidden state.
Since LSTM's structure is too complex, GRU(Gated Recurrent Unit) is a alternative model for solving gradient vanishing problem.



$$z_t = \sigma(W_{hz}h_{t-1} + W_{xz}x_t + b_z)$$
$$r_t = \sigma(W_{hr}h_{t-1} + W_{xr}x_t + b_r)$$
$$\tilde{h}_t = \tanh(W_{hh}(r_t \otimes h_{t-1}) + W_{xh}x_t + b_h)$$
$$h_t = (1 - z_t) \otimes h_{t-1} + z_t \otimes \tilde{h}_t$$

Figure 6.5: GRU

Unlike LSTM, GRU uses single weight to both manage forget and remember. First, we combine memory and current input, and use that information to generate two gate value, one(z_t) for storing and forget, one(r_t) to filter memory to process. Then with filtered memory, r_t, create information to store, $\tilde{h}_t$. Finally we sum previous memory and $\tilde{h}_t$ with weight z_t.

6.3 Attention mechanism

Attention mechanism is another method for solving gradient vanishing problem. The main idea is not generating direct result, rather first create weight to input sequence, then create output based on that weight. Here, each basic cells denotes either RNN cell, LSTM cell, or GRU cell. The main structure of attention machanism is separated to two parts, encoder and decoder.

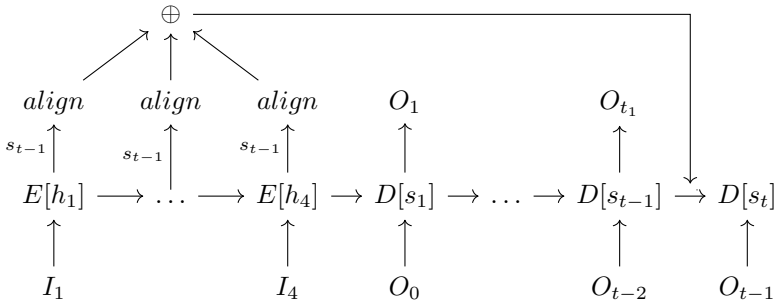


Figure 6.6: Attention

The encoder first performs exactly same as what we did before, get each input and generate hidden state vector h_t with basic cells. Now let matrix $H = (h_1, \ldots, h_n)$.
Decoder will create score value e_{ij}, denoting how similar are hidden state vector of decoder s_{i-1} and output of encoder's vector, h_j.

$$e_{ij} = a(s_{i-1}, h_j)$$

Here, a is alignment model, where you can use $H^T V s_{i-1}$, $v^T \tanh(WH + V s_{i-1})$ where v, V, W are trained parameter.
After this step, we transform e_{ij} into probability value, by applying softmax function.

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

Finally, context vector that decoder uses to predict output is defined as following, which is weighted sum of encoder's matrix by attention probability value.

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

Now concatenate context vector and output of decoder, and finally transforming concatenated vector to output sequence, we complete attention mechanism.

Bibliography

[1] Attention machanism, ratsgo's blog. https://ratsgo.github.io/from%20frequency%20to%20semantics/2017/10/06/attention/