Recurrent Neural Networks (RNNs): A gentle Introduction and Overview

Colin Acker

October 2023

1 Notes

Recurrent Neural Networks take into account previous inputs $\boldsymbol{X}_{0:t-1}$ instead of just $\boldsymbol{X}_t$. Let us denote the hidden state at time t as $\boldsymbol{H}_t \in \mathbb{R}^{nxh}$ and the input at time t as $\boldsymbol{X}_t \in \mathbb{R}^{nxd}$, where n is the number of samples, d is the number of inputs of each sample, and h is the number of hidden units. We use a weight matrix $\boldsymbol{W}_{xh} \in \mathbb{R}^{dxh}$, hidden-state-to-hidden-state matrix $\boldsymbol{W}_{hh} \in \mathbb{R}^{hxh}$, and bias parameters $\boldsymbol{b}_h \in \mathbb{R}^{1xh}$. Putting these together, we get

$$\boldsymbol{H}_t = \phi_h(\boldsymbol{X}_t \boldsymbol{W}_{xh} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hh} + \boldsymbol{b}_h)$$

$$\boldsymbol{O}_t = \phi_o(\boldsymbol{H}_t \boldsymbol{W}_{ho} + \boldsymbol{b}_o)$$

Where ϕ_h, ϕ_o are activation functions. Note the RNN is recursive as $\boldsymbol{H}_t$ contains $\boldsymbol{H}_{t-1}$, which contains $\boldsymbol{H}_{t-2}$, and so on. Backpropagation Through Time (BPTT) generalizes the backpropagation algorithm for RNNs. Let us define a loss function:

$$\mathcal{L}(\boldsymbol{O}, \boldsymbol{Y}) = \sum_{t=1}^{T} l_t(\boldsymbol{O}_t, \boldsymbol{Y}_t)$$

We can now calculate the derivative of the loss function, w.r.t. each weight matrix. After some math, we get:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{ho}} = \sum_{t=1}^{T} \frac{\partial l_t}{\partial \boldsymbol{O}_t} \frac{\partial \boldsymbol{O}_t}{\partial \phi_o} * \boldsymbol{H}_t$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{hh}} = \sum_{t=1}^{T} \frac{\partial l_t}{\partial \boldsymbol{O}_t} \frac{\partial \boldsymbol{O}_t}{\partial \phi_o} * \boldsymbol{W}_{ho} \sum_{k=1}^{t} \frac{\partial \boldsymbol{H}_t}{\partial \boldsymbol{H}_k} * \frac{\partial \boldsymbol{H}_k}{\partial \boldsymbol{W}_{hh}}$$

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{W}_{xh}} = \sum_{t=1}^{T} \frac{\partial l_t}{\partial \boldsymbol{O}_t} \frac{\partial \boldsymbol{O}_t}{\partial \phi_o} * \boldsymbol{W}_{ho} \sum_{k=1}^{t} \frac{\partial \boldsymbol{H}_t}{\partial \boldsymbol{H}_k} * \frac{\partial \boldsymbol{H}_k}{\partial \boldsymbol{W}_{xh}}$$

These expressions can be further written using these equalities:

$$\sum_{k=1}^{t} \frac{\partial \boldsymbol{H}_t}{\partial \boldsymbol{H}_k} * \frac{\partial \boldsymbol{H}_k}{\partial \boldsymbol{W}_{hh}} = \sum_{k=1}^{t} (\boldsymbol{W}_{hh}^T)^{t-k} * \boldsymbol{H}_k$$

$$\sum_{k=1}^{t} \frac{\partial \boldsymbol{H}_t}{\partial \boldsymbol{H}_k} * \frac{\partial \boldsymbol{H}_k}{\partial \boldsymbol{W}_{hh}} = \sum_{k=1}^{t} (\boldsymbol{W}_{hh}^T)^{t-k} * \boldsymbol{X}_k$$

It is apparent that the change in loss with respect to both $\boldsymbol{W}_{hh}$ and $\boldsymbol{W}_{xh}$ is dependent on the sum of all powers (up to $t-k$) of $\boldsymbol{W}_{hh}^T$. This is the most defining feature of RNN's.

Vanishing or exploding gradients is a big issue with RNN's. From above, we see that $\frac{\partial \boldsymbol{H}_t}{\partial \boldsymbol{H}_k}$ introduces matrix multiplication over a potentially very long sequence. Vanishing gradients occur when there are values (< 1) in the matrix multiplication, and exploding gradients occur when there are values (> 1) in the matrix multiplication. This motivates Long Short-Term Memory Units (LSTMs). LSTM's store information in structures called gated cells. We use an output gate $\boldsymbol{O}_t$ to read entries of the cell, an input gate $\boldsymbol{I}_t$ to read data into the cell, and a forget gate $\boldsymbol{F}_t$ to reset the content of the cell.

$$\boldsymbol{O}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xo} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{ho} + \boldsymbol{b}_o)$$
$$\boldsymbol{I}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xi} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hi} + \boldsymbol{b}_i)$$
$$\boldsymbol{F}_t = \sigma(\boldsymbol{X}_t \boldsymbol{W}_{xf} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hf} + \boldsymbol{b}_f)$$

The sigmoid function σ transforms the output, so that each vector has entries $\in (0,1)$. They introduce the candidate memory cell $\tilde{\boldsymbol{C}}_t$ now, which in essence captures the data from the input vector.

$$\tilde{\boldsymbol{C}}_t = \tanh(\boldsymbol{X}_t \boldsymbol{W}_{xc} + \boldsymbol{H}_{t-1} \boldsymbol{W}_{hc} + \boldsymbol{b}_c)$$

To finish things up, they construct the new memory content $\boldsymbol{C}_t$ using the formula:

$$\boldsymbol{C}_t = \boldsymbol{F}_t \odot \boldsymbol{C}_{t-1} + \boldsymbol{I}_t \odot \tilde{\boldsymbol{C}}_t$$

Where $\odot$ is element-wise multiplication. So, the input gate controls how much of the new information should be added. The candidate memory cell chooses which information should be added. The forget gate controls how much of the previous data should be added. Finally, we have the formula for the hidden states $\boldsymbol{H}_t \in \mathbb{R}^{nxh}$:

$$\boldsymbol{H}_t = \boldsymbol{O}_t \odot \tanh(\boldsymbol{C}_t)$$

Deep Recurrent Neural Networks (DRNNs) are easily accomplished by stacking l RNN's.

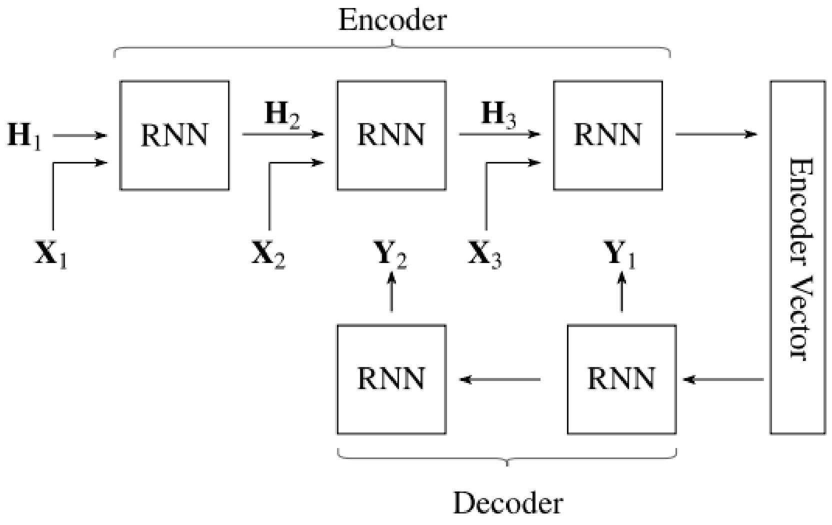
$$\boldsymbol{H}_t^{(1)} = \phi_1(\boldsymbol{X}_t, \boldsymbol{H}_{t-1}^{(1)})$$

$$\boldsymbol{H}_t^{(l)} = \phi_l(\boldsymbol{H}_t^{(l-1)}, \boldsymbol{H}_{t-1}^{(l)})$$

Where the output is computed using only the hidden state of layer L:

$$\boldsymbol{O}_t = \phi_o(\boldsymbol{H}_t^{(L)} \boldsymbol{W}_{ho} + \boldsymbol{b}_o)$$

Moving on to Encoder-Decoder architecture, recall that it is a type of neural network architecture with an encoder network and a decoder network. The encoder encodes the input into a state, and the decoder decodes the state into an output. Here is how you can use RNN's for the Encoder-Decoder architecture:


Encoder
H1
RNN
H2
RNN
H3
RNN
X1
X2
Y2
X3
Y1
Encoder Vector
RNN
RNN
Decoder