Notes: Recurrent Neural Network (RNN)

Notes on general structure of a recurrent neural network

Sarbajit Sarkar

December, 2025

Abstract

These notes provide a comprehensive theoretical framework for Recurrent Neural Networks (RNNs), a class of neural algorithms specifically designed to process sequential data, addressing the "amnestic" limitations of standard Multi-Layer Perceptrons (MLPs). The document details the unique architecture of RNNs, where neuron layers are arranged temporally, allowing the system to maintain a hidden state that acts as a memory of previous inputs. A rigorous mathematical formulation is presented for the forward pass, defining the element-wise and matrix operations for updating hidden states and generating outputs using activation functions like tanh and softmax . The core of the analysis focuses on the derivation of Backpropagation Through Time (BPTT), establishing the recursive relationships necessary to calculate the partial derivatives of the cost function with respect to the model parameters. Finally, the notes outline the training loop, utilizing these calculated gradients to optimize the network via gradient descent algorithms.

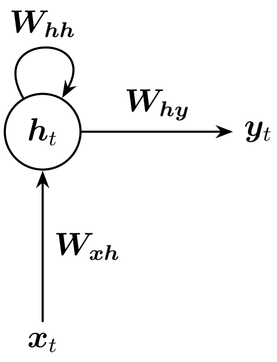
1 Introduction

Recurrent neural networks are a category of neural network algorithms designed to work with sequential data. Basic neural networks like MLPs are amnestic. It can not handle sequential data input (example, sentences in human language). RNNs are specifically designed to handle to process those kind of data.

2 Architecture of the system

There is significant difference in the concept of a *multi layer perceptron* and a *recurrent neural network*. Unlike MLP, the layers of neurons are layered temporally in case of an RNN. It is capable to take sequential vector input one at a time in every timestep, return an output and update hidden state of itself according to the sequence of input.

2.1 Recurrent neuron layer



Analogy

We can think *recurrent neural networks* be a very complex function, $F(x_t) = y_t$. The function have some parameters called *weights and biases* and a "memory" called *hidden state*. As we input some entity to the function, it returns an output and updates its own *hidden state* according to the current hidden state and input.

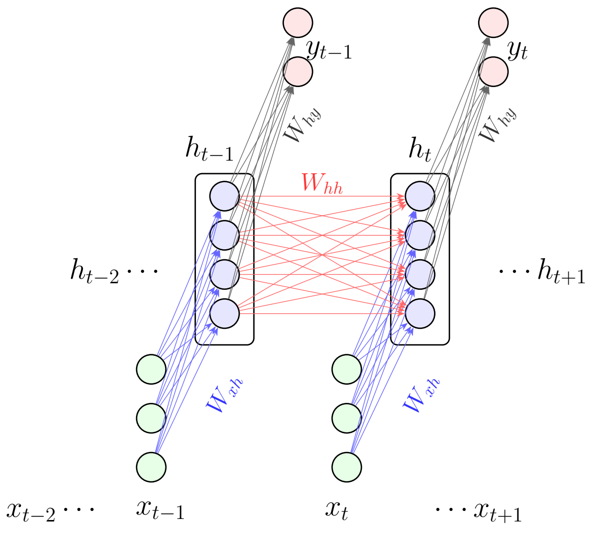


Figure 1: Unrolled through time picture of recurrent neural network

Despite the dissimilarities, we can compare RNNs with a basic neural network (read MLP). Assume RNN as a neural network that have the same set of weights and biases for all hidden layers. Also, the hidden layers have the same width. Always remember that unlike MLP these layers are structured temporally (in the time steps). Also, we can provide inputs and get outputs at each layer, that is not the case for MLP.

Mathematical formulation

The input, output and hidden state at timestep t is given by

$$\boldsymbol{x}_t = \begin{bmatrix} x_1^{(t)} \\ x_2^{(t)} \\ \vdots \\ x_m^{(t)} \end{bmatrix}_{m\times 1} \qquad \boldsymbol{y}_t = \begin{bmatrix} y_1^{(t)} \\ y_2^{(t)} \\ \vdots \\ y_k^{(t)} \end{bmatrix}_{k\times 1} \qquad \boldsymbol{h}_t = \begin{bmatrix} h_1^{(t)} \\ h_2^{(t)} \\ \vdots \\ h_n^{(t)} \end{bmatrix}_{n\times 1}$$

Weights and biases are defined as,

$$\boldsymbol{W_{xh}} = \begin{bmatrix} w_{1,1}^{(xh)} & w_{1,2}^{(xh)} & \cdots & w_{1,m}^{(xh)} \\ w_{2,1}^{(xh)} & w_{2,2}^{(xh)} & \cdots & w_{2,m}^{(xh)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1}^{(xh)} & w_{n,2}^{(xh)} & \cdots & w_{n,m}^{(xh)} \end{bmatrix}_{n\times m} \qquad \boldsymbol{W_{hh}} = \begin{bmatrix} w_{1,1}^{(hh)} & w_{1,2}^{(hh)} & \cdots & w_{1,n}^{(hh)} \\ w_{2,1}^{(hh)} & w_{2,2}^{(hh)} & \cdots & w_{2,n}^{(hh)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n,1}^{(hh)} & w_{n,2}^{(hh)} & \cdots & w_{n,n}^{(hh)} \end{bmatrix}_{n\times n}$$

$$\boldsymbol{W_{hy}} = \begin{bmatrix} w_{1,1}^{(hy)} & w_{1,2}^{(hy)} & \cdots & w_{1,n}^{(hy)} \\ w_{2,1}^{(hy)} & w_{2,2}^{(hy)} & \cdots & w_{2,n}^{(hy)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,1}^{(hy)} & w_{k,2}^{(hy)} & \cdots & w_{k,n}^{(hy)} \end{bmatrix}_{k\times n} \qquad \boldsymbol{b}_h = \begin{bmatrix} b_1^{(h)} \\ b_2^{(h)} \\ \vdots \\ b_n^{(h)} \end{bmatrix}_{n\times 1} \qquad \boldsymbol{b}_y = \begin{bmatrix} b_1^{(y)} \\ b_2^{(y)} \\ \vdots \\ b_k^{(y)} \end{bmatrix}_{k\times 1}$$

Forward pass

Element-wise,

$$h_i^{(0)} = 0$$
$$z_i^{(t)} = w_{ij}^{(hh)} h_j^{(t-1)} + w_{ik}^{(xh)} x_k^{(t)} + b_i^{(h)}$$
$$h_i^{(t)} = f\left(z_i^{(t)}\right)$$
$$o_i^{(t)} = w_{ij}^{(hy)}\, h_j^{(t)} + b_i^{(y)}$$
$$y_i^{(t)} = g\left(o_i^{(t)}\right)$$

Matrix form,

$$\boldsymbol{h}_0 = \boldsymbol{0}_{n\times 1}$$
$$\boldsymbol{z}_t = \boldsymbol{W_{hh}} \boldsymbol{h}_{t-1} + \boldsymbol{W_{xh}} \boldsymbol{x}_t + \boldsymbol{b}_h$$
$$\boldsymbol{h}_t = f\left(\boldsymbol{z}_t\right)$$
$$\boldsymbol{o}_t = \boldsymbol{W_{hy}} \boldsymbol{h}_t + \boldsymbol{b}_y$$
$$\boldsymbol{y}_t = g\left(\boldsymbol{o}_t\right)$$

Note,

- $f()$ is the activation function. Commonly tanh() is used.
- $g()$ is output activation function, eg: *SOFTMAX*
- The system is initialized as, $\boldsymbol{h}_0 = \begin{bmatrix} 0 & 0 & \cdots & 0 \end{bmatrix}^{\mathsf{T}}$
- *Einstein summation convention* is used (repeated indices imply summation).
- RNN is specifically designed to handle sequential input like $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots$

3 Finetune the network

3.1 The cost function

Recurrent neural networks produce an output at every timestep. To calculate the total cost, we need to sum the cost for each timestep t.

$$C = \sum_{t=0}^{T} C_t(\boldsymbol{y}_t, \hat{\boldsymbol{y}}_t)$$

Where, $\boldsymbol{y}_t$ is the predicted output and $\hat{\boldsymbol{y}}_t$ is the desired output at timestep t. The cost function could be a standard metric like *mean squared error* $C_t = \frac{1}{2}\left(y_i^{(t)} - \hat{y}_i^{(t)}\right)^2$

3.2 Backpropagation through time (BPTT)

The BPTT method is similar to the backpropagation in case of MLP.

Objective

To find the partial derivative of cost with respect to model parameters,

$$\frac{\partial C}{\partial w_{ij}^{(xh)}}, \frac{\partial C}{\partial w_{ij}^{(hh)}}, \frac{\partial C}{\partial w_{ij}^{(hy)}}, \frac{\partial C}{\partial b_i^{(h)}}, \frac{\partial C}{\partial b_i^{(y)}}$$

To finally calculate the gradient,

$$\nabla C = \begin{bmatrix} \text{vec}(\nabla_{W_{xh}} C) \\ \text{vec}(\nabla_{W_{hh}} C) \\ \text{vec}(\nabla_{W_{hy}} C) \\ \nabla_{b_h} C \\ \nabla_{b_y} C \end{bmatrix} \qquad \text{where, } \text{vec}(\nabla_{W_{xh}} C) = \begin{bmatrix} \frac{\partial C}{\partial w_{1,1}^{(xh)}} \\ \frac{\partial C}{\partial w_{1,2}^{(xh)}} \\ \vdots \\ \frac{\partial C}{\partial w_{n,m}^{(xh)}} \end{bmatrix} \qquad \nabla_{b_h} C = \begin{bmatrix} \frac{\partial C}{\partial b_1^{(h)}} \\ \frac{\partial C}{\partial b_2^{(h)}} \\ \vdots \\ \frac{\partial C}{\partial b_n^{(h)}} \end{bmatrix}$$

Calculations

Define output error signal $\boldsymbol{e}_t$ as

$$e_i^{(t)} = \frac{\partial C_t}{\partial o_i^{(t)}} = \left(y_i^{(t)} - \hat{y}_i^{(t)}\right) g'\left(o_i^{(t)}\right) \qquad \left[\text{for, } C_t = \frac{1}{2}\left(y_i^{(t)} - \hat{y}_i^{(t)}\right)^2\right]$$

$$\boldsymbol{e}_t = \nabla_{\boldsymbol{o_t}} C = (\boldsymbol{y}_t - \hat{\boldsymbol{y}}_t) \odot g'(\boldsymbol{o}_t) \qquad \text{[Matrix form]}$$

By applying the chain rule we have

$$\frac{\partial C_t}{\partial w_{ij}^{(hy)}} = \frac{\partial C_t}{\partial o_i^{(t)}} \frac{\partial o_i^{(t)}}{\partial w_{ij}^{(hy)}} = e_i^{(t)} h_j^{(t)} \qquad \frac{\partial C_t}{\partial b_i^{(y)}} = \frac{\partial C_t}{\partial o_i^{(t)}} \frac{\partial o_i^{(t)}}{\partial b_i^{(y)}} = e_i^{(t)}$$

$$\implies \frac{\partial C}{\partial w_{ij}^{(hy)}} = \sum_{t=1}^{T} e_i^{(t)} h_j^{(t)} \qquad \implies \frac{\partial C}{\partial b_i^{(y)}} = \sum_{t=1}^{T} e_i^{(t)}$$

Matrix notation,

$$\nabla_{W_{hy}} C = \sum_{t=1}^{T} \boldsymbol{e}_t \boldsymbol{h}_t^{\intercal} \qquad \nabla_{b_y} C = \sum_{t=1}^{T} \boldsymbol{e}_t$$

Define error signal $\boldsymbol{\delta}_t$ as

$$\delta_i^{(t)} = \frac{\partial C}{\partial z_i^{(t)}} = \frac{\partial C}{\partial h_i^{(t)}} f'\left(z_i^{(t)}\right) \qquad \boldsymbol{\delta}_t = \nabla_{\boldsymbol{z}_t} C = \nabla_{\boldsymbol{h}_t} C \odot f'(\boldsymbol{z}_t)$$

Now we calculate the recursive relation of error signal.

$$\frac{\partial C}{\partial h_i^{(t)}} = \underbrace{\frac{\partial C_t}{\partial o_j^{(t)}}\frac{\partial o_j^{(t)}}{\partial h_i^{(t)}}}_{\text{gradient from output}} + \underbrace{\frac{\partial C}{\partial z_j^{(t+1)}}\frac{\partial z_j^{(t+1)}}{\partial h_i^{(t)}}}_{\text{gradient from future timestamp}}$$

$$\implies \frac{\partial C}{\partial h_i^{(t)}} = e_j^{(t)} w_{ji}^{(hy)} + \delta_j^{(t+1)} w_{ji}^{(hh)}$$

Substituting this back into the definition of $\delta_i^{(t)}$:

$$\delta_i^{(t)} = \frac{\partial C}{\partial h_i^{(t)}} f'(z_i^{(t)}) = \left(w_{ji}^{(hy)} e_j^{(t)} + w_{ji}^{(hh)} \delta_j^{(t+1)}\right) f'(z_i^{(t)})$$

For the last timestep there is no error propagating back from future. So, $\delta_i^{(T+1)} = 0$

$$\therefore \delta_i^{(T)} = w_{ji}^{(hy)} e_j^{(T)} \ f'(z_i^{(T)})$$

$\therefore$ We have the recursion relation for the error signal.

Element-wise,	Matrix form,
$\delta_i^{(T)} = w_{ji}^{(hy)} e_j^{(T)} \ f'\left(z_i^{(T)}\right)$	$\boldsymbol{\delta}_T = W_{hy}^\intercal \boldsymbol{e}_T \ \odot \ f'(\boldsymbol{z}_T)$
$\delta_i^{(t)} = \left(w_{ji}^{(hy)} e_j^{(t)} + w_{ji}^{(hh)} \delta_j^{(t+1)}\right) \ f'\left(z_i^{(t)}\right)$	$\boldsymbol{\delta}_t = \left(W_{hy}^\intercal \boldsymbol{e}_t \ + W_{hh}^\intercal \boldsymbol{\delta}_{t+1}\right) \ \odot \ f'(\boldsymbol{z}_t)$

Now we have

$$z_i^{(t)} = w_{ij}^{(hh)} h_j^{(t-1)} + w_{ik}^{(xh)} x_k^{(t)} \ + b_i^{(h)}$$

$$\therefore \quad \frac{\partial z_i^{(t)}}{\partial w_{ij}^{(hh)}} = h_j^{(t-1)} \qquad \frac{\partial z_i^{(t)}}{\partial w_{ij}^{(xh)}} = x_j^{(t)} \qquad \frac{\partial z_i^{(t)}}{\partial b_i^{(h)}} = 1$$

Apply chain rule

$$\frac{\partial C_t}{\partial w_{ij}^{(hh)}} = \frac{\partial C_t}{\partial z_i^{(t)}} \frac{\partial z_i^{(t)}}{\partial w_{ij}^{(hh)}} = \delta_i^{(t)} h_j^{(t-1)} \qquad \frac{\partial C_t}{\partial b_i^{(h)}} = \frac{\partial C_t}{\partial z_i^{(t)}} \frac{\partial z_i^{(t)}}{\partial b_i^{(h)}} = \delta_i^{(t)}$$

$$\implies \frac{\partial C}{\partial w_{ij}^{(hh)}} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial w_{ij}^{(hh)}} = \sum_{t=1}^{T} \delta_i^{(t)} h_j^{(t-1)} \qquad \implies \frac{\partial C}{\partial b_i^{(h)}} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial b_i^{(h)}} = \sum_{t=1}^{T} \delta_i^{(t)}$$

$$\frac{\partial C_t}{\partial w_{ij}^{(xh)}} = \frac{\partial C_t}{\partial z_i^{(t)}} \frac{\partial z_i^{(t)}}{\partial w_{ij}^{(xh)}} = \delta_i^{(t)} x_j^{(t)}$$

$$\implies \frac{\partial C}{\partial w_{ij}^{(xh)}} = \sum_{t=1}^{T} \frac{\partial C_t}{\partial w_{ij}^{(xh)}} = \sum_{t=1}^{T} \delta_i^{(t)} x_j^{(t)}$$

Matrix form,

$$\nabla_{W_{hh}} C = \sum_{t=1}^{T} \boldsymbol{\delta}_t \ \boldsymbol{h}_{t-1}^\intercal \qquad \nabla_{W_{xh}} C = \sum_{t=1}^{T} \boldsymbol{\delta}_t \ \boldsymbol{x}_t^\intercal \qquad \nabla_{b_h} C = \sum_{t=1}^{T} \boldsymbol{\delta}_t$$

Summary

$$e_i^{(t)} = \frac{\partial C_t}{\partial o_i^{(t)}} = \frac{\partial C_t}{\partial y_i^{(t)}}\ g'\left(o_i^{(t)}\right) \qquad \boldsymbol{e_t} = \nabla_{\boldsymbol{y}_t} C\ \odot\ g'(\boldsymbol{o}_t)$$

$$\frac{\partial C}{\partial w_{ij}^{(hy)}} = \sum_{t=1}^{T} e_i^{(t)} h_j^{(t)} \qquad \nabla_{W_{hy}} C = \sum_{t=1}^{T} \boldsymbol{e}_t \boldsymbol{h}_t^{\intercal}$$

$$\frac{\partial C}{\partial b_i^{(y)}} = \sum_{t=1}^{T} e_i^{(t)} \qquad \nabla_{b_y} C = \sum_{t=1}^{T} \boldsymbol{e}_t$$

$$\delta_i^{(T)} = w_{ji}^{(hy)} e_j^{(T)}\ f'\left(z_i^{(T)}\right) \qquad \boldsymbol{\delta}_T = W_{hy}^{\intercal} \boldsymbol{e}_T\ \odot\ f'(\boldsymbol{z}_T)$$

$$\delta_i^{(t)} = \left(w_{ji}^{(hy)} e_j^{(t)} + w_{ji}^{(hh)} \delta_j^{(t+1)}\right)\ f'\left(z_i^{(t)}\right) \qquad \boldsymbol{\delta}_t = \left(W_{hy}^{\intercal} \boldsymbol{e}_t\ + W_{hh}^{\intercal} \boldsymbol{\delta}_{t+1}\right)\ \odot\ f'(\boldsymbol{z}_t)$$

$$\frac{\partial C}{\partial w_{ij}^{(hh)}} = \sum_{t=1}^{T} \delta_i^{(t)} h_j^{(t-1)} \qquad \nabla_{W_{hh}} C = \sum_{t=1}^{T} \boldsymbol{\delta}_t\ \boldsymbol{h}_{t-1}^{\intercal}$$

$$\frac{\partial C}{\partial w_{ij}^{(xh)}} = \sum_{t=1}^{T} \delta_i^{(t)} x_j^{(t)} \qquad \nabla_{W_{xh}} C = \sum_{t=1}^{T} \boldsymbol{\delta}_t\ \boldsymbol{x}_t^{\intercal}$$

$$\frac{\partial C}{\partial b_i^{(h)}} = \sum_{t=1}^{T} \delta_i^{(t)} \qquad \nabla_{b_h} C = \sum_{t=1}^{T} \boldsymbol{\delta}_t$$

So now we have all the pieces to assemble the gradient of cost function with respect to model parameters, ∇C.

3.3 Training

The training process of RNN is very similar to the training process of MLP. It involves gradient descent of model parameters to optimize the cost function. We need a large collection of sequential data to train a model efficiently. A training loop consists of 4 steps.

Forward pass

Activation and output of every timestep is calculated using the formula. Thus, $\boldsymbol{h}_t$, $\boldsymbol{o}_t$, $\boldsymbol{y}_t$ at each timestep t is calculated.

Backward pass

Error of each time step is calculated using BPTT. Thus we have $\boldsymbol{\delta}_t$ and $\boldsymbol{e}_t$.

Gradient calculation

Gradient of cost function, ∇C is calculated using the formula. Generally, Gradient of each training data of the batch is calculated independently and then average is taken.

Let assume ith training datapoint is $\{x_1^{(i)}, x_2^{(i)}, \ldots x_T^{(i)}\}$. The gradient for each datapoint $\left(\nabla C^{(i)}\right)$ in the batch is calculated and then averaged. Thus,

$$\nabla C = \frac{1}{N} \sum_{i=1}^{N} \nabla C^{(i)}$$

Parameter update

The model parameters are updated as per the *gradient descent* algorithm.

$$\boldsymbol{v}' = \boldsymbol{v} - \alpha \, \nabla C \qquad \text{Where, } \boldsymbol{v} = \begin{bmatrix} \text{vec}(W_{xh}) \\ \text{vec}(W_{hh}) \\ \text{vec}(W_{hy}) \\ \boldsymbol{b_h} \\ \boldsymbol{b_y} \end{bmatrix}$$

α is a scalar referred as learning rate

$\boldsymbol{v}'$ is updated model parameter vector

More efficient algorithms like *stochastic gradient descent* could also be applied to update model parameters.

4 Conclusion

The derivation presented confirms that Recurrent Neural Networks can effectively model temporal dependencies by utilizing a shared set of weights and biases across all time steps. The analysis of the cost function demonstrates that the total error is the cumulative sum of errors at each specific timestep t. Crucially, the Backpropagation Through Time (BPTT) method yields a recursive definition for the error signal $\boldsymbol{\delta}_t$, which propagates backward from the final timestep T to update the hidden layer weights. By aggregating these error signals, the document provides the complete analytical formulas for the gradients. These gradients form the basis for the parameter update step in the training loop, where weights are adjusted iteratively to minimize the cost function using optimization techniques such as Stochastic Gradient Descent.