Recurrent Neural Networks

Tadeu Silva (Deds)

July 22, 2021

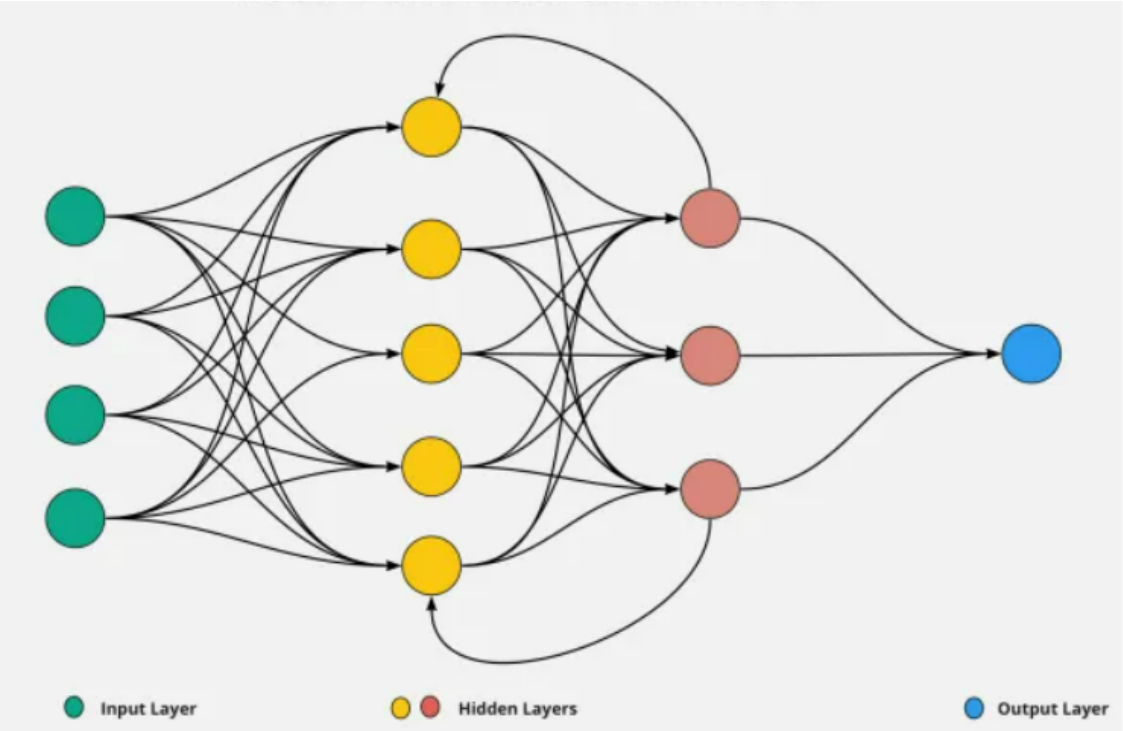


Figure 1: Neural Network Unfolding Source: dataaspirant

1 Introduction

A Recurrent Neural Network is a type of architecture of Deep Learning which introduces the concept of **cell state** of a neuron. This allows for handling of sequential/temporal data (like blocs of text). This allows different configurations for training/testing data, as follows:

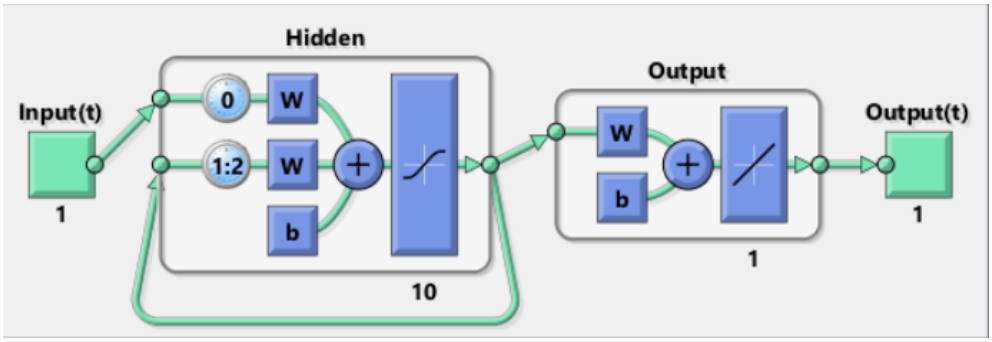


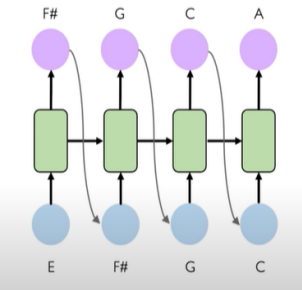
Figure 2: Architecture of Recurrent Neural Network

1.1 Options for Input/Output

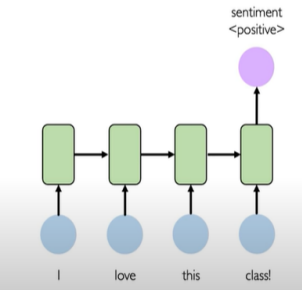
RNN's are very useful because they provide **memory** for last hidden state, allowing the neurons to remember past configurations. Those elements are really important when handling highly temporal-sensitive data, like understanding a sentence.

For example the sentence **The food was good, not bad at all** vs **The food was bad, not good at all** shows the importante of order in text. Another important feature is correlation with long periods, like for example the sentence **Bob is eating an apple. Lisa likes orange. Lisa is having lunch with Bob and she is drinking water.** has various states depending of what the question is asked. So we could ask *who is eating an Apple?* or *Who is drinking water?* and while the latter is easier to answer because the answer is at the end of the phrase, the former is not because we have to propagate the understanding of what Bob is doing throughout the hole sequence of data.

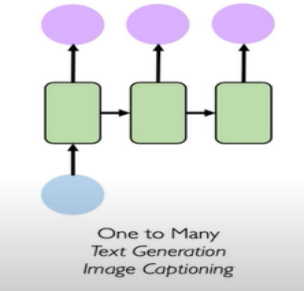
This kind of flexibility is reflected on different ways of data handling, like:



(a) Many to Many



(b) Many to One



(c) One to Many

Figure 3: Different ways of Input/Output

Source of images: MIT 6.S191: Recurrent Neural Networks - Ava Soleimany

- **Many to Many:** Prediction of the next note for compose music
- **Many to One:** Prediction of sentiment based on given text
- **One to Many:** Generating Text for given Image (number of persons on picture, etc)

2 Forward Pass

Given an input $\overrightarrow{x}_t$ in the Input Layer, the Hidden Layer has output:

$$a_t^1 = \sigma(z_t^1) = \sigma({}^{XH}W^1 \cdot \overrightarrow{x_t} + {}^{HH}W \cdot a_{t-1}^1 + b^1) \tag{1}$$

where ${}^{XH}W$ is the **Input-Hidden Weight** Matrix, ${}^{HH}W$ is the **Hidden-Hidden Weight** Matrix, b is the **bias** vector and $\overrightarrow{x_t}$ is the **input** vector.
Obs:

- $\overrightarrow{x_t}$ is a $(m, 1)$ vector
- ${}^{XH}W$ is a (n, m) matrix
- ${}^{HH}W$ is a (n, n) matrix
- b is a $(n, 1)$ matrix

Notice that the difference here is that we take samples of $\overrightarrow{x}$ throughout time. So if we're taking a NLP approach, we may have:

$$x = ['I', 'love', 'deep', 'learning'] \tag{2}$$

where:

- $x[0] =' I'$
- $x[1] =' love'$
- $x[2] =' learning'$
- $x[3] =' deep'$

and we're taking samples of size $T = 4$. So we would repeat equation (1) for every entry of $\overrightarrow{x}$, where we are repetidly updating the 'cell state' of the neuron through time, making sure that the whole phrase is taking in consideration. Of course that when implementing we would change the strings to numbers, via hot-encoding algorithm or word embedding (for not causing too sparse and large binary vectors).

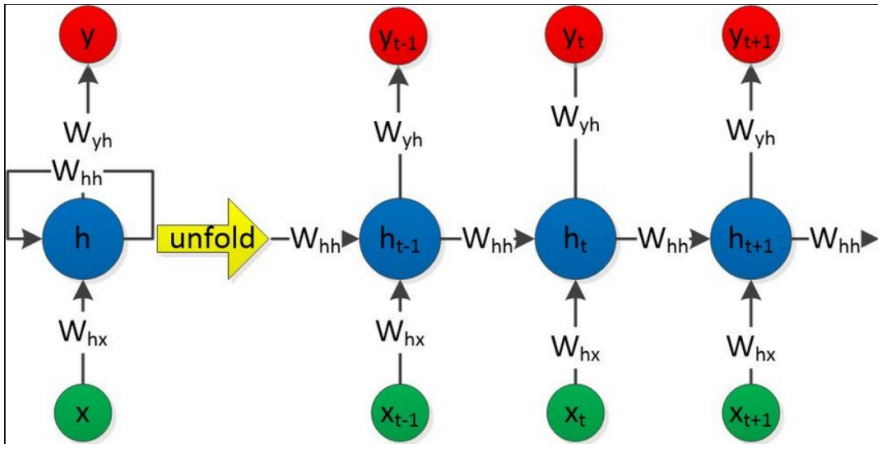


Figure 4: Acumulation through time in a RNN. W_{hh}, W_{hx}, W_{yh} fixed in time. Source: researchgate

2.1 Example

In a simple 2-3-2 Layer Type of Network, we have:

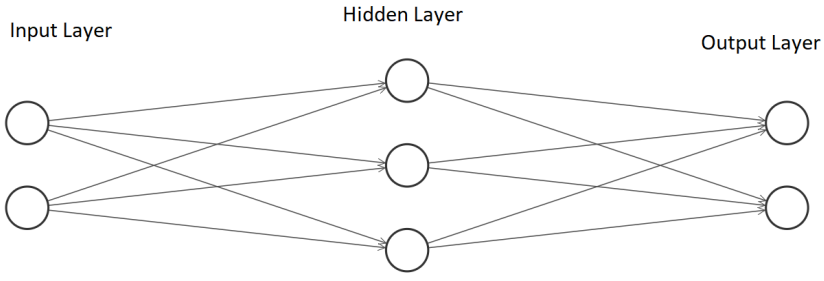


Figure 5: Simple 2-3-2 Recurrent Neural Network Example

Input Layer - Hidden Layer

$$\begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{bmatrix}_t = \sigma\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}_t \cdot \begin{bmatrix} {}^{hx}W_{11} & {}^{hx}W_{12} \\ {}^{hx}W_{21} & {}^{hx}W_{22} \\ {}^{hx}W_{31} & {}^{hx}W_{32} \end{bmatrix} + \begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{bmatrix}_{t-1} \cdot \begin{bmatrix} {}^{hh}W_{11} & {}^{hh}W_{12} & {}^{hh}W_{13} \\ {}^{hh}W_{21} & {}^{hh}W_{22} & {}^{hh}W_{23} \\ {}^{hh}W_{31} & {}^{hh}W_{32} & {}^{hh}W_{33} \end{bmatrix} + \begin{bmatrix} b_1^1 \\ b_2^1 \\ b_3^1 \end{bmatrix} \right) \tag{3}$$

Hidden Layer - Output Layer

$$\begin{bmatrix} a_1^2 \\ a_2^2 \end{bmatrix}_t = \sigma\left(\begin{bmatrix} z_1^2 \\ z_2^2 \end{bmatrix}_t \right) = \sigma\left(\begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{bmatrix}_t \cdot \begin{bmatrix} {}^{hy}W_{11} & {}^{hy}W_{12} & {}^{hy}W_{13} \\ {}^{hy}W_{21} & {}^{hy}W_{22} & {}^{hy}W_{23} \end{bmatrix} + \begin{bmatrix} b_1^2 \\ b_2^2 \end{bmatrix} \right) \tag{4}$$

3 Backward Pass

3.1 Loss

In the Backward pass, we first compute our **loss** (aka the error of our output) and average along the data, to get a measure of our total uncertainty. The difference on the RNNs is that we need to compute all the gradients from previous timesteps. We have:

$$loss = (\frac{1}{n}) \sum_{i=0}^{n} \mu\left(loss(a_{i,t}^2, y_{i,t})\right) = (\frac{1}{T})(\frac{1}{n}) \sum_{i=0}^{n} \sum_{t=0}^{T} \mu\left(loss(a_{i,t}^2, y_{i,t})\right) \tag{5}$$

Where μ is the temporal average on the column matrix. So for our example:

$$loss = (\frac{1}{T})(\frac{1}{2}) \sum_{t=0}^{T} \mu\left(\begin{bmatrix} (a_1^2 - y_1)^2 \\ (a_2^2 - y_2)^2 \end{bmatrix}_t \right) = (\frac{1}{T})(\frac{1}{2}) \sum_{t=0}^{T} \left((a_1^2 - y_1)^2 + (a_2^2 - y_2)^2\right)_t \tag{6}$$

where n is the dimension of the output layer, using the **Mean Squared Error** (which is not recommendable for probability distribution problems, like NLP.

3.2 Backpropagation Through Time (BTT)

The second part is to calculate how each weight and bias had influence in the loss, for every timestep taken. So we want to calculate:

$$\nabla C = \begin{bmatrix} \frac{\partial C}{\partial W} \\ \\ \frac{\partial C}{\partial b} \end{bmatrix}, \frac{\partial C}{\partial W} = \begin{bmatrix} \frac{\partial C}{\partial {}^{hx}W} \\ \\ \frac{\partial C}{\partial {}^{hh}W} \\ \\ \frac{\partial C}{\partial {}^{hy}W} \end{bmatrix}, \frac{\partial C}{\partial W} = \begin{bmatrix} \frac{\partial C}{\partial b_1} \\ \\ \frac{\partial C}{\partial b_2} \end{bmatrix} \tag{7}$$

We do that using the Backpropagation (chain rule). Remembering from calculus, the derivative of a composite function $f(g(x))$ can be written as:

$$\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx} \tag{8}$$

So applying these concepts for the layers, we have:

$$\frac{\partial C}{\partial W} = \left(\frac{\partial C}{\partial a}\right)\left(\frac{\partial a}{\partial z}\right)\left(\frac{\partial z}{\partial W}\right) \tag{9}$$

$$\frac{\partial C}{\partial b} = \left(\frac{\partial C}{\partial a}\right)\left(\frac{\partial a}{\partial z}\right)\left(\frac{\partial z}{\partial b}\right) \tag{10}$$

The activation function σ depends on the problem at hand. But let's suppose we are at a **classification** problem, so we'll go with the **Softmax** function, defined as:

$$\sigma(z) = \frac{e^z}{\sum_{j=0}^{n} e^{z_j}} \tag{11}$$

So the derivative is:

$$\frac{d\sigma}{\partial z} = z(1-z)$$

Let's denote the following two operations:

- $\times$: element-wise multiplication, i.e. same dimensions only
- $\cdot$: dot product, i.e (m,k) $\cdot$ (k,n) = (m,n)

This means that if:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \Rightarrow AXB = \begin{bmatrix} a_{11} \cdot b_{11} & a_{12} \cdot b_{12} \\ a_{21} \cdot b_{21} & a_{22} \cdot b_{22} \end{bmatrix}$$

Analogously, if:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} \\ b_{21} \end{bmatrix} \Rightarrow A \cdot B = \begin{bmatrix} a_{11} \cdot b_{11} + a_{12} \cdot b_{21} \\ a_{21} \cdot b_{11} + a_{22} \cdot b_{21} \end{bmatrix}$$

Analyzing the nuances for each layer, we have:

1. Output - Hidden Layer

For the Output Layer, we get:

$$\frac{\partial C}{\partial^{hy} W} = \left(\frac{\partial C}{\partial a^2}\right)\left(\frac{\partial a^2}{\partial z^2}\right)\left(\frac{\partial z^2}{\partial^{hy} W}\right) \tag{12}$$

$$\frac{\partial C}{\partial a^2} = (\frac{1}{T})\sum_{t=0}^{T} 2(a_2^2 - y_i)_t = (\frac{2}{T})\sum_{t=0}^{T}\begin{bmatrix}(a_1^2 - y_1)\\(a_2^2 - y_2)\end{bmatrix}_t$$

$$\frac{\partial a^2}{\partial z^2} = \frac{d\sigma}{\partial z^2} = (\frac{1}{T})\sum_{t=0}^{T} z_t^2(1 - z^2)_t = (\frac{1}{T})\sum_{t=0}^{T}\begin{bmatrix}z_1^2(1 - z_1^2)\\z_2^2(1 - z_2^2)\end{bmatrix}_t$$

$$\frac{\partial z^2}{\partial^{hy} W} = (\frac{1}{T})\sum_{t=0}^{T} a_t^1 = (\frac{1}{T})\sum_{t=0}^{T}\begin{bmatrix}a_1^1\\a_2^1\\a_3^1\end{bmatrix}_t$$

So we have:

$$\frac{\partial C}{\partial^{hy} W} = (\frac{1}{T})\sum_{t=0}^{T}\left[\begin{bmatrix}2(a_1^2 - y_1)\\2(a_2^2 - y_2)\end{bmatrix}_t \times \begin{bmatrix}z_1^2(1 - z_1^2)\\z_2^2(1 - z_2^2)\end{bmatrix}_t \cdot \left(\begin{bmatrix}a_1^1\\a_2^1\\a_3^1\end{bmatrix}_t\right)^T\right]$$

Analogously, for the bias:

$$\frac{\partial C}{\partial b^2} = \left(\frac{\partial C}{\partial a^2}\right)\left(\frac{\partial a^2}{\partial z^2}\right)\left(\frac{\partial z^2}{\partial b^2}\right) \tag{13}$$

So the two terms are the same. For the last one, note that because of the definition of z^2:

$$\frac{\partial z^2}{\partial b^2} = 1$$

Then we get:

$$\frac{\partial C}{\partial a^2} = (\frac{1}{T})\sum_{t=0}^{T}\left[\begin{bmatrix}2(a_1^2 - y_1)\\2(a_2^2 - y_2)\end{bmatrix}_t \times \begin{bmatrix}z_1^2(1 - z_1^2)\\z_2^2(1 - z_2^2)\end{bmatrix}_t\right]$$

1. Hidden - Input Layer

Analogously we have:

$$\frac{\partial C}{\partial^{hx} W} = \left(\frac{\partial C}{\partial a^1}\right)\left(\frac{\partial a^1}{\partial z^1}\right)\left(\frac{\partial z^1}{\partial^{hx} W}\right) \tag{14}$$

$$\frac{\partial C}{\partial a^1} = \left(\frac{\partial C}{\partial a^2}\right)\left(\frac{\partial a^2}{\partial z^2}\right)\left(\frac{\partial z^2}{\partial a^1}\right)$$

Note that, by definition:

$$\frac{\partial z^2}{\partial a^1} = {}^{hy}W$$

So:

$$\frac{\partial C}{\partial a^1} = \left((\frac{1}{T}) \sum_{t=0}^{T} \left[\begin{bmatrix} 2(a_1^2 - y_1) \\ 2(a_2^2 - y_2) \end{bmatrix}_t \times \begin{bmatrix} z_1^2(1 - z_1^2) \\ z_2^2(1 - z_2^2) \end{bmatrix}_t \right]^T \cdot \begin{bmatrix} {}^{hy}W_{11} & {}^{hy}W_{12} & {}^{hy}W_{13} \\ {}^{hy}W_{21} & {}^{hy}W_{22} & {}^{hy}W_{23} \end{bmatrix} \right)^T$$

The second term from equation (14) is:

$$\frac{\partial a^1}{\partial z^1} = \begin{bmatrix} z_1^1(1 - z_1^1) \\ z_2^1(1 - z_2^1) \\ z_3^1(1 - z_3^1) \end{bmatrix}_t$$

For the last term, let's check what happens with t=0:

$$z_0^1 = x_0 \cdot {}^{hx}W + b^1 \Rightarrow \frac{\partial z_0^1}{\partial {}^{hx}W} = x_0$$

Now for t = 1:

$$z_1^1 = x_1 \cdot {}^{hx}W + a_0^1 \cdot {}^{hh}W + b^1 = x_1 \cdot {}^{hx}W + \sigma(z_0^1) \cdot {}^{hh}W + b^1$$
$$\Rightarrow \frac{\partial z_1^1}{\partial {}^{hx}W} = x_1 + z_0^1(1 - z_0^1)x_0 \cdot {}^{hh}W$$

If we do it one more time, for t = 2, we can see the pattern happening:

$$z_2^1 = x_2 \cdot {}^{hx}W + a_1^1 \cdot {}^{hh}W + b^1 = x_2 \cdot {}^{hx}W + \sigma(z_1^1) \cdot {}^{hh}W + b^1$$
$$\Rightarrow \frac{\partial z_2^1}{\partial {}^{hx}W} = x_2 + z_1^1(1 - z_1^1)\big[x_1 + z_0^1(1 - z_0^1)x_0 \cdot {}^{hh}W\big] \cdot {}^{hh}W$$

So we are acumulating the derivatives of past timesteps. Reorganizing the terms, we can rewrite the above equations as:

$$t = 0 : \frac{\partial z_0^1}{\partial {}^{hx}W} = x_0$$
$$t = 1 : \frac{\partial z_1^1}{\partial {}^{hx}W} = x_1 + \sigma'(z_0^1)\Big(\frac{\partial z_0^1}{\partial {}^{hx}W}\Big) \cdot {}^{hh}W$$
$$t = 2 : \frac{\partial z_2^1}{\partial {}^{hx}W} = x_2 + \sigma'(z_1^1)\Big(\frac{\partial z_1^1}{\partial {}^{hx}W}\Big) \cdot {}^{hh}W$$
$$t = k : \frac{\partial z_k^1}{\partial {}^{hx}W} = x_k + \sigma'(z_{k-1}^1)\Big(\frac{\partial z_{k-1}^1}{\partial {}^{hx}W}\Big) \cdot {}^{hh}W$$

Congratulations! If you understood the above derivation, you now understands the **Backpropagation Through Time**, the most complex of the backpropagation family for Deep Learning.

Wrapping things up, from equation (14) we have:

$$\frac{\partial C}{\partial {}^{hx}W} = (\frac{1}{T})\sum_{t=0}^{T}\left(\left(\frac{\partial C}{\partial a_2}\right) \times \begin{bmatrix} z_1^1(1-z_1^1) \\ z_2^1(1-z_2^1) \\ z_3^1(1-z_3^1) \end{bmatrix}_t \cdot \left(x_t + \sigma^{'}(z_{t-1}^1)\left(\frac{\partial z_{t-1}^1}{\partial {}^{hx}W}\right) \cdot {}^{hh}W\right)^T\right)$$

The good news is that the other derivative for the weight ${}^{hh}W$ is very similar to the previous one. So this will be a good reminder that you understood the equations. Let's dive in:

Generally we have:

$$\frac{\partial C}{\partial {}^{hh}W} = \left(\frac{\partial C}{\partial a^1}\right)\left(\frac{\partial a^1}{\partial z^1}\right)\left(\frac{\partial z^1}{\partial {}^{hh}W}\right) \tag{15}$$

The first two terms are the same as above, there is:

$$\frac{\partial C}{\partial a^1} = \left((\frac{1}{T})\sum_{t=0}^{T}\left[\begin{bmatrix} 2(a_1^2 - y_1) \\ 2(a_2^2 - y_2) \end{bmatrix}_t \times \begin{bmatrix} z_1^2(1-z_1^2) \\ z_2^2(1-z_2^2) \end{bmatrix}_t\right]^T \cdot \begin{bmatrix} {}^{hy}W_{11} & {}^{hy}W_{12} & {}^{hy}W_{13} \\ {}^{hy}W_{21} & {}^{hy}W_{22} & {}^{hy}W_{23} \end{bmatrix}\right)^T$$

And:

$$\frac{\partial a^1}{\partial z^1} = \begin{bmatrix} z_1^1(1-z_1^1) \\ z_2^1(1-z_2^1) \\ z_3^1(1-z_3^1) \end{bmatrix}_t$$

The last term is the one we need to be careful, due the time dependence. So let's do the same analysis. For t = 0 we have:

$$z_0^1 = x_0 \cdot {}^{hx}W + b^1 \Rightarrow \frac{\partial z_0^1}{\partial {}^{hh}W} = 0$$

Now, for t = 1 we have:

$$z_1^1 = x_1 \cdot {}^{hx}W + a_0^1 \cdot {}^{hh}W + b^1 \Rightarrow \frac{\partial z_1^1}{\partial {}^{hh}W} = a_0^1$$

Finally, for t = 2:

$$z_2^1 = x_2 \cdot {}^{hx}W + a_1^1 \cdot {}^{hh}W + b^1 \Rightarrow \frac{\partial z_2^1}{\partial {}^{hh}W} = a_1^1$$

So, for t = k:

$$z_k^1 = x_k \cdot {}^{hx}W + a_{k-1}^1 \cdot {}^{hh}W + b^1 \Rightarrow \frac{\partial z_k^1}{\partial {}^{hh}W} = a_{k-1}^1$$

Much easier! Summarizing, from equation (15) we then get:

$$\frac{\partial C}{\partial {}^{hh}W} = (\frac{1}{T})\sum_{t=0}^{T}\left(\left(\frac{\partial C}{\partial a_2}\right) \times \begin{bmatrix} z_1^1(1-z_1^1) \\ z_2^1(1-z_2^1) \\ z_3^1(1-z_3^1) \end{bmatrix}_t \cdot \begin{bmatrix} a_1^1 \\ a_2^1 \\ a_3^1 \end{bmatrix}_{t-1}\right)$$

Finally, for the bias we have:

$$\frac{\partial C}{\partial b^1} = \left(\frac{\partial C}{\partial a^1}\right)\left(\frac{\partial a^1}{\partial z^1}\right)\left(\frac{\partial z^1}{\partial b^1}\right) \tag{16}$$

So, the same way as before, by definition:

$$\frac{\partial z^1}{\partial b^1} = 1$$

We then have:

$$\frac{\partial C}{\partial b^1} = \left(\frac{\partial C}{\partial a^1}\right)\left(\frac{\partial a^1}{\partial z^1}\right)$$

3.3 Optimizing

For the last step, we optimize the weights accordingly to the changes calculated on the previous section. There are a few optimizers, but the two most used are: **SGD** (Stochastic Gradient Descent) and **Adam**.

SGD - Stochastic Gradient Descent
The SGD is based in the simple idea that: the gradient shows the growth direction. Because we want to **decrease** our error/loss, we simple get the opposite direction adding the minus sign. So the equation is:

$${}^{hy}W = W^l - lr \times \frac{\partial C}{\partial {}^{hy}W} \tag{17}$$

$${}^{hx}W = W^l - lr \times \frac{\partial C}{\partial {}^{hx}W} \tag{18}$$

$${}^{hh}W = W^l - lr \times \frac{\partial C}{\partial {}^{hh}W} \tag{19}$$

$$b^1 = b^l - lr \times \frac{\partial C}{\partial b^1} \tag{20}$$

$$b^2 = b^l - lr \times \frac{\partial C}{\partial b^2} \tag{21}$$

where lr is the **learning rate**, a parameter $(lr < 1)$ just to make sure that we don't make great jumps and 'miss' the local minima.