Back Propagation Through Time for Recurrent Neural Networks

Ramkumar

2025-04-27

Table of contents

Preface

Here, I have derived the *back propagation through time* equations for the recurrent neural networks (RNN) as a part of my academics work. This derivation was made with an example RNN that has 2 input, 3 hidden neurons and 2 output.

The derivation was then extended to a general RNN having n input, $N^{(1)}$ hidden neurons and $N^{(2)}$ output. I concluded by specifying the pseudocode for RNN computation with single data having T subsamples, which is easy to extend for a dataset having multiple datapoints of different T values.

1 Introduction

Recurrent Neural Networks (RNNs) are the type of neural networks used for mapping sequential data. It has same architecture of regular multi-layer perceptrons (MLPs), except that it will take additional components in its input along with data.

RNNs require the sequential data to be split into sub-parts and then it will process each sub-part. The additional component that will be concatenated to the input vector of current sub-part is the hidden layer output vector that was obtained for the previous sub-part. By this way, the information about data sequence is learned by the network.

Hence, the back propagation equations will be different from the regular MLPs for these RNNs as they have to handle the information sequence for updating weights. Thus, it is called back propagation through time (BPTT).

For this derivation of BPTT, a simple RNN that as input vector size $n = 2$, output vector size $N^{(2)} = 2$ and hidden neurons $N^{(1)} = 3$ was taken as shown in Figure 1.1 .

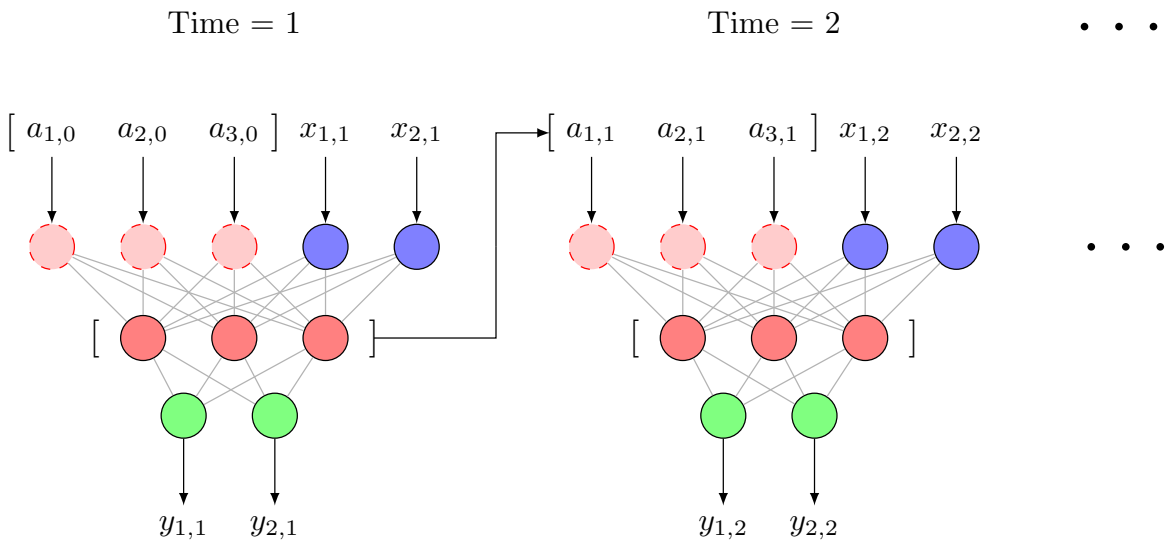

Figure 1.1: Recurrent Neural Network for BPTT derivation

2 Forward Propagation in RNN

For the BPTT derivation, we need to know the parameters of RNN that are to be updated. For that, we need to perform forward propagation to get the model equations. The RNN considered for derivation has 2 layers: one hidden layer having $N^{(1)}$ neurons and one output layer having $N^{(2)}$ neurons.

Let $\mathbf{x}_t \in \mathbb{R}^n$ be the input vector for current subpart t. And let $\mathbf{a}_{t-1} \in \mathbb{R}^{N^{(1)}}$ be the hidden layer output vector for the previous subpart $t-1$. Then, the layer-wise forward propagation equations are given below.

2.1 Layer 1 equations

From Figure 1.1, the output size of layer 1 is $N^{(1)} = 3$ and input vector size $n = 2$.

Let $\mathbf{z}_t = [z_{1,t}, z_{2,t}, z_{3,t}]^T$ be the pre-activation output vector. Then the forward propagation equations will be

$$\begin{aligned} z_{1,t} &= w_{1,1}a_{1,t-1} + w_{1,2}a_{2,t-1} + w_{1,3}a_{3,t-1} + u_{1,1}x_{1,t} + u_{1,2}x_{2,t} + b_1 \\ z_{2,t} &= w_{2,1}a_{1,t-1} + w_{2,2}a_{2,t-1} + w_{2,3}a_{3,t-1} + u_{2,1}x_{1,t} + u_{2,2}x_{2,t} + b_2 \\ z_{3,t} &= w_{3,1}a_{1,t-1} + w_{3,2}a_{2,t-1} + w_{3,3}a_{3,t-1} + u_{3,1}x_{1,t} + u_{3,2}x_{2,t} + b_3 \end{aligned} \tag{2.1}$$

In vector form, it will be writen as

$$\mathbf{z}_t = \mathbf{W}\mathbf{a}_{t-1} + \mathbf{U}\mathbf{x}_t + \mathbf{b} \tag{2.2}$$

where, $\mathbf{x}_t = [x_{1,t}, x_{2,t}]^T$ and $\mathbf{a}_{t-1} = [a_{1,t-1}, a_{2,t-1}, a3, t-1]^T$.

$\mathbf{W} \in \mathbb{R}^{N^{(1)} \times N^{(1)}}$, $\mathbf{U} \in \mathbb{R}^{N^{(1)} \times n}$ and $\mathbf{b} \in \mathbb{R}^{N^{(1)} \times 1}$ are the layer parameters.

Now, the Equation 2.2 will be passed through the activation function $h(.)$ to induce non-linearity and the final output obtained from this layer will be

$$\mathbf{a}_t = h\left(\mathbf{z}_t\right). \tag{2.3}$$

2.2 Layer 2 equations

Equation 2.3 gives the output vector of previous layer, which will be taken as the input vector to the current layer as it happens in regular neural networks.

Let, $\mathbf{o}_t \in \mathbb{R}^{N^{(2)}}$ be the pre-activation output vector for the current layer. Then the forward propagation equation will be

$$\mathbf{o}_t = \mathbf{V}\mathbf{a}_t + \mathbf{c} \tag{2.4}$$

Here, $\mathbf{V} \in \mathbb{R}^{N^{(2)} \times N^{(1)}}$ and $\mathbf{c} \in \mathbb{R}^{N^{(2)} \times 1}$ are the layer parameters.

Then, the activation function will be applied to $\mathbf{o}_t$. Here, for this derivation, the activation function is assumed to be softmax as the primary motive of this work is to develop RNN from scratch for autocompletion of words. So, the input and output will be a word-embedding vector of corresponding characters in the words.

The final output vector of current layer will be

$$\hat{\mathbf{y}}_t = softmax(\mathbf{o}_t) \tag{2.5}$$

2.3 Loss calculation

For the case of using softmax activation (which is used for multiclass classification problem), the apropriate loss function will be categorical cross entropy

$$L_t = -\mathbf{y}_t^T \log(\hat{\mathbf{y}}_t). \tag{2.6}$$

It can be noted in Equation 2.6 that the loss is calculated sequence-wise, i.e. 1 loss per t. So, the total loss will be

$$L = \sum_{t=1}^{T} L_t. \tag{2.7}$$

Next, we will see how to calculate the derivatives of Equation 2.6 with respect to all the RNN model parameters: $\mathbf{W}, \mathbf{U}, \mathbf{b}, \mathbf{V}$ and $\mathbf{c}$.

3 Back Propagation Through Time

Here, we will start from backwards, from last variable in output layer to first variable in input layer.

We will start with derivative of L_t with $\mathbf{o}_t$. It is given as

$$\frac{\partial L_t}{\partial \mathbf{o}_t} = \hat{\mathbf{y}}_t - \mathbf{y}_t. \tag{3.1}$$

The derivation of how Equation 3.1 came is explained in an execellent way here.

For dimensional compatiblity in later stages, I am going rewrite above vector equation in matrix form as

$$\frac{\partial L_t}{\partial \mathbf{o}_t} = \delta_t^{(2)} = diag(\hat{\mathbf{y}}_t - \mathbf{y}_t). \tag{3.2}$$

The Equation 3.2 yield a loss gradient matrix $\delta_t^{(2)} = \partial L_t / \partial \mathbf{o}_t$ of size $N^{(2)} \times N^{(2)}$, which in our example network (Figure 1.1) will be a 2×2 matrix as shown below.

$$\delta_t^{(2)} = \begin{bmatrix} \hat{y}_{1,t} - y_{1,t} & 0 \\ 0 & \hat{y}_{2,t} - y_{2,t} \end{bmatrix}$$

In all the below derivations, the explicit matrix/tensor forms will be writen with dimensions considering the example RNN network shown in Figure 1.1.

3.1 Computing loss gradients of layer 2 parameters

Now, to compute loss gradient with respect to $\mathbf{c}$ by taking Equation 2.4

$$\begin{aligned} \frac{\partial L_t}{\partial \mathbf{c}} &= \frac{\partial L_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{c}} \\ &= \delta_t^{(2)} \begin{bmatrix} \frac{\partial o_{1,t}}{\partial c_1} & \frac{\partial o_{1,t}}{\partial c_2} \\ \frac{\partial o_{2,t}}{\partial c_1} & \frac{\partial o_{2,t}}{\partial c_2} \end{bmatrix} = \delta_t^{(2)} \mathbf{I}_{N^{(2)}} = \delta_t^{(2)} \end{aligned} \tag{3.3}$$

It should be noted that in Equation 3.3, the $\frac{\partial \mathbf{o}_t}{\partial \mathbf{c}}$ is a vector-to-vector gradient resulting in a matrix as shown in the above steps.

> **Note**
>
> $\frac{\partial L_t}{\partial \mathbf{c}}$ is a matrix of dimension $N^{(2)} \times N^{(2)}$, with 1st dimension being number of components in the loss term.

Then in similar way, computing loss gradient with respect to $\mathbf{V}$ by taking the same Equation 2.4 as

$$\frac{\partial L_t}{\partial \mathbf{V}} = \frac{\partial L_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{V}} \tag{3.4}$$

In the above equation, we need to compute $\frac{\partial \mathbf{o}_t}{\partial \mathbf{V}}$ which is a vector-to-matrix gradient resulting in a 3D tensor as follows.

$$\begin{aligned} \frac{\partial \mathbf{o}_t}{\partial \mathbf{V}} &= \left[\left[\frac{\partial o_{1,t}}{\partial \mathbf{V}} \right] \left[\frac{\partial o_{2,t}}{\partial \mathbf{V}} \right] \right] \\ &= \left[\begin{bmatrix} a_{1,t} & a_{2,t} & a_{3,t} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ a_{1,t} & a_{2,t} & a_{3,t} \end{bmatrix} \right] \\ &= reshape \left(\mathbf{a}_t^T \otimes \mathbf{I}_{N^{(2)}},\ N^{(2)} \times N^{(2)} \times N^{(1)} \right) \end{aligned} \tag{3.5}$$

Substituting Equation 3.5 in Equation 3.4 will yield

$$\frac{\partial L_t}{\partial \mathbf{V}} = \delta_t^{(2)} reshape \left(\mathbf{a}_t^T \otimes \mathbf{I}_{N^{(2)}},\ N^{(2)} \times N^{(2)} \times N^{(1)} \right).$$

Here, $\otimes$ is the Kronecker product that gives $\mathbf{a}_t^T \otimes \mathbf{I}_{N^{(2)}}$ as a 2D matrix of $N^{(2)} \times (N^{(2)} * N^{(1)})$ dimension. And $reshape\,(,\ )$ is an tensor reshaping operator that transforms the 2D matrix of shape $N^{(2)} \times (N^{(2)} * N^{(1)})$ into a 3D Tensor of shape $N^{(2)} \times N^{(2)} \times N^{(1)}$.

But, $\delta_t^{(2)}$ is a matrix of shape $N^{(2)} \times N^{(2)}$ and the other operand is a 3D tensor in the above equation. So, again $reshape\,(,\ )$ operation is performed to convert the tensor into matrix, perform multiplication and then again convert back the resulting matrix into tensor as shown below.

$$\frac{\partial L_t}{\partial \mathbf{V}} = reshape \left(\delta_t^{(2)} reshape \left(\frac{\partial \mathbf{o}_t}{\partial \mathbf{V}},\ N^{(2)} \times (N^{(2)} * N^{(1)}) \right),\ N^{(2)} \times N^{(2)} \times N^{(1)} \right) \tag{3.6}$$

> **Note**
>
> $\frac{\partial L_t}{\partial \mathbf{V}}$ is a 3D tensor of dimension $N^{(2)} \times N^{(2)} \times N^{(1)}$, with 1^{st} dimension being number of components in the loss term.

This concludes the gradients derivation for layer 2. Next, similar steps will be performed for gradeient computation for layer 1.

3.2 Computing loss gradients of layer 1 parameters

To start with layer 1 parameters, we need a couple of intermediate derivatives that are computed below.

$$\begin{aligned}
\frac{\partial L_t}{\partial \mathbf{a}_t} &= \frac{\partial L_t}{\partial \mathbf{o}_t} \frac{\partial \mathbf{o}_t}{\partial \mathbf{a}_t} \\
&= \delta_t^{(2)} \begin{bmatrix} \frac{\partial o_{1,t}}{\partial a_{1,t}} & \frac{\partial o_{1,t}}{\partial a_{2,t}} & \frac{\partial o_{1,t}}{\partial a_{3,t}} \\ \frac{\partial o_{2,t}}{\partial a_{1,t}} & \frac{\partial o_{2,t}}{\partial a_{2,t}} & \frac{\partial o_{2,t}}{\partial a_{3,t}} \end{bmatrix} \\
&= \delta_t^{(2)} \begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,3} \\ v_{2,1} & v_{2,2} & v_{3,3} \end{bmatrix} \\
&= \delta_T^{(2)} \mathbf{V}
\end{aligned} \tag{3.7}$$

And

$$\begin{aligned}
\frac{\partial L_t}{\partial \mathbf{z}_t} &= \frac{\partial L_t}{\partial \mathbf{a}_t} \frac{\partial \mathbf{a}_t}{\partial \mathbf{z}_t} \\
&= \frac{\partial L_t}{\partial \mathbf{a}_t} \begin{bmatrix} \frac{\partial a_{1,t}}{\partial z_{1,t}} & \frac{\partial a_{1,t}}{\partial z_{2,t}} & \frac{\partial a_{1,t}}{\partial z_{3,t}} \\ \frac{\partial a_{2,t}}{\partial z_{1,t}} & \frac{\partial a_{2,t}}{\partial z_{2,t}} & \frac{\partial a_{2,t}}{\partial z_{3,t}} \\ \frac{\partial a_{3,t}}{\partial z_{1,t}} & \frac{\partial a_{3,t}}{\partial z_{2,t}} & \frac{\partial a_{3,t}}{\partial z_{3,t}} \end{bmatrix} \\
&= \frac{\partial L_t}{\partial \mathbf{a}_t} \begin{bmatrix} h'(z_{1,t}) & 0 & 0 \\ 0 & h'(z_{1,t}) & 0 \\ 0 & 0 & h'(z_{1,t}) \end{bmatrix} \\
&= \frac{\partial L_t}{\partial \mathbf{a}_t} diag(h'(\mathbf{z}_t)) \\
&= \delta_t^{(1)}
\end{aligned} \tag{3.8}$$

> **Note**
>
> $\delta_t^{(1)}$ is a matrix of size $N^{(2)} \times N^{(1)}$.

Now, we will start with the $\mathbf{b}$ parameter's loss gradient.

$$\frac{\partial L_t}{\partial \mathbf{b}} = \frac{\partial L_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{b}} \tag{3.9}$$

It is evident from Equation 2.2 and Equation 2.3 that $\mathbf{a}_t$ is dependent on $\mathbf{z}_t$. However, $\mathbf{z}_t$ is dependent upon $\mathbf{a}_{t-1}$, and the chain goes on till $t = 0$. Let me call this as *sequence chain link*. Because of this, the $\mathbf{z}_t$ derivative in above equation will be written as

$$\begin{aligned}\frac{\partial \mathbf{z}_t}{\partial \mathbf{b}} &= \frac{\partial \mathbf{b}}{\partial \mathbf{b}} + \mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{b}} \\ &= \mathbf{I}_{N^{(1)}} + \mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{b}}\end{aligned}$$

We will keep $\partial \mathbf{a}_{t-1}/\partial \mathbf{b}$ as it is for now. Substituting above expression in Equation 3.9 gives

$$\frac{\partial L_t}{\partial \mathbf{b}} = \delta_t^{(1)}\left[\mathbf{I}_{N^{(1)}} + \mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{b}}\right]. \tag{3.10}$$

> **Note**
>
> $\frac{\partial L_t}{\partial \mathbf{b}}$ is a vector of shape $N^{(2)} \times N^{(1)}$.

Next, we compute

$$\begin{aligned}\frac{\partial \mathbf{a}_t}{\partial \mathbf{b}} &= \frac{\partial \mathbf{a}_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{b}} \\ &= diag(h'(\mathbf{z}_t))\mathbf{I}_{N^{(1)}} \\ &= diag(h'(\mathbf{z}_t))\end{aligned} \tag{3.11}$$

which is a $N^{(1)} \times N^{(1)}$ matrix.

Now, we move to compute $\mathbf{W}$ parameter's loss gradient. From Equation 2.2,

$$\begin{aligned}\frac{\partial L_t}{\partial \mathbf{W}} &= \frac{\partial L_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{W}} \\ &= \delta_t^{(1)}\left[\frac{\partial \mathbf{W}\mathbf{a}_{t-1}}{\partial \mathbf{W}} + \mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{W}}\right] \\ &= \delta_t^{(1)}\omega_1.\end{aligned} \tag{3.12}$$

Lets compute each term in ω_1 in above equation. Let

$$\lambda = \mathbf{W}\mathbf{a}_{t-1} = \begin{bmatrix} w_{1,1}a_{1,t-1} + w_{1,2}a_{2,t-1} + w_{1,3}a_{3,t-1} \\ w_{2,1}a_{1,t-1} + w_{2,2}a_{2,t-1} + w_{2,3}a_{3,t-1} \\ w_{3,1}a_{1,t-1} + w_{3,2}a_{2,t-1} + w_{3,3}a_{3,t-1} \end{bmatrix}$$

Now,

$$\begin{aligned} \frac{\partial \mathbf{W}\mathbf{a}_{t-1}}{\partial \mathbf{W}} = \frac{\partial \lambda}{\partial \mathbf{W}} &= \left[\left[\frac{\partial \lambda_1}{\partial \mathbf{W}} \right] \left[\frac{\partial \lambda_2}{\partial \mathbf{W}} \right] \left[\frac{\partial \lambda_3}{\partial \mathbf{W}} \right] \right] \\ &= \left[\begin{bmatrix} a_{1,t-1} & a_{2,t-1} & a_{3,t-1} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ a_{1,t-1} & a_{2,t-1} & a_{3,t-1} \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ a_{1,t-1} & a_{2,t-1} & a_{3,t-1} \end{bmatrix} \right] \\ \frac{\partial \mathbf{W}\mathbf{a}_{t-1}}{\partial \mathbf{W}} &= reshape\left(\mathbf{a}_{t-1}^T \otimes \mathbf{I}_{N^{(1)}},\ N^{(1)} \times N^{(1)} \times N^{(1)} \right). \end{aligned} \tag{3.13}$$

We keep $\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{W}}$ as it is for now in Equation 3.12 like previously did. Then,

$$\mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{W}} = reshape\left(\mathbf{W}\ reshape\left(\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{W}},\ N^{(1)} \times (N^{(1)} * N^{(1)}) \right),\ N^{(1)} \times N^{(1)} \times N^{(1)} \right). \tag{3.14}$$

Substituting Equation 3.13 and Equation 3.14 in Equation 3.12 can compute ω_1 term. Thus,

$$\frac{\partial L_t}{\partial \mathbf{W}} = reshape\left(\delta_t^{(1)}\ reshape\left(\omega_1,\ N^{(1)} \times (N^{(1)} * N^{(1)}) \right),\ N^{(2)} \times N^{(1)} \times N^{(1)} \right). \tag{3.15}$$

> **Note**
>
> $\frac{\partial L_t}{\partial \mathbf{W}}$ is a 3D tensor of shape $N^{(2)} \times N^{(1)} \times N^{(1)}$.

And computing the needed gradient for the next timestep

$$\frac{\partial \mathbf{a}_t}{\partial \mathbf{W}} = diag(h'(\mathbf{z}_t))\omega_1.$$

Now, computing the last parameter $\mathbf{U}$ derivative of loss function

$$\begin{aligned} \frac{\partial L_t}{\partial \mathbf{U}} &= \frac{\partial L_t}{\partial \mathbf{z}_t}\frac{\partial \mathbf{z}_t}{\partial \mathbf{U}} \\ &= \delta_t^{(1)}\left[\frac{\partial \mathbf{U}\mathbf{x}_t}{\partial \mathbf{U}} + \mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{U}} \right] \\ &= \delta_t^{(2)}\omega_2. \end{aligned} \tag{3.16}$$

Now to compute ω_2 in above equation. Let

$$\psi = \mathbf{U}\mathbf{x}_t = \begin{bmatrix} u_{1,1}x_{1,t} + u_{1,2}x_{2,t} \\ u_{2,1}x_{1,t} + u_{2,2}x_{2,t} \\ u_{3,1}x_{1,t} + u_{3,2}x_{2,t} \end{bmatrix}.$$

Then

$$\begin{aligned} \frac{\partial \mathbf{U}\mathbf{x}_t}{\partial \mathbf{U}} = \frac{\partial \psi}{\partial \mathbf{U}} &= \left[\left[\frac{\partial \psi_1}{\partial \mathbf{U}} \right] \left[\frac{\partial \psi_2}{\partial \mathbf{U}} \right] \left[\frac{\partial \psi_3}{\partial \mathbf{U}} \right] \right] \\ &= \left[\begin{bmatrix} x_{1,t} & x_{2,t} \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ x_{1,t} & x_{2,t} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ x_{1,t} & x_{2,t} \end{bmatrix} \right] \\ \frac{\partial \mathbf{U}\mathbf{x}_t}{\partial \mathbf{U}} &= reshape\left(\mathbf{x}_t^T \otimes \mathbf{I}_{N^{(1)}},\ N^{(1)} \times N^{(1)} \times n \right). \end{aligned} \tag{3.17}$$

Now, in the other term, we will keep $\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{U}}$ as it is for now like previously did. Then,

$$\mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{U}} = reshape\left(\mathbf{W} reshape\left(\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{U}},\ N^{(1)} \times (N^{(1)} * n) \right),\ N^{(1)} \times N^{(1)} \times n \right) \tag{3.18}$$

Now, by substituting Equation 3.17 and Equation 3.18 in Equation 3.16, we can compute ω_2. Thus,

$$\frac{\partial L_t}{\partial \mathbf{U}} = reshape\left(\delta_t^{(1)} reshape\left(\omega_2,\ N^{(1)} \times (N^{(1)} * n) \right),\ N^{(2)} \times N^{(1)} \times n \right) \tag{3.19}$$

> **Note**
>
> $\frac{\partial L_t}{\partial \mathbf{U}}$ is a 3D tensor of size $N^{(2)} \times N^{(1)} \times n$.

And computing the needed gradient for the next timestep

$$\frac{\partial \mathbf{a}_t}{\partial \mathbf{U}} = diag(h'(\mathbf{z}_t))\omega_2$$

This concludes the loss equation derivatives computation for all model parameters.

3.3 Summary of loss gradients

Below is the summary of the loss gradient equations for all parameters.

$$
\begin{aligned}
\frac{\partial L_t}{\partial \mathbf{c}} &= \delta_t^{(2)} \\
\frac{\partial L_t}{\partial \mathbf{V}} &= reshape\left(\delta_t^{(2)}\ reshape\left(\frac{\partial \mathbf{o}_t}{\partial \mathbf{V}},\ N^{(2)} \times (N^{(2)} * N^{(1)})\right),\ N^{(2)} \times N^{(2)} \times N^{(1)}\right) \\
\frac{\partial L_t}{\partial \mathbf{b}} &= \delta_t^{(1)}\left[\mathbf{I}_{N^{(1)}} + \mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{b}}\right] \\
\frac{\partial L_t}{\partial \mathbf{W}} &= reshape\left(\delta_t^{(1)}\ reshape\left(\omega_1,\ N^{(1)} \times (N^{(1)} * N^{(1)})\right),\ N^{(2)} \times N^{(1)} \times N^{(1)}\right) \\
\frac{\partial L_t}{\partial \mathbf{U}} &= reshape\left(\delta_t^{(1)} reshape\left(\omega_2,\ N^{(1)} \times (N^{(1)} * n)\right),\ N^{(2)} \times N^{(1)} \times n\right)
\end{aligned}
$$

In the next section, we will discuss about updating model parameters and the pseudocode for RNN training.

4 Updating parameters and RNN training

In the last section, we have derived the loss gradients for all model parameters. It can be noted that the loss gradients have an extra dimension than its corresponding model parameters.

For example,
the dimension of $\frac{\partial L_t}{\partial \mathbf{U}}$ is $\Big(N^{(2)} \times N^{(1)} \times n\Big)$ whereas the dimension of $\mathbf{U}$ is $\Big(N^{(1)} \times n\Big)$.

This leading dimension $N^{(2)}$ indicates the number of components in the final output vector. Thus, **every parameter has a loss gradient derived from every output vector component.** So, the loss gradients will be summed on the first axis (loss axis) to accumulate the update from all loss components and then used to update the parameters.

Hence, the update from each loss component will be accumulated by adding the loss gradients on the first axis and then used to update the parameters.

4.1 Updating parameters

The parameters update equations are given below. The summation is performed on the loss axis direction.

$$\mathbf{c}_i := \mathbf{c}_i - \alpha \sum_{t=1}^{N^{(2)}} \sum_{k=1}^{N^{(2)}} \frac{\partial L_t}{\partial \mathbf{c}}\bigg|_{k,i}, \quad i = 1, 2, \ldots, N^{(2)}$$

$$\mathbf{V}_{i,j} := \mathbf{V}_{i,j} - \alpha \sum_{t=1}^{N^{(2)}} \sum_{k=1}^{N^{(2)}} \frac{\partial L_t}{\partial \mathbf{V}}\bigg|_{k,i,j}, \quad i = 1, 2, \ldots, N^{(2)}, \quad j = 1, 2, \ldots, N^{(1)}$$

$$\mathbf{b}_i := \mathbf{b}_i - \alpha \sum_{t=1}^{N^{(2)}} \sum_{k=1}^{N^{(2)}} \frac{\partial L_t}{\partial \mathbf{b}}\bigg|_{k,i}, \quad i = 1, 2, \ldots, N^{(1)}$$

$$\mathbf{W}_{i,j} := \mathbf{W}_{i,j} - \alpha \sum_{t=1}^{N^{(2)}} \sum_{k=1}^{N^{(2)}} \frac{\partial L_t}{\partial \mathbf{W}}\bigg|_{k,i,j}, \quad i = 1, 2, \ldots, N^{(1)}, \quad j = 1, 2, \ldots, N^{(1)}$$

$$\mathbf{U}_{i,j} := \mathbf{U}_{i,j} - \alpha \sum_{t=1}^{N^{(2)}} \sum_{k=1}^{N^{(2)}} \frac{\partial L_t}{\partial \mathbf{U}}\bigg|_{k,i,j}, \quad i = 1, 2, \ldots, N^{(1)}, \quad j = 1, 2, \ldots, n$$

Here, α is the learning rate.

> **Note**
>
> The above equations are updating parameters based on one sub-part of data only! Hence, the algorithms like Batch Gradient Descent and their variants (refer here) will involve additional summation over the batch of data being used to update the parameters in the above equations.

4.2 Training RNNs

We now have the loss gradients equations in hand. In this section we will see on how to use all these derived equations to Train an RNN.

First, the data will be loaded and then encoding will be performed. Here, encoding will be needed as the input data will be a word (a sequence of characters). Encoding may be skipped for numerical data sequence.

The size of input and output vectors will be determined during the encoding step. Thus, the number of hidden neurons will be have to be chosen along with learning rate.

We have to have a set of hidden vector and its derivatives at $t = 0$ for the algorithm to work. Hence we have initialized the hidden vector and tensor variables with zeros. Then we have to initialize the network parameters with appropriate techniques such as Xavier Initialization.

Then, the training loop starts. We will perform forward propagation to get the initial prediction of output vector $\hat{\mathbf{y}}$ which is then used to compute loss value. Then, the $\hat{\mathbf{y}}$ will be taken to back-propagation through time along with other network parameters for loss gradients computation. Finally, the network parameters will be updated with the computed loss gradients.

The training loop will be continued until the model converges. The usual convergence criteria will be to set a threshold on loss value and the model is taken as converged/trained when loss goes below threshold.

The algorithm for RNN training is given below (click on the image for enlarged view).

Algorithm RNN training

load dataset $\{x_t, y_t\},\ t = 1, 2, \ldots, T$

Perform encoding on $\{x_t\}$ ▷ word embedding

Choose $N^{(1)}$ ▷ hidden neurons count

Choose α ▷ learning rate

initialize $a_0 = \{0\}$, $\frac{\partial a_0}{\partial W} = \{0\}$ ▷ initializing 0^{th} hidden vector/tensor values

initialize $\frac{\partial a_0}{\partial U} = \{0\}$, $\frac{\partial a_0}{\partial b} = \{0\}$

randomly initialize W,U,b,V and c ▷ initializing model parameters

repeat

Call Forward Propagation ( )

Call BPTT ( )

Call Parameter Update ( )

until Convergence

Each main step in the above algorithm is defined as a function. The forward propagation function is given below.

function Forward Propagation ( )

for $t = 1, 2, \ldots, T$ **do**

$z_t = Wa_{t-1} + Ux_t + b$ ▷ recurrent layer equations

$a_t = h(z_t)$

$o_t = Va_t + c$ ▷ output layer equations

$\hat{y}_t = softmax(o_t)$

end for

Perform decoding on $\{\hat{y}_t\}$ ▷ For validation

end function

Here, the decoding step in the end of forward propagation is optional and will be used only during model inference. Hence, that step can be skipped.

Then the control in algorithm then goes to back-propagation through time function which is defined below.

function BPTT( ) ▷ Back Propagation Through Time

for $t = 1, 2, \ldots, T$ **do**

Compute the following in the order

$$
\begin{aligned}
\delta_t^{(2)} &= diag(\hat{\mathbf{y}}_t - \mathbf{y}_t) \\
\frac{\partial L_t}{\partial \mathbf{c}} &= \delta_t^{(2)} \\
\frac{\partial \mathbf{o}_t}{\partial V} &= reshape\left(\mathbf{a}_t^T \otimes \mathbf{I}_{N^{(2)}},\ N^{(2)} \times N^{(2)} \times N^{(1)}\right) \\
\frac{\partial L_t}{\partial V} &= reshape\left(\delta_t^{(2)}\ reshape\left(\frac{\partial \mathbf{o}_t}{\partial \mathbf{V}},\ N^{(2)} \times \left(N^{(2)} * N^{(1)}\right)\right),\ N^{(2)} \times N^{(2)} \times N^{(1)}\right) \\
\frac{\partial L_t}{\partial \mathbf{a}_t} &= \delta_t^{(2)} \mathbf{V} \\
\delta_t^{(1)} &= \frac{\partial L_t}{\partial \mathbf{a}_t}\ diag(h'(\mathbf{z}_t)) \\
\frac{\partial L_t}{\partial \mathbf{b}} &= \delta_t^{(1)} \left[\mathbf{I}_{N^{(1)}} + \mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{b}}\right] \\
\frac{\partial \mathbf{a}_t}{\partial \mathbf{b}} &= diag(h'(\mathbf{z}_t)) \\
\frac{\partial \mathbf{W}\mathbf{a}_{t-1}}{\partial \mathbf{W}} &= reshape\left(\mathbf{a}_{t-1}^T \otimes \mathbf{I}_{N^{(1)}},\ N^{(1)} \times N^{(1)} \times N^{(1)}\right) \\
\mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{W}} &= reshape\left(\mathbf{W}\ reshape\left(\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{W}},\ N^{(1)} \times \left(N^{(1)} * N^{(1)}\right)\right),\ N^{(1)} \times N^{(1)} \times N^{(1)}\right) \\
\omega_1 &= \frac{\partial \mathbf{W}\mathbf{a}_{t-1}}{\partial \mathbf{W}} + \mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{W}} \\
\frac{\partial L_t}{\partial \mathbf{W}} &= reshape\left(\delta_t^{(1)}\ reshape\left(\omega_1,\ N^{(1)} \times \left(N^{(1)} * N^{(1)}\right)\right),\ N^{(2)} \times N^{(1)} \times N^{(1)}\right) \\
\frac{\partial \mathbf{a}_t}{\partial \mathbf{W}} &= diag(h'\left(\mathbf{z}_t\right))\omega_1 \\
\frac{\partial \mathbf{U}\mathbf{x}_t}{\partial \mathbf{U}} &= reshape\left(\mathbf{x}_t^T \otimes \mathbf{I}_{N^{(1)}},\ N^{(1)} \times N^{(1)} \times n\right) \\
\mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{U}} &= reshape\left(\mathbf{W}\ reshape\left(\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{U}},\ N^{(1)} \times \left(N^{(1)} * n\right)\right),\ N^{(1)} \times N^{(1)} \times n\right) \\
\omega_2 &= \frac{\partial \mathbf{U}\mathbf{x}_t}{\partial \mathbf{U}} + \mathbf{W}\frac{\partial \mathbf{a}_{t-1}}{\partial \mathbf{U}} \\
\frac{\partial L_t}{\partial \mathbf{U}} &= reshape\left(\delta_t^{(1)}\ reshape\left(\omega_2,\ N^{(1)} \times \left(N^{(1)} * n\right)\right),\ N^{(2)} \times N^{(1)} \times n\right) \\
\frac{\partial \mathbf{a}_t}{\partial \mathbf{U}} &= diag(h'(\mathbf{z}_t))\omega_2
\end{aligned}
$$

end for

end function

Here, the loss parameter gradient equations are given in the order of execution so that the dependency of variables is satisfied.

Once back-propagation is done and the gradients are available, the control then moves to the parameter update function that is defined below.

---

function PARAMETER UPDATE( )

$$\mathbf{W}_{i,j} := \mathbf{W}_{i,j} - \alpha \sum_{t=1}^{T} \sum_{k=1}^{N^{(2)}} \left(\frac{\partial L_t}{\partial \mathbf{W}} \right)_{k,i,j}, \; i = 1, 2, \ldots, N^{(1)}, \; j = 1, 2, \ldots, N^{(1)}$$

$$\mathbf{U}_{i,j} := \mathbf{U}_{i,j} - \alpha \sum_{t=1}^{T} \sum_{k=1}^{N^{(2)}} \left(\frac{\partial L_t}{\partial \mathbf{U}} \right)_{k,i,j}, \; i = 1, 2, \ldots, N^{(1)}, \; j = 1, 2, \ldots, n$$

$$\mathbf{b}_i := \mathbf{b}_i - \alpha \sum_{t=1}^{T} \sum_{k=1}^{N^{(2)}} \left(\frac{\partial L_t}{\partial \mathbf{b}} \right)_{k,i}, \; i = 1, 2, \ldots, N^{(1)}$$

$$\mathbf{V}_{i,j} := \mathbf{V}_{i,j} - \alpha \sum_{t=1}^{T} \sum_{k=1}^{N^{(2)}} \left(\frac{\partial L_t}{\partial \mathbf{V}} \right)_{k,i,j}, \; i = 1, 2, \ldots, N^{(2)}, \; j = 1, 2, \ldots, N^{(1)}$$

$$\mathbf{c}_i := \mathbf{c}_i - \alpha \sum_{t=1}^{T} \sum_{k=1}^{N^{(2)}} \left(\frac{\partial L_t}{\partial \mathbf{c}} \right)_{k,i}, \; i = 1, 2, \ldots, N^{(2)}$$

end function

And then, the loop continues until the convergence criteria is met. This concludes the RNN training algorithm.

5 Summary

In this work, the back-propagation through time procedure and the appropriate loss gradient equations were derived for the Recurrent Neural Networks.

- Here, the derivation was made with the example RNN that has 3 hidden neurons and 2 as size of input/output vectors to confirm the dimensional consistency between the vector/matrix/tensors in the equations.
- The derived loss gradient equations are generalized for an RNN with $N^{(1)}$ hidden neurons, n input vector size and $N^{(2)}$ output vector size. Here, the RNN with only two layers is used for the development. But the procdure is straightforward for extending it to deep RNNs.
- Here, the derived loss gradients will have an extra dimension specified as the first dimension in the matrix/tensor ($N^{(2)}$ in $(N^{(2)} \times N^{(1)} \times n$ for example). This dimension is the dimension of the loss vector called as loss axis. That is, there will be one gradient per loss component for each parameter. Thus, summing along the loss axis will be performed before updating the model parameters to get contribution from all output vector components for the update.
- And in the last section, the training algorithm for RNNs was given.

This derivation was made as a part of the course work. Later, these equations and the training algorithm will be used to solve simple problems and the link to their documentations will be updated here soon.