

vanilla Recurrent Neural-Network: back-propagation derivation

Harsha Vardhan

May 5, 2022

Abstract

This document contains derivation of the gradients for a vanilla recurrent neural-network, using back-propagation (or reverse-mode differentiation). For the implementation of the neural-network see the accompanying notebooks.

1 Network Architecture

A Recurrent Neural-Network can take many different architectures, and here we are considering a common many-to-many architecture. Also, for a brief description of other common RNN architectures, see section-4.

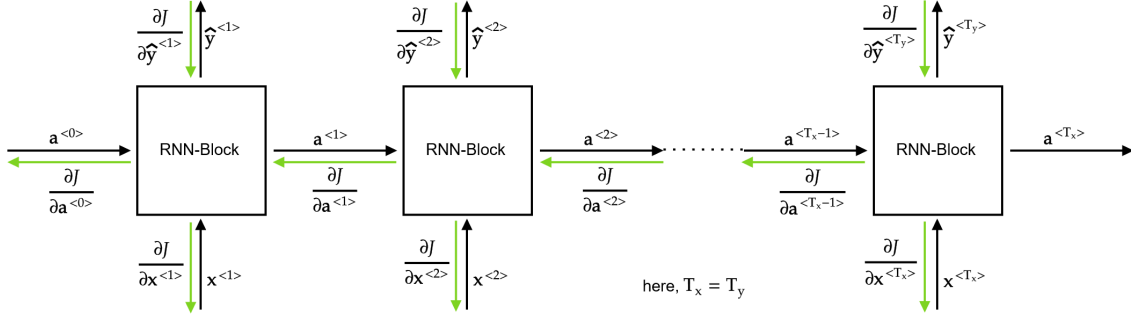


Figure 1: a recurrent neural network with many-to-many architecture, such that $T_x = T_y$. Where, T_x is the number of input time-steps, and T_y is the number of output time-steps. Also, here 'RNN-Block' would be the block for vanilla RNN; see section-1.1 for more information.

1.1 vanilla RNN Block

A RNN-block could be of many types, such as the vanilla-block, LSTM-block, GRU-block, etc. In this section we define the vanilla RNN-Block, since in this article we are dealing with a vanilla RNN.

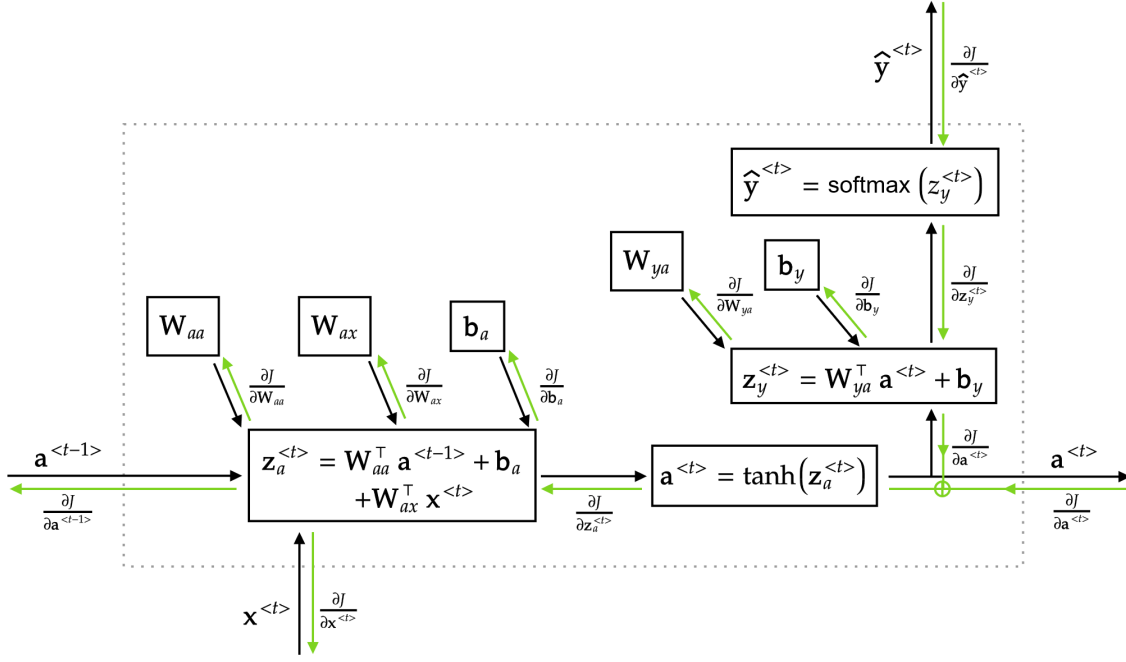


Figure 2: a vanilla RNN-block along with the back-propagation routes. Note that the gradient $\frac{\partial J}{\partial \mathbf{a}^{(t)}}$ come from two directions - the $(t+1)^{th}$ time-step and from t^{th} time-steps' output - both of which get added.

2 Forward Propagation

Given $\mathbf{a}^{(t-1)}$ from the $(t-1)^{th}$ time-step, the equations for forward propagation through the t^{th} time-step, are as follows:

$$\begin{aligned} \mathbf{z}_a^{(t)} &= \mathbf{W}_{aa}^\top \mathbf{a}^{(t-1)} + \mathbf{W}_{ax}^\top \mathbf{x}^{(t)} + \mathbf{b}_a \\ \mathbf{a}^{(t)} &= \tanh(\mathbf{z}_a^{(t)}) \\ \mathbf{z}_y^{(t)} &= \mathbf{W}_{ya}^\top \mathbf{a}^{(t)} + \mathbf{b}_y \\ \hat{\mathbf{y}}^{(t)} &= \text{softmax}(\mathbf{z}_y^{(t)}) \end{aligned}$$

where,

- $\mathbf{x}^{(t)}$ a $(n_x, 1)$ dimensional input-vector, at the t^{th} time-step
- \mathbf{W}_{ax} a (n_x, n_a) dimensional weights-matrix
- \mathbf{W}_{aa} a (n_a, n_a) dimensional weights-matrix
- \mathbf{W}_{ya} a (n_a, n_y) dimensional weights-matrix
- \mathbf{b}_a a $(n_a, 1)$ dimensional bias-vector
- \mathbf{b}_y a $(n_y, 1)$ dimensional bias-vector

Notation: in the above equations, the sub-script for matrices and vectors must be interpreted as follows:

- Let \mathbf{A}_{pq} be a matrix. Then the sub-script 'p' denotes that the matrix is used for computing some p-like quantity, and the sub-script 'q' denotes that the matrix is multiplied by some q-like quantity.

- Let \mathbf{a}_q be a vector. Then the sub-script 'q' denotes that the vector is used for computing some q-like quantity.

3 Optimization: gradient-descent

The optimization is performed according to the following equations:

$$\begin{aligned}\mathbf{W}_{aa} &:= \mathbf{W}_{aa} - \alpha \sum_{t=1}^{T_x} \nabla_{\mathbf{W}_{aa}}^{(t)} J \\ \mathbf{W}_{ax} &:= \mathbf{W}_{ax} - \alpha \sum_{t=1}^{T_x} \nabla_{\mathbf{W}_{ax}}^{(t)} J \\ \mathbf{W}_{ya} &:= \mathbf{W}_{ya} - \alpha \sum_{t=1}^{T_x} \nabla_{\mathbf{W}_{ya}}^{(t)} J \\ \mathbf{b}_a &:= \mathbf{b}_a - \alpha \sum_{t=1}^{T_x} \nabla_{\mathbf{b}_a}^{(t)} J \\ \mathbf{b}_y &:= \mathbf{b}_y - \alpha \sum_{t=1}^{T_x} \nabla_{\mathbf{b}_y}^{(t)} J\end{aligned}$$

Notice that we are summing the gradients over all the time-steps before updating the parameters.

3.1 Back-propagation

The gradients in the above equations, for any particular time-step t , are derived using back-propagation as follows:

3.1.1 Computing $\frac{\partial J}{\partial \hat{\mathbf{y}}^{(t)}}$

Since, $\hat{\mathbf{y}}^{(t)}$ is computed using the softmax() activation-function, the loss J is computed using the cross-entropy loss. Also, let $\mathbf{y}^{(t)}$ be the output-label corresponding to the t^{th} time-step. Then,

$$\begin{aligned}\frac{\partial J}{\partial \hat{\mathbf{y}}^{(t)}} &= \begin{bmatrix} \frac{\partial J}{\partial \hat{y}_1^{(t)}} & \frac{\partial J}{\partial \hat{y}_2^{(t)}} & \cdots & \frac{\partial J}{\partial \hat{y}_{n_y}^{(t)}} \end{bmatrix} \\ &= \begin{bmatrix} \frac{y_1^{(t)}}{\hat{y}_1^{(t)}} & \frac{y_2^{(t)}}{\hat{y}_2^{(t)}} & \cdots & \frac{y_{n_y}^{(t)}}{\hat{y}_{n_y}^{(t)}} \end{bmatrix}\end{aligned}$$

3.1.2 Computing $\frac{\partial J}{\partial \mathbf{z}_y^{(t)}}$

$$\frac{\partial J}{\partial \mathbf{z}_y^{(t)}} = \frac{\partial J}{\partial \hat{\mathbf{y}}^{(t)}} \frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{z}_y^{(t)}}$$

Note: For the derivative of the softmax() function, i.e. $\frac{\partial \hat{\mathbf{y}}^{(t)}}{\partial \mathbf{z}_y^{(t)}}$, see ..\notes\softmax-function.ipynb.

3.1.3 Computing $\frac{\partial J}{\partial \mathbf{W}_{ya}}$, $\frac{\partial J}{\partial \mathbf{b}_y}$, and $\frac{\partial J}{\partial \mathbf{a}^{(t)}}$

$$dJ = \text{tr} \left(\frac{\partial J}{\partial \mathbf{z}_y^{(t)}} d\mathbf{z}_y^{(t)} \right)$$

where, $d\mathbf{z}_y^{(t)}$ can be expanded as follows:

$$\begin{aligned} d\mathbf{z}_y^{(t)} &= d(\mathbf{W}_{ya}^\top \mathbf{a}^{(t)} + \mathbf{b}_y) \\ &= d\mathbf{W}_{ya}^\top \mathbf{a}^{(t)} + \mathbf{W}_{ya}^\top d\mathbf{a}^{(t)} + d\mathbf{b}_y \end{aligned}$$

when differentiating w.r.t. \mathbf{W}_{ya} , we have $d\mathbf{a}^{(t)} = 0$, and $d\mathbf{b}_y = 0$. So,

$$\begin{aligned} dJ &= \text{tr} \left(\frac{\partial J}{\partial \mathbf{z}_y^{(t)}} d\mathbf{W}_{ya}^\top \mathbf{a}^{(t)} \right) \\ &= \text{tr} \left((\mathbf{a}^{(t)})^\top d\mathbf{W}_{ya} \left(\frac{\partial J}{\partial \mathbf{z}_y^{(t)}} \right)^\top \right) \\ &= \text{tr} \left(\left(\frac{\partial J}{\partial \mathbf{z}_y^{(t)}} \right)^\top (\mathbf{a}^{(t)})^\top d\mathbf{W}_{ya} \right) \\ &\Rightarrow \frac{\partial J}{\partial \mathbf{W}_{ya}} = \left(\frac{\partial J}{\partial \mathbf{z}_y^{(t)}} \right)^\top (\mathbf{a}^{(t)})^\top \end{aligned}$$

when differentiating w.r.t. \mathbf{b}_y , we have $d\mathbf{a}^{(t)} = 0$, and $d\mathbf{W}_{ya} = 0$. So,

$$\begin{aligned} dJ &= \text{tr} \left(\frac{\partial J}{\partial \mathbf{z}_y^{(t)}} d\mathbf{b}_y \right) \\ &\Rightarrow \frac{\partial J}{\partial \mathbf{b}_y} = \frac{\partial J}{\partial \mathbf{z}_y^{(t)}} \end{aligned}$$

when differentiating w.r.t. $\mathbf{a}^{(t)}$, we have $d\mathbf{W}_{ya} = 0$, and $d\mathbf{b}_y = 0$. So,

$$\begin{aligned} dJ &= \text{tr} \left(\frac{\partial J}{\partial \mathbf{z}_y^{(t)}} \mathbf{W}_{ya}^\top d\mathbf{a}^{(t)} \right) \\ &\Rightarrow \frac{\partial J}{\partial \mathbf{a}^{(t)}} = \frac{\partial J}{\partial \mathbf{z}_y^{(t)}} \mathbf{W}_{ya}^\top \end{aligned}$$

3.1.4 Computing $\frac{\partial J}{\partial \mathbf{z}_a^{(t)}}$

$$\begin{aligned} \frac{\partial J}{\partial \mathbf{z}_a^{(t)}} &= \frac{\partial J}{\partial \mathbf{a}^{(t)}} \frac{\partial \mathbf{a}^{(t)}}{\partial \mathbf{z}_a^{(t)}} \\ &= \frac{\partial J}{\partial \mathbf{a}^{(t)}} \text{diag}(\mathbf{1}_{(n_a,1)} - \tanh(\mathbf{z}_a^{(t)})) \\ &= \frac{\partial J}{\partial \mathbf{a}^{(t)}} \circ (\mathbf{1}_{(n_a,1)} - \tanh(\mathbf{z}_a^{(t)}))^\top \end{aligned}$$

In the above expression, we have

- $\frac{\partial J}{\partial \mathbf{a}^{(t)}}$ is computed by taking the sum of the gradient propagation from $\hat{\mathbf{y}}^{(t)}$ with the gradient propagating from the $(t+1)^{th}$ time-step. Note that when $t = T_x$, the gradient from the $(t+1)^{th}$ time-step will be zero.
- $\frac{\partial \mathbf{a}^{(t)}}{\partial \mathbf{z}_a^{(t)}}$ is the derivative across the $\tanh()$ function. For the derivation of this derivative see `..\notes\hyperbolic-tangent-function.ipynb`.

3.1.5 Computing $\frac{\partial J}{\partial \mathbf{W}_{aa}}$, $\frac{\partial J}{\partial \mathbf{W}_{ax}}$, $\frac{\partial J}{\partial \mathbf{b}_a}$, and $\frac{\partial J}{\partial \mathbf{a}^{(t-1)}}$

$$dJ = \text{tr} \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} d\mathbf{z}_a^{(t)} \right)$$

where, $d\mathbf{z}_a^{(t)}$ can be expanded as follows:

$$\begin{aligned} d\mathbf{z}_a^{(t)} &= d(\mathbf{W}_{aa}^\top \mathbf{a}^{(t-1)} + \mathbf{W}_{ax}^\top \mathbf{x}^{(t)} + \mathbf{b}_a) \\ &= d\mathbf{W}_{aa}^\top \mathbf{a}^{(t-1)} + \mathbf{W}_{aa}^\top d\mathbf{a}^{(t-1)} + d\mathbf{W}_{ax}^\top \mathbf{x}^{(t)} + d\mathbf{b}_a \end{aligned}$$

when differentiating w.r.t. \mathbf{W}_{aa} , we have $d\mathbf{a}^{(t-1)} = 0$, $d\mathbf{W}_{ax} = 0$, and $d\mathbf{b}_a = 0$. So,

$$\begin{aligned} dJ &= \text{tr} \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} d\mathbf{W}_{aa}^\top \mathbf{a}^{(t-1)} \right) \\ &= \text{tr} \left((\mathbf{a}^{(t-1)})^\top d\mathbf{W}_{aa} \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \right)^\top \right) \\ &= \text{tr} \left(\left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \right)^\top (\mathbf{a}^{(t-1)})^\top d\mathbf{W}_{aa} \right) \\ &\Rightarrow \frac{\partial J}{\partial \mathbf{W}_{aa}} = \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \right)^\top (\mathbf{a}^{(t-1)})^\top \end{aligned}$$

when differentiating w.r.t. \mathbf{W}_{ax} , we have $d\mathbf{a}^{(t-1)} = 0$, $d\mathbf{W}_{aa} = 0$, and $d\mathbf{b}_a = 0$. So,

$$\begin{aligned} dJ &= \text{tr} \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} d\mathbf{W}_{ax}^\top \mathbf{x}^{(t)} \right) \\ &= \text{tr} \left((\mathbf{x}^{(t)})^\top d\mathbf{W}_{ax} \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \right)^\top \right) \\ &= \text{tr} \left(\left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \right)^\top (\mathbf{x}^{(t)})^\top d\mathbf{W}_{ax} \right) \\ &\Rightarrow \frac{\partial J}{\partial \mathbf{W}_{ax}} = \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \right)^\top (\mathbf{x}^{(t)})^\top \end{aligned}$$

when differentiating w.r.t. \mathbf{b}_a , we have $d\mathbf{W}_{ax} = 0$, $d\mathbf{W}_{aa} = 0$, and $d\mathbf{a}^{(t-1)} = 0$. So,

$$\begin{aligned} dJ &= \text{tr} \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} d\mathbf{b}_a \right) \\ &\Rightarrow \frac{\partial J}{\partial \mathbf{b}_a} = \frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \end{aligned}$$

when differentiating w.r.t. $\mathbf{a}^{(t-1)}$, we have $d\mathbf{W}_{ax} = 0$, $d\mathbf{W}_{aa} = 0$, and $d\mathbf{b}_a = 0$. So,

$$\begin{aligned} dJ &= \text{tr} \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \mathbf{W}_{aa}^\top d\mathbf{a}^{(t-1)} \right) \\ &\Rightarrow \frac{\partial J}{\partial \mathbf{a}^{(t-1)}} = \frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \mathbf{W}_{aa}^\top \end{aligned}$$

3.2 Gradient or Jacobian?

In the above derivations, we have used the numerator layout while performing matrix-derivatives. One of the consequences of this decision is that the derivatives that we have computed are in-fact jacobians and not gradients. Fortunately, gradients are just transpose of jacobians. So, based on our derivations the gradients would be the following:

$$\begin{aligned}\nabla_{\mathbf{W}_{aa}}^{(t)} J &= \left(\frac{\partial J}{\partial \mathbf{W}_{aa}} \right)^\top = \mathbf{a}^{(t-1)} \frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \\ \nabla_{\mathbf{W}_{ax}}^{(t)} J &= \left(\frac{\partial J}{\partial \mathbf{W}_{ax}} \right)^\top = \mathbf{x}^{(t)} \frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \\ \nabla_{\mathbf{W}_{ya}}^{(t)} J &= \left(\frac{\partial J}{\partial \mathbf{W}_{ya}} \right)^\top = \mathbf{a}^{(t)} \frac{\partial J}{\partial \mathbf{z}_y^{(t)}} \\ \nabla_{\mathbf{b}_a}^{(t)} J &= \left(\frac{\partial J}{\partial \mathbf{b}_a} \right)^\top = \left(\frac{\partial J}{\partial \mathbf{z}_a^{(t)}} \right)^\top \\ \nabla_{\mathbf{b}_y}^{(t)} J &= \left(\frac{\partial J}{\partial \mathbf{b}_y} \right)^\top = \left(\frac{\partial J}{\partial \mathbf{z}_y^{(t)}} \right)^\top\end{aligned}$$

4 Appendix-I

The following are some of the common architectures for a Recurrent Neural-Network:

Many-to-many: such that $T_x \neq T_y$

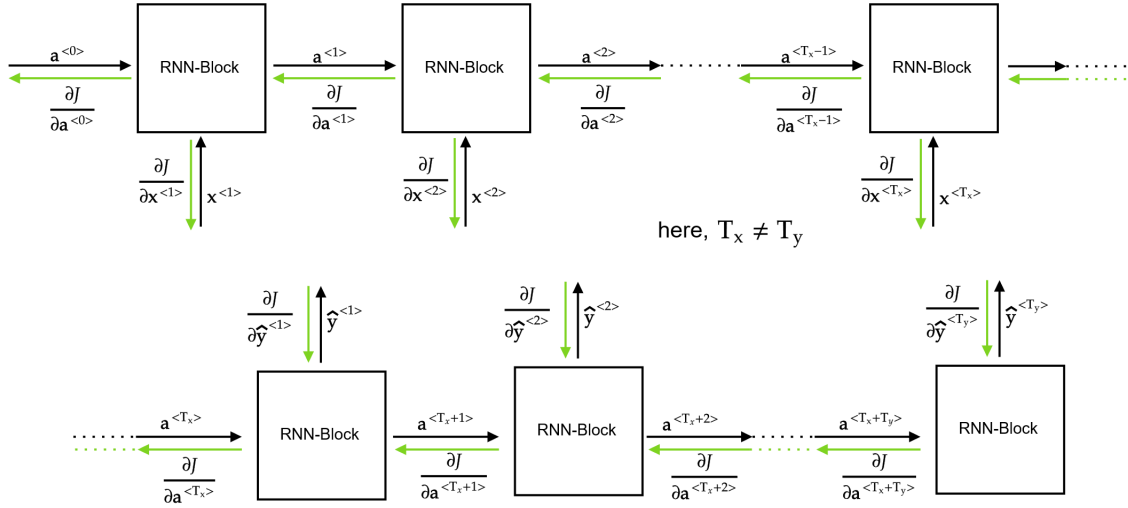


Figure 3: a recurrent neural network with many-to-many architecture, such that $T_x \neq T_y$. Where, T_x is the number of input time-steps, and T_y is the number of output time-steps.

Many-to-one: This is a special case for the many-to-many architecture above, obtained by setting $T_y = 1$, i.e.

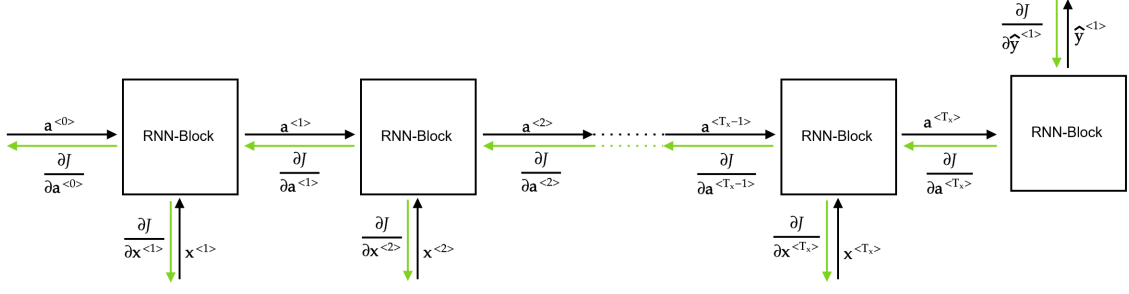


Figure 4: a recurrent neural network with many-to-one architecture.

One-to-many: sometimes, the activation's $\mathbf{a}^{(t)}$ are also passed from one time-step to the next.

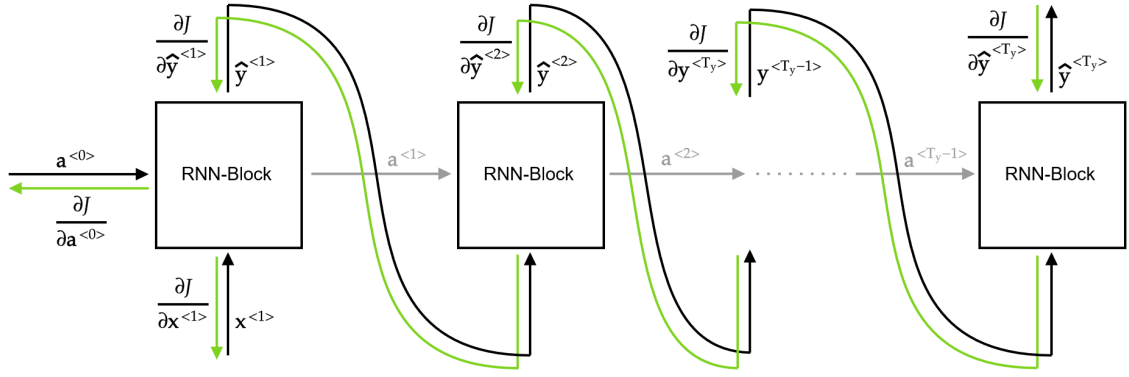


Figure 5: a recurrent neural network with one-to-many architecture. The muted-arrows in the figure denote optional connections between time-steps.

One-to-one: This is a special case for the many-to-many architecture (see section-1), obtained by setting $T_x = T_y = 1$, i.e.

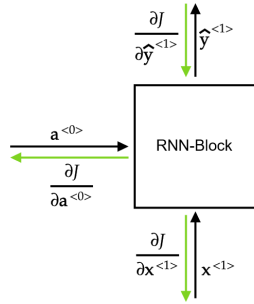


Figure 6: a recurrent neural network with one-to-one architecture. Note that this is essentially a dense neural-network with a single hidden-layer with a $\tanh()$ activation function.

4.1 Additional Architectures

Deep-RNN: a network obtained by stacking l many-to-many RNN's

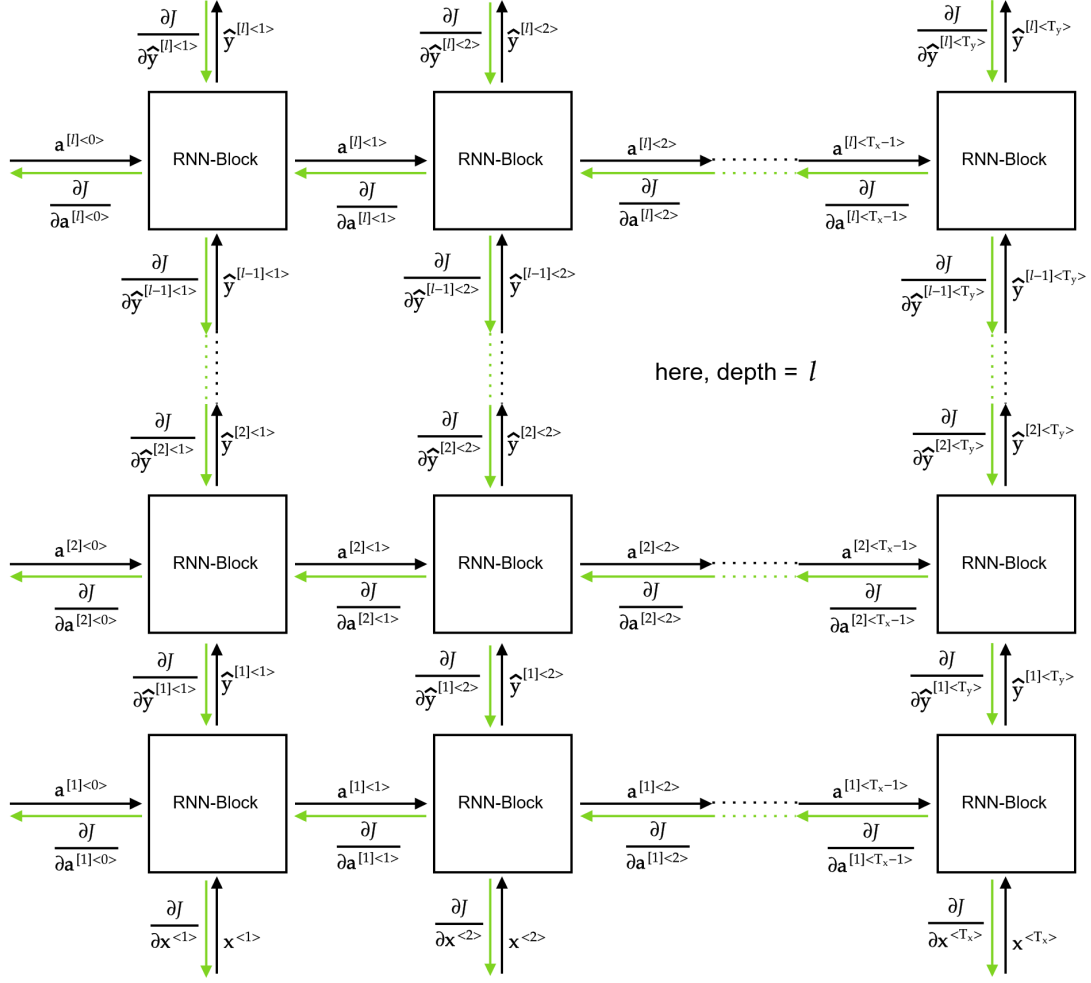


Figure 7: a unidirectional deep recurrent neural-network with a network-depth of l .

Bi-RNN: here, at any time-step t , the affine vector $\mathbf{z}^{(t)}$ is computed as follows:

$$\mathbf{z}^{(t)} = \mathbf{W}_{yaf} f \mathbf{a}^{(t)} + \mathbf{W}_{yab} b \mathbf{a}^{(T_x-t)} + \mathbf{b}_y$$

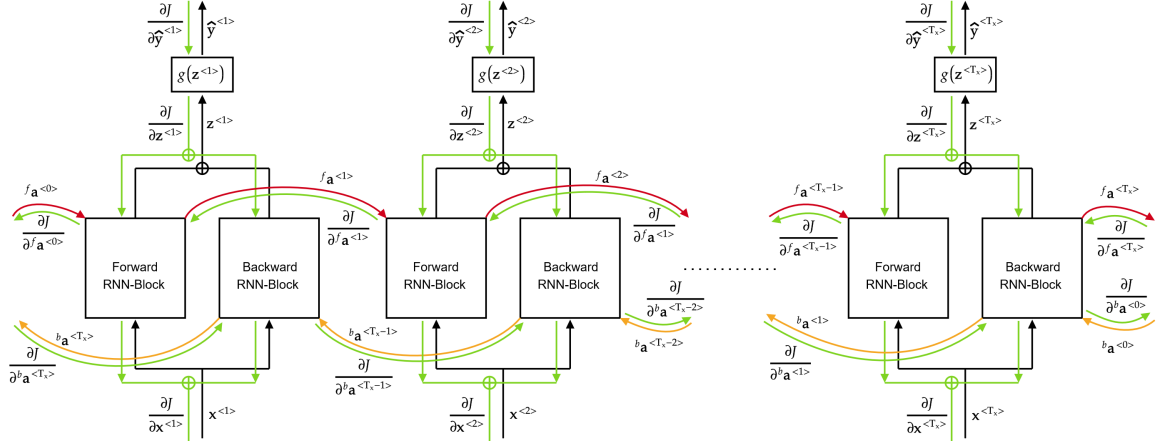


Figure 8: a bi-directional recurrent-neural network with a one-to-one architecture. Here, $g()$ is the activation function.

Deep Bi-RNN: similar to Deep-RNN's, these are obtained by stacking multiple one-to-one Bi-RNN's as layers.