Feedback Neural Networks (a.k.a. Recurrent neural networks)

Jae Yun JUN KIM[*]

March 3, 2020

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle.
This creates an internal state of the network which allows it to exhibit dynamic temporal behavior.
Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs.
Possible applications: handwriting recognition / speech recognition.

Example: XOR operation.
Consider its truth table:

Input		Output
0	0	0
0	1	1
1	0	1
1	1	0

Table 1: The truth table for the XOR operation

Then, let us create a sequence from the above table
000011101110.
If we did not know where this sequence came from, it would have been very difficult to find its structure. From this observation, a question arises naturally: how to learn automatically the structure of a given sequence of data, so that we can make predictions?
We might be able to answer this question using an RNN.

1 Simple recurrent neural networks

There are two well-known types of RNNs: Elman RNN and Jordan RNN.

1.1 Elman RNN

1.1.1 Forward propagation

Input data matrix

[*]ECE Paris Graduate School of Engineering, 37 quai de Grenelle 75015 Paris, France; jae-yun.jun-kim@ece.fr

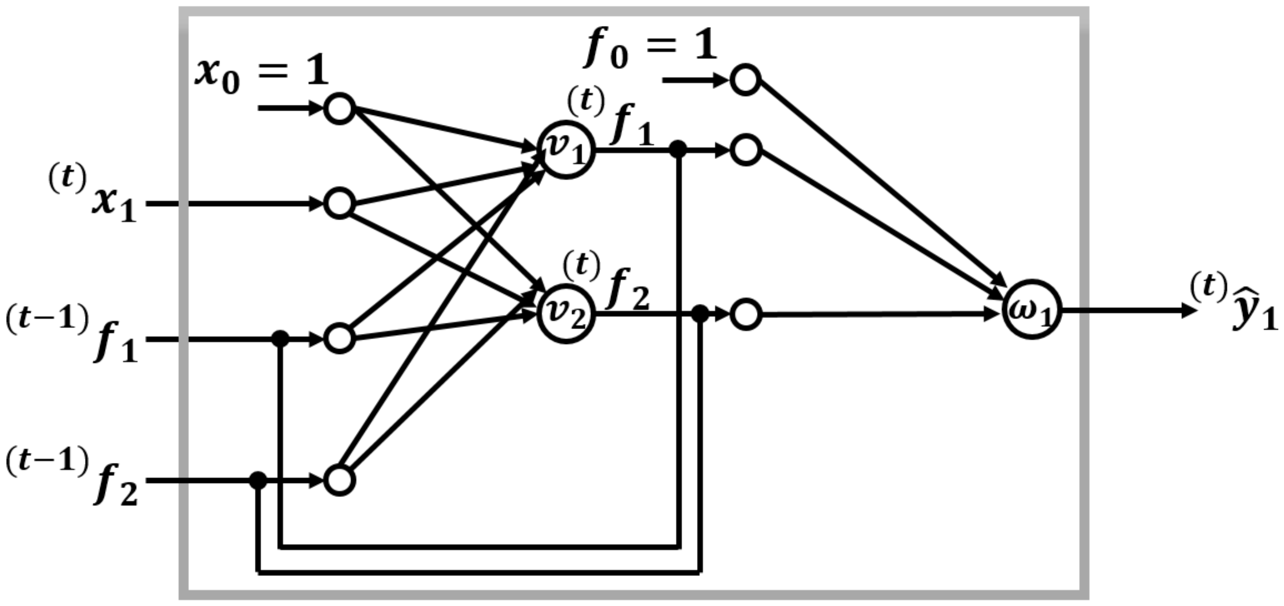


Figure 1: Elman RNN

$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \vdots \\ \left(x^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I\times N}, \qquad \text{where} \quad x^{(i)} \in \mathbb{R}^{N\times 1} \tag{1}$$

Extended input data matrix

$$\overline{X} = \begin{bmatrix} \mathcal{I} & X & {}^{(t-1)}F \end{bmatrix} = \begin{bmatrix} \left(\overline{x}^{(1)}\right)^T \\ \vdots \\ \left(\overline{x}^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I\times(N+1+K)}, \qquad \text{where} \quad \mathcal{I} = [1, \cdots, 1]^T \in \mathbb{R}^{I\times 1} \tag{2}$$

$$\overline{\overline{X}} = \overline{X} \cdot v = \begin{bmatrix} \left(\overline{x}^{(1)}\right)^T \cdot v \\ \vdots \\ \left(\overline{x}^{(I)}\right)^T \cdot v \end{bmatrix} \in \mathbb{R}^{I\times K} \tag{3}$$

Define

$$F = \left(1 + \exp\left(-\overline{\overline{X}}\right)\right)^{-1} = \begin{bmatrix} \left(f^{(1)}\right)^T \\ \vdots \\ \left(f^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I\times K} \tag{4}$$

$$\overline{F} = \begin{bmatrix} \mathcal{I} & F \end{bmatrix} = \begin{bmatrix} \left(\overline{f}^{(1)}\right)^T \\ \vdots \\ \left(\overline{f}^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I\times(K+1)}, \qquad \text{where} \quad \mathcal{I} = [1, \cdots, 1]^T \in \mathbb{R}^{I\times 1} \tag{5}$$

$$\overline{\overline{F}} = \overline{F} \cdot \omega = \begin{bmatrix} \left(\overline{f}^{(1)}\right)^T \cdot \omega \\ \vdots \\ \left(\overline{f}^{(I)}\right)^T \cdot \omega \end{bmatrix} \in \mathbb{R}^{I \times J} \tag{6}$$

Define

$$G = \left(1 + \exp\left(-\overline{\overline{F}}\right)\right)^{-1} = \begin{bmatrix} \left(g^{(1)}\right)^T \\ \vdots \\ \left(g^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I \times J} \tag{7}$$

1.2 Jordan RNN

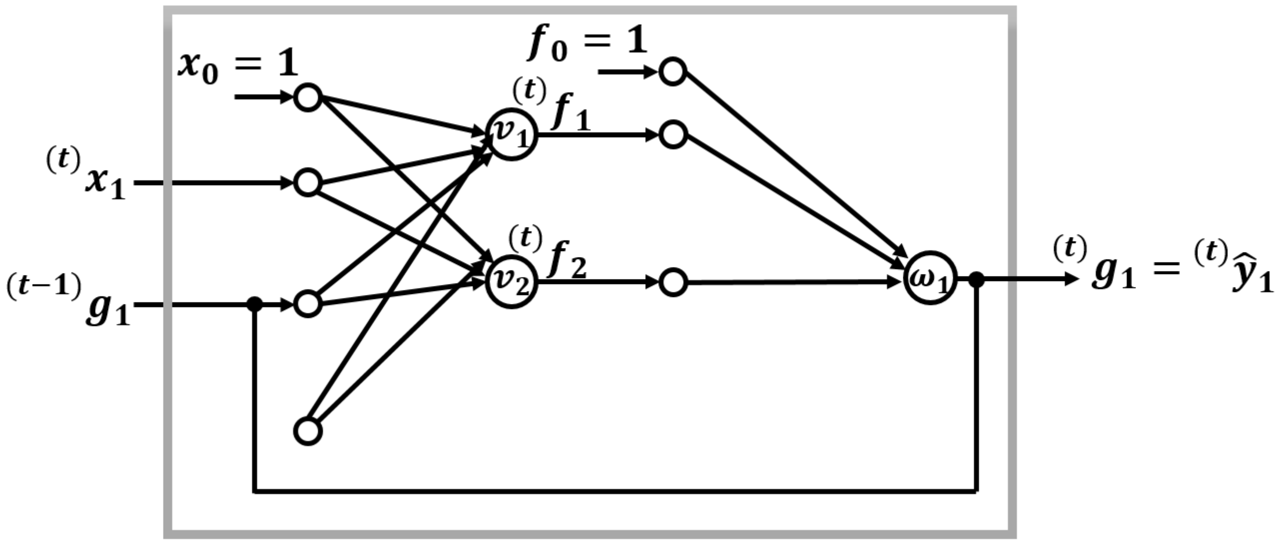


Figure 2: Jordan RNN

1.2.1 Forward propagation

Input data matrix

$$X = \begin{bmatrix} \left(x^{(1)}\right)^T \\ \vdots \\ \left(x^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I \times N}, \qquad \text{where} \quad x^{(i)} \in \mathbb{R}^{N \times 1} \tag{8}$$

Extended input data matrix

$$\overline{X} = \begin{bmatrix} \mathcal{I} & X & {}^{(t-1)}G \end{bmatrix} = \begin{bmatrix} \left(\overline{x}^{(1)}\right)^T \\ \vdots \\ \left(\overline{x}^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I \times (N+1+J)}, \qquad \text{where} \quad \mathcal{I} = [1, \cdots, 1]^T \in \mathbb{R}^{I \times 1} \tag{9}$$

$$\overline{\overline{X}} = \overline{X} \cdot v = \begin{bmatrix} \left(\overline{x}^{(1)}\right)^T \cdot v \\ \vdots \\ \left(\overline{x}^{(I)}\right)^T \cdot v \end{bmatrix} \in \mathbb{R}^{I \times K} \tag{10}$$

Define

$$F = \left(1 + \exp\left(-\overline{\overline{X}}\right)\right)^{-1} = \begin{bmatrix} \left(f^{(1)}\right)^T \\ \vdots \\ \left(f^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I \times K} \tag{11}$$

$$\overline{F} = \begin{bmatrix} \mathcal{I} & F \end{bmatrix} = \begin{bmatrix} \left(\overline{f}^{(1)}\right)^T \\ \vdots \\ \left(\overline{f}^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I \times (K+1)}, \qquad \text{where} \quad \mathcal{I} = [1, \cdots, 1]^T \in \mathbb{R}^{I \times 1} \tag{12}$$

$$\overline{\overline{F}} = \overline{F} \cdot \omega = \begin{bmatrix} \left(\overline{f}^{(1)}\right)^T \cdot \omega \\ \vdots \\ \left(\overline{f}^{(I)}\right)^T \cdot \omega \end{bmatrix} \in \mathbb{R}^{I \times J} \tag{13}$$

Define

$$G = \left(1 + \exp\left(-\overline{\overline{F}}\right)\right)^{-1} = \begin{bmatrix} \left(g^{(1)}\right)^T \\ \vdots \\ \left(g^{(I)}\right)^T \end{bmatrix} \in \mathbb{R}^{I \times J} \tag{14}$$

2 Example

Suppose that we have a sequence of eight bits: $XXXXXXXX$, where $X \in \{0, 1\}$. Then,

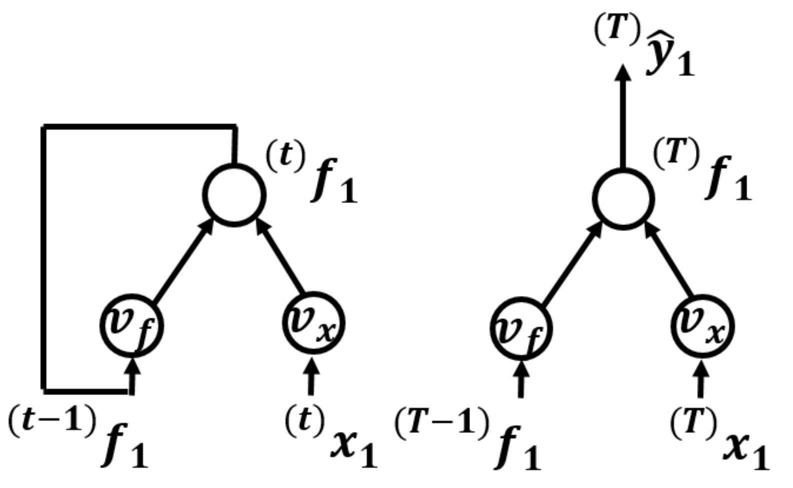
$$y = \sum_{t=1}^{T} x_t \tag{15}$$

where and $T = 8$.

Note:

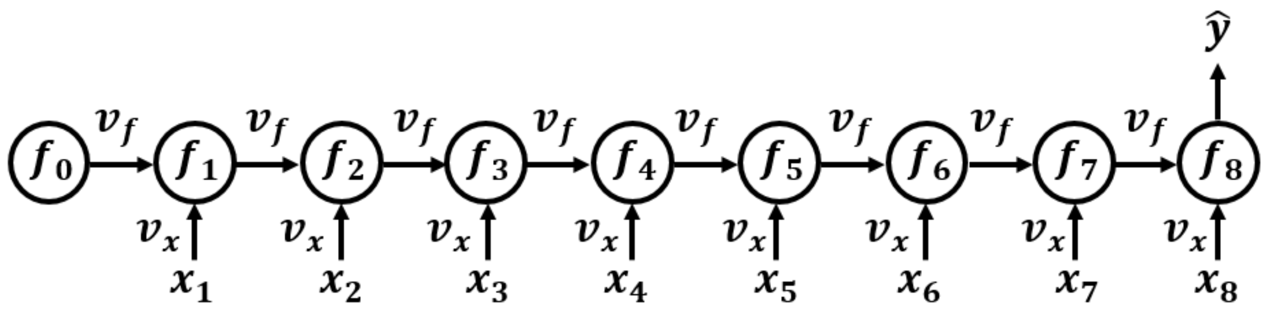
Given the fact that

- If $\forall t, x_t = 0$, then $y = 0$. Hence, we do not need the intercept term.
- y is linear with respect to the input. Hence, the activation function is not needed (it can be the identity function).
- Consider the Elman RNN for simplicity



(a) For intermediate states: for $t =$ 1 : 7 (b) For the last state: for $t = T =$ 8

Figure 3: Elman RNN example



$\hat{y}$
f_0 f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8
v_f
v_x
x_1 x_2 x_3 x_4 x_5 x_6 x_7 x_8

Figure 4: Sequence of states

2.1 Forward propagation

$$\begin{aligned} f_t &= f_{t-1} v_f + x_t v_x \\ \hat{y} &= f_T \end{aligned} \tag{16}$$

2.2 Backpropagation

We define the error (loss) function as

$$E = \frac{1}{2} \sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)} \right)^2 \tag{17}$$

The parameters can be updated using the gradient descent method:

$$\begin{aligned} v_x &\leftarrow v_x - \alpha_x \frac{\partial E}{\partial v_x} \\ v_f &\leftarrow v_f - \alpha_f \frac{\partial E}{\partial v_f} \end{aligned} \tag{18}$$

Now, let us try to find $\frac{\partial E}{\partial v_x}$ and $\frac{\partial E}{\partial v_f}$.

$$\frac{\partial E}{\partial v_x} = \sum_{t=1}^{T} \frac{\partial E}{\partial f_t} \frac{\partial f_t}{\partial v_x} \tag{19}$$

First, we compute $\frac{\partial E}{\partial f_t}$ for $t = 1:8$ as follows

$$\begin{aligned}
\frac{\partial E}{\partial f_8} &= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial f_8} = \sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)}\right) \\
\frac{\partial E}{\partial f_7} &= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial f_8} \frac{\partial f_8}{\partial f_7} = \sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)}\right) v_f \\
&\vdots \\
\frac{\partial E}{\partial f_t} &= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial f_8} \frac{\partial f_8}{\partial f_7} \cdots \frac{\partial f_{t+1}}{\partial f_t} = \sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)}\right) v_f^{(T-t)} \\
&\vdots \\
\frac{\partial E}{\partial f_1} &= \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial f_8} \frac{\partial f_8}{\partial f_7} \cdots \frac{\partial f_2}{\partial f_1} = \sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)}\right) v_f^{(T-1)}
\end{aligned} \tag{20}$$

On the other hand,

$$\frac{\partial f_t}{\partial v_x} = x_t^{(i)} \tag{21}$$

Hence,

$$\begin{aligned}
\frac{\partial E}{\partial v_x} &= \sum_{t=1}^{T} \sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)}\right) v_f^{(T-t)} x_t^{(i)} \\
&= \sum_{t=1}^{T} \left\{ \left[\sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)}\right) x_t^{(i)} \right] v_f^{(T-t)} \right\}
\end{aligned} \tag{22}$$

Then, we compute $\frac{\partial E}{\partial v_f}$.

$$\frac{\partial E}{\partial v_f} = \sum_{t=1}^{T} \frac{\partial E}{\partial f_t} \frac{\partial f_t}{\partial v_f} \tag{23}$$

From the previous steps, we already know that

$$\frac{\partial E}{\partial f_t} = \sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)}\right) v_f^{(T-t)} \tag{24}$$

And,

$$\frac{\partial f_t}{\partial v_f} = f_{t-1}^{(i)}. \tag{25}$$

Hence,

$$\frac{\partial E}{\partial v_f} = \sum_{t=1}^{T} \left\{ \left[\sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)}\right) f_{t-1}^{(i)} \right] v_f^{(T-t)} \right\} \tag{26}$$

3 Resilient propagation

This is an algorithm proposed by Martin Riedmiller and Heinrich Brann.

3.1 Motivation

From the example considered previously, we know that

$$\frac{\partial E}{\partial f_1} = \frac{\partial E}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial f_T} \frac{\partial f_T}{\partial f_{T-1}} \cdots \frac{\partial f_t}{\partial f_{t-1}} \cdots \frac{\partial f_2}{\partial f_1} = \sum_{i=1}^{I} \left(\hat{y}^{(i)} - y^{(i)}\right) v_f^{(T-1)} \tag{27}$$

Suppose now that $T >> 1$, then as $T \to \infty$,

$$\frac{\partial E}{\partial f_1} \to \begin{cases} 0, & \text{if} \quad v_f < 1 \quad \text{(known as vanishing problem)} \\ \infty, & \text{if} \quad v_f > 1 \quad \text{(known as explosion problem)} \end{cases} \tag{28}$$

Then, how can we solve this problem? A simple approach could be using the **resilient propagation**.

3.2 Definition

The **resilient propagation** is similar to the backpropagation approach (in the sense of computing gradients), but it is different in that it does not use the actual values of the gradients. But, it is different in that it does not use the actual values of the gradients. But, it only uses the information coming from the signs of the gradients.
Hence, the new update rules are,

$$\begin{aligned} v_x &\leftarrow v_x - \text{sign}\left(\frac{\partial E}{\partial v_x}\right) \Delta_x \\ v_f &\leftarrow v_f - \text{sign}\left(\frac{\partial E}{\partial v_f}\right) \Delta_f \end{aligned} \tag{29}$$

where

$$\Delta_x \leftarrow \begin{cases} \Delta_x \cdot \eta_p, & \text{if} \quad \text{sign}\left(\frac{\partial E}{\partial v_x}\right)_{\text{previous}} = \text{sign}\left(\frac{\partial E}{\partial v_x}\right)_{\text{current}} \\ \Delta_x \cdot \eta_n, & \text{if} \quad \text{sign}\left(\frac{\partial E}{\partial v_x}\right)_{\text{previous}} \neq \text{sign}\left(\frac{\partial E}{\partial v_x}\right)_{\text{current}} \end{cases} \tag{30}$$

$$\Delta_f \leftarrow \begin{cases} \Delta_f \cdot \eta_p, & \text{if} \quad \text{sign}\left(\frac{\partial E}{\partial v_f}\right)_{\text{previous}} = \text{sign}\left(\frac{\partial E}{\partial v_f}\right)_{\text{current}} \\ \Delta_f \cdot \eta_n, & \text{if} \quad \text{sign}\left(\frac{\partial E}{\partial v_f}\right)_{\text{previous}} \neq \text{sign}\left(\frac{\partial E}{\partial v_f}\right)_{\text{current}} \end{cases} \tag{31}$$

where $\eta_p = 1.2 > 1$, $\eta_n = 0.5 < 1$, and $\Delta_{x_{\text{init}}} = \Delta_{f_{\text{init}}} = 0.001$.

3.2.1 Why $\eta_n = 0.5 < 1$?

Every time the partial derivative of the corresponding weight changes its sign, this indicates that the last update was too big and the algorithm has jumped over a local minimum. Hence, the update value is decreased by η_n to reduce the update step size.

3.2.2 Why $\eta_p = 1.2 > 1$?

If the derivative retains its sign, the update value is slightly increased in order to accelerate the convergence in shallow regions.

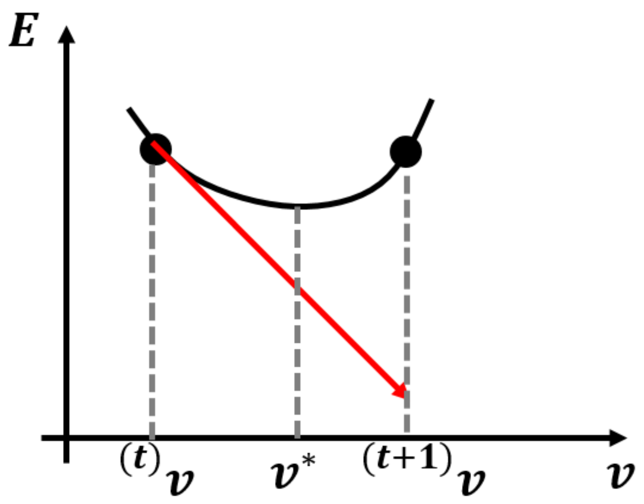


Figure 5: Resilient propagation with large step size

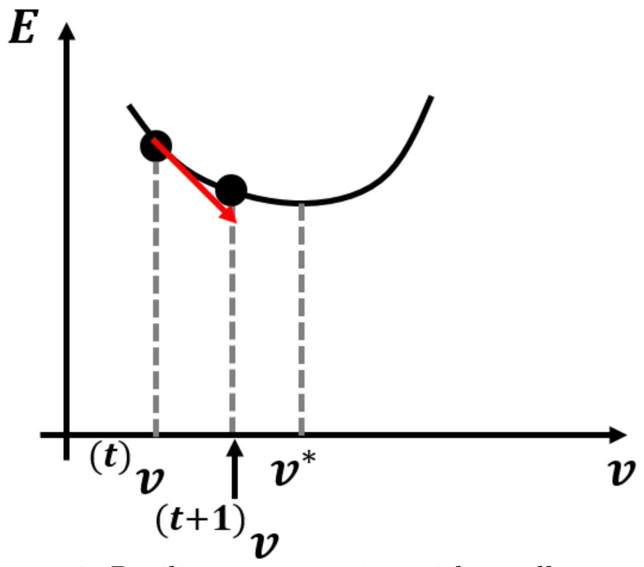


Figure 6: Resilient propagation with small step size

4 Gradient clipping

Another simple way to (possibly) get around the explosion problem is by clipping the gradients. If $\|\frac{\partial E}{\partial V}\| > \eta$ for some $\eta > 0$, then clip the gradient as

$$\frac{\partial E}{\partial V} \longleftarrow \eta \frac{\frac{\partial E}{\partial V}}{\|\frac{\partial E}{\partial V}\|}.$$