Gated Recurrent Unit Networks

DEFINITION

- Gated Recurrent Unit (GRU) is a type of recurrent neural network (RNN) that was introduced **by Cho et al. in 2014** as a simpler alternative to Long Short-Term Memory (LSTM) networks. Like LSTM, GRU can process sequential data such as text, speech, and time-series data.
- The basic idea behind GRU is to use gating mechanisms to selectively update the hidden state of the network at each time step.
- The gating mechanisms are used to control the flow of information in and out of the network. The GRU has two gating mechanisms, called the reset gate and the update gate.

Reset gate: $r_t = \text{sigmoid}(W_r * [h_{t-1}, x_t])$

Update gate: $z_t = \text{sigmoid}(W_z * [h_{t-1}, x_t])$

Candidate hidden state: $h_t' = \tanh(W_h * [r_t * h_{t-1}, x_t])$

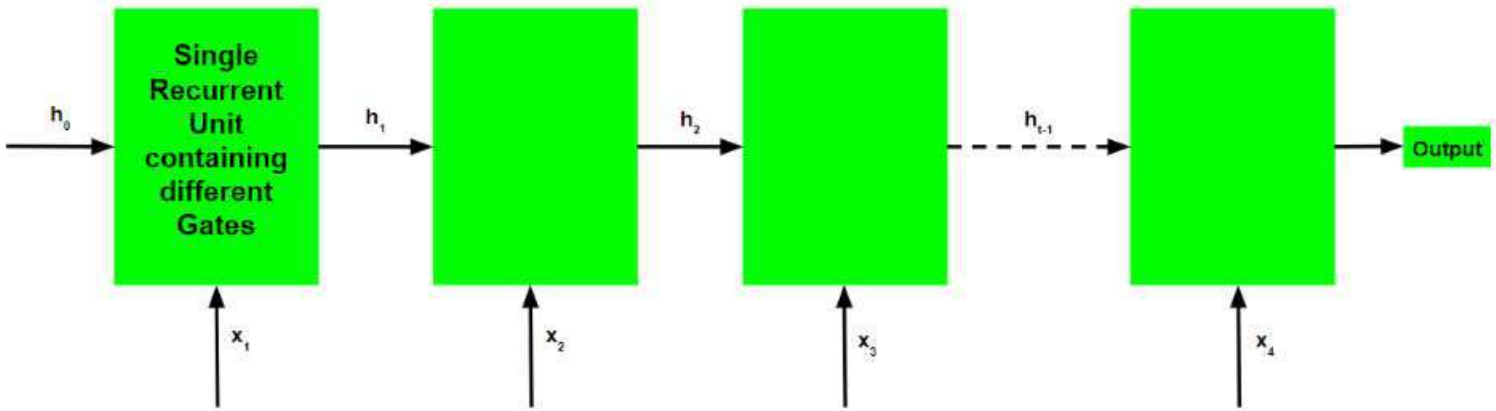
Hidden state: $h_t = (1 - z_t) * h_{t-1} + z_t * h_t'$

where W_r, W_z, and W_h are learnable weight matrices, x_t is the input at time step t, h_{t-1} is the previous hidden state, and h_t is the current hidden state.

Prerequisites: Recurrent Neural Networks, Long Short Term Memory Networks

- To solve the Vanishing-Exploding gradients problem often encountered during the operation of a basic Recurrent Neural Network, many variations were developed. One of the most famous variations is the **Long Short Term Memory Network(LSTM)**. One of the lesser-known but equally effective variations is the **Gated Recurrent Unit Network(GRU)**.

- **Update Gate(z):** It determines how much of the past knowledge needs to be passed along into the future. It is analogous to the Output Gate in an LSTM recurrent unit.
- **Reset Gate(r):** It determines how much of the past knowledge to forget. It is analogous to the combination of the Input Gate and the Forget Gate in an LSTM recurrent unit.
- **Current Memory Gate():** It is often overlooked during a typical discussion on Gated Recurrent Unit Network. It is incorporated into the Reset Gate just like the Input Modulation Gate is a sub-part of the Input Gate and is used to introduce some non-linearity into the input and to also make the input Zero-mean.

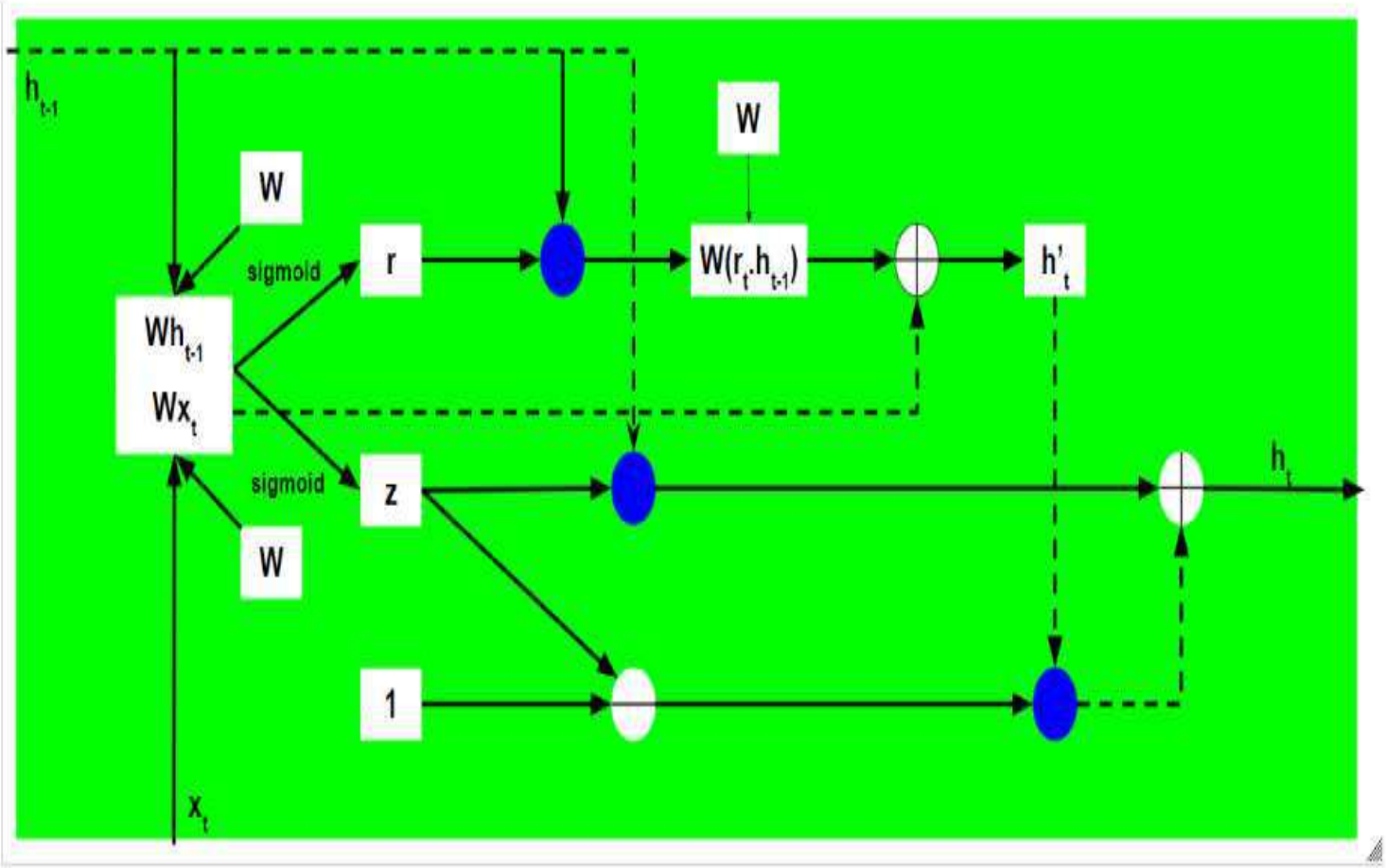

Single Recurrent Unit containing different Gates
h0
h1
h2
ht-1
Output
x1
x2
x3
x4

Working of a Gated Recurrent Unit:

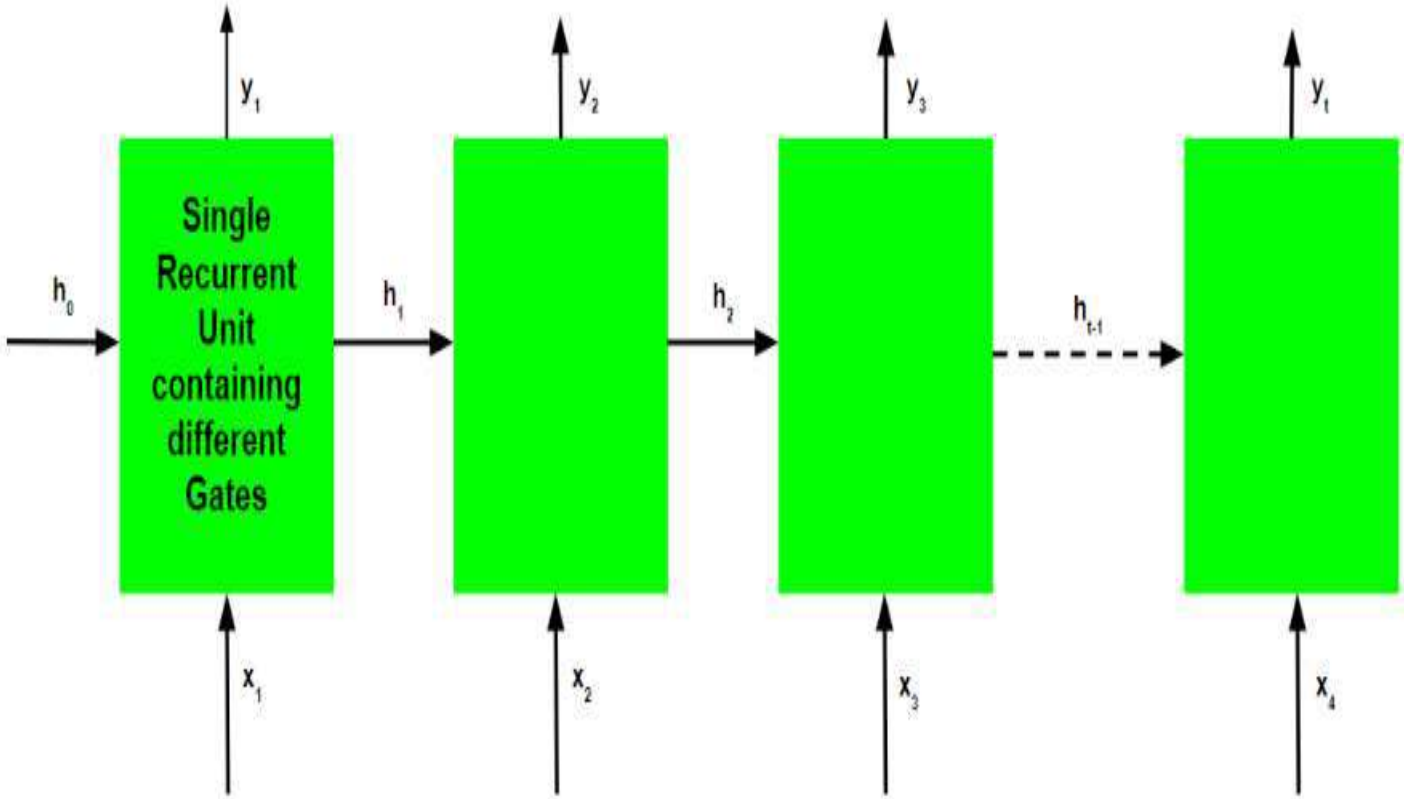
- Take input the current input and the previous hidden state as vectors.
- Calculate the values of the three different gates by following the steps given below:-
  - For each gate, calculate the parameterized current input and previously hidden state vectors by performing element-wise multiplication (Hadamard Product) between the concerned vector and the respective weights for each gate.
  - Apply the respective activation function for each gate element-wise on the parameterized vectors. Below given is the list of the gates with the activation function to be applied for the gate.

- **Update Gate : Sigmoid Function**
- **Reset Gate : Sigmoid Function.**
- The process of calculating the Current Memory Gate is a little different. First, the Hadamard product of the Reset Gate and the previously hidden state vector is calculated. Then this vector is parameterized and then added to the parameterized current input vector.
- To calculate the current hidden state, first, a vector of ones and the same dimensions as that of the input is defined. This vector will be called ones and mathematically be denoted by 1. First, calculate the Hadamard Product of the update gate and the previously hidden state vector. Then generate a new vector by subtracting the update gate from ones and then calculate the Hadamard Product of the newly generated vector with the current memory gate. Finally, add the two vectors to get the currently hidden state vector.

The above-stated working is stated as below:-



- Note that the blue circles denote element-wise multiplication. The positive sign in the circle denotes vector addition while the negative sign denotes vector subtraction(vector addition with negative value). The weight matrix W contains different weights for the current input vector and the previous hidden state for each gate.
- Just like Recurrent Neural Networks, a GRU network also generates an output at each time step and this output is used to train the network using gradient descent.


y1
y2
y3
yt
Single
Recurrent
Unit
containing
different
Gates
h0
h1
h2
ht-1
x1
x2
x3
x4

Let $\overline{y}_t$ be the predicted output at each time step and y_t be the actual output at each time step. Then the error at each time step is given by:-

$$E_t = -y_t log(\overline{y}_t)$$

The total error is thus given by the summation of errors at all time steps.

$$E = \sum_t E_t$$
$$\Rightarrow E = \sum_t -y_t log(\overline{y}_t)$$

Similarly, the value $\frac{\partial E}{\partial W}$ can be calculated as the summation of the gradients at each time step.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

Using the chain rule and using the fact that $\overline{y}_t$ is a function of h_t and which indeed is a function of $\overline{h}_t$, the following expression arises:-

$$\frac{\partial E_t}{\partial W} = \frac{\partial E_t}{\partial \overline{y}_t} \frac{\partial \overline{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots\cdots \frac{\partial h_0}{\partial W}$$

Thus the total error gradient is given by the following:-

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial \overline{y}_t} \frac{\partial \overline{y}_t}{\partial h_t} \frac{\partial h_t}{\partial h_{t-1}} \frac{\partial h_{t-1}}{\partial h_{t-2}} \cdots\cdots \frac{\partial h_0}{\partial W}$$

Note that the gradient equation involves a chain of ∂h_t which looks similar to that of a basic Recurrent Neural Network but this equation works differently because of the internal workings of the derivatives of h_t.

How do Gated Recurrent Units solve the problem of vanishing gradients?

The value of the gradients is controlled by the chain of derivatives starting from $\frac{\partial h_t}{\partial h_{t-1}}$. Recall the expression for h_t:-

$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot \overline{h}_t$

Using the above expression, the value for $\frac{\partial h_t}{\partial h_{t-1}}$is:-

$\frac{\partial h_t}{\partial h_{t-1}} = z + (1 - z)\frac{\partial \overline{h}_t}{\partial h_{t-1}}$

Recall the expression for $\overline{h}_t$:-

$\overline{h}_t = tanh(W \odot x_t + W \odot (r_t \odot h_{t-1}))$

Using the above expression to calculate the value of $\frac{\partial \overline{h_t}}{\partial h_{t-1}}$:-

$\frac{\partial \overline{h_t}}{\partial h_{t-1}} = \frac{\partial(tanh(W \odot x_t + W \odot (r_t \odot h_{t-1})))}{\partial h_{t-1}} \Rightarrow \frac{\partial \overline{h_t}}{\partial h_{t-1}} = (1 - \overline{h}_t^2)(W \odot r)$

Since both the update and reset gate use the sigmoid function as their activation function, both can take values either 0 or 1.

Case 1(z = 1):

In this case, irrespective of the value of r, the term $\frac{\partial \overline{h_t}}{\partial h_{t-1}}$is equal to z which in turn is equal to 1.

Case 2A(z=0 and r=0):

In this case, the term $\frac{\partial \overline{h_t}}{\partial h_{t-1}}$is equal to 0.

- **Case 2B(z=0 and r=1):**
- In this case, the term is equal to . This value is controlled by the weight matrix which is trainable and thus the network learns to adjust the weights in such a way that the term comes closer to 1.
- Thus the Back-Propagation Through Time algorithm adjusts the respective weights in such a manner that the value of the chain of derivatives is as close to 1 as possible.