GRU

A Gated Recurrent Unit (GRU) is a variant of the RNN architecture and uses gating mechanisms to control and manage the flow of information between cells in the neural network.

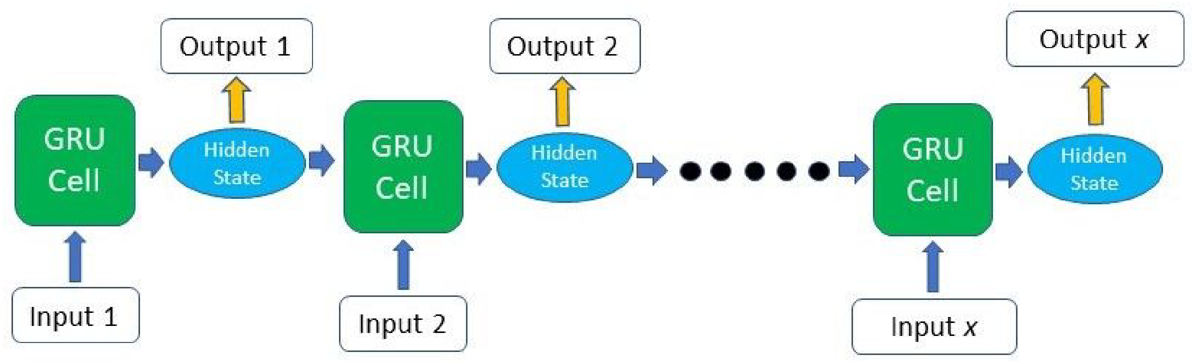
The Gated Recurrent Unit (GRU) is also a type of Recurrent Neural Network (RNN) and the younger sibling of the more popular Long Short-Term Memory (LSTM) network and. Just like its sibling GRUs are able to effectively retain long-term dependencies in sequential data. And additionally, they can address the short-term memory issue plaguing vanilla RNNs.

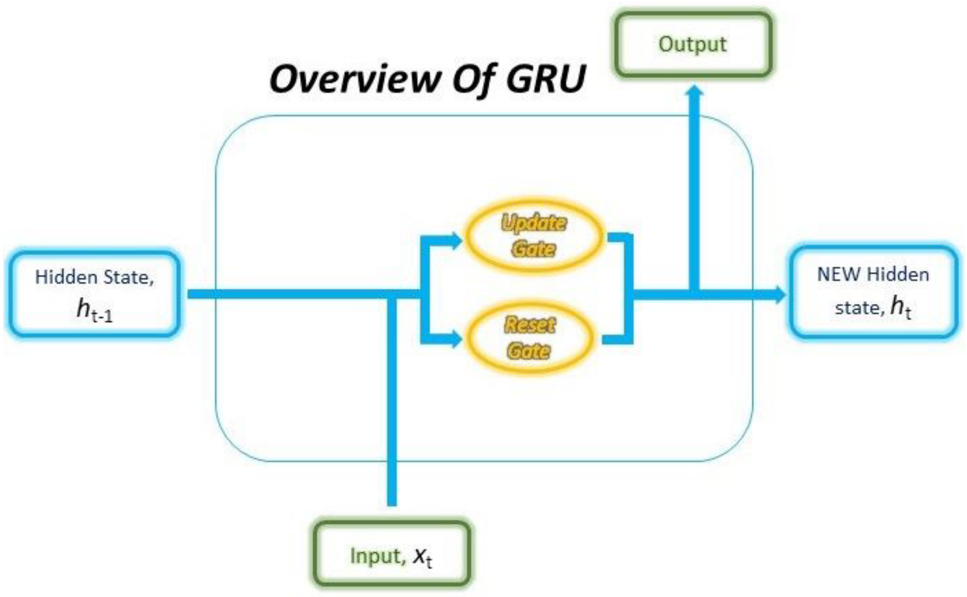
The structure of the GRU allows it to adaptively capture dependencies from large sequences of data without discarding information from earlier parts of the sequence. This is achieved through its gating units similar to the ones in LSTMs which also resolves the vanishing/exploding gradient problem of traditional RNNs. The update gate and reset gate gates are responsible for regulating the information to be kept or discarded at each time step. These gates are two vectors which decide what information should be passed to the output. The special thing about them is that they can be trained to keep information from long ago, without washing it through time or remove information which is irrelevant to the prediction.

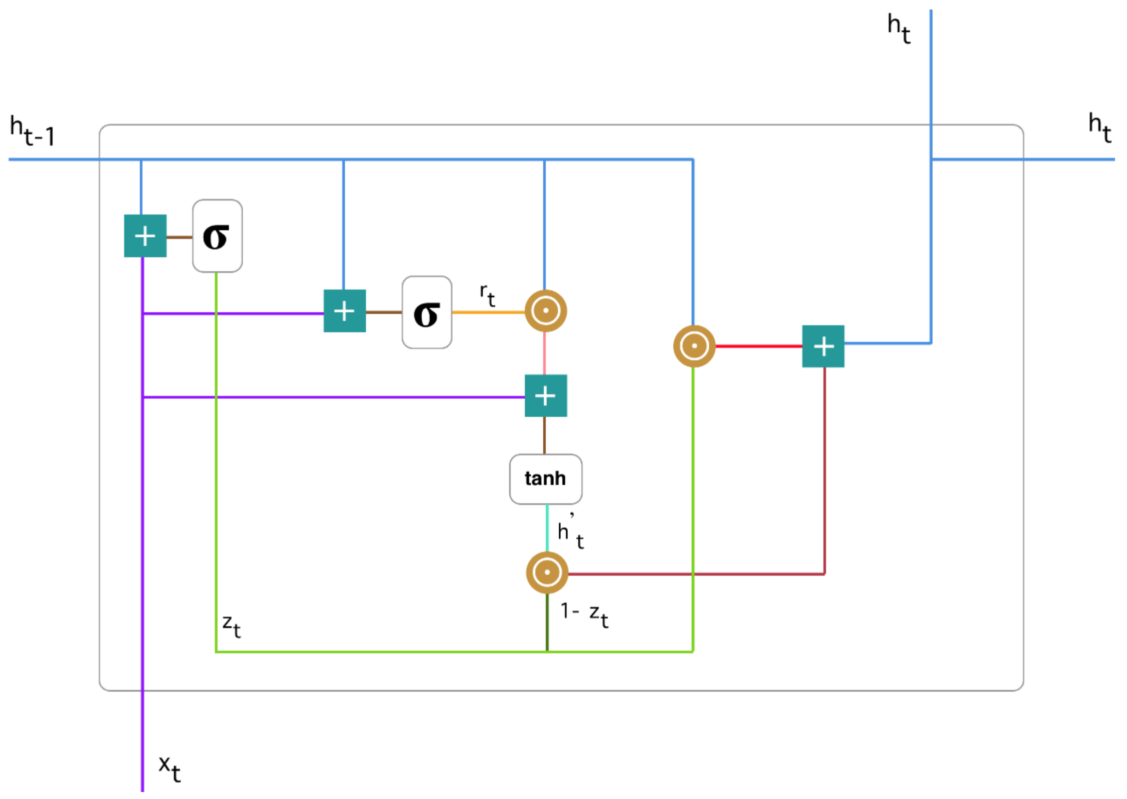
How GRU works

While LSTMs have two different states passed between the cells which are the cell state and the hidden state which carry the long and short-term memory respectively, GRUs only have one hidden state transferred between time steps. This hidden state is able to hold both the long-term and short-term dependencies at the same time due to the gating mechanisms and computations that the hidden state and input data go through.

The GRU cell contains only two gates which are the update gate and reset gate. Just like the gates in LSTMs these gates in the GRU are trained to selectively filter out any irrelevant information while keeping what's useful. These gates are essentially vectors containing values between 0 to 1 which will be multiplied with the input data and/or hidden state. A 0 value in the gate vectors indicates that the corresponding data in the input or hidden state is unimportant and will therefore return as a zero. On the other hand, a 1 value in the gate vector means that the corresponding data is important and will be used.







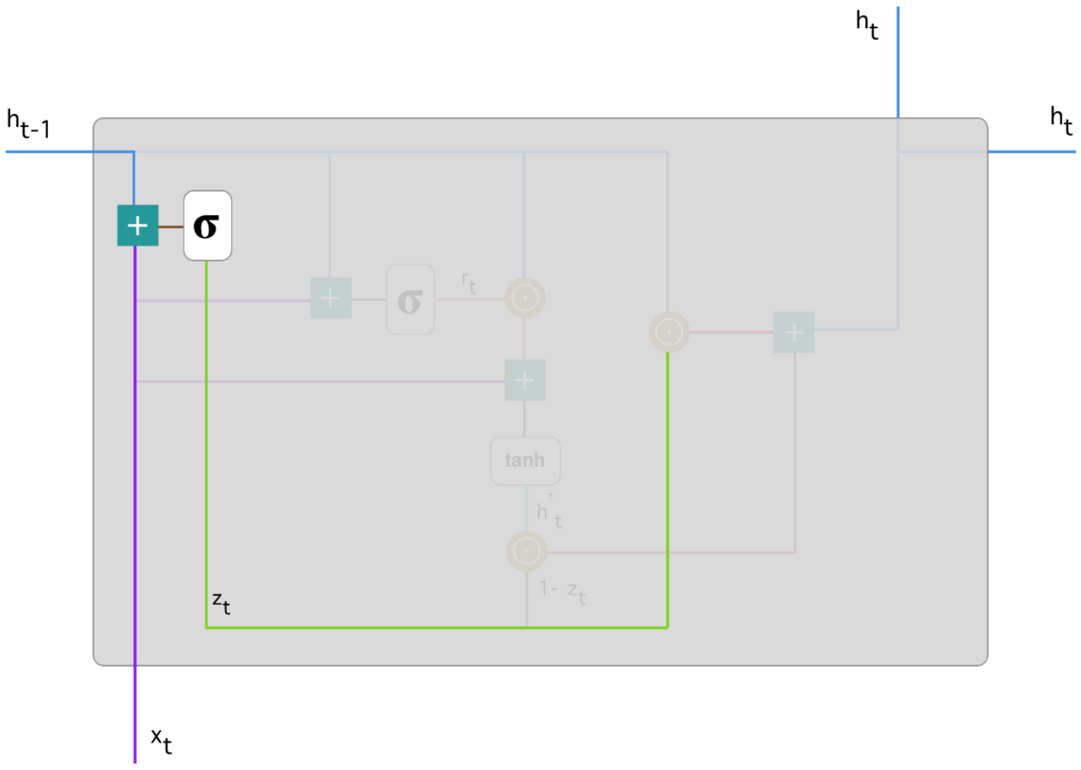
+	σ	⊙	tanh
“plus” operation	“sigmoid” function	“Hadamard product” operation	“tanh” function

While the structure may look rather complicated due to the large number of connections, the mechanism behind it can be broken down into below four main steps.

Update gate - The update gate helps the model to determine how much of the past information (from previous time steps) needs to be passed along to the future. It is responsible for determining the amount of previous information (prior time steps) that needs to be passed along the next state.

We multiply x_t by some weight W(z) and plug it into the network unit. The same goes for h_(t-1) which holds the information for the previous t-1 units which is also multiplied by some weight U(z). Both results are added together and a sigmoid activation function is applied to squash the result between 0 and 1.

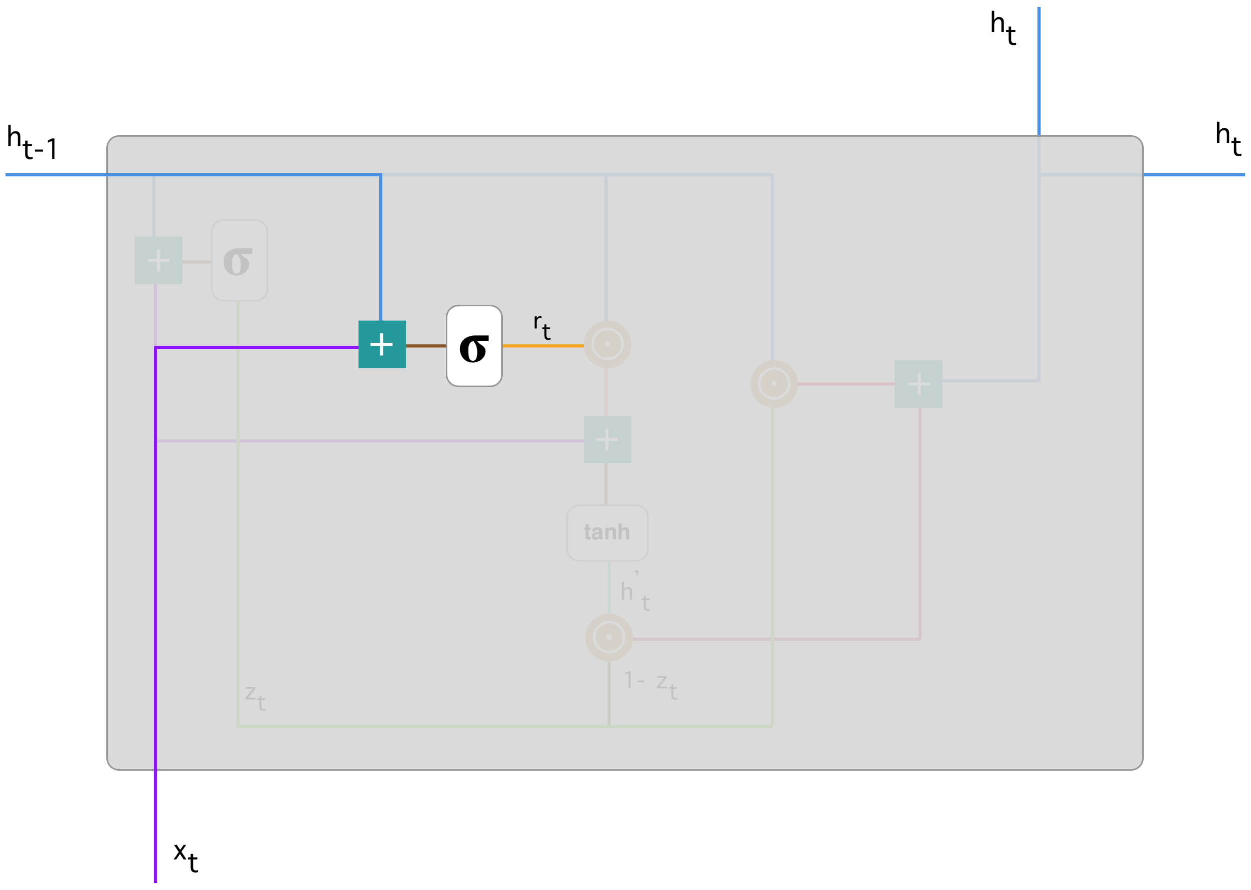
$$z_t = \sigma(W^{(z)}x_t + U^{(z)}h_{t-1})$$


h_t
h_{t-1}
h_t
σ
σ
r_t
tanh
h'_t
z_t
$1- z_t$
x_t

Reset gate - This gate is used from the model to decide how much of the past information to forget/neglect

This formula is the same as the one for the update gate. The difference comes in the weights and the gate’s usage.

$$r_t = \sigma(W^{(r)}x_t + U^{(r)}h_{t-1})$$



Current memory content - We introduce a new memory content which will use the reset gate to store the relevant information from the past. It is calculated using the forget gate.

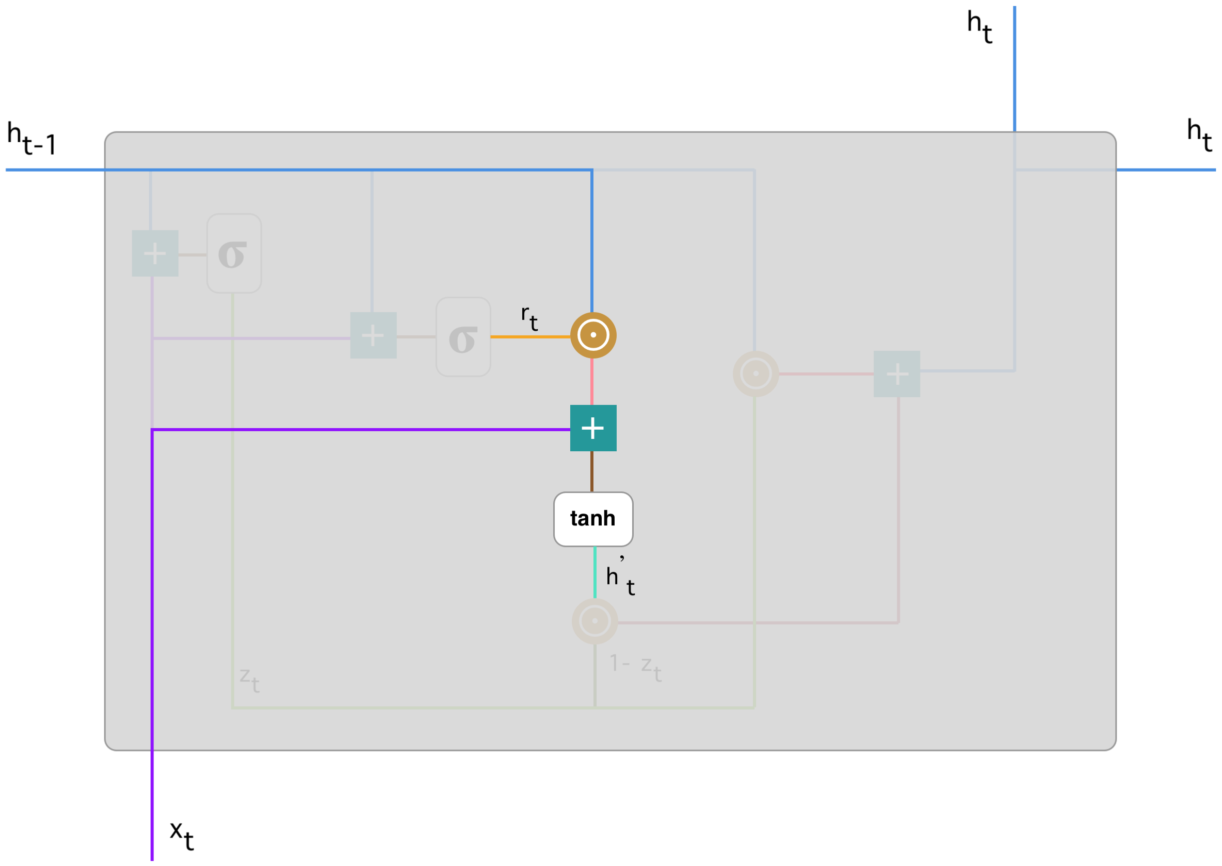
$$h_t' = \tanh(Wx_t + r_t \odot Uh_{t-1})$$

1. Multiply the input x_t with a weight W and h_(t-1) with a weight U.

2. Calculate the Hadamard (element-wise) product between the reset gate r_t and Uh_(t-1). That will determine what to remove from the previous time steps.

3. Sum up the results of step 2 and Wx_t.

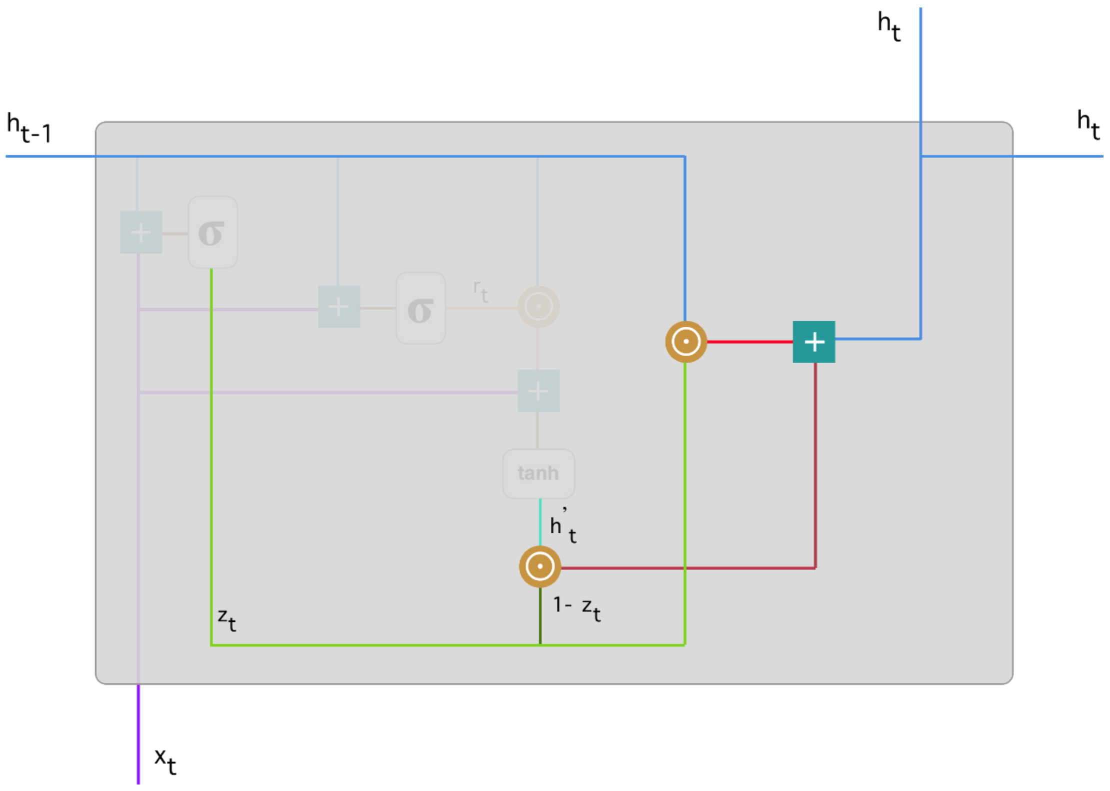
4. Apply the nonlinear activation function tanh on result of step 3



Final memory at current time step - As the last step the network needs to calculate h_t - the vector which holds information for the current unit and passes it down to the network. In order to do that the update gate is needed. It determines what to collect from the current memory content h't and what from the previous steps h_(t-1)

$$h_t = z_t \odot h_{t-1} + (1 - z_t) \odot h_t'$$

1. Apply element-wise multiplication (Hadamard) to the update gate z_t and h_(t-1).
2. Apply element-wise multiplication (Hadamard) to (1-z_t) and h’t.
3. Sum the results from step 1 and 2.



GRU vs RNN

- LSTM ’s and GRU’s can learn long-term dependencies that normal RNNs fundamentally can’t. They were created as the solution to short-term memory. They can remember information for a long period of time. This is possible as they have internal mechanisms called gates that can regulate the flow of information. These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions at the other end.

- LSTM’s and GRU’s overcome the vanishing gradient problem which RNN’s usually face.

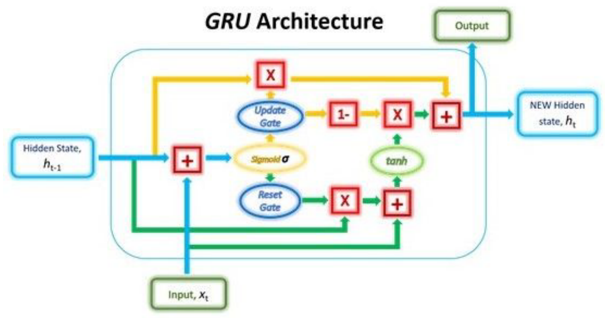
GRU vs LSTM

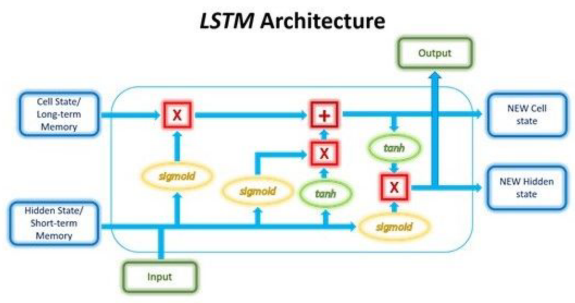
- While both GRUs and LSTMs contain gates, the main difference between these two structures lies in the number of gates and their specific roles.

The role of the Update gate in the GRU is very similar to the Input and Forget gates in the LSTM. However, the control of adding new memory content to the network differs between these two.

In LSTM, the Forget gate determines which part of the previous cell state to retain, the Input gate determines the amount of new memory to be added. These two gates are independent of each other, meaning that the amount of new information added through the Input gate is completely independent of the information retained through the Forget gate.

In GRU, the Update gate is responsible for determining which information from the previous memory to retain and is also responsible for controlling the new memory to be added. This means that the retention of previous memory and addition of new information to the memory in the GRU is NOT independent.





- Since there are fewer number of gates in the GRU (two gates) as compared to the LSTM (three gates), number of weights and parameters to update during training are less in GRU than LSTM. Thus GRUs use less memory and are faster to train