Gated Recurrent Unit (GRU)

GRU stands for Gated Recurrent Unit, which is a type of recurrent neural network (RNN) architecture that is similar to LSTM (Long Short-Term Memory). Like LSTM, GRU is designed to model sequential data by allowing information to be selectively remembered or forgotten over time. However, GRU has a simpler architecture than LSTM, with fewer parameters, which can make it easier to train and more computationally efficient.

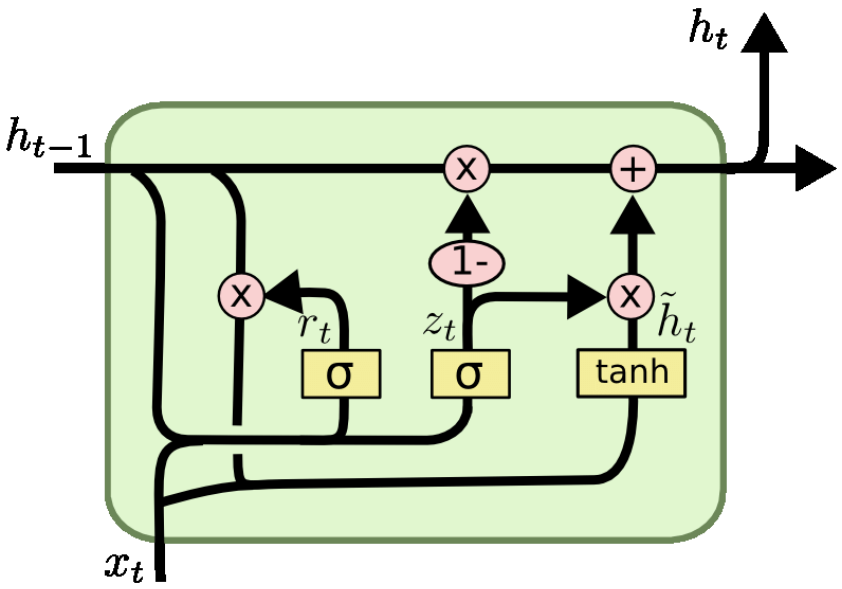
The main difference between GRU and LSTM is the way they handle the memory cell state. In LSTM, the memory cell state is maintained separately from the hidden state and is updated using three gates: the input gate, output gate, and forget gate. In GRU, the memory cell state is replaced with a "candidate activation vector," which is updated using two gates: the reset gate and update gate.

GRU Architecture:

The GRU architecture consists of the following components:

1. Input layer: The input layer takes in sequential data, such as a sequence of words or a time series of values, and feeds it into the GRU.
2. Hidden layer: The hidden layer is where the recurrent computation occurs. At each time step, the hidden state is updated based on the current input and the previous hidden state. The hidden state is a vector of numbers that represents the network's "memory" of the previous inputs.
3. Reset gate: The reset gate determines how much of the previous hidden state to forget. It takes as input the previous hidden state and the current input, and produces a vector of numbers between 0 and 1 that controls the degree to which the previous hidden state is "reset" at the current time step.
4. Update gate: The update gate determines how much of the candidate activation vector to incorporate into the new hidden state. It takes as input the previous hidden state and the current input, and produces a vector of numbers between 0 and 1 that controls the degree to which the candidate activation vector is incorporated into the new hidden state.
5. Candidate activation vector: The candidate activation vector is a modified version of the previous hidden state that is "reset" by the reset gate and combined with the current input. It is computed using a tanh activation function that squashes its output between -1 and 1.
6. Output layer: The output layer takes the final hidden state as input and produces the network's

output. This could be a single number, a sequence of numbers, or a probability distribution over classes, depending on the task at hand.



- The reset gate r and update gate z are computed using the current input x and the previous hidden state h_t-1

```
r_t = sigmoid(W_r * [h_t-1, x_t])
z_t = sigmoid(W_z * [h_t-1, x_t])
```

where W_r and W_z are weight matrices that are learned during training.

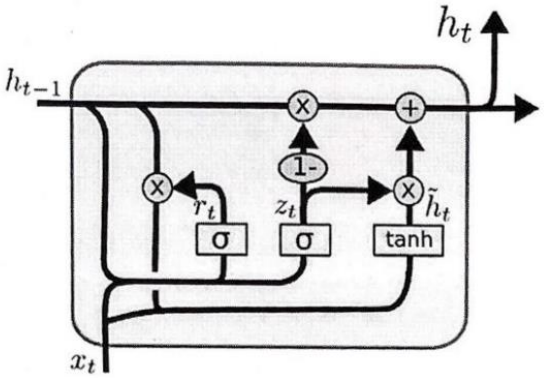
- The candidate activation vector h_t~ is computed using the current input x and a modified version of the previous hidden state that is "reset" by the reset gate:

```
h_t~ = tanh(W_h * [r_t * h_t-1, x_t])
```

- The new hidden state h_t is computed by combining the candidate activation vector with the previous hidden state, weighted by the update gate:

```
h_t = (1 - z_t) * h_t-1 + z_t * h_t~
```

Summary:



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Overall, the reset gate determines how much of the previous hidden state to remember or forget, while the update gate determines how much of the candidate activation vector to incorporate into the new hidden state. The result is a compact architecture that is able to selectively update its hidden state based on the input and previous hidden state, without the need for a separate memory cell state like in LSTM.