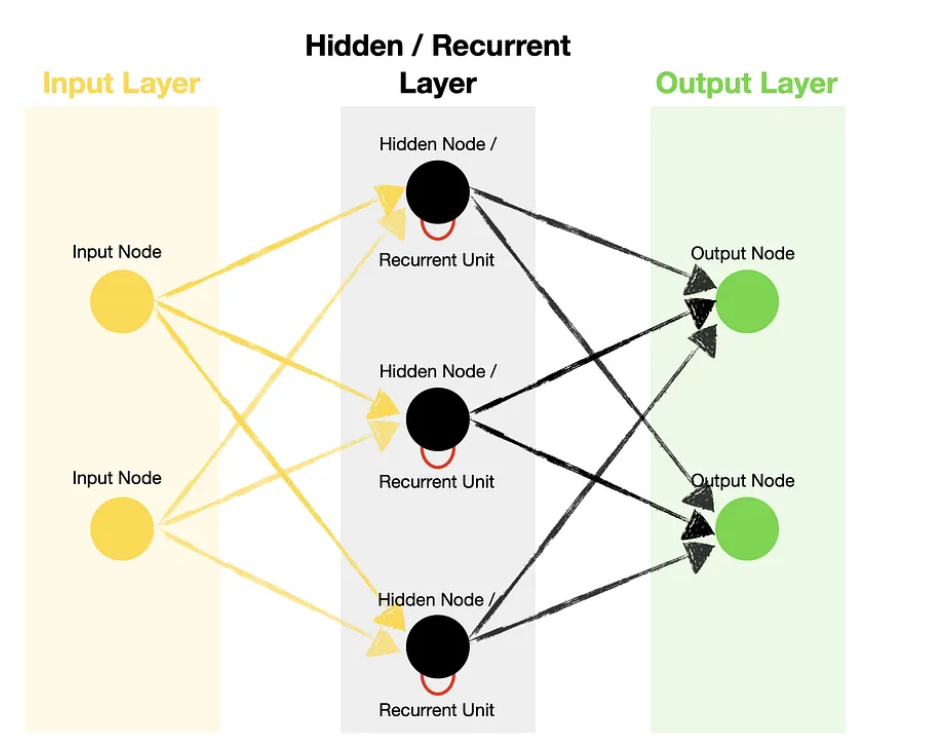
**GRU Recurrent Neural Networks — A Smart Way to Predict Sequences in Python**

# Intro

**Gated Recurrent Units (GRU)** and [**Long Short-Term Memory (LSTM)**](https://towardsdatascience.com/lstm-recurrent-neural-networks-how-to-teach-a-network-to-remember-the-past-55e54c2ff22e)have been introduced to tackle the issue of vanishing / exploding gradients in the standard [Recurrent Neural Networks (RNNs)](https://towardsdatascience.com/rnn-recurrent-neural-networks-how-to-successfully-model-sequential-data-in-python-5a0b9e494f92).

# How is GRU constructed, and how does it differ from standard RNN and LSTM?

Let’s remind ourselves of the typical RNN structure, which contains input, hidden and output layers. Note that you can have any number of nodes, and the below 2–3–2 design is just for illustration.



Unlike [Feed Forward Neural Networks](https://towardsdatascience.com/feed-forward-neural-networks-how-to-successfully-build-them-in-python-74503409d99a), RNNs contain **recurrent units** in their hidden layer, which allow the algorithm to process **sequence data**. This is done by recurrently passing hidden states from previous **timesteps** and combining them with inputs of the current one.

Timestep — single processing of the inputs through the recurrent unit. The number of timesteps is equal to the length of the sequence.

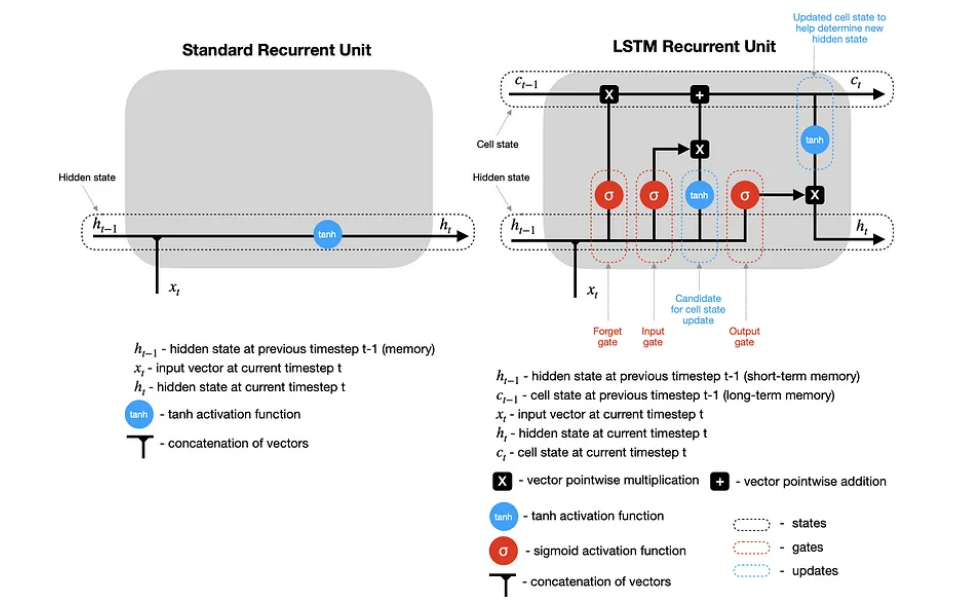
## The recurrent unit architecture inside a standard RNN and LSTM

We know that RNNs utilize **recurrent units** to learn from the sequence data, which is true for all three types — standard RNN, LSTM, and GRU.

However, what happens inside the recurrent unit is very different between them.

For example, standard RNN uses a hidden state to remember information. Meanwhile, LSTM and GRU introduce gates to control what to remember and what to forget before updating the hidden state. In addition to that, LSTM also has a cell state, which acts as long-term memory.

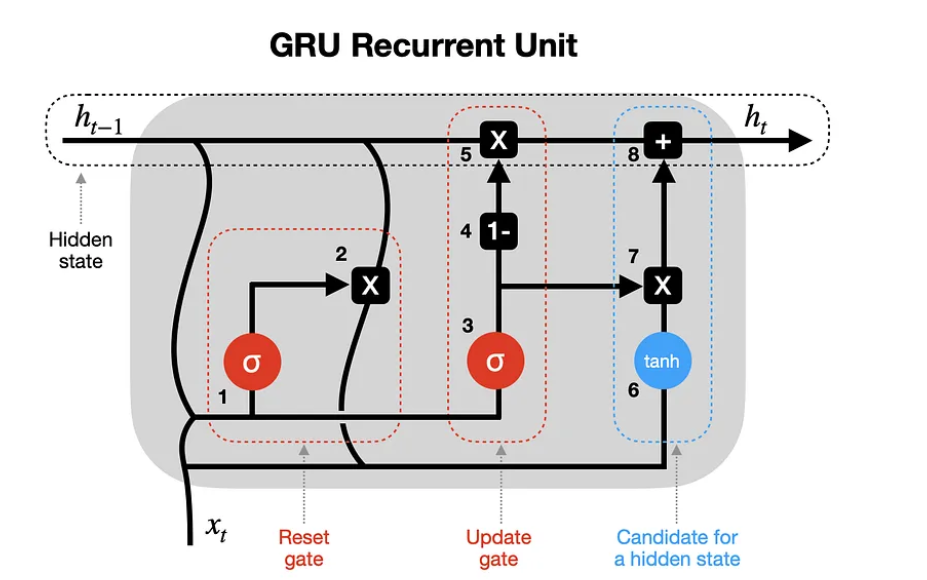
Here are simplified recurrent unit diagrams (weights and biases not shown) for standard RNN and LSTM. See how they compare to each other.

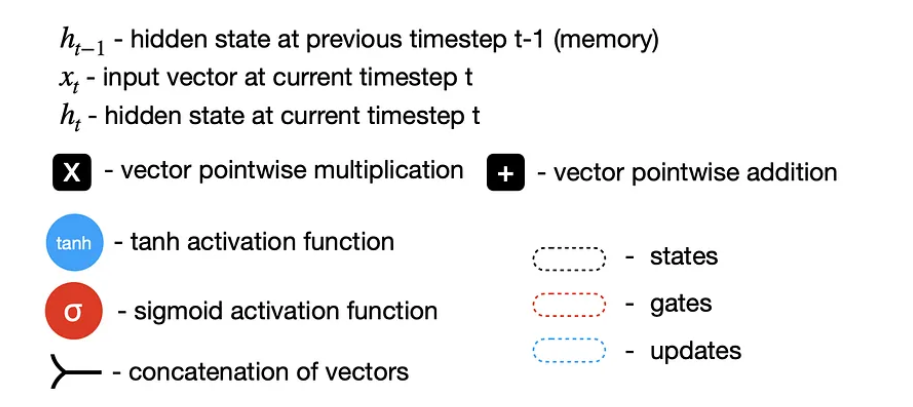


Note that in both cases, after the hidden state (and the cell state for LSTM) is calculated at timestep t, they are **passed back to the recurrent unit** and combined with the input at timestep t+1 to calculate the new hidden state (and cell state) at timestep t+1. This process repeats for t+2, t+3, …, t+n until the predefined number (n) of timesteps is reached.

## **How does GRU work?**

GRU is similar to LSTM, but it has fewer gates. Also, it relies solely on a hidden state for memory transfer between recurrent units, so there is no separate cell state. Let’s analyze this simplified GRU diagram in detail (weights and biases not shown).





**1–2 Reset gate**— previous hidden state (h\_t-1) and current input (x\_t) are combined (multiplied by their respective weights and bias added) and passed through a reset gate. Since the sigmoid function ranges between 0 and 1, step one sets which values should be discarded (0), remembered (1), or partially retained (between 0 and 1). Step two resets the previous hidden state multiplying it with outputs from step one.

**3–4–5 Update gate** — step three may seem analogous to step one, but keep in mind that weights and biases used to scale these vectors are different, providing a different sigmoid output. So, after passing a combined vector through a sigmoid function, we subtract it from a vector containing all 1s (step four) and multiply it by the previous hidden state (step five). That’s one part of updating the hidden state with new information.

**6–7–8 Hidden state candidate**— after resetting a previous hidden state in step two, the outputs are combined with new inputs (x\_t), multiplying them by their respective weights and adding biases before passing through a tanh activation function (step six). Then the hidden state candidate is multiplied by the results of an update gate (step seven) and added to previously modified h\_t-1 to form the new hidden state h\_t.

Next, the process repeats for timestep t+1, etc., until the recurrent unit processes the entire sequence.

Now, we will use GRU to create a **many-to-many** prediction model, which means using a sequence of values to predict the following sequence. Note that GRU could also be used in one-to-one (not recommended because it’s not sequence data), many-to-one, and one-to-many setups.