Recurrent Neural Networks

Recurrent Neural Networks

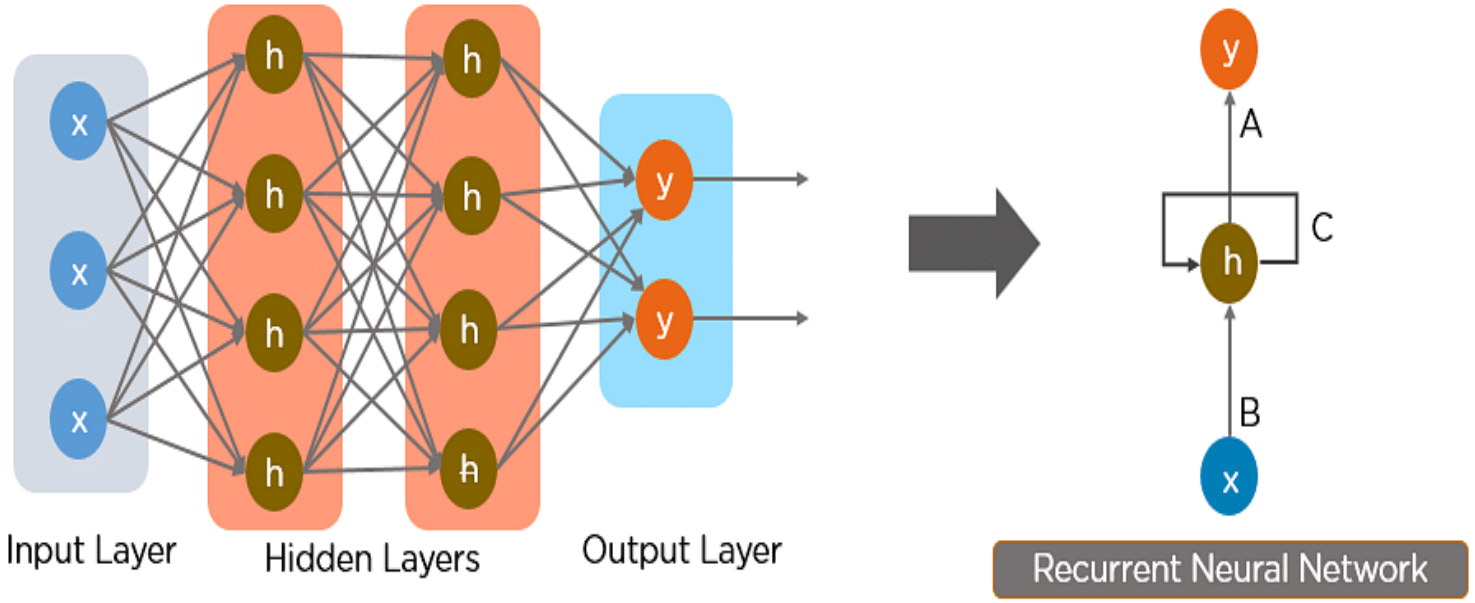
- Recurrent neural networks (RNN) are the state of the art algorithm for sequential data and are used by Apple's Siri and Google's voice search.
- It is a class of neural networks tailored to deal with temporal data.
- It is the first algorithm that remembers its input, due to an internal memory, which makes it perfectly suited for machine learning problems that involve sequential data.
- An increase in computational power along with massive amounts of data that we now have to work with, and the invention of long short-term memory (LSTM) in the 1990s, has really brought RNNs to the foreground.
- Because of their internal memory, RNN’s can remember important things about the input they received, which allows them to be very precise in predicting what’s coming next.

Recurrent Neural Networks

- This is why they're the preferred algorithm for sequential data like time series, speech, text, financial data, audio, video, weather forcasting, etc.
- In a feed-forward network, the information only moves in one direction — from input layer, through hidden layers, to output layer. The information moves straight through the network and never touches a node twice.
- Feed-forward networks have no memory of the input they receive and are bad at predicting what’s coming next.
- As a feed-forward network only considers the current input, it has no notion of order in time.
- It simply can’t remember anything about what happened in the past except its training.

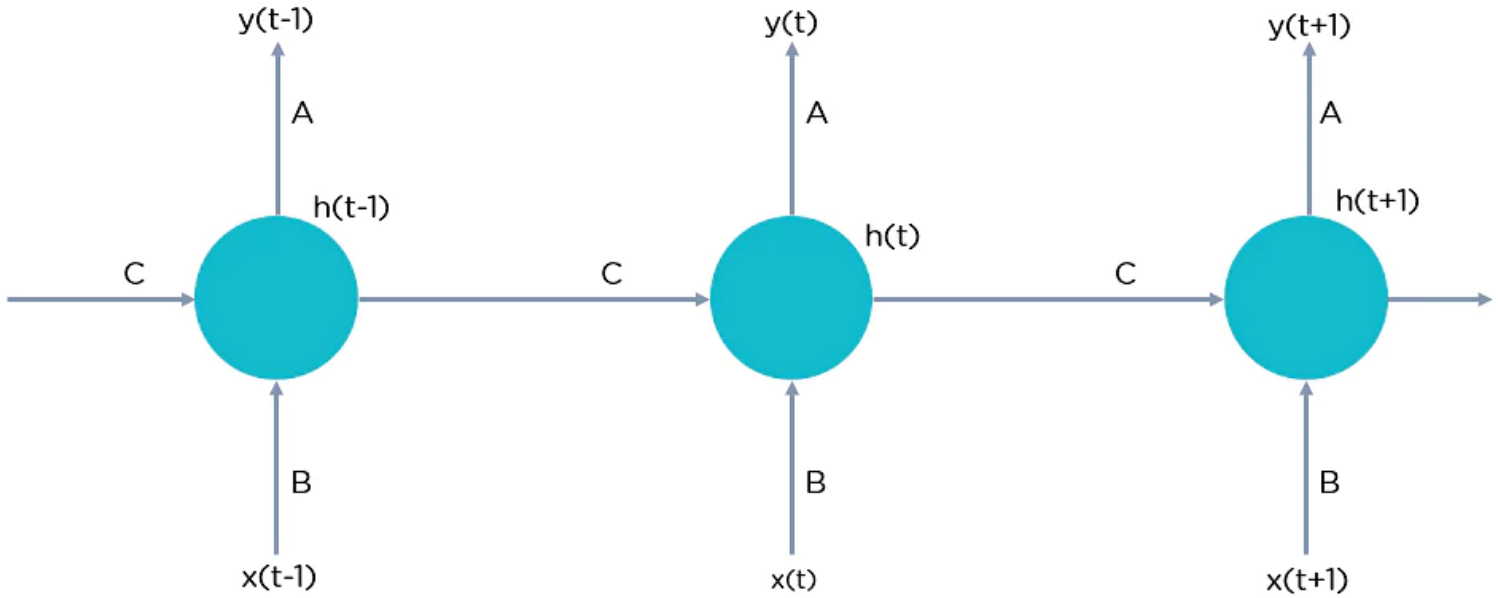
Recurrent Neural Networks

- In a RNN the information cycles through a loop. When it makes a decision, it considers the current input and also what it has learned from the inputs it received previously.
- The two images below illustrate the difference in information flow between a RNN and a feed-forward neural network.



Recurrent Neural Networks

- Here, “x” is input layer, “h” is hidden layer, and “y” is output layer. A, B, and C are the network parameters used to improve the output of the model.
- At any given time t, the current input is a combination of input x_t and x_{t-1}. The output at any given time is fetched back to the network to improve on the output.



$h(t) = f_c(h(t-1), x(t))$

h(t) = new state
f_c = function with parameter c
h(t-1) = old state
x(t) = input vector at time step t

Recurrent Neural Networks

Why Recurrent Neural Networks?

- RNN were created because there were a few issues in the feed-forward neural networks as given below:

  - they cannot handle sequential data

  - they consider only the current input

  - they cannot memorize the previous inputs

- The solution to these issues is RNN that can handle sequential data, accepting the current input data, and previously received inputs.
- They can memorize previous inputs due to their internal memory
- Middle layer ‘h’ can consist of multiple hidden layers, each with its own activation functions and weights and biases.

Recurrent Neural Networks

- RNN standardize different activation functions and weights and biases so that each hidden layer has the same parameters.
- So, instead of creating multiple hidden layers, it creates one and loop over it as many times as required.
- RNNs are used to caption an image by analyzing the activities present.
- Any time series problem, like predicting the prices of stocks in a particular month, can be solved using an RNN.
- Text mining and sentiment analysis can be carried out using an RNN for Natural Language Processing (NLP).
- Given an input in one language, RNNs can be used to translate the input into different languages as output.

Recurrent Neural Networks

- There are four types of Recurrent Neural Networks:

  - One to One: It allows a single input and a single output. It has fixed input and output sizes and acts as a traditional neural network. The one-to-one application can be found in *Image Classification*.

  - One to Many: This type of network has a single input and multiple outputs. An example of this is the image caption. Music generation is also an example.

  - Many to One: This type of network takes a sequence of inputs and generates a single output. Sentiment analysis is a good example of this kind of network where a given sentence can be classified as expressing positive or negative sentiments.

  - Many to Many: This type of network takes a sequence of inputs and generates a sequence of outputs. Machine translation is one of the examples. Name entity recognition is also an example.

Recurrent Neural Networks

There are two issues in standard RNNs:

i. Vanishing Gradient Problem

- Recurrent Neural Networks enable us to model time-dependent and sequential data problems, such as stock market prediction, machine translation, and text generation.
- They are however hard to train because of the gradient problem.
- They suffer from the problem of vanishing gradients. The gradients carry information used in RNN, and when the gradient becomes too small, the parameter updates become insignificant. This makes the learning of long data sequences difficult.

Recurrent Neural Networks

There are two issues in standard RNNs:

ii. Exploding Gradient Problem

- While training a neural network, if the slope tends to grow exponentially instead of decaying, this is called an Exploding Gradient.
- This problem arises when the large error gradients accumulate, resulting in very large updates to the neural network model weights during the training process.
- Long training time, poor performance, and bad accuracy are the major issues in gradient problems.
- The gradient problems can be addressed by using **Long Short-Term Memory Network (LSTMs)** and **Gated Recurrent Unit (GRU).**

Long Short-Term Memory Network

Long Short-Term Memory Network (LSTM).

- Suppose we want to predict the last word in text:

  "The clouds are in the _____."

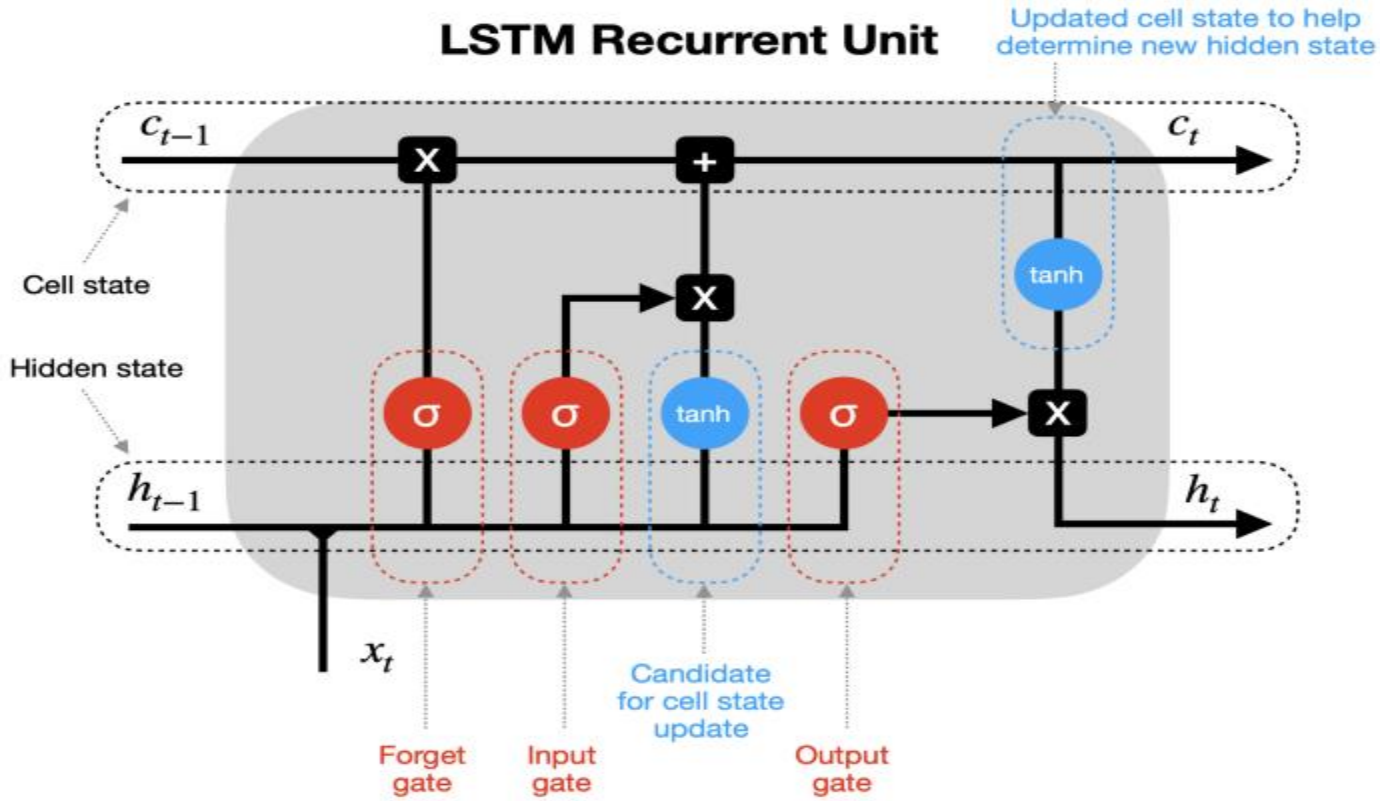
- The most obvious answer to this is "**sky**." We do not need any further context to predict the last word in the above sentence.
- Consider the sentence: "I have been staying in Spain for the last 10 years… I can speak fluent _____."
- The word we predict depends on the previous few words in context. Here, we need the context of Spain to predict the last word in the text, and the most suitable answer to this sentence is "Spanish."
- The gap between the relevant information and the point where it's needed may have become very large. LSTMs help to solve this problem.

Long Short-Term Memory Network

- LSTMs are a special kind of RNN - capable of learning long-term dependencies by remembering the information for long periods is the default behavior.
- Instead of having a single neural network layer, four interacting layers are communicating extraordinarily.
- Symbols used in LSTM:
- *h_{t-1} - hidden state at previous time step (t-1);*

  it is also called short-term memory.
- *C_{t-1} - cell state at previous time step (t-1) it is called long- term memory.*
- *x_t - input vector at current time step (t);*
- *h_t - hidden state at current time step (t);*
- *C_t - cell state at current time step (t);*
- *- concatenation of vectors*
- *- vector pointwise addition*
- x *- vector pointwise multiplication*

Long Short-Term Memory Network



LSTM works in a 3-step process.

Step 1: Decide how much past data It should remember.

- This step decides which information should be omitted from the cell in that particular time step. The sigmoid function determines this. It looks at previous state (h_{t-1}) along with current input x_t. It is also called forget gate.

Long Short-Term Memory Network

- It computes the function:

$$f_t = \sigma(W_f . [h_{t-1}, x_t] + b_f)$$

f_t - forget gate that decides which information to delete that is not important from the previous time step.

Consider the following two sentences:

- Let the output of h_{t-1} be "Alice is good in Physics. John, on the other hand, is good at Chemistry."
- Let the current input x_t be "John plays football well. He told me yesterday over the phone that he had served as the captain of his college football team."
- The forget gate realizes there might be a change in context after encountering the first full stop. It compares with the current input sentence x_t. The next sentence talks about John, so the information on Alice is deleted. The position of the subject is vacated and assigned to John.

Long Short-Term Memory Network

Step 2: Decide how much this unit adds to current state.

- In second step, there are two parts: one is sigmoid function, and other is tanh function.
- In sigmoid function, it decides which values to let through (0 or 1). tanh gives weightage to the values which are passed, deciding their level of importance (-1 to 1).

$$i_t = \sigma(W_i . [h_{t-1}, x_t] + b_i)$$

$$C_t = \sigma(W_C . [h_{t-1}, x_t] + b_C)$$

i_t - input gate that decides which information to let through based on its importance in current time step.

- With the current input x_t, the input gate analyzes the important information - John plays football, and the fact that he was the captain of his college team is important.
- "He told me yesterday over the phone" is less important; hence it's forgotten. This process of adding some new information can be done via the input gate.

Long Short-Term Memory Network

Step 3: Decide what part of current cell state makes it to the output.

- This step decides what the output will be. First, we run a Sigmoid layer that decides what parts of the cell state make it to the output. Then, we put the cell state through tanh to push the values to be between -1 and 1 and multiply it by the output of the Sigmoid gate.

$$o_t = \sigma(W_o.[h_{t-1}, x_t] + b_o)$$
$$h_t = o_t * tanh(C_t)$$

o_t - output gate that allows to pass in the information to impact the output in the current time step.

- Consider the example to predict next word in sentence: "John played tremendously well against the opponent and won for his team. For his contributions, brave ____ was awarded player of the match."
- There could be many choices for the empty space. The current input brave is an adjective, and adjectives describe a noun. So, "John" could be the best output after brave.

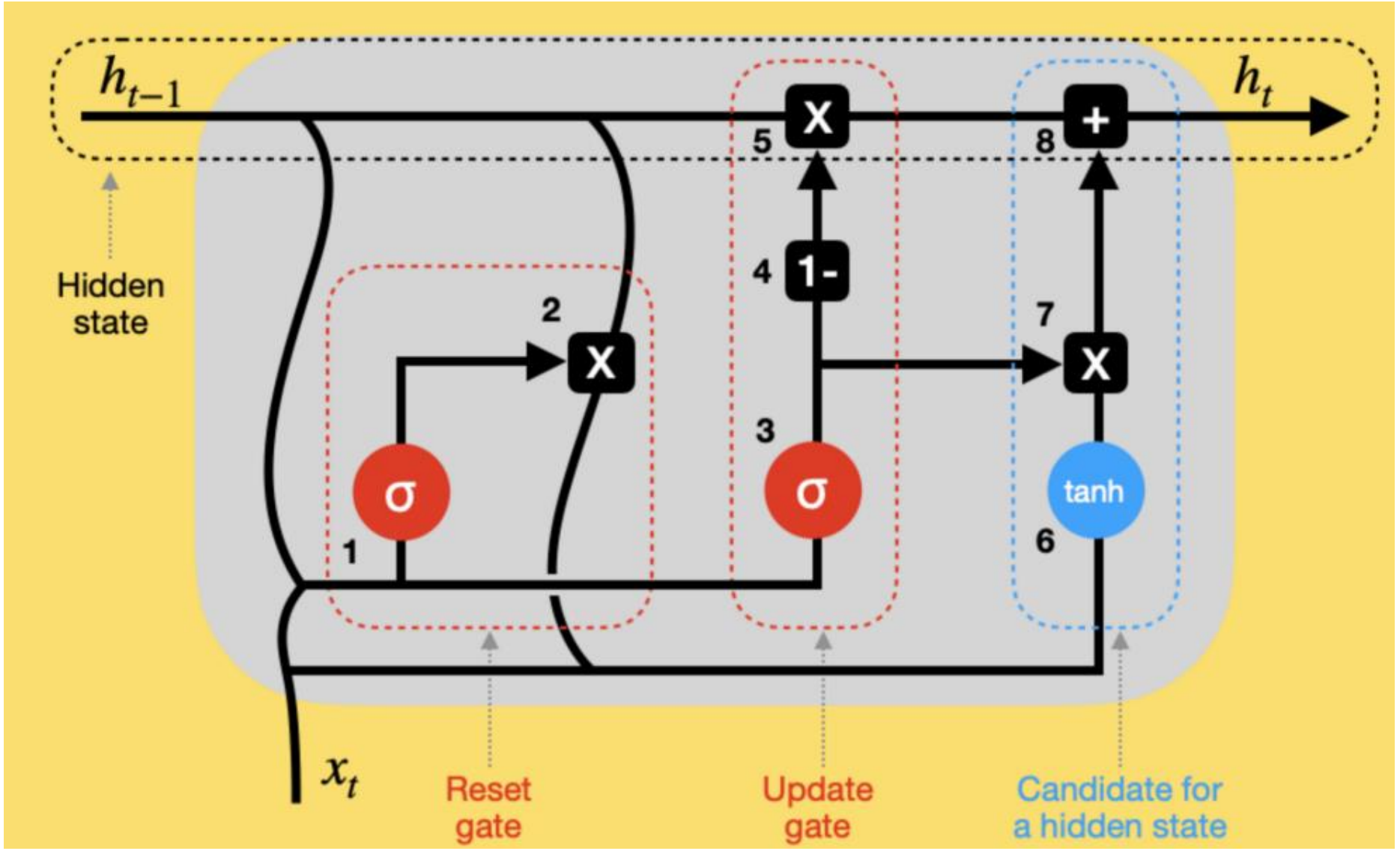
Long Short-Term Memory Network

- In summary, an LSTM module has a cell state and three gates that provides RNNs with the power to selectively learn, unlearn, or retain information from each of units.
- Cell state in LSTM helps the information to flow through the units without being altered by allowing only a few linear interactions.
- Each unit has an input, output and a forget gate which can add or remove the information to the cell state.
- The forget gate decides which information from the previous cell state should be forgotten for which it uses a Sigmoid function.
- The input gate controls the information flow to the current cell state using a point-wise multiplication operation of ‘Sigmoid’ and ‘tanh’ respectively.
- Finally, the output gate decides which information should be passed on to the next hidden state.

Gated Recurrent Units (GRU)

- Gated Recurrent Unit (GRU) aims to solve the vanishing gradient problem which comes with a standard recurrent neural network.
- To solve the vanishing gradient problem of a standard RNN, GRU uses **update gate** and **reset gate**, which are two vectors that decide what information should be passed to the output.
- The special thing about GRUs 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.
- 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.

Gated Recurrent Units (GRU)



Gated Recurrent Units (GRU)

- *h_{t-1} - hidden state at previous time step (t-1);*
  it is also called memory.
- *x_t - input vector at current time step (t);*
- *h_t - hidden state at current time step (t);*
- *- concatenation of vectors*
- *+ - vector pointwise addition*
- x - *vector pointwise multiplication*

- **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 sigmoid function ranges between 0 and 1, step1 sets which values should be discarded (0), remembered (1), or partially retained (between 0 and 1). Step2 resets previous hidden state multiplying it with outputs from step1.

Gated Recurrent Units (GRU)

- **3–4–5 Update gate** - step3 may seem analogous to step1, but the 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 (step4) and multiply it by the previous hidden state (step5). 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 step2, 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 (step6). Then the hidden state candidate is multiplied by the results of an update gate (step7) and added to previously modified h_{t-1} to form new hidden state h_t.
- The process repeats for timesteps t+1, t+2, etc., until the recurrent unit processes the entire sequence.

Gated Recurrent Units (GRU)

i. Update gate: We start with calculating the **update gate z_t** for time step **t** as follows:

$$z_t = \sigma(W_z x_t + U_z h_{t-1})$$

- When x_t is plugged into the network unit, it is multiplied by its own weight W_z.
- The same goes for h_{t-1} that holds the information for the previous *t-1* units and is multiplied by its own weight U_z.
- Both results are added together and a Sigmoid function is applied to squash the result between 0 and 1.
- This **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 really powerful because the model can decide to copy all the information from the past and eliminate the risk of vanishing gradient problem.

Gated Recurrent Units (GRU)

ii. Reset gat: This gate is used to decide how much of the past information to forget from the model. It is calculated:

$$r_t = \sigma(W_r x_t + U_r h_{t-1})$$

- This formula is same as the one for update gate, but the difference comes in the weights and the gate's usage.

iii. Current memory content: we start with the usage of reset gate. We introduce a new memory content that uses the reset gate to store the relevant information from past, as given below:

$$h'_t = tanh(W x_t + r_t \odot U h_{t-1})$$

here ⊙ refers to elementwise multiplication.

Suppose we have a sentiment analysis problem for determining one's opinion about a book from a review Renu wrote. The text starts with "This is a fantasy book which illustrates…" and after a couple paragraphs ends with "I didn't quite enjoy the book because I think it captures too many details." To determine the overall level of satisfaction from the book we only need the last part of the review. In that case as the neural network approaches to the end of text it will learn to assign r_t vector close to 0, washing out the past and focusing only on the last sentences.

Gated Recurrent Units (GRU)

iv. Final memory at current time step

- As the last step, the network needs to calculate h_t - vector that holds information for the current unit and passes it down to the network.
- In order to do that the update gate is needed that determines what to collect from the current memory content - h'_t and what from the previous steps - h_{t-1}, as given below:

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

here $\odot$ refers to elementwise multiplication.

For the example about the book review. This time, the most relevant information is positioned in the beginning of the text. The model can learn to set the vector z_t close to 1 and keep a majority of the previous information. Since z_t will be close to 1 at this time step, (1-z_t) will be close to 0 which will ignore big portion of the current content (in this case the last part of the review which explains the book plot) which is irrelevant for our prediction.