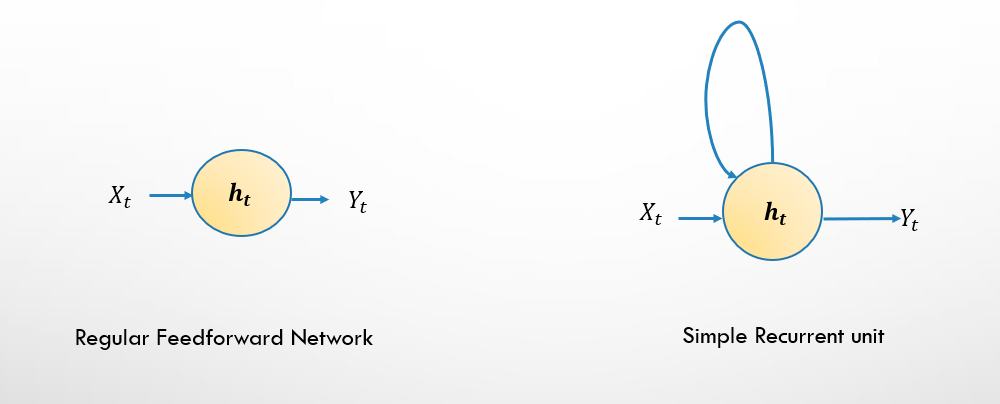
So, if we are trying to use such data to predict any reasonable output, we need a network with access to some prior knowledge about the data to understand it completely. That is where Recurrent neural networks come to the rescue. To understand what memory is in RNNs, what the recurrence unit in RNNs is, and how they store information from the previous sequence, let us first understand the architecture of RNNs.

A Recurrent Neural Network (RNN) is a neural network that processes sequential data. It has a unique architecture allows it to maintain a "memory" of previous inputs and use that information to make predictions about future inputs. The basic structure of an RNN consists of a single hidden layer with recurrent connections. The input is fed into the network one time step at a time, and the output of the previous time step is fed back into the network as input for the current time step. This allows the network to maintain a "memory" of previous inputs and use that information to make predictions about future inputs. The time step concept refers to the fact that the input is processed sequentially, one time-step at a time, and the output of each time-step is used as input for the next time step.

**THE ARCHITECTURE OF RECURRENT NEURAL NETWORK:**

The architecture of an RNN can be conceptualized by considering a basic feedforward neural network with a single hidden layer. Introducing a feedback connection from the hidden layer to itself allows the network to access information from the previous time step, acting as a memory mechanism. This feedback loop permits the transfer of information from one step to the next, enhancing the network's memory.



A diagram of a network diagram

Description automatically generated





In RNNs, the function f, representing hidden layer non-linearities, can be sigmoid, tanh, or ReLU, serving as a hyperparameter.

Viewing an RNN as multiple copies of a feedforward network, each conveying a message to a successor, unfolds the recurrent connections over time, resembling a feedforward neural network with multiple hidden layers. To illustrate, imagine a sequence of length 5. Unfolding the recurrent neural network in time and removing recurrent connections yields a feedforward neural network with five hidden layers.

A diagram of a diagram

Description automatically generated

Where,

ht - Current state i. e. state at time step t

ht-1 - previous state i. e. state at time step t – 1

Wh - Weight at the recurrent neuron

Wx - Weight at input neuron

Yt - Output at time step t

It is like ho is the input, and each Xt is just some additional control signal at each step. We can see that hidden to hidden weight Wh is repeated at every layer. So, it is like a deep network with the same shared weights between each layer. Similarly, Wx is shared between the five Xs going into hidden layers.

The recurrent neural network works as follows:

The operation of an RNN involves converting independent activations into dependent activations by employing the same weights and biases across all layers. This parameter sharing signifies performing the same task at each step with varying inputs, reducing the overall parameters to be learned. Consequently, the complexity of memorizing previous outputs diminishes as each output serves as input to the subsequent hidden layer. This consolidation of weights and biases forms a unified recurring structure within the RNN. These layers of the same weights and biases merge into one single recurring structure.

The above diagram has outputs at each time step, but this may not be necessary depending on the task. For example, when predicting the sentiment of a sentence, we may only care about the final output, not the prediction after each word. Similarly, we may not need inputs at each time step. The main feature of an RNN is its hidden state, which captures some information about a sequence.

**VANISHING AND EXPLODING GRADIENTS:**

Backpropagation Through Time (BPTT):

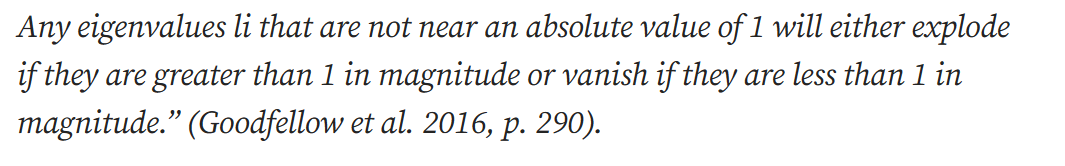
Backpropagation Through Time (BPTT) is an algorithm used to train RNNs. It is a variant of the standard backpropagation algorithm used in feedforward neural networks. BPTT computes gradients by propagating the gradient backwards in time. For each time step, the gradients are computed and accumulated. These accumulated gradients are then used to update the weights of the RNN. Truncated BPTT is a common approach where the backward pass is limited to a fixed number of time steps. This saves computational time and helps mitigate the vanishing/exploding gradient problem.

Issues with BPTT

BPTT has some issues that can make it challenging to train RNNs. One issue is that it can be computationally expensive, especially when processing long sequences. Another issue is that it can suffer from vanishing and exploding gradients, making training unstable or very slow. More advanced RNN architectures, such as Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU), have been developed to address these issues.

Vanishing and Exploding Gradients

Vanishing and exploding gradients are common issues that arise when training RNNs. Vanishing gradients occur when they become very small as they propagate backwards through the network, making it difficult to update the network weights. Exploding gradients occur when the gradients become very large, causing the weights to update too much and leading to numerical instability.



Gradient clipping is an effective method to mitigate the exploding gradients issue. The method is very simple; if a gradient becomes too large it is rescaled to a smaller value. The algorithm for gradient clipping works as in the equation below. where g is the gradient, c is a hyperparameter and ||g|| is the norm of g.



Skip connections enable the network to take a short-cut by skipping some of the layers if it deems it optimal during training. In practice this allows the network to bypass parts of the network where vanishing gradients occurs, passing on gradients from earlier layers to later layers in the network.

A diagram of a diagram

Description automatically generated

To implement this in Keras, use the functional API to link the layers. Below is an example of the use of skip connections.

A screenshot of a computer code

Description automatically generated