Recurrent Neural Network (RNN)

Recurrent Neural Network (RNN) in TensorFlow

A recurrent neural network (**RNN**) is a kind of artificial neural network mainly used in **speech recognition** and **natural language processing** (NLP). RNN is used in deep learning and in the development of models that imitate the activity of neurons in the human **brain**.

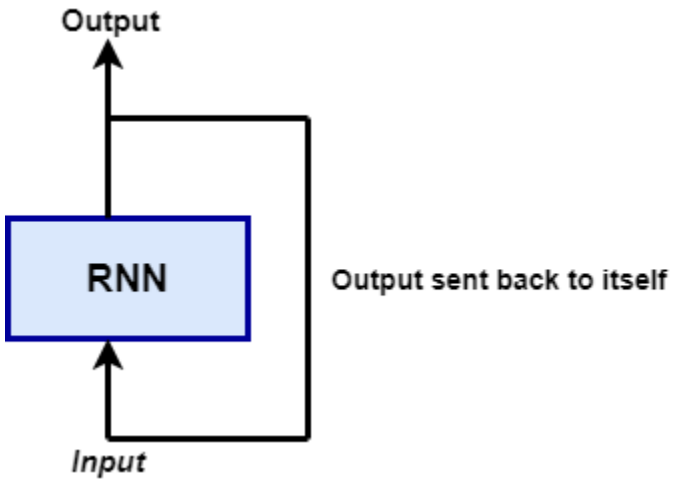
Recurrent Networks are designed to **recognize patterns** in sequences of data, such as **text, genomes, handwriting, the spoken word,** and **numerical** time series data emanating from sensors, stock markets.

A recurrent neural network looks similar to a traditional neural network except that a memory-state is added to the neurons. The computation is to include a simple memory.

The recurrent neural network is a type of deep learning-oriented algorithm, which follows a sequential approach. In neural networks, we always assume that each input and output is dependent on all other layers. These types of neural networks are called recurrent because they sequentially perform mathematical computations.

In the neural network (Feed Forward) output is independent means no output is dependent on previous output so, it fails to learn about the characteristics of the previous output.

But in the same case example: sentence 2nd of the poem is dependent on sentence 1st of the poem so in such case, RNN is used, where RNN is able to learn characteristics of previous output. Hence able to find sequences.



Application of RNN

RNN has multiple uses when it comes to predicting the future. In the financial industry, RNN can help predict stock prices or the sign of the stock market direction (i.e., **positive** or **negative**).

RNN is used for an autonomous car as it can avoid a car accident by anticipating the route of the vehicle.

RNN is widely used in **image captioning, text analysis, machine translation,** and **sentiment analysis**. **For example,** one should use a movie review to understanding the feeling the spectator perceived after **watching the movie**. Automating this task is very useful when the movie company can not have more time to review, consolidate, label, and analyze the reviews. The machine can do the job with a higher level of accuracy.

Following are the 3 major application of RNN:

1. Machine Translation
2. Speech Recognition
3. Sentiment Analysis

Limitations of RNN

RNN is supposed to carry the information in time. However, it is quite challenging to propagate all this information when the time step is too long. When a network has too many deep layers, it becomes untrainable. This problem is called: vanishing gradient problem.

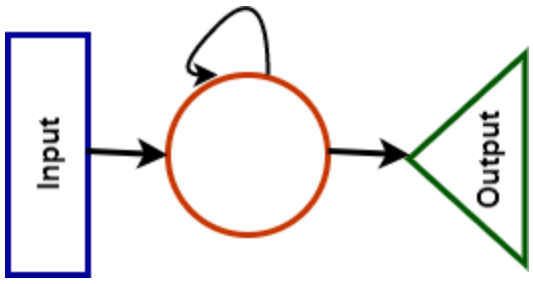
If we remember, the neural network updates the weight use of the gradient descent algorithm. The gradient grows smaller when the network progress down to lower layers.

The gradient stays constant, meaning there is no space for improvement. The model learns from a change in its gradient; this change affects the network's output. If the difference in the gradient is too small (i.e., the weight change a little), the system can't learn anything and so the output. Therefore, a system facing a vanishing gradient problem cannot converge towards the right solution.

The recurrent neural will perform the following.

The recurrent network first performs the conversion of independent activations into dependent ones. It also assigns the same weight and bias to all the layers, which reduces the complexity of RNN of parameters. And it provides a standard platform for memorization of the previous outputs by providing previous output as an input to the next layer.

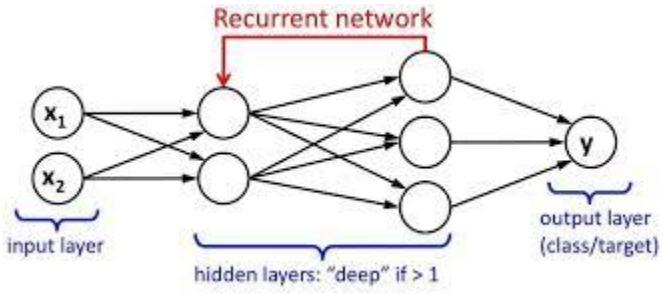
These three layers having the same weights and bias, combine into a single recurrent unit.



1. For calculating the current state-
2. **$h_t = f(h_{t-1}, X_t)$**
3. Where h_t= **current state**
4. H_{t-1}= **previous state**
5. X_t= **input state**
6. To apply the activation function tanh, we have-
7. **ht = tanh ($W_{hh}h_t$-1+ $W_{xh}X_t$)**
8. Where:
9. W_{hh} = **weight of recurrent neuron and,**
10. W_{xh} = **weight of the input neuron**
11. **The formula for calculating output:**
12. **$Y_t = W_{hy}h_t$**

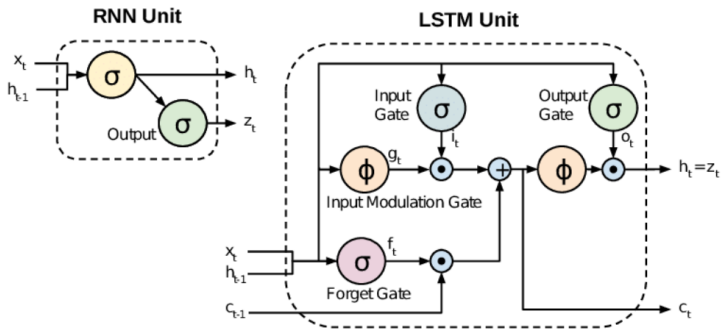
Training through RNN

- The network takes a single time-step of the input.
- We can calculate the current state through the current input and the previous state.
- Now, the current state through h_{t-1} for the next state.
- There is n number of steps, and in the end, all the information can be joined.
- After completion of all the steps, the final step is for calculating the output.
- At last, we compute the error by calculating the difference between actual output and the predicted output.
- The error is backpropagated to the network to adjust the weights and produce a better outcome.



The solution to overcome the issue of RNN is to use LSTM

LSTMs are often referred to as fancy RNNs. Vanilla RNNs do not have a cell state. They only have hidden states and those hidden states serve as the memory for RNNs.
Meanwhile, LSTM has both cell states and hidden states. The cell state has the ability to remove or add information to the cell, regulated by " additional gates". And because of this "cell", LSTM able to handle the long-term dependency

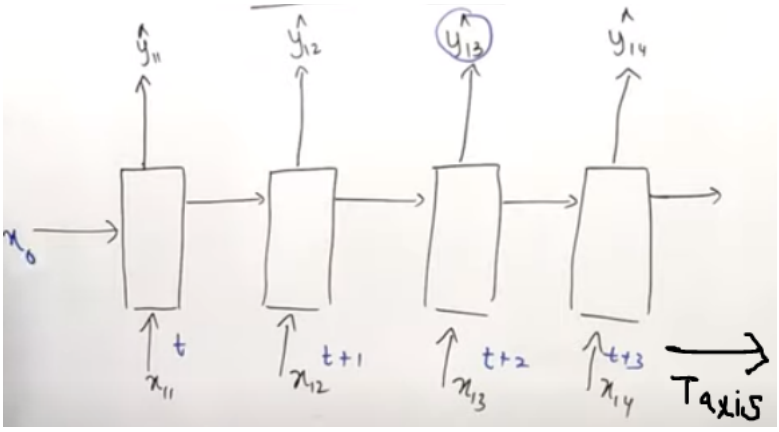


The LSTM network can be organized into an architecture called the Encoder-Decoder LSTM that allows the model to be used to both support variable-length input sequences and to predict or output variable-length output sequences.

Bi-Directional RNN

Uni-Directional

To get output y13, all information needed new and previous.

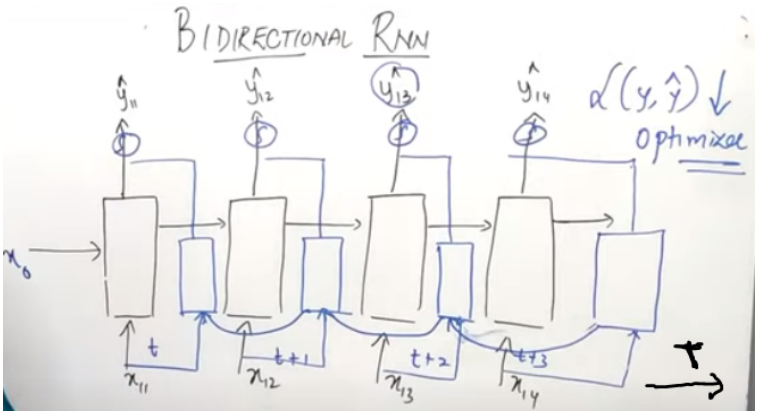


Now, y13 can be also be depended on the future event.

Example:

I like eating _____

The 3 words are dependent on the future word.



An extra RNN is added with the Uni-Directional RNN. And the connections are made as shown to provide future information to the previous cell.

Input from down is given to the next cell (the next cell pass that info left to right). The result of that cell is then given to the result of the previous cell.

Cons

Bi-Directional is slow

Fails with time-bound i.e in speech Recognization as words come afterword spoken.

 # word2vec

Word2vec is a technique for natural language processing published in 2013. The word2vec algorithm uses a neural network model to learn word associations from a large corpus of text. Once trained, such a model can detect synonymous words or suggest additional words for a partial sentence. As the name implies, word2vec represents each distinct word with a particular list of numbers called a vector. The vectors are chosen carefully such that a simple mathematical function (the cosine similarity between the vectors) indicates the level of semantic similarity between the words represented by those vectors.

Word2vec can utilize either of two model architectures to produce a distributed representation of words: continuous bag-of-words (CBOW) or continuous skip-gram. In the continuous bag-of-words architecture, the model predicts the current word from a window of surrounding context words. The order of context words does not influence prediction (bag-of-words assumption). In the continuous skip-gram architecture, the model uses the current word to predict the surrounding window of context words. The skip-gram architecture weighs nearby context words more heavily than more distant context words. According to the authors' note, CBOW is faster while skip-gram does a better job for infrequent words.