**Q2. Applications of Recurrent Neural Networks (RNN)**

RNNs are powerful Machine Learning models and have found use in a wide range of areas. It is distinctly different from CNN models like GoogleNet.

RNNs are widely used in following domains/ applications:

* Prediction problems
* Language Modelling and Generating Text
* Machine Translation
* Speech Recognition
* Generating Image Descriptions
* Video Tagging
* Text Summarization
* Call Center Analysis
* Face detection, OCR Applications as Image Recognition
* Music composition

**Prediction problems**

RNNs are generally useful in working with sequence prediction problems. Sequence prediction problems come in many forms and are best described by types of inputs and outputs it supports.

Sequence prediction problems include:

**One-to-Many:** In this type of problem, an observation is mapped as input to a sequence with multiple steps as an output.

**Many-to-One:** Here a sequence of multiple steps as input are mapped to a class or quantity prediction.

**Many-to-Many:**  A sequence of multiple steps as input are mapped to a sequence with multiple steps as output. The Many-to-Many problem is often referred to as sequence-to-sequence, or seq2seq.

The problem with Recurrent Neural Networks was that they were traditionally difficult to train. The Log Short Term Memory, or LSTM, network is one of the most successful RNN because it solves the problems of training a recurrent network and in turn has been used on a wide range of applications. RNNs and LSTMs have received the most success when working with sequences of words and paragraphs, generally in the field of natural language processing (NLP). They are also used as generative models that produce a sequence output, not only with text, but on applications such as generating handwriting.

Recurrent Neural Networks are not appropriate for tabular datasets such as those in a csv file or spreadsheet. They are also not appropriate for dataset containing image data.

RNNs and LSTMs on being tested with time series forecasting problems, produced poor results. Autoregression methods, even linear methods often perform much better. Even simple MLP’s applied on the same data performed better than LSTMs.

**Language Modelling and Generating Text**

1. Language Modelling and Generating Text

Taking a sequence of words as input, we try to predict the possibility of the next word. This can be considered to be one of the most useful approaches for translation since the most likely sentence would be the one that is correct. In this method, the probability of the output of a particular time\_step is used to sample the words in the next iteration.

**Machine Translation**

1. Machine Translation

RNNs in one form or the other can be used for translating text from one language to other. Almost all of the Translation systems being used today use some advanced version of an RNN. The input can be the source language and the output will be in the target language which the user wants.

Currently one of the most popular and prominent Machine translation application is Google Translate. There are even numerous custom recurrent neural network applications used to refine and confine content by various platforms. Ecommerce platforms like Flipkart, Amazon, and eBay make use of machine translation in many areas and it also help with the efficiency of search results.

**Speech Recognition**

1. Speech Recognition

RNNS can be used for predicting phonetic segments considering sound waves from a medium as an input source. The set of inputs consists of phoneme or acoustic signals from an audio which are processed in a proper manner and taken as inputs. The RNN network will compute the phonemes and then produce a phonetic segment along with the likelihood of output. The steps used in speech recognition are as follows:

* The Input data is first processed and recognized through a neural network. The result consists of a varied collection of input sound waves.
* The Information contained in the sound wave is further classified by intent and through keywords related to the query.
* Then input sound waves are classified into phonetic segments and are pieced together into cohesive words using an RNN application. The output consists of a pattern of phonetic segments put together into a singular whole in a logical manner.

**Generating Image Descriptions**

1. Generating Image Descriptions

A combination of CNNs and RNNs are used to provide a description of what exactly is happening inside an image. CNN does the segmentation part and RNN the uses the segmented data to recreate the descriptions.

**Video Tagging**

1. Video Tagging

RNNs can be used for video search where we can do image description of a videos divided into numerous frames.

**Text Summarization**

1. Text Summarization

This application can provide major help in summarizing content from literatures and customizing them for delivery within applications which cannot support large volumes of text. For example, if a publisher wants to display the summary of one of his books on its backpage to help the readers get an idea of the content present within, Text Summarization would be helpful.

**Call Center Analysis**

1. Call Center Analysis

This can be considered as one of the major applications of RNNs in the field of audio processing. Customer metrics are measured on the output of the call rather than the call itself. However, analyzing the call itself will provide businesses with the solutions to why the support staff succeeded and what were the steps taken in resolving their customer issue. This Learning can then be studied and reapplied to other similar scenarios or to train new support representatives. Hence the entire process can be automated based on the synthesized speech from the call for analysis purpose. Such synthesized speech can be further taken as an input to a tone analysis algorithm to measure the emotions and sentiments related to various parts of the conversation. This would help the business identify when the customer is satisfied with the service and support and when a customer has faced issues. Although these things can be done by involving simple human observations, automation will help business quantify the results and derive insights from them for future knowledge since all the output data would be stored in a database

**Face Detection, OCR Application as Image Recognition**

1. Face Detection, OCR Applications as Image Recognition

Image recognition is one of the major applications of computer vision. It is also one of the most accessible form of RNN to explain.

In its Core, the algorithm is designed to consider one unit of image as input and produce the description of the image in the form of multiple groups of output.

The Image recognition framework includes:

**Convolutional Neural Network** that processes the image and recognizes the features of the pictures,

**Recurrent Neural Networks** that makes use of the known features to make sense of the image and put together a proper description of the input image.

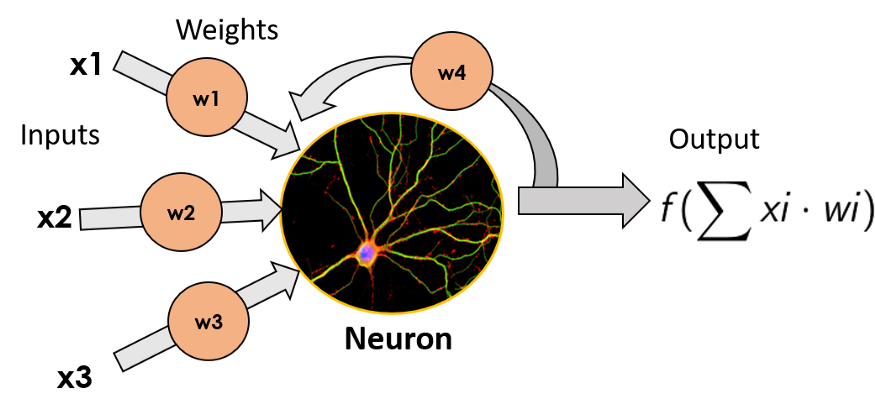
There are numerous benefits of Image recognition in the field of business – it can be used as a streamlining tool to make it easier for the customer to operate with the service, find relevant images, navigate through information, and make purchases and furthermore adding to the security of the customer. The Most prominent industries that are making use of image recognition are Search engines, eCommerce, Social Media, Security and Networking.

**Other Applications of RNNs**

A few other applications of RNNs include:

* Time series anomaly detection
* Rhythm learning
* Music composition
* Handwriting recognition
* Grammar learning
* Human action recognition
* Protein Homology Detection
* Predicting subcellular localization of proteins
* Several prediction tasks in the area of business process management and Prediction in medical care pathways.

**3Q. Write about how the inputs are selected for LSTM/RNN models. Explain in terms of timesteps, samples, and features.**



A recurrent neuron, where the output data is multiplied by a weight and fed back into the input

**What is RNN?**

**R**ecurrent **N**eural **N**etwork is basically a generalization of feed-forward neural network *that has an internal memory*. RNNs are a special kind of neural networks that are designed to effectively deal with **sequential data**. This kind of data includes **time series** (a list of values of some parameters over a certain period of time) **text documents**, which can be seen as a sequence of words, or **audio,**which can be seen as a sequence of sound frequencies over time.  
RNN is recurrent in nature as it performs the same function for every input of data while the output of the current input depends on the past one computation. For making a decision, it considers the current input and the output that it has learned from the previous input.

*Cells that are a function of inputs from previous time steps are also known as***memory cells***.*

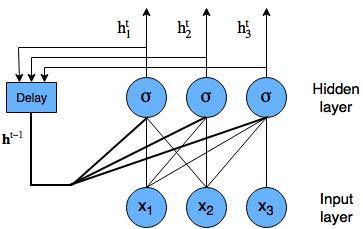
Unlike feed-forward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. In other neural networks, all the inputs are independent of each other. But in RNN, all the inputs are related to each other.

**Why RNN?**

The basic challenge of classic feed-forward neural network is that it has no memory, that is, each training example given as input to the model is treated independent of each other. In order to work with sequential data with such models — you need to show them the entire sequence in one go as one training example. This is problematic because number of words in a sentence could vary and more importantly this is not how we tend to process a sentence in our head.

When we read a sentence, we read it word by word, keep the prior words / context in memory and then update our understanding based on the new words which we incrementally read to understand the whole sentence. This is the basic idea behind the RNNs — they iterate through the elements of input sequence while maintaining a internal “state”, which encodes everything which it has seen so far. The “state” of the RNN is reset when processing two different and independent sequences.

Recurrent neural networks are a special type of neural network where the outputs from previous time steps are fed as input to the current time step.

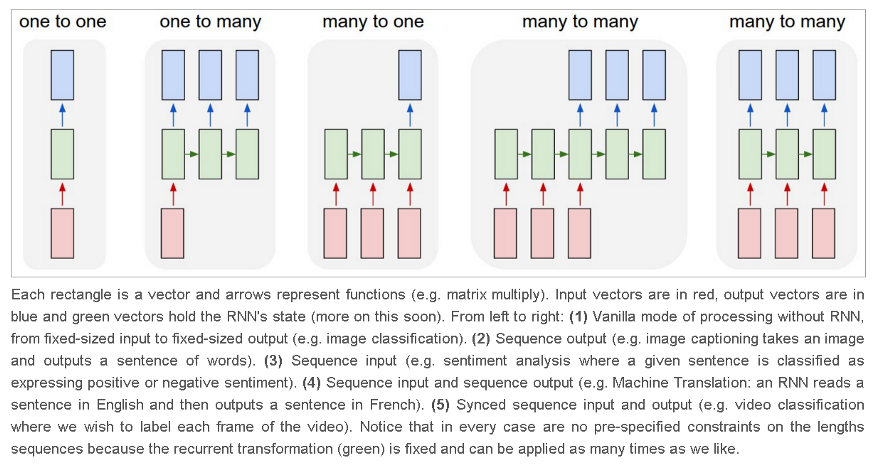


Basic Recurrent neural network with three input nodes

The way RNNs do this, is by taking the output of each neuron (input nodes are fed into a hidden layer with sigmoid or tanh activations), and feeding it back to it as an input. By doing this, it does not only receive new pieces of information in every time step, but it also adds to these new pieces of information a w̲e̲i̲g̲h̲t̲e̲d̲ v̲e̲r̲s̲i̲o̲n̲ of the **previous output**. As you can see the hidden layer outputs are passed through a conceptual delay block to allow the input of **h** ᵗ⁻¹into the hidden layer. What is the point of this? Simply, the point is that we can now model time or sequence-dependent data.  
This makes these neurons have a kind of **“memory”** of the previous inputs it has had, as they are somehow quantified by the output being fed back to the neuron.

**Different types of RNN’s**

The core reason that recurrent nets are more exciting is that they allow us to operate over *sequences* of vectors: Sequences in the input, the output, or in the most general case both. A few examples may make this more concrete:



The problem with RNNs is that as time passes by and they get fed more and more new data, **they start to “*forget*”** about the previous data they have seen *(vanishing gradient problem)*, as it gets **diluted** between the new data, the transformation from activation function, and the weight multiplication. This means they have a **good *short term memory***, but a slight problem when trying to remember things that have happened a while ago (data they have seen many time steps in the past).  
The more time steps we have, the more chance we have of back-propagation gradients either accumulating and exploding or vanishing down to nothing.  
Consider the following representation of a recurrent neural network:

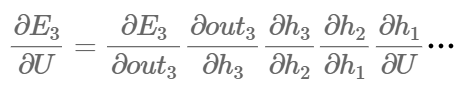
https://miro.medium.com/max/240/1*cL2HAU5Q9qcwD-LKjgPdWw.png

https://miro.medium.com/max/328/1*pTQkhzc6_jnfiE-Lkco9sw.png

Here, **ht**is the new state (*current time stamp)*, **ht**₋₁is the previous state (*previous time stamp)*while **xₜ**is the current input.  
Where ***U***and ***V***are the weight matrices connecting the inputs and the recurrent outputs respectively. We then often will perform a softmax of all **hₜ**the outputs. Notice, however, that if we go back three time steps in our recurrent neural network, we have the following:

https://miro.medium.com/max/659/1*qt7iz4AT7ESBLZmZxuQPXA.png

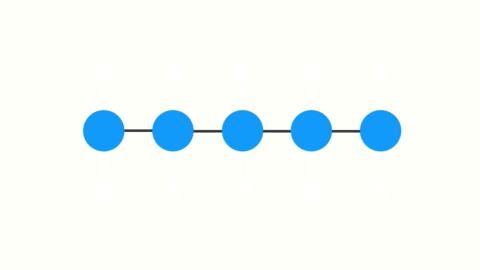
From the above you can see, as we work our way back in time, we are essentially adding deeper and deeper layers to our network. This causes a problem — consider the gradient of the error with respect to the weight matrix ***U*** during back-propagation through time, it looks something like this:



The equation above is only a rough approximation of what is going on during *back-propagation through time.* Each of these gradients will involve calculating the gradient of the sigmoid function. The problem with the sigmoid function occurs when the input values are such that the output is close to either 0 or 1 — at this point, the gradient is very small (saturating).

**For ex**:- Lets say the value decreased like 0.863 →0.532 →0.356 →0.192 →0.117 →0.086 →0.023 →0.019..  
you can see that there is no much change in last 3 iterations.

It means that when you multiply many sigmoid gradients together you are multiplying many values which are potentially much less than zero — this leads to a vanishing gradient problem.



Gradients shrink as it back-propagates through time

The gradient values will exponentially shrink as it propagates through each time step. Because the gradient will become basically zero when dealing with many prior time steps, the weights won’t adjust to take into account these values, and therefore the network won’t learn any relationships separated by a long significant periods of time. So, Vanishing gradient problem results in long-term dependencies being ignored during training.

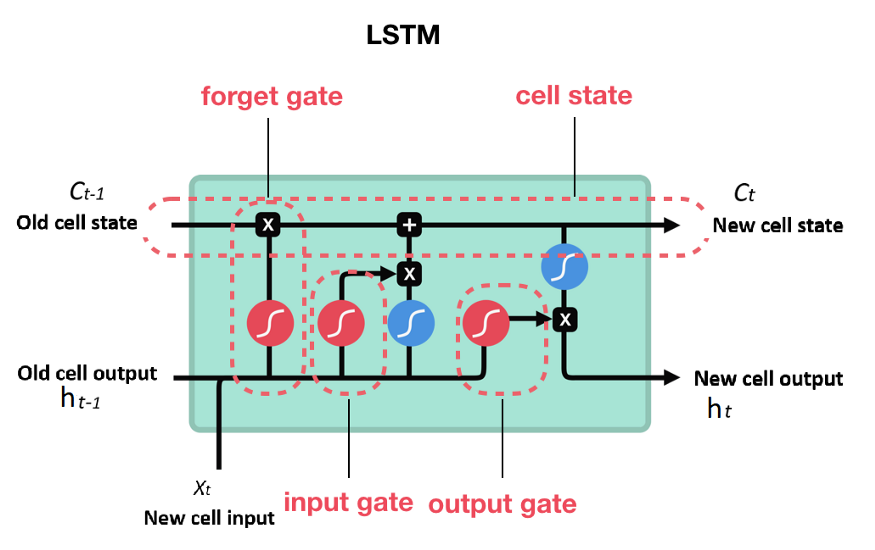
# **Enhancing our memory — Long Short Term Memory Networks (LSTM)**

Long-Short Term Memory networks or**LSTMs** are a variant of RNN that solve the Long term memory problem of the former.

They have a more **complex cell structure** than a normal recurrent neuron, that allows them to better regulate how to **learn or forget**efficientlyfrom the different input sources.

The key to LSTMs is the **cell state**(cell memory), the horizontal line running through the top of the diagram, through which the information flows along and the internal mechanism called **gates**that can regulate the flow of information.  
The cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions.

*Cell State*basically e*ncodes the information of the inputs (relevant info.) that have been observed up to that step (at every step).*



Representation of an LSTM cell

**Cell state** is a memory of the LSTM cell and **hidden state**(cell output) is an output of this cell.

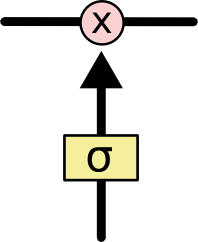
*Cells do have internal cell state, often abbreviated as “c”, and cells output is what is called a “hidden state”, abbreviated as “h”.  
Regular RNNs have just the hidden state and no cell state. Therefore, RNNs have difficulty of accessing information from a long time ago.*

Note: Hidden state is an output of the LSTM cell, used for Prediction. It contains the information of previous inputs (from cell state/memory) along with current input (decided according which context is important).

**Hidden state (hₜ ₋ ₁) and cell input (xₜ) data is used to control what to do with memory (cell state) cₜ : to forget or to write new information.**

We decide what to do with memory knowing about previous cell output (hidden state) and current input and we do this using gates.

**Gates** are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a point-wise multiplication operation.



LSTM Gate

The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.  
A value of zero means “let nothing through,” while a value of one means “let everything through!”

*An LSTM has three of these gates, to protect and control the cell state.*

These gates can learn which data in a sequence is important to keep or throw away. By doing that, it can pass relevant information down the long chain of sequences to make predictions.

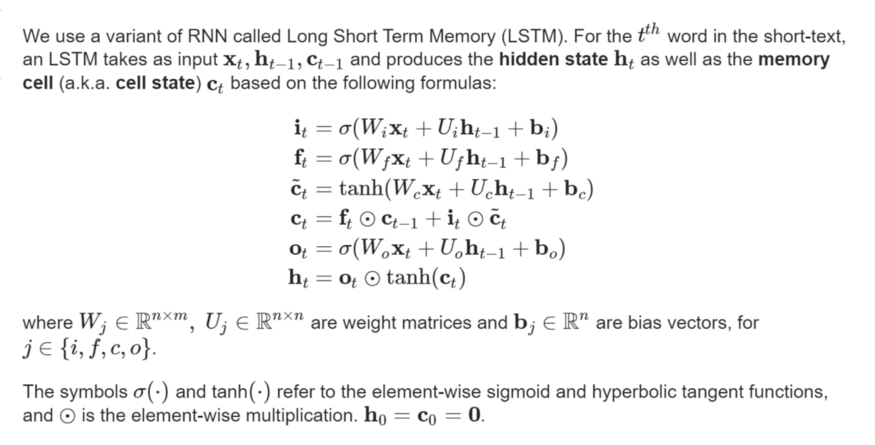
An LSTM neuron can do this learning by incorporating acell state and three different gates: the input gate, the forget gate and the output gate. In each time step, the cell can decide what to do with the state vector: read from it, write to it, or delete it, thanks to an explicit gating mechanism.  
With the **input gate**, the cell can decide whether to update the cell state or not. With the **forget gate** the cell can erase its memory, and with the **output gate** the cell can decide whether to make the output information available or not.

LSTMs also mitigate the problems of **exploding and vanishing gradients.**

To reduce the vanishing (and exploding) gradient problem, and therefore allow deeper networks and recurrent neural networks to perform well in practical settings, there needs to be a way to reduce the multiplication of gradients which are less than zero.

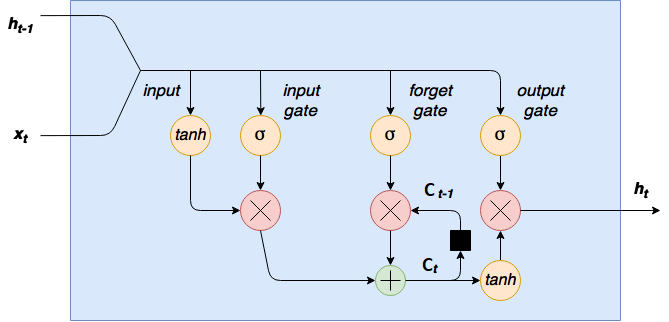
The LSTM cell is a specifically designed unit of logic that will help reduce the vanishing gradient problem sufficiently to make recurrent neural networks more useful for long-term memory tasks i.e. text sequence predictions.  
The way it does so is by creating an internal memory state which is simply added to the processed input, which greatly reduces the multiplicative effect of small gradients. The time dependence and effects of previous inputs are controlled by an interesting concept called a forget gate, which determines which states are remembered or forgotten. Two other gates, the input gate and output gate, are also featured in LSTM cells.

Here’s a brief summary of all the internal formulation and working of different gates,cell state,hidden state and current input, explained through mathematical formulas, referenced from a research paper <https://arxiv.org/abs/1603.03827> (~LSTM for text classification):



LSTM summarized

Let’s first have a look at LSTM cell more carefully.



LSTM cell another view

The data flow is from left-to-right in the diagram above, with the current input **xₜ**and the previous cell output **hₜ**₋₁concatenated together and entering the top “data rail”. The **long-term memory** is usually called the **cell state Ct**. The looping arrows indicate recursive nature of the cell. This allows information from previous intervals to be stored within the LSTM cell. Here’s where things get interesting.

**Input Gate:**

The input gate is also called the **save vector.**These gates determine which information should enter the cell state / long-term memory OR which information should be saved to the cell state or should be forgotten.

First, the (combined) input is squashed between -1 and 1 using a tanh activation function.  
This squashed input (from tanh) is then multiplied element-wise by the output of the input gate. The input gate is basically a hidden layer of sigmoid activated nodes, with weighted **xₜ**and input values **hₜ**₋ ₁, which outputs values of between 0 and 1 and when multiplied element-wise by the input determines which inputs are switched on and off (actually, the values aren’t binary, they are a continuous values between 0 & 1). In other words, it is a kind of input filter or gate (it tells what to learn and add to the memory from current input and the context its given and also how much {sigmoid gives values between 0&1} of what to learn).

*Simplistic (could be wrong) view: Tanh gives the Standardized (between -1 & 1) value of the actual unscaled (combined) input vector and the sigmoid layer is the controller of what percentage (values between 0 & 1 or 0 to 100%) of what inputs should be passed on of the scaled (from tanh) values to be added to the memory considering the current and previous context.*

But Why tanh activation?

Because the equation of the cell state is a summation between the previous cell state, sigmoid function alone will only add memory and not be able to remove/forget memory. If you can only add a float number between [0,1], that number will never be zero / turned-off / forget. This is why the input modulation gate has an **tanh** activation function. Tanh has a range of [-1, 1] and allows the cell state to forget certain memories.

**Forget Gate:**

The forget gateis also called the **remember vector**. The output of the forget gate tells the cell state which information to forget by multiplying 0 to a position in the matrix. If the output of the forget gate is 1, the information is kept in the cell state.

Although initially it is randomly initialized, but it basically LEARNS What exactly to FORGET (when the current input and previous Context is given) from the memory (cell state).

**Output Gate:**

The output gate is also called the **focus vector.**It basically highlights, out of all the possible values from the matrix(long memory), which information should be moving forward to the next hidden state.

Note: The **working memory** is usually called the **hidden state**(**ht**).It is basically → ht (LSTM OUTPUT) → What part of the existing memory (Ct) should be fed as context for the next round. This is analogous to the hidden state in RNN and HMM.

## X = numpy.reshape(dataX, (len(dataX), seq\_length, 1))

**Samples** - This is the len(dataX), or the amount of data points you have.

**Time steps** - This is equivalent to the amount of time steps you run your recurrent neural network. If you want your network to have memory of 60 characters, this number should be 60.

**Features** - this is the amount of features in every time step. If you are processing pictures, this is the amount of pixels. In this case you seem to have 1 feature per time step.

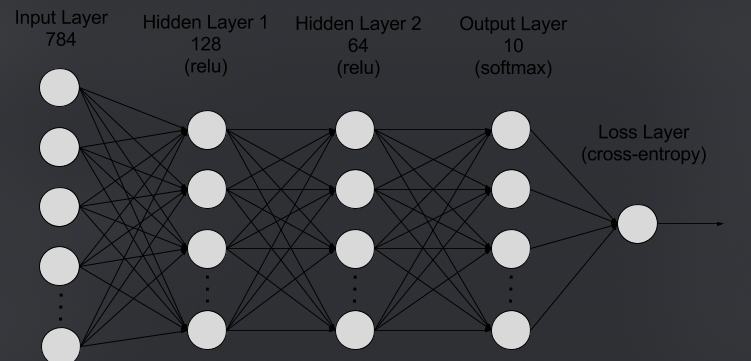
**4Q . Disadvantages of MLP when dealing with sequence data?**

A multilayer perceptron (MLP) is a class of feedforward artificial neural network. A MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training. Its multiple layers

and non-linear activation distinguish MLP from a linear perceptron. It can distinguish data that is not linearly separable.

“MLP” is not to be confused with “NLP”, which refers to natural language

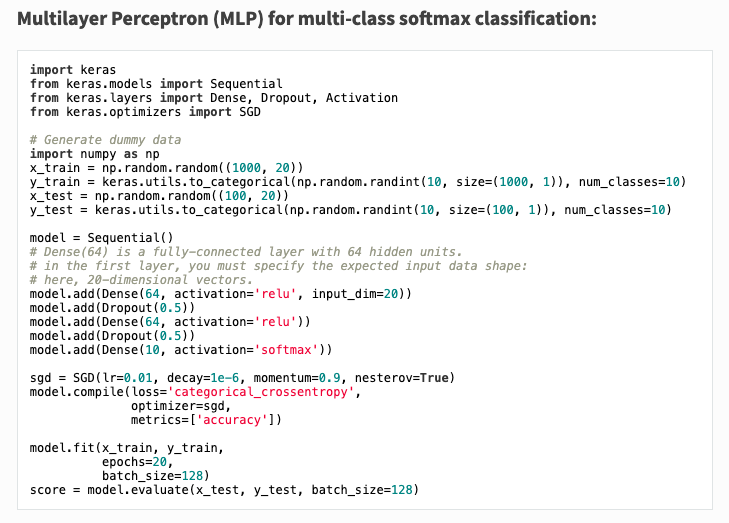
Multilayer perceptron wikipedia page



Multilayer Perceptron (MLP): used to apply in computer vision, now succeeded by Convolutional Neural Network (CNN). MLP is now deemed insufficient for modern advanced computer vision tasks. Has the characteristic of fully connected layers, where each perceptron is connected with every other perceptron. Disadvantage is that the number of total parameters can grow to very high (number of perceptron in layer 1 multiplied by # of p in layer 2 multiplied by # of p in layer 3…). This is inefficient because there is redundancy in such high dimensions. Another disadvantage is that it disregards spatial information. It takes flattened vectors as inputs. A light weight MLP (2–3 layers) can easily achieve high accuracy with MNIST dataset.

Disadvantages of MLP include too many parameters because it is fully connected. Parameter number = width x depth x height. Each node is connected to another in a very dense web — resulting in redundancy and inefficiency.

MLP in Keras: Tensorflow uses high level Keras API to give developers an easy-to-use deep learning framework. Here’s how to implement an MLP in Keras.



Multilayer Perceptron implementation in Keras

There was one point in time where MLP was the state-of-art neural networks. As the neural network architecture gets more complex or deeper, or evolve, MLP looks increasing simpler and more vanilla.