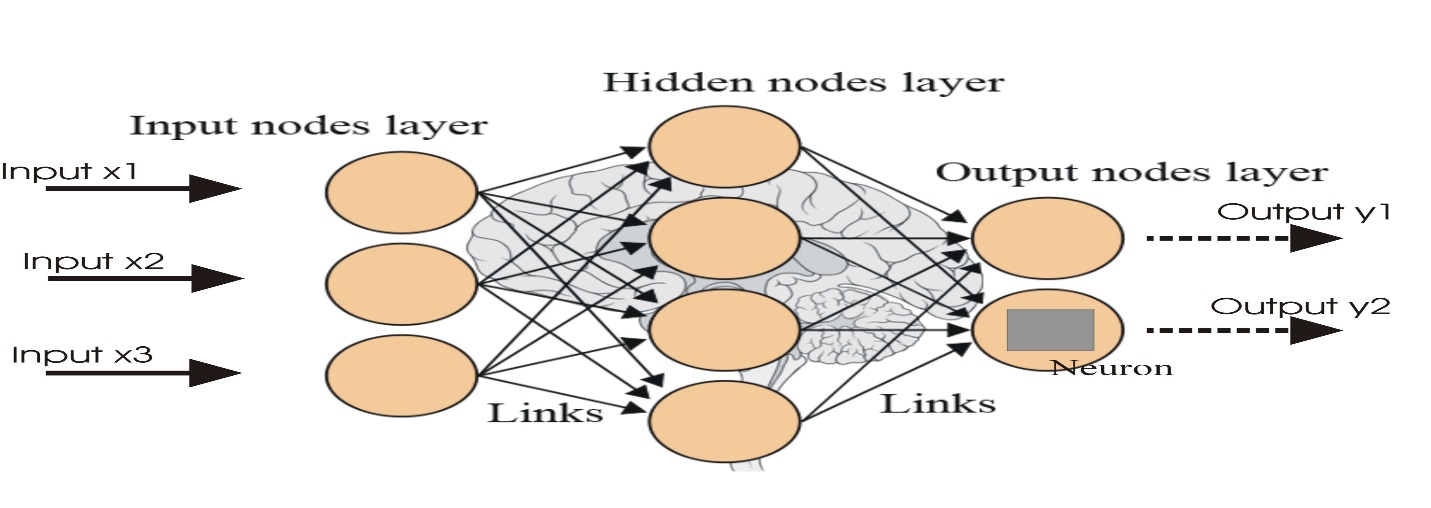
**Neural Networks:**

* Artificial neural networks have the ability to learn from supplied data, known as adaptive learning, while the ability of a neural network to create its own organization or representation of information is known as self-organization
* Artificial Neural networks or deep learning is a class of algorithms inspired by how human brain works.
* Neural network is compared with the human brain is that, they operate like non-linear parallel information-processing systems which rapidly perform computations such as pattern recognition and perception.
* ANN are good at breaking the problem or learning to solve the problem in multiple steps.
* In all machine algorithms, we build some kind of transient state, which allows machines to learn in a sophisticated manner.
* ANN operates on something referred as hidden states. These hidden states are something similar to neurons and are transient form which has a probabilistic behavior.
* A grid of such hidden states act as a bridge between input and output. (there are generally no connections in the same layer)



* Feed-forward neural networks, consists of a sequence of layers each fully connected to the next one. Each layer contains several neurons that interconnect with each other by weight links.

**Initialization of the parameters, weights and biases plays an important role in determining the final model. There is a lot of literature on initialization strategy. Helps in avoiding the model getting struck at local minima.**

* Doing experiment by using various initialization strategies is the key point to ensure that information propagated upwards and backwards in the network at the early stage of training. The initialization strategy should be selected according to the activation function used
* The activation function defines the output of a neuron. There are many activation functions:
  + Threshold function: not differentiable when performing back propagation
  + Sigmoid function: differentiable and continuous, non-linear in nature
  + Hyperbolic tangent: activation function range from -1 to+1
  + Maxout: smooth approximation to many logistic units.

Many more ….

**Back propagation:**

Used to train feed-forward or multi-layer perceptron. Minimize the cost function by changing the weights and biases in the network. Multiple ephoc’s are executed (training cycles) when the error defined by the cost function is minimized.

**Gradient decent**

* + Mini match gradient
  + Stochastic gradient
  + Full batch gradient

**Cost functions:**

There are various cost functions

* + Mean squared error
  + Cross entropy
  + Negative log-likelihood loss

**Learning rate:**

Learning rate controls the change in weights from one iteration to another. Smaller learning rates are stable and vice versa.

**Momentum**

Provides inertia to escape local minima. The idea to add some portion of previous weights.

**Softmax:**

Neural transfer function generalized from logistic function implemented in the output layer. It is only used in classification that is incorporated in the output layer that will give the probabilities of each occurring class.

Widely used neural networks are:

1. **CNN:**

There are four main steps in CNN: convolution, subsampling, activation and full connectedness.

* + ReLU activation function is mostly used
  + Weight initialization:
    - Gaussian random variable, not used in deep nets
    - Xavier initialization

Convolution: For me easiest way to understand a *convolution* is by thinking of it as a sliding window function applied to a matrix.

In a traditional feedforward neural network we connect each input neuron to each output neuron in the next layer. That’s also called a fully connected layer, or affine layer. In CNNs we don’t do that. Instead, we use convolutions over the input layer to compute the output

CNNs are basically just several layers of convolutions with *nonlinear activation functions* like [ReLU](https://en.wikipedia.org/wiki/Rectifier_(neural_networks)) or [tanh](https://reference.wolfram.com/language/ref/Tanh.html) applied to the results. Widely used in image classification.



Imagine that the matrix on the left represents a black and white image. Each entry corresponds to one pixel, 0 for black and 1 for white (typically it’s between 0 and 255 for grayscale images). The sliding window is called a *kernel,* *filter,* or *feature detector.* Here we use a 3×3 filter, multiply its values element-wise with the original matrix, then sum them up. To get the full convolution we do this for each element by sliding the filter over the whole matrix.

In CNN terminology, the 3×3 matrix is called a ‘**filter**‘ or ‘kernel’ or ‘feature detector’ and the matrix formed by sliding the filter over the image and computing the dot product is called the ‘Convolved Feature’ or ‘Activation Map’ or the ‘**Feature Map**‘. It is important to note that filters acts as feature detectors. Different values of the filter matrix will produce different Feature Maps for the same input image. Filter is just some random 3x3 matrix in yellow. Each new filter will create a new Feature Map.

In practice, a CNN *learns* the values of these filters on its own during the training process (although we still need to specify parameters such as number of filters, filter size, architecture of the network etc. before the training process). The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.

1. **Feature map is controlled using 3 steps:**

**Depth:** Depth corresponds to the number of filters we use for the convolution operation

**Stride:** Stride isthe number of pixels by which we slide our filter matrix over the input matrix. Stride =1,2 is 1 pixel matrix and 2 pixel matrix

**Zero-padding:** Assigning zero’s in the border

1. **The Pooling Step**

Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc.

**The overall training process of the Convolution Network may be summarized as below:**

**Step1:** We initialize all filters and parameters / weights with random values

**Step2:** The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.

Let’s say the output probabilities for the boat image above are [0.2, 0.4, 0.1, 0.3]. Since weights are randomly assigned for the first training example, output probabilities are also random.

Step3: Calculate the total error at the output layer (summation over all 4 classes)

 Total Error = ∑  ½ (target probability – output probability) ²

**Step4:** Use Backpropagation to calculate the gradients of the error with respect to all weights in the network and use gradient descent to update all filter values / weights and parameter values to minimize the output error.

The weights are adjusted in proportion to their contribution to the total error.When the same image is input again, output probabilities might now be [0.1, 0.1, 0.7, 0.1], which is closer to the target vector [0, 0, 1, 0].

This means that the network has learnt to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.

Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter

**Step5:** Repeat steps 2-4 with all images in the training set.

1. RNN: widely used in NLP.

The idea behind RNNs is to make use of sequential information. In a traditional neural network we assume that all inputs (and outputs) are independent of each other. But for many tasks that’s a very bad idea. If you want to predict the next word in a sentence you better know which words came before it. RNNs are called *recurrent* because they perform the same task for every element of a sequence, with the output being depended on the previous computations. Another way to think about RNNs is that they have a “memory” which captures information about what has been calculated so far. In theory RNNs can make use of information in arbitrarily long sequences, but in practice they are limited to looking back only a few steps (more on this later). Here is what a typical RNN looks like:



Training a RNN is similar to training a traditional Neural Network. We also use the backpropagation algorithm, but with a little twist. Because the parameters are shared by all time steps in the network, the gradient at each output depends not only on the calculations of the current time step, but also the previous time steps. For example, in order to calculate the gradient at t=4 we would need to back propagate 3 steps and sum up the gradients. This is called Backpropagation through Time (BPTT). If this doesn’t make a whole lot of sense yet, don’t worry, we’ll have a whole post on the gory details. For now, just be aware of the fact that vanilla RNNs trained with BPTT[have difficulties](http://www.jmlr.org/proceedings/papers/v28/pascanu13.pdf) learning long-term dependencies (e.g. dependencies between steps that are far apart) due to what is called the vanishing/exploding gradient problem. There exists some machinery to deal with these problems, and certain types of RNNs (like LSTMs) were specifically designed to get around them.

**Deep learning:**

A deep network usually means an artificial neural network that has more than one hidden layer. Training the deep hidden layers required more computational power. Having a greater depth seemed to be better because intuitively neurons can make the use of the work done by the neuron in the layer below resulting in distributed representation of the data.

Deep neural architectures are composed of multiple layers utilizing non-linear operations, such as in neural nets with many hidden layers. There are often various factors of variation in the dataset, like aspects of the data separately and often independently may vary.

* **Restricted Boltzmann Machine and Deep Belief Networks**

One of the unsupervised algorithms is Restricted Boltzmann Machines (RBM) that is used to pre-train deep belief network. The RBM is a simplified version of the Boltzmann Machine, inspired by statistical mechanics, which models energy based probabilities for the underlying distributions of the given data sets from which conditional distributions can be derived.

**Techniques to deal with class imbalance**

* + SMOTE: Synthetic Minority Over-sampling Technique
  + Cost-sensitive learning in neural networks