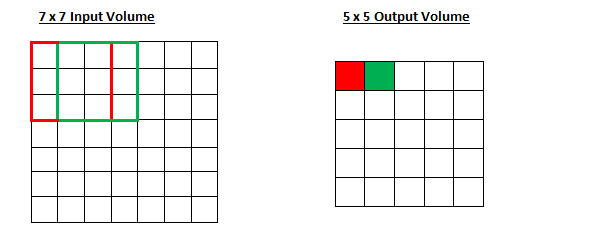
**Question 4**

**CNN**- Convolutional Neural network is a class of deep feed forward artificial neural networks commonly applied to analyzing visual imaginery it uses variation of multilayer perceptrons which require minimal preprocessing. This methos has been inspired by biological processes . the connectivity pattern between neurons resembles the organization of animal visual cortex. The cortical neurons responds to stimuli only in a restricted region of visual field known as receptive field. Cnn use relatively little preprocessing the network learns the filter itself. The cnn consist of an input layer as well as multiple hidden layers.

The hidden layers consist of convolutional layer, pooling layers, fully connected layers and normalization layers.

**Stride:-** Stride controls how the filter convolves around the input volume. the filter convolves around the input volume by shifting one unit at a time. The amount by which the filter shifts is the stride. In that case, the stride was implicitly set at 1. Stride is normally set in a way so that the output volume is an integer and not a fraction. Let’s look at an example. Let’s imagine a 7 x 7 input volume, a 3 x 3 filter. Stride controls how depth columns around the spatial dimensions (width and height) are allocated. When the stride is 1 then we move the filters one pixel at a time. This leads to heavily overlapping receptive fields between the columns, and also to large output volumes. When the stride is 2 (or rarely 3 or more) then the filters jump 2 pixels at a time as they slide around. The receptive fields overlap less and the resulting output volume has smaller spatial dimensions

****

**Padding:-** Sometimes it is convenient to pad the input with zeros on the border of the input volume. The size of this padding is a third hyperparameter. Padding provides control of the output volume spatial size. In particular, sometimes it is desirable to exactly preserve the spatial size of the input volume.The spatial size of the output volume can be computed as a function of the input volume size {\displaystyle W}, the kernel field size of the Conv Layer neurons {\displaystyle K}, the stride with which they are applied {\displaystyle S}, and the amount of zero padding {\displaystyle P} used on the border. The formula for calculating how many neurons "fit" in a given volume is given by {\displaystyle (W-K+2P)/S+1}. If this number is not an integer, then the strides are set incorrectly and the neurons cannot be tiled to fit across the input volume in a symmetric way. In general, setting zero padding to be {\displaystyle P=(K-1)/2} when the stride is {\displaystyle S=1} ensures that the input volume and output volume will have the same size spatially. Though it's generally not completely necessary to use up all of the neurons of the previous layer, for example, you may decide to use just a portion of padding. In order to build deep neural networks one modification to the basic convolutional operation that you need to really use is padding take

Example :- 6\*6 image when convolve it with a 3\*3 filter the result is 4\*4 output with a four by four matrix because the number of possible positions with the three by three filter is only four by four possible positions to solve both of these problems, both the shrinking output, and building a deep neural network of you have maybe a hundred layer of deep net, then it'll shrinks a bit on every layer which end up with a very small image.

This process throws away a lot of the information from the edges of the image in order to fix both of these problems. Apply fully convolutional operation by padding the image with an additional one border with one pixel all around the edges. Instead of a 6 \* 6 image its now padded this to 8 \* 8 image and if it convolve an eight by eight image with a 3 \* 3 image filter the output the 6 \*6 image, this process managed to preserve the original input size of 6 \* 6

If you have a stride of 1 and if you set the size of zero padding to

https://adeshpande3.github.io/assets/ZeroPad.png

where K is the filter size, then the input and output volume will always have the same spatial dimensions.

The formula for calculating the output size for any given conv layer is

https://adeshpande3.github.io/assets/Output.png

where O is the output height/length, W is the input height/length, K is the filter size, P is the padding, and S is the stride.

**Question 5**

**Optimization Techniques for Training Deep Models**

Deep learning algorithms involve optimization in many contexts. Performing inference in models such as PCA involves solving an optimization problem. We often use analytical optimization to write proofs or design algorithms.

Of all the many optimization problems involved in deep learning, the most diﬃcult is neural network training. It is quite common to invest days to months of time on hundreds of machines to solve even a single instance of the neural network training problem. Because this problem is so important and so expensive, a specialized set of optimization techniques have been developed for solve these problems. Optimization algorithms helps us to minimize (or maximize) an **Objective** function (another name for Error function)**E(x)**which is simply a mathematical function dependent on the Model’s internal **learnable parameters** which are used in computing the target values**(Y)** from the set of predictors**(X)** used in the model. For example — we call the **Weights (W)** and the **Bias(b)** values of the neural network as its internal learnable parameters which are used in computing the output values and are learned and updated in the direction of optimal solution i.e. minimizing the **Loss**by the network’s training process and also play a major role in the training process of the Neural Network Model .

Types of optimization Algorithm:-

1. **First Order Optimization Algorithms**— these algorithms minimize or maximize a Loss function **E(x)**using its Gradient values with respect to the parameters. Most widely used First order optimization algorithm is Gradient Descent**.** The First order derivative tells us whether the function is decreasing or increasing at a particular point. First order derivative basically give us a line which is tangentialto a point on its Error Surface.

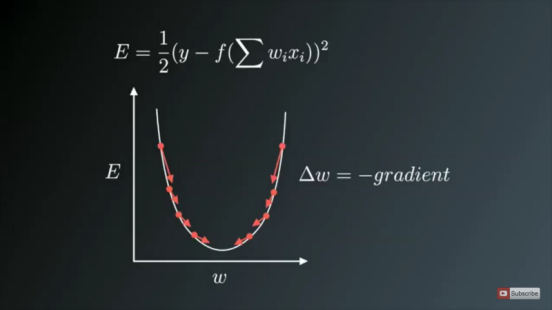
Gradient Function:- A **Gradient** is simply a vector which is a multi-variable generalization of a derivative(**dy/dx**) which is the instantaneous rate of change of y with respect to x**.**The difference is that to calculate a derivative of a function which is dependent on more than one variable or multiple variables, a Gradient takes its place. And a gradient is calculated using Partial Derivatives**.**Also another major difference between the Gradient and a derivative is that a Gradient of a function produces a Vector Field.

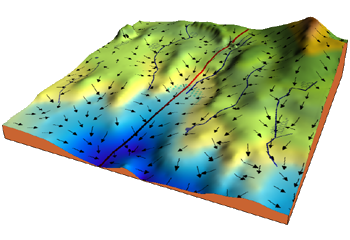
Gradient Descent: - Gradient Descent is the most important technique and the foundation of how we train and optimize Intelligent Systems**.**What is does is

* Find the Minima.
* Control the variance update the Model’s parameters.
* Leads us to Convergence.

It optimizes the Neural Network. Now gradient descent is majorly used to do **Weights updates** in a Neural Network Model , i.e update and tune the Model’s parameters in a direction so that we can minimize the **Loss function**. Now we all know a Neural Network trains via a famous technique called **Back-propagation ,**in which we first propagate forward calculating the dot product of Inputs signals and their corresponding Weights and then apply a [activation function](https://medium.com/towards-data-science/activation-functions-and-its-types-which-is-better-a9a5310cc8f)to those sum of products, which transforms the input signal to an output signal and also is important to model complex Non-linear functions and introduces **Non-linearity’s** to the Model which enables the Model to learn almost any arbitrary functional mappings.

Gradient Descent propagate **backwards** in the Network carrying **Error** terms and updating **Weights** values using Gradient Descent, in which we calculate the gradient of Error(E) functionwith respect to the Weights (W) or the parameters , and update the parameters (here Weights) in the opposite direction of the Gradient of the Loss function w.r.t to the Model’s parameters.

**



**Variants of Gradient Descent**

The traditional Batch Gradient Descent will calculate the gradient of the whole Data set but will perform only one update , hence it can be very slow and hard to control for datasets which are very very large and don’t fit in the Memory. How big or small of an update to do is determined by the Learning Rate -**η ,**and itis guaranteed to converge to the global minimum for convex error surfaces and to a local minimum for non-convex surfaces.Another thing while using Standard batch Gradient descent is that it computes redundant updates for large data sets.

#### 1. Stochastic gradient descent

#### 2. **Mini Batch Gradient Descent**

**Challenges faced while using Gradient Descent and its variants —**

1. Choosing a proper learning rate can be difficult. A learning rate that is too small leads to painfully slow convergence i.e will result in small baby steps towards finding optimal parameter values which minimize loss and finding that valley which directly affects the overall training time which gets too large. While a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.
2. Additionally, the same learning rate applies to all parameter updates. If our data is sparse and our features have very different frequencies, we might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.
3. Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding getting trapped in their numerous sub-optimal local minima. Actually, Difficulty arises in fact not from local minima but from saddle points, i.e. points where one dimension slopes up and another slopes down. These saddle points are usually surrounded by a plateau of the same error, which makes it notoriously hard for SGD to escape, as the gradient is close to zero in all dimensions.