$x_0 = 3$

$x_1 = x_0 - 0.2 f'(x_0)$

$= 3 - 0.2 \times 4$

$= 3 - 0.8$

$= 2.2$

$x_2 = x_1 - 0.2 \times f'(x_1)$

$= 2.2 - 0.2 f'(2.2)$

$\hookrightarrow 2 \times 2.2 - 2$

$4.4 - 2$

$= 2.4$

$= 2.2 - 0.2(2.4)$

$= 2.2 - 0.48$

$= 1.72$

$x_3 = x_2 - 0.2 \times f'(x_2)$

$= 1.72 - 0.2 \times (2 \times 1.72 - 2)$

$= 1.72 - 0.2(1.44)$

$= 1.72 - 0.288$

$= 1.432$

$x_4 = x_3 - 0.2 \times f'(x_3)$

$= 1.432 - 0.2 f'(1.432)$

$= 1.432 - 0.2(2 \times 1.432 - 2)$

$= 1.432 - 0.2(0.864)$

$= 1.2592$

$x_5 = x_4 - 0.2 \times f'(x_4)$

$= 1.2592 - 0.2 f'(1.2592)$

$= 1.2592 - 0.2(2 \times 1.2592 - 2)$

$= 1.2592 - 0.2 \times 0.5184$

$= 1.1556$

$x_6 = x_5 - 0.2 \times f'(x_5)$

$= 1.1556 - 0.2 \times f'(1.1556)$

$= 1.1556 - 0.2(2 \times 1.1556 - 2)$

$= 1.1556 - 0.2(0.3112)$

$= 1.09336.$

$x_7 = x_6 - 0.2 * f'(x_6)$

$= 1.09336 - 0.2(2 \times 1.09336 - 2)$

$= 1.09336 - 0.2 \times 0.18$

$= 1.05$

$x_8 = x_7 - 0.2 * f'(x_7)$

$= 1.05 - 0.2(2 \times 1.05 - 2)$

$= 1.03$

$x_9 = x_8 - 0.2 * f'(x_8)$

$= 1.03 - 0.2 * (2 \times 1.03 - 2)$

$= 1.03 - 0.012$

$= 1.01$

$x_{10} = x_9 - 0.2 \times f'(x_9)$

$= 1.01 - 0.2 \times (2 \times 1.01 - 2)$

$= 1.01 - 0.004$

$= 1.006$

$x_{11} = x_{10} - 0.2 \times f'(x_{10})$

$= 1.006 - 0.2(2 \times 1.006 - 2)$

$= 1.006 - 0.0024$

$= 1.003$

$x_{12} = x_{11} - 0.2 \times f'(x_{11})$

$= 1.003 - 0.2(2 \times 1.003 - 2)$

$= 1.003 - 0$

$= 1.001$

$x_{13} = x_{12} - 0.2 \times f'(12)$

$= 1.001 - 0.2(2 \times 1.001 - 2)$

$= 1.000$

⋮

→ cost function provides with a qualification of how incorrect the model is in estimating the ideal 'y'.

→ The primary approach for minimizing the cost in deep learning paradigms is to pair an approach called gradient descent with another approach called Back propagation
↳ coming back & adjusting the weights

Batch gradient descent:-

Batch gradient descent is also known as Vanilla Gradient Descent.

→ In this it divides the large data into some batches.

$$\text{No of batches} = \left\lceil \frac{\text{size of training dataset}}{\text{Batch size}} \right\rceil$$

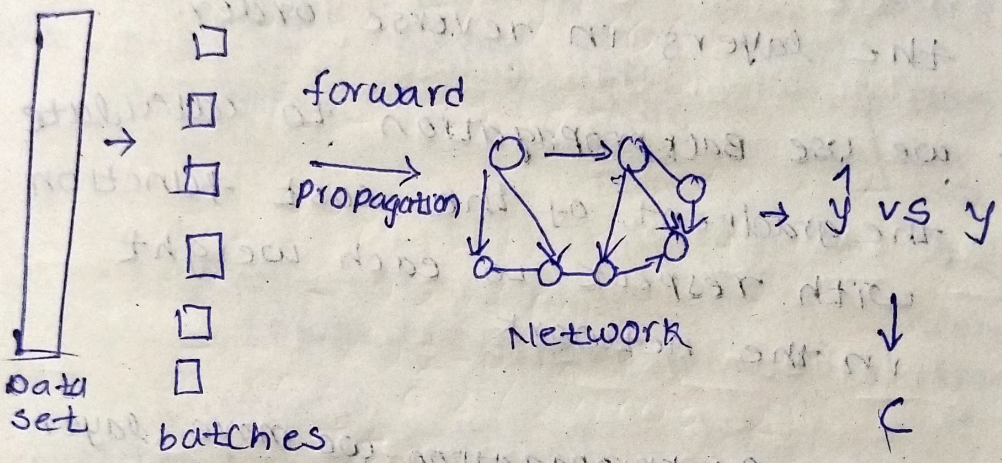
ex:- $\left\lceil \frac{60,000}{128} \right\rceil$

step-1:- sample a mini-batch of 'x' values.

step-2:- Forward propagates 'x' through network to estimate 'y' with $\hat{y}$

3) calculate the cost 'C' by comparing y and $\hat{y}$.

4) descent gradient of 'C' to adjust 'w' and 'b', enabling 'x' to better predict 'y'.



Backpropagation:

→ to efficiently adjust parameters to multiple layers of artificial neurons stochastic gradient descent is combined with back propragation

→ Back progation is an important application of chain-rule in calculus.

→ forward propagation carries information about input 'x' through successive layers of neurons to approximate 'y' with 'ŷ'.

→ Backpropagation carries information about the cost 'c' backwards through the layers in reverse order.

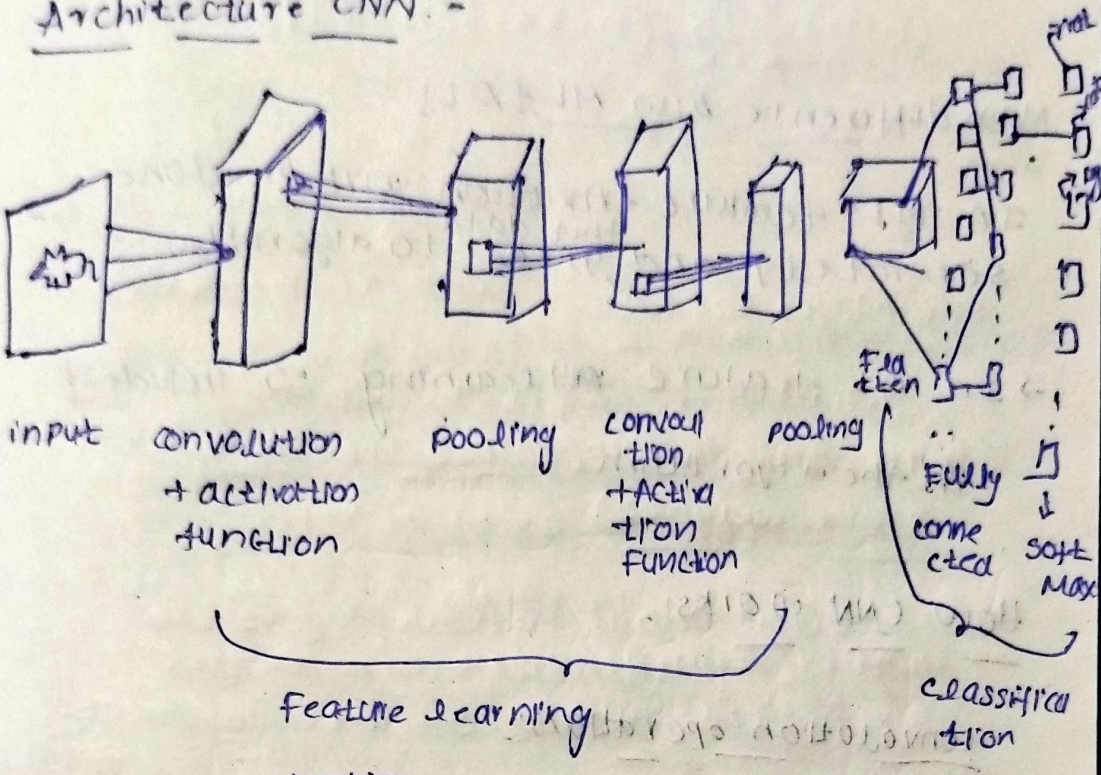
→ we use Backpropagation to calculate the gradient of the cost function with respect to each weight in the network.

→ Using Backpropagation we move layer by layer backwards to the network starting at the cost in the o/p layer and we find the gradients of every single parameter.

→ A given parameters gradient can be used to adjust the parameter up (or) down.

Unit-3

Architecture CNN:-



convolutional (multiplication) Neural Network:-

These are a class of neural network especialized for image processing like other neural networks, they transform input to output through many layers,

→ In CNN layers have:-

1) convolution step
2) a pooling step (optional)
3) a non-linear activation

→ Each layer in a neural network transform the input tensor into the output tensor through linear and non-linear operations.

→ All these intermediate tensors are called activations and they are all

different representation of the input.

Main difference b/w ML & DL:-

In ML feature extraction will be done seperately and that data gives to algorithms

→ In DL feature learning is included in the algorithm.

How CNN works:-

convolution operation:-

$$s(t) = \int x(a)\, w(t-a)\, da \rightarrow (1)$$

$$s(t) = (x * w)(t) \rightarrow (2)$$

→ First argument to the convolution is reffered to as input.

→ second argument is the kernel.

→ The o/p is reffered to the feature map

→ the data is collected every time is called continous data

→ the data is collected at a srecific time with time interval is called discreate data

Discrete continuous convolution operation:-

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a) \cdot w(t-a) \quad \rightarrow (3)$$

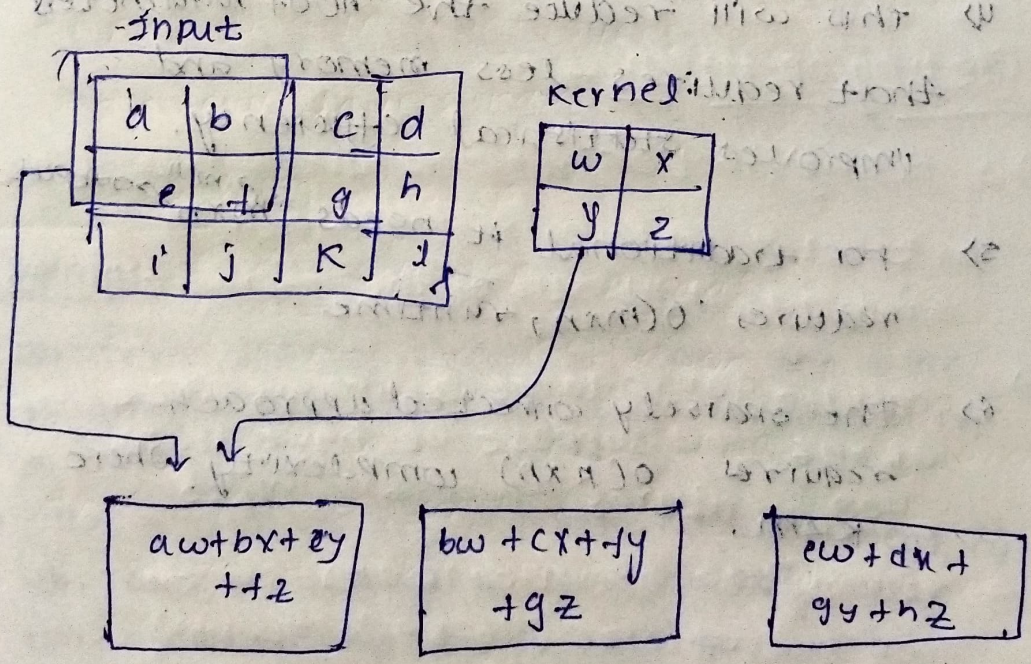
→ the time index t can then take only integer values.

→ the input is usually a multidimensional array of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm.

→ we call refer to these array as tensors

→ we usually assume that these functions are zero everywhere but the finite set of points for which we store the values.

Cross-correlation operation:-



→ Stride defines the no. of columns & the no. of rows shifted.

$(n \times n) \xrightarrow[K \times K]{} (n-k+1)(n-k+1)$

↓

order of resultant matrix when the data (nxn) is multiplied with kernel (KxK)

Advantages of CNNs:-

slow connection (i.e) change in input doesn't affect the o/p more.

Sparse Interactions (sparse connectivity or sparse weights)

1) Traditional neural network layers use matrix multiplications by a matrix parameters with a seperate parameters

2) Every o/p unit interacts with every input unit

} Trad. Neural

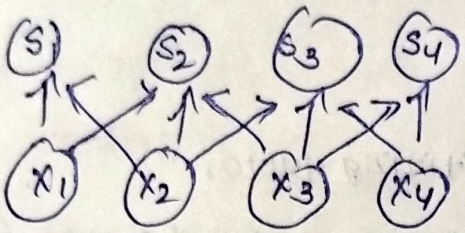
3) It is accomplished by making the kernel smaller than the i/p

4) This will reduce the no of parameter that requires less memory and improves statistical efficiency.

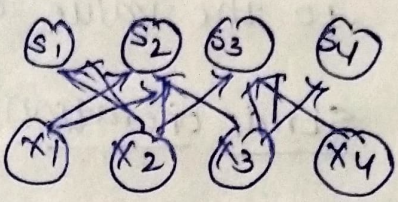
5) For traditional it needs $m \times n$ (input, output) requires $O(m \times n)$ runtime

6) The sparsely connected approach requires $O(k \times n)$ complexity where $k < m$.

convolutional NN!-



traditional method!-



⇒ x-inputs and s-output then the inputs of 's' are called as receptive fields

The Receptive Fields of units in the deeper layers of a CNN is larger than the receptive fields of units in a shallow layers.

→ this effect increases if the network includes architectural features like
1) strided convolution
2) pooling

→ Even though the direct connections in a convolutional network are very sparse - units in the deeper layers can be indirectly connected to all (or) most of the input units.

Parameter sharing!-

→ parameter sharing refers to using the same parameter for more than one function in a model.

→ In a traditional network, each element of the weight matrix is used exactly once when computing the output of a layer.

→ the parameter sharing mechanism uses tied weights which means the value of

the weight applied to one input is tied to the value of weight applied elsewhere.

Stride operation:-

→ stride is known as shifting factor.

→ stride = 1 means shifting of 1 row & 1 column both horizontally and vertically.

→ let us say n : order of input matrix
k : order of kernel
s : shifting factor (stride)

$$\left(\frac{n-k}{s}+1\right) * \left(\frac{k-k}{s}+1\right)$$

let, n=6, k=3, s=2

The output matrix would be the size 2x2

→ stride is used to reduce the size of the matrix.

Pooling operation:-

→ A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs.

→ consider for instance of image size 96 x 96 and suppose we have learned 400 features over 8x8 inputs.

→ convolution operation results in an output of size (96-8+1) * (96-8+1)

= 7921

→ Suppose we have 400 features (dimensions) this results in a vector of size: 7921×400

→ It is very difficult to process and prone to overfitting.

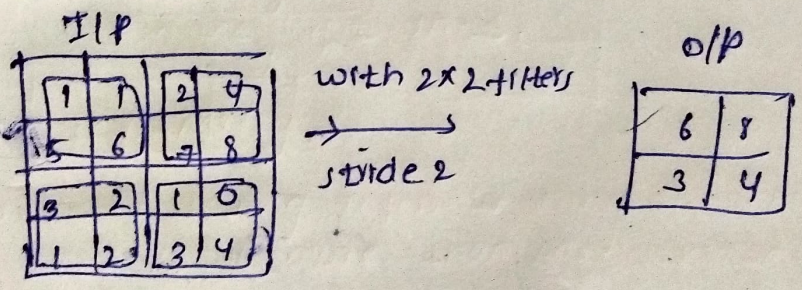
→ Formally, after obtaining convolved features as described earlier, we divide the size of region say m×m to pool our convolved features we divide our convolved features into disjoint m×n regions, and take mean of it.

→ It is used to reduce the computation complexity.

→ For translation and rotational it is invarient.

→ Types of pooling:-
1) max pooling
2) mean "
3) Average "

Max pooling:-



Mean pooling:-