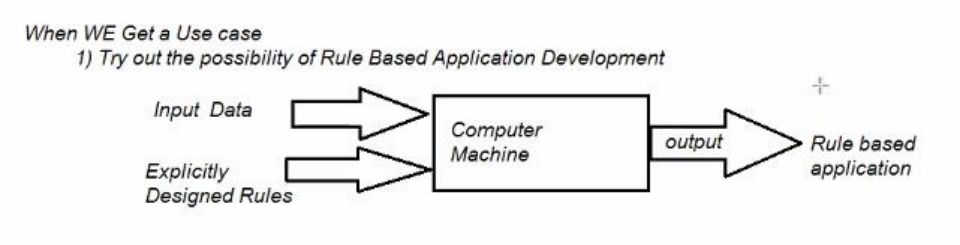
Data Science Training – Day 4

Topic: Neural Network / Deep Learning

- ANN
- CNN
- RNN

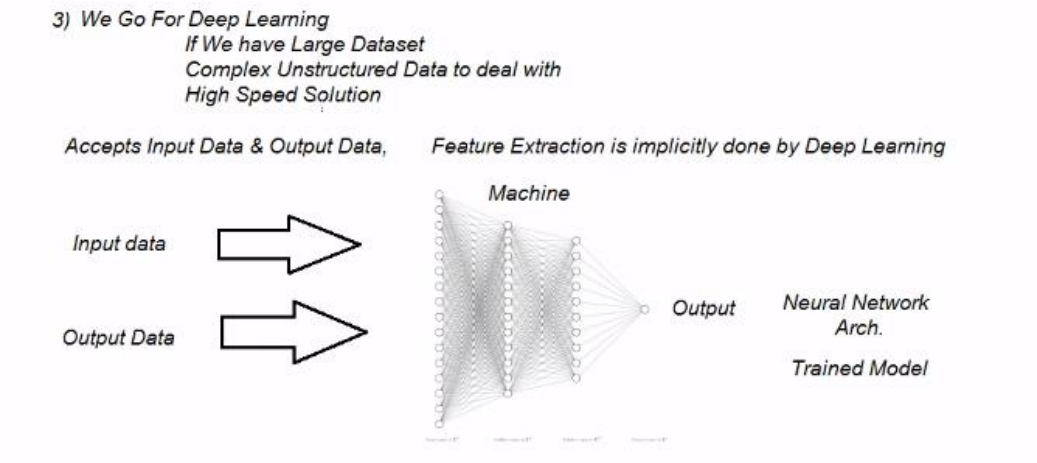
Use case:

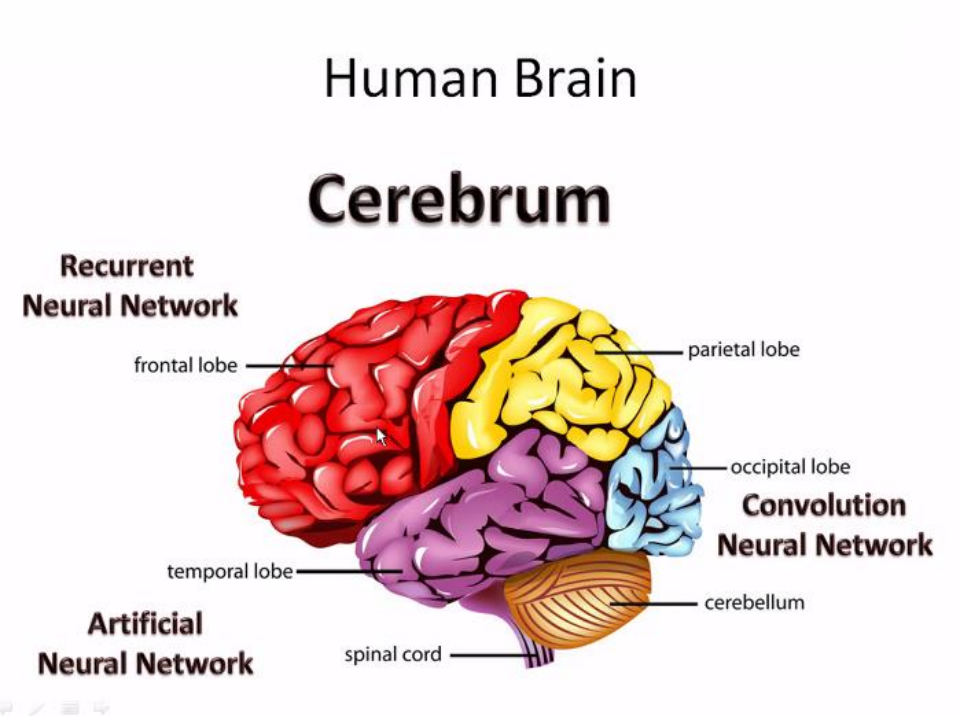
Explicit coding the logic



Develop the solution with explicit defined the rule-based application

Independent variable	Dependent variable
Input Data	Output Data
YrsExp	Salary
Investment, Tax, Revenue	Profit
Age, Salary, City	Response





CNN - human interpretation by seeing pictures.

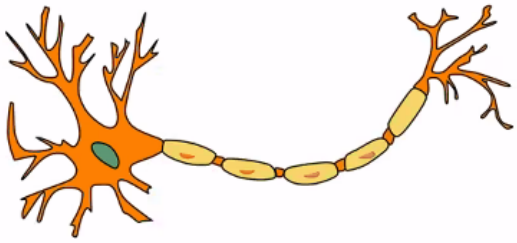
ANN -

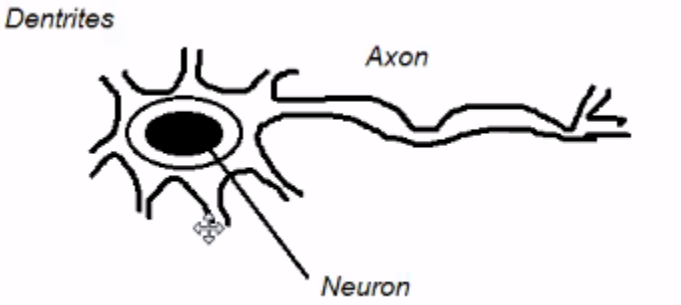
RNN – What did you have for breakfast? As time goes, you start forgetting it. On Thursday, what did you had for breakfast?

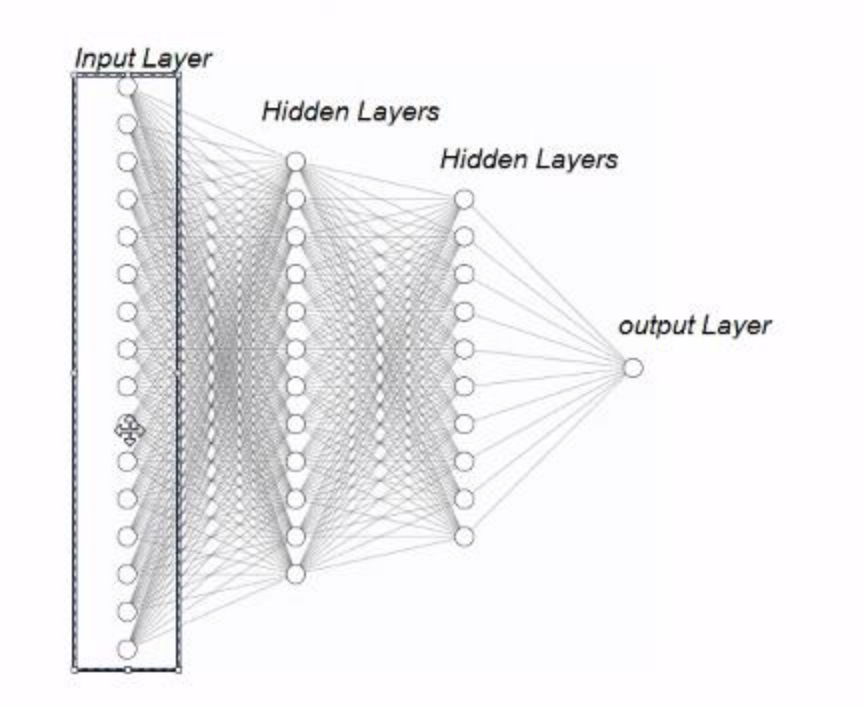
Process data in sequence, as time goes in sequence, your brain starts forgetting things.

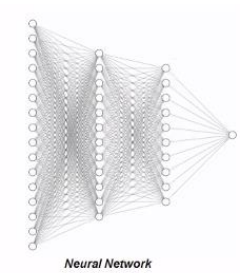
Deep Learning

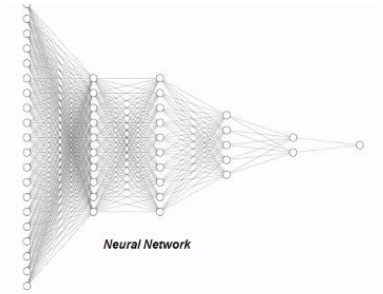
- In deep learning, these layered representations are learned via models called "neural networks", structured in literal layers stacked one after the other.
- The term "neural network" is a reference to neurobiology.

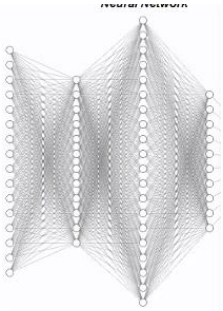


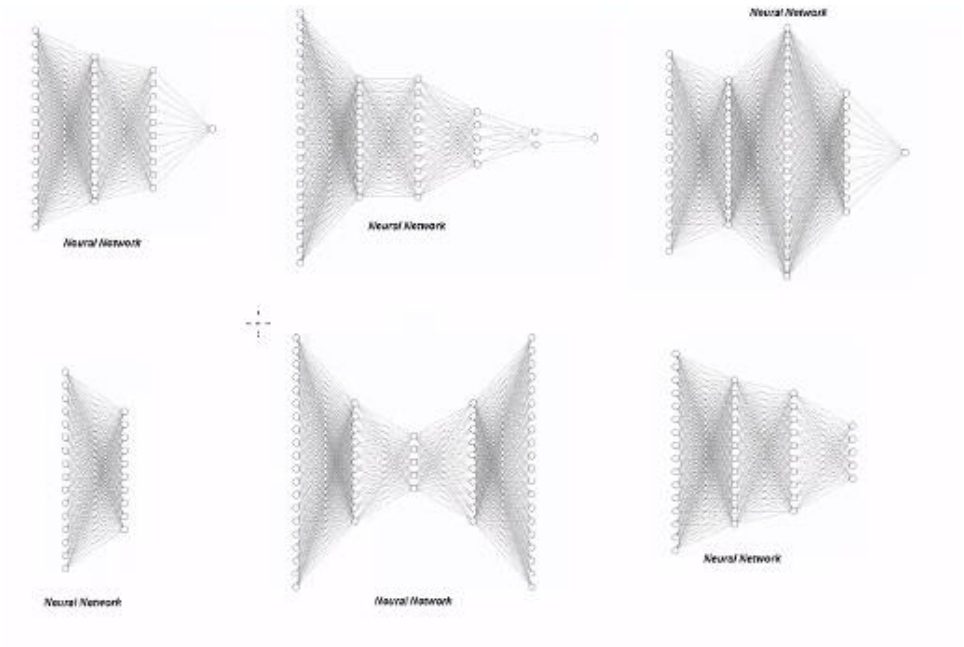


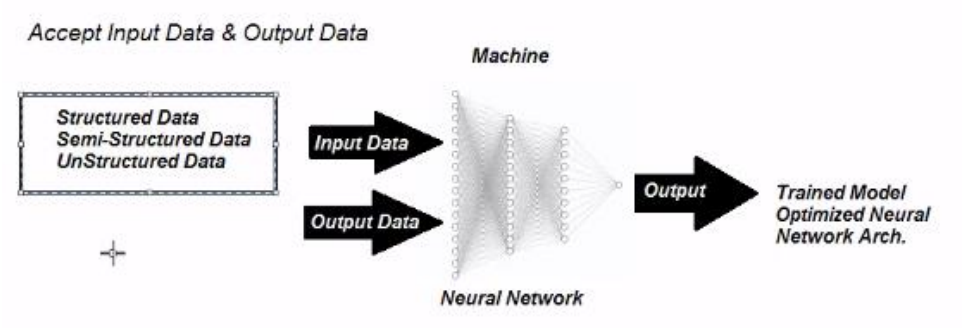

Input Layer
Hidden Layers
Hidden Layers
output Layer


Neural Network


Neural Network







Layered structured

Represented as neuron

Input Layer

Passing Input Data to NN
Each Neuron in Input Layer
represents X feature /column

Hidden Layer

Capture Patterns in the data
All calculation that needs to be done on input to transform it into desired output is done here

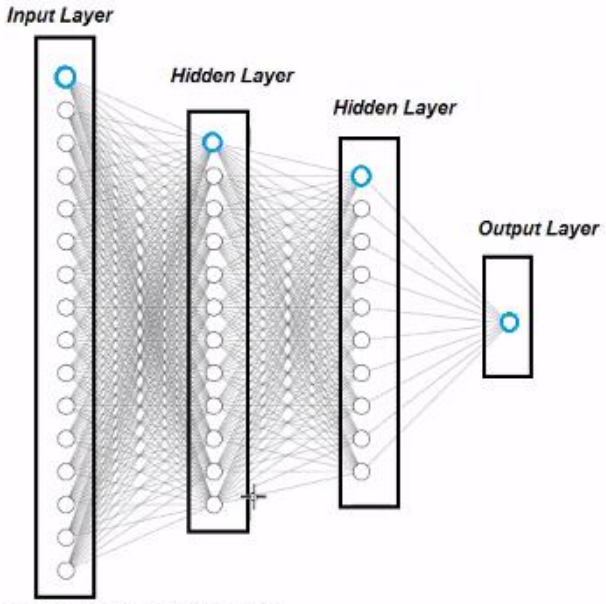
Each neuron in HL captures a pattern

NumOfNeurons in HL == Trial & Error

3 neurons = 3 patterns
30 neurons = 30 patterns
300 neurons = 300 patterns

Output Layer

Return output



.....

min
1 2 3 4 5 6 .....

12 24 48 .....

underfitting
Not learning enough

Error 27k 10k 5k 1k 500

Training error High
Testing error High

Overfitting issues

Training error Low
Testing error High

.....

min
1 2 3 4 5 6 .....

12 24 48 .....

underfitting
Not learning enough

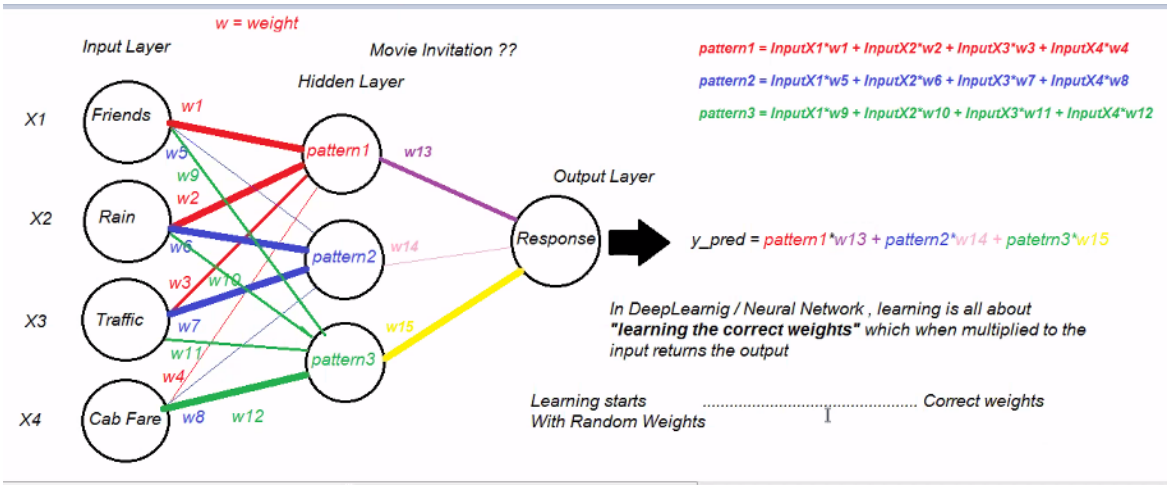
Error 27k 10k 5k 1k 500

Training error High
Testing error High

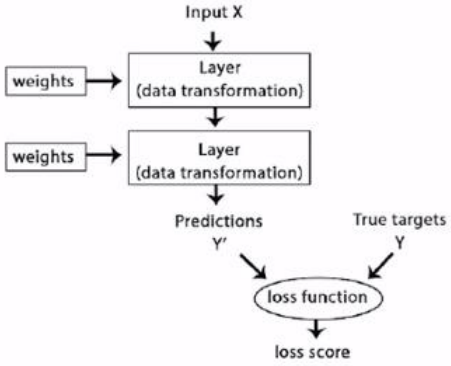
Overfitting issues

Training error Low
Testing error High

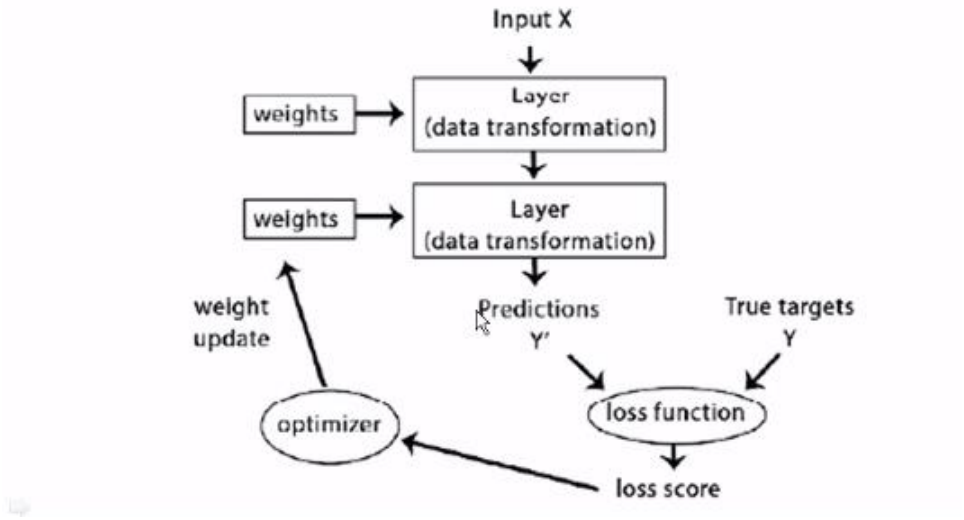
Regression	**Continuous value Single Pred (1 neuron)**
Binary Classification	**Discrete Binary value 0/1 Pred (1 neuron)**
Multi-Class Classification	**Discrete Value (more than 1 neuron)**

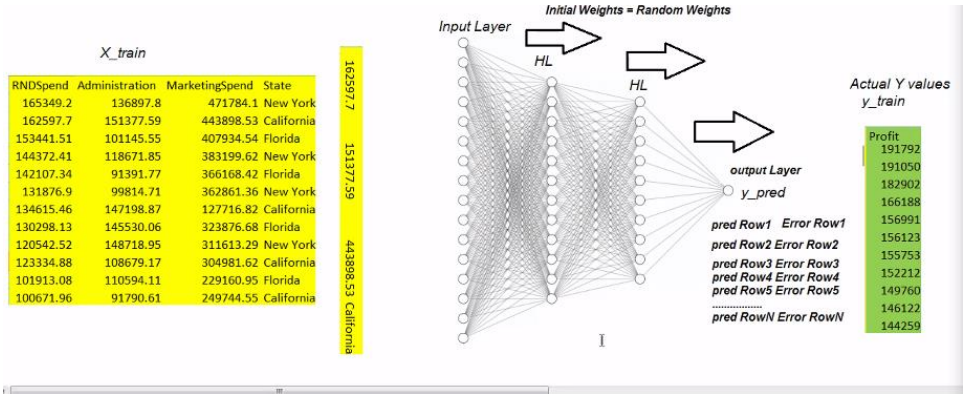


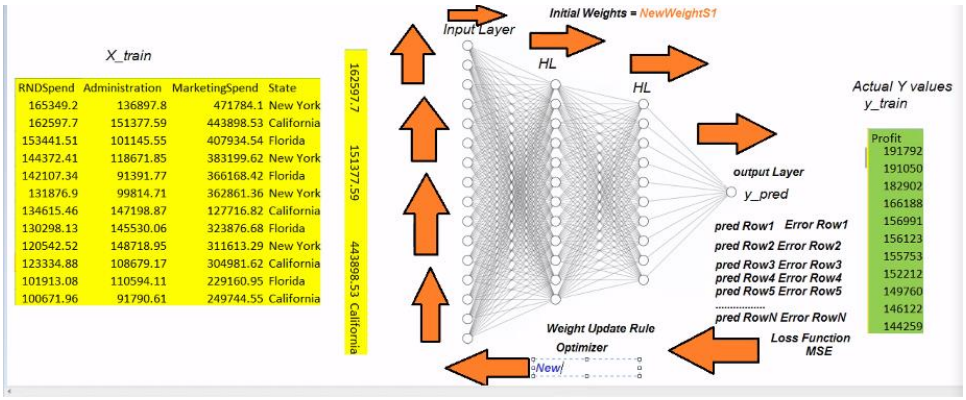
- To control the output of a neural network, you need to be able to measure how far this output is from what you expected.
- This is the job of the **"loss function"** of the network, also called **"objective function".**
- The loss function takes the predictions of the network and the true target , and computes a distance score, capturing how well the network has done on this specific example.

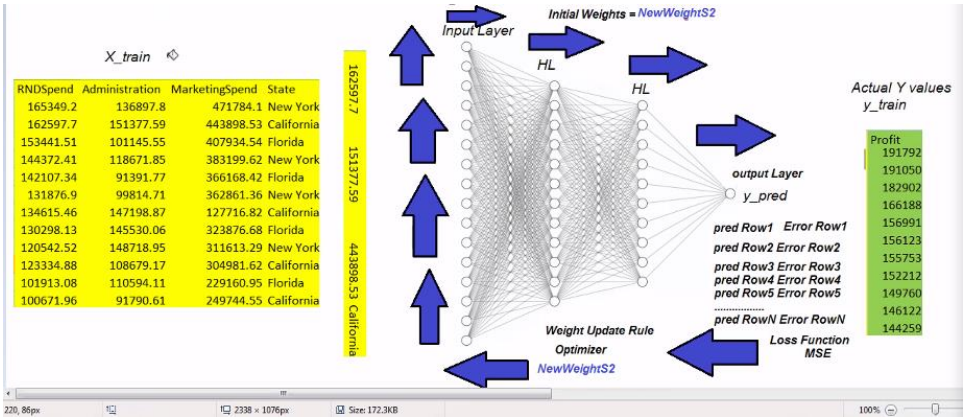


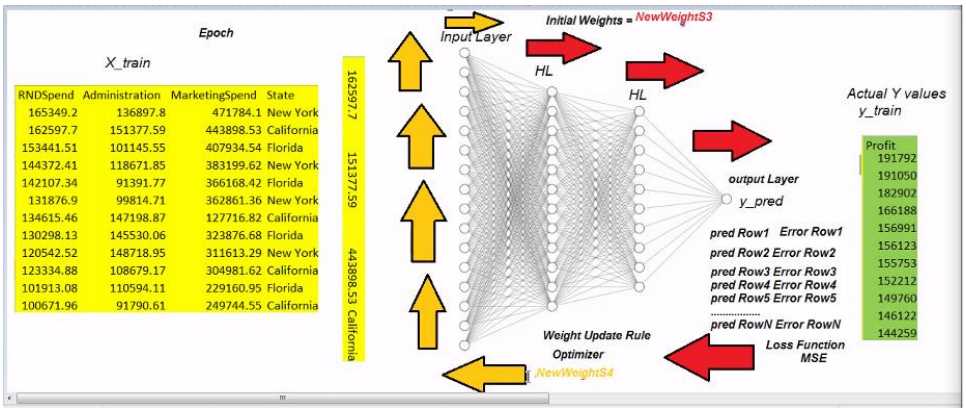
- The fundamental trick in deep learning is to use the loss score as a feedback signal to adjust the value of the weights by a little bit, in a direction that would lower the loss score for the current example.
- This adjustment is the job of the "optimizer", which implements what is called the **"backpropagation"** algorithm, the central algorithm in deep learning.











UnderFitting	Not Learning Enough
Good Fit	Learned well to generalize
OverFitting	Too much of Learning

TrainingAcc	TestingAcc	
60%	60%	Low Acc Underfitting
90%	90%	Good Fit
90%	95%	Good Fit
90%	75%	OverFitting

```
numOfEpochs = To be decided by developer (Trial & Error)

Step1: Initial Weights = Random Weight Allocation (All rows in training set is used against the initial weight)
Epoch1    y_pred = InputX1*weight1 + InputX2*weight2 + InputX3*weight3    .............+InputXn*weightn
          Error = lossFunction(y_pred,y_actual)
          WeightUpdate = optimizer ( weight update Rule)
                         NewWeightS1

Step2: Initial Weights = NewWeightS1
Epoch2    y_pred = InputX1*weight1 + InputX2*weight2 + InputX3*weight3    .............+InputXn*weightn
          Error = lossFunction(y_pred,y_actual)
          WeightUpdate = optimizer ( weight update Rule)
                         NewWeightS2

Step3: Initial Weights = NewWeightS2
Epoch3    y_pred = InputX1*weight1 + InputX2*weight2 + InputX3*weight3    .............+InputXn*weightn
          Error = lossFunction(y_pred,y_actual)
          WeightUpdate = optimizer ( weight update Rule)
                         NewWeightS3
.................
StepN: Initial Weights = NewWeightSN-1
EpochN    y_pred = InputX1*weight1 + InputX2*weight2 + InputX3*weight3    .............+InputXn*weightn
          Error = lossFunction(y_pred,y_actual)
          WeightUpdate = optimizer ( weight update Rule)
                         NewWeightSN   => Final Weights giving max accuracy and lowest error / loss
```

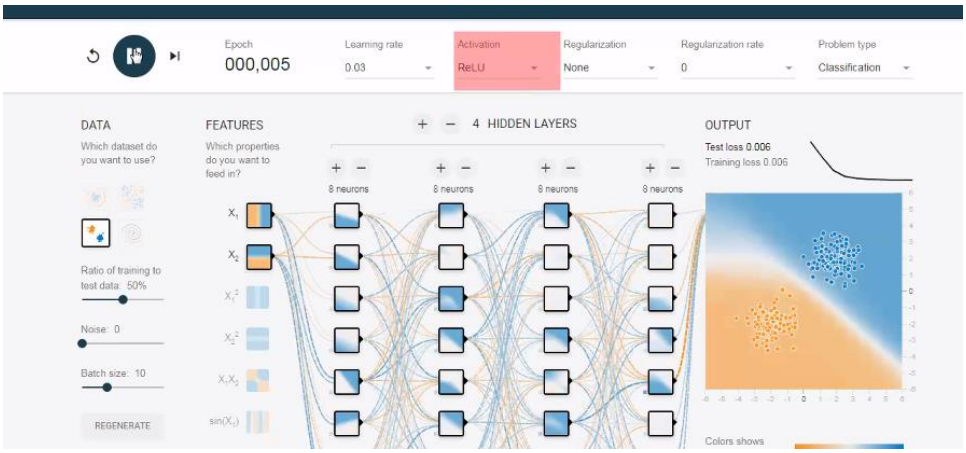
```
Formula at the heart of Each Neuron [Linear Spread of Data]

y_pred = ActivationFunction(bias + InputX1*weight1 + InputX2*weight2 + InputX3*weight3.....+InputXn*weightn)


Activation Function

Linear Activation Functions                    (Linear Spread of data)

Non-Linear Activation Functions                (Linear Spread of data & Non-Linear Spread of data)
          Sigmoid , Tanh, ReLu,LeakyReLu,ExpReLu


Linear Regression - Linear Spread of Data

y_pred = b0 +  InputX1*b1 + InputX2*b2 + InputX3*b3    .............+InputXn*bn
```

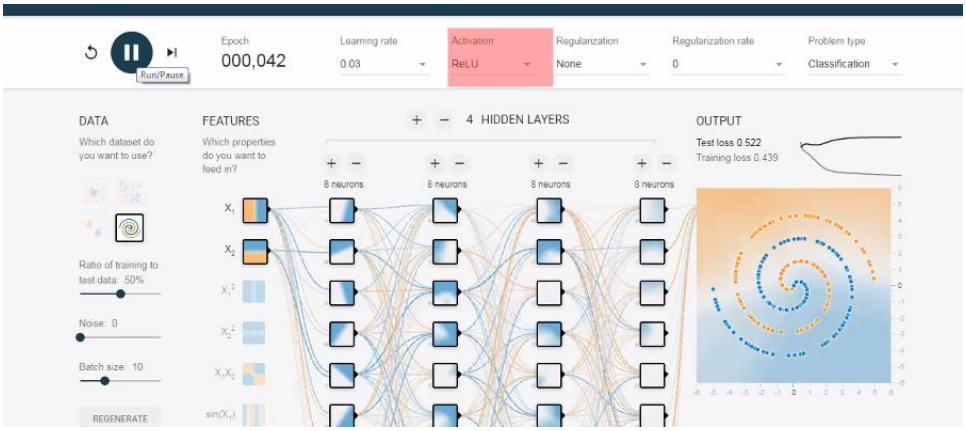
Bias = intercept

Activation function

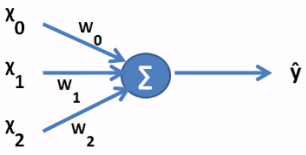
Tensor flow playground - To see how neural network work

To see how activation function work





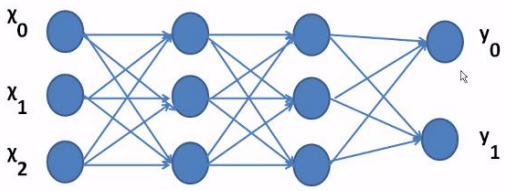
Neural Network with Single Neuron



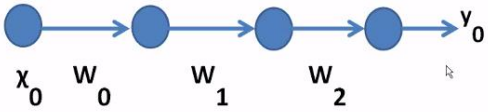
$$\hat{y} = x_0 w_0 + x_1 w_1 + x_2 w_2 = \sum_i x_i w_i$$

$$\begin{pmatrix} w_0 & w_1 & w_2 \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix}$$

Muti-layer Perceptron

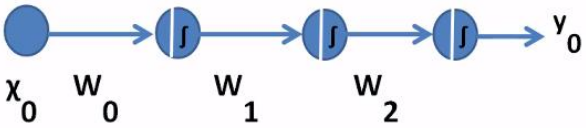


Linear Function



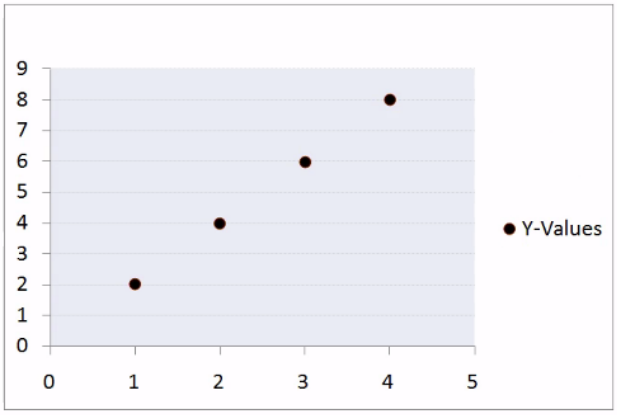
$$y_0 = x_0 w_0 w_1 w_2 = x_0 w_C$$

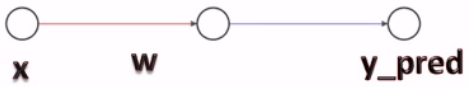
Non-Linear Function



$$y_0 = \sigma (\sigma (\sigma (x_0 w_0) w_1) w_2)$$

Simple Example





y_pred = xw

E = $(\hat{y}-y)^2 = (xw-y)^2$ *sum of squared Error*

$\frac{\partial \mathbf{E}}{\partial \mathbf{w}} = 2x(xw-y)$ Derivative

$\mathbf{w - \alpha \frac{\partial E}{\partial w} = w - \alpha\, 2x(xw-y)}$ Update Rule

- Random weight initialization w=0.5
- learning rating α = 0.1

- $w_{new} = w - 0.1 * 2x(xw-y)$
- (x,y)
- (2,4) w=0.5 ← 0.5 - 0.1*2*2(2*0.5-4) = 1.7
- (1,2) w =1.7 ← 1.7-0.1*2*1(1*1.7-2) = 1.76
- (3,6) w= 1.76 ← 1.76-0.1*2*3(3*1.76-6)=2.192
- w ~ 2

Weight update Rule:

SGD:

Epoch1	x	y	LR	weights	NewWeights	y_pred	y_test	Error / Loss	
	2	4	0.1	0.5	1.7	3.4	4	-0.6	individual Row Error
	1	2	0.1	1.7	1.76	1.76	2	-0.24	individual Row Error
	3	6	0.1	1.76	2.192	6.576	6	0.576	individual Row Error
	4	8	0.1	2.192	1.5776	6.3104	8	-1.6896	individual Row Error
								-0.4884	

Epoch2	x	y	LR	weights	NewWeights	y_pred	y_test	Error / Loss	
	2	4	0.1	1.5776	1.91552	3.83104	4	-0.16896	individual Row Error
	1	2	0.1	1.91552	1.932416	1.932416	2	-0.067584	individual Row Error
	3	6	0.1	1.932416	2.0540672	6.162202	6	0.1622016	individual Row Error
	4	8	0.1	2.054067	1.88105216	7.524209	8	-0.47579136	individual Row Error
								-0.13753344	Epoch Error

Epoch3	x	y	LR	weights	NewWeights	y_pred	y_test	Error / Loss	
	2	4	0.1	1.881052	1.976210432	3.952421	4	-0.047579136	individual Row Error
	1	2	0.1	1.97621	1.980968346	1.980968	2	-0.019031654	individual Row Error
	3	6	0.1	1.980968	2.015225324	6.045676	6	0.045675971	individual Row Error
	4	8	0.1	2.015225	1.966504288	7.866017	8	-0.133982847	individual Row Error
								-0.038729417	Epoch Error

Epoch4	x	y	LR	weights	NewWeights	y_pred	y_test	Error / Loss	
	2	4	0.1	1.966504	1.993300858	3.986602	4	-0.013398285	individual Row Error
	1	2	0.1	1.993301	1.994640686	1.994641	2	-0.005359314	individual Row Error
	3	6	0.1	1.994641	2.004287451	6.012862	6	0.012862353	individual Row Error
	4	8	0.1	2.004287	1.990567608	7.96227	8	-0.03772957	individual Row Error
								-0.010906204	Epoch Error

BGD:

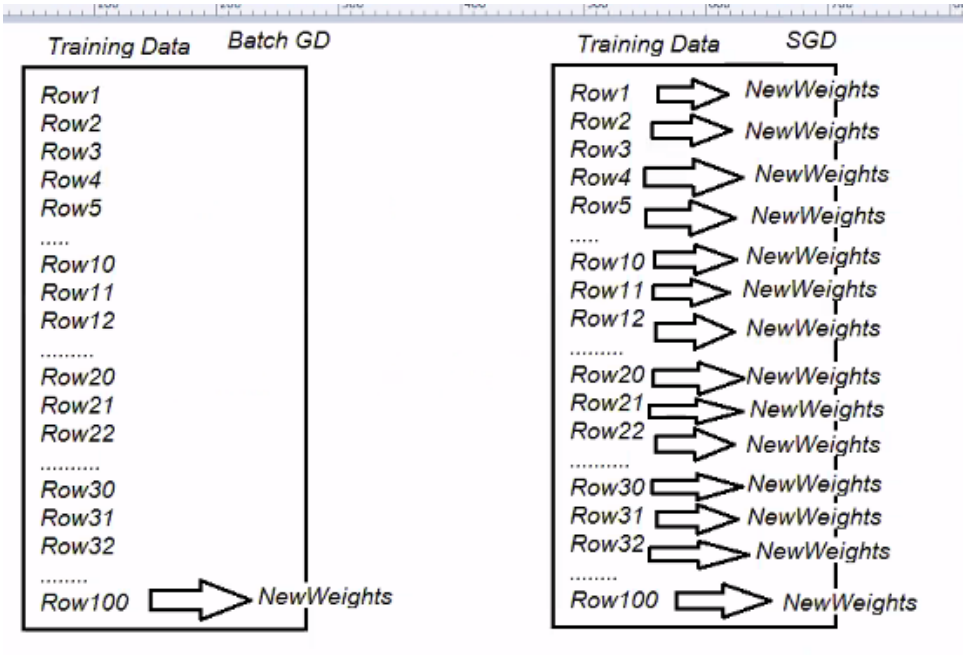
Epoch1	x	y	LR	weights	NewWeights	y_pred	y_test	Error / Loss
	2	4	0.1	0.5	1.7	3.4	4	-0.6
	1	2	0.1	0.5	0.8	0.8	2	-1.2
	3	6	0.1	0.5	3.2	9.6	6	3.6
	4	8	0.1	0.5	5.3	21.2	8	13.2
					2.75	NewWeights		3.75

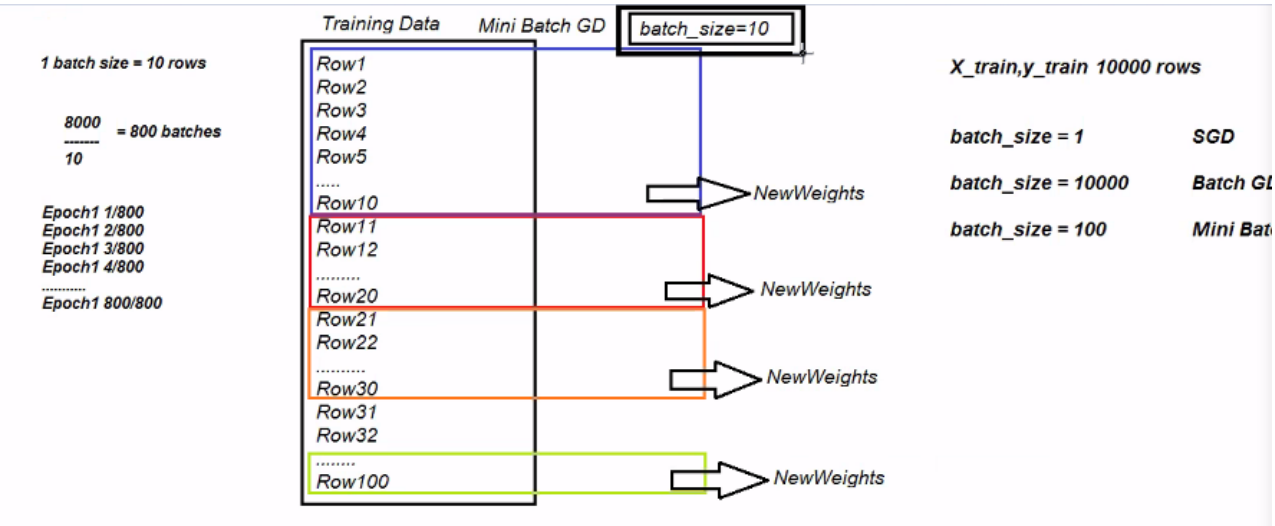
Epoch2	x	y	LR	weights	NewWeights	y_pred	y_test	Error	
	2	4	0.1	2.75	2.15	4.3	4	0.3	individual Row Error
	1	2	0.1	2.75	2.6	2.6	2	0.6	individual Row Error
	3	6	0.1	2.75	1.4	4.2	6	-1.8	individual Row Error
	4	8	0.1	2.75	0.35	1.4	8	-6.6	individual Row Error
					1.625			-1.875	Epoch Error

Epoch3	x	y	LR	weights	NewWeights	y_pred	y_test	Error	
	2	4	0.1	1.625	1.925	3.85	4	-0.15	individual Row Error
	1	2	0.1	1.625	1.7	1.7	2	-0.3	individual Row Error
	3	6	0.1	1.625	2.3	6.9	6	0.9	individual Row Error
	4	8	0.1	1.625	2.825	11.3	8	3.3	individual Row Error
					2.1875			0.9375	Epoch Error

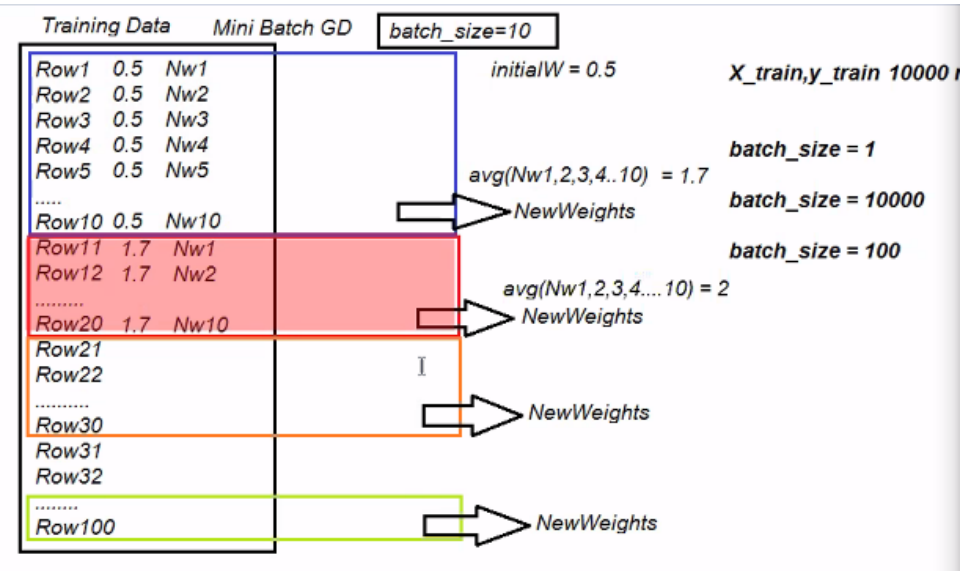
Epoch4	x	y	LR	weights	NewWeights	y_pred	y_test	Error
	2	4	0.1		1.6	3.2	4	-0.8
	1	2	0.1		0.4	0.4	2	-1.6
	3	6	0.1		3.6	10.8	6	4.8
	4	8	0.1		6.4	25.6	8	17.6
					3			5

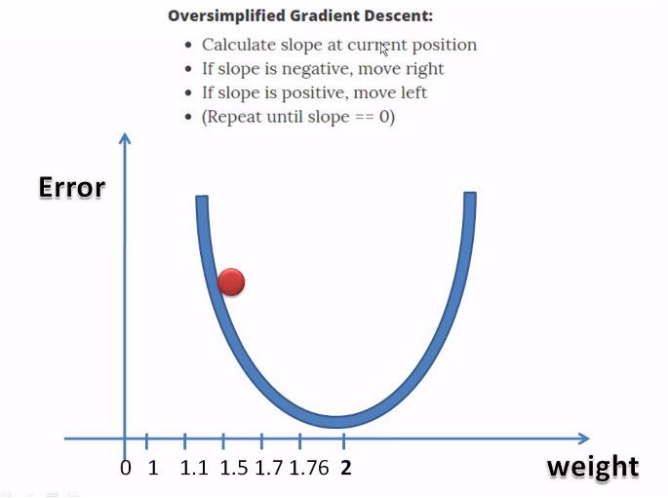
Epoch5	x	y	LR	weights	NewWeights	y_pred	y_test	Error	
	2	4	0.1	1.90625	1.98125	3.9625	4	-0.0375	individual Row Error
	1	2	0.1	1.90625	1.925	1.925	2	-0.075	individual Row Error
	3	6	0.1	1.90625	2.075	6.225	6	0.225	individual Row Error
	4	8	0.1	1.90625	2.20625	8.825	8	0.825	individual Row Error
					2.046875			0.234375	Epoch Error





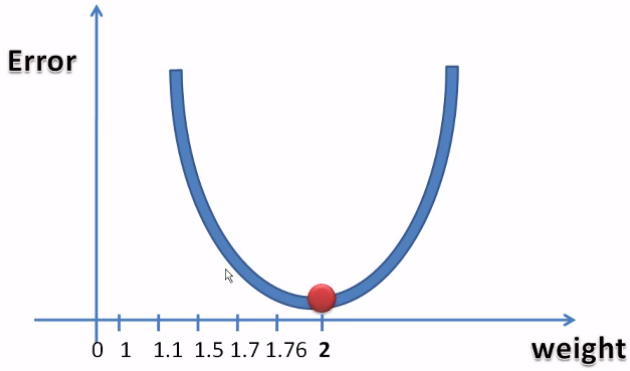
Mini batch is good approach





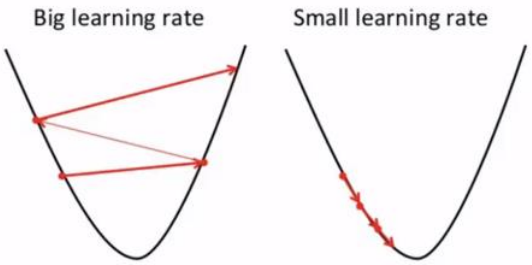
Oversimplified Gradient Descent:

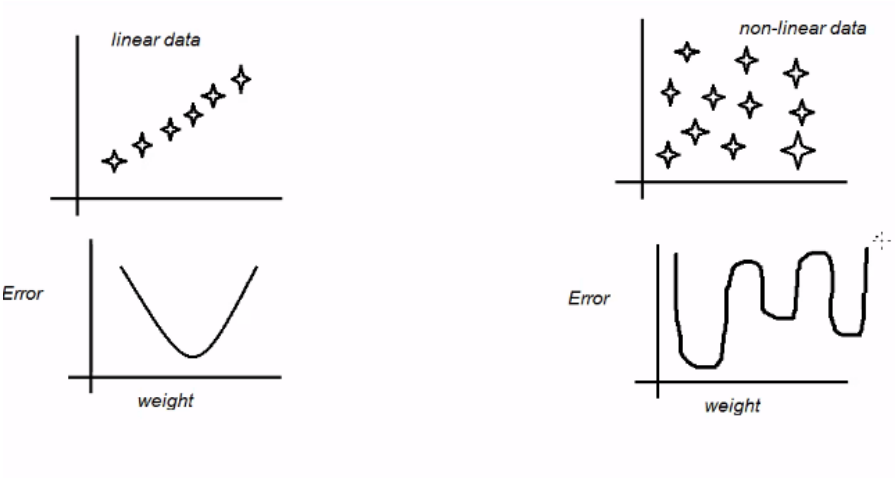
- Calculate slope at current position
- If slope is negative, move right
- If slope is positive, move left
- (Repeat until slope == 0)

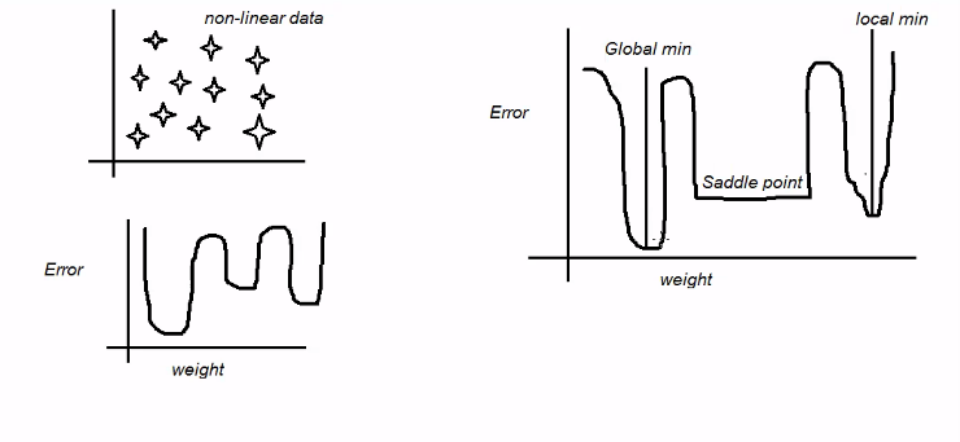


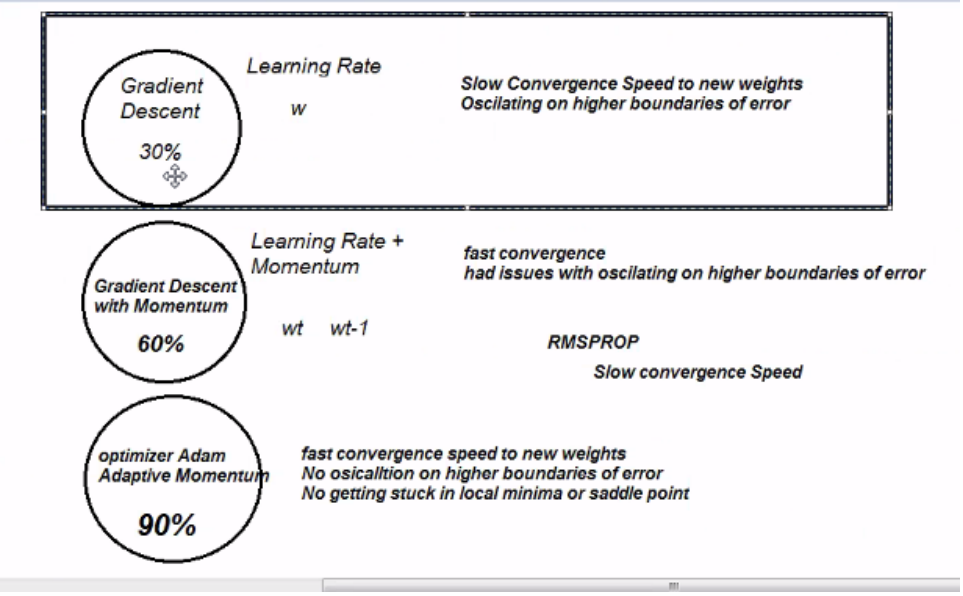
Learning rate

1. **if it is too small, then the model will take some time to learn.**
2. **if it is too large, model will converge as our pointer will shoot and we'll not be able to get to minima.**







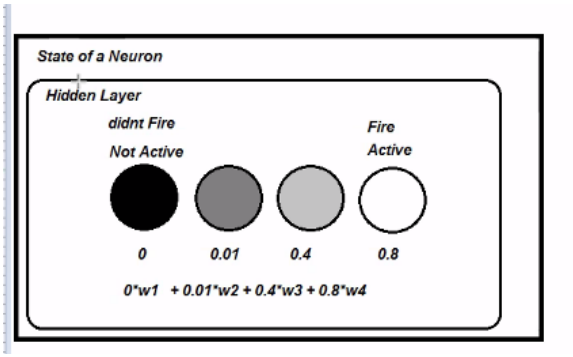


Adaptive Momentum

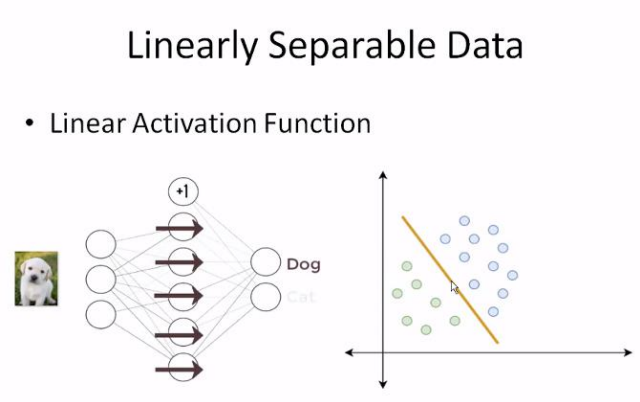
- Adam method is used to accelerate the gradient descent algorithm by taking into consideration the exponentially weighted average of the gradients.

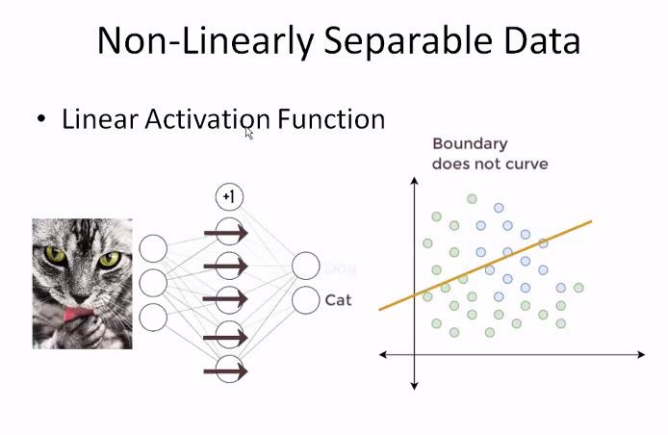
- **new weight ←(old weight)-(learning rate)(gradient)**
- **new weight ←(old weight)-(learning rate)(gradient) + past gradient**
- **(accumulator)←(old accumulator)(momentum)+gradient**
- **momentum → weighted average of past gradients**
- **new weight ←(old weight)-(learning rate)(accumulator)**

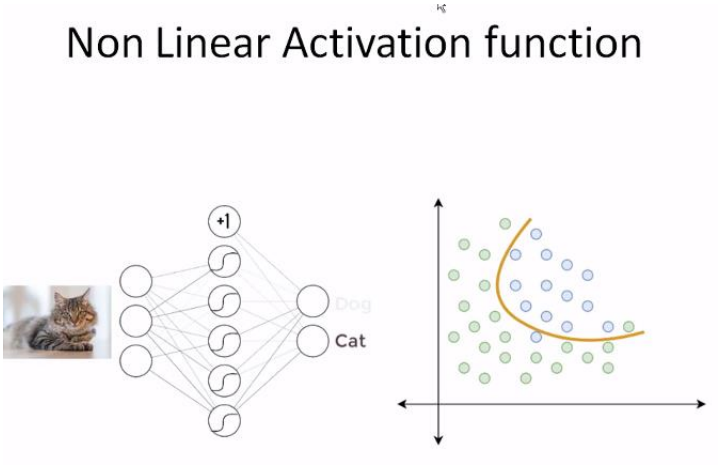
Not fire means not contirbute

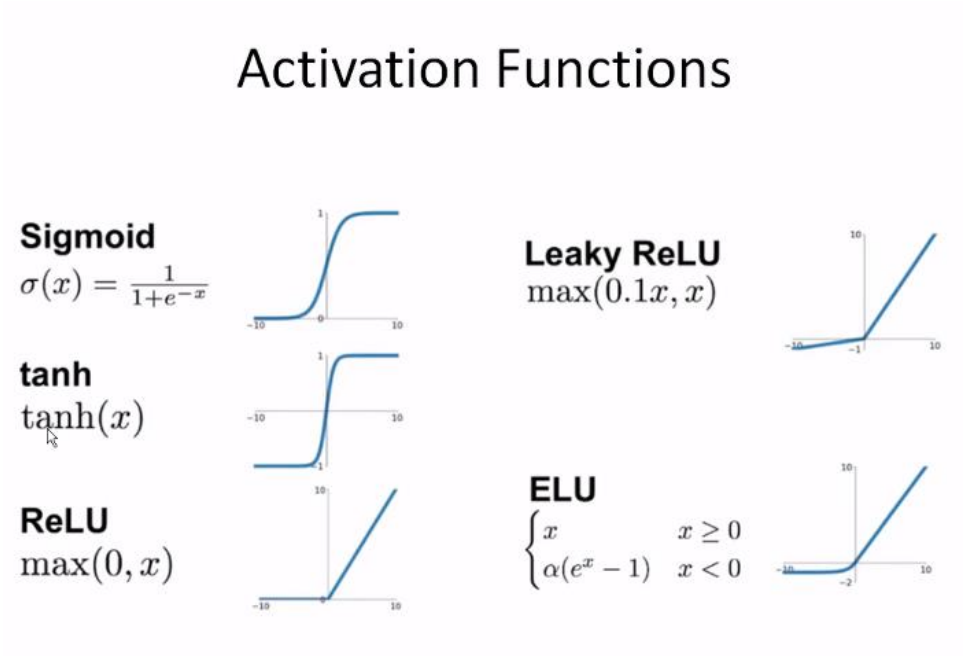


Activation Functions:









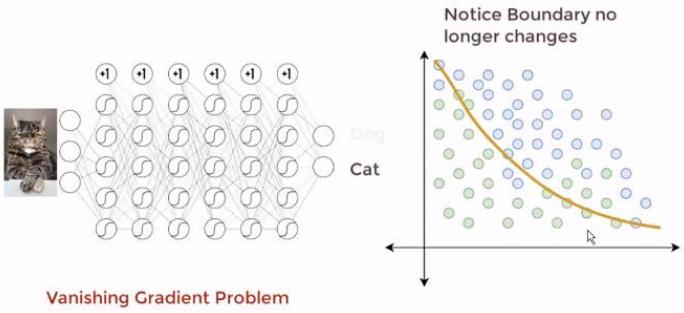
Sigmoid – squeezing in nature – Range – 0 to 1 - Put in problem in case of complex situation

Tanh –Range: -1 to 1 - squeezing in nature - Put in problem in case of complex situation

Vanishing gradient problem due to squeezing nature

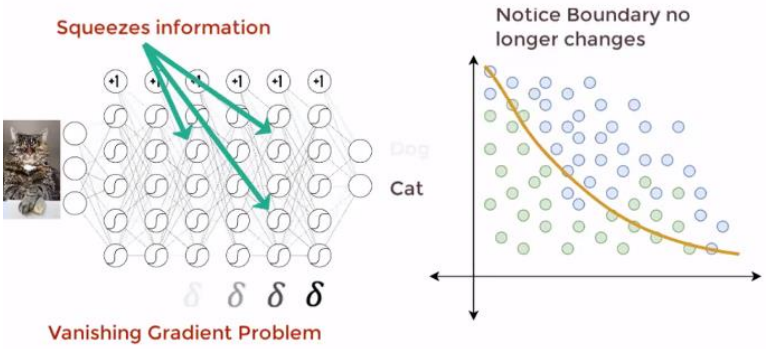
ReLU - (0, x)

Even if the model goes through many examples it doesnt learn much



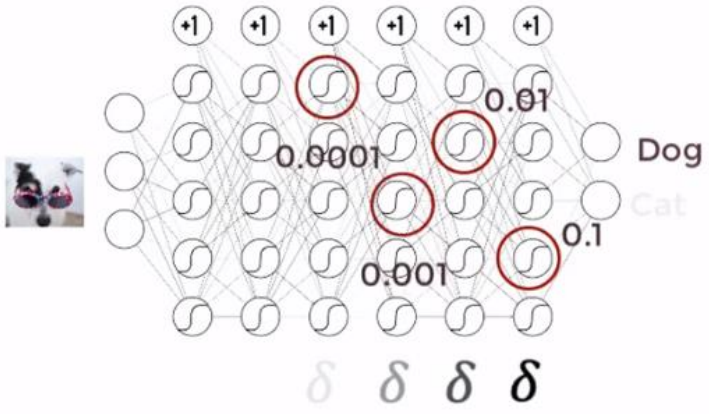
Vanishing Gradient Problem

During backpropogation the gradients become smaller and smaller and smaller until they eventually vanish no gradients means no learning



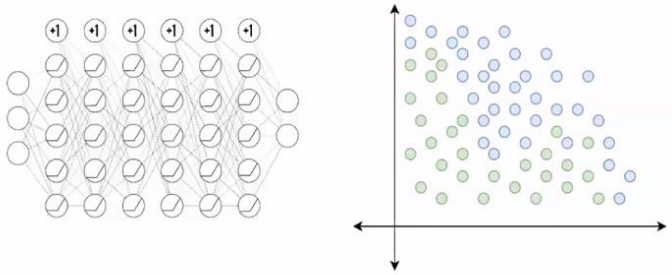
Vanishing Gradient Problem

Vanishing Gradient

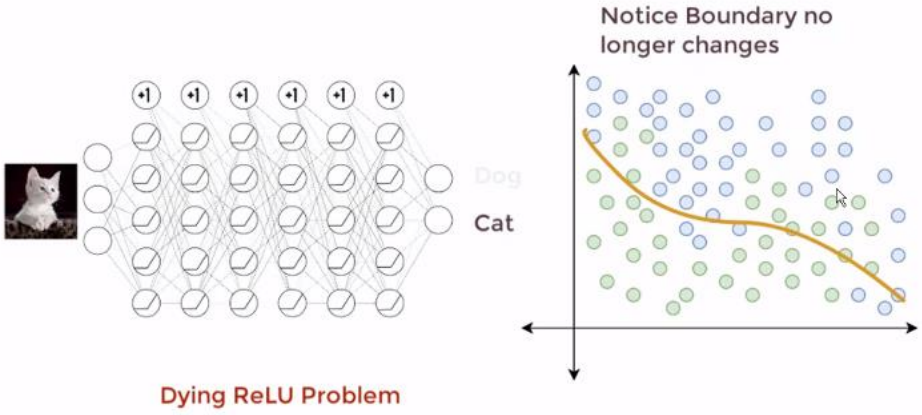


Vanishing Gradient Problem

Remedy to vanishing gradient is to use activation function that doesnot squeeze values, like ReLu

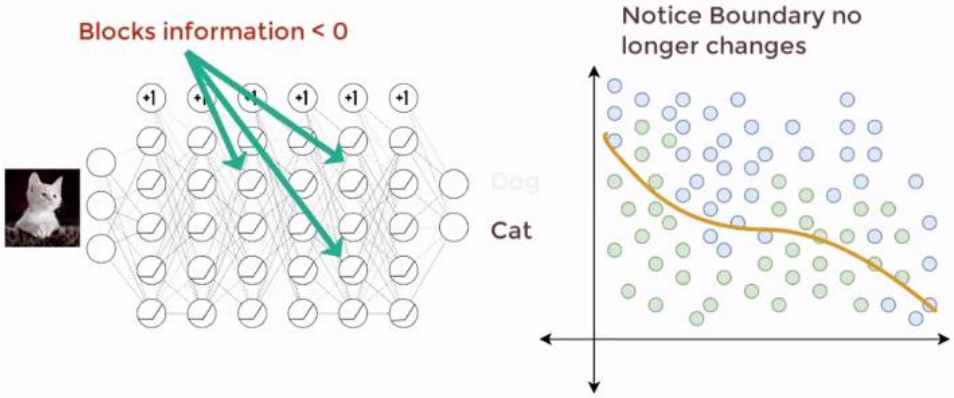


During backpropogation the gradients become smaller and smaller and smaller until they eventually vanish no gradients means no learning. Same problem like vanishing gradient

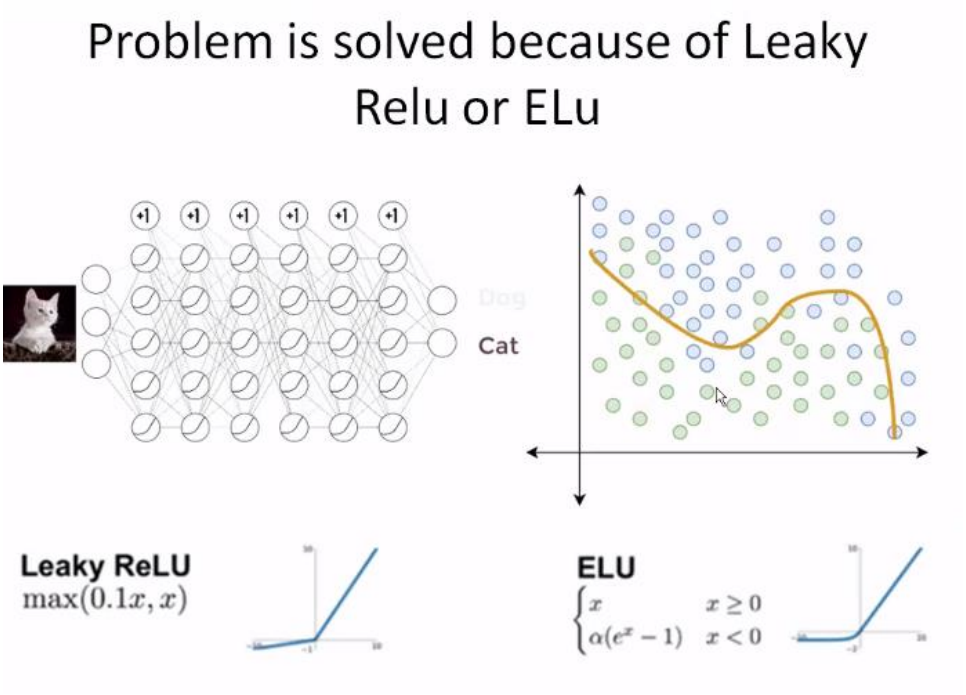


Dying ReLU Problem

Dying ReLu causes as it blocks information less than 0



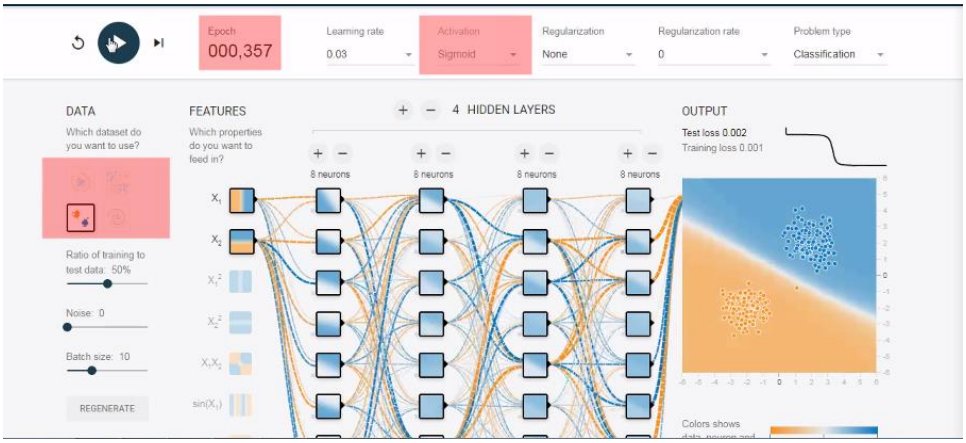
Dying ReLU Problem



Hyper Parameter Tunning (values to be found with trial and error)

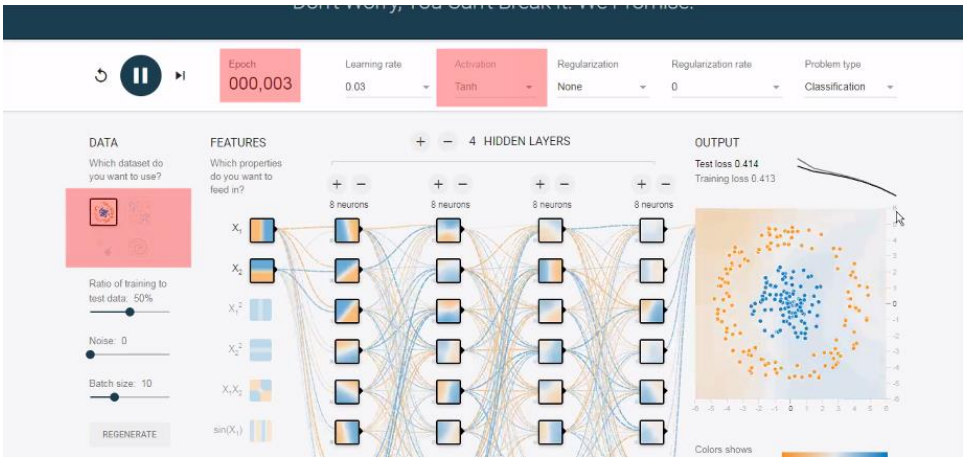
1)Num of Neurons in Hidden Layer
2)Num of Hidden Layers
3)Num of Epochs
4)Activation Functions
5)batch_size

Explain activation function in tensorflow playground:

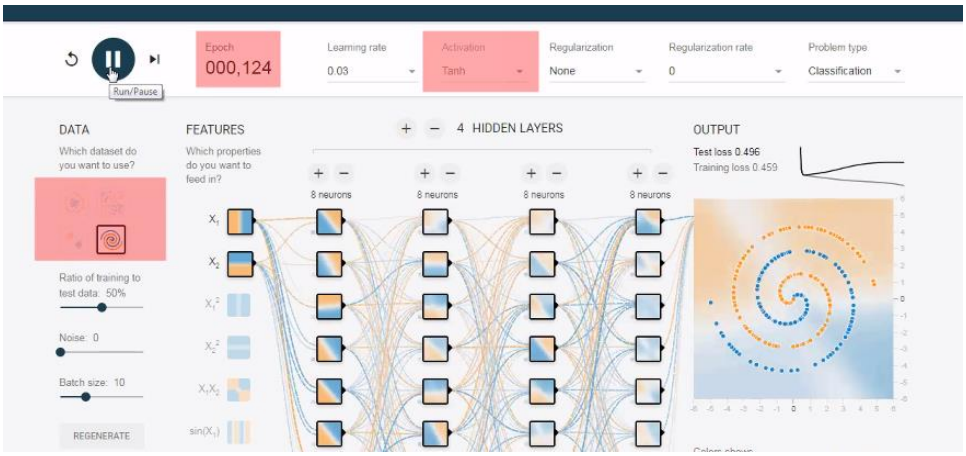


Took too much time to find the boundary

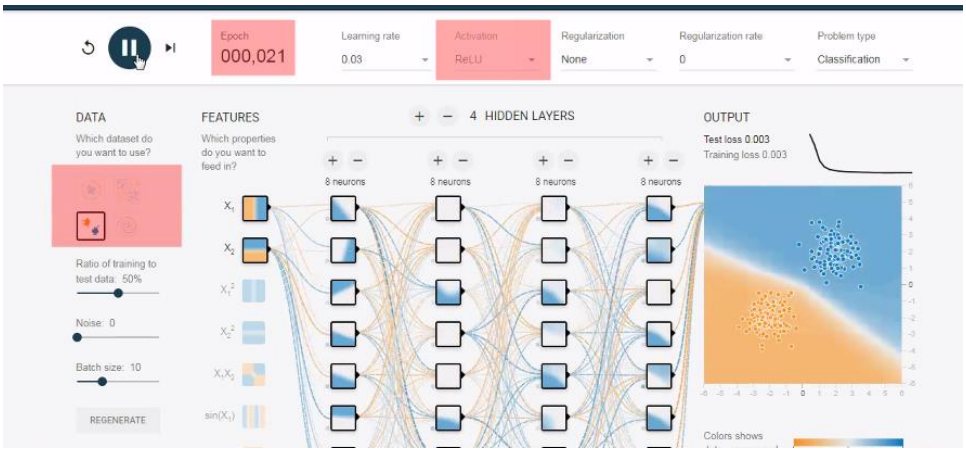
Tanh – advanatge – bigger boundaries

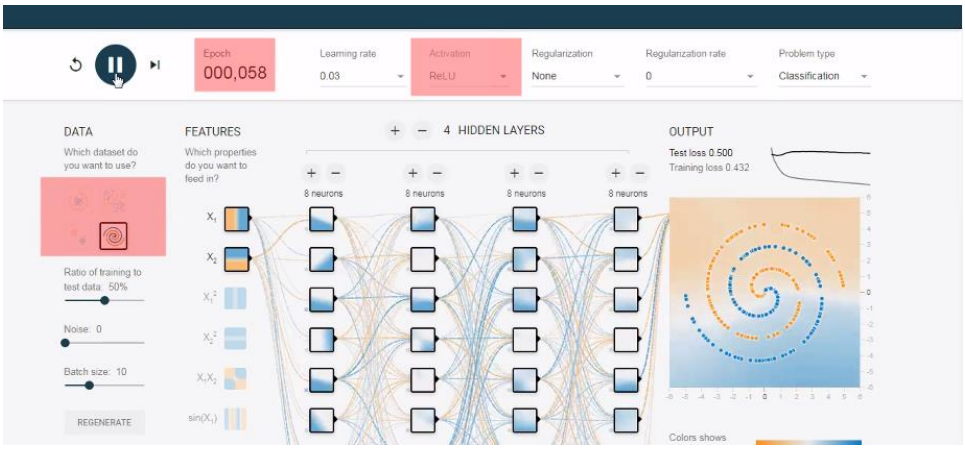


Problem of tanh:



ReLu: Identify the boundary in few numbers of epoch





Guidelines to choose the activation function:

Hyper Parameter Tunning (values to be found with trial and error)

1)Num of Neurons in Hidden Layer
2)Num of Hidden Layers
3)Num of Epochs
4)Activation Functions
5)batch_size

	InputLayer	HL	HL	OutputLayer	LossFunction
Regression		ReLu	ReLu	ReLu	mean_squared_error
Binary-Classification		ReLu	ReLu	Sigmoid	binary_classification
Multi-Class-Classification		ReLu	ReLu	Softmax	categorical_crossentropy

Implementation:

Step 1: Start anaconda prompt

Step 2:

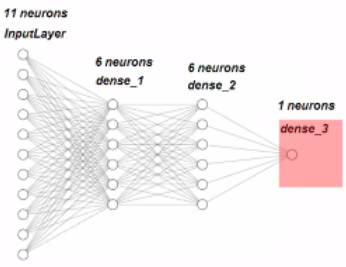
python -m pip install --upgrade pip

pip install keras tensorflow

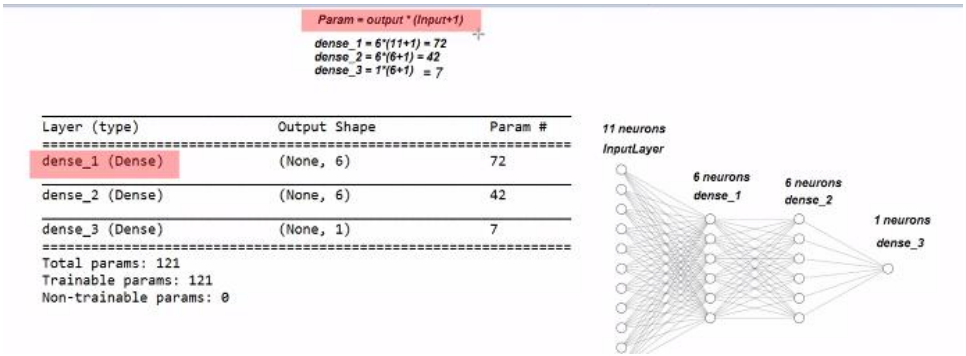
ANN:

Summary:

```
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 6)                 72
_________________________________________________________________
dense_2 (Dense)              (None, 6)                 42
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 7
=================================================================
Total params: 121
Trainable params: 121
Non-trainable params: 0
```



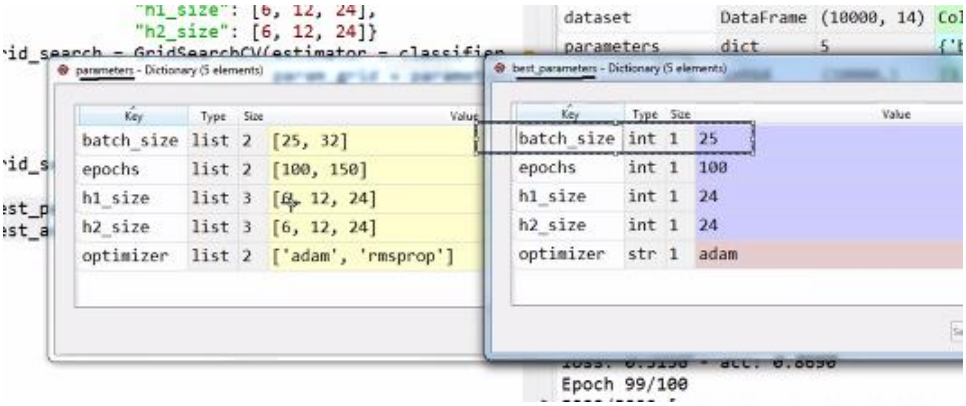
How to get the param count?

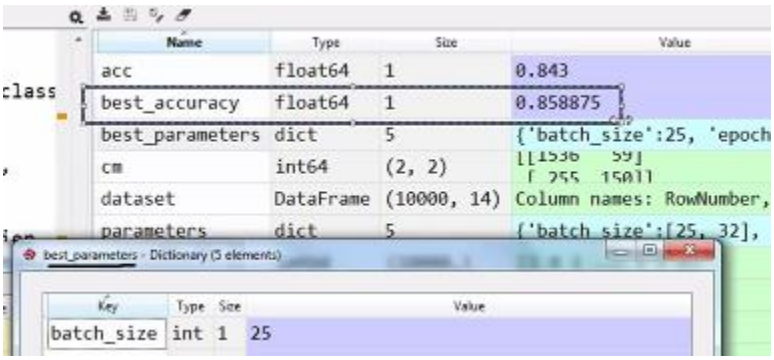


How many times I need to run?

Accuracy not stable – For that use GridSearchCV(sklearn Obj) - Trail and error with k fold cross validation

Neural network Keras object ------typecast --------sklearn obj





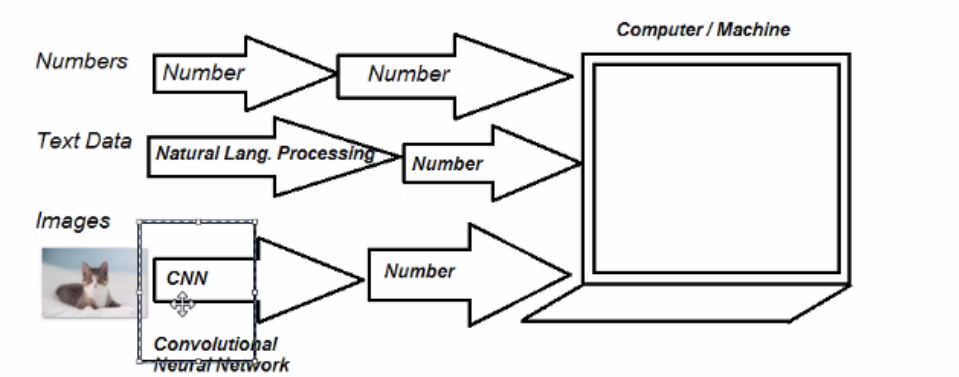
SGD:

Batch size 1 -> Slow learning -> Overfitting

BGD:

Batch size 8000 - > Fast learning -> Accuracy didn't increase – stays at 79% -> Too much of generalization

CNN:


Computer / Machine
Numbers
Number
Number
Text Data
Natural Lang. Processing
Number
Images
CNN
Number
Convolutional
Neural Network

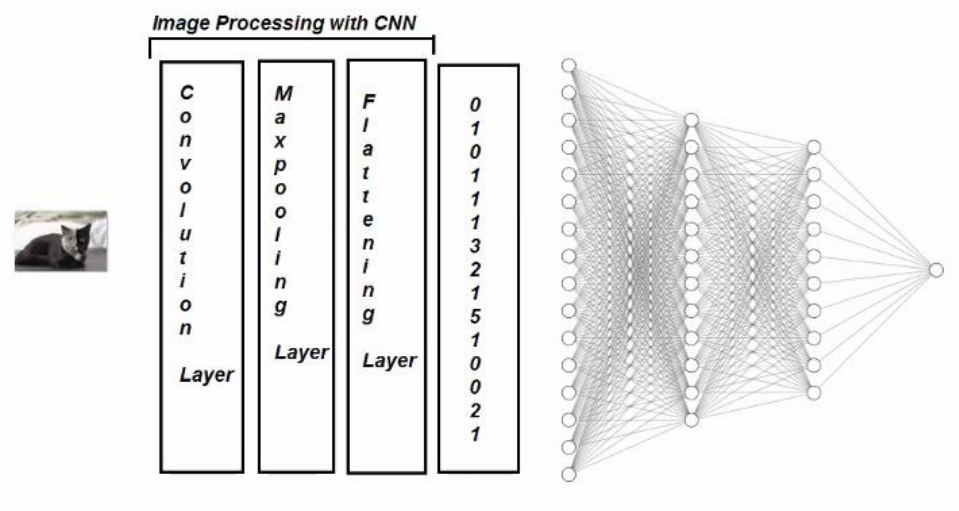


Image Processing with CNN
Convolution Layer
Maxpooling Layer
Flattening Layer
01011132151002 1

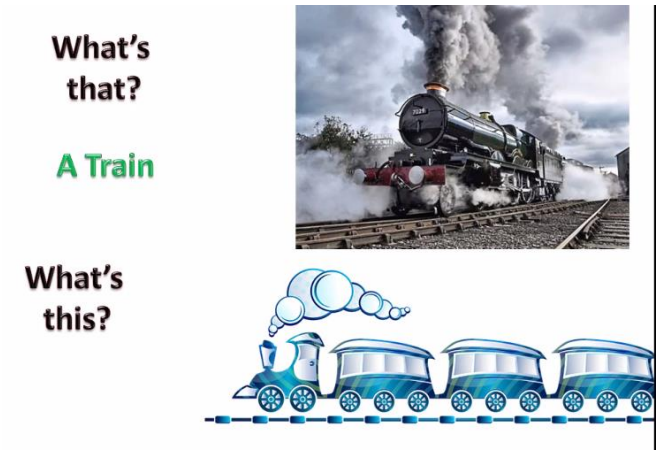

X_train
Input Data
y_train
Output Data
Dog
Cat
Dog
Cat
Cat
X_test
y_pred
????
????

Convolution Neural Network

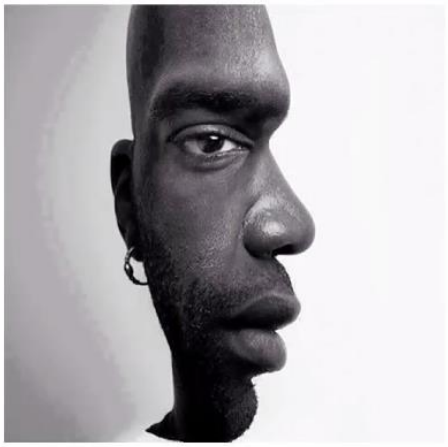
- Facebook uses neural nets for their automatic tagging algorithms
- Google for their photo search
- Amazon for their product recommendations
- Pinterest for their home feed personalization
- Instagram for their search infrastructure.

How Humans identify images??

- For humans, this task of recognition is one of the first skills we learn from the moment we are born and is one that comes naturally and effortlessly as adults.
- Most of the time we are able to immediately characterize the scene and give each object a label, all without even consciously noticing.

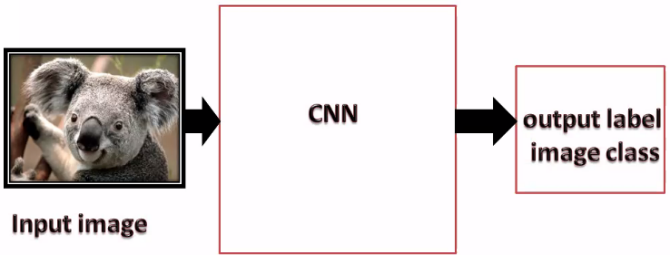


illusions

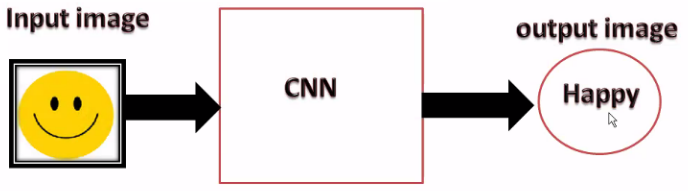


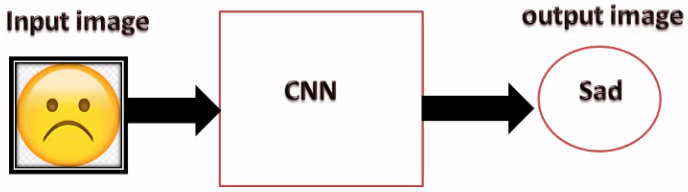
illusions

Convolution Neural Network



Convolution Neural Network





How Does Computer See an image??

When a computer sees an image ,it will see an array of pixel values. Depending on the resolution and size of the image, it will see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values).

What we see

What computers see

How Does Computer See an image??

What we see

What computers see

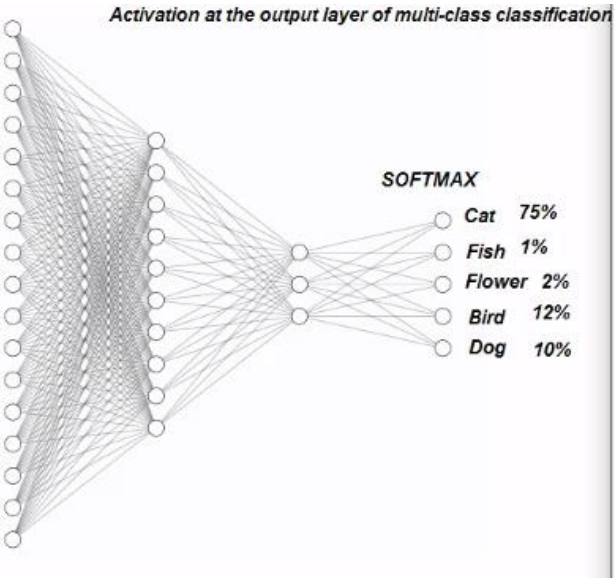
These numbers, when we perform image classification, are the only inputs available to the computer.

The idea is that you give the computer this array of numbers and it will output numbers that describe the probability of the image being a certain class

(80% for flower, 15% for sky etc).

Multi-Class Classification

	y0	y1	y2	y3	y4	
	class0	class1	class2	class3	class4	
	Cat	Fish	Flower	Bird	Dog	
Input Image						
	75%	1%	2%	12%	10%	=100%



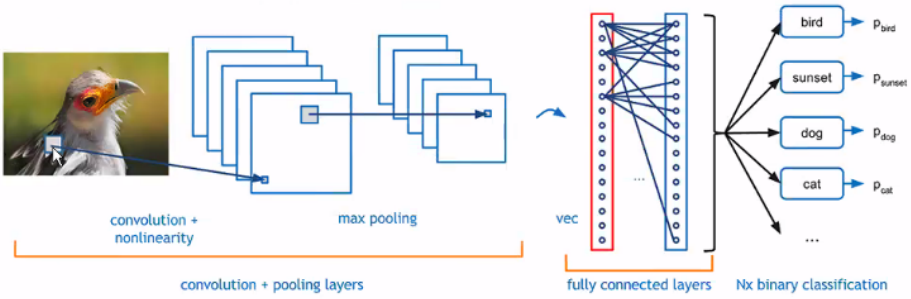
	Cat	Fish	Flower	Bird	Dog
			y_train		
X_train	y0	y1	y2	y3	y4
	1	0	0	0	0
	0	0	0	0	1
	1	0	0	0	0
	0	0	0	0	1

What We Want the Computer to Do

- we want the computer to do is to be able to differentiate between all the images it's given and figure out the **unique features** that make a dog a dog , that make a cat a cat or that make a flower a flower .

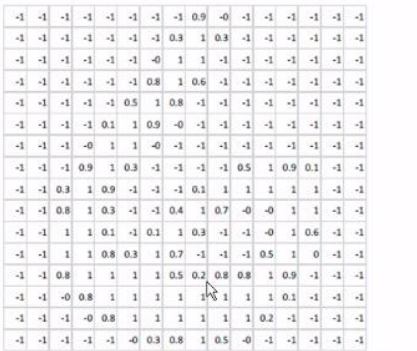
Convolution Neural Network

CNNs take the image, pass it through a series of convolutional, nonlinear, pooling (downsampling), and fully connected layers, and get an output.



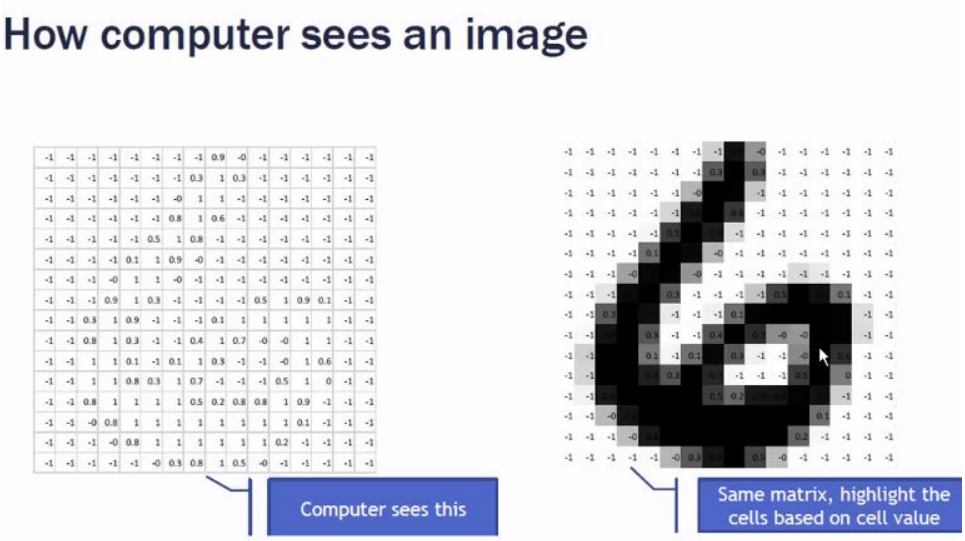
How computer sees an image

Humans see this

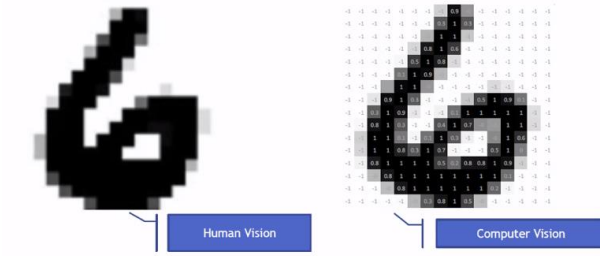


Computer sees this

How computer sees an image

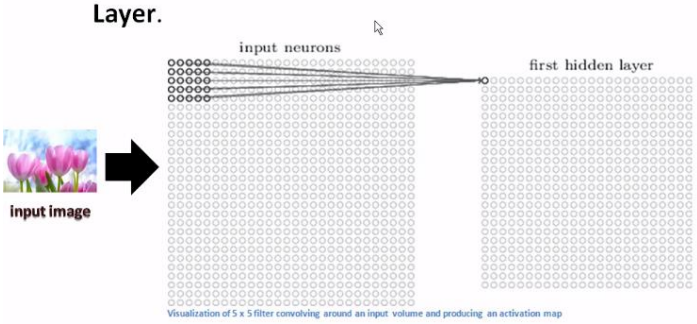


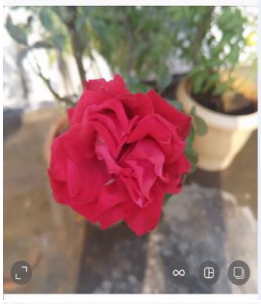
How computer sees an image

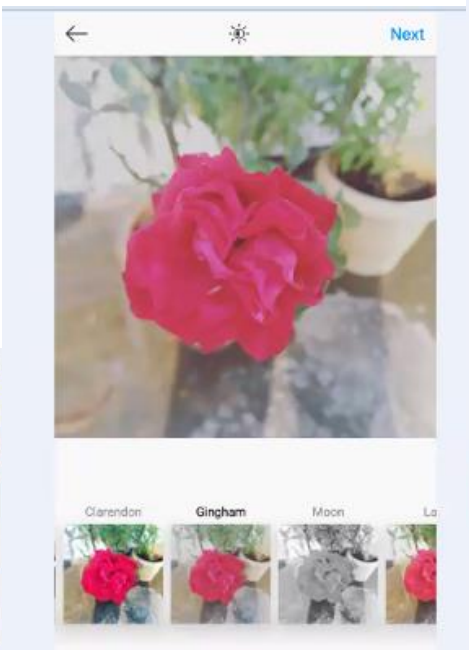


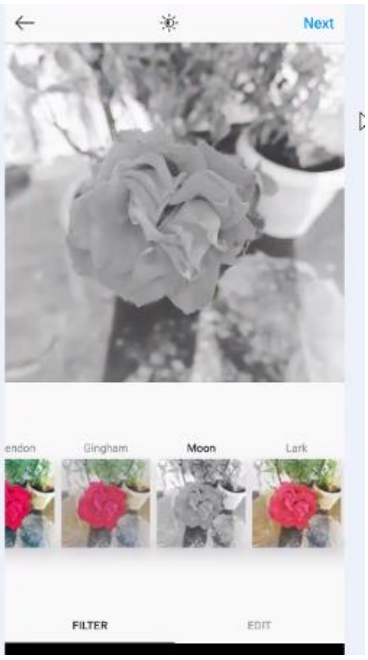
Convolution Neural Network

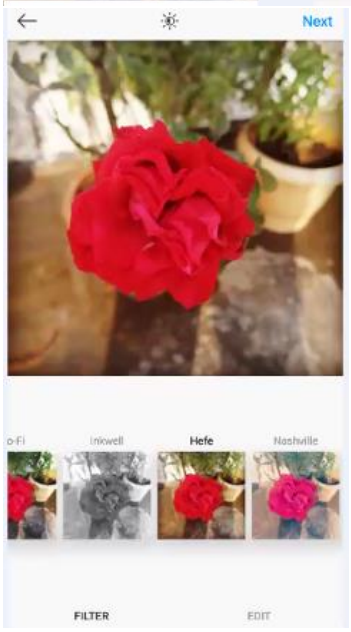
- **First Layer – Math Part**
  - The first layer in a CNN is always a **Convolutional Layer**.

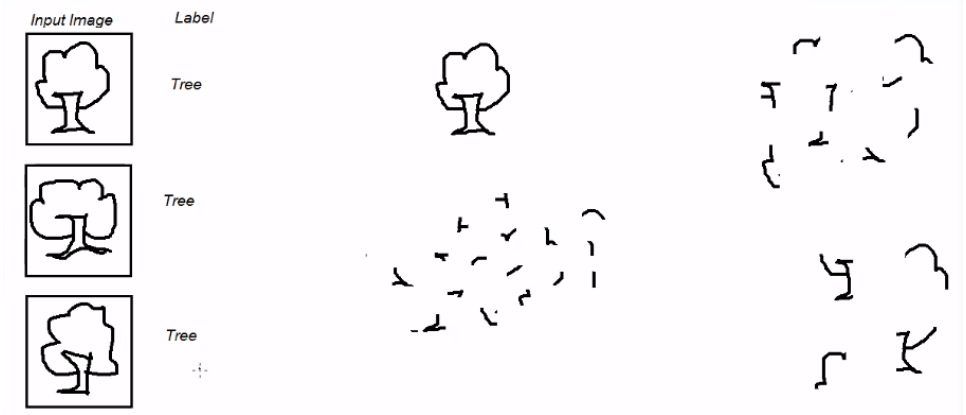


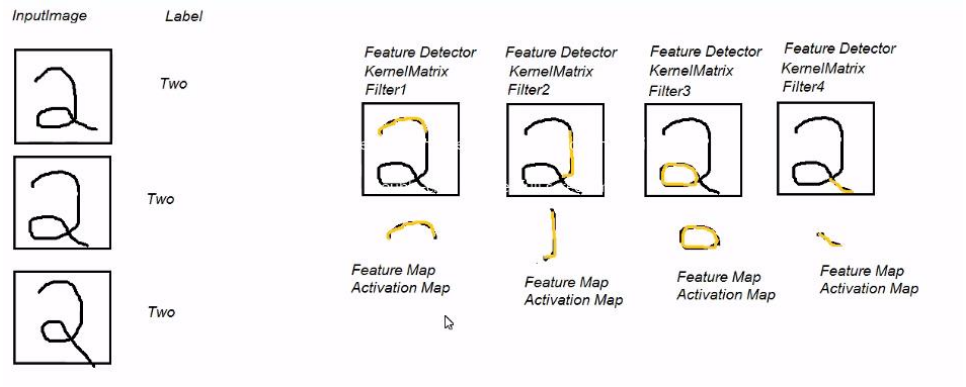



Next
Clarendon
Gingham
Moon


Next
Gingham
Moon
Lark
FILTER
EDIT

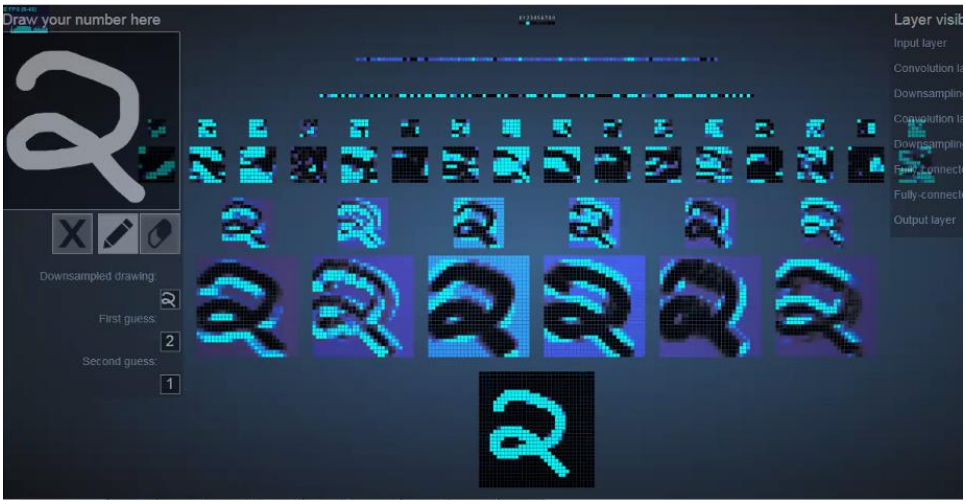

Next
Inkwell
Hefe
Nashville
FILTER
EDIT

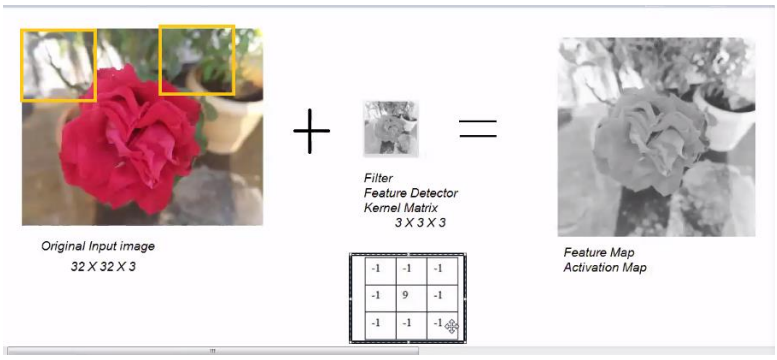

Input Image
Label
Tree
Tree
Tree



http://www.cs.cmu.edu/~aharley/nn_vis/cnn/2d.html

This identify the edges:





Kernel Matrix examples

1	1	1
1	1	1
1	1	1

Unweighted 3x3 smoothing kernel

0	1	0
1	4	1
0	1	0

Weighted 3x3 smoothing kernel with Gaussian blur

0	-1	0
-1	5	-1
0	-1	0

Kernel to make image sharper

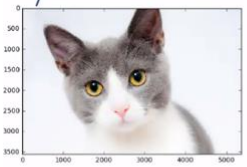
-1	-1	-1
-1	9	-1
-1	-1	-1

Intensified sharper image

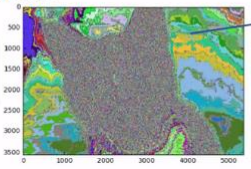
Gaussian Blur

Sharpened image

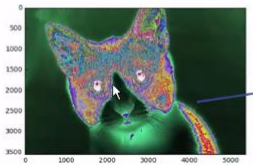
Convoluted features



Different filters capture different features



A filter to capture overall shape and edges



A filter to ignore white spaces on image

The depth in convolution layer

- Every filter gives us a resultant matrix (activation map)

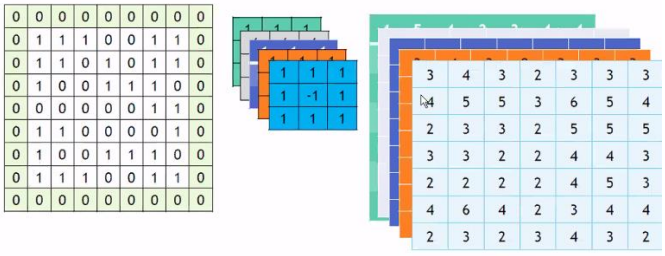
0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	1	1	0
0	1	1	0	1	0	1	1	0
0	1	0	0	1	1	1	0	0
0	0	0	0	0	0	1	1	0
0	1	1	0	0	0	0	1	0
0	1	0	0	1	1	1	0	0
0	1	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0	0

1	1	1
1	1	1
1	1	1

4	5	4	2	3	4	4
5	6	5	4	6	6	5
3	3	3	3	6	6	5
3	3	2	2	4	5	4
3	3	2	2	4	5	4
5	6	4	3	4	5	4
3	4	3	3	4	4	3

The depth in convolution layer

- If we apply 10 different filters then we will get 10 resultant matrices. The depth is 10



Max Pooling

0	1	0	0	0
0	1	1	1	0
1	0	1	2	1
1	4	2	1	0
0	0	1	2	1

Pooling

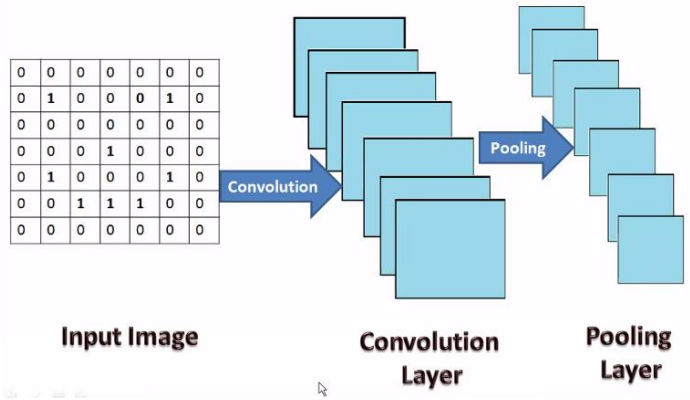
1	1	0
4	2	1
0	2	1

Feature Map

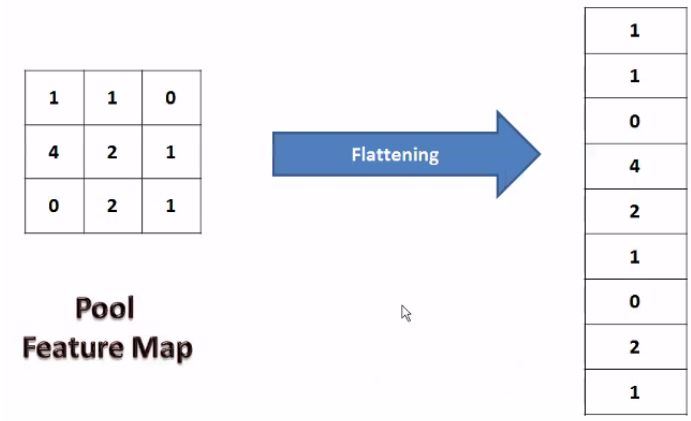
Pool Feature Map

Max pooling is usually done with 2x2 windows and stride 2, so as to downsample the feature maps

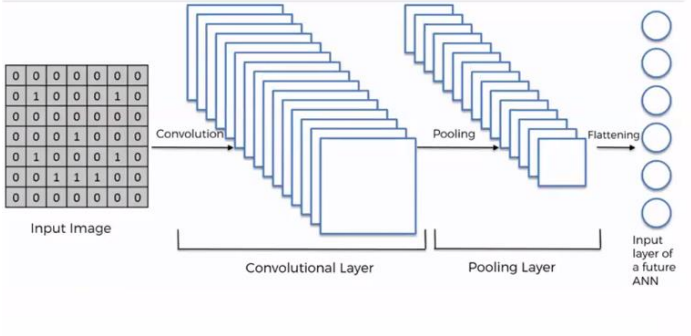
Max Pooling



Flattening



Flattening



Full Connection