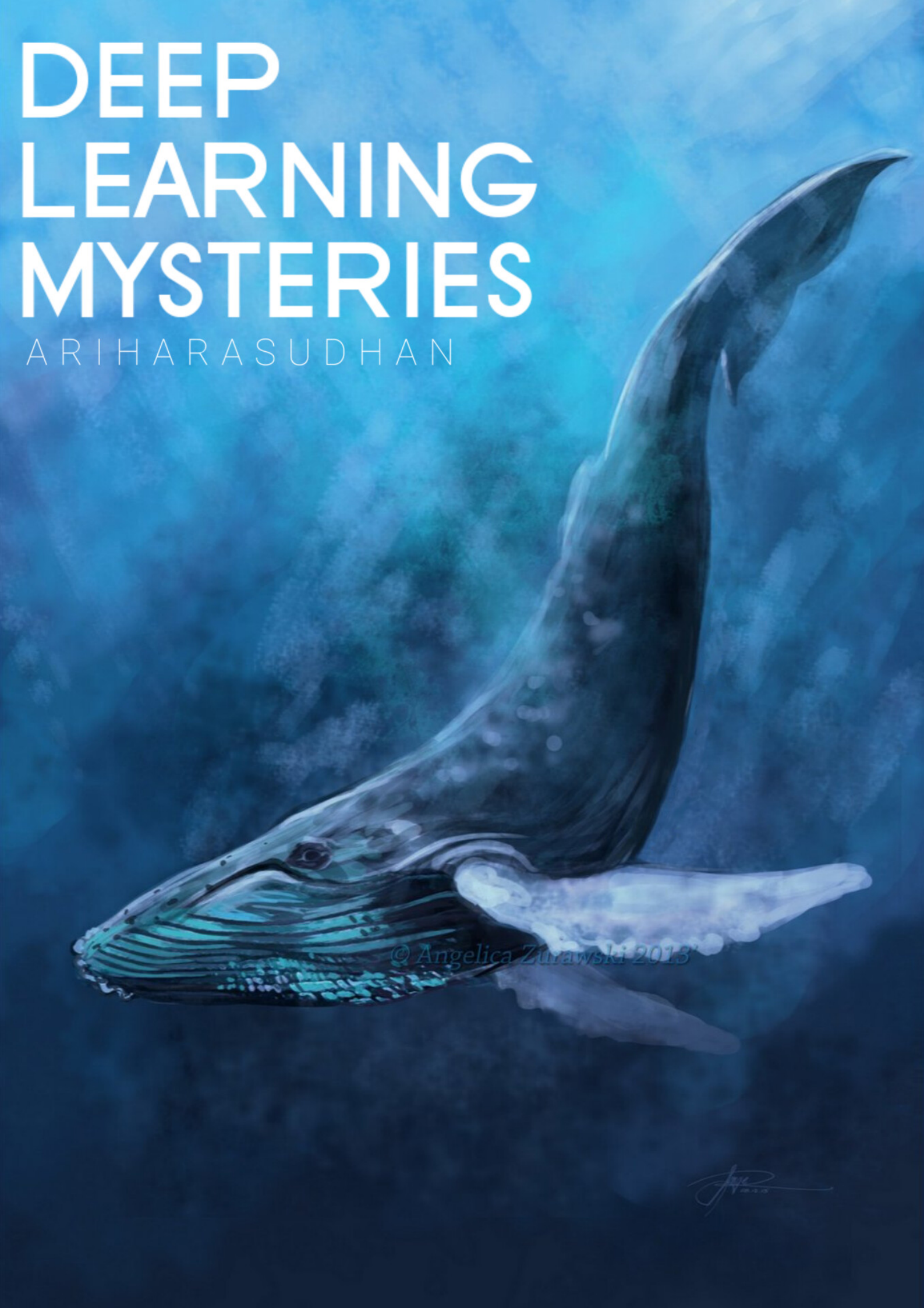
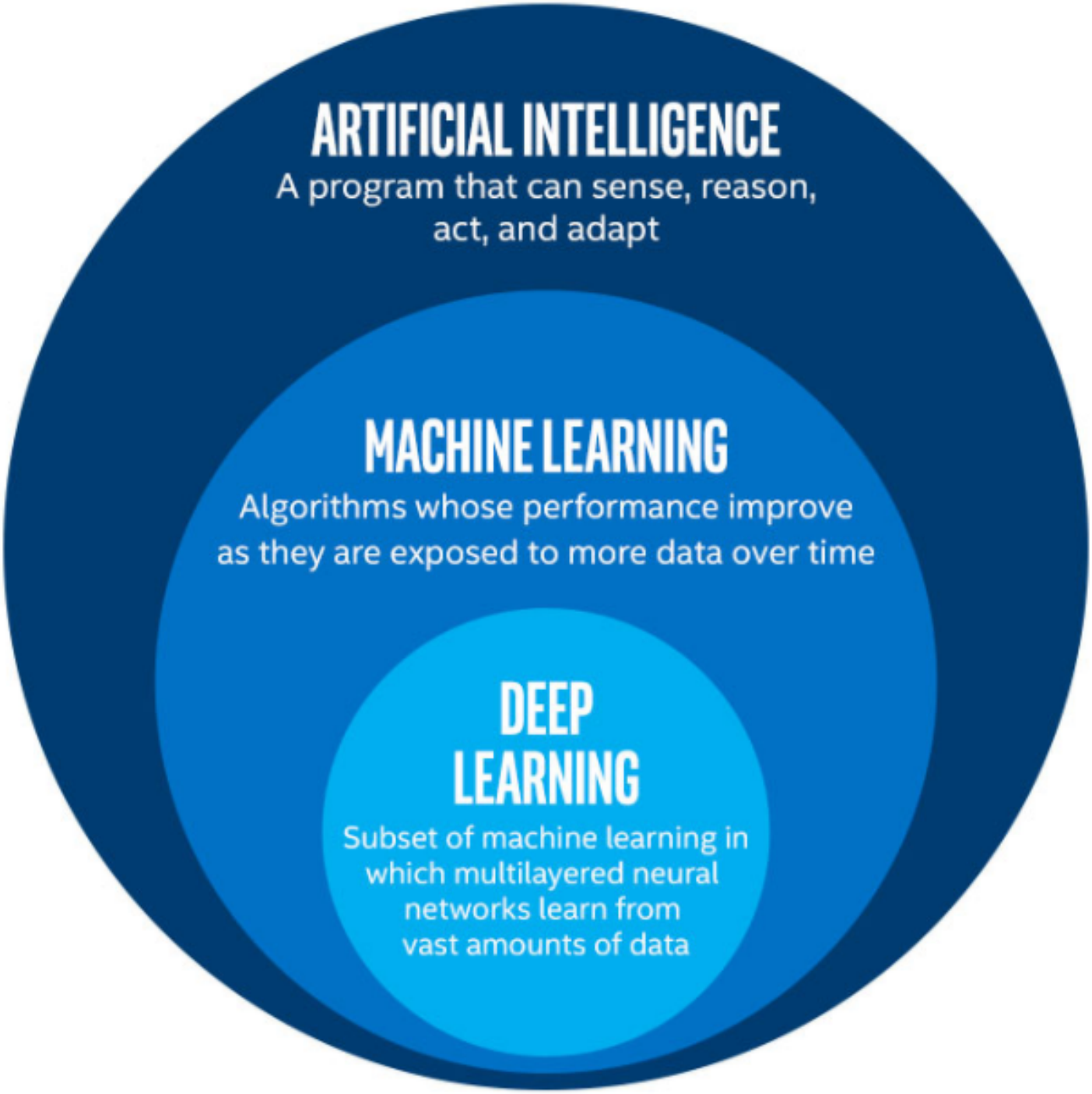
DEEP LEARNING MYSTERIES

ARIHARASUDHAN

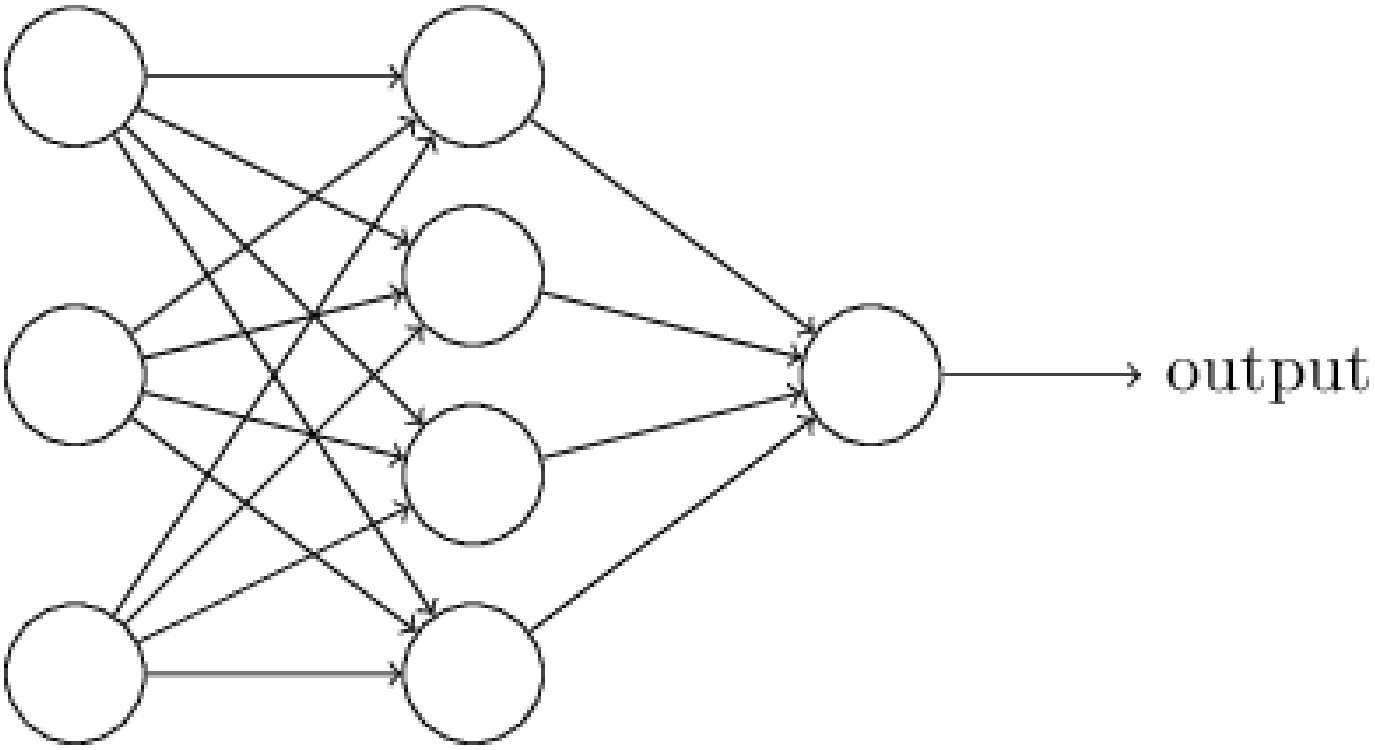


THE INTRODUCTION



The image shown above is shown always whenever we want to learn what actually the deep learning is.

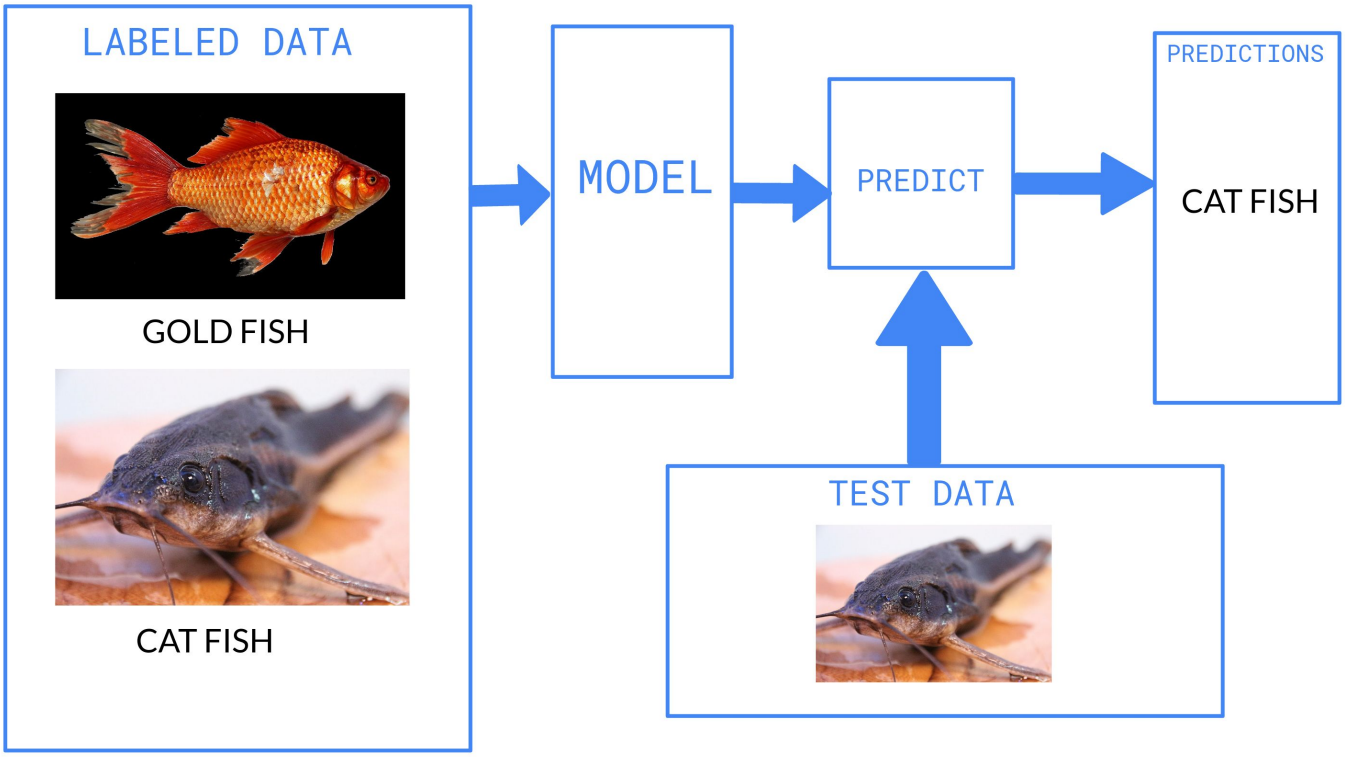
THE SIMULATED NEURAL NETWORKS



The Simulated Neural Networks are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

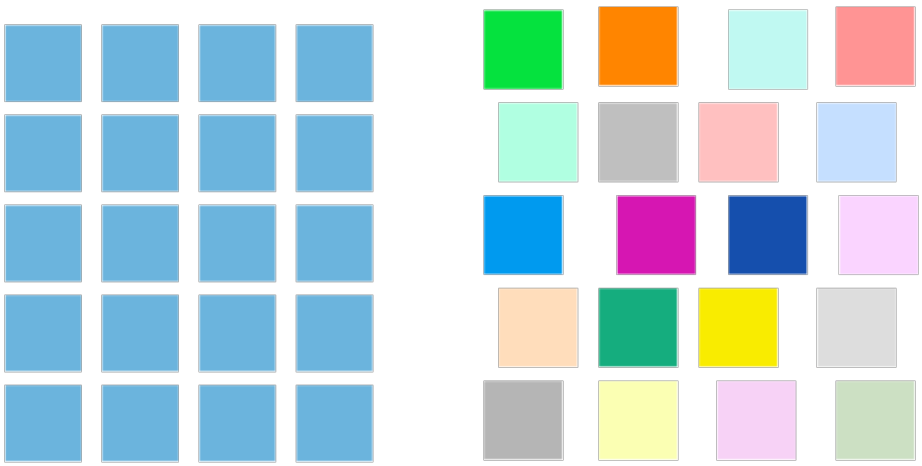
SUPERVISED LEARNING

Supervised learning is a process of providing input data as well as correct output data to the machine learning model. The aim of a supervised learning algorithm is to find a mapping function to map the input variable(x) with the output variable(y). In supervised learning, models are trained using labelled dataset, where the model learns about each type of data. Once the training process is completed, the model is tested on the basis of test data (a subset of the training set), and then it predicts the output.



STRUCTURED VS. UNSTRUCTURED DATA

The data which is to the point, factual, and highly organized is referred to as structured data. It is quantitative in nature, i.e., it is related to quantities that means it contains measurable numerical values.



Unstructured data is the data that lacks any predefined model or format. It resides in various different formats like text, images, audio and video files, etc. It is qualitative in nature and sometimes stored in a non-relational database or NO-SQL.

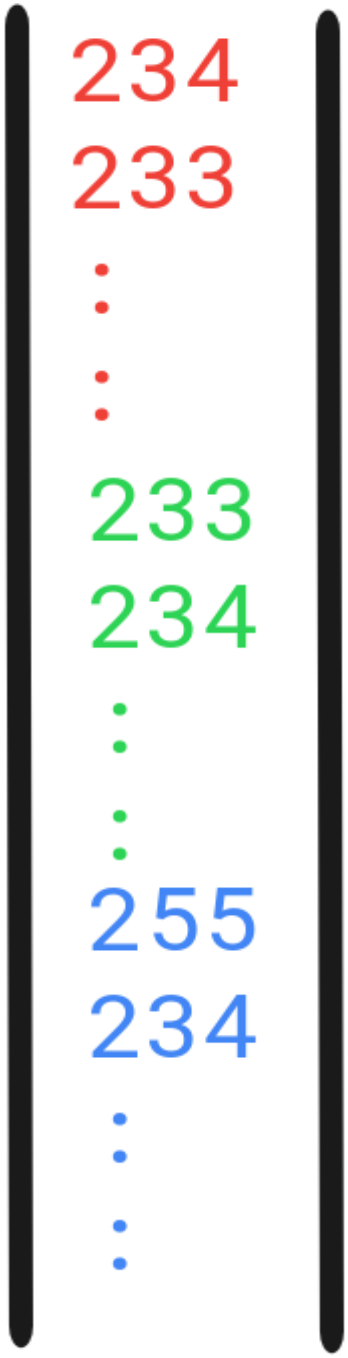
BINARY CLASSIFICATION

Binary classification classifies a data as 1 or 0 (to be specific, CAT or DOG :-).

FISH : 1 NON-FISH : 0

An Image is represented as 3D Array for representing 3 channels such as RGB. If the image has a dimension of 240x320, then totally, there will be 240x320x3 pixel values ranging from 0 to 255 for representing the entire image. The image which is full of pixel intensity values is converted into a Feature Vector.

The image data is unrolled into a vector that consists of the entire values as shown below.



NOTATIONS TO USE :

(x,y) - where x is the Feature VECTOR,y is the set of labels{0,1}

Let m be the training example , $m = \{(x^i,y^i)^j\}$

m_{test} be the number of test examples. We can represent the training set as a row in a vector. Similarly, the labels can also be represented.

Training = $[x^1,x^2,..,x^j]$

Labels = $[Y^1,Y^2,..,Y^j]$

In Python, the shape of these is (1,j).

LOGISTIC REGRESSION

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class.

Independent variables : The input characteristics or predictor factors applied to the dependent variable's predictions.

Dependent variable: The target variable in a logistic regression model, which we are trying to predict.

Logistic function : The formula used to represent how the independent and dependent variables relate to one another. The logistic function transforms the input variables into a probability value between 0 and 1, which represents the likelihood of the dependent variable being 1 or 0.

Let the independent input features be

$$X = \begin{bmatrix} x_{11} & \dots & x_{1m} \\ x_{21} & \dots & x_{2m} \\ \vdots & \ddots & \vdots \\ x_{n1} & \dots & x_{nm} \end{bmatrix}$$

And the dependent variable is Y having only binary value i.e 0 or 1.

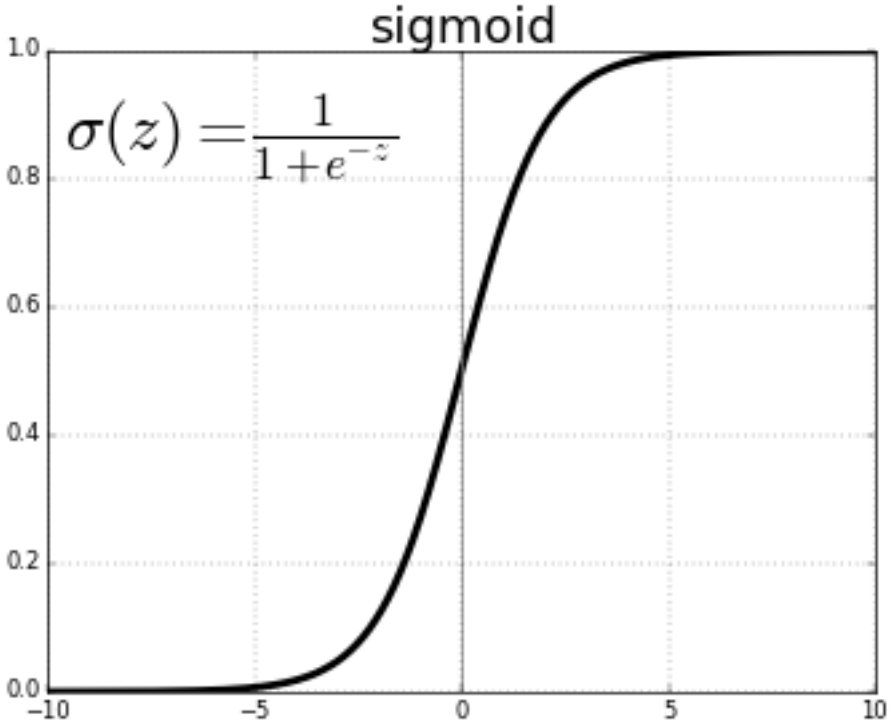
$$Y = \begin{cases} 0 & \text{if } Class\ 1 \\ 1 & \text{if } Class\ 2 \end{cases}$$

Then apply the multi-linear function to the input variables X

$$z = (\sum_{i=1}^{n} w_i x_i) + b$$

Here x_i is the ith observation of X, $w_i = [w_1, w_2, w_3, \cdots, w_m]$ is the weights or Coefficient and b is the bias term also known as intercept. simply this can be represented as the dot product of weight and bias. $z = w \cdot X + b$

Whatever we discussed above is the linear regression. Now we use the sigmoid function where the input will be z and we find the probability between 0 and 1. i.e predicted y.



THE LOG LOSS FUNCTION

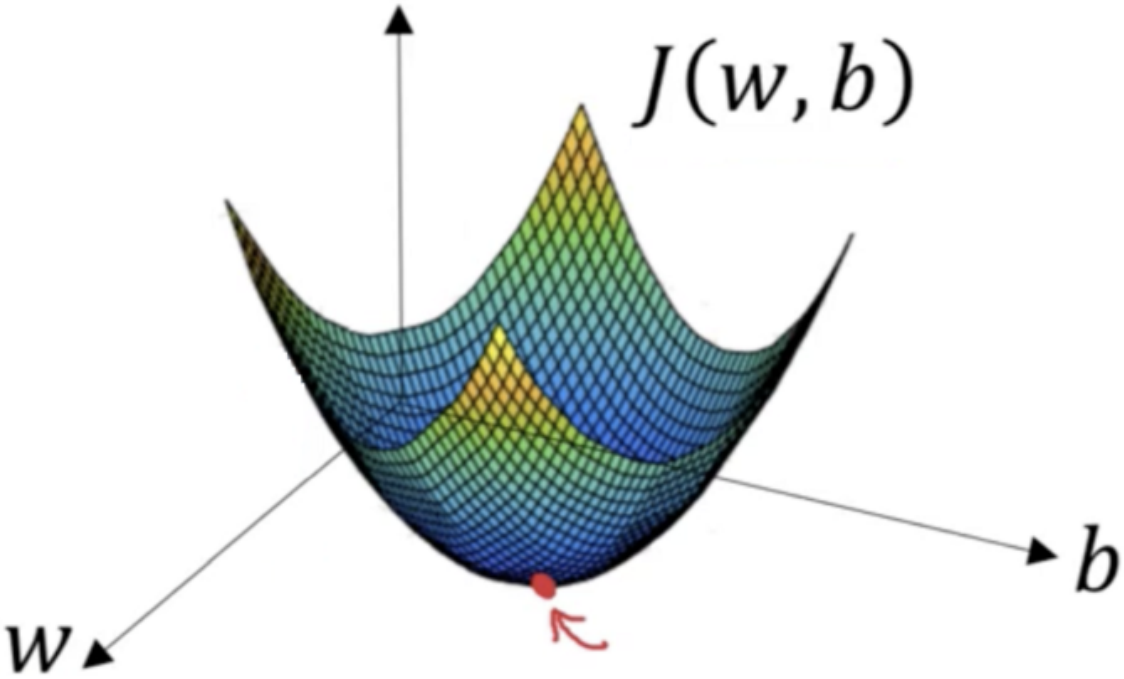
Log-loss is **indicative of how close the prediction probability is to the corresponding actual/true value** (0 or 1 in case of binary classification). The more the predicted probability diverges from the actual value, the higher is the log-loss value. The Cost Function is the average loss over the entire training which can be represented as,

$$Cost = -\frac{1}{n}\sum_{i=1}^{n}[y_i \log \hat{y}_i + (1-y_i)\log(1-\hat{y}_i)]$$

The cost function can be represented as J(w,b). There are some other loss functions such as Mean Squared Error which is not applied for the problems like Logistic Regression since it may cause some problems while optimizing. The loss value is high when the predicted output is not even close to the ground truth and vice versa.

THE GRADIENT DESCENT

We need to find the values for weights and bias such that we can reduce the cost. For this, the gradient descent technique is used.



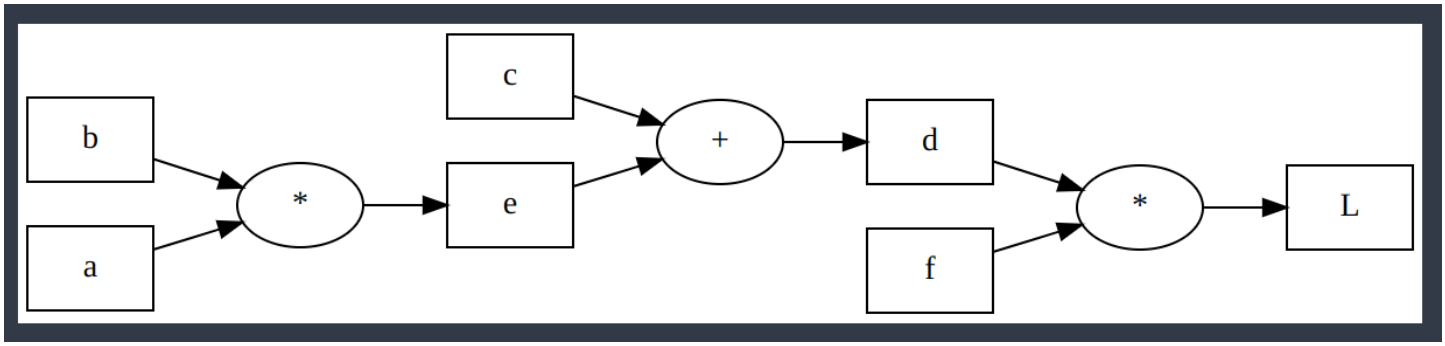
The value of weight and bias should be updated in a way such that it will move towards the global minimum. For that, following is what is followed.

REPEAT UNTIL MINIMUM IS REACHED {
w = w – α∂w [∂w = ∂J(w,b)/∂w]
b = b – α∂b [∂w = ∂J(w,b)/∂b]
}

THE DERIVATIVES USING GRAPH

Let’s draw a computation graph for L shown below.

a*b
d = e+c
L = d*f



The derivatives can be computed as the following : (CHAIN RULE :-)

```
L =  f.d
d(L)/df = d = 4.00
d(L)/dd = f = -2.00

L = f.(c+e)
d(L)/dc = f = -2.00
d(L)/de = f = -2.00

L = f.(c+a*b)
d(L)/da = f.b = 6
d(L)/db = f.a = -4
```

VECTORIZATION

Using the conventional loops, it is almost very very hard to address a huge amount of data. Here the concept of vectorization speaks. Vectorization is a technique of implementing array operations without using for loops. Instead, we use functions defined by various modules which are highly optimized that reduces the running and execution time of code. Vectorized array operations will be faster than their pure Python equivalents, with the biggest impact in any kind of numerical computations.

Advanced Vector Extensions (Intel® AVX) is a set of instructions for doing Single. Instruction Multiple Data (SIMD) operations.

Logistic Regression-Vectorization

```
import numpy as np
from numpy import log,dot,exp,shape
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
X,y = make_classification(n_featues=4)
from sklearn.model_selection import train_test_split

X_tr,X_te,y_tr,y_te = train_test_split(X,y,test_size=0.1
def standardize(X_tr):
    for i in range(shape(X_tr)[1]):
        X_tr[:,i] = (X_tr[:,i] - np.mean(X_tr[:,i]))/np.std(X_tr[:,i])

def F1_score(y,y_hat):
    tp,tn,fp,fn = 0,0,0,0
    for i in range(len(y)):
        if y[i] == 1 and y_hat[i] == 1:
            tp += 1
        elif y[i] == 1 and y_hat[i] == 0:
            fn += 1
        elif y[i] == 0 and y_hat[i] == 1:
            fp += 1
        elif y[i] == 0 and y_hat[i] == 0:
            tn += 1
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    f1_score = 2*precision*recall/(precision+recall)
    return f1_score

class LogidticRegression:
    def sigmoid(self,z):
        sig = 1/(1+exp(-z))
        return sig
    def initialize(self,X):
        weights = np.zeros((shape(X)[1]+1,1))
        X = np.c_[np.ones((shape(X)[0],1)),X]
        return weights,X
    def fit(self,X,y,alpha=0.001,iter=400):
        weights,X = self.initialize(X)
        def cost(theta):
            z = dot(X,theta)
            cost0 = y.T.dot(log(self.sigmoid(z)))
            cost1 = (1-y).T.dot(log(1-self.sigmoid(z)))
            cost = -((cost1 + cost0))/len(y)
            return cost
        cost_list = np.zeros(iter,)
        for i in range(iter):
            weights = weights - alpha*dot(X.T,self.sigmoid(dot(X,weights))-np.reshape(y,
(len(y),1)))
            cost_list[i] = cost(weights)
        self.weights = weights
        return cost_list
    def predict(self,X):
        z = dot(self.initialize(X)[1],self.weights)
        lis = []
        for i in self.sigmoid(z):
            if i>0.5:
                lis.append(1)
            else:
                lis.append(0)
        return lis
standardize(X_tr)
standardize(X_te)
obj1 = LogidticRegression()
model= obj1.fit(X_tr,y_tr)
y_pred = obj1.predict(X_te)
y_train = obj1.predict(X_tr)
#Let's see the f1-score for training and testing data
print(F1_score(y_tr,y_train))
print(F1_score(y_te,y_pred))
```

BROAD-CASTING

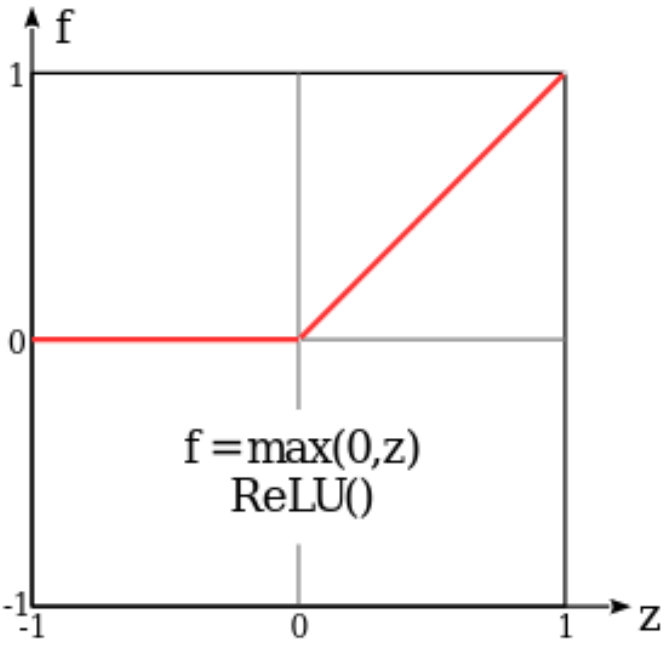
Broadcasting is the name given to the method that NumPy uses to allow array arithmetic between arrays with a different shape or size. The following snippet does broad casting.

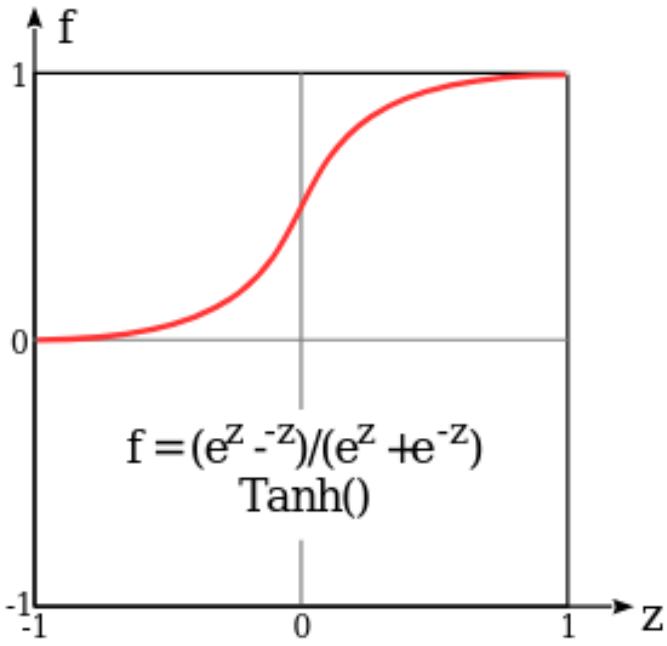
```
import numpy as np
a = np.array([17, 11, 19])
b = 3
c = a + b
print(c)
```

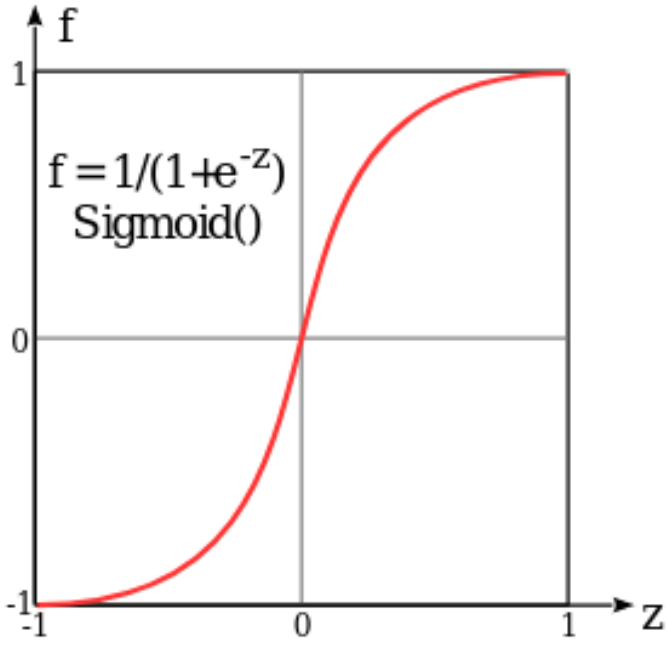
To know more about the numpy package of python, kindly refer the following link : arihara-sudhan.github.io/books.html

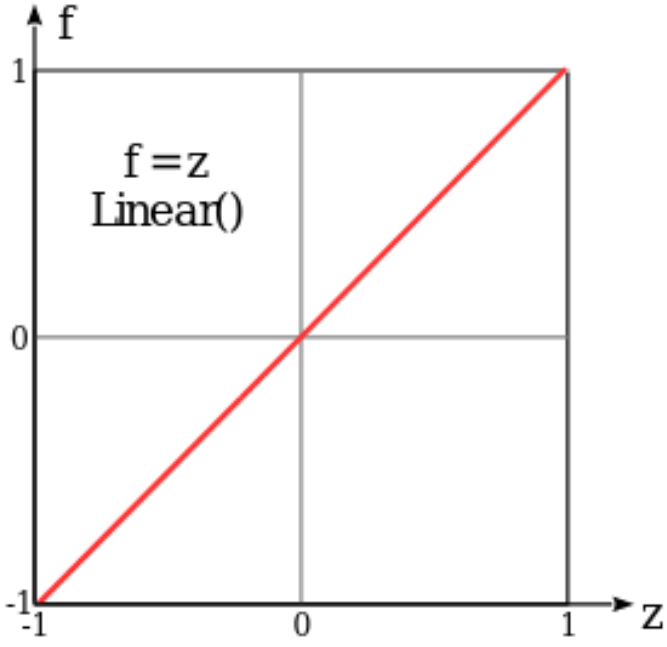
ACTIVATION FUNCTIONS

In broad terms, activation functions are necessary **to prevent linearity**. Without them, the data would pass through the nodes and layers of the network only going through linear functions (w*x+b). Refer the following link for more: arihara-sudhan.github.io/articles.html



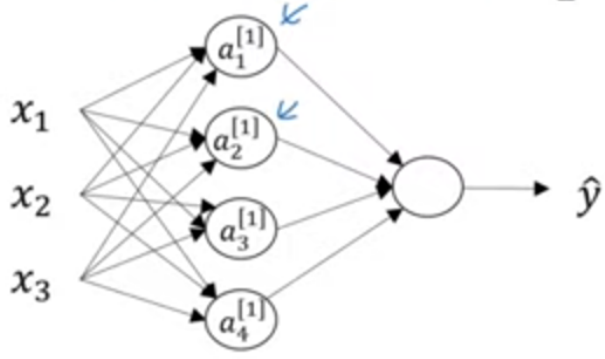






COMPUTATION – NEURAL NETWORK

Neural Network Representation



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$

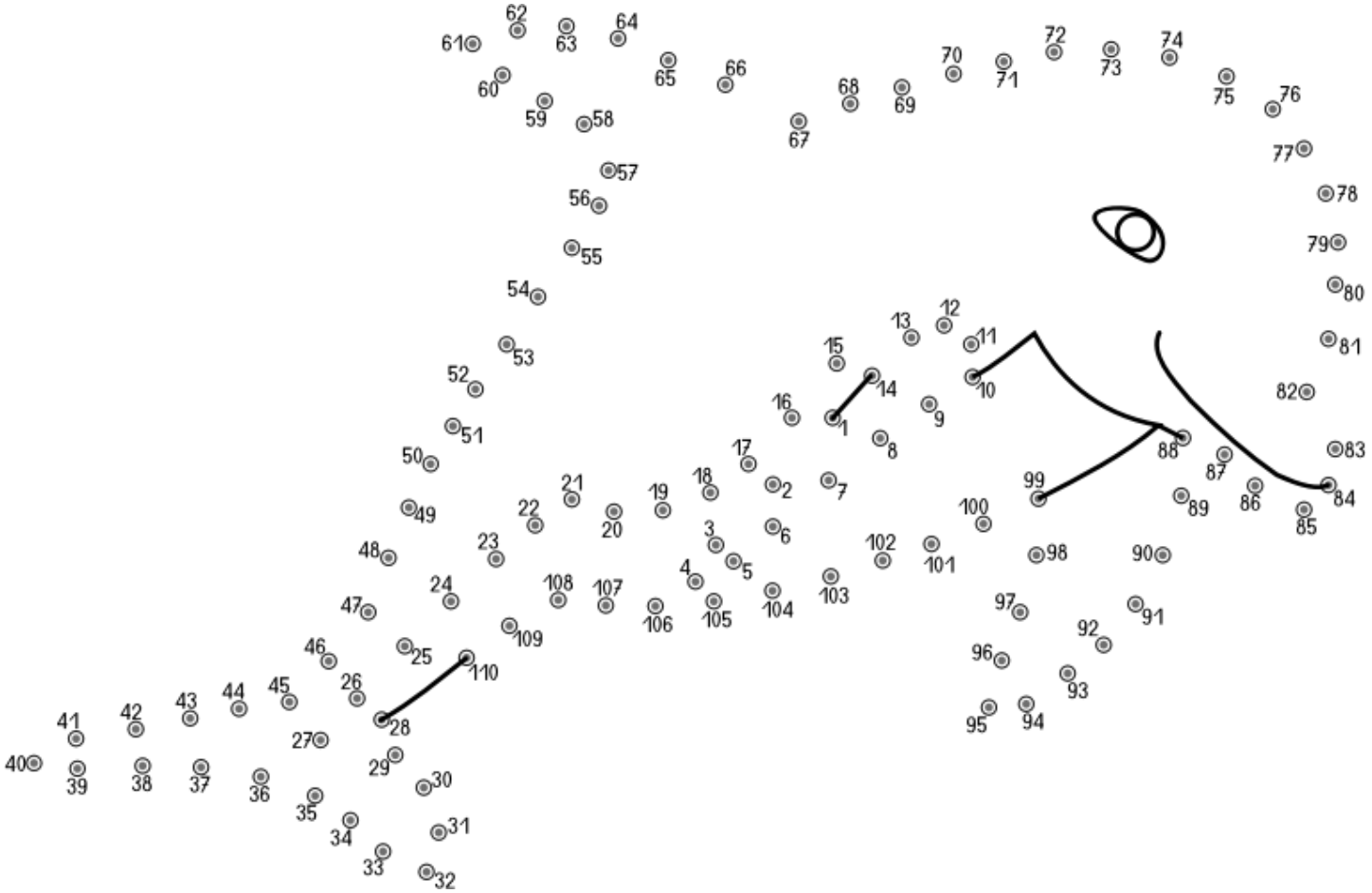
$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

RANDOM INITIALIZATION

The values for the weights should be randomized and small. If they are zero, then the activation functions compute exactly the same thing which is USELESS :-)
Besides, the values to be given for the weights should be very small too. They cause various problems like Vanishing Gradient Problem, Exploding Gradient Problem and so more specific problems. For an instance, when we use tanh activation function in our neural network, if we have given large values for the weights, it will cause the tanh to have straight lines which will drastically slow down the process of computing the gradient descent value. Same case is there in sigmoid too.

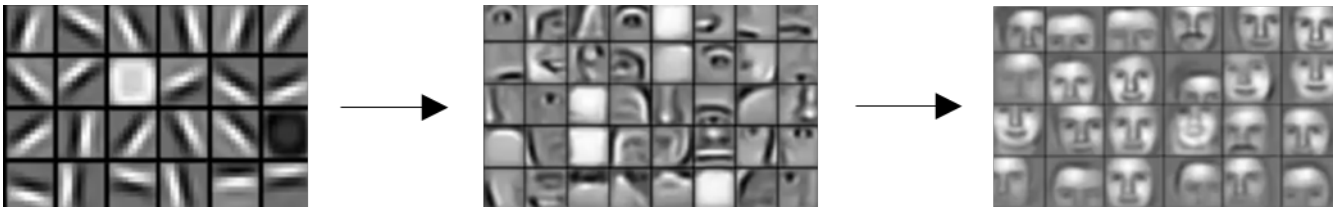
```
w = np.random.randn((2,2))*0.01
b = np.zeros((2,1))
```

DEEP DEEP DEEPER!

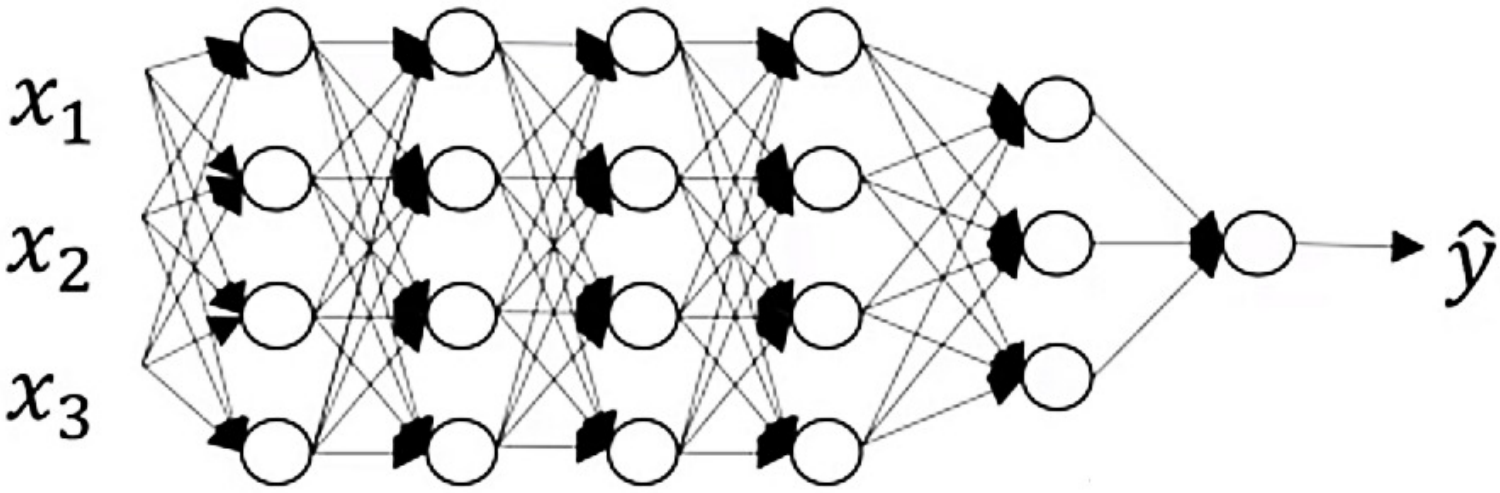


The more dots we have, the more sophisticated shape we get! The advantage of multiple layers is that they can learn features at various levels of abstraction. For example, if you train a deep convolutional neural network to classify images, you will find that the first layer will train itself to recognize very basic things like edges, the next layer will train itself to recognize collections of edges such as

shapes, the next layer will train itself to recognize collections of shapes like eyes or noses, and the next layer will learn even higher-order features like faces.



Multiple layers are much better at generalizing because they learn all the intermediate features between the raw data and the high-level classification.



5 hidden layers

THE FORWARD PASS

Propagating the computations of all neurons within all layers moving from left to right. This starts with the feeding of your feature vector(s)/tensors into the input layer, and ends with the final prediction generated by the output layer. Forward pass computations occur during training in order to evaluate the objective/loss function under the current network parameter settings in each iteration, as well as during inference (prediction after training) when applied to new/unseen data.

THE BACKWARD PASS

Known as back-propagation, or “backprop”, this is a step executed during training in order to compute the objective/loss function gradient with respect to the network’s parameters for updating them during a single iteration of some form of gradient descent. It is named as such because, when viewing a neural network as a computation graph, it starts by computing loss function derivatives at the output layer, and propagates them back towards the input layer (effectively, this is the chain rule from Calculus in action) in order to compute derivatives for, and make updates to, all parameters in all layers.

MATRIX DIMENSIONS

Remember! We deal with matrices literally. When we search it in WikiPedia (Tamizh), it clearly states this in the last line.

செயற்கை நரம்பணுப் பிணையம்

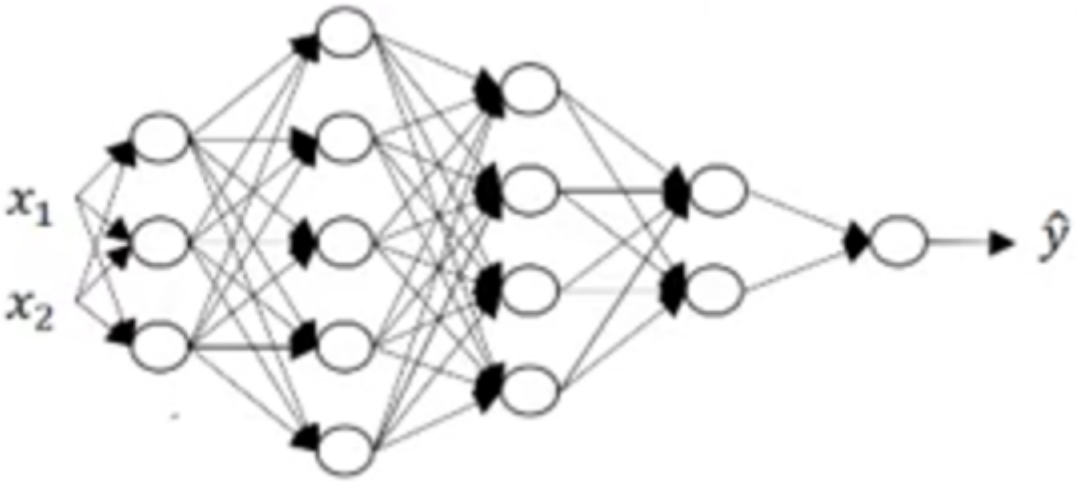
https://

கட்டுரை உரையாடல் படிக்கவும் தொகு

கட்டற்ற கலைக்களஞ்சியமான விக்கிப்பீடியாவில் இருந்து.

செயற்கை நரம்பணுப் பிணையம் (ஆங்கிலம்: Artificial neural network) என்பது உயிரி நரம்புப் பிணையத்தில் இருந்து ஊக்கம்பெறப்பெற்ற ஒரு கணிமை அல்லது கணித மாதிரி. இது பிணையப்பட்ட செயற்கை நரம்பணுக்களைக் கொண்டது. இதன் அடிப்படை அணிக் கணிதம் ஆகும்.

So, it is very very important to know the right dimensions for the matrices.



The neural network shown above has 5 layers. The input layer is not taken in account.

Each neurons compute two stuffs. First one is the weighted sum and the second one is the so-called activation. Let the weighted sum of the first neuron be $z_1^{[1]}$ and that of the second neuron be $z_1^{[2]}$. Similary, let the weight to the first neuron in the first layer be $w_1^{[1]}$ and that of the second neuron be $w_1^{[2]}$. Fix this case for everything. From the given network, we can observe that the weighted sum from the first layer is of [3 1] dimension $([z_1^{[1]}\ z_1^{[2]}\ z_1^{[3]}]^T)$. We have the two inputs which will be of [2 1] dimension $([x_1\ x_2]^T)$. The dimension for the weights vector will be of [3 2] dimension. This is the case for all the other layers appropriately.

LET’S GENERALIZE THE DIMENSIONS

Let N1, N2, N3, N4 and N5 be the number of neurons in the respective layers.

$$Z = W.X + B$$

If we consider the layer 2 which has N2 number of neurons,It’s output will be,

$$Z_{N2x1} = W_{N2xN1} . X_{N1x1} + B_{N2x1}$$

[The dimensions are in blue]

```
def init_weights():
    Wh = np.random.randn(INPUT_LAYER_SIZE,
HIDDEN_LAYER_SIZE) * \
                np.sqrt(2.0/INPUT_LAYER_SIZE)
    Wo = np.random.randn(HIDDEN_LAYER_SIZE,
OUTPUT_LAYER_SIZE) * \
                np.sqrt(2.0/HIDDEN_LAYER_SIZE)

def init_bias():
    Bh = np.full((1, HIDDEN_LAYER_SIZE), 0.1)
    Bo = np.full((1, OUTPUT_LAYER_SIZE), 0.1)
    return Bh, Bo
```

THE HYPERPARAMETERS

Hyperparameters are parameters whose values control the learning process and determine the values of model parameters that a learning algorithm ends up learning. The prefix ‘hyper_’ suggests that they are ‘top-level’ parameters that control the learning process and the model parameters that result from it. Some examples are,

- Choice of optimization algorithm (e.g., gradient descent, stochastic gradient descent, or Adam optimizer)
- Learning Rate
- Choice of activation function in a neural network (nn) layer (e.g. Sigmoid, ReLU, Tanh)
- The choice of cost or loss function the model will use
- Number of hidden layers in a nn
- Number of activation units in each layer
- The drop-out rate in nn (dropout probability)
- Number of iterations (epochs) in training a nn
- Kernel or filter size in convolutional layers
- Pooling size
- Batch size

TRAIN – DEV – TEST SET

Training set is what is used for training. The goal here is accurately predicting the unseen data on the basis of the features learned from the training data. The validation dataset is a set of data separated from the training dataset. This is for purposes like overcoming the problems of overfitting.Test data is used for testing the data for generalization.While building a Deep Learning model, it is a common practice to classify labelled data into three sets – namely training, dev and test sets. We train the algorithm using training set, validate using dev set, analyse errors and repeat until error on training and dev sets reduce.

We pick the best performing model from the before approach and later use this model to find accuracy on test set.

THE BIAS

The difference between the average value predicted by our Machine Learning model and the correct target value is known as **Bias**. A model that makes incorrect predictions about a dataset is called a **biased model**. This model oversimplifies the target function to make it easier to learn.

Underfitting: A model with High Bias tends to underfit the data as it oversimplifies the solution by failing to learn how to train the data efficiently. This results in a linear function.

Oversimplification: Due to the model being too simple, the biased model is unable to learn complex features of a training data, thus, making it inefficient when solving complex problems.

Low Training Accuracy: Due to the inability to correctly process training data, the biased model shows high-training loss resulting in low-training accuracy

THE VARIANCE

The amount of variability in the target function in response to a change in the training data is known as **Variance**. When a model takes into consideration the noise and fluctuation in the data, it is said to be of High Variance.

Overfitting: A model with *High Variance* tends to overfit the data as it overcomplicates the solution and fails to generalize new test data. This results in a *non-linear function*.

Overcomplication: Due to the model being too complex, the model learns a much more complex curve and fails to work efficiently on simple problems.

Low Testing Accuracy: Model will show a huge test data loss.

THE REGULARIZATION

One way to reduce the risk of overfitting or the variance is the so-called Regularization and another way is using data of large extent. So, how regularization will help us ?

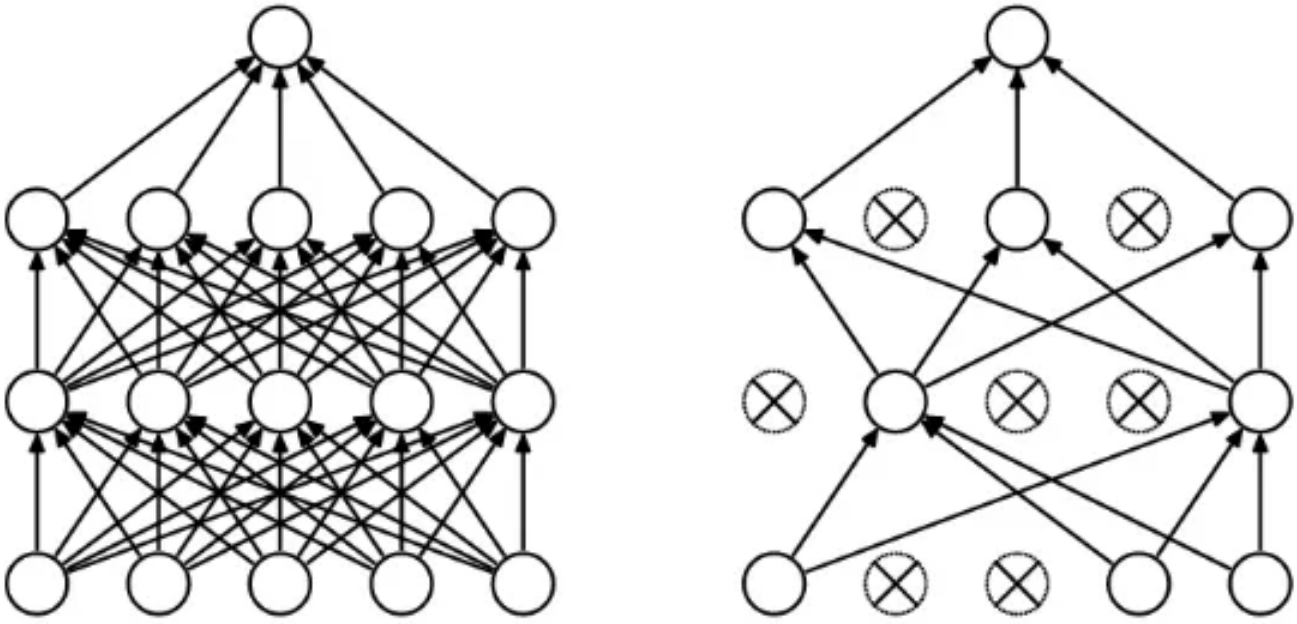
L2: $\frac{\lambda}{2}\|\mathbf{w}\|^2 = \frac{\lambda}{2}\sum_{j=1}^{m} w_j^2$

$$J(\mathbf{w}) = \sum_{i=1}^{n}\left[-y^{(i)}\log\left(\phi(z^{(i)})\right) - \left(1-y^{(i)}\right)\log\left(1-\phi(z^{(i)})\right)\right] + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

The above given solution is called the L2 Regularization. There is another sort of regularization namely L1 Regularization which sparses the weight vector. (That’s not widely used even for the compression process) L2 is prescribed.

DROP-OUT REGULARIZATION

The term “dropout” refers to dropping out the nodes (input and hidden layer) in a neural network (as seen in Figure 1). All the forward and backwards connections with a dropped node are temporarily removed, thus creating a new network architecture out of the parent network. The nodes are dropped by a dropout probability of p.



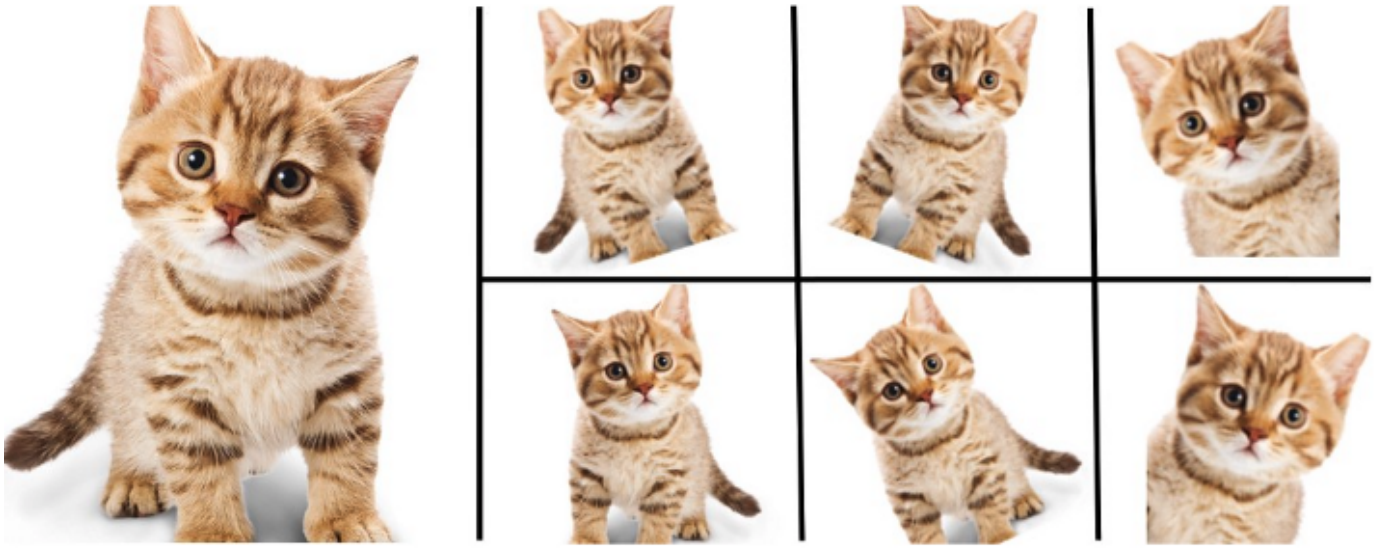
(a) Standard Neural Net (b) After applying dropout.

It reduces the so-called CoAdaption.

```
# Define PyTorch model, with dropout at input
class SonarModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.dropout = nn.Dropout(0.2)
        self.layer1 = nn.Linear(60, 60)
        self.act1 = nn.ReLU()
        self.layer2 = nn.Linear(60, 30)
        self.act2 = nn.ReLU()
        self.output = nn.Linear(30, 1)
        self.sigmoid = nn.Sigmoid()
```

DATA AUGMENTATION REGULARIZATION

Data augmentation is **a technique of artificially increasing the training set by creating modified copies of a dataset using existing data**. It includes making minor changes to the dataset or using deep learning to generate new data points. It is used in CNN for increasing the amount of image data. It helps well in overfitting.

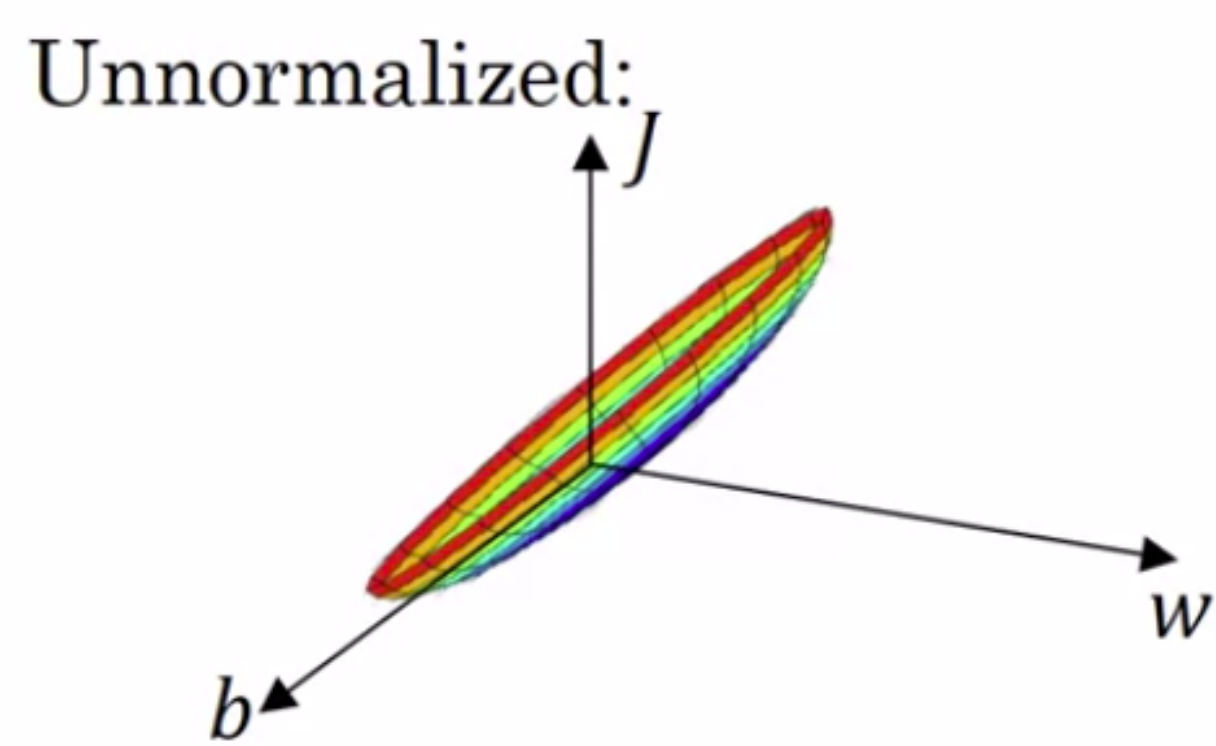


EARLY STOPPING REGULARIZATION

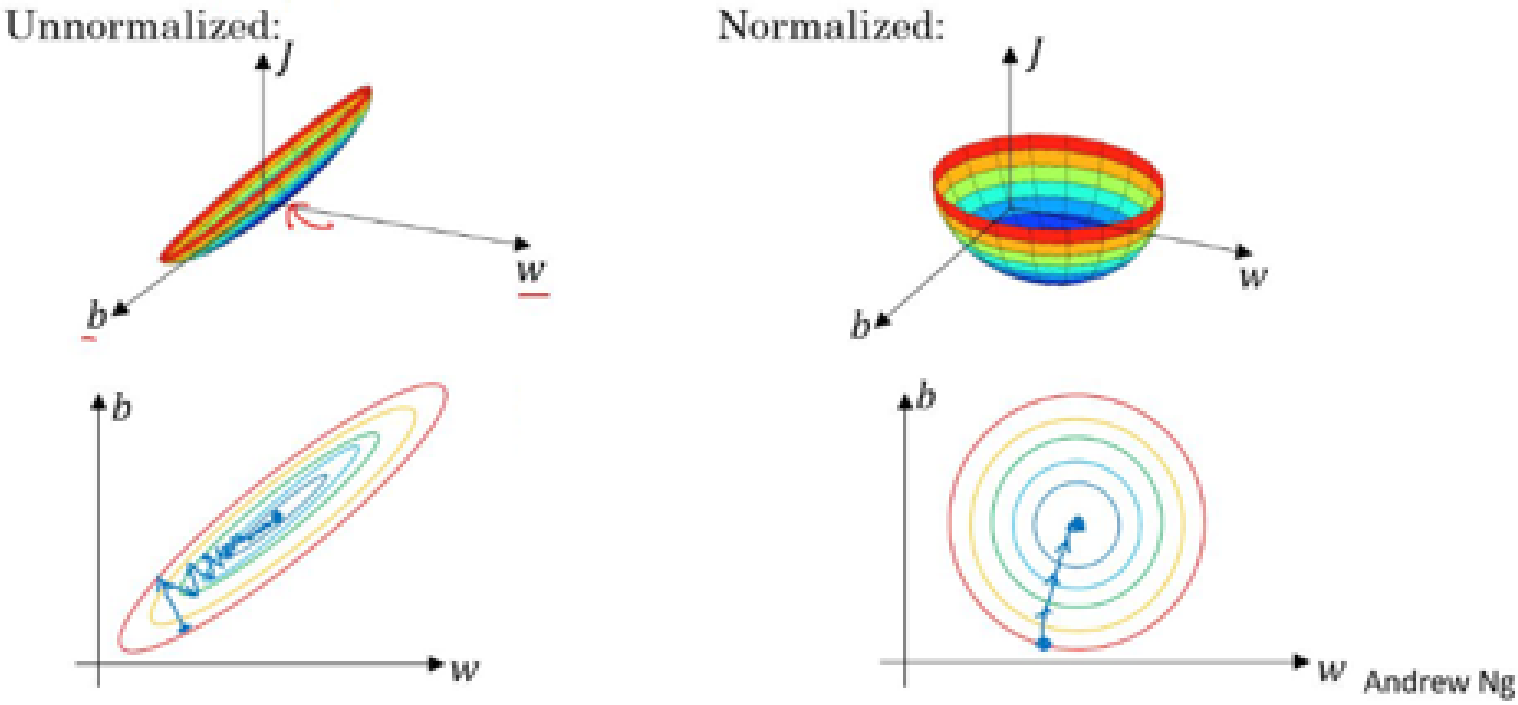
The model tries to chase the loss function crazily on the training data, by tuning the parameters. Now, we keep another set of data as the validation set and as we go on training, we keep a record of the loss function on the validation data, and when we see that there is no improvement on the validation set, we stop, rather than going all the epochs.

THE NORMALIZATION

What if we haven’t normalized the input values ? The cost function will look like the following.



As we observe from below,

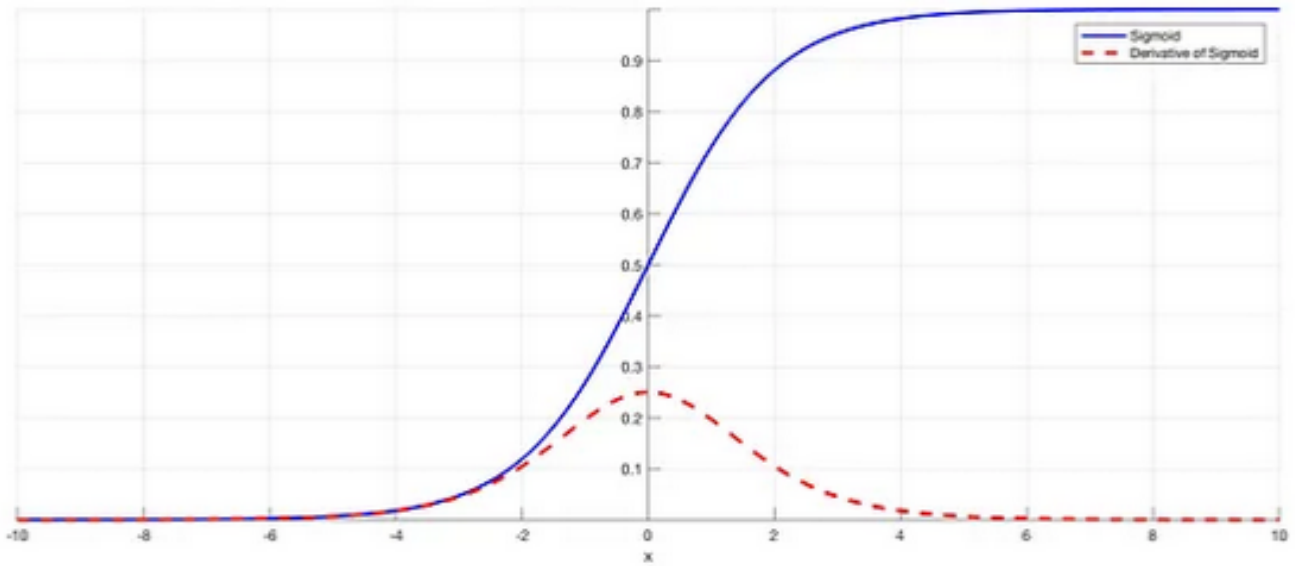


It will be quite tedious to find the local minima as it follows a zig-zag pattern. Meanwhile, the normalized one won’t allow such noises to take place in our model. When we have to normalize then ? If the given data are not even close, go for normalization. For an instance, if the feature one is 1 to 100 and the feature two is 1 to 1000000000000000, we will get the crushed cost function as shown above. So, to normalize into a considerable extent is good and good indeed.

THE VANISHING GRADIENT PROBLEM

As more layers using certain activation functions are added to neural networks, the gradients approaches zero, making the network hard to train.

It is mainly caused because of certain activation functions such as the sigmoid which squishes the input in between 0 and 1. A large change in the input of the sigmoid function will result a small change in the output.



As the input value increases, the differentiation approaches ZERO causing the vanishing gradient problem. For shallow network with only a few layers that use these activations, this isn’t a big problem. However, when more layers are used, it can cause the

gradient to be too small for training to work effectively. Gradients of neural networks are found using backpropagation. Simply put, backpropagation finds the derivatives of the network by moving layer by layer from the final layer to the initial one. By the chain rule, the derivatives of each layer are multiplied down the network to

compute the derivatives of the initial layers.

THE EXPLODING GRADIENT PROBLEM

In deep networks or recurrent neural networks, error gradients can accumulate during an update and result in very large gradients. These in turn result in large updates to the network weights, and in turn, an unstable network. At an extreme, the values of weights can become so large as to overflow and result in NaN values. The explosion occurs through exponential growth by repeatedly multiplying gradients through the network

layers that have values larger than 1.0.

BATCH GRADIENT DESCENT

In Batch Gradient Descent, all the training data is taken into consideration to take a single step. Then, the average of these gradients will be taken to update the parameters.

STOCHASTIC GRADIENT DESCENT

Suppose our dataset has 5 million examples, then just to take one step the model will have to calculate the gradients of all the 5 million examples. This does not seem an efficient way. To tackle this problem we have Stochastic Gradient Descent. In Stochastic Gradient Descent (SGD), we consider just one example at a time to take a

single step. We do the following steps in **one epoch** for SGD:
(1) Take an example
(2) Feed it to Neural Network
(3) Calculate it’s gradient
(4) Use the gradient calculated in step 3 to update the weights
(5) Repeat steps 1–4 for all the examples

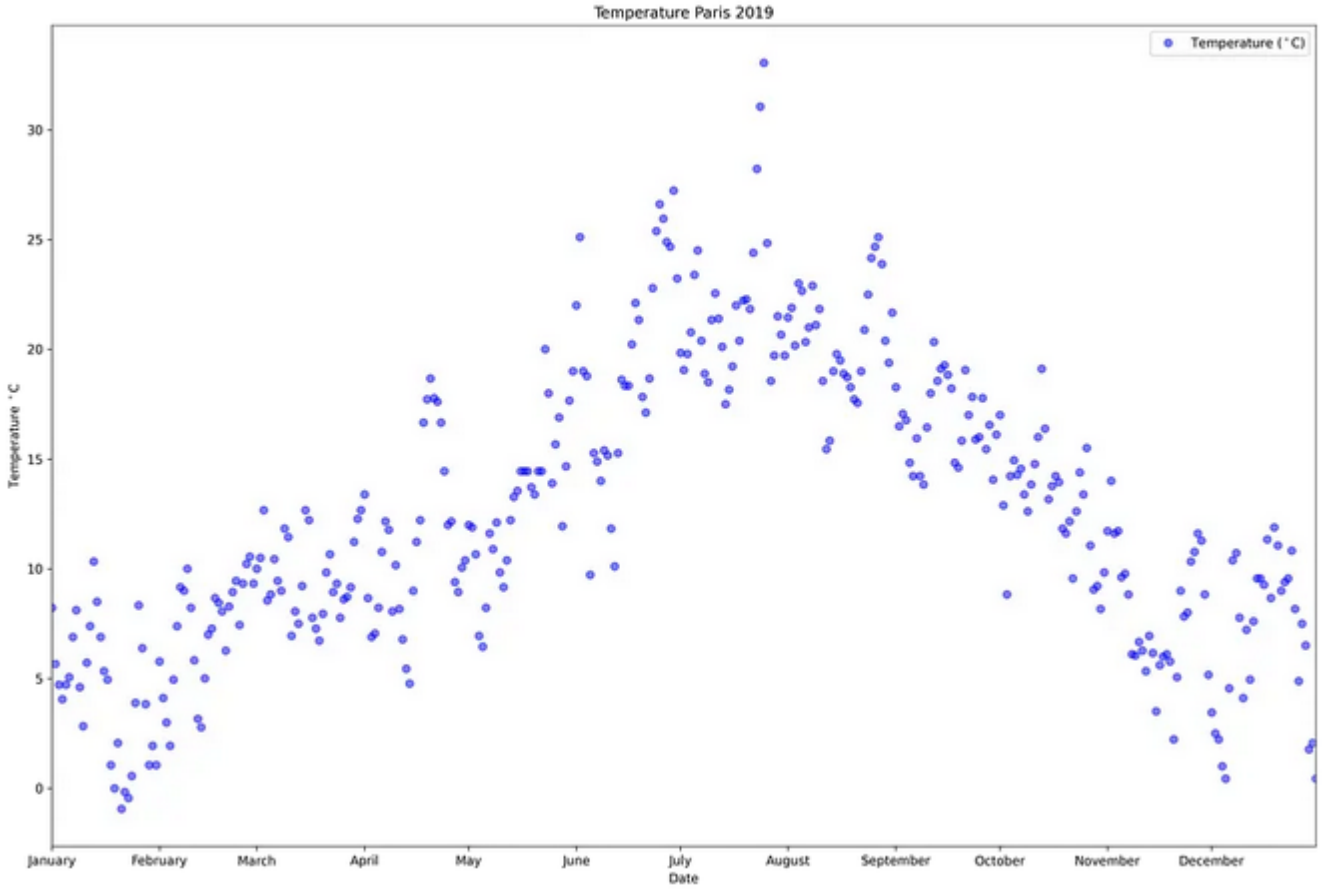
MINI-BATCH GRADIENT DESCENT

Batch Gradient Descent converges directly to minima. SGD converges faster for larger datasets. But, since in SGD we use only one example at a time, we cannot implement the vectorized implementation on it. This can slow down the computations. To tackle this problem, a mixture of Batch Gradient Descent and SGD is used. Neither we use all the

dataset all at once nor we use the single example at a time.
We use a batch of a fixed number of training examples which is less than the actual dataset and call it a mini-batch. Doing this helps us achieve the advantages of both the former variants we saw. So, after creating the mini-batches of fixed size, we do the following steps in **one epoch:**
(1) Pick a mini-batch
(2) Feed it to Neural Network
(3) Calculate the mean gradient of the mini-batch
(4) Use the mean gradient we calculated in step 3 to update the weights
(5) Repeat steps 1–4 for the mini-batches we created

THE EWMA

The Exponentially Weighted Moving Average (EWMA) is commonly used as a smoothing technique in time series. However, due to several computational advantages (fast, low-memory cost), the EWMA is behind the scenes of many optimization algorithms in deep learning, including Gradient Descent with Momentum, RMSprop, Adam, etc.



To find the exponentially weighted sum values,

$$W_1 = 0.9 \cdot W_0 + 0.1 \cdot \theta_1$$
$$W_2 = 0.9 \cdot W_1 + 0.1 \cdot \theta_2$$
$$\vdots$$
$$W_t = 0.9 \cdot W_{t-1} + 0.1 \cdot \theta_t$$

The pattern given above is used. On generalizing the pattern, the formula for finding this is,

$$W_t = \begin{cases} 0 & t = 0 \\ \beta \cdot W_{t-1} + (1-\beta) \cdot \theta_t & t > 0 \end{cases}$$

The terms used are,

$$W_t = \underbrace{\beta \cdot W_{t-1}}_{\text{trend}} + \overbrace{(1-\beta) \cdot \theta_t}^{\text{current value}}$$

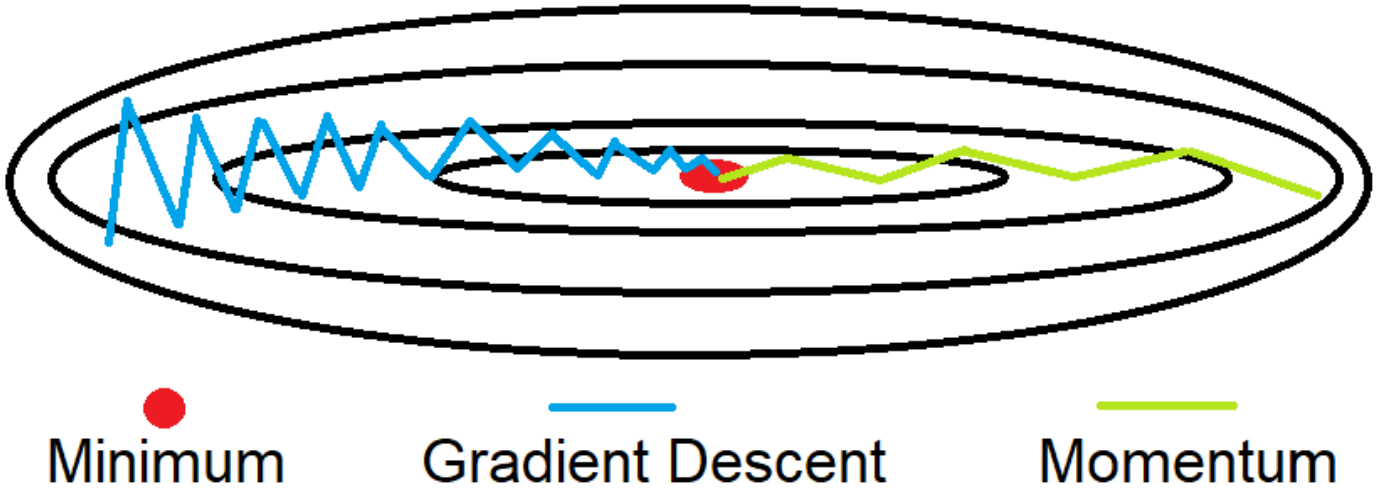
We can see that the value of β determines how important the previous value is (the trend), and (1-β) how important the current value is.

$$W_t = (1-\beta)(\theta_t + \beta \cdot \theta_{t-1} + \beta^2 \cdot \theta_{t-2} + \cdots + \beta^{t-3} \cdot \theta_3 + +\beta^{t-2} \cdot \theta_2 + +\beta^{t-1} \cdot \theta_1)$$

Which can be written as a dot product as shown below :

$$W_t = (1-\beta) \cdot (\beta^0, \beta^1, \ldots, \beta^{t-2}, \beta^{t-1}) * (\theta_t, \theta_{t-1}, \ldots, \theta_2, \theta_1)$$

GRADIENT DESCENT WITH MOMENTUM



Rather than taking the conventional gradient descent values and propagating it, a minor change is made.

We use dW and db to update our parameters W and b during the backward propagation as follows:

W = W — learning rate * dW

b = b — learning rate * db

In momentum we take the exponentially weighted averages of dW and db, instead of using dW and db independently for each epoch.

VdW = β*VdW + (1 — β)*dW

Vdb = β*Vdb + (1 — β)*db

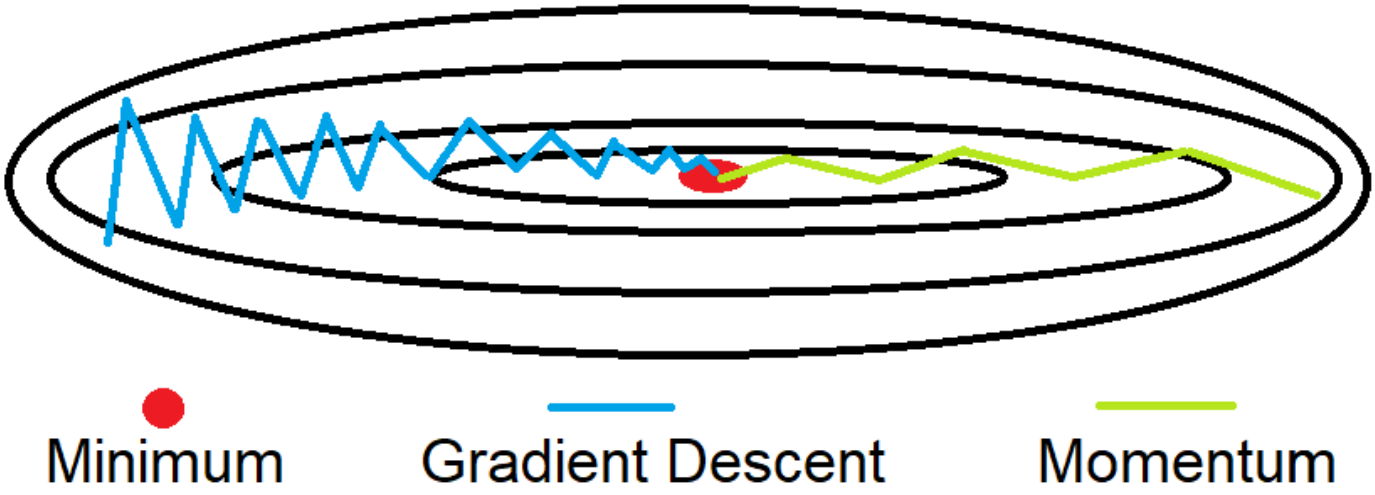
W = W — learning rate * VdW

b = b — learning rate * Vdb

Where beta ‘β’ is a different hyperparameter called momentum, ranging from 0 to 1. To calculate the new weighted average, it sets the weight between the average of previous values and the current value.

RMSProp Optimizer

It means Root Mean Square Prop.



$$SdW = \beta * SdW + (1 - \beta) * dW^2$$

$$Sdb = \beta * Sdb + (1 - \beta) * db^2$$

$$W = W - \text{learning rate} * (dW / \sqrt{SdW})$$

$$b = b - \text{learning rate} * (db / \sqrt{Sdb})$$

ADAM Optimizer

It is a combination of both Gradient Descent with Momentum and RMSProp Optimizer. ADAM stands for Adaptive Momentum Estimation.

$$VdW = \beta_1 * VdW + (1 - \beta_1) * dW$$

$$Vdb = \beta_1 * Vdb + (1 - \beta_1) * db$$

$$SdW = \beta_2 * SdW + (1 - \beta_2) * dW^2$$
$$Sdb = \beta_2 * Sdb + (1 - \beta_2) * db^2$$

$$VdW^{CORRECTED} = VdW/(1 - \beta_1)t$$

$$Vdb^{CORRECTED} = Vdb/(1 - \beta_1)t$$

$$SdW^{CORRECTED} = SdW/(1 - \beta_2)t$$

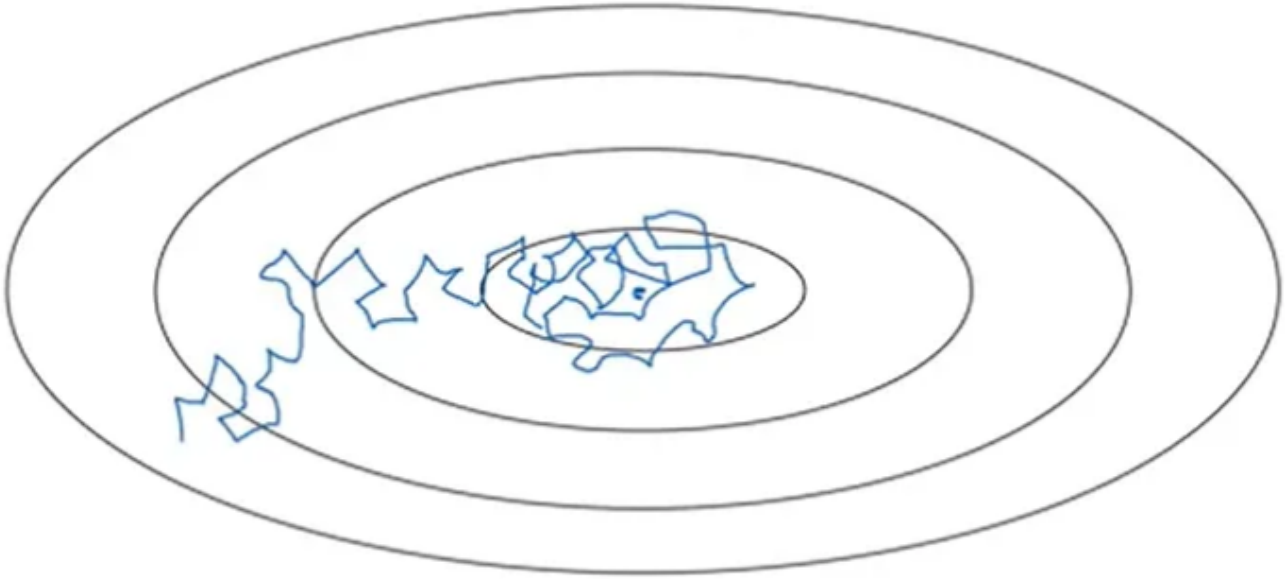
$$Sdb^{CORRECTED} = Sdb/(1 - \beta_2)t$$

$$W = W - \text{learning rate} * (VdW^{CORRECTED}/(\sqrt{SdW^{CORRECTED}}+\varepsilon))$$
$$b = b - \text{learning rate} * (Vdb^{CORRECTED}/(\sqrt{Sdb^{CORRECTED}}+\varepsilon))$$

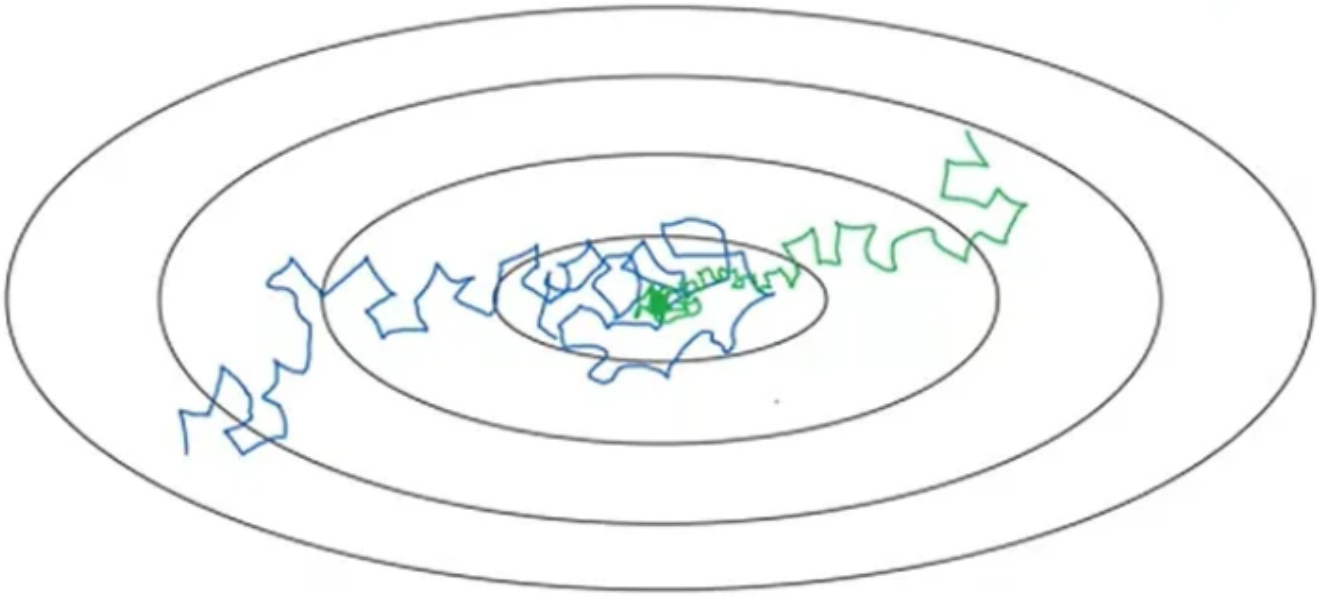
LEARNING RATE DECAY

While training neural networks with Stochastic or Mini Batch Gradient Descent and a constant learning rate our algorithm usually converges towards minima in a noisy manner (less noisier in MBGD) and end up oscillating far away from actual minima. To overcome this scenario , decaying the learning rate over time would helps the network converge smoothly without noises.

With a constant learning rate :



With learning rate decay :



A formula used for decaying the learning rate over time is,

$\alpha=(1/(1+decayRate\times epochNumber))*\alpha_0$

Suppose we have $\alpha 0 = 0.2$ and decay rate=1 , then for the each epoch we can examine the fall in learning rate α as:

Epoch 1: alpha 0.1

Epoch 2: alpha 0.067

Epoch 3: alpha 0.05

Epoch 4: alpha 0.04

Some other methods used for leaning rate decay are,

@ Exponential Decay

$\alpha = (decayRate^{(epochNumber)})*\alpha_0$

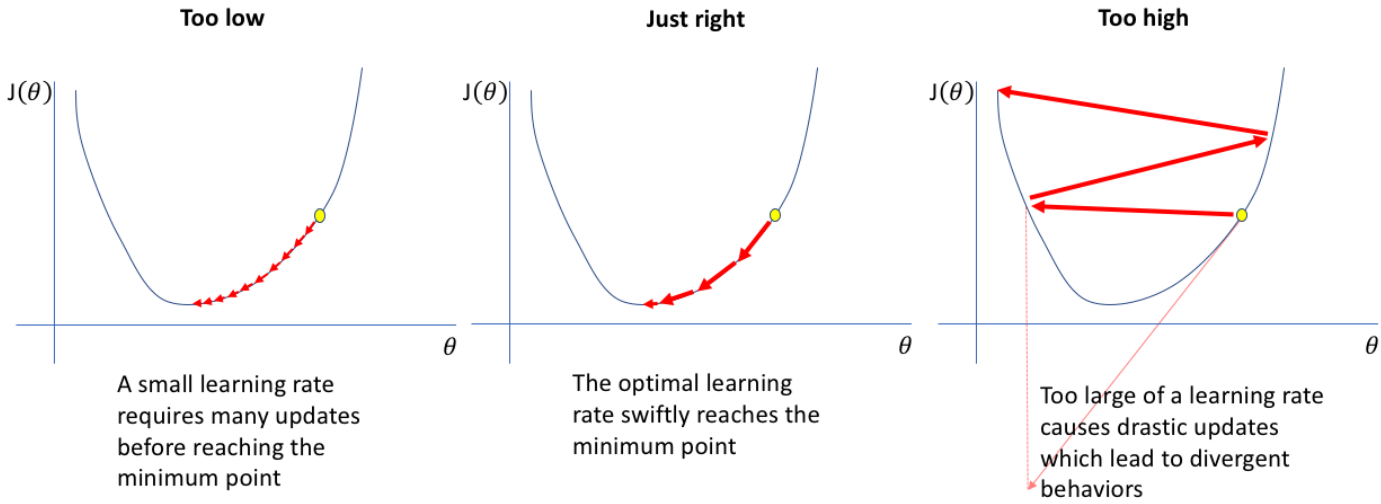
@ Discrete Staircase

In this method learning rate is decreased in some discrete steps after every certain interval of time , for example you are reducing learning rate to its half after every 10 secs.

@ Epoch Number Based

$\alpha = (k/ \sqrt{epoch\_no})* \alpha_0$

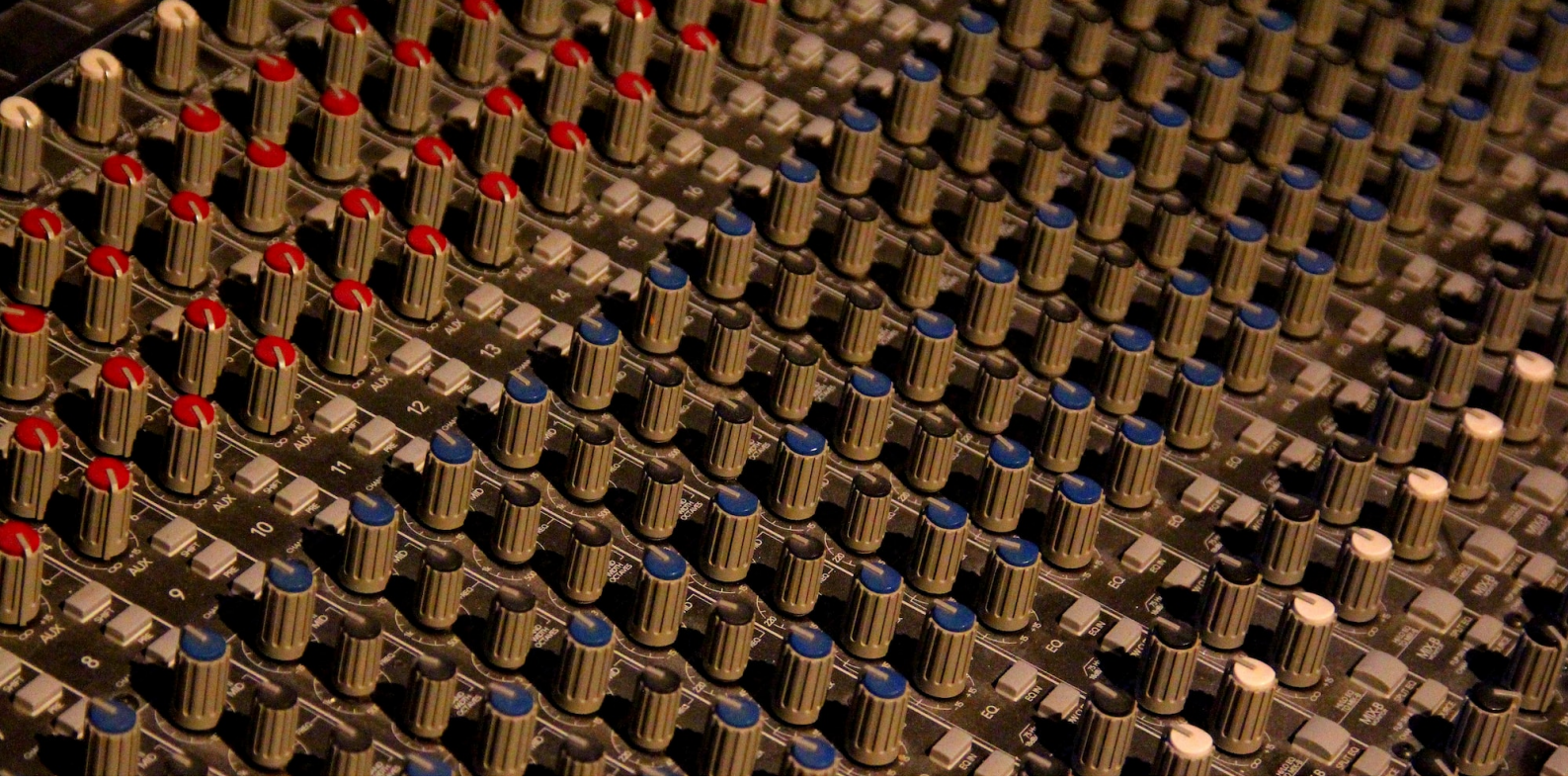
Why learning rate should be decayed gradually ?



As we observe there, it is obvious that having very low learning rate as well as very high learning rate is not a good choice. The only “Just RIGHT” choice is decaying the weight as we approach the local minima. We will be avoiding the problem of over shooting the goal as well as the slowly moving problem.

HYPER-PARAMETER TUNING

Often the general effects of hyper-parameters on a model are known, but how to best set a hyper-parameter and combinations of interacting hyper-parameters for a given dataset is challenging. There are often general heuristics or rules of thumb for configuring hyper-parameters. A better approach is to objectively search different values for model hyperparameters and choose a subset that results in a model that achieves the best performance on a given dataset. This is called **hyper-parameter optimization** or hyperparameter tuning.

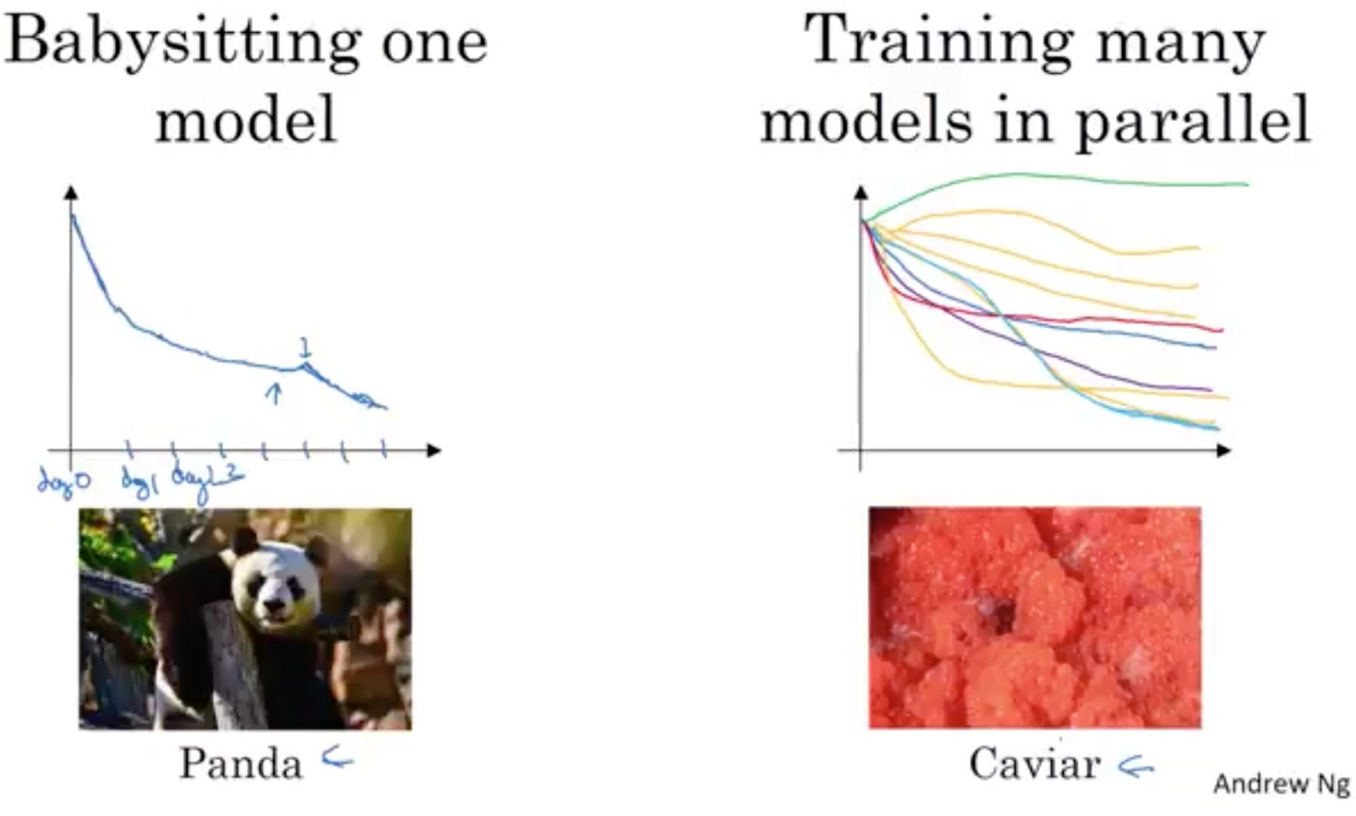


A range of different optimization algorithms may be used, although two of the simplest and most common methods are random search and grid search.

- **Random Search**. Define a search space as a bounded domain of hyperparameter values and randomly sample points in that domain.
- **Grid Search**. Define a search space as a grid of hyperparameter values and evaluate every position in the grid.

TUNING APPROACHES FOR SCENARIOS

When we have low computational power but a huge amount of data, we have to go for the so-called Babysitting process. It is an approach of tweaking a hyperparameter over time and watching the performance (patiently). Meanwhile, when we have a sufficient computational power, we can find out a good hyperparameter which does well using the parallel model monitoring.



BATCH NORMALIZATION

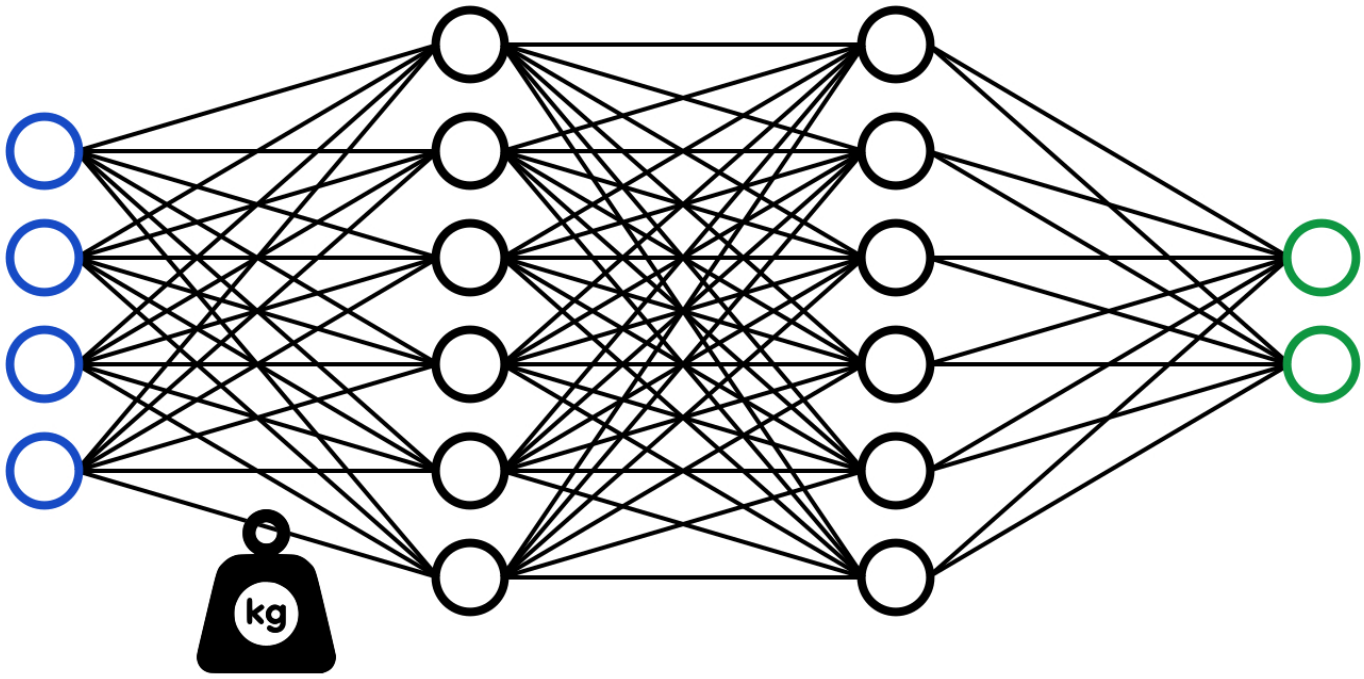
Both of the following have the same goal of put all the data-points on the same scale.

@NORMALIZATION:

A Normalization process converts the data ranging from 0 to 100 into 0 to 1. (DONE BEFORE FEEDING IN)

@STANDARDIZATION:

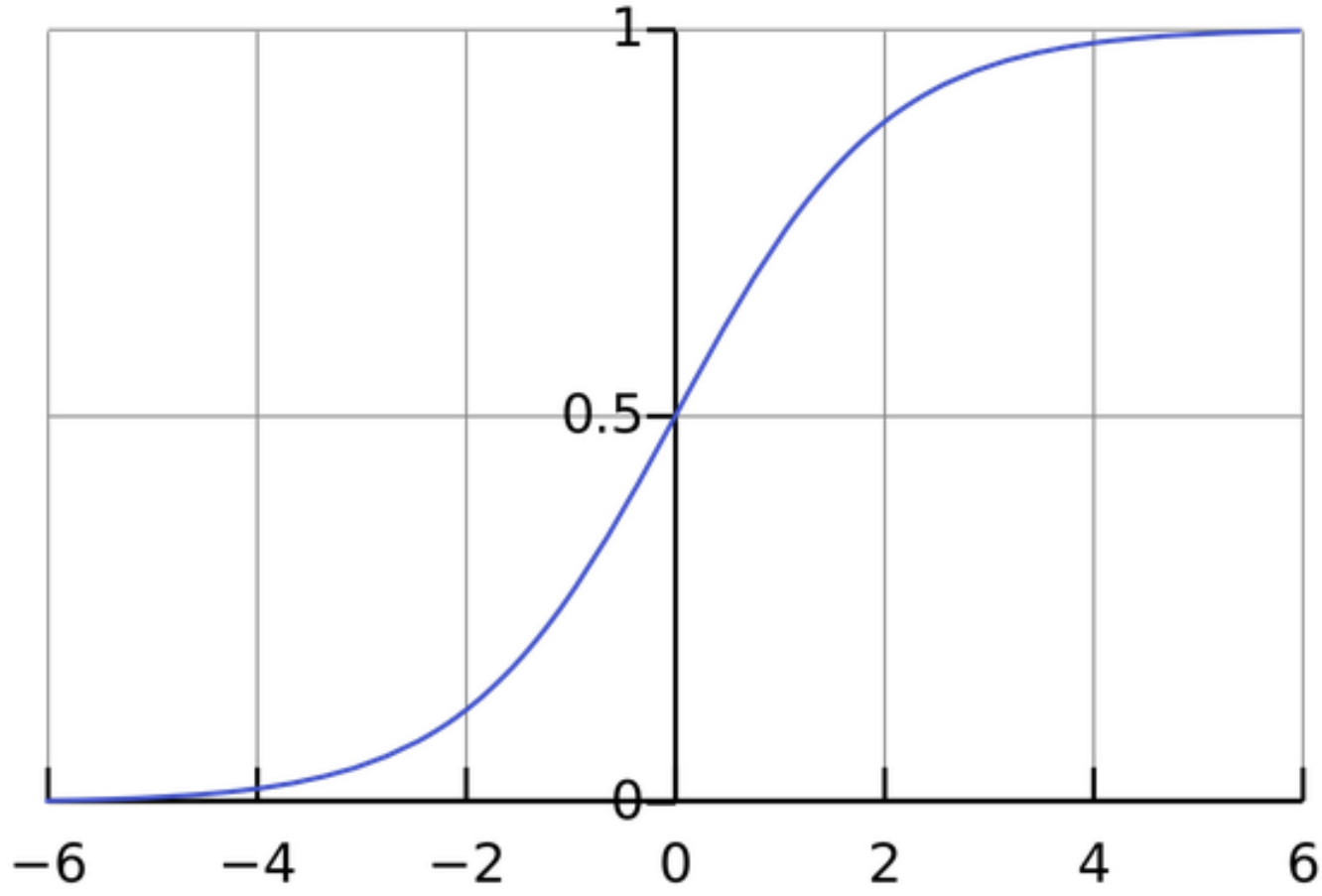
It consists of the processes of subtracting the mean from the data-point and dividing by the standard deviation.



When is Batch Normalization? Assume we have a quite-changed value of weight in a particular layer. i.e., Let’s say we have a weight in the layer1 which is greater than others. What will happen ? This imbalance will cause some noises to the layers next to it. Here, we have to apply the Batch Normalization. The output of the activation function is batch-normalized. That output is multiplied and added with an arbitrarily selected values.

SOFTMAX CLASSIFIER

If we want to categorize an object which is from a collection of objects of different classes, we need more than a binary classification. Here we go with the so-called SoftMax Classification. It yields the probability distribution values for the different objects representing the possibility of belonging to a particular class.



The softmax function shown above has the following form.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Softmax is used for multi-classification problems, prior to applying softmax, some vector components could be negative, or greater than one, and they might not sum to 1, softmax layer outputs a probability distribution then the values of the output sum to 1. This additional constraint helps training converge more quickly than it otherwise would. If our output vector is $[5\ 2\ -1\ 3]^T$, the softmax classifier will result $[0.8\ 0.04\ 0.00\ 0.11]^T$.

There is another one called hardmax which puts 1 for the greatest and 0 for the remaining values.

THE ORTHOGONALIZATION

Let’s consider tuning a TV. As per our assumption, TV has a knobs to rotate the image shown, to vertically stretch, to horizontally stretch and so on. With one knob, it is not a piece of cake.

CHAIN OF ASSUMPTIONS :

@ Perform well in Train Data

If not, make the network bigger; Use an apt optimizer

@ Perform well in Dev Data

If not, Do Regularization and Increase Train Set Size

@ Perform well in Test Data

If not, make the dev set bigger

@ Perform well in Real World

If not, change the dev set or the cost function

SINGLE EVALUATION METRIC

We have some evaluation metrics such as Precision and Recall. Precision is the measure of correctness in the classified outputs.
@ If 75 out of 100 animals are classified as cats and that is true, our precision here is 100%. [of examples recognized as x, what percentage is actually x]
@ If 75 out of 75 cats are classified as cats, our recall here is 100%. [what percentage of actual x are correctly classified]
@ Thirdly we have the F1 Score which can be roughly seen as the average of Precision and Recall. The F1 score combines precision and recall using their harmonic mean.

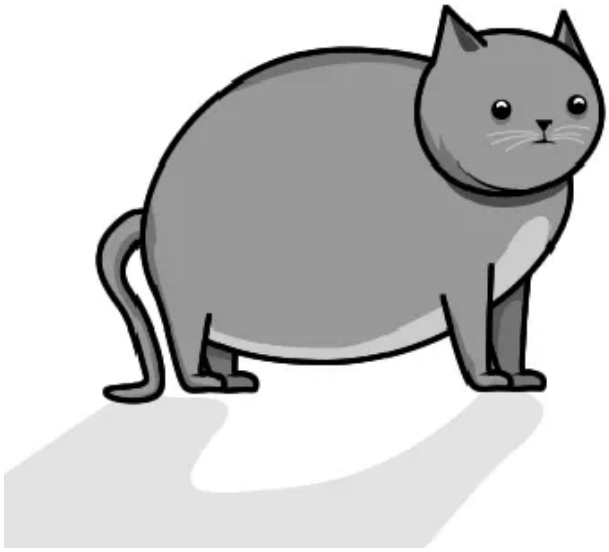
Maximizing the F1 score implies simultaneously maximizing both precision and recall. Thus, the F1 score has become the choice of researchers for evaluating their models in conjunction with accuracy.

TRANSFER LEARNING

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task. It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems.

OPTIMIZING & SATISFICING METRICS

Let’s say you’ve decided that you care about the classification accuracy of your cat’s classifier, this could have been an F1 score or some other measure of accuracy, but let’s say you also care about running time in addition to accuracy.



You want to select a classifier that maximizes accuracy, but subject to time, that must be less than 100 milliseconds or equal to it.

So in this case, we would say that accuracy is a metric that optimizes, because you want to maximize accuracy. In terms of accuracy you want to do as well as possible so that run time is what we call a satisfying metric.

Classifier	Accuracy	Running time
A	90%	80 ms
B	92%	95 ms
C	95%	1 500 ms

Accuracy is the optimizing metric, because you want the classifier to correctly detect a cat image as accurately as possible. The running time which is set to be under 100 ms in this example,is the satisficing metric which mean that the metric has to meet expectation set.

TRAIN/DEV/TEST DATA DISTRIBUTION

Assume we have the data of faces of people of different countries as shown below.

1. AMERICA
2. EUROPE
3. UK
4. SOUTH AMERICA
5. INDIA
6. CHINA
7. OTHER ASIA

It is a **BAD** choice to use first 4 for dev-set and the remaining for test-set since the distribution is not same. It is important to choose the dev and test sets from the same distribution and it must be taken randomly from all the data. So, both the dev and the test set must be taken randomly from the given.

A SNIPPET :

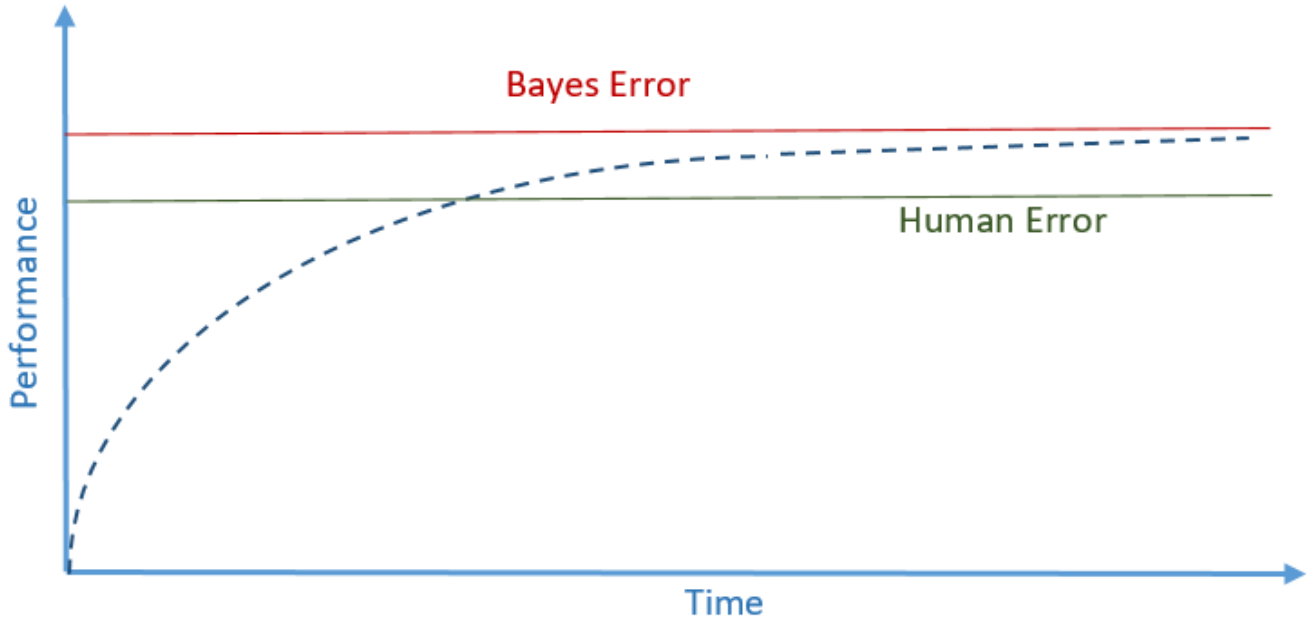
```
import random
random.shuffle(yourSet)
train,test,dev=yourSet[:70],yourSet[70:95],
yourSet[95:]
```

SIZE OF TRAIN/DEV/TEST SET

When we have a billion extent of training data, it is enough to go for a dev and test set of size 1%. But, make sure to randomize!

THE HUMAN LEVEL PERFORMANCE

Why should we compare ML Algorithms with Human Level Performance ?



The Time here can be any number of weeks or months. Here, the performance progresses towards the Human Level Performance. When the HLP is surpassed, the performance slows down. As we keep training the model over time, the performance increases but never surpasses a particular limit which is the so-called Bayesian Optimal Error.

Why does the performance slow down after surpassing the HLP? If the model is worse than human-level performance, there are tools that can be utilized to improve this performance. However, after that point, it gets hard for the model to learn because we no longer can show things that the machine learning system doesn’t already know. For the case of the system performing sub-human-level, we can get labeled data from humans. But once it gets better than us, then we’re kind of stuck.

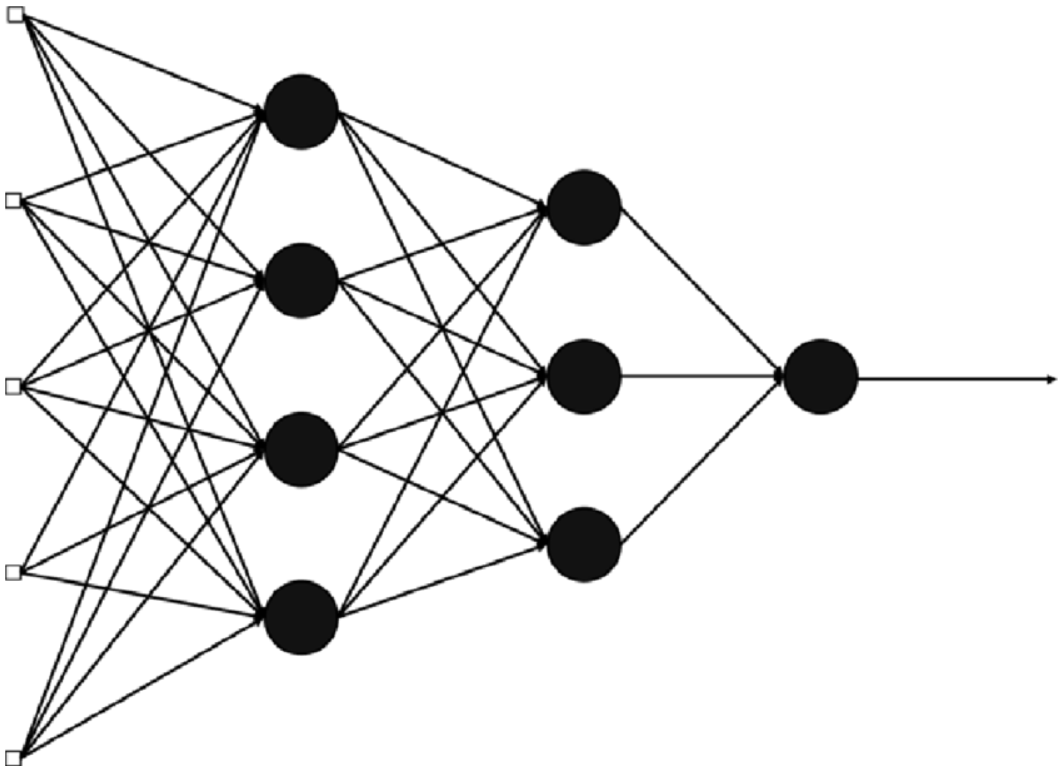


Figure 1: A Placeholder :-)

MISMATCHED TRAIN & DEV SET ERROR

Value of the error (misclassified examples) on the training set is labeled as Bias and the difference between the value of error on dev set and value of error on training set is labeled as Variance. Based on the values of Bias and Variance, we proceed with different ways. If the values of bias and Variance are convincing, we stop the iteration and go with the testing on test set. This is the usual cycle in deep learning. Some cases are, **Train-set error – 1%** and **Dev-set error – 10%** : Model is **overfitting** train set and not being able to generalise unseen examples. This is called **High Variance** and can be reduced by training again and regularization.

Train set error – 10% and **dev set error – 11%** means that our model is **under fitting** train set. This is called High Bias and can be reduced by training with a bigger network or a different neural network architecture. **Train set error – 0.5%** and **dev set error – 1%** means that our model is performing well and we can use this model to test on test set.

HLP AS A PROXY OF BAYES ERROR

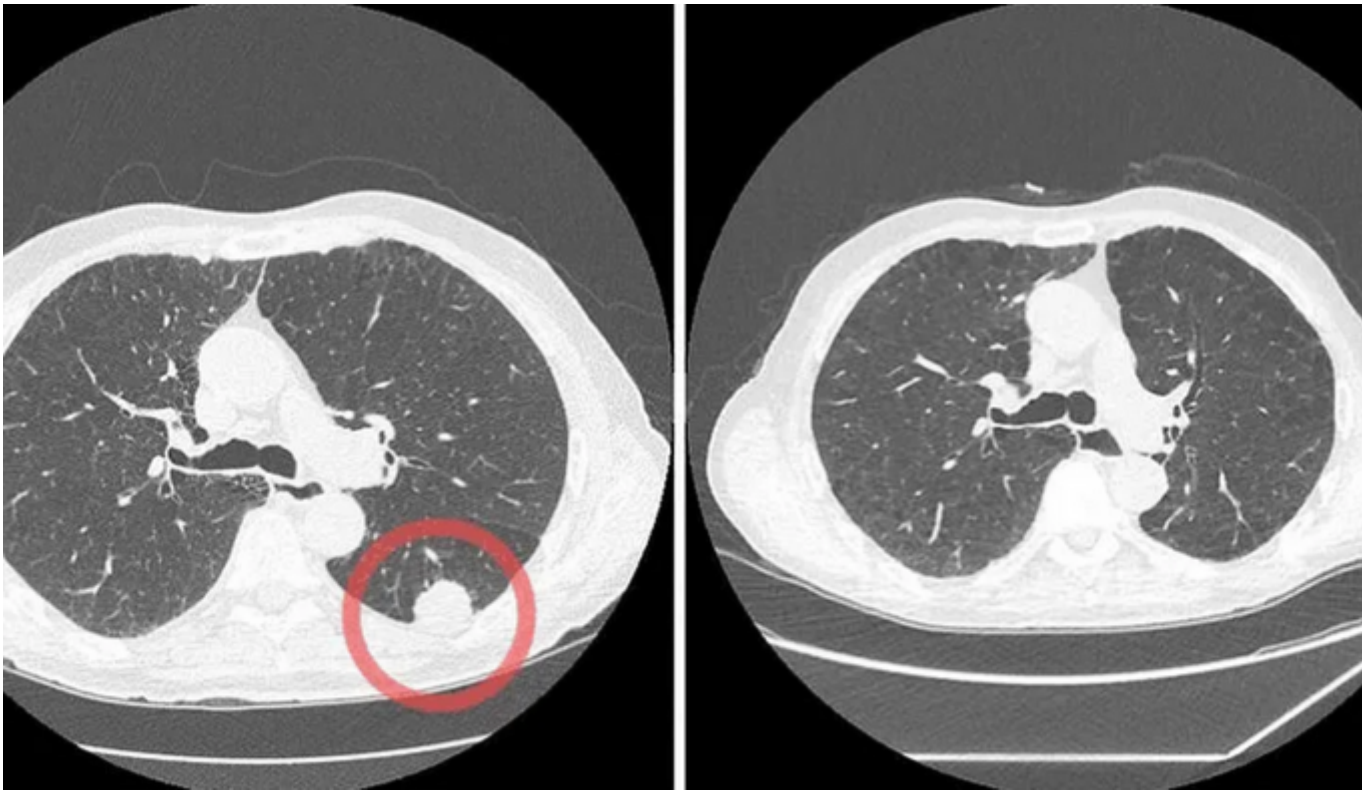


Figure 2: Images representing a lung with and without cancer.

Let’s suppose that these are the performance achieved by the groups in a hospital on diagnosis:

(a) Untrained human - 20% error

(b) Doctor - 6% error

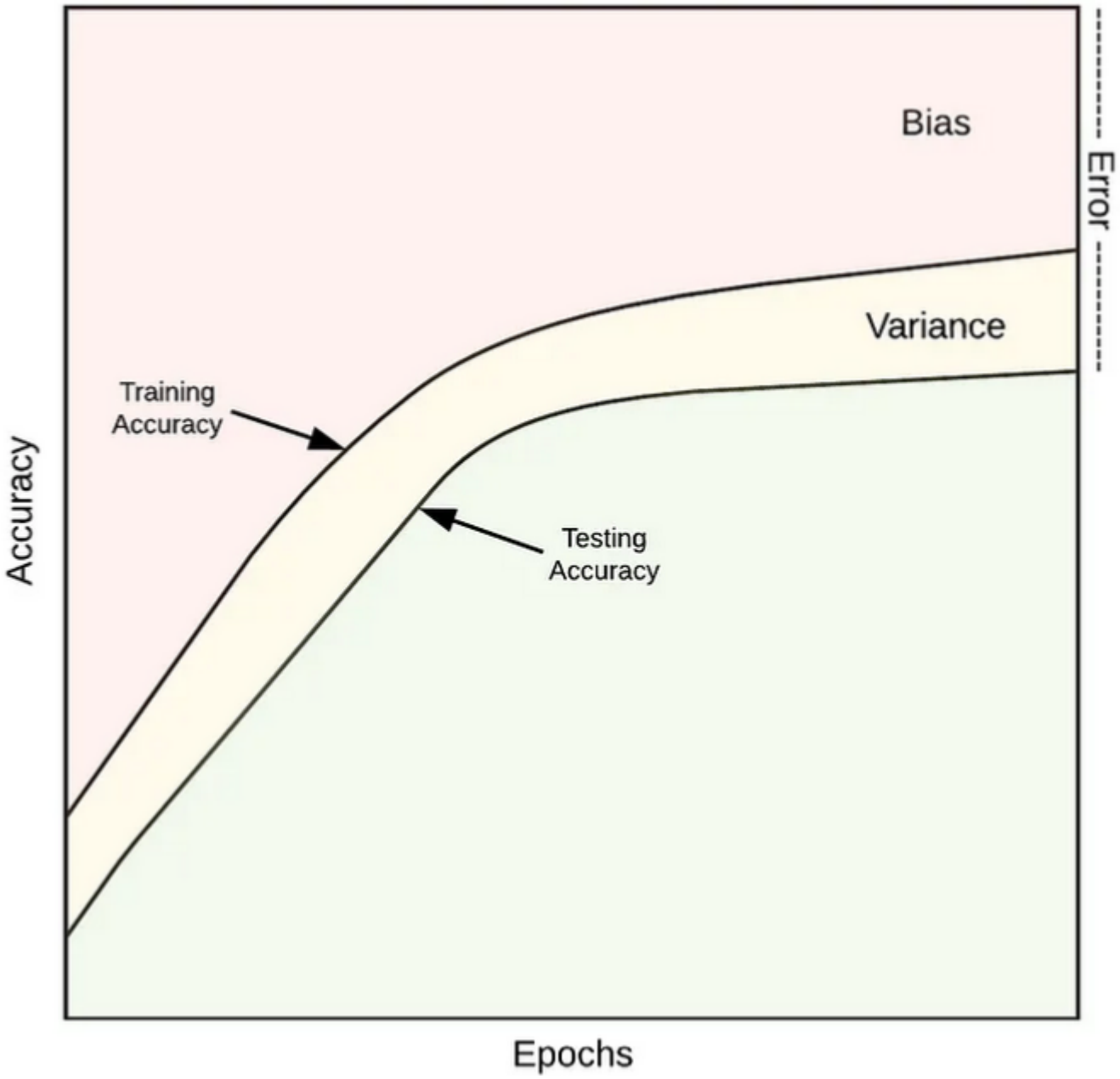
(c) Experienced doctor - 3% error

(d) Team of experienced doctors - 0.6% error.

The last one seems to be optimum and is considered as Bayes Error. Here, we can define the HLP as a proxy of Bayes Error as HLP<=0.6%.

SO WE HAVE TO...

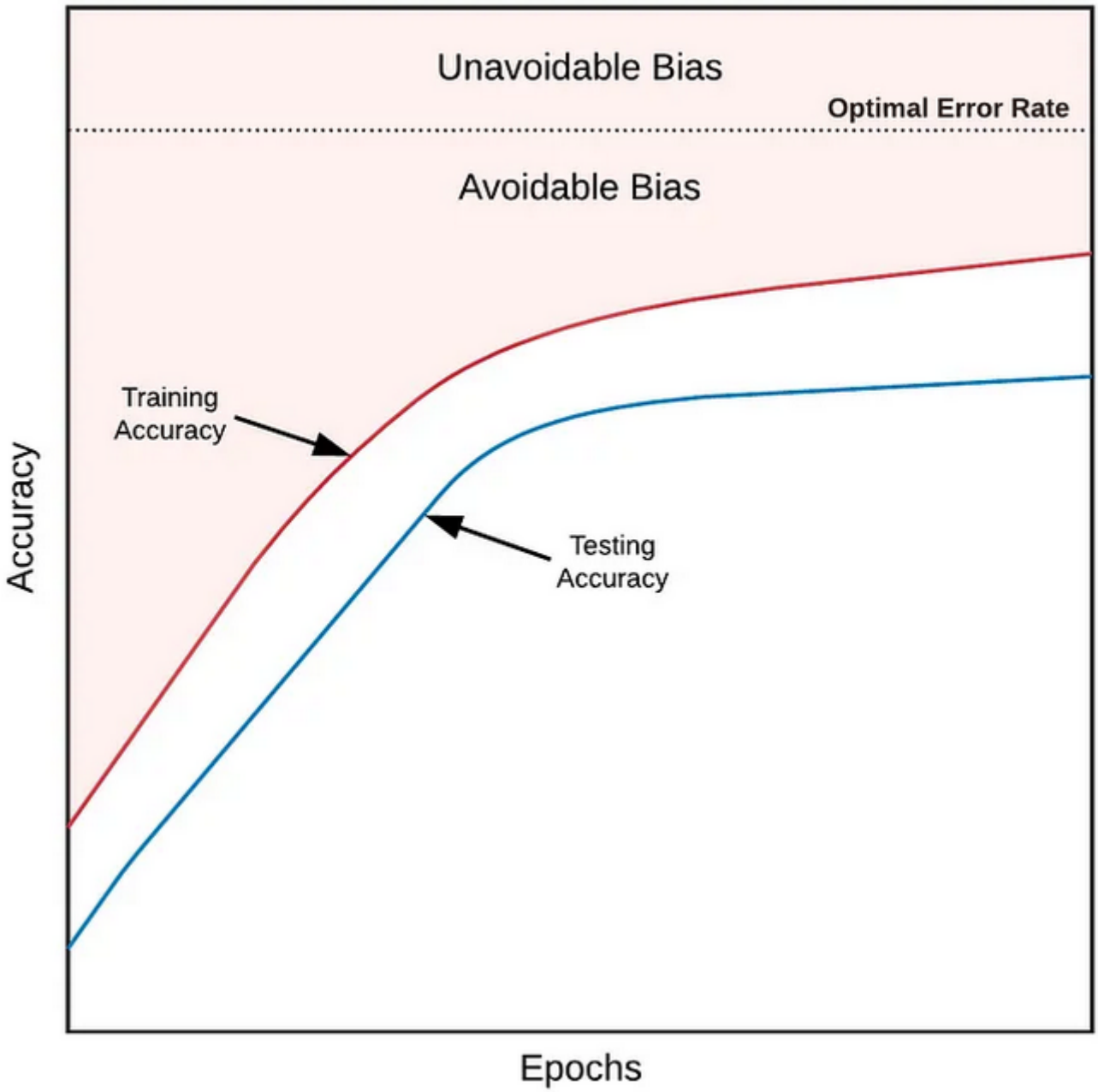
@ Fit the training set very well (Achieving LOW AVOIDABLE BIAS)
@ Training Set performance generalizes to the test set/ dev set (Achieving LOW AVOIDABLE VARIANCE)



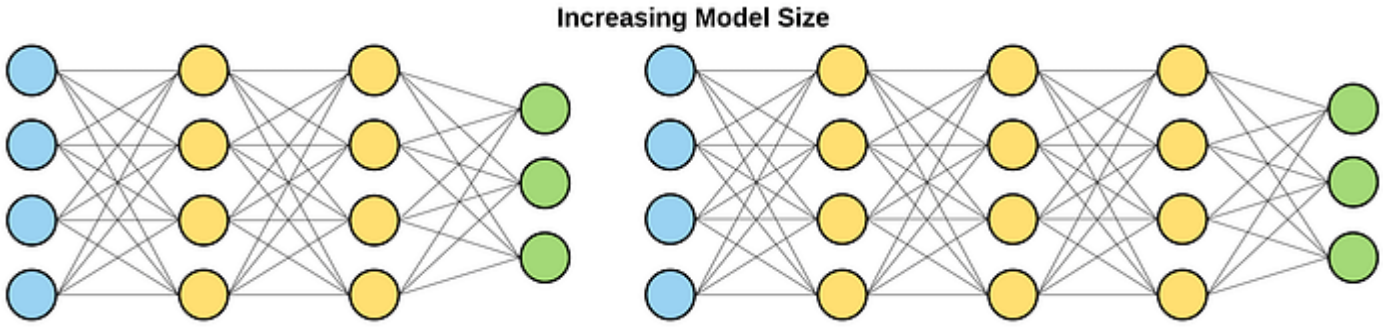
Two takeaways from the above graph:(1).Bias is the error for the training set (2).Variance is the gap b/w the training and testing accuracy

REDUCING AVOIDABLE BIAS

Avoidable bias is the difference between the optimal error rate and the training error. This is the error that we can try to reduce to achieve the optimal error rate.



Increase model size: The larger the model the more parameters to tune. More parameters allow the model to learn more complicated relationships. You can increase the size of the model by adding more layers or nodes to the model. The better the model can learn from the data the closer it gets to the optimal error rate.



Reduce Regularization: Reducing regulation for a model allows the model to fit the training data better.But,less regularization means that your model won’t generalize as well. This is what we call as The BIAS-VARIANCE TRADE OFF.

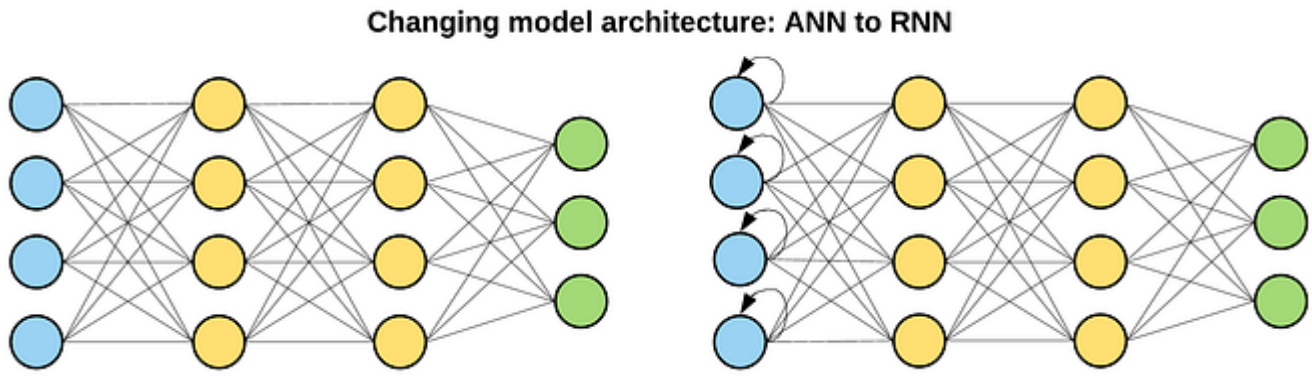
Change model architecture: These techniques can change **both** bias & variance.

1.Layer-activation functions (tanh,relu,sigmoid,…)

2.What the model is learning (ANN,CNN,RNN,KNN,…)

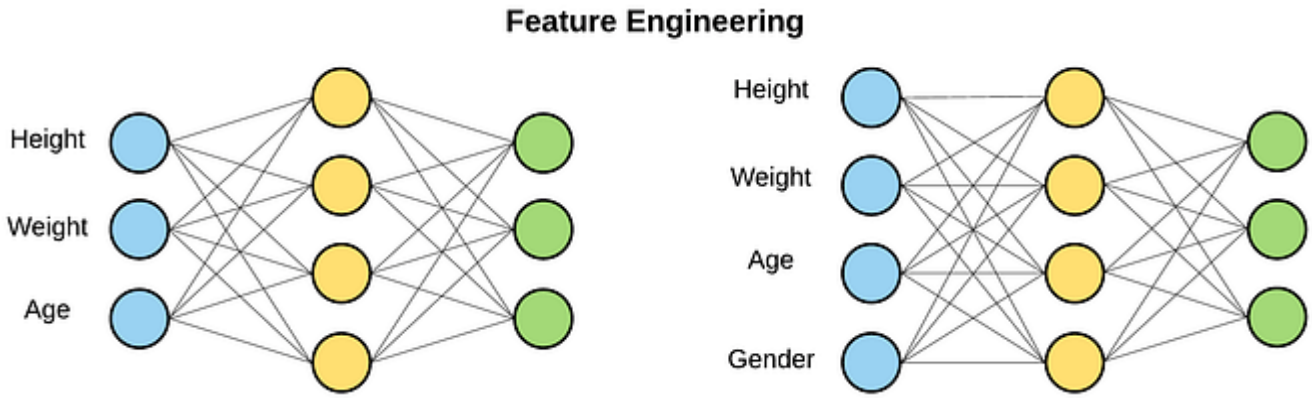
3.How the model is learning (Adam,SGD,RMSprop,…)

4.Change other hyperparameters (Learning-rate,image-size,…)



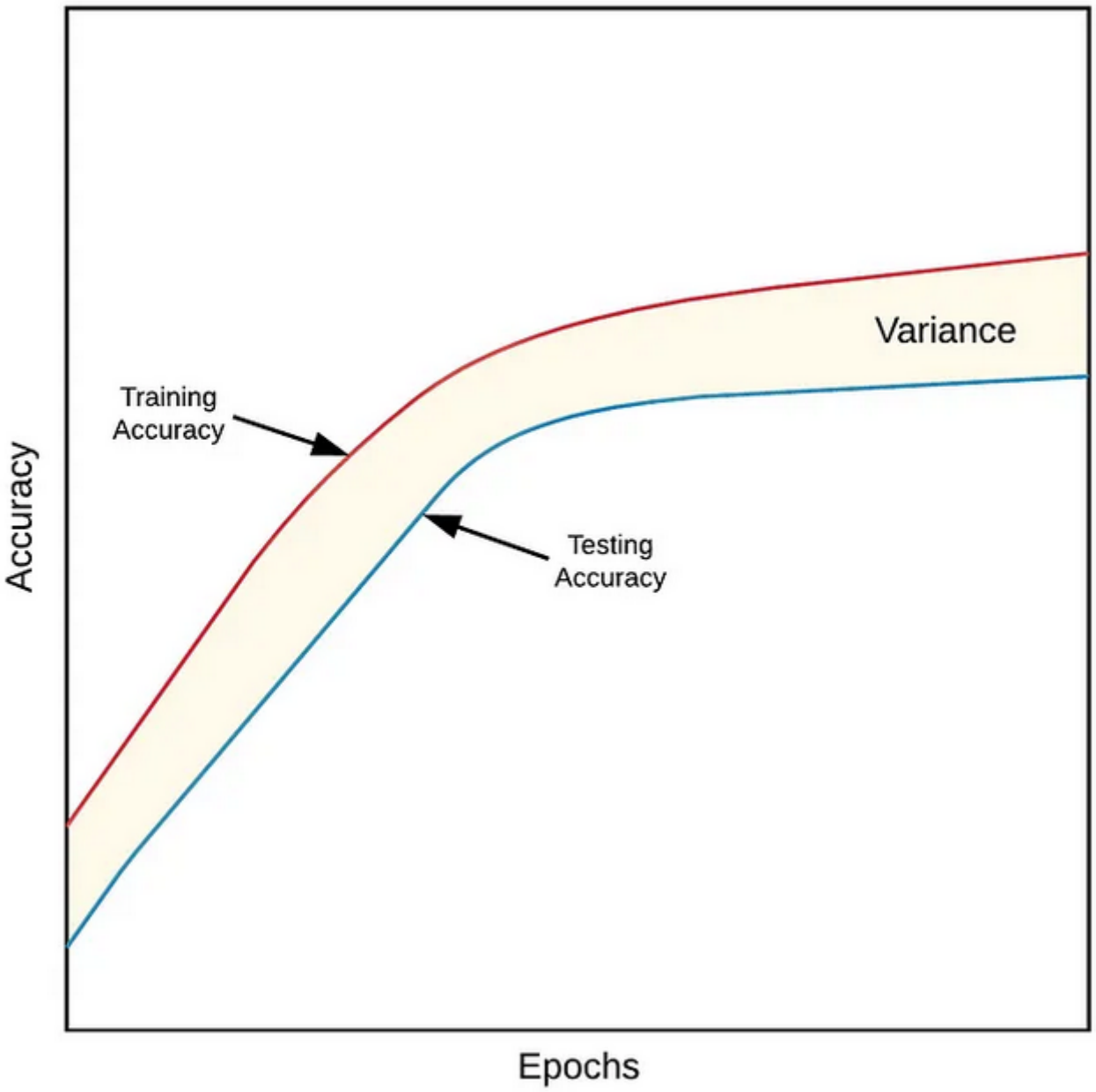
Add new features : Adding new features to the training data can give more information to the model that it can use to learn from.

This can be done through a process called feature engineering. During this process, you can also add features back in that you cut earlier in development.



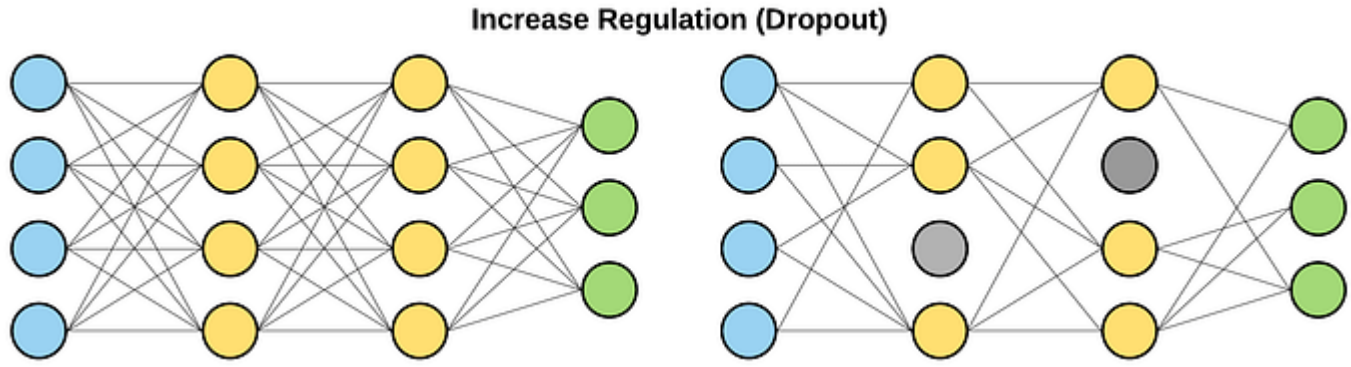
REDUCING AVOIDABLE VARIANCE

The variance describes how well your model can generalize to data it has not seen yet. We define variance as the difference between the training accuracy and the testing accuracy.



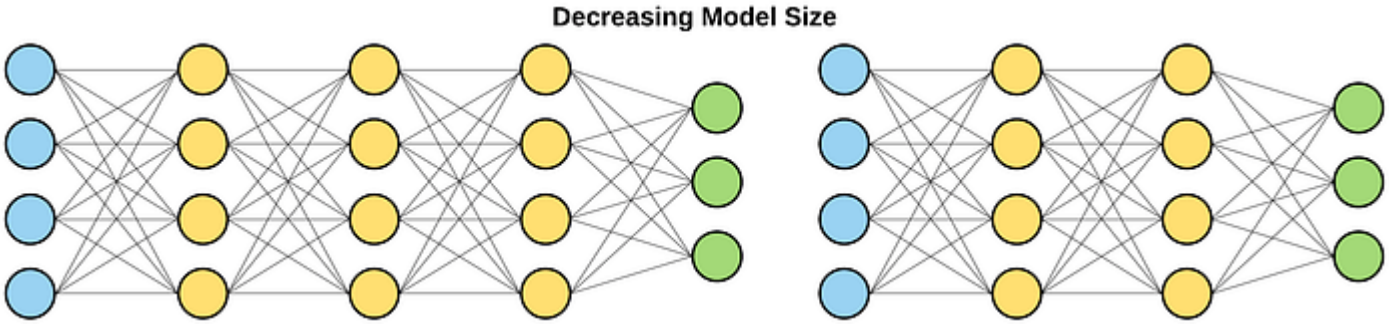
Add more data : Adding more data is the simplest way to, *almost* always, increase your model’s performance.

Increase Regularization : Adding regularization prevents the model from overfitting on the data. Although this reduces variance, it will always increase bias. In addition to reducing variance, adding regularization can have significant positive impacts.

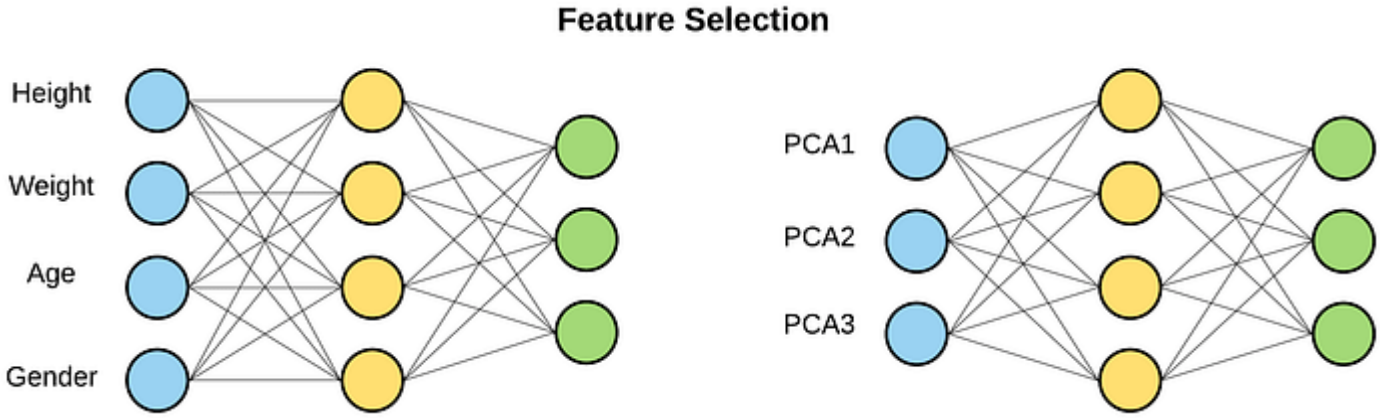


Decrease model size : Decreasing the model size will help reduce overfitting on the training data. Although this technique is the easiest, it reduces the model’s capability of learning the complex patterns of the data set.

The same results are usually seen by adding regularization and thus that method is more preferred.



Feature selection : Reducing the dimensionality of your data set by removing features that are not needed is a great way to reduce the variance of your model. You can use Principal Component Analysis (PCA) to filter out features or combine them into several principal components.



DATA MISMATCH

In some cases, it’s easy to get a large amount of data for training, but this data probably won’t be perfectly representative of the data that will be used in production.

Suppose you want to create a mobile app to take pictures of flowers and automatically determine their species. You can easily download millions of pictures of flowers on the web, but they won’t be perfectly representative of the pictures that will actually be taken using the app on a mobile device. Perhaps you only have 10,000 representative pictures (i.e., actually taken with the app). **The most important rule to remember is that the validation set and the test set must be as representative as possible of the data you expect to use in production.**

So they should be composed exclusively of representative pictures : you can shuffle those **representative images** and put half in the **validation set** and a half in the **test set** (making sure that no duplicates or near-duplicates end up in both sets). But **after training your model on the web pictures,** if performance of the model on the **validation set** is disappointing, you will **not know** whether this is because your model has overfitting the training set, or whether this is just due to the mismatch. **One solution is to hold out some of the training pictures (from the web) in yet another set called as the train-dev set.**

So you’ve got 10k representative images and 500k images from web, Create **train-dev set** which is full of images from web. Train model on the **training set.** Evaluate on the **train-dev set** If it performs well, then the model is **not** overfitting the training set.Evaluate on the **train-dev set.** If it performs poorly, then it must have overfitted the training set, so you should try to simplify or regularize the model, get more training data, and clean up the training data. If it performs poorly on the **validation set,** the problem must be coming from the data mismatch.

COMPUTER VISION

Computer vision is a field of artificial intelligence (AI) that enables computers and systems to derive meaningful information from digital images, videos and other visual inputs. There are two major stuffs in Computer Vision one which is the so-called Image Classification and another is the so-called Object Detection. Read more : HERE

EDGE DETECTION : A CONVOLUTION

Each layer in a CNN is capable of recognizing a particular feature such as Edge, Color and so on. Here, let’s see how edge detection is made. A Random matrix is multiplied in an element wise manner with a given grayscale image from which an edge is to be detected.

10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0
10	10	10	0	0	0

6 x 6

*

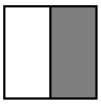
1	0	-1
1	0	-1
1	0	-1

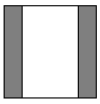
3 x 3

=

-0	30	30	0
0	30	30	0
0	30	30	0
0	30	30	0

4 x 4



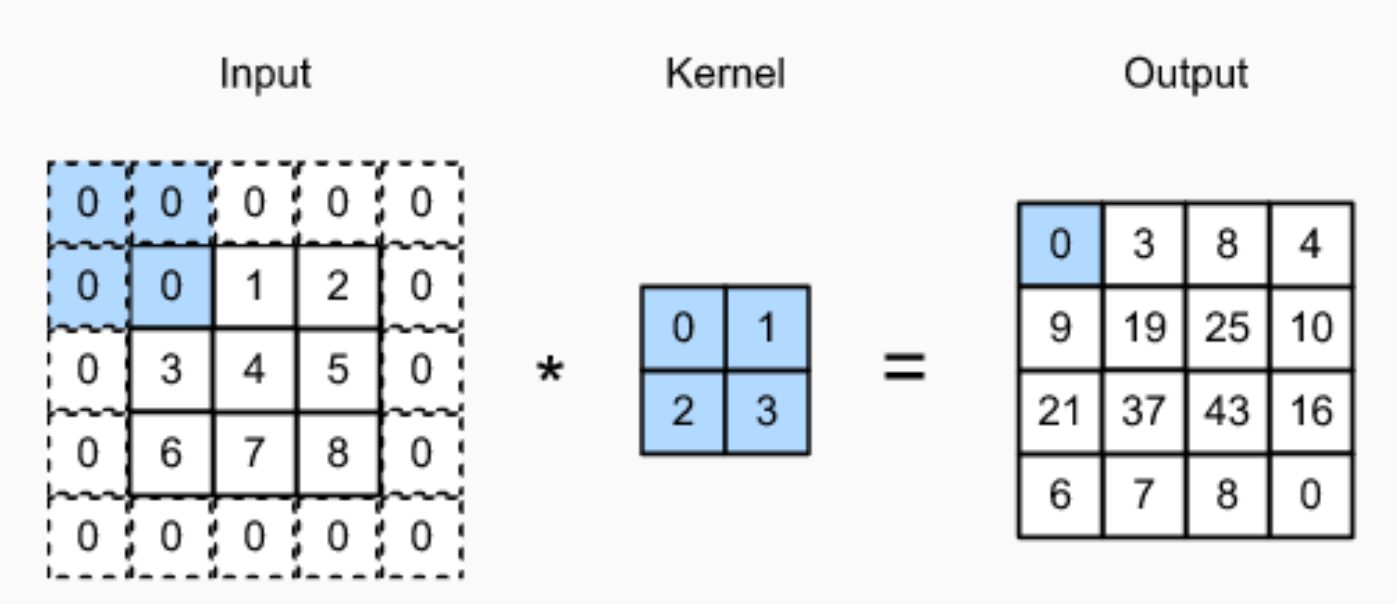


The resultant image detects the vertical image of the 6x6 gray-scale image as a white region in the middle. The 3x3 matrix used here is known to be the kernel matrix. We can apply different kernels to extract a particular pattern from the given image. This operation is said to be the so-called Convolution.

“IF WE CONVOLVE NXN MATRIX WITH KxK KERNEL, RESULTANT WILL BE A N-K+1 x N-K+1 MATRIX”

PADDING

The volume shouldn’t shrink too fast while convolved repeatedly with kernels. Padding with zero is a helpful remedy for this problem.[Also for Edge Detection]



Based on padding , whether we do it or not, we end up with two definitions.

@ VALID CONVOLUTION : No padding

@ SAME CONVOLUTION : Padded such that the output dimension is same as input dimension.

Padding Size, P should be (K-1)/2

[DIMENSION : N+2P-K+1 X N+2P-K+1]

STRIDED CONVOLUTION

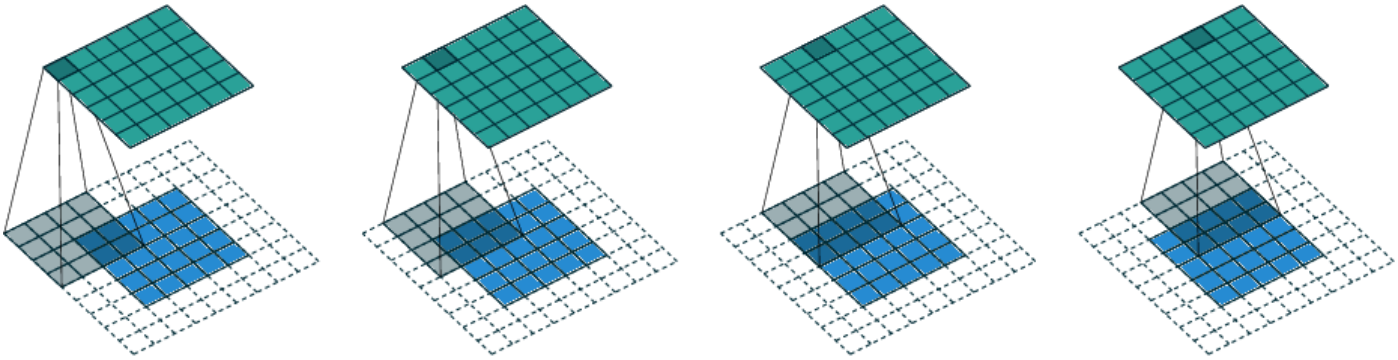


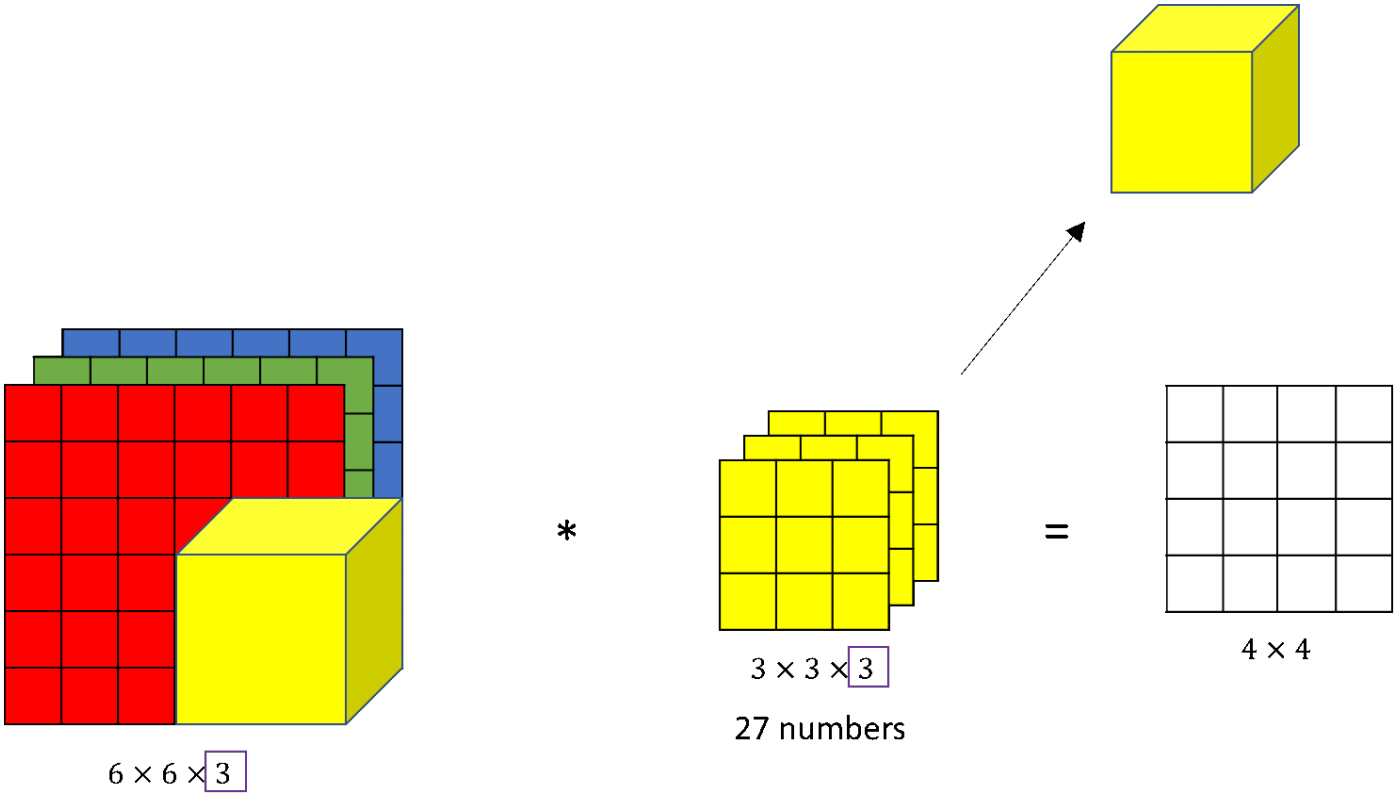
Figure 3: STRIDE : 0

In Edge detection, we convolved with no hops. But, what if we hop while convolving ? The step size for convolving is the so-called stride. It is also a hyper-parameter. For a definition, Stride is a parameter of the neural network's filter that modifies the amount of movement over the image. While applying such stride, the formula for new dimesion is,

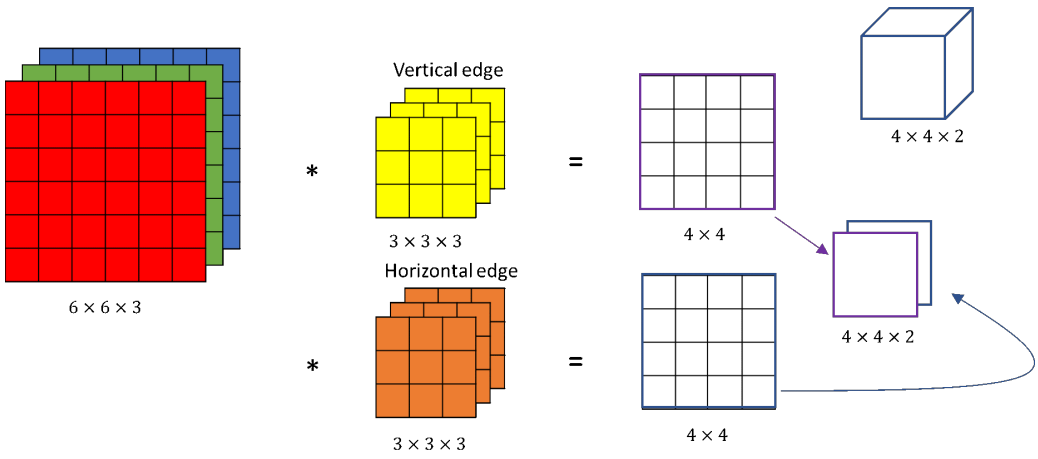
$\lfloor (N+2P+K)/S +1 \rfloor \times \lfloor (N+2P+K)/S +1 \rfloor$

where, S : Stride Size

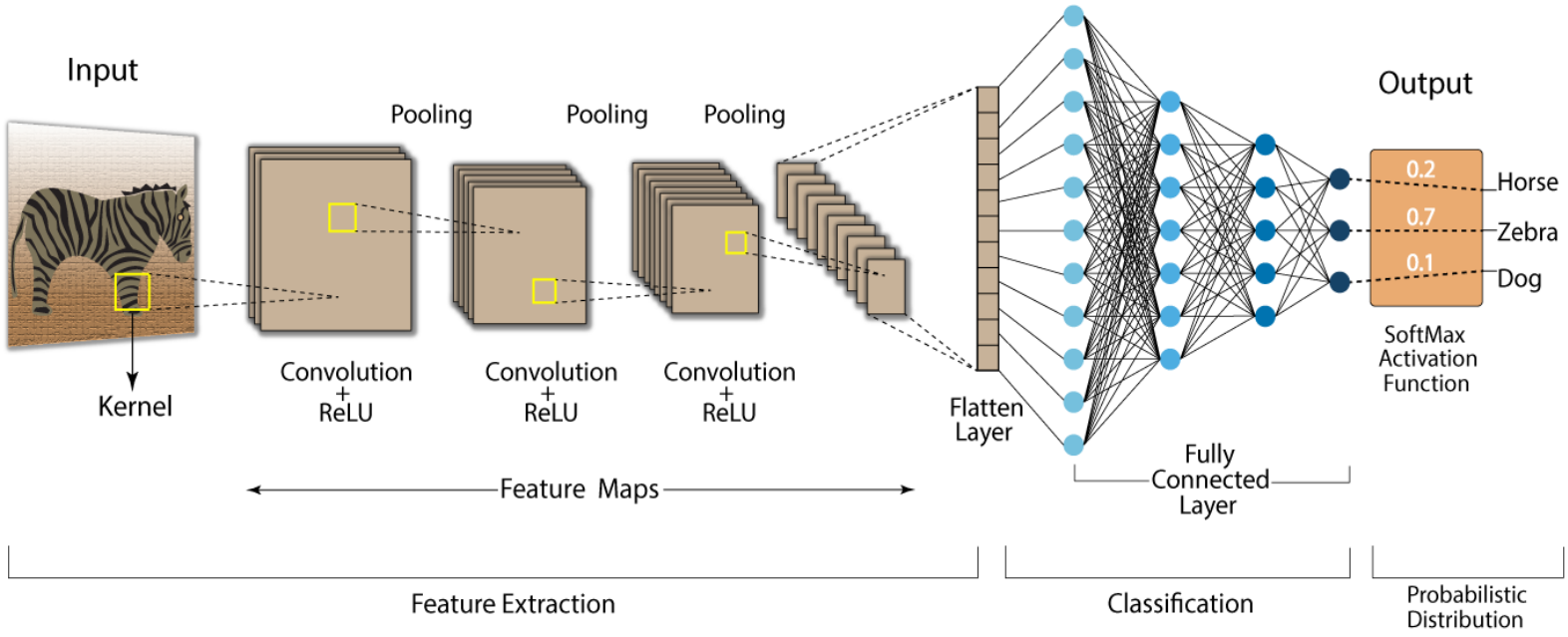
CONVOLUTION OVER RGB IMAGE



The Kernels are convolved over the 3D Array of pixels of 3 channels as usual. The first resultant value which is the sum of element-wise products from the three channels. If we have multiple kernels to apply, just stack the kernels as shown below.



A SIMPLE CNN



CHECK OUT : HERE

POOLING PROCESS

Pooling is a sort of process performed to pick more/less dominant feature. Max-Pooling is a sort of operation followed after the series of convolutions. It is for reducing the dimension of the output from the convolutional layer by using a kernel which moves with a stride and captures the maximum value from the activation map.

We must be thinking that is down-scaling the images is the only use of it. There are many advantages of using Max-Pooling over other Pooling operations. Max Pooling adds translation invariance.

1. Shift Invariance
2. Rotational Invariance
3. Scale Invariance

The three types of pooling :

1. Max pooling : The maximum pixel value of the batch is selected.
2. Min pooling : The minimum pixel value of the batch is selected
3. Average pooling : Theaverage value of all the pixels is selected.

We cannot say that a particular pooling method is better over other generally. The choice of pooling operation is made based on the data at hand. Max pooling selects the brighter pixels from the image. It is useful when the background of the image is dark and we are interested in only the lighter pixels of the image. For example: in MNIST dataset, the digits are represented in white color and the background is black. So, max pooling is used. Similarly, min pooling is used in the other way round. Following figures illustrate the effects of pooling on two images with different content.

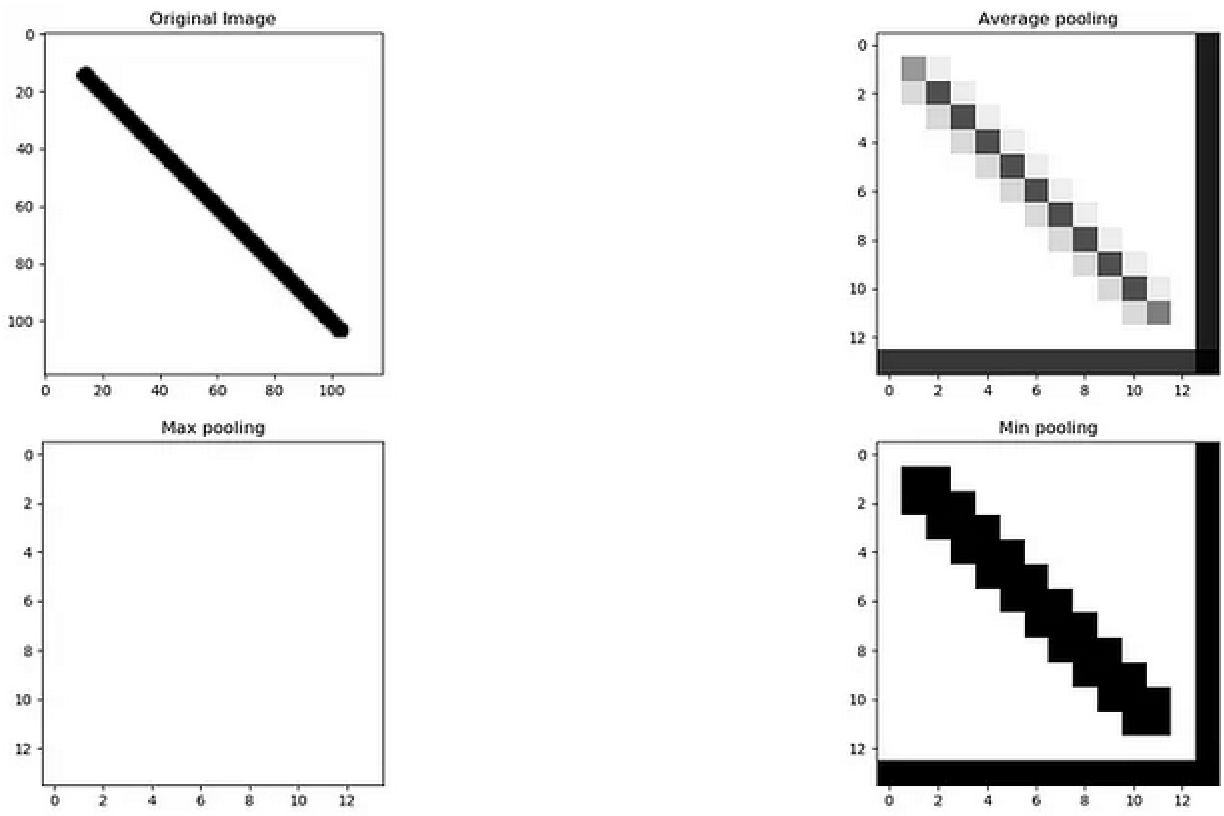
Conv1

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.32	0.45	0.61	0.72	0.54	0.36	0.14	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.30	0.58	1.18	1.41	1.64	1.64	1.64	1.64	1.91	1.95	2.21	2.31	2.10	2.04	1.69	0.87	0.48	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.47	1.22	1.83	2.57	2.73	3.00	3.00	3.00	3.00	3.39	3.50	3.83	3.97	3.71	3.57	3.16	2.30	1.46	0.54	0.05	0.00	0.00	0.00	0.00	0.00	0.00
1.07	2.05	3.16	4.21	4.39	4.74	4.74	4.54	4.20	4.01	3.85	3.90	3.92	4.10	4.48	4.65	4.06	3.11	1.53	0.54	0.05	0.00	0.00	0.00	0.00	0.00
1.14	2.01	3.13	3.58	3.28	3.22	3.22	3.09	2.91	2.72	2.51	2.51	2.51	2.65	2.99	3.91	4.38	4.12	2.85	1.37	0.33	0.00	0.00	0.00	0.00	0.00
0.92	1.37	2.06	2.17	1.79	1.71	1.71	1.49	1.14	0.82	0.53	0.53	0.53	0.77	1.27	2.43	3.72	4.53	3.98	2.38	0.96	0.00	0.00	0.00	0.00	0.00
0.14	0.34	0.28	0.33	0.05	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.13	1.29	2.83	4.25	4.66	2.76	1.65	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.35	1.16	2.91	3.93	5.13	3.07	2.19	0.12	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.08	0.42	0.97	1.33	1.59	1.91	2.39	3.69	4.36	5.19	3.83	2.50	0.71	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.23	0.49	1.10	1.68	2.29	2.65	2.96	3.41	3.72	4.61	4.91	5.24	4.12	2.53	1.20	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.12	0.67	1.32	1.89	2.57	3.17	3.96	4.43	4.77	4.38	4.22	4.49	4.79	5.19	4.53	2.72	1.60	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.38	0.96	1.82	2.60	3.32	3.77	3.94	3.80	3.65	3.38	2.97	2.93	3.56	4.09	5.00	3.84	2.48	0.97	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.82	1.71	2.96	3.51	3.68	3.30	3.04	2.48	2.23	1.93	1.29	1.28	2.43	3.29	4.83	3.43	2.34	0.56	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.85	1.96	2.72	3.18	2.58	1.77	1.32	0.65	0.38	0.12	0.00	0.39	2.22	3.27	4.63	2.96	2.02	0.16	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.66	1.40	1.65	1.74	0.74	0.24	0.00	0.00	0.00	0.00	0.00	0.83	2.51	3.67	4.25	2.61	1.50	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.07	0.48	0.20	0.48	0.00	0.00	0.00	0.00	0.00	0.00	0.70	1.72	3.28	3.98	3.56	2.23	0.80	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.27	2.27	3.98	3.90	2.82	1.58	0.22	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.30	2.26	3.10	4.46	3.42	2.04	0.80	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	0.89	2.56	3.36	3.81	2.51	1.22	0.22	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.02	1.18	2.01	2.84	2.42	1.46	0.33	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.03	1.02	1.34	1.98	1.31	0.60	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

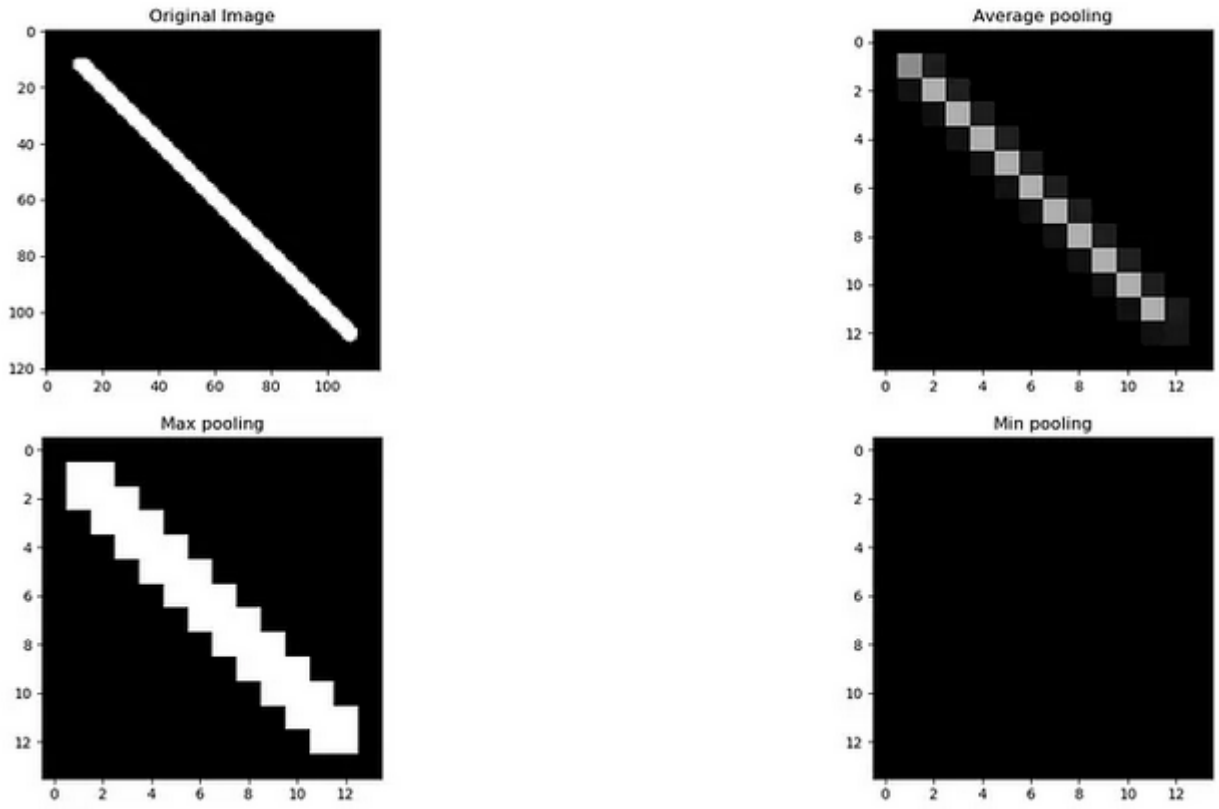
Max Pool

0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.32	0.61	0.72	0.36	0.00	0.00	0.00	0.00	0.00
1.22	2.57	3.00	3.00	3.39	3.83	3.97	3.57	2.30	0.54	0.00	0.00	0.00
2.05	4.21	4.74	4.74	4.20	3.90	4.10	4.65	4.38	2.85	0.33	0.00	0.00
1.37	2.17	1.79	1.71	1.14	0.53	0.77	2.43	4.53	4.66	1.65	0.00	0.00
0.00	0.00	0.00	0.00	0.08	0.97	1.59	2.39	4.36	5.19	2.50	0.00	0.00
0.00	0.00	0.12	1.32	2.57	3.96	4.77	4.38	4.91	5.24	2.72	0.00	0.00
0.00	0.00	1.71	3.51	3.77	3.94	3.65	2.97	4.09	5.00	2.48	0.00	0.00
0.00	0.00	1.96	3.18	2.58	1.32	0.38	0.83	3.67	4.63	2.02	0.00	0.00
0.00	0.00	0.48	0.48	0.00	0.00	0.00	2.27	3.98	3.56	0.80	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.89	3.36	4.46	2.04	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	1.18	2.84	2.42	0.33	0.00	0.00	0.00

MIN-POOLING :

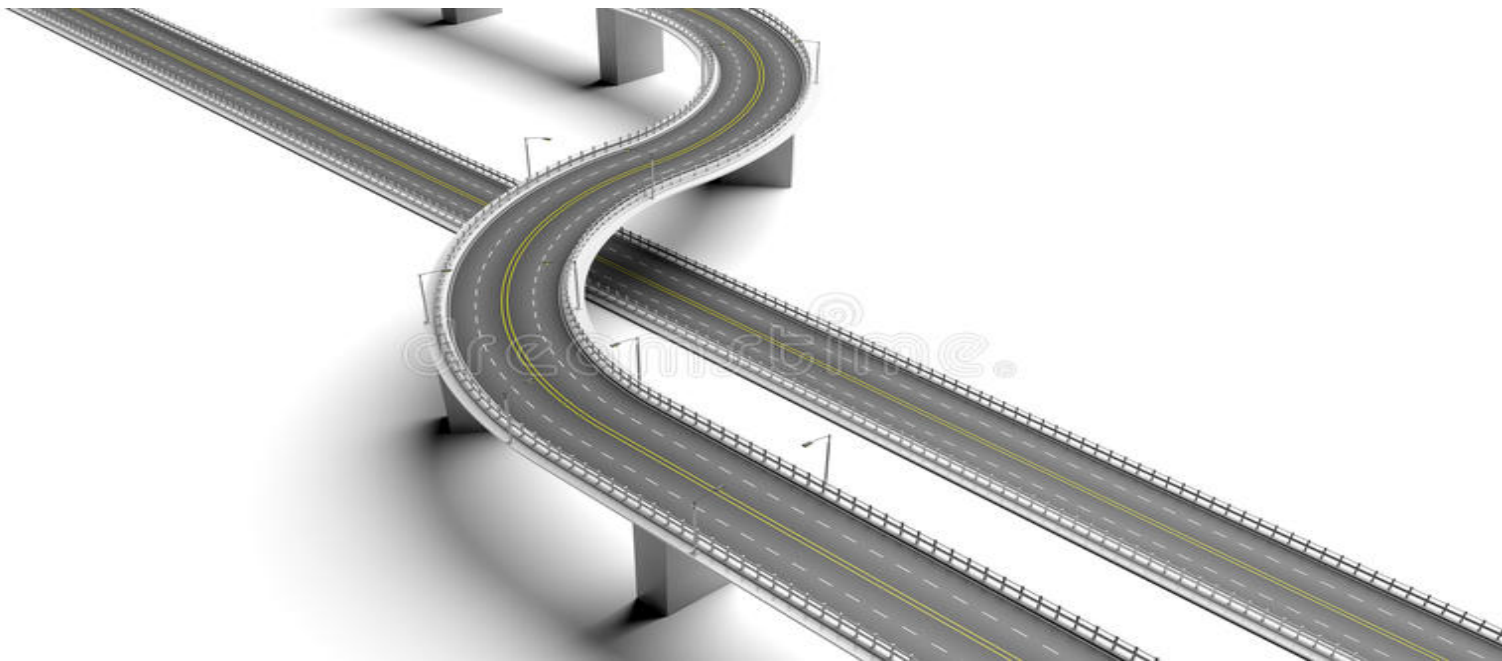


MAX-POOLING :

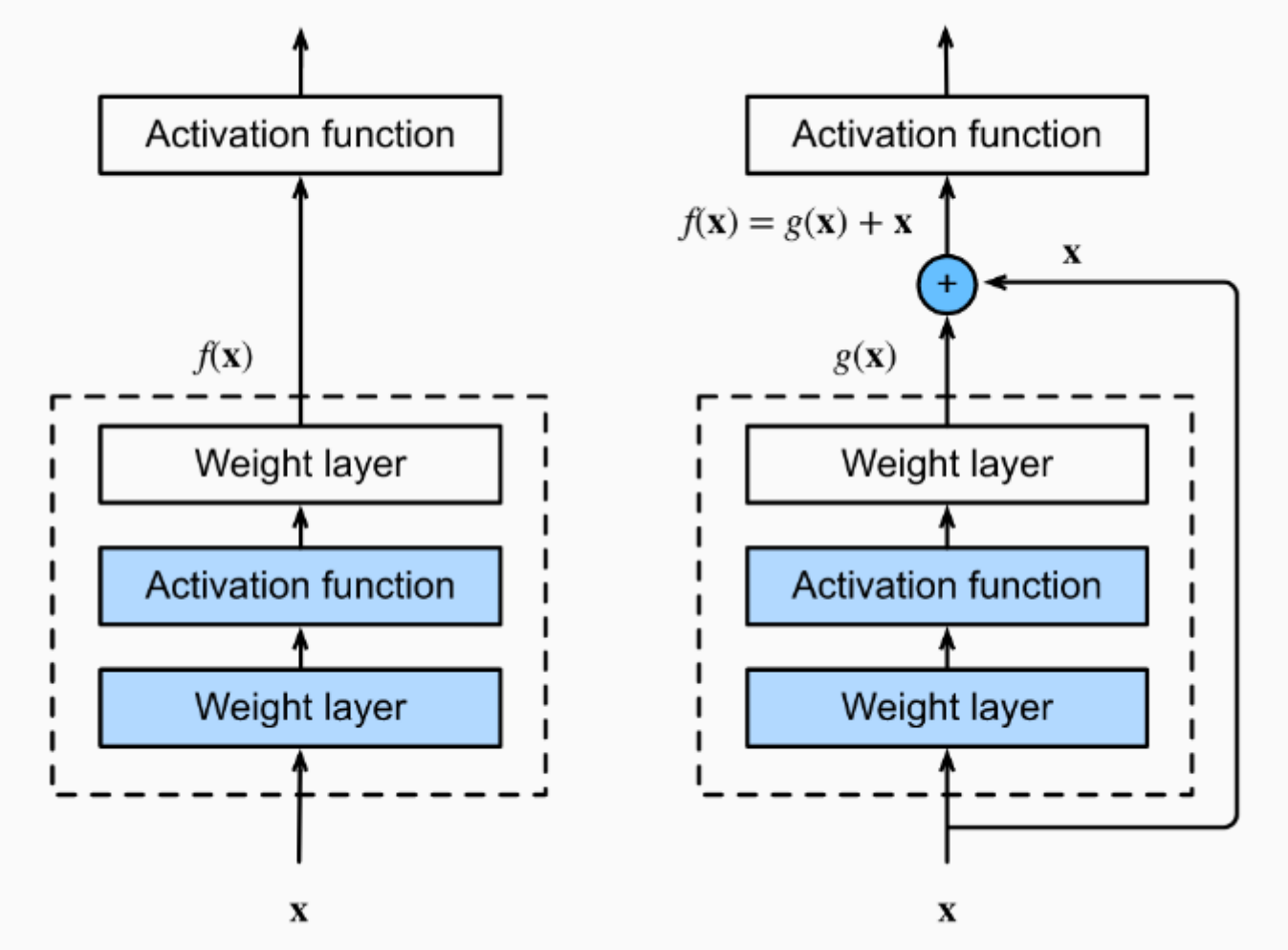


RESNET ARCHITECTURE

The Very-Very Deep Neural Networks are hard to train because of the dangers called Vanishing Gradient and Exploding Gradient Problems.



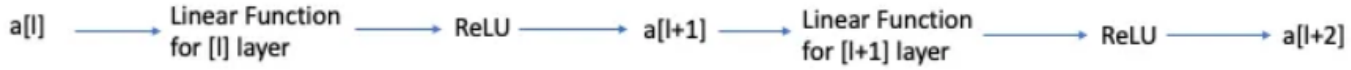
What if we can bypass some layers and try to overcome these gradient problems ? We’re about to incorporate the skip-connections here.



This allows us to take activation from one layer and feed it to another layer, even much deeper in the neural network, hence sustaining the learning parameters of the network.

WHY RESNET WORKS ?

Consider the following PLAIN NETWORK (the one w/o skip connections)



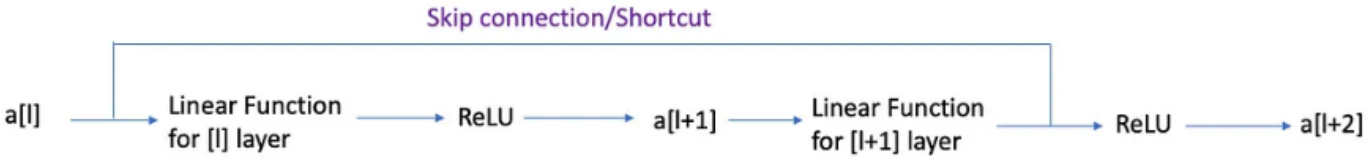
$z[l+1] = w[l+1]a[l] + b[l+1]$

ReLU: $g(z(l+1)) = a[l+1]$

$z[l+2] = w[l+2]a[l+1]+b[l+2]$

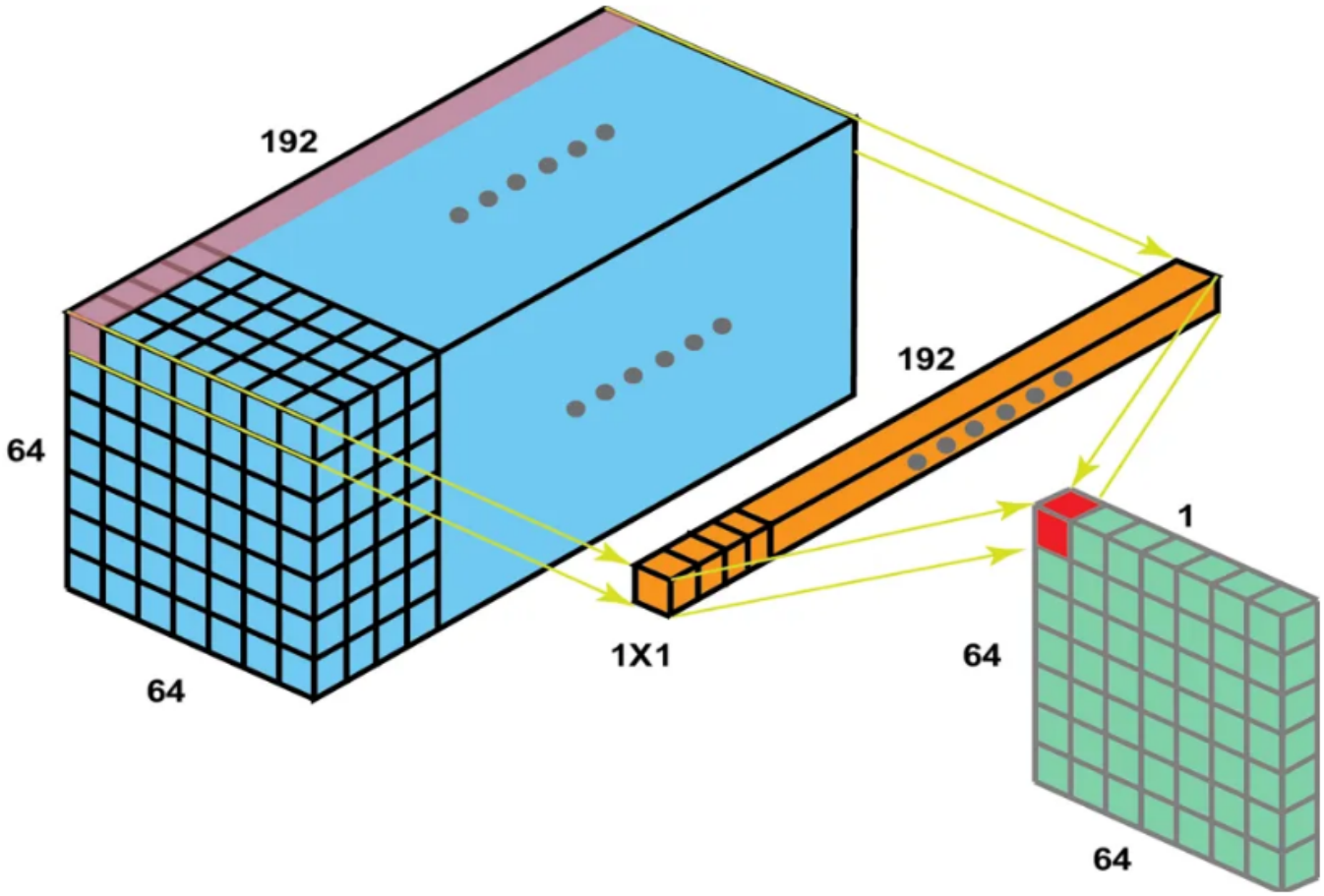
ReLU: $g(z[l+2]) = a[l+2]$

However, a residual block in its [l+2]th layer, along with the activations from a[l] i.e. g(z[l]), also uses the parameters from the previous activation function, i.e. a[l] itself.

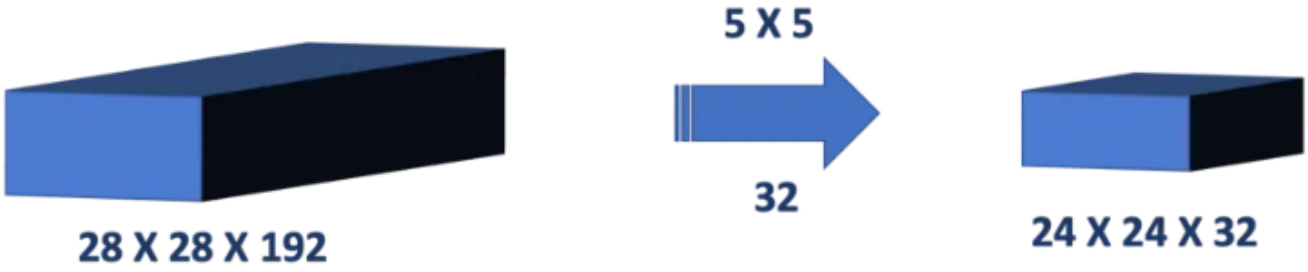


1x1 CONVOLUTION : N/W IN N/W

Multiplying a NxNx1 Vector with a 1x1 seems funny as it looks like a simple scaling of the input vector. But, when we have an NxNxM vector, it will be something more essential in the case of convolutional neural network. So, what is it? 1X1 Conv is used to reduce the number of channels while introducing non-linearity.

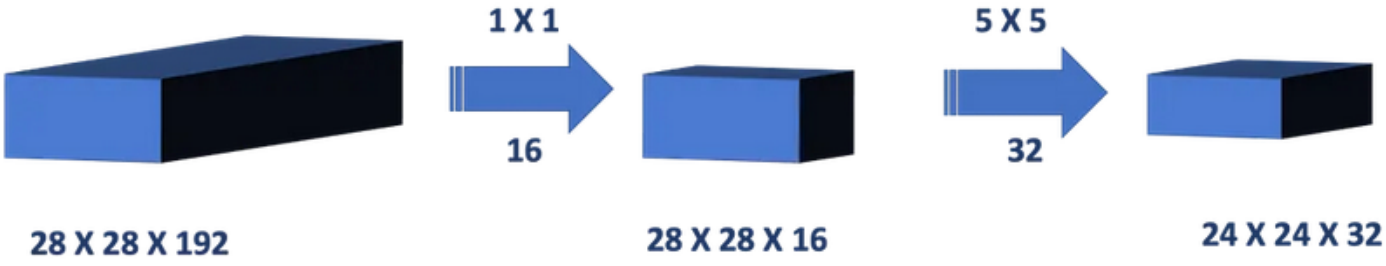


It can be used for computational load reduction. In normal method (w/o 1x1 Conv), the number of operations required are so-so higher in number.



Number of Operations : (28X28X32) X (5X5X192) = 120.422 Million Ops

Meanwhile, with dimensionality reduction, it will be considerably efficient than the one before.

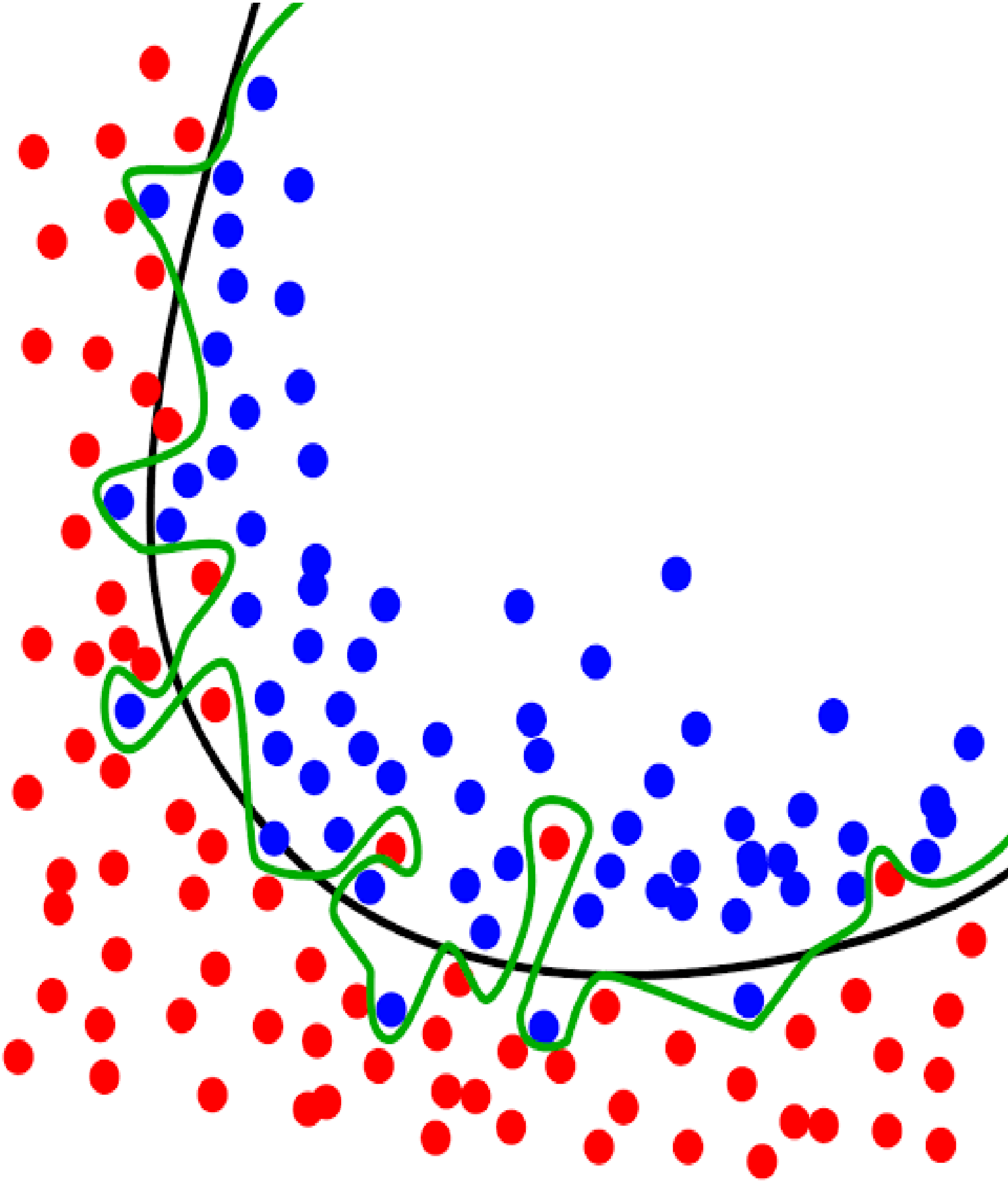


Number of Operations for 1 X 1 Conv Step : (28X28X16) X (1X1X192) = 2.4 Million Ops
Number of Operations for 5 X 5 Conv Step : (28X28X32) X (5X5X16) = 10 Million Ops
Total Number of Operations = 12.4 Million Ops

INCEPTION : PREMISE

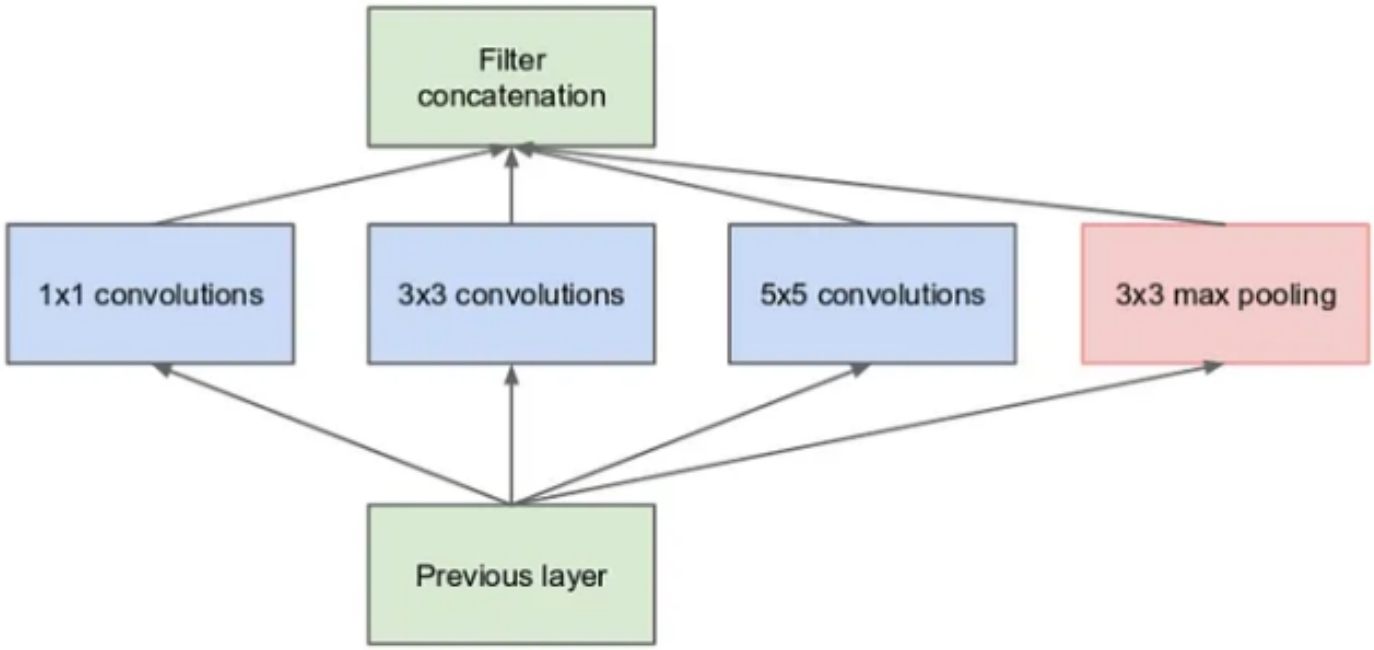
Various parts in an image may have extremely large variations in space occupation. For an instance, the dog in the images shown above occupy different amount of space considerably. Because of this huge variation in the location of the information, choosing the right kernel size for the convolution operation becomes tough. A larger kernel is preferred for information that is distributed more globally, and a smaller kernel is preferred for information that is distributed more locally.

Very deep networks prone to over-fitting and it’s hard to pass gradient updates through the network.



THE INCEPTION MODULE

Why not have filters with **multiple sizes** operate on the **same level**? The network essentially would get a bit **“wider”** rather than **“deeper”**. This is the birth of the so-called Inception Module.

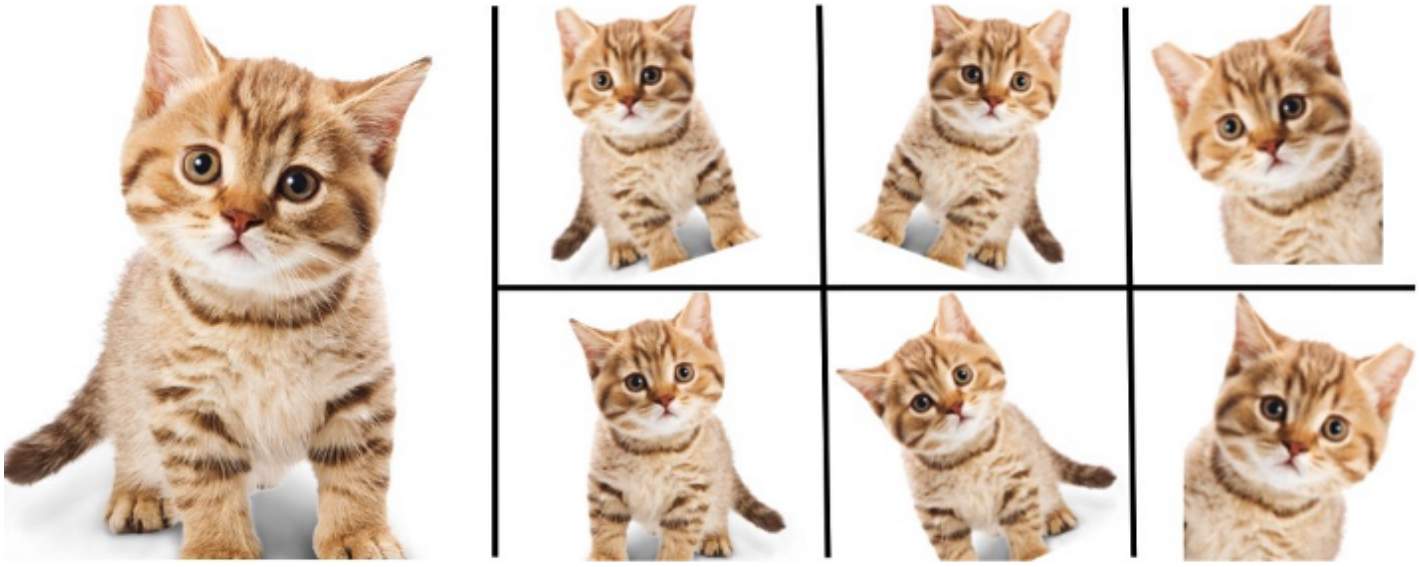


An **Inception Module** is an image model block that aims to approximate an optimal local sparse structure in a CNN.

Put simply, it allows for us to use multiple types of filter size, instead of being restricted to a single filter size, in a single image block, which we then concatenate and pass onto the next layer.

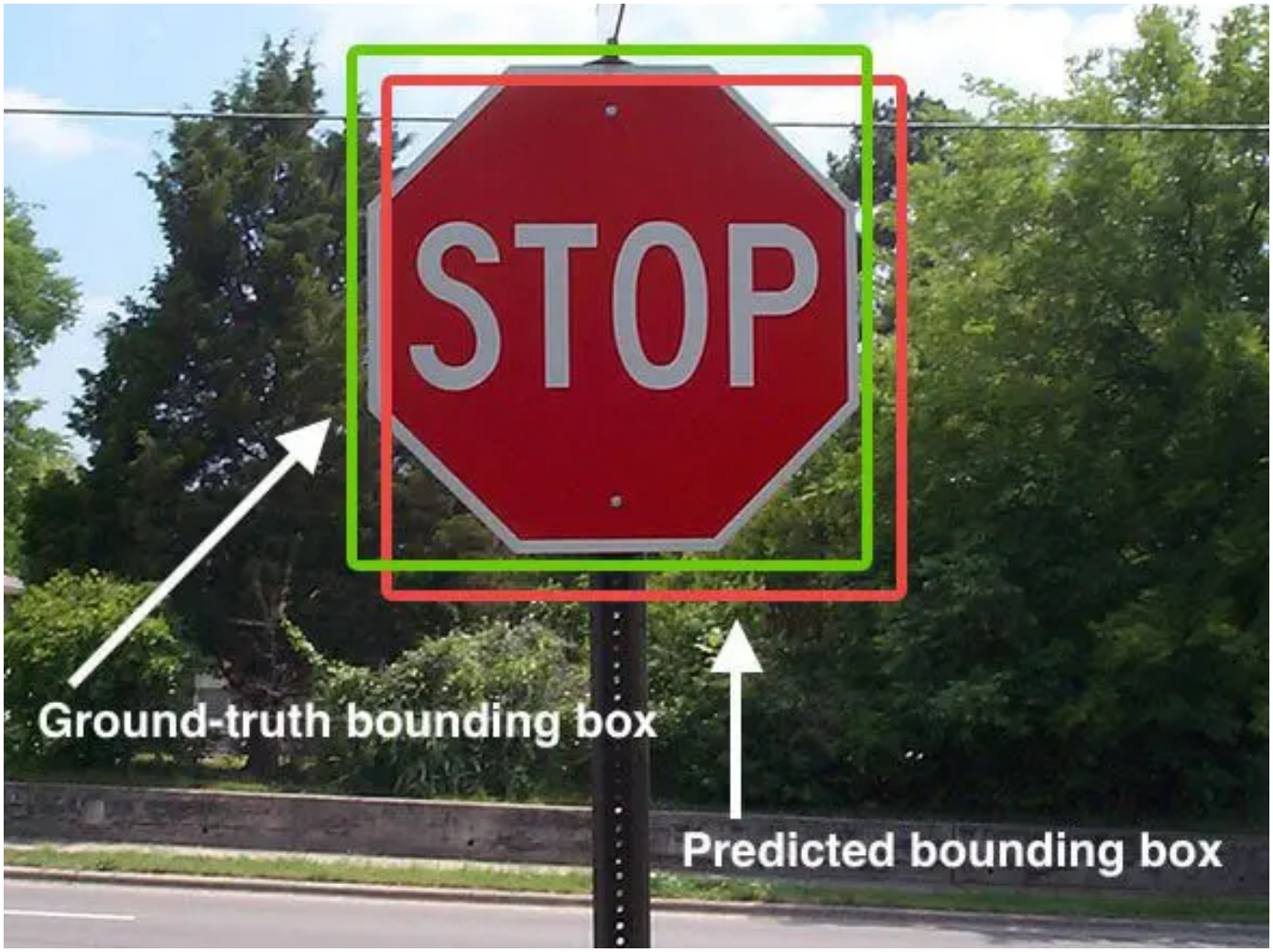
DATA AUGMENTATION

Data augmentation is a technique of artificially increasing the training set by creating modified copies of a dataset using existing data.



INTERSECTION OVER UNION

Intersection over Union (IoU) is used to evaluate the performance of object detection by comparing the ground truth bounding box to the predicted bounding box.



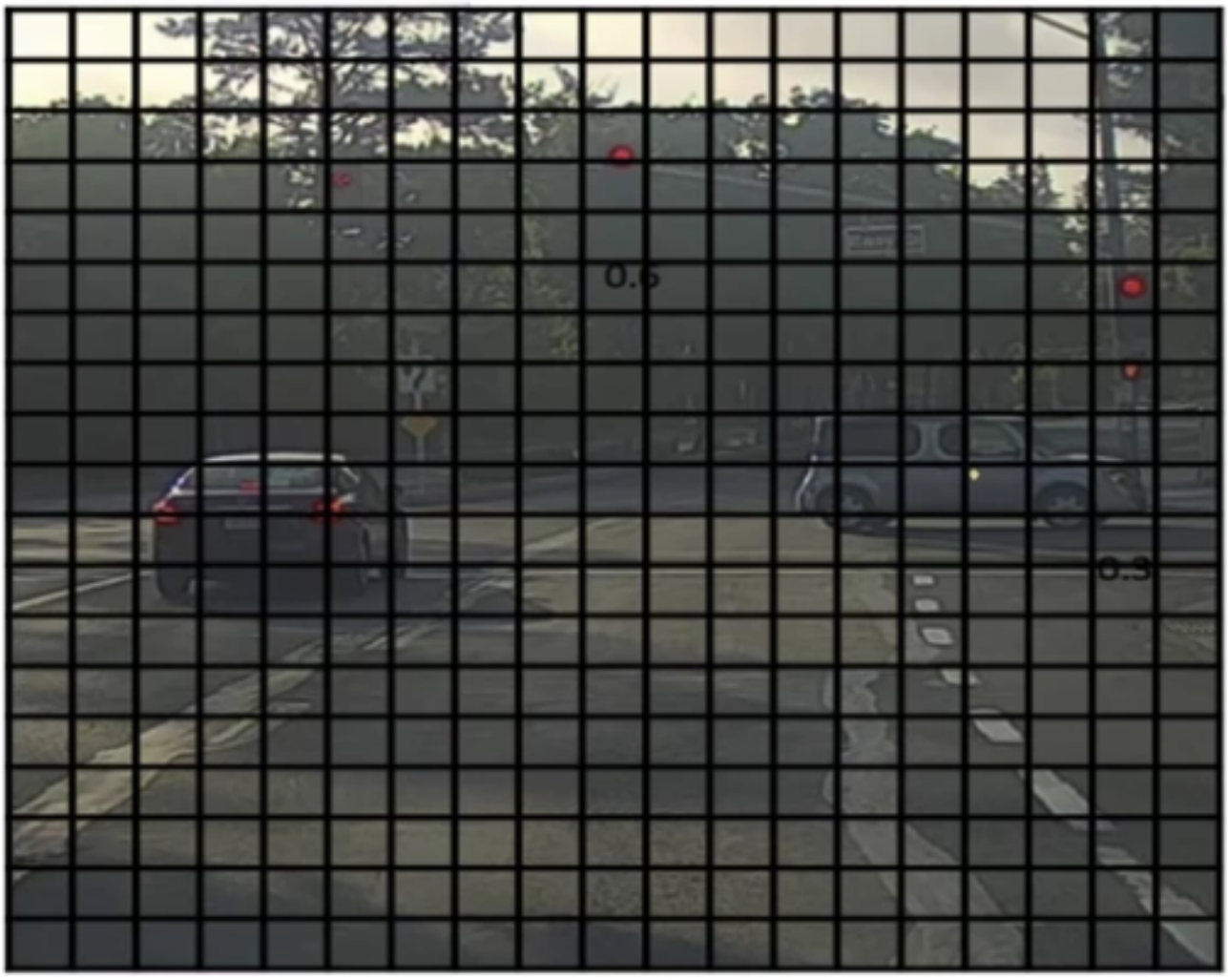
It is calculated as the ratio of the size of the intersection area and the size of the union of the two.

$$IoU = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

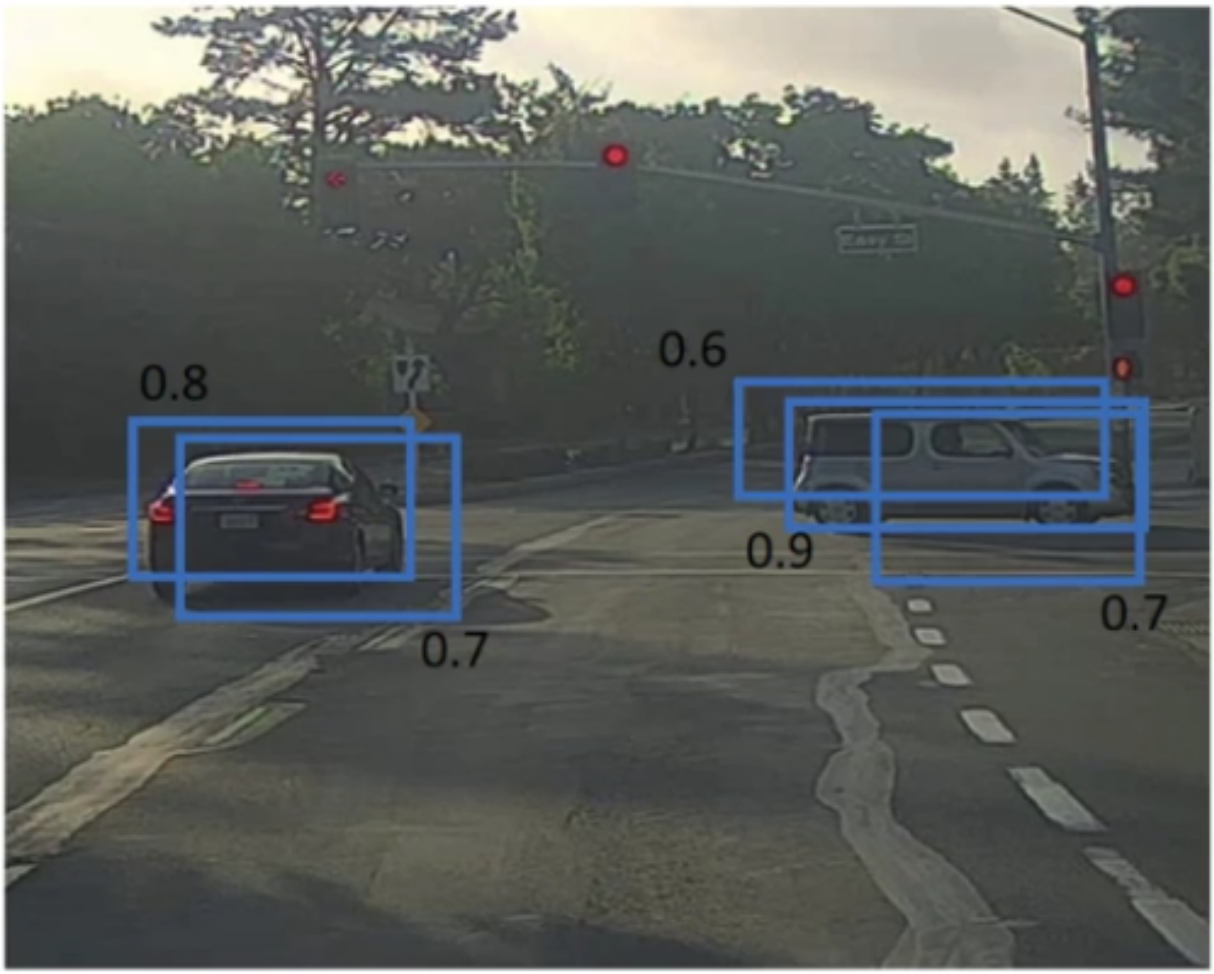
As we can observe below, predicted bounding boxes that heavily overlap with the ground-truth bounding boxes have higher scores than those with less overlap. This makes Intersection over Union an excellent metric for evaluating custom object detectors.

NON-MAX SUPPRESSION

There is a chance of detecting a single object multiple times. To overcome this problem, there is a need for the so-called Non-Max suppression.



If the car is detected multiple times, there will be different rectangles drawn.

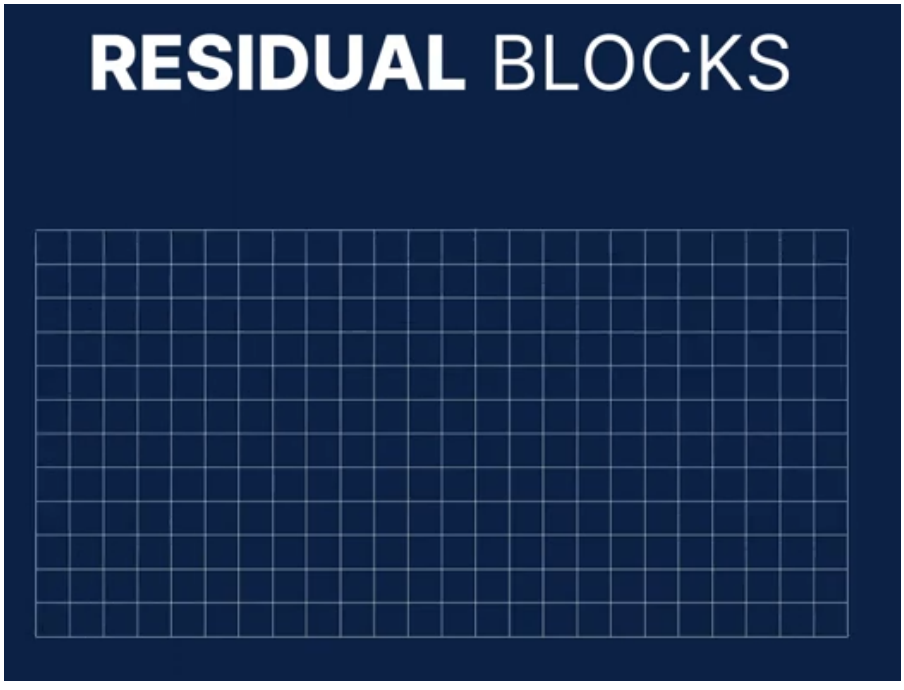


The Non-Max suppression simply cleans up these so they end up with one detection rather than more. It looks for the detected objects with higher probabilities. On finding a detection with high probability, it suppresses the remaining and hence the name Non-Max Suppression.

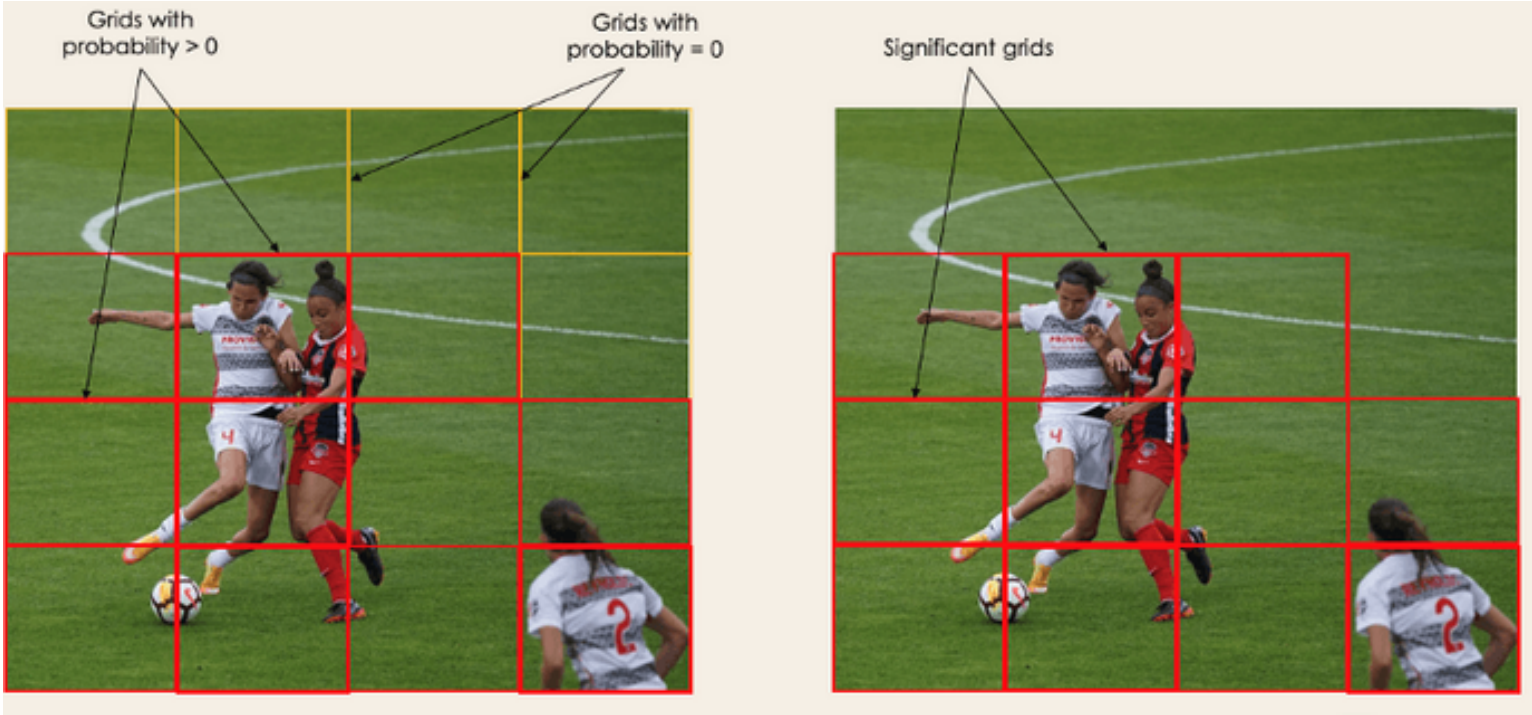
THE YOLO ALGORITHM

YOLO uses a single algorithm to detect an object. It stands for YOU ONLY LOOK ONCE (As only a single forward propagation is required for object detection). It is a regression problem , giving the class probabilities of the detected objects. The CNN used produces the class probabilities and the bounding boxes simultaneously. It works in the following ways.

@ RESIDUAL BLOCKS (sxs)



@ Bounding Box Regression



@ Intersection Over Union

Most of the time, a single object in an image can have multiple grid box candidates for prediction, even though not all of them are relevant. The goal of the IOU (a value between 0 and 1) is to discard such grid boxes to only keep those that are relevant.

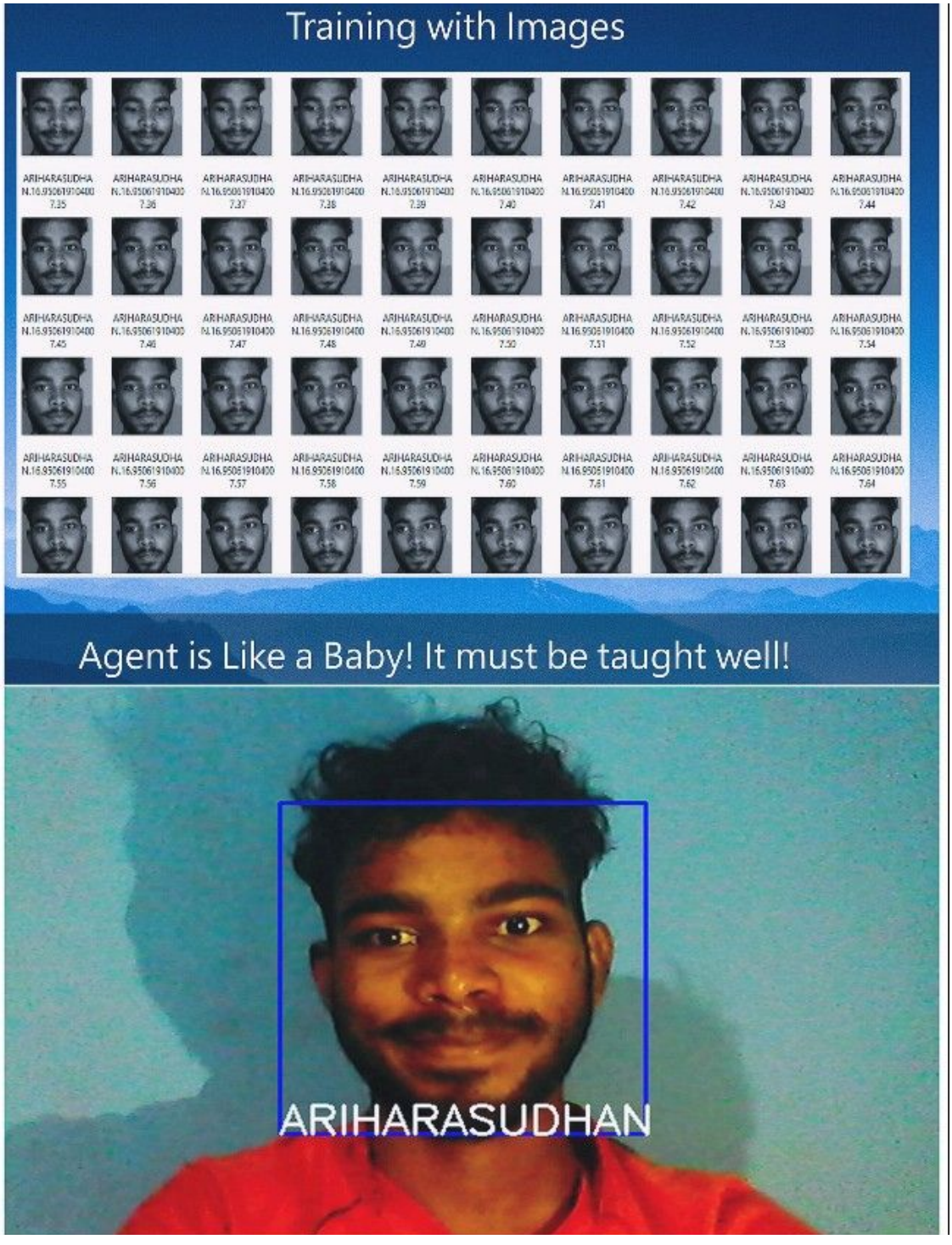
REGION PROPOSAL ALGORITHMS

These region proposals can be noisy, overlapping and may not contain the object perfectly but among these region proposals, there will be a proposal which will be very close to the actual object in the image. We can then classify these proposals using the object recognition model. The region proposals with the high probability scores are locations of the object. Region proposal algorithms identify prospective objects in an image using segmentation. In segmentation, we group adjacent regions which are similar to each other based on some criteria such as color, texture etc. Unlike the sliding window approach where we are looking for the object at all pixel locations and at all scales, region proposal algorithm work by grouping pixels into a smaller number of segments.

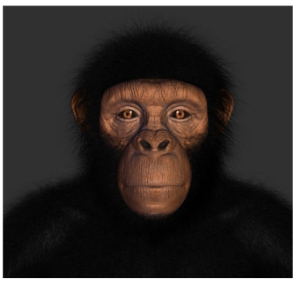
So the final number of proposals generated are many times less than sliding window approach. This reduces the number of image patches we have to classify. These generated region proposals are of different scales and aspect ratios. An important property of a region proposal method is to have a very high recall. This is just a fancy way of saying that the regions that contain the objects we are looking have to be in our list of region proposals. Generally, the following steps are followed :
(1) Add all bounding boxes corresponding to segmented parts to the list of regional proposals
(2) Group adjacent segments based on similarity
(3) Go to step 1

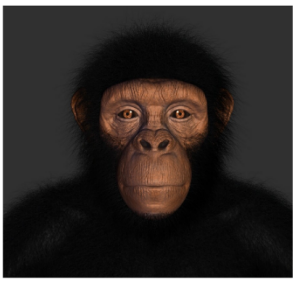
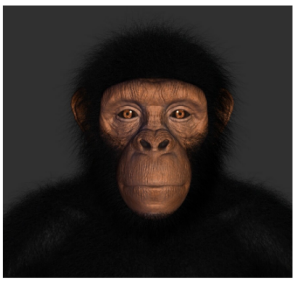
ONE SHOT LEARNING

In most of the real world systems of face detection, one-shot learning is used. But, is it possible to learn from just one shot ? Ahh.. I mean from just one training example ?



Let’s consider an organization which does face recognition for authentication. Should we re-train the model when a new employee is added ? No… It’s so worse. So, how to proceed? There is a simple technique called, Degree of Difference. What we have to do is, we have to compute the difference between two images based on which an intended person is authenticated. The idea is as shown below :

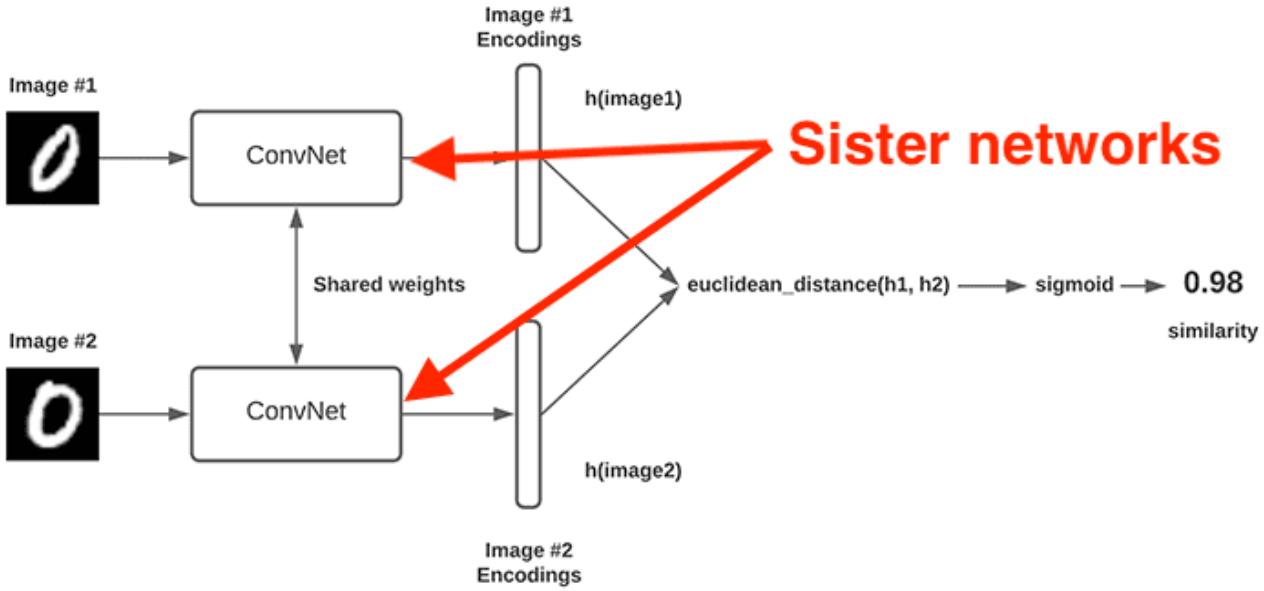
 -  = D > τ

 -  = D <= τ

τ is a hyper-parameter. Whenever the difference is small, it will be less than or equal to τ.

SIAMESE NEURAL NETWORK

It helps to build models with good accuracy even with fewer samples per class and imbalanced class distribution.



A Siamese network is a class of neural networks that contains one or more identical networks. We feed a pair of inputs to these networks. Each network computes the features of one input. And, then the similarity of features is computed using their difference or the dot product.

For same class input pairs, target output is 1 and for different classes input pairs, the output is 0.

THE TRIPLET LOSS

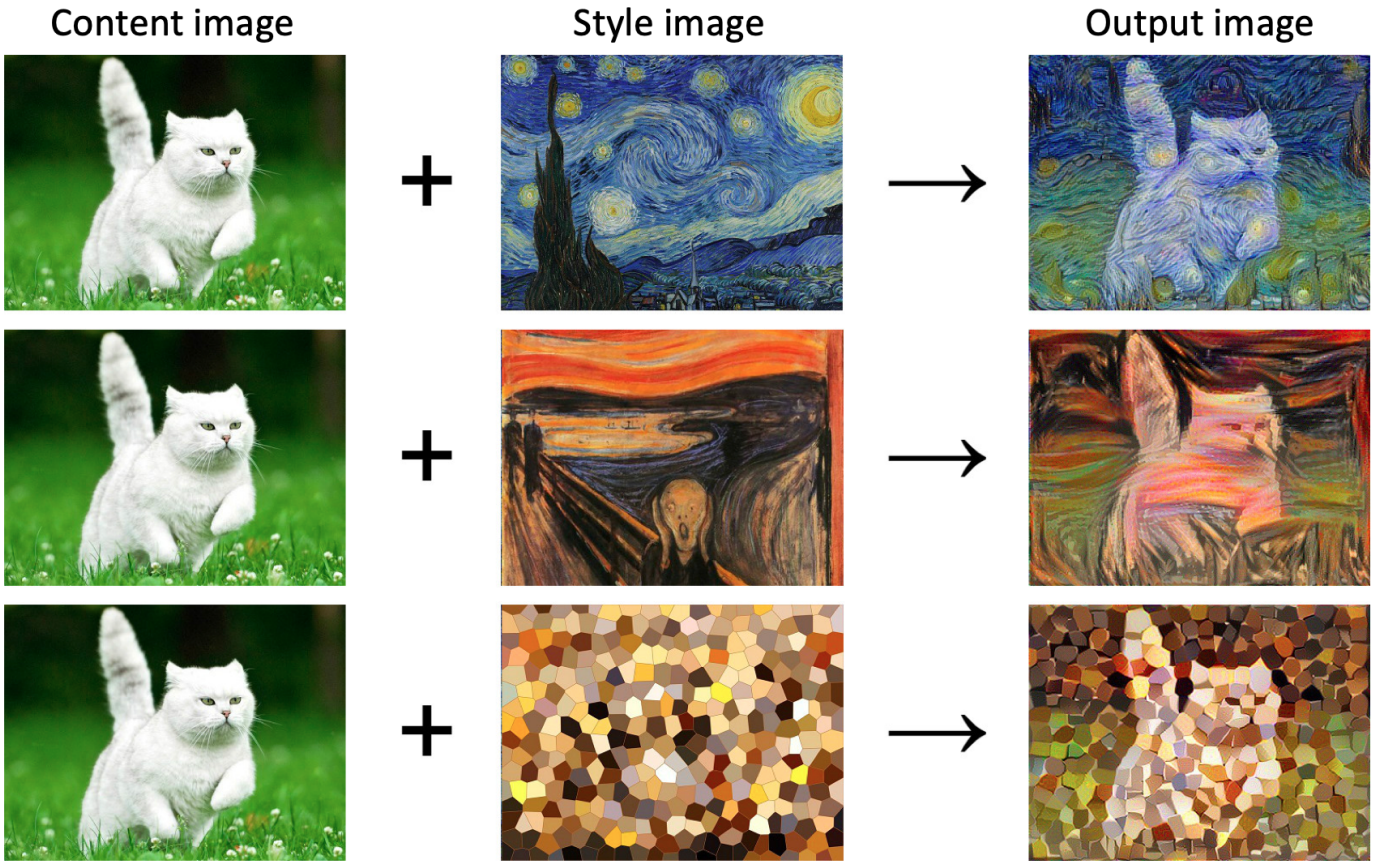
Triplet loss is a loss function for machine learning algorithms where a reference input (called anchor) is compared to a matching input (called positive) and a non-matching input (called negative).

$$\mathcal{L}\left(A, P, N\right) = \max(\| \mathrm{f}(A) - \mathrm{f}(P) \|_2 - \| \mathrm{f}(A) - \mathrm{f}(N) \|_2 + \alpha, 0)$$

This can then be used in a cost function, that is the sum of all losses :

$$\mathcal{J} = \sum_{i=1}^{M} \mathcal{L}\left(A^{(i)}, P^{(i)}, N^{(i)}\right)$$

NEURAL STYLE TRANSFER



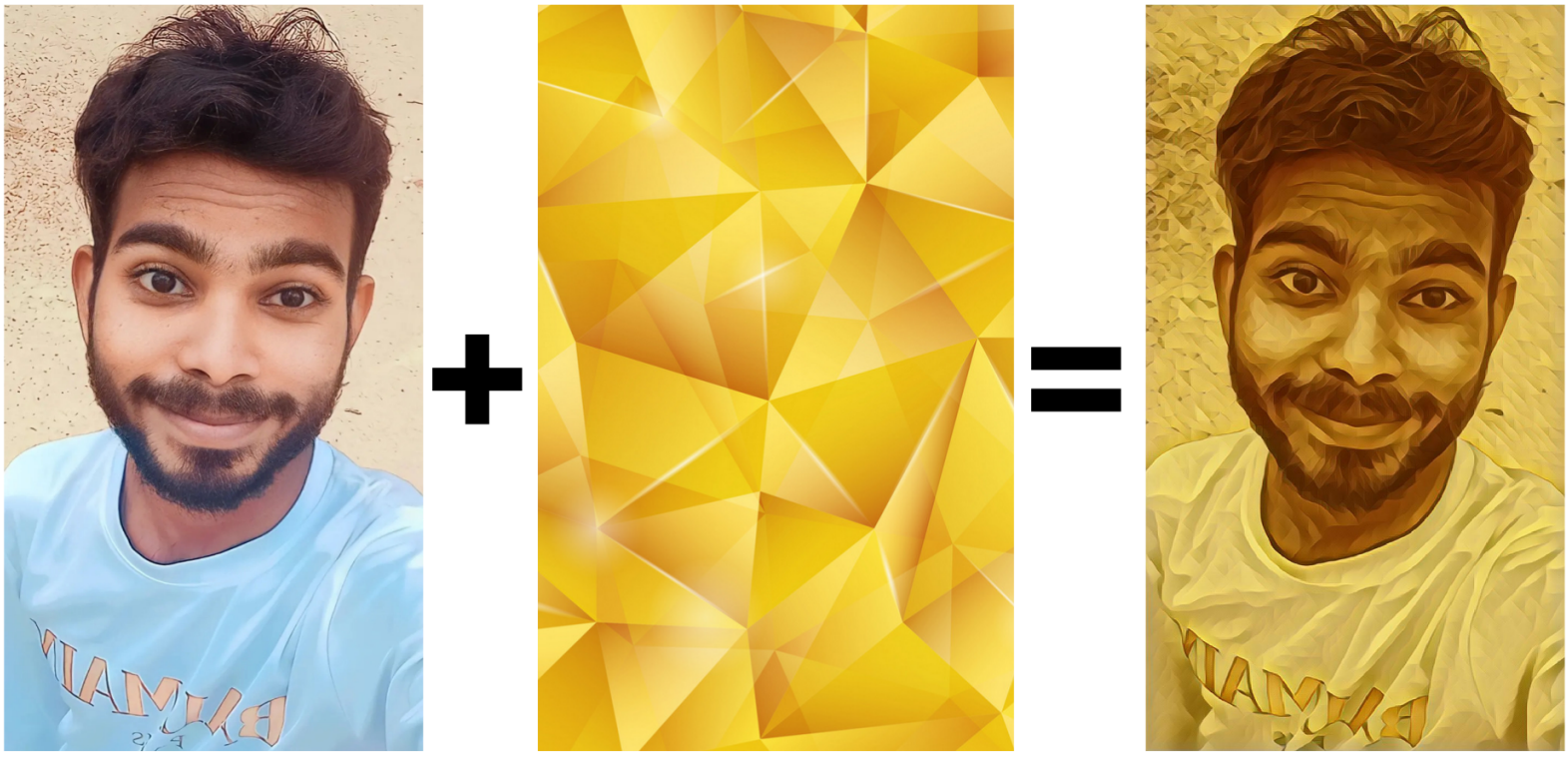
It is a trend in CNN to create such stylish images. It transfers the style from one image to another. The Neural Style Transfer Cost Function can be defined as,

$$J(G) = \alpha J_{content}(C,G) + \beta J_{style}(S,G)$$

where,

C - Content Image
S - Style Image
G - Generated Image

Cost Function measures similarity of the given parameters.



So, how to create such one ?
(1) Initiate the Generated Image Randomly [E.g: 100x100x3]
(2) Use Gradient Descent and loop

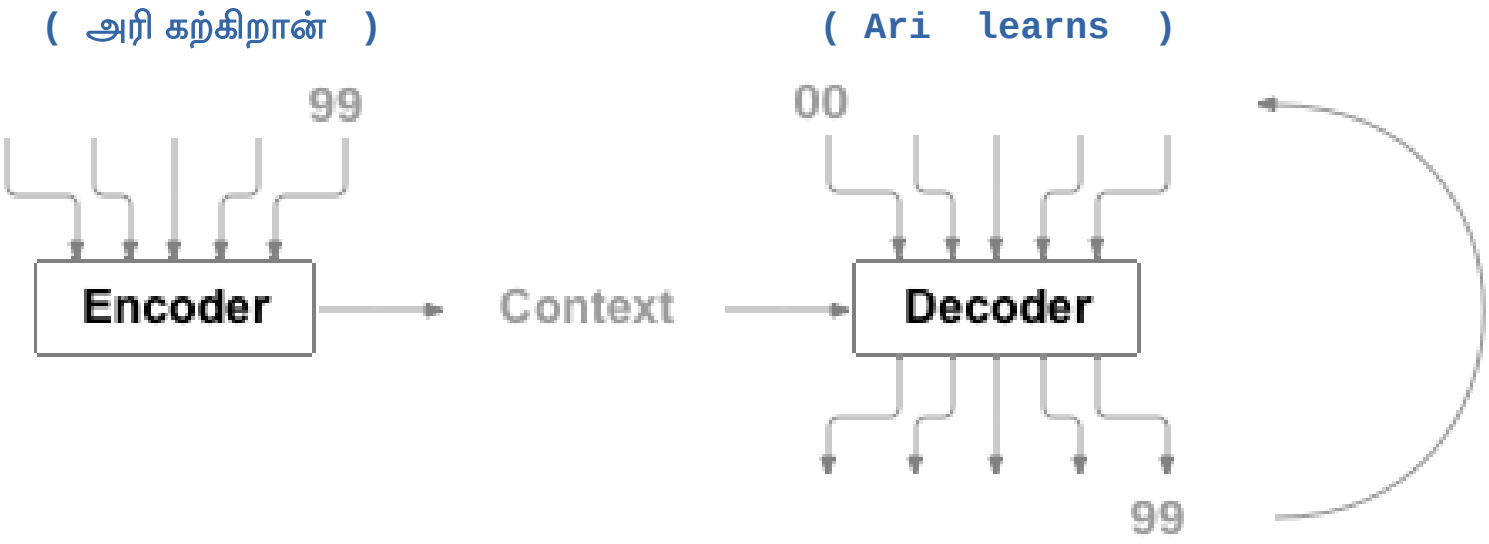
SEQUENCE TO SEQUENCE MODELS

Let’s say we want to translate a Tamizh sentence into English as following.

அரி கற்கிறான் - Ari learns

Tamizh (Input)		English (Output)	
அரி	$x^{<1>}$	Ari	$Y^{<1>}$
கற்கிறான்	$x^{<2>}$	learns	$Y^{<2>}$

So we want to map X to Y. Sequence models are the **machine learning models that input or output sequences of data**. Sequential data includes text streams, audio clips, video clips, time-series data and etc. Recurrent Neural Networks (RNNs) is a popular algorithm used in sequence models.



A Recurrent Neural Network, or RNN, is a network that operates on a sequence and uses its own output as input for subsequent steps.

A Sequence to Sequence network, or seq2seq network, or Encoder Decoder network, is a model consisting of two RNNs called the encoder and decoder. The encoder reads an input sequence and outputs a single vector, and the decoder reads that vector to produce an output sequence.

BLEU SCORE

BLEU stands for Bilingual evaluation Understudy. It is a metric used to evaluate the quality of machine generated text by comparing it with a reference text that is supposed to be generated. Usually, the reference text is generated by a manual evaluator or a translator.

The basic idea involves computing the precision – which is the fraction of candidate words in reference.

$$P = \frac{m}{w_t}$$

$m = 4$: Number of candidate words in reference

$w_t = 4$: Total number of words in candidate

Clipping the count:

For each word in the candidate, clip the count to the maximum occurances in the reference sentence.

BLEU Score with n-grams:

The above computation takes into account individual words or unigrams of candidate that occur in target. However, for a more accurate evaluation of the match, one could compute bi-grams or even tri-grams and average the

score obtained from various n-grams to compute the overall BLEU score. For instance consider the following example.

Candidate sentence: The the cat

Reference Sentence: The cat is on the mat

Comparing metrics for candidate "the the cat"

Model	Set of grams	Score
Unigram	"the", "the", "cat"	$\frac{1+1+1}{3}=1$
Bigram	"the the", "the cat"	$\frac{0+1}{2}=\frac{1}{2}$

To compute the *BLEU score for the entire corpus,* one can compute the BLEU score for individual candidates with their references for each query sentence in the dataset and take an average.

ATTENTION MECHANISM

A sequence to sequence model has two components, an **encoder** and a **decoder**. The encoder encodes a source sentence to a concise vector (called the **context vector**), where the decoder takes in the context vector as an input and computes the translation using the encoded representation.

A problem with these models are that, performance decays as the length of the input sentence increases. The reason being —

- The words to be predicted depends on the context of the input sentence and not on the single word. So, basically to predict a word in French, we might make use of 2–3 words in the English sentence. That is

how humans translate one language to another.

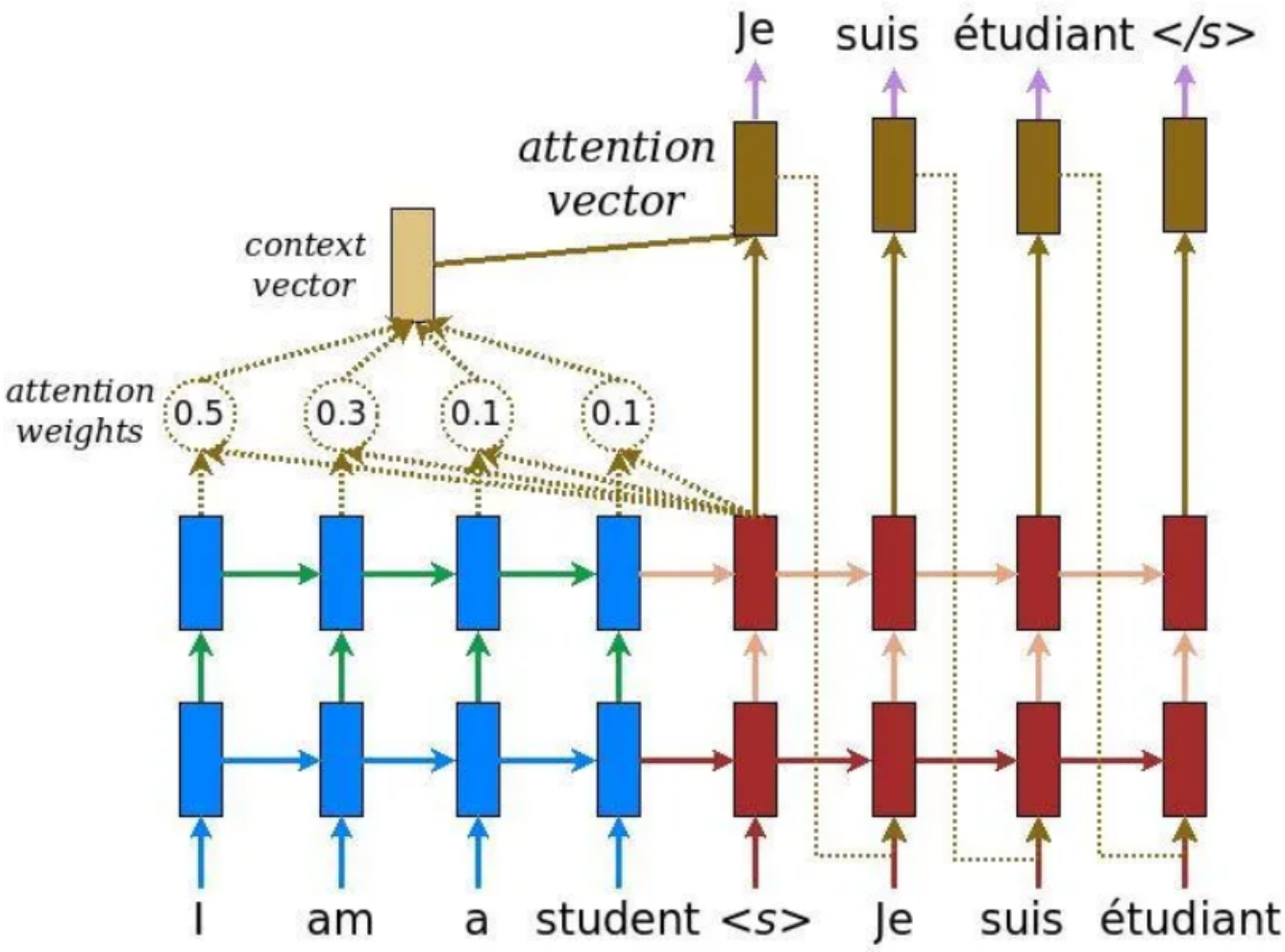
- Another limitation is that with longer sentences, we have to compress all the information of the input sentence into a fixed length vector. Not all words in the sentence are important to predict the correct word.

Now, with the length of the input sentence increasing, over time our LSTM/GRU loses the context of the long sentence thereby losing the meaning of the whole sentence and eventually resulting in poor performance. The whole idea of attention is instead of relying just on the context vector, the decoder can have access to the past states of the encoder.

At each decoding step, the decoder gets to look at any particular state of the encoder. Attention mechanism tries to identify which parts of the input sequence are relevant to each word in the output and uses the **relevant information** to select the appropriate output. In order to tackle the limitation, we use the weighted sum of selected number of past encoded states. We have two constraints:

1.Number of past states necessary
2.Weight for the selected past states

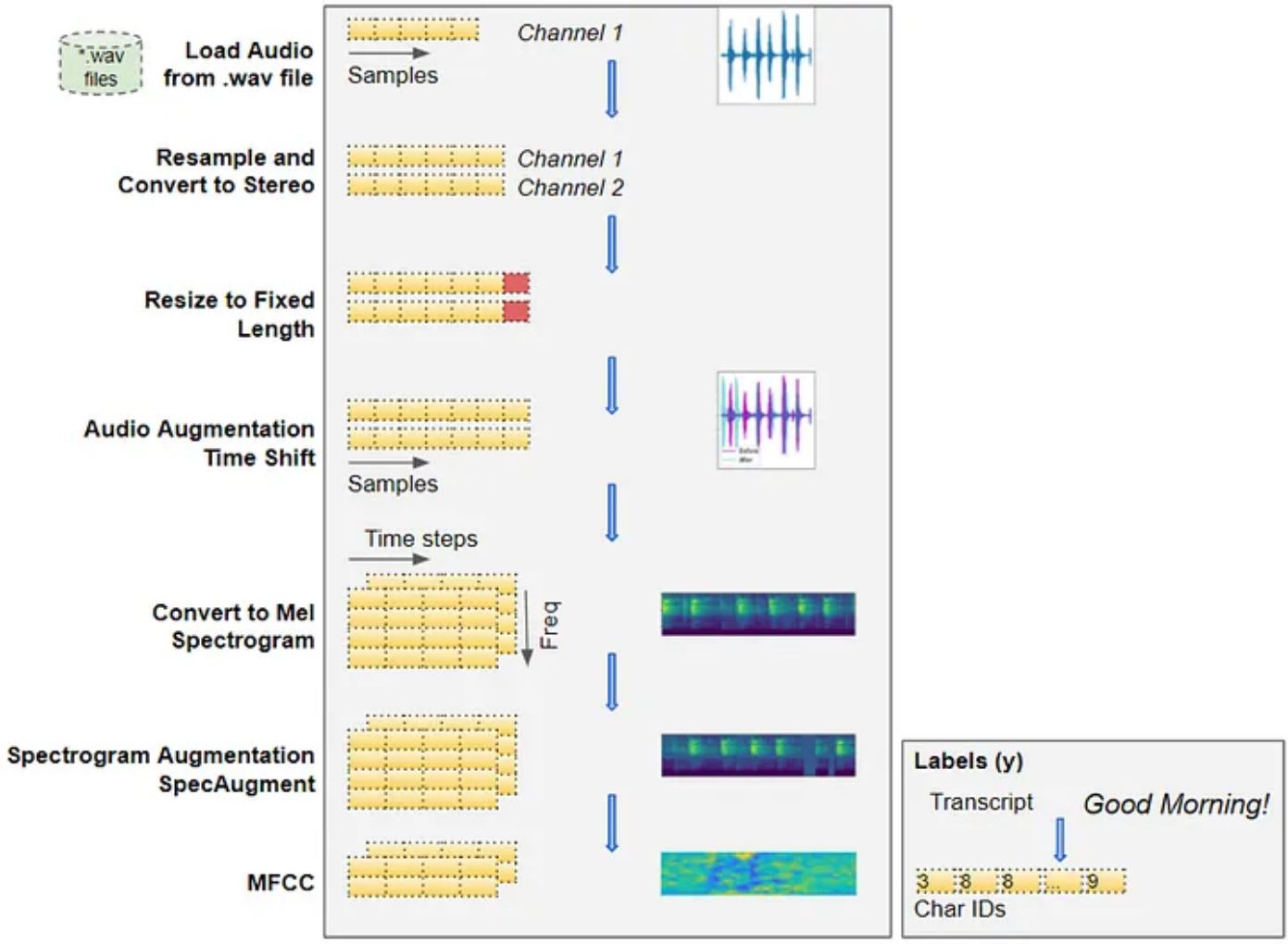
Since these constraints can be learned with back-prop, we can assume this to be as a layer which fits between the encoder and decoder.



SPEECH RECOGNITION

Over the last few years, Voice Assistants have become ubiquitous with the popularity of Google Home, Amazon Echo, Siri, Cortana, and others. These are the most well-known examples of Automatic Speech Recognition (ASR).

STEPS FOLLOWED



Load Audio Files

- Start with input data that consists of audio files of the spoken speech in an audio format such as “.wav” or “.mp3”.

- Read the audio data from the file and load it into a 2D Numpy array. This array consists of a sequence of numbers, each representing a measurement of the intensity or amplitude of the sound at a particular moment in time. The number of such measurements is determined by the sampling rate. For instance, if the sampling rate was 44.1kHz, the Numpy array will have a single row of 44,100 numbers for 1 second of audio.
- Audio can have one or two channels, known as mono or stereo, in common parlance. With two-channel audio, we would have another similar sequence of amplitude numbers for the second channel.

Convert to uniform dimensions: sample rate, channels, and duration

- We might have a lot of variation in our audio data items. Clips might be sampled at different rates, or have a different number of channels. The clips will most likely have different durations. As explained above this means that the dimensions of each audio item will be different.
- Since our deep learning models expect all our input items to have a similar size, we now perform some data cleaning steps to standardize the dimensions of our audio data. We resample the audio so that every item has the same sampling rate.

- We convert all items to the same number of channels. All items also have to be converted to the same audio duration. This involves padding the shorter sequences or truncating the longer sequences.
- If the quality of the audio was poor, we might enhance it by applying a noise-removal algorithm to eliminate background noise so that we can focus on the spoken audio.

Data Augmentation of raw audio

- We could apply some data augmentation techniques to add more variety to our input data and help the model learn to generalize to a wider range of inputs.

Spectrograms

- This raw audio is now converted to Spectrograms. A Spectrogram captures the nature of the audio as an image by decomposing it into the set of frequencies that are included in it.

MFCC

- For human speech, in particular, it sometimes helps to take one additional step and convert the Mel Spectrogram into MFCC (Mel Frequency Cepstral Coefficients). MFCCs produce a compressed representation of the Mel Spectrogram by extracting only the most essential frequency coefficients, which correspond to the frequency ranges at which humans speak.

Data Augmentation of Spectrograms

- We can now apply another data augmentation step on the Mel Spectrogram images, using a technique known as SpecAugment. This involves Frequency and Time Masking that randomly masks out either vertical (ie. Time Mask) or horizontal (ie. Frequency Mask) bands of information from the Spectrogram.

We have now transformed our original raw audio file into Mel Spectrogram (or MFCC) images after data cleaning and augmentation. We also need to prepare the target labels from the transcript.

This is simply regular text consisting of sentences of words, so we build a vocabulary from each character in the transcript and convert them into character Ids. This gives us our input features and our target labels. This data is ready to be input into our deep learning model.

THANK YOU ANDREW NG