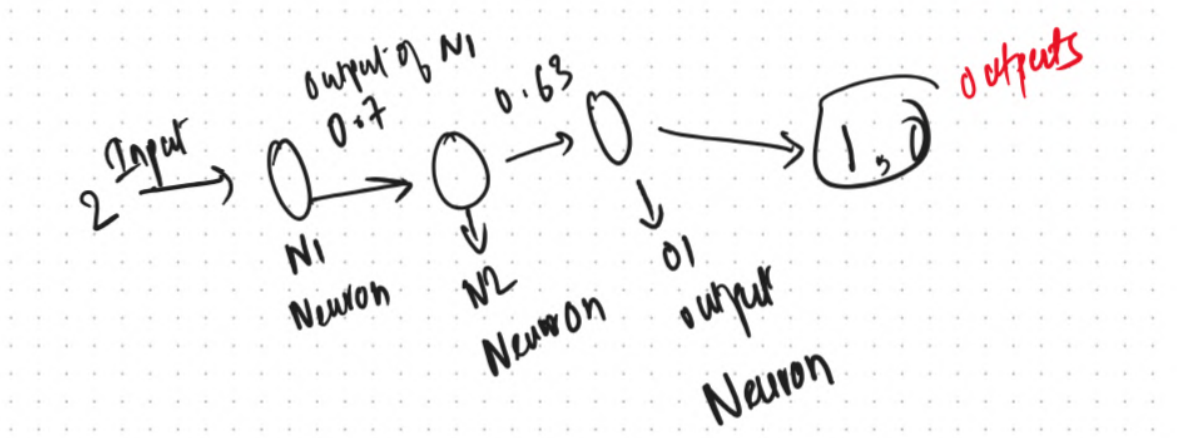
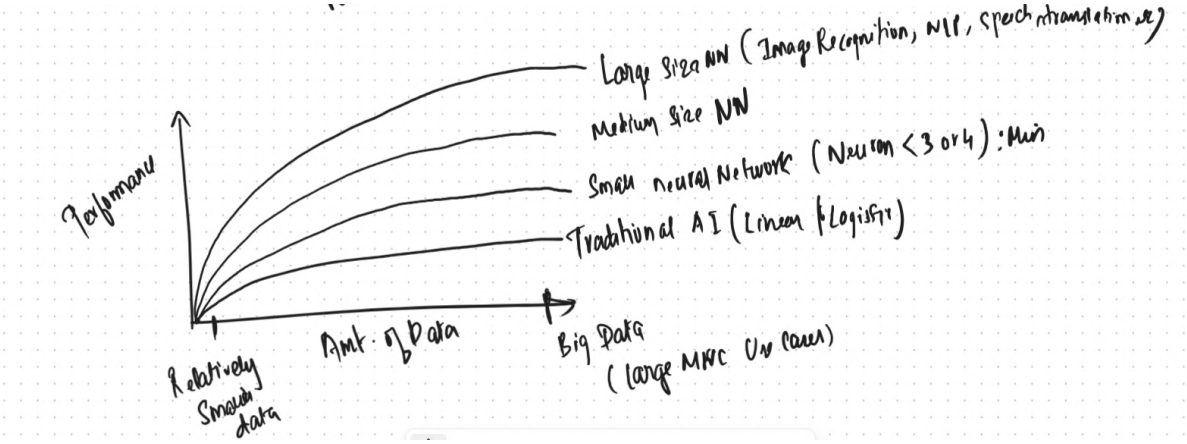
Advanced Learning Algorithms by Andrew NG

Author	Andrew NG
Link	https://www.coursera.org/learn/advanced-learning-algorithms/lecture/3wO3h/welcome
Score	⭐⭐⭐⭐
Status	In progress

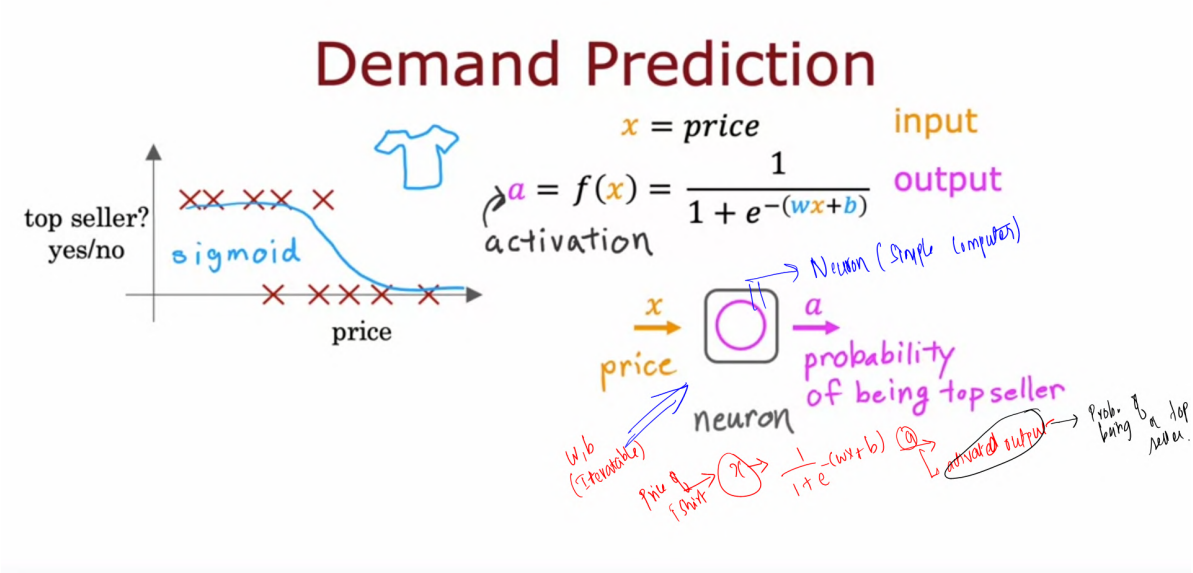
▼ Neural Network Intuition

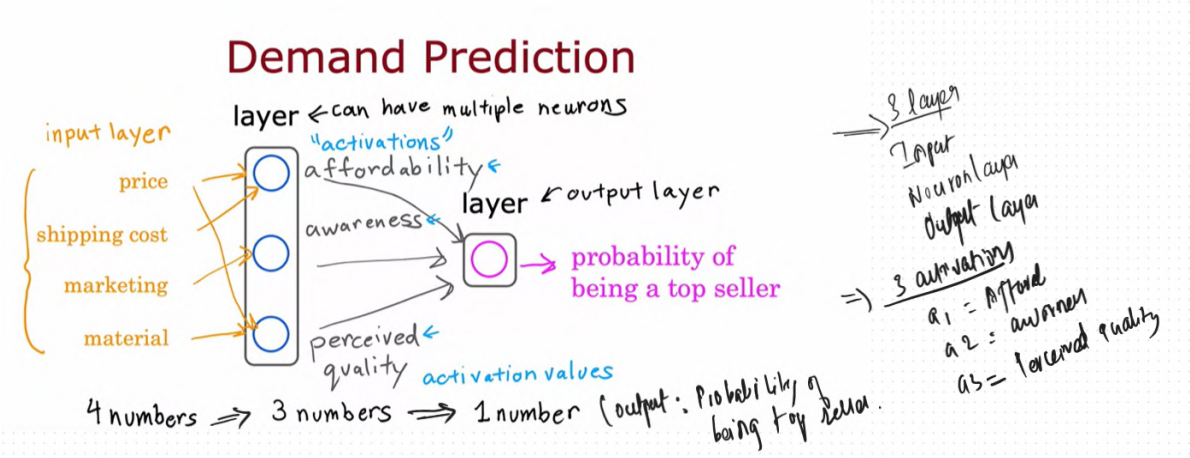
- Neural network was used to mimic the human brain by utilizing the software.
- the inputs are sent to neuron within which some calculations happens(functions and weights updates) post which we receive an output. This output then becomes an input for another neuron. Recent times we have been implying multiple neurons for our application. Though the neurons represent the surface level activities of the brain, it quiet effective for modern day application
-



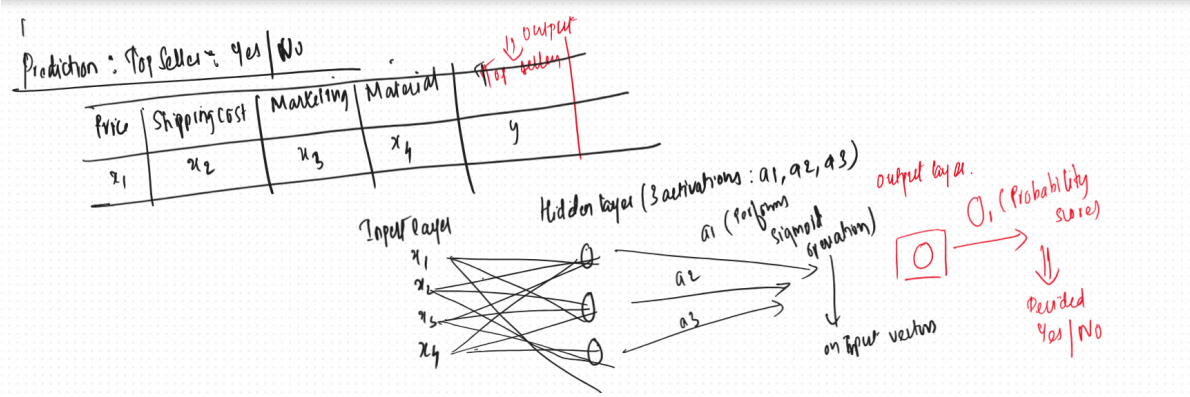


▼ Demand Prediction

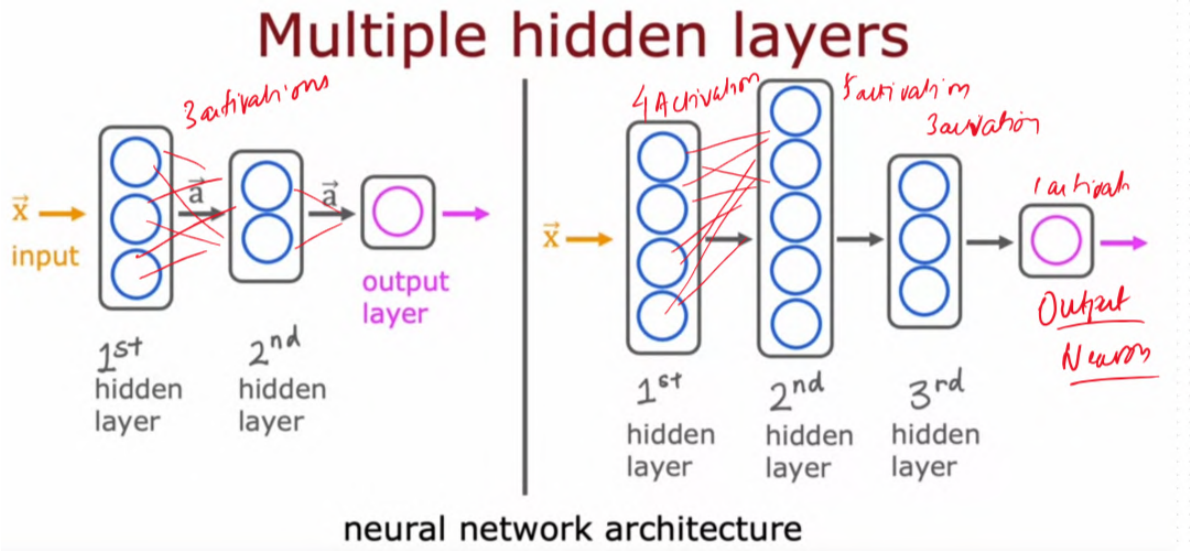




- In above diagram, we dont necessary choose what input should be given to the neural layer as it becomes cumbersome process if there are many variables (say 100 variables), So we will connect all the input variables to all the neurons with neural layer.
- Each layer takes input vectors and gives output vectors
- Neural layers are called “**Hidden layers**”.
- Earlier it was manual feature engineering, however here the neural layer performs feature engineering. Audacious thing here is that the output is not directly derived from input like linear or logistic regression, it is derived from hidden layer of 3 neurons.

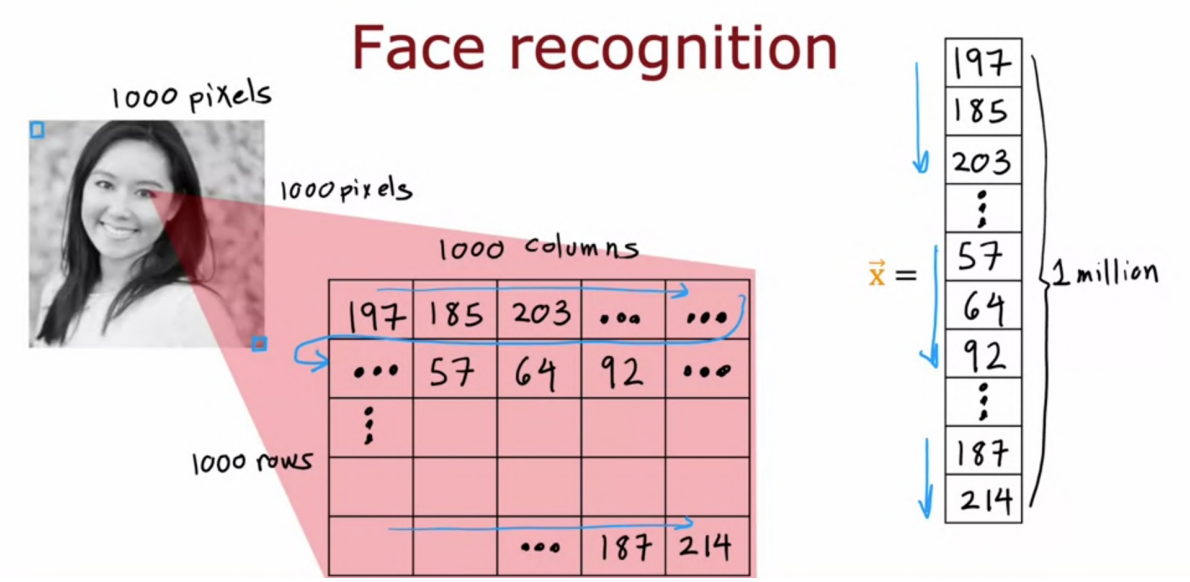


- No feature engineering is required as hidden layer takes care of it. Manual feature selection might not ensure high quality of inputs to model, however hidden layer takes care of it.
- While handling the neural network, we should be able to decide “how many layers of neural network is needed” and “how many neurons per layer
    - Answer is : neural network archietecture

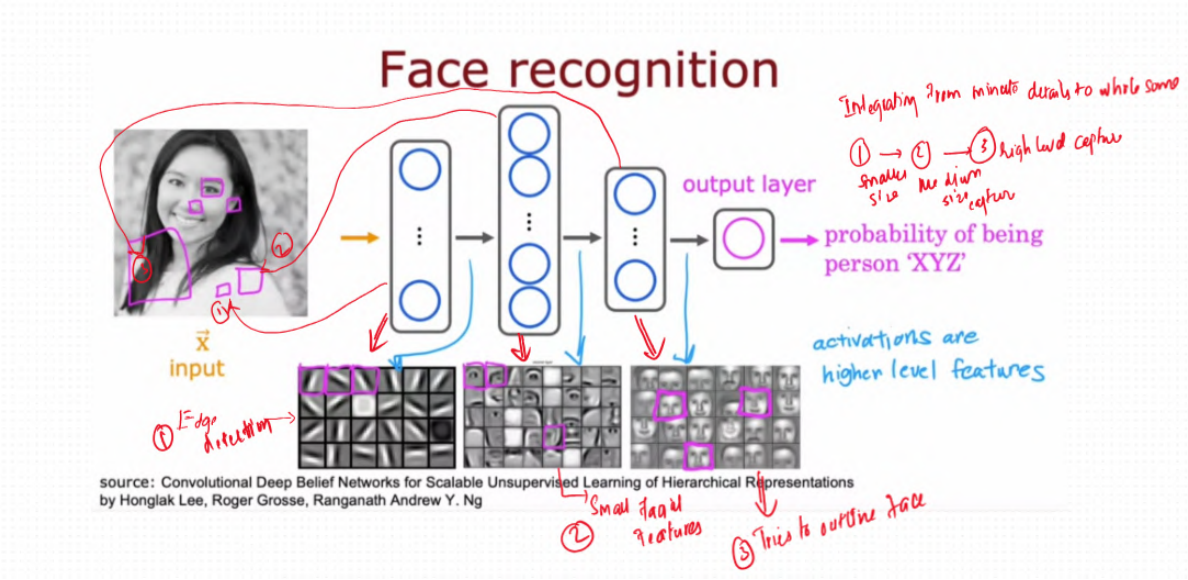


▼ Example : Face Recognition

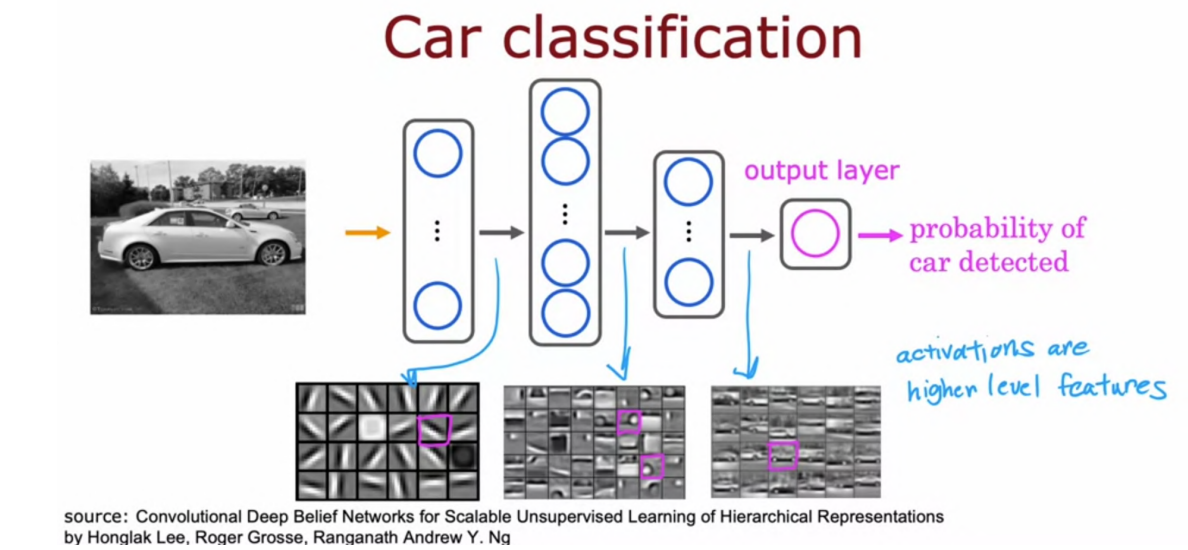
- the images are recorded in the form of matrix. We know the image size. So Length * breadth of image = Number of pixels
- Below 1000 rows * 1000 colum pixel = 1M pixels
- the intensity of light are captured, within this 1M pixels then they are unpacked to vectors in an array as stated below, which becomes input for hidden layers.



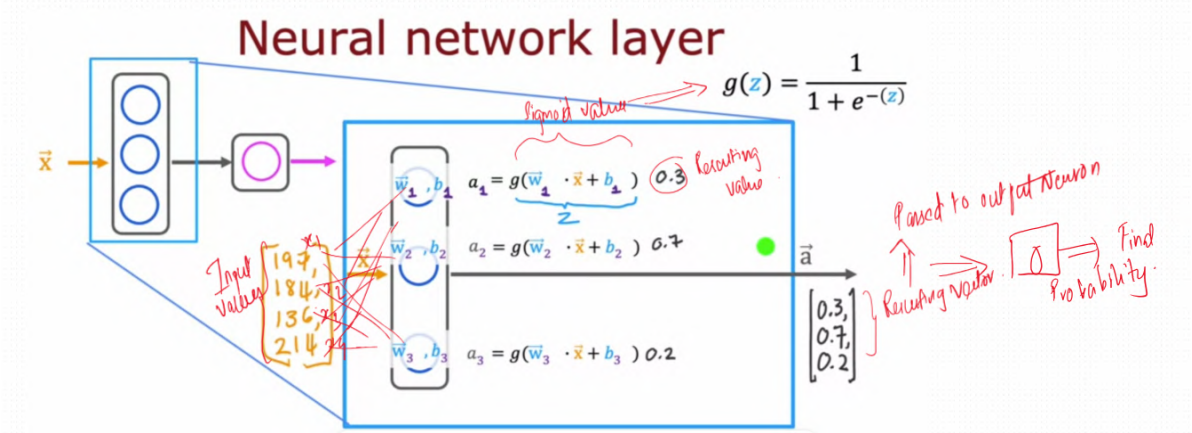
- In the below image, the first hidden layer is being visualized here we can find it tries to identify the edge of the image first.
- The beauty is that no one said or programmed to detect the edge first then smaller parts then whole face outline. It is done on its own from data.



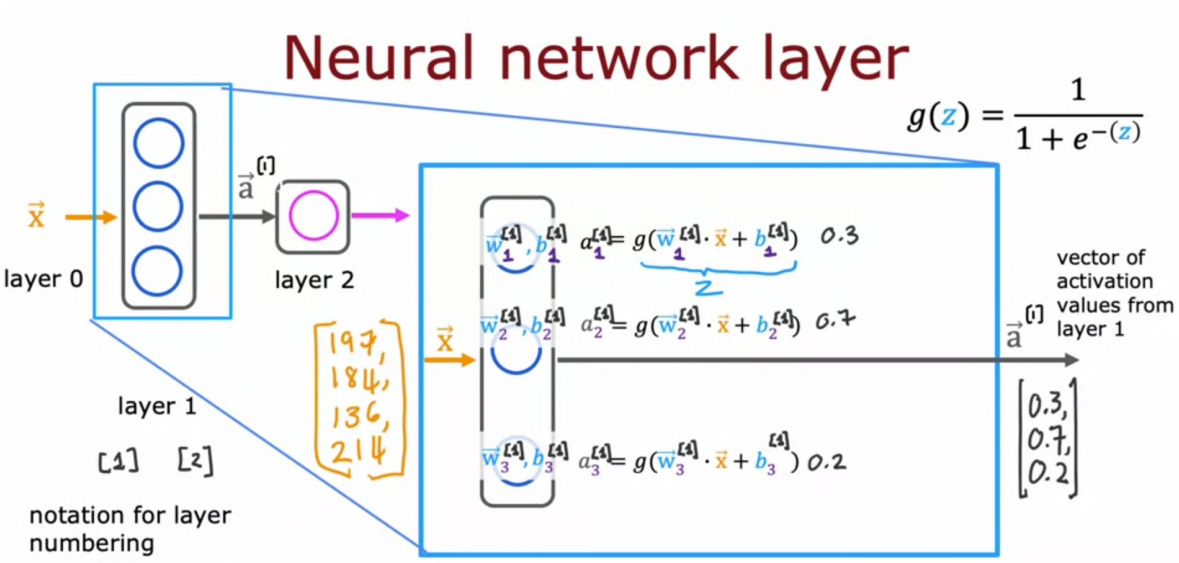
- Simply, changing the input data alone, the hidden layer learns on its own , the edges, smaller parts and finally a car. Here we are adding no extra hidden layer or extra neurons. The same model is applied to car as well.

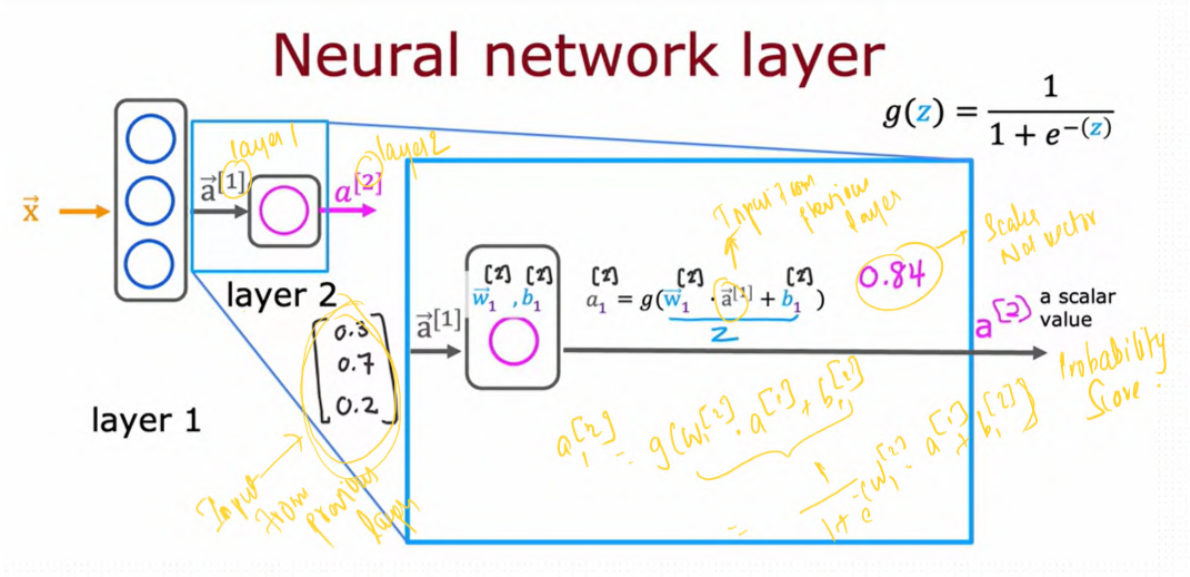


▼ Simple Neural Network Layer



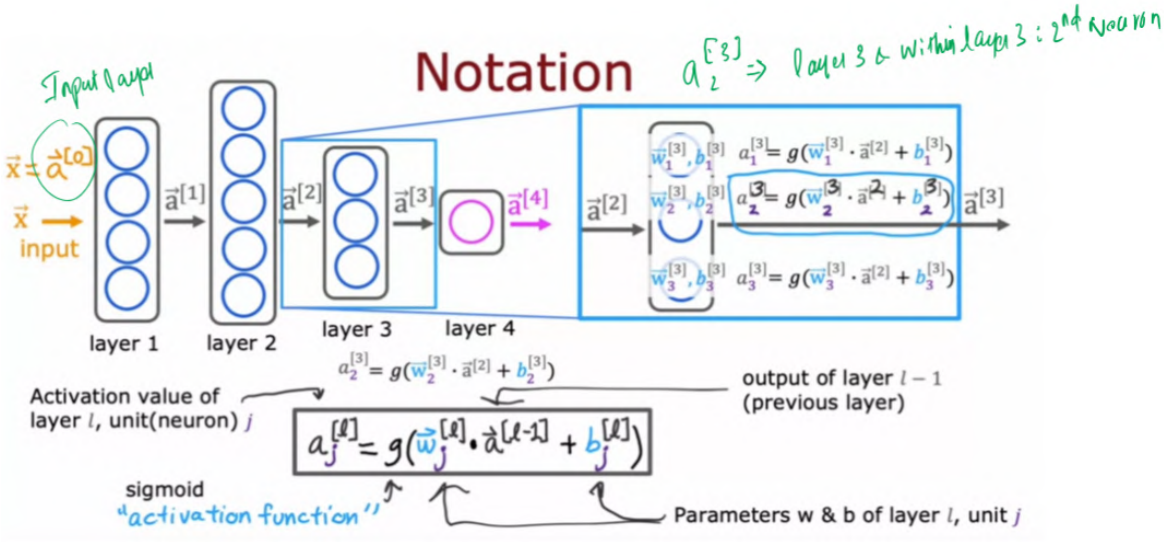
here the **[1]** denotes the nth hidden layer

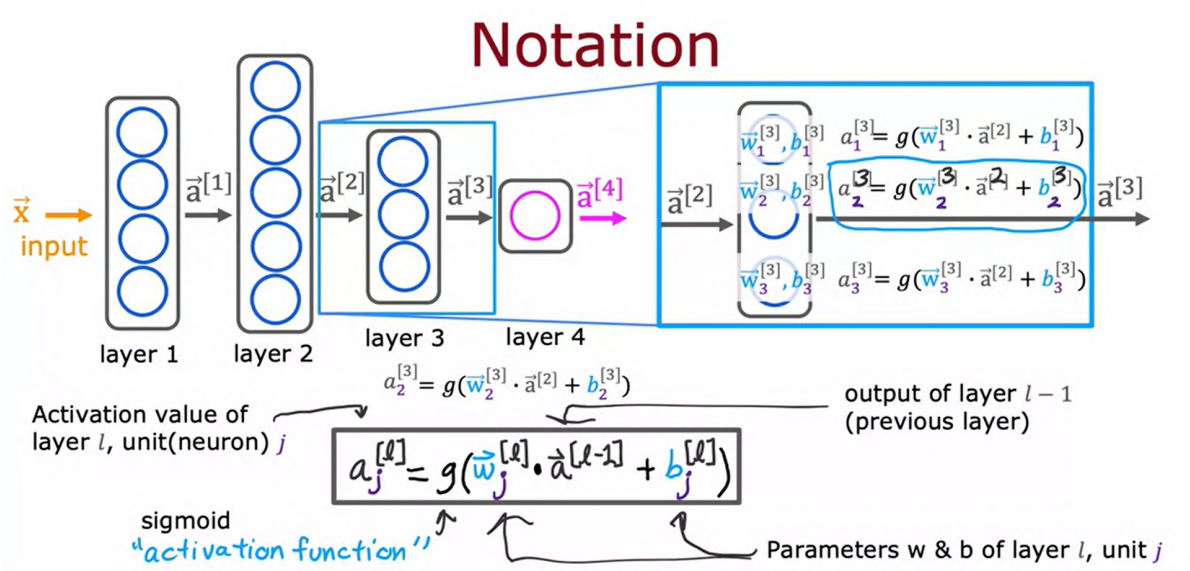




▼ Complex Neural Network layer

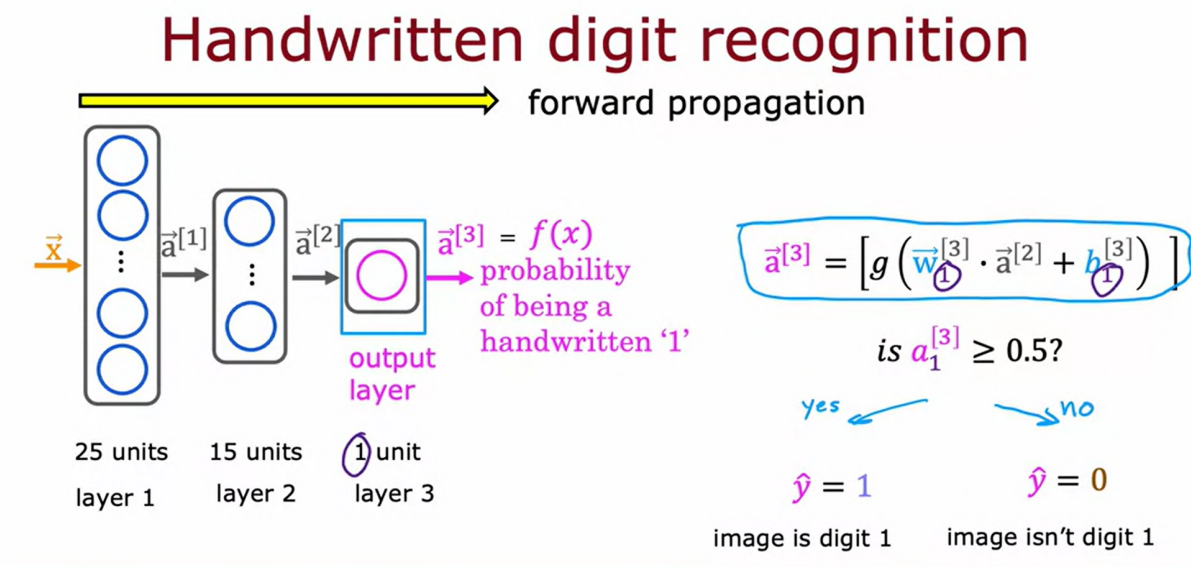
-





▼ Foreword propagation

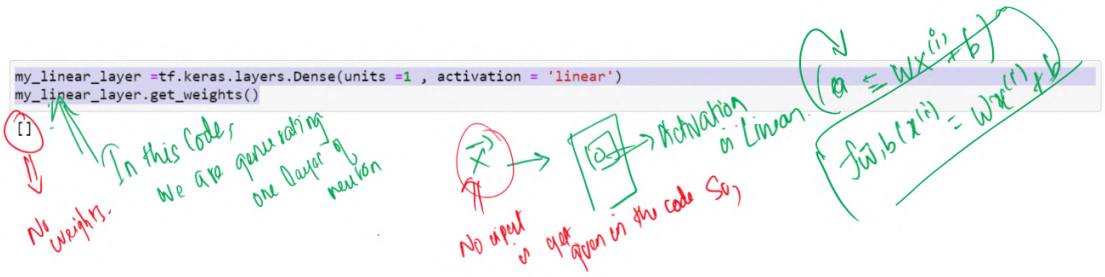
- Here the input layer is taken followed by layer1 then layer 2 ..... final output layer : which results in scaler, then we may use threshold to perform classification or we can use that prediction for regression. This sequential activity from left to right as stated below is called forward propagation. We can download the weights and neural network used for different application from the internet and use in our application.



- This is quiet fascinating that one model generated by larger MNC for a specific purpose can be used for different application by allowing it train

on different dataset.

- Tensorflow is a framework developed by google , keras(framework, a layer centric interface) is developed by an individual which later combined with tensorflow.
- **A Neuron without activation(sigmoid, reLu etc.) is just “a Linear/ Regression Model”**
- **A tensor is another name of array where the shape is in format of (1,1)**
- The intializing of weights is random.



- Input to linear layer

```
a1 = linear_layer(X_train[0].reshape(1,1))
print(a1)
```

```
tf.Tensor([[-1.03]], shape=(1, 1), dtype=float32)
```

- Get weights and set weights.

```
w, b= linear_layer.get_weights()
print(f"w = {w}, b={b}")
```

```
w = [[-1.03]], b=[0.]
```

- For setting the weights in linear_layer.

```
set_w = np.array([[200]])
set_b = np.array([100])

# set_weights takes a list of numpy arrays
linear_layer.set_weights([set_w, set_b])
print(linear_layer.get_weights())
```

```
[array([[200.]], dtype=float32), array([100.], dtype=float32)]
```

- 1000 pixels * 1000 pixels = 1 Million pixels this pixels is reshape to 2 dimensional array by applying **.reshape(-1,1)** which means the output will be **array[1Million,1].**

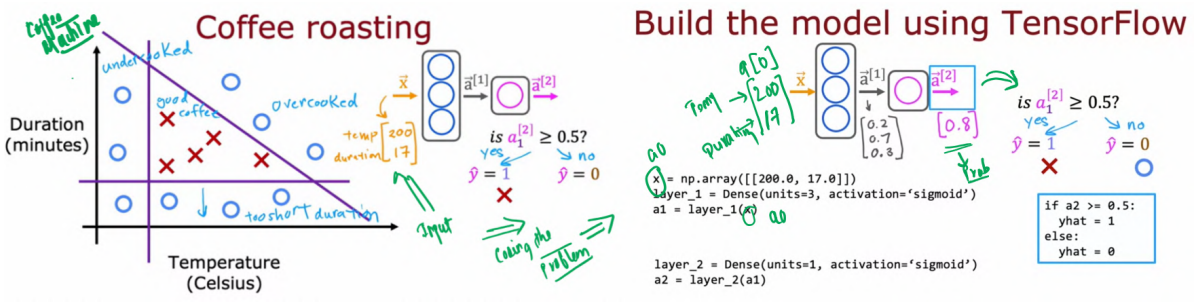
```
print(np.array([0., 1, 2, 3, 4, 5], dtype=np.float32))
print(np.array([0., 1, 2, 3, 4, 5], dtype=np.float32).reshape(-1,1))
```

```
[0. 1. 2. 3. 4. 5.]
[[0.]
 [1.]
 [2.]
 [3.]
 [4.]
 [5.]]
```

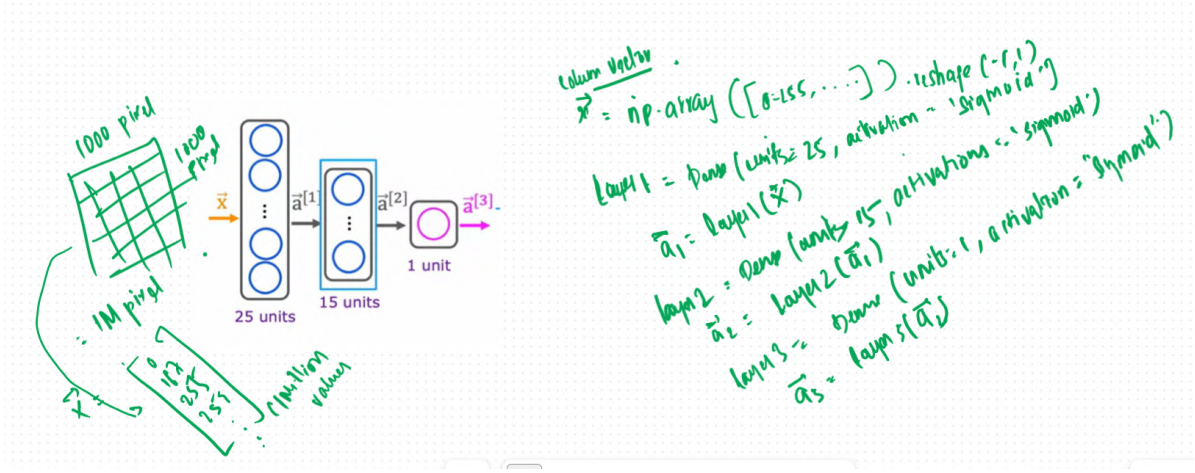
- Keras are generally very useful in creating multilayer models , the sequential function helps in achieving the same.

▼ How to implement tensor flow

- Tensorflow is tool to implement the deep learning projects.
- Other popular tool is pytorch.
- **Dense :** A type of neural layer or hidden layer.

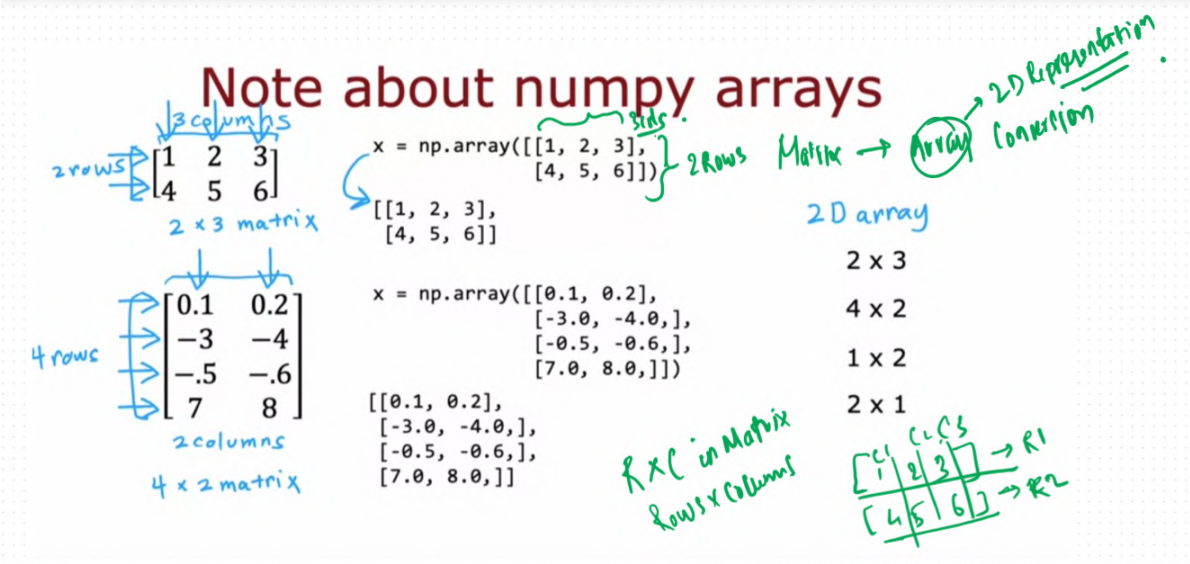


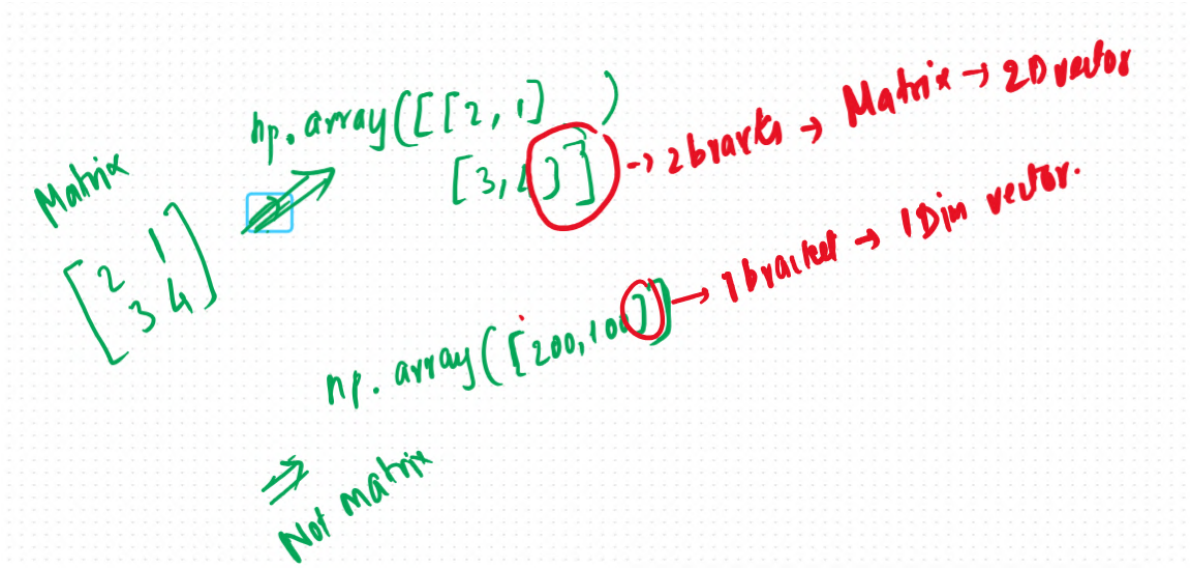
- Simple implementation of tensor flow.



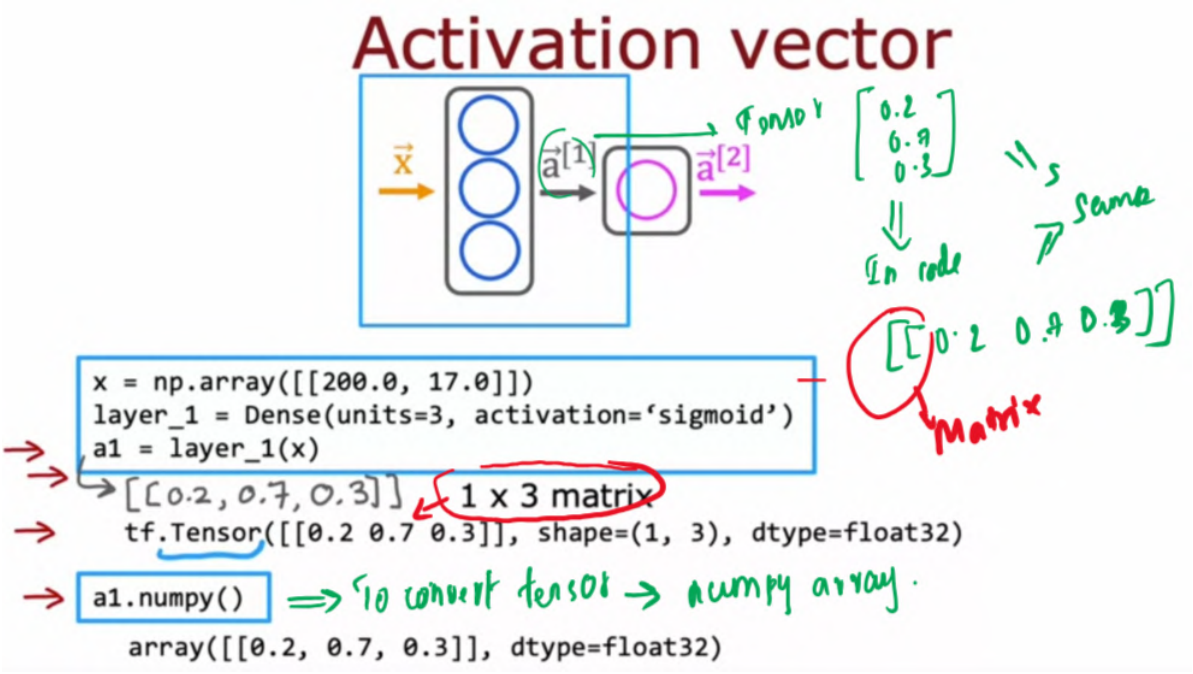
▼ How TensorFlow understands data

- Numpy was initial form of library which is standard for implementing the linear and logistic regressions
- Google brain created the tensor flow, so the contradiction could occur between numpy and tensorflow. Tensor is a datatype created by google brain team to carry out the function and data manipulation & handling. Tensor is a matrix a 2-Dim vector.
- Matrix → numpy array





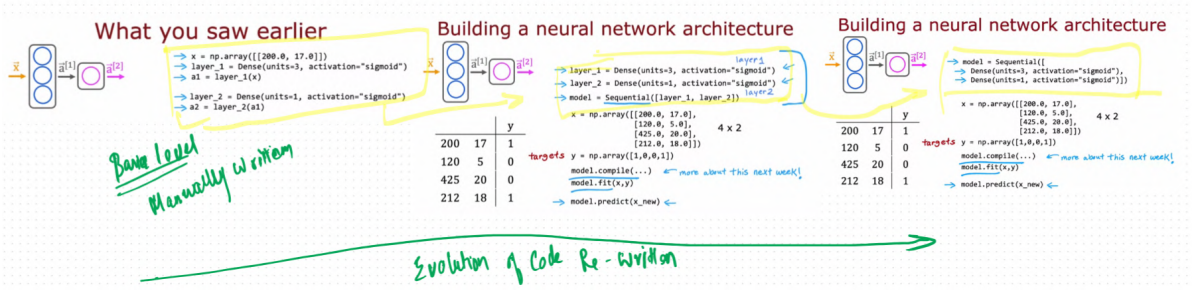
- **Linear regression and logistic regression could handle the 1-Dim vector**
- **However for tensorflow the conventions are using the matrices which are 2-D vectors to represent the data. So this is computationally efficient in handling data.**



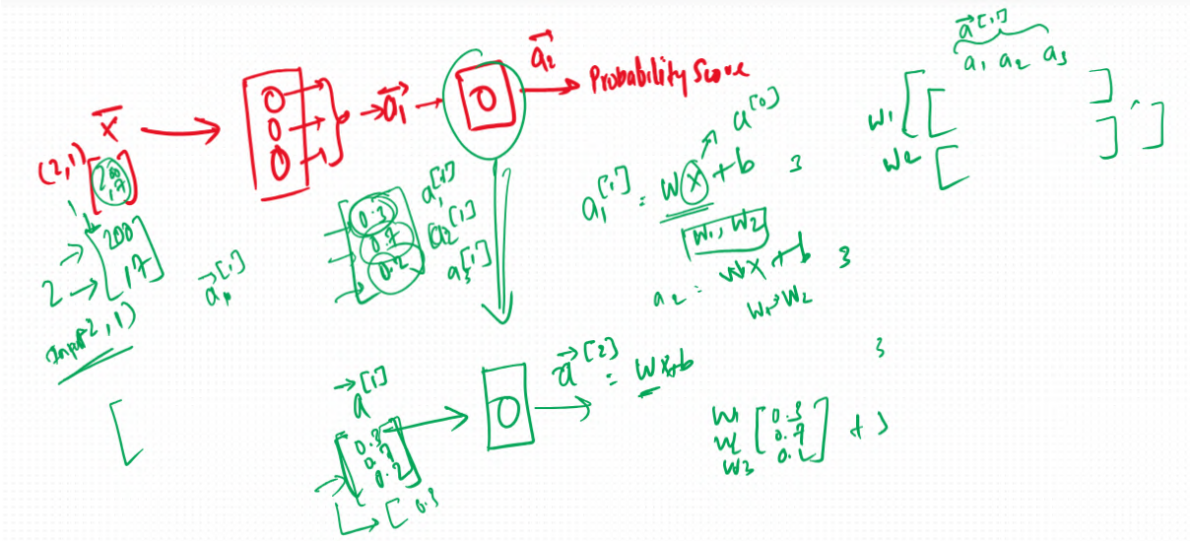
-

▼ How to build a TensorFlow neural network

- We are creation "**sequential**" function to avoid the carryout of activations to other layers manually, This will be taken care within the sequential function.



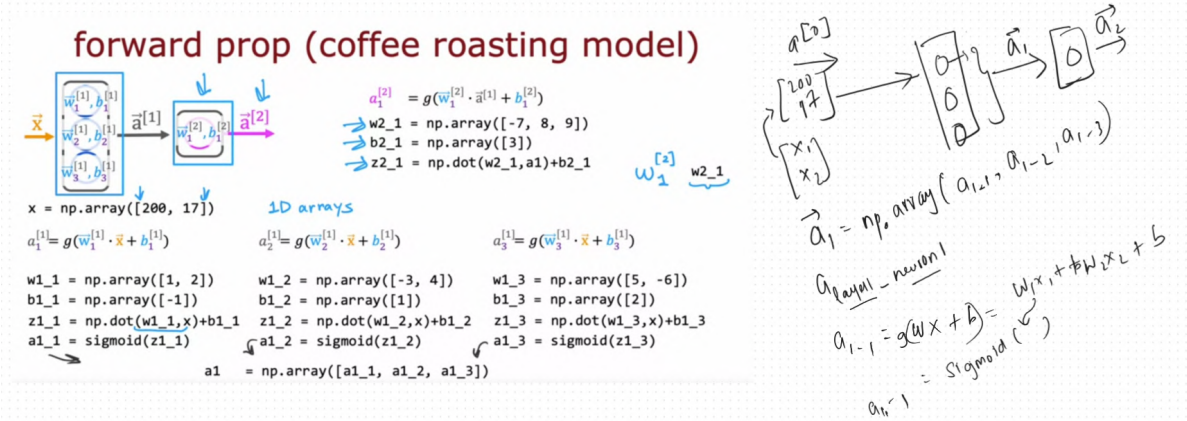
- The input shape imputed within model is not mandatory as this can be represented while fitting the model.
- **Using Sigmoid function at the output layer is not good practice as**



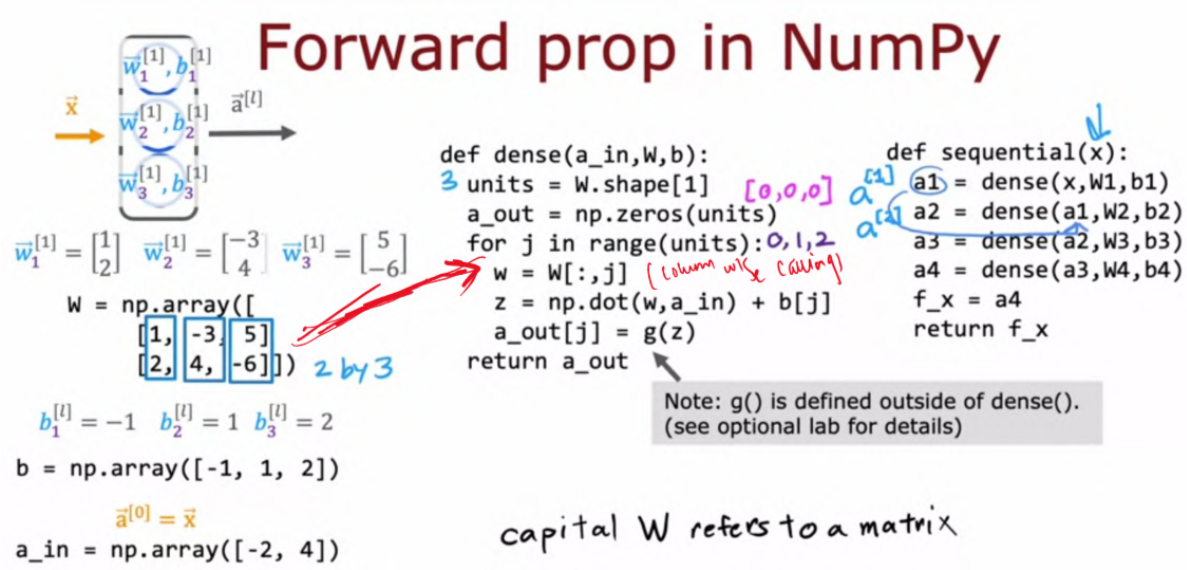
-
- The `model.compile` statement defines a loss function and specifies a compile optimization.
- The `model.fit` statement runs gradient descent and fits the weights to the data.
- What is epochs ?
  - Epochs defines the number of times the training set should be exposed to neural model.
  - if Epochs = 10, which means the whole training set is exposed to model 10 times.
  - these epochs are broken to batches, by default a batch in tensorflow consist of 32 units.
  - if 200,000 records, then batches would be 200,000/32 = 6250 batches.
- Always each unit(neuron) in a layer tries to differentiate and identify. Different neurons within same layer has different methodology of identification. This is similar to average of marginal gain.**It is worth noting that the network learned these functions on its own through the process of gradient descent.** They are very much the same sort of functions a person might choose to make the same decisions.

Small neural network creation

▼ How to build the neural network forward propagation with python



▼ How to build the forward propagation simplistically

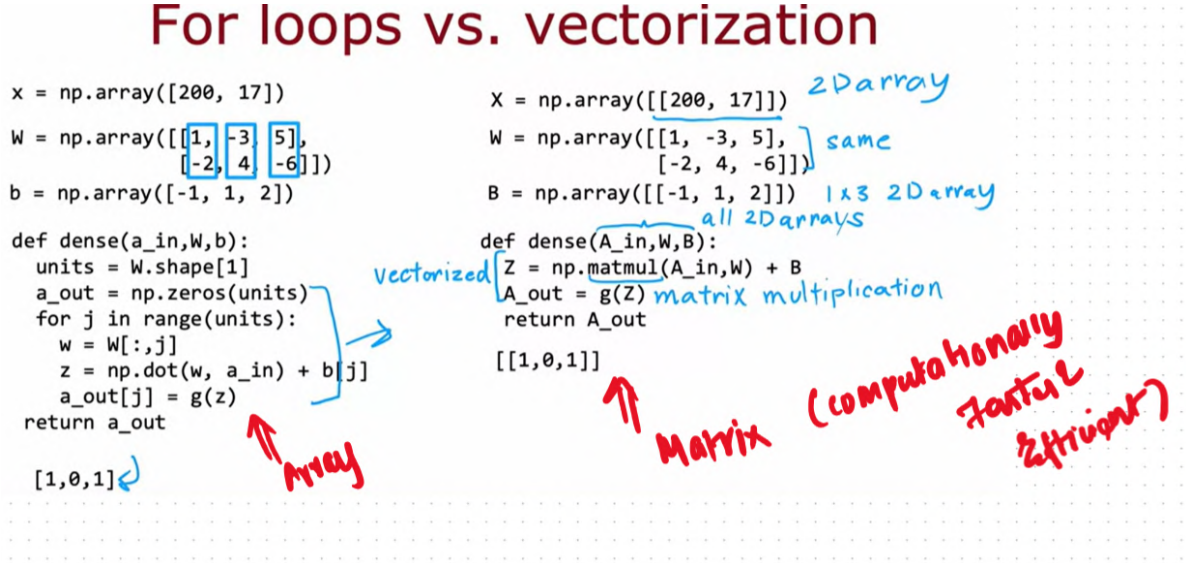


▼ What is AGI ?

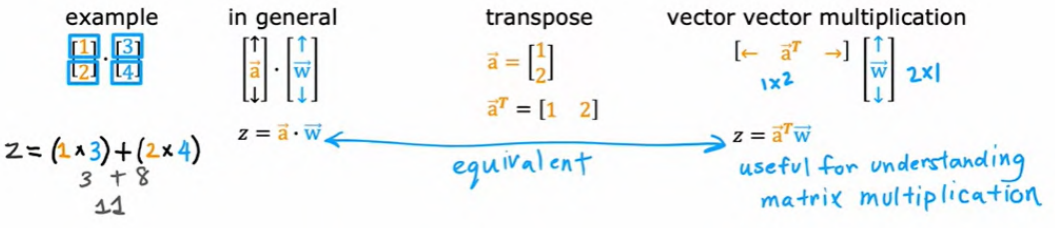
- Artificial General Intelligence
- ANI : Artificial narrow intelligence : specific domain only

▼ What is vectorization, how is it efficient?

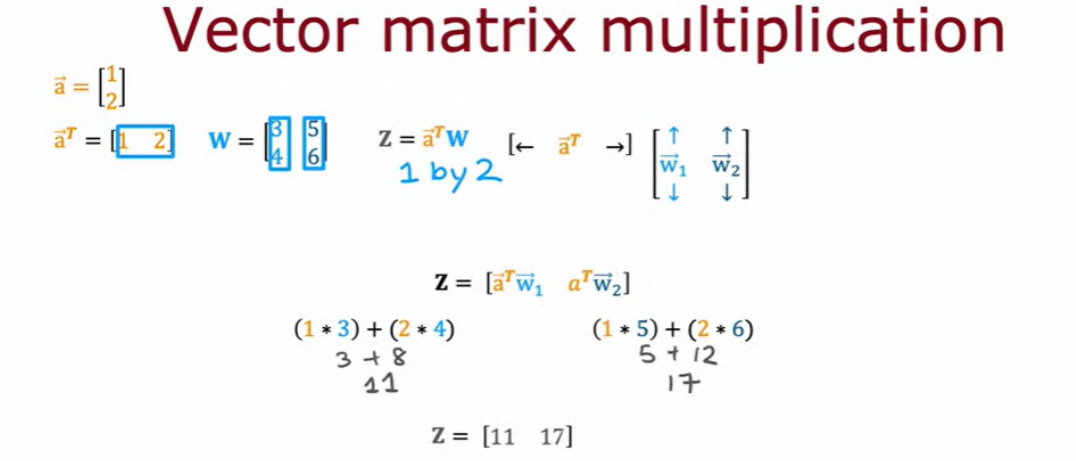
-



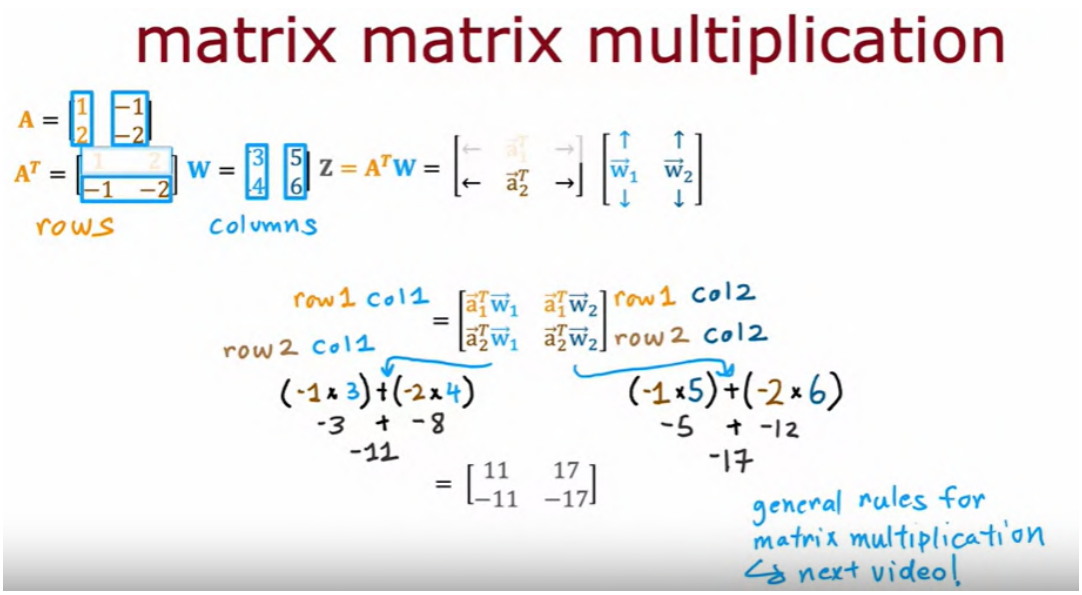
- How does this matmul works ?
  - how dot product works ? : **Vector to vector multiplication**



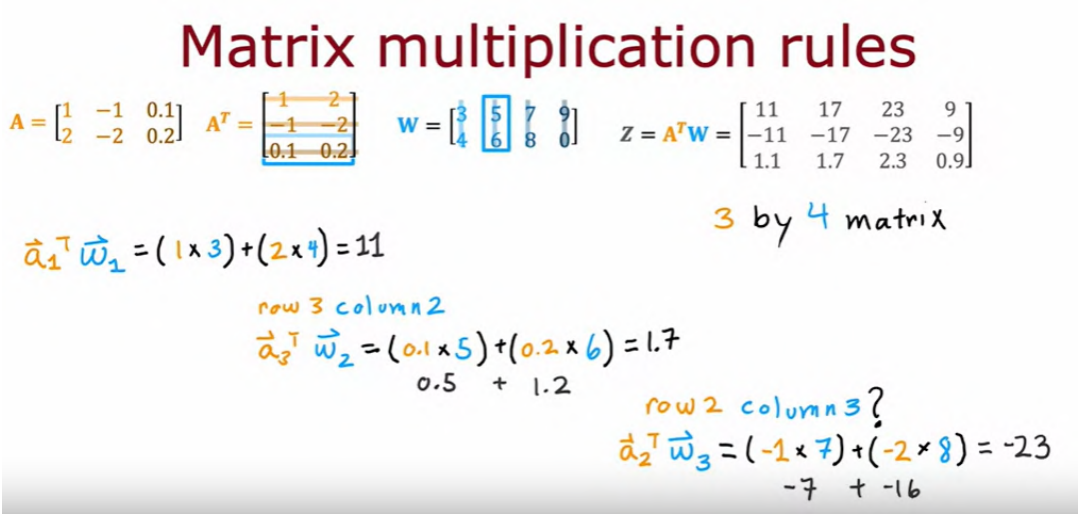
  - how vector matrix multiplication occurs ? **vector to matrix multiplication**

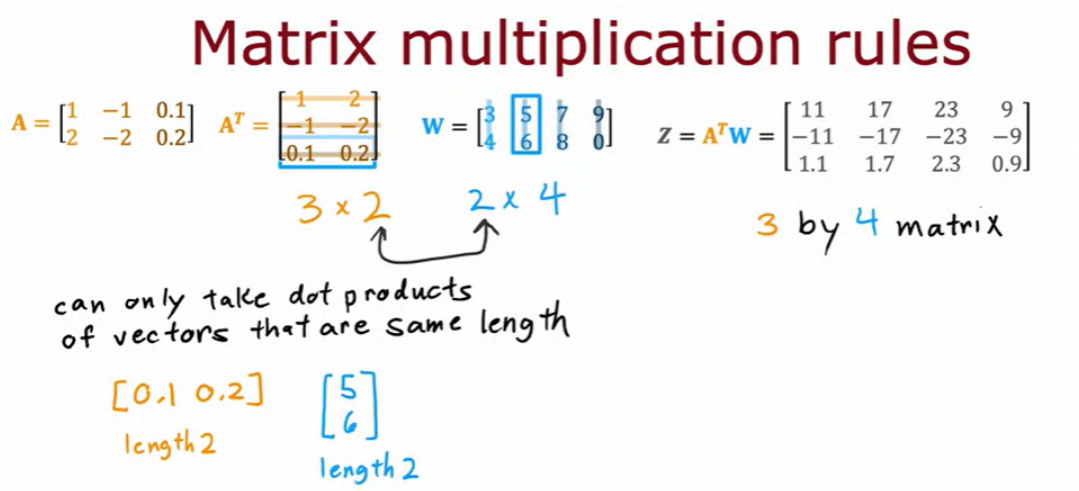


- **What is a matrix ?** : When multiple vectors compiled together in columns to form matrix.
  - However when a matrix is transposed then the vectors are arranged in row wise.
- How matrix to matrix multiplication occurs ?



-

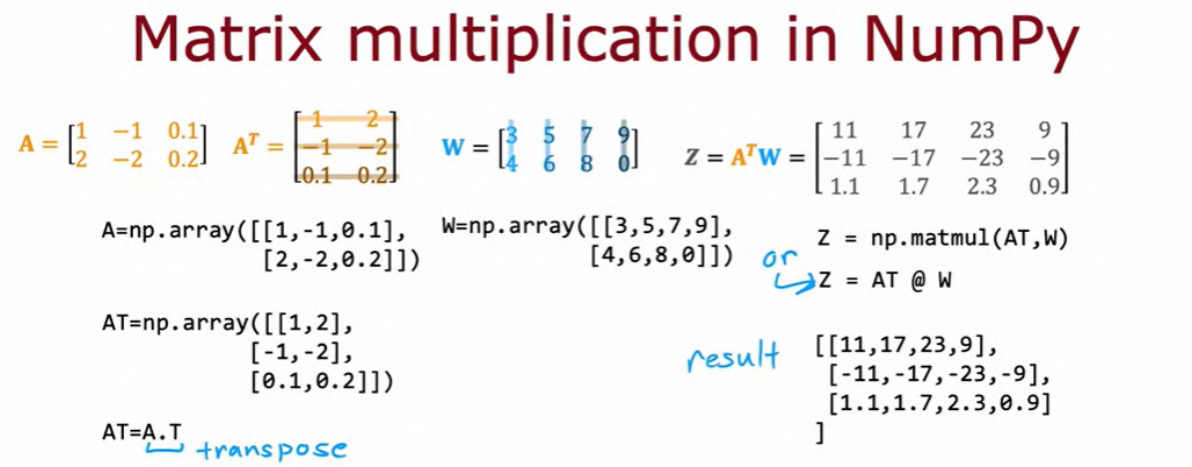




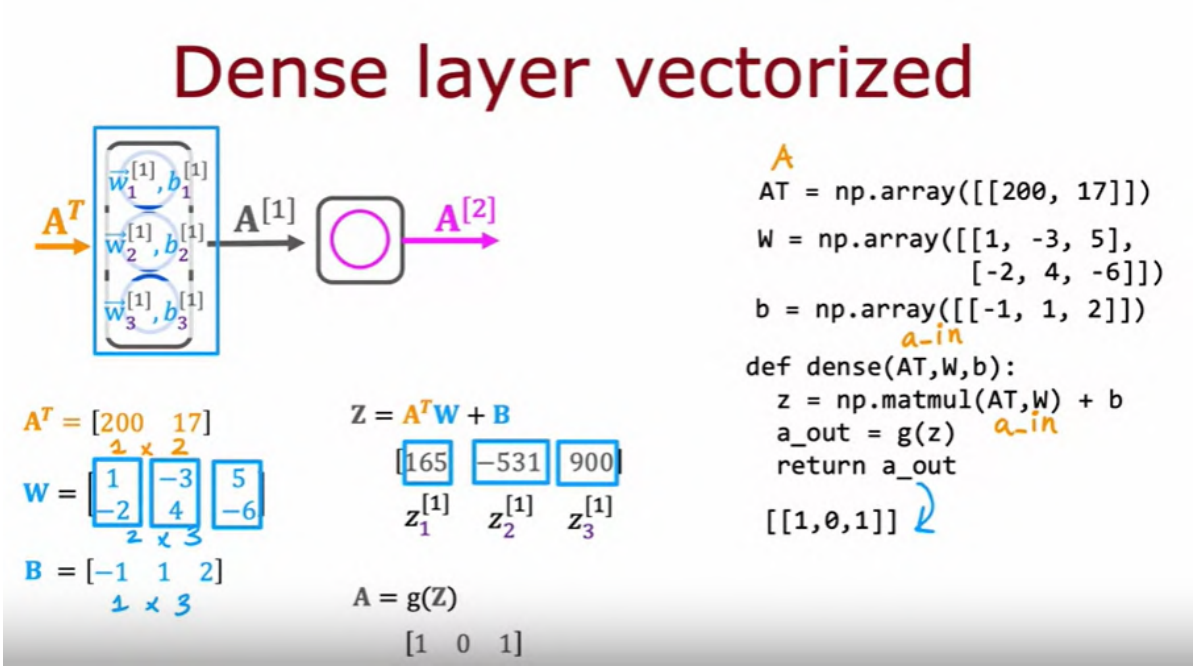
▪

▼ if vectorization is about matrix creation, then how to multiple using python codes ?

- Here **T is the transpose function:**

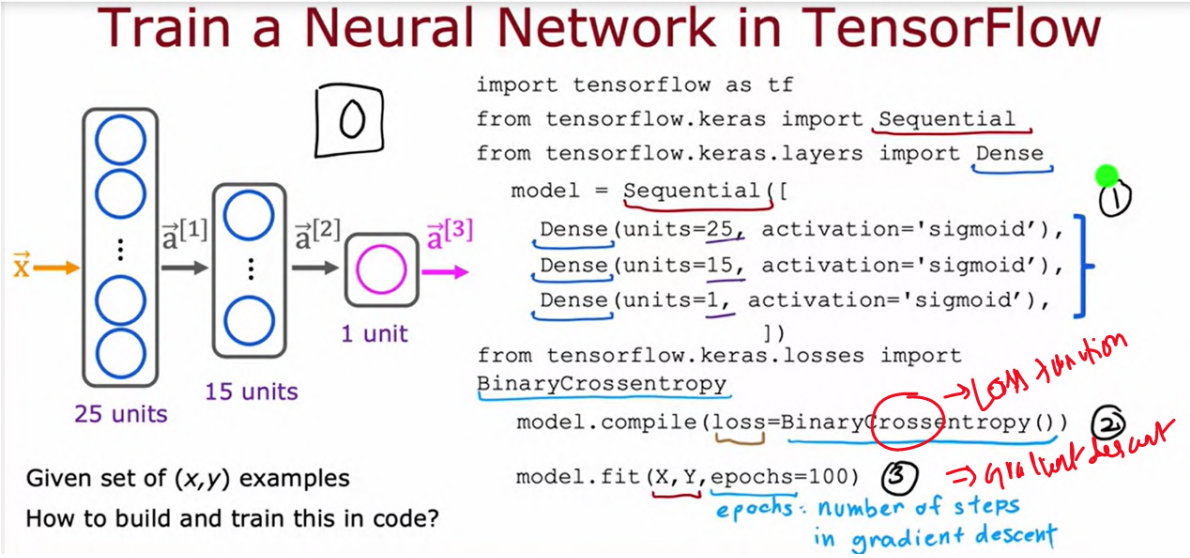


- Always remember when vectors are stacked in column wise that becomes the matrix, same happened here with W.

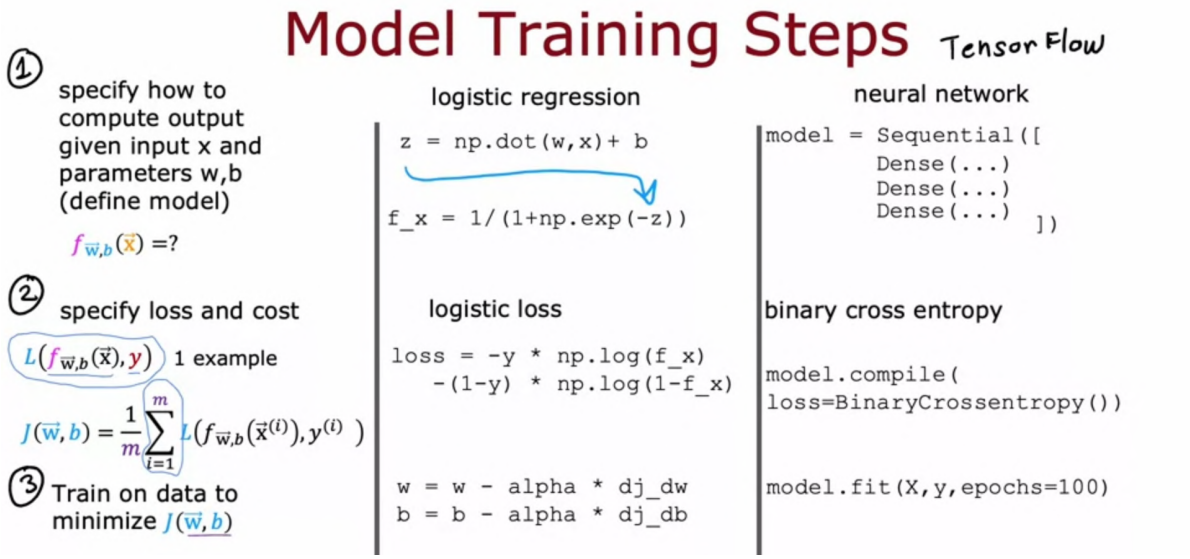


▼ How to train neural network ?

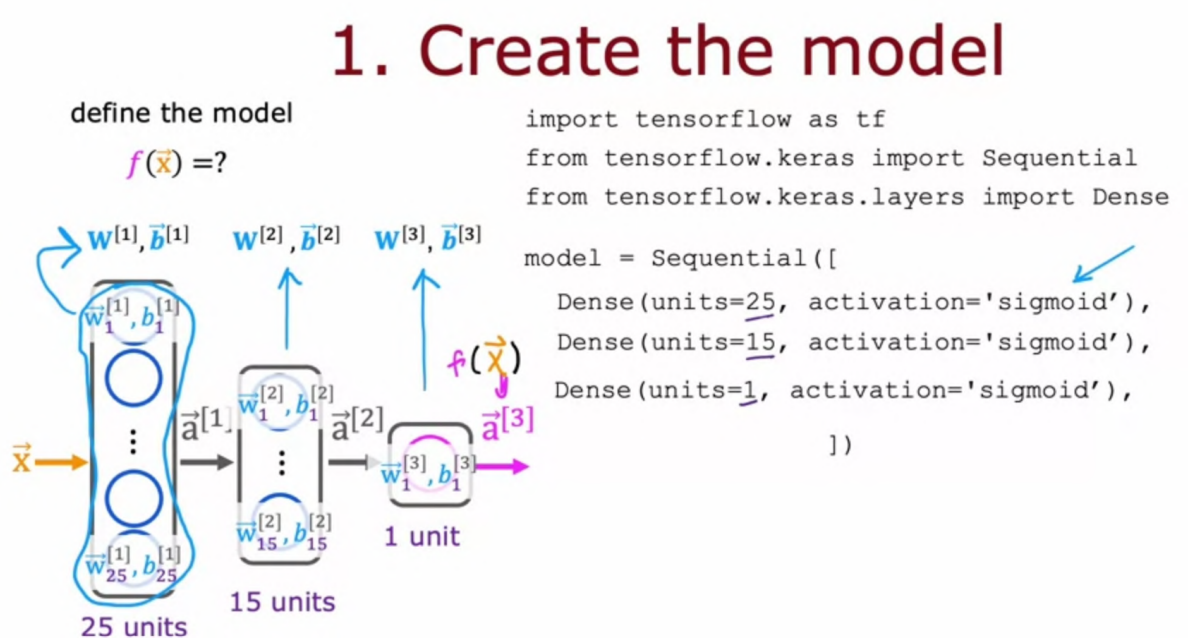
- First step : create a neural architechture using sequential function.
- Second step : model. Compile : explore the loss function
- 3rd Step : model.fit : apply gradient descent where the epochs is number of iteration taken within the gradient descent



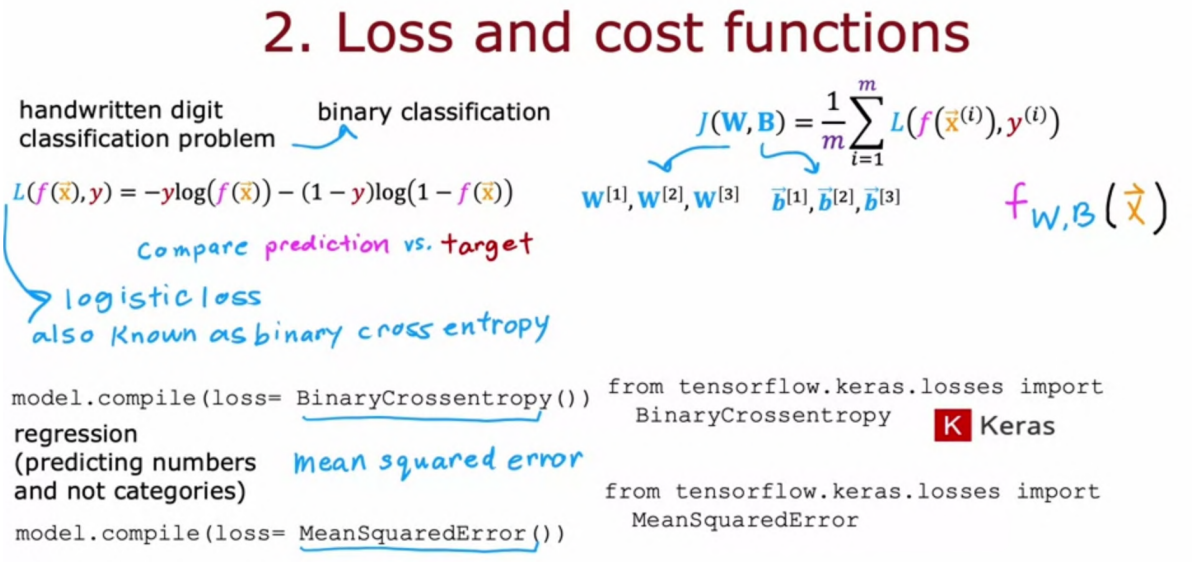
- Analogies of logistic regression and neural network :



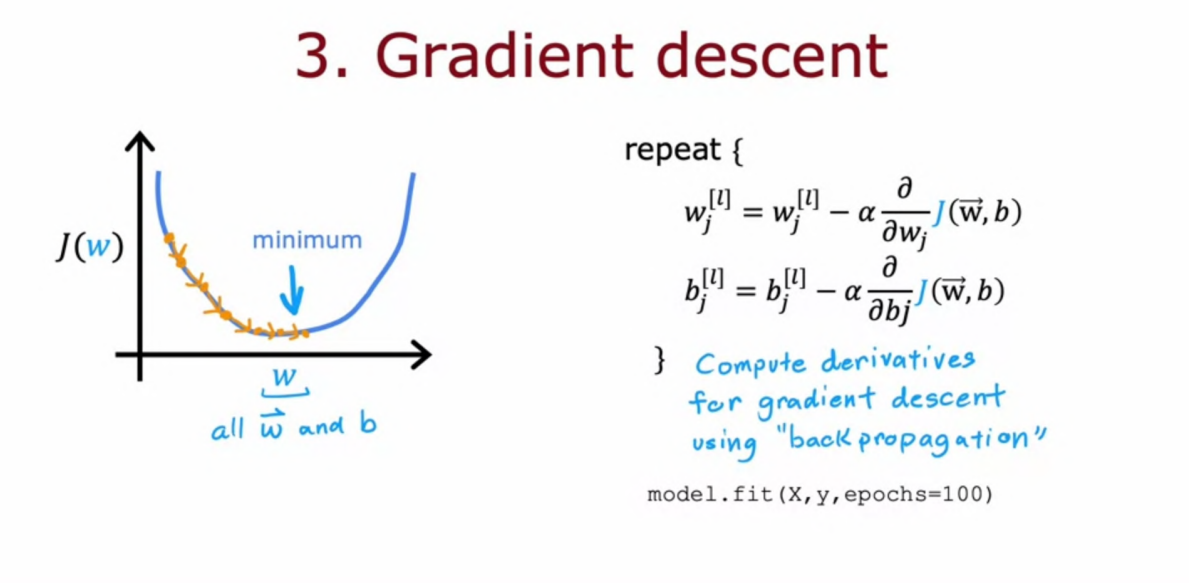
- Step 1 :



- Step 2 : Mentioning the loss function for the neuron , if classification and binary then we use binary cross entropy function, if they are regression problem then we might use mean squared error as loss function.



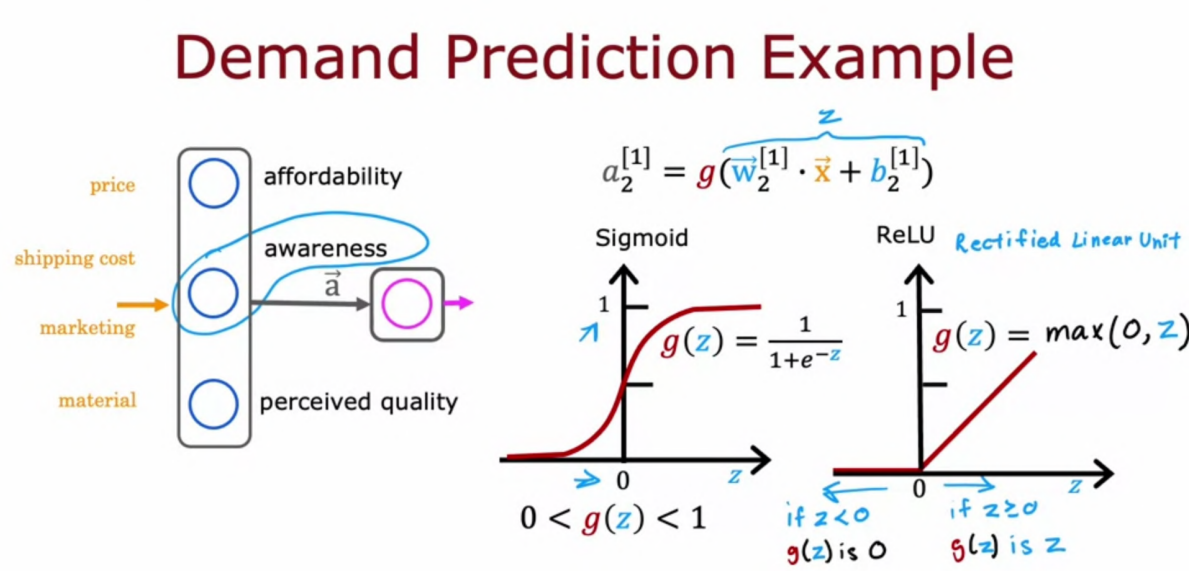
- Step 3 : fitting the model : this does not limit to just applying gradient descent : this gradient descent is achieved by back propagation > while passing backwards the weights are updated thus resulting in updation of weights causing the application of gradient descent.



- Now a days, developers are using the libraries for implementing the deep learning rather than building it from scratch.

▼ What are different types of activation functions ?

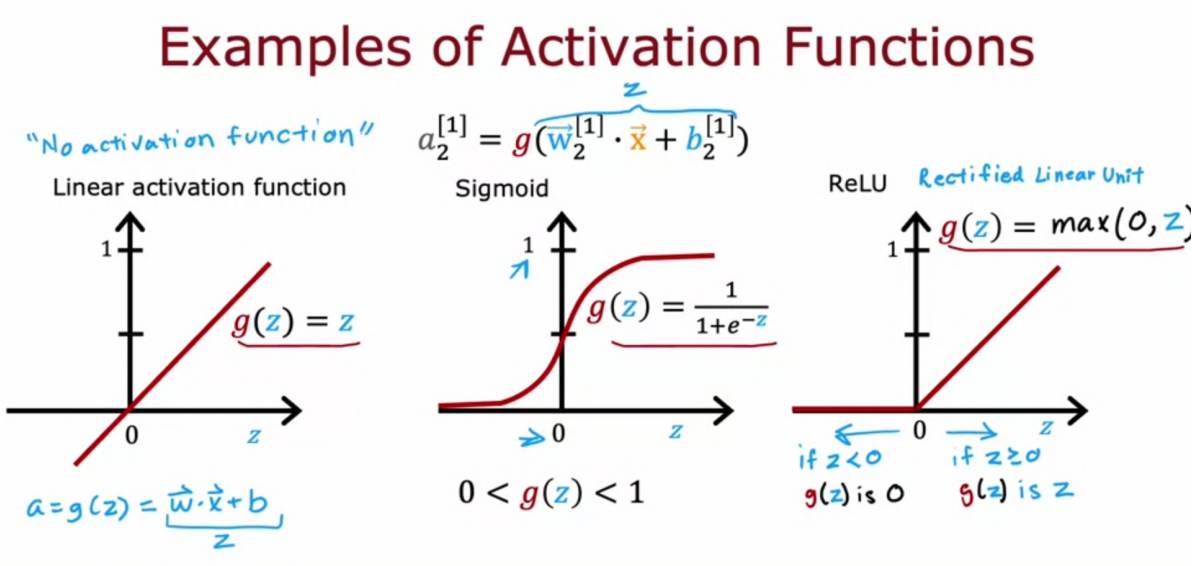
- ReLU : Rectified linear unit : Just a naming conventions used within the industry:



- In Linear activation : this is completely different from not using an activation function, however people sometime refer to this as no activation

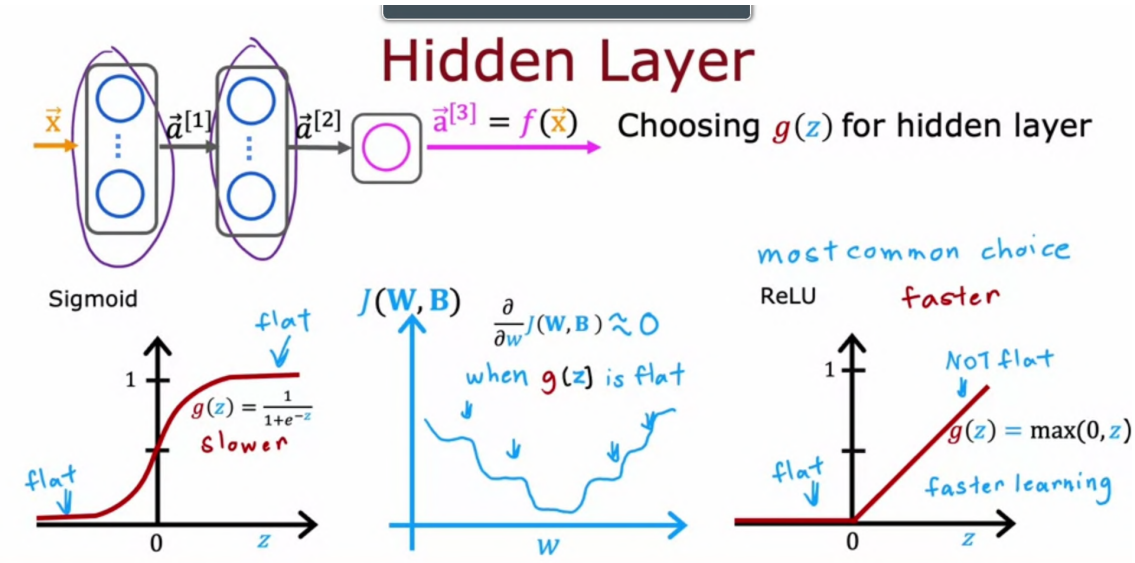
function. But this is great point to start with.

- Major activation function within the market :

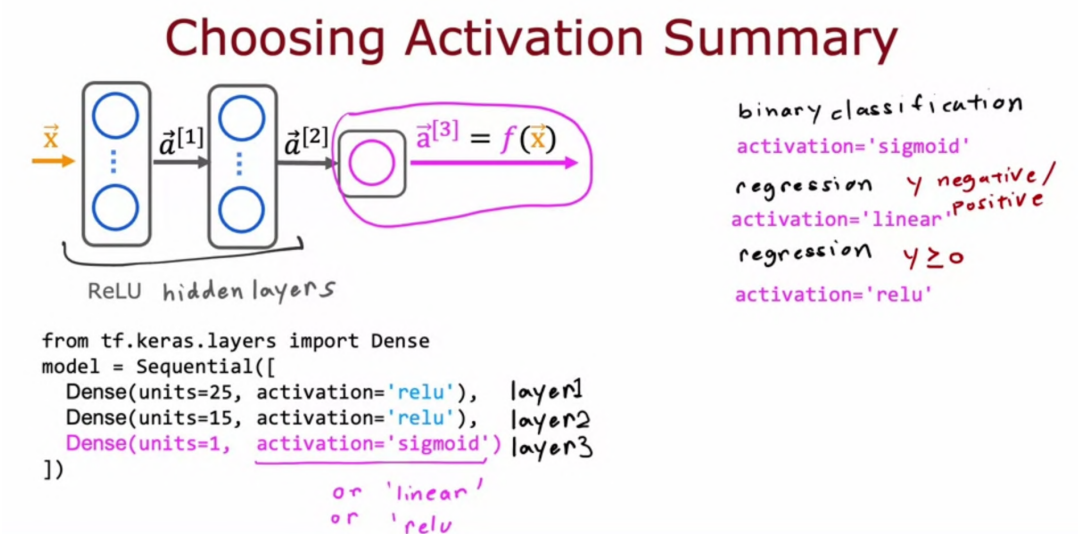


▼ How do we select the better activation functions for your application ?

- For binary classification function : **Sigmoid** classification function
- For regression problem : **Linear activation** would fit this case (this can be applied if the output can contain negative values too eg. Stock prices etc.)
- For regression problem : with non negative output > like house price prediction : **ReLU** would better fit the problem.
- For multiclass classification : **Softmax** function
- For activation within the neural layer or hidden layer : popular choice here is **ReLU** rather than a sigmoid function. But why ?
    - **ReLU is computation is faster** as it has only max(0,z) however for sigmoid it has 1/(1+e^(-z))
    - ReLU is flat on only one side meaning, the value does not change for only one side, however in sigmoid function both the sides are constrained with boundaries. So the gradient descent would be slower for sigmoid than the ReLU function.



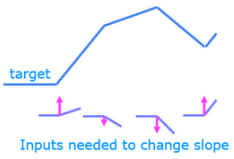
- **Recommendation**
  - Output based
    - if classification : Use sigmoid function
    - if regression : use Linear based function and ReLU for non negative outputs
    - if multiclass classification : you can used softmax function
  - Hidden layer
    - Use ReLU function for better performance.
  -



Other Activation function:

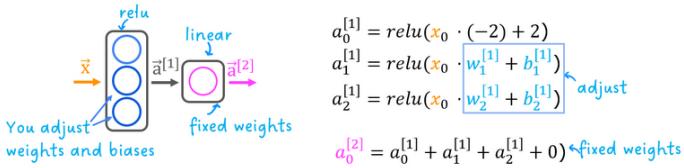
- Leaky ReLU activation functions, TanH activations functions , Switch activation functions

Why Non-Linear Activations?

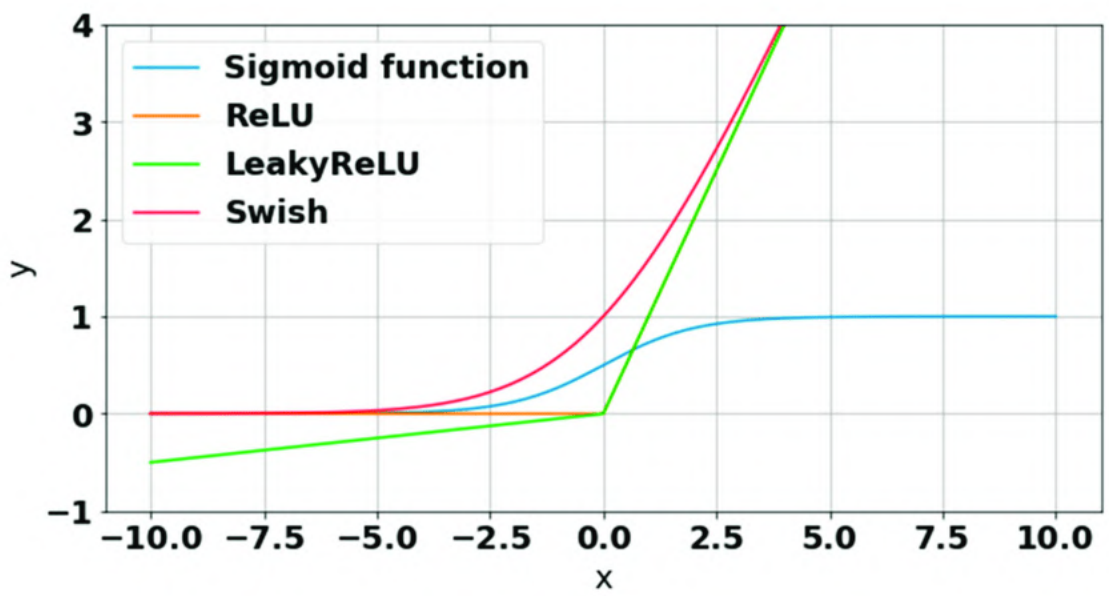


The function shown is composed of linear pieces (piecewise linear). The slope is consistent during the linear portion and then changes abruptly at transition points. At transition points, a new linear function is added which, when added to the existing function, will produce the new slope. The new function is added at transition point but does not contribute to the output prior to that point. The non-linear activation function is responsible for disabling the input prior to and sometimes after the transition points. The following exercise provides a more tangible example.

The exercise will use the network below in a regression problem where you must model a piecewise linear target :

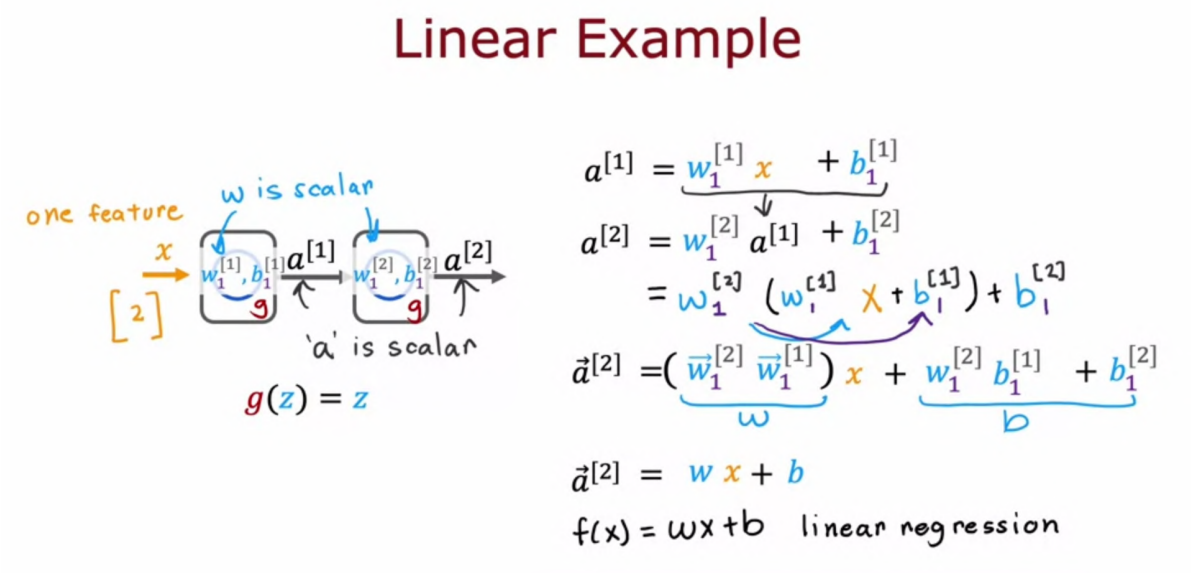


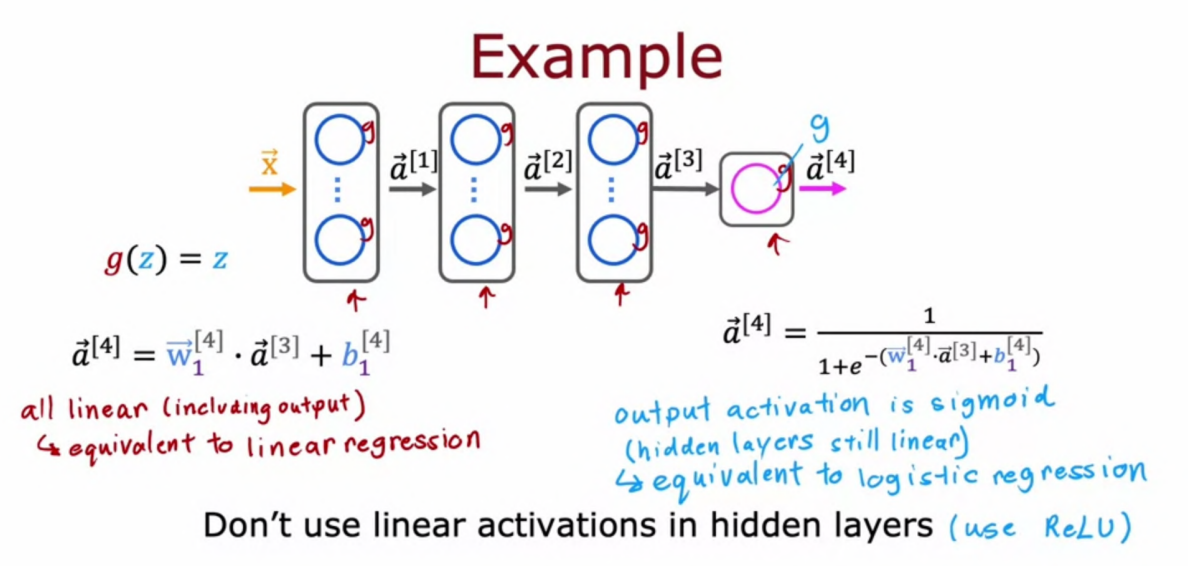
The network has 3 units in the first layer. Each is required to form the target. Unit 0 is pre-programmed and fixed to map the first segment. You will modify weights and biases in unit 1 and 2 to model the 2nd and 3rd segment. The output unit is also fixed and simply sums the outputs of the first layer.



▼ Why do activation functions are essential for performance ?

- here g(z) = z , which means linear example, after passing a neuron a same impact has been created. So, there is no need of layer 1, which is meaning less. No major learning has been done. **So never use a linear functions within the hidden layer**



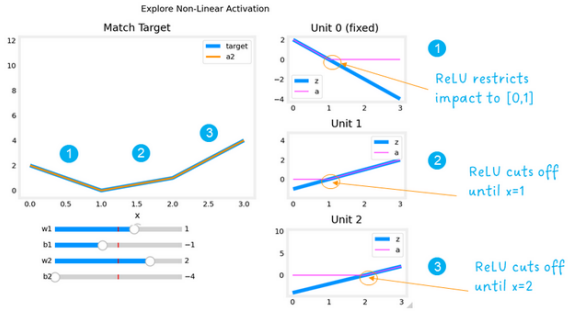


▼ how will you build neural network classification problem with multiple classes ?

▼ ReLU often introduces non linearity within the activations. how ?

The goal of this exercise is to appreciate how the ReLU's non-linear behavior provides the needed ability to turn functions off until they are needed. Let's see how this worked in this example. The plots on the right contain the output of the units in the first layer.
Starting at the top, unit 0 is responsible for the first segment marked with a 1. Both the linear function z and the function following the ReLU a are shown. You can see that the ReLU cuts off the function after the interval [0,1]. This is important as it prevents Unit 0 from interfering with the following segment.

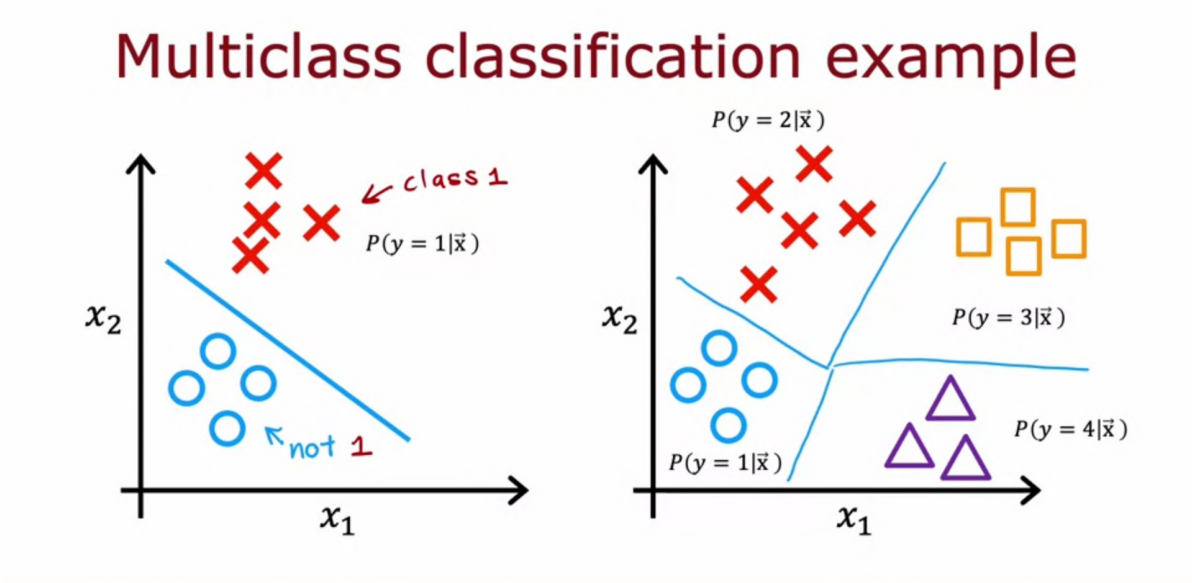
Unit 1 is responsible for the 2nd segment. Here the ReLU kept this unit quiet until after x is 1. Since the first unit is not contributing, the slope for unit 1, $w_1^{[1]}$, is just the slope of the target line. The bias must be adjusted to keep the output negative until x has reached 1. Note how the contribution of Unit 1 extends to the 3rd segment as well.



Unit 2 is responsible for the 3rd segment. The ReLU again zeros the output until x reaches the right value.The slope of the unit, $w_2^{[1]}$, must be set so that the sum of unit 1 and 2 have the desired slope. The bias is again adjusted to keep the output negative until x has reached 2.

The "off" or disable feature of the ReLU activation enables models to stitch together linear segments to model complex non-linear functions

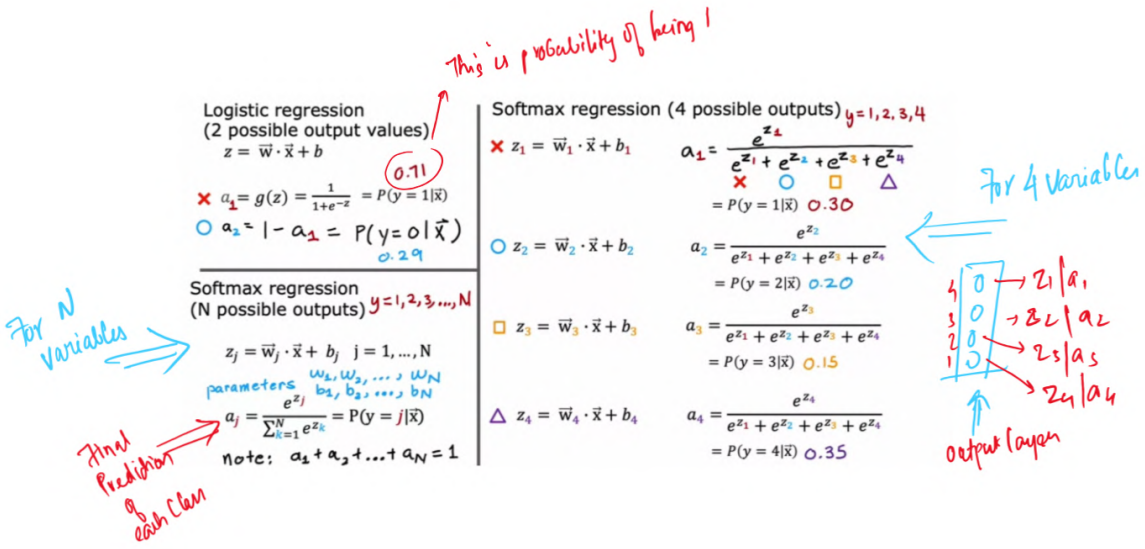
- Output labels are more than 2.
- Decision boundaries helps in drawing the classes out of it



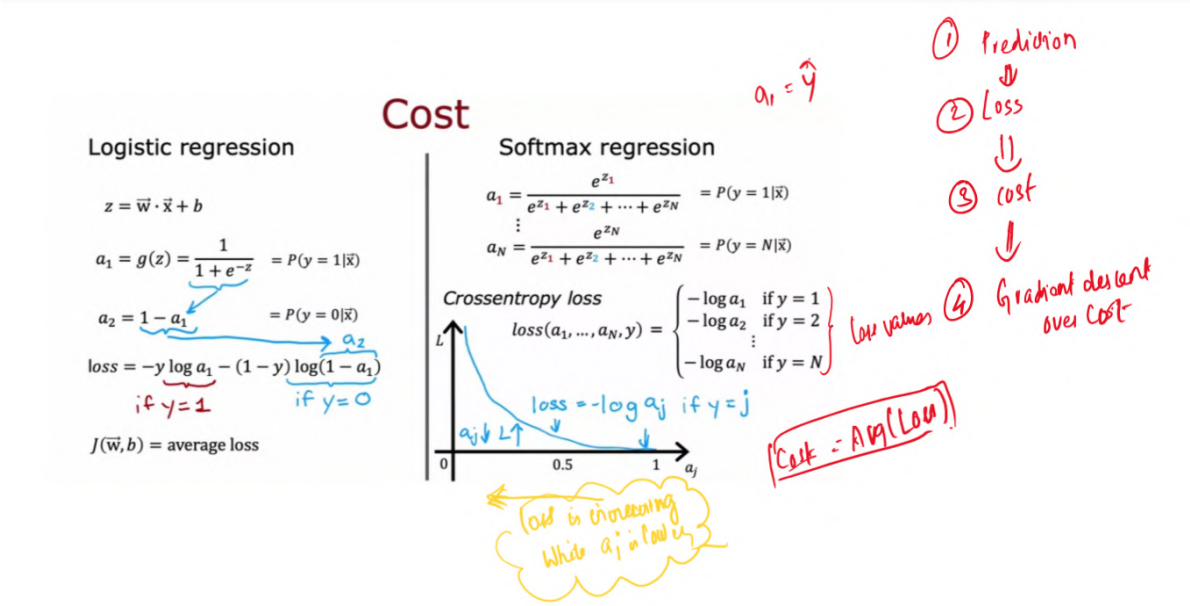
- we will be using the SoftMax regression function which is a generalized form of sigmoid function
-

▼ What is SoftMax function ? how it is used in multi-class classification problems?

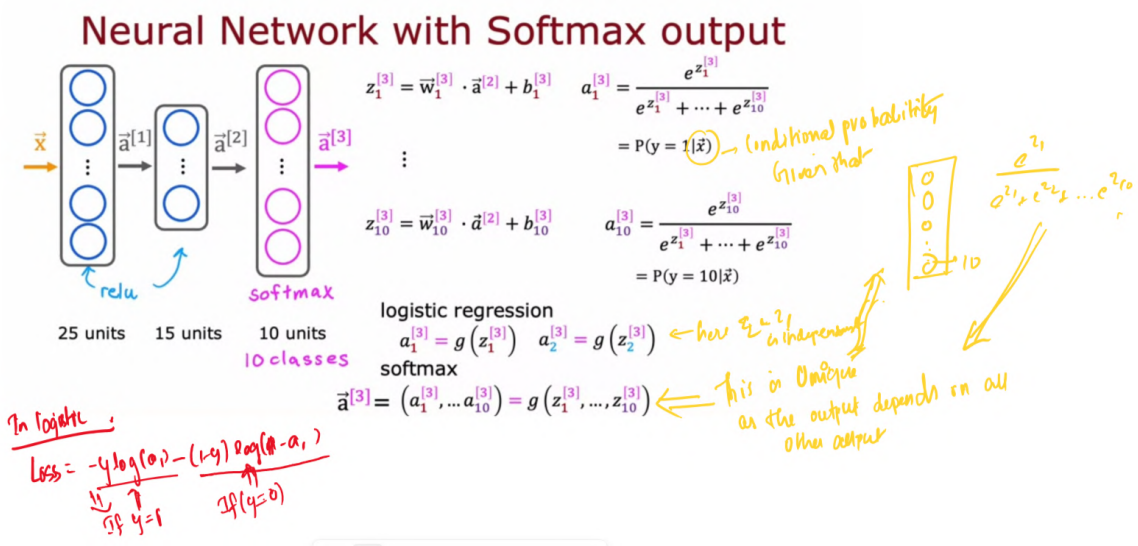
- When SoftMax function applied for 2 outputs then it becomes a logistic regression.



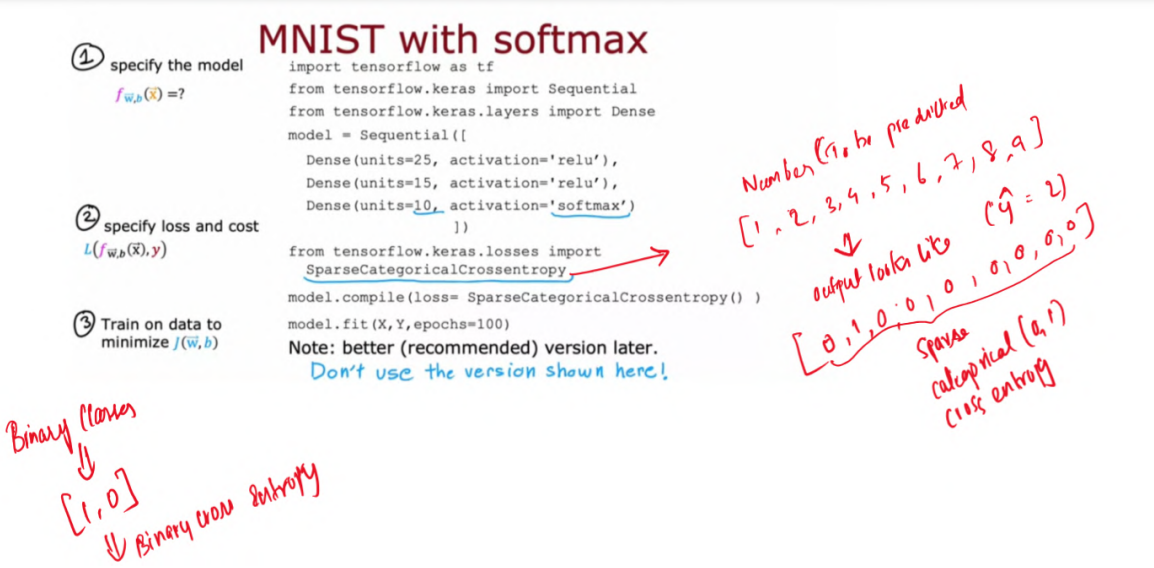
- cost function within SoftMax function.



- uniqueness of SoftMax function :

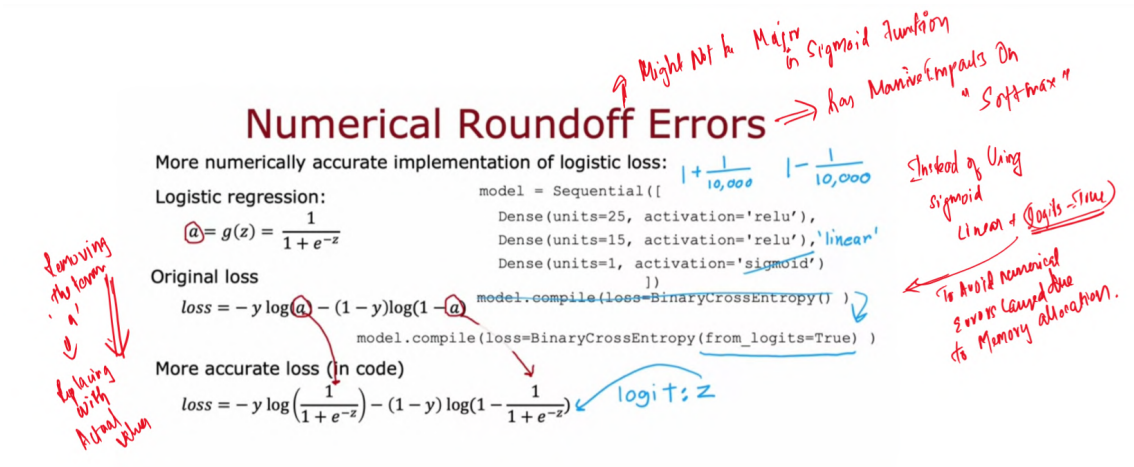


- Loss function and steps of implementation :

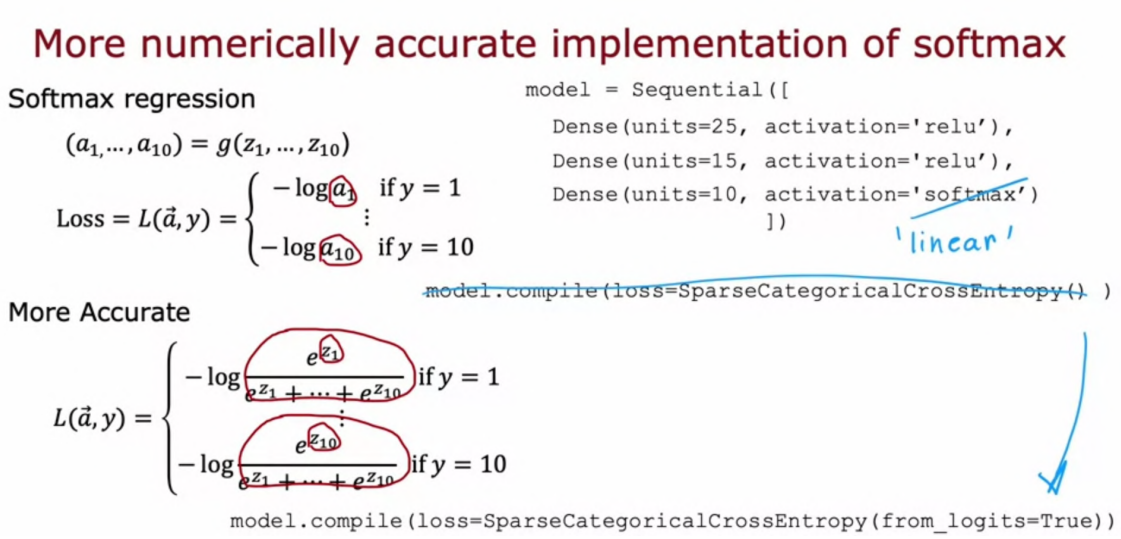


▼ how to implement the improved version of SoftMax function ?

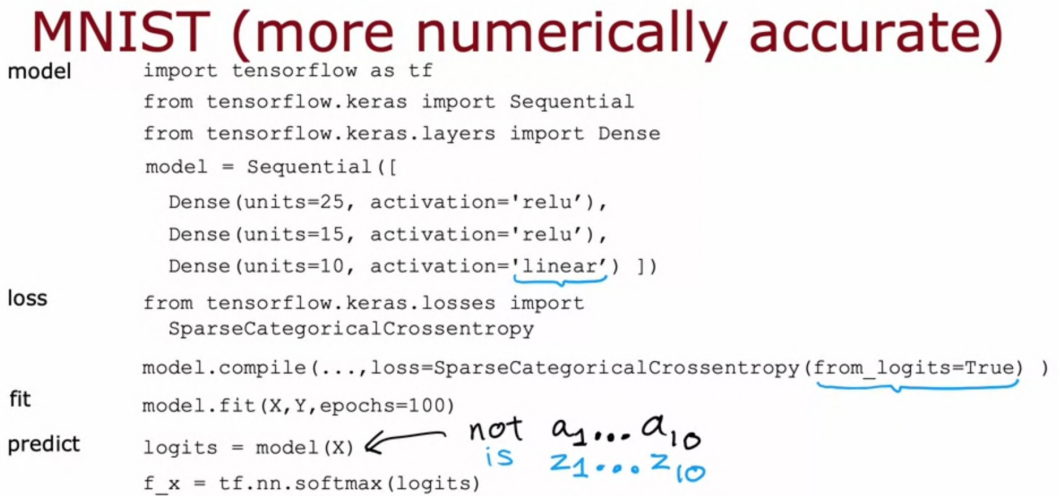
- There occurs a rounding off error in the above implementation of SoftMax function
- The memory of each digits are finite, so instead of storing them in separate variable, it would be optimal for us to express within the equation for better and accurate(re-arranges to a optimized execution) results.
-



-



- Implementation :
  - More accurate implementation of Softmax function



  - More accurate way of implementing logistic function :

logistic regression (more numerically accurate)

model

```
model = Sequential([
  Dense(units=25, activation='sigmoid'),
  Dense(units=15, activation='sigmoid'),
  Dense(units=1, activation='linear')
                ])
from tensorflow.keras.losses import
  BinaryCrossentropy
```

loss

```
model.compile(..., BinaryCrossentropy(from_logits=True))
```

```
model.fit(X,Y,epochs=100)
```

fit

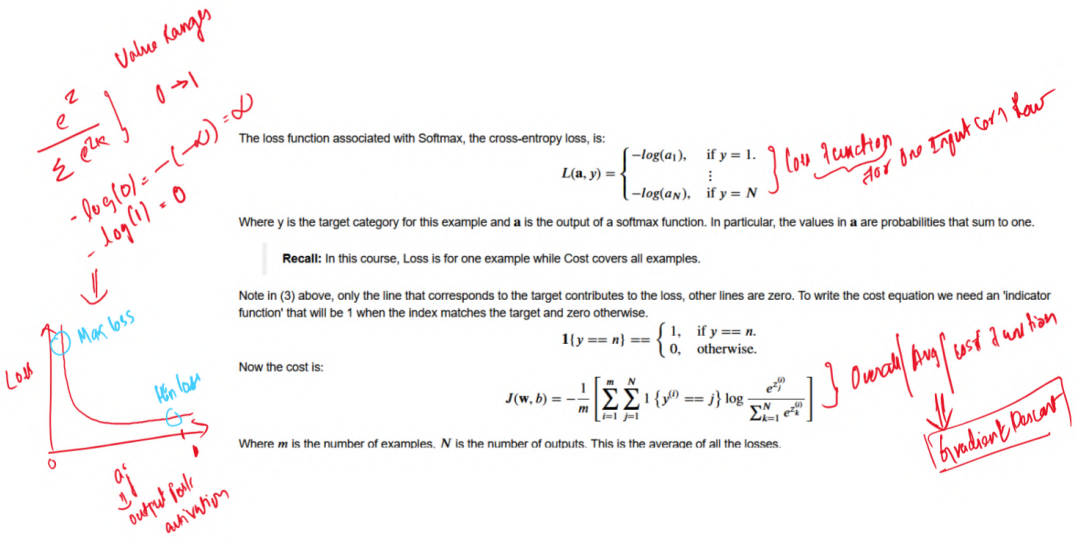
```
logit = model(X)
```

predict

```
f_x = tf.nn.sigmoid(logit)
```

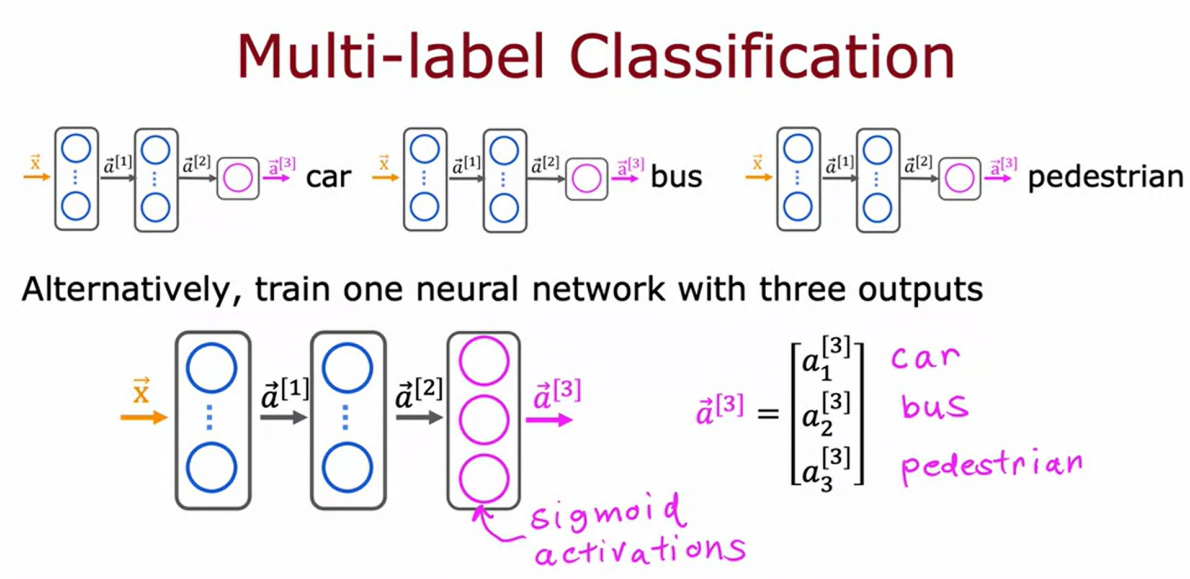
▼ Loss and cost function of SoftMax regression function ?

-



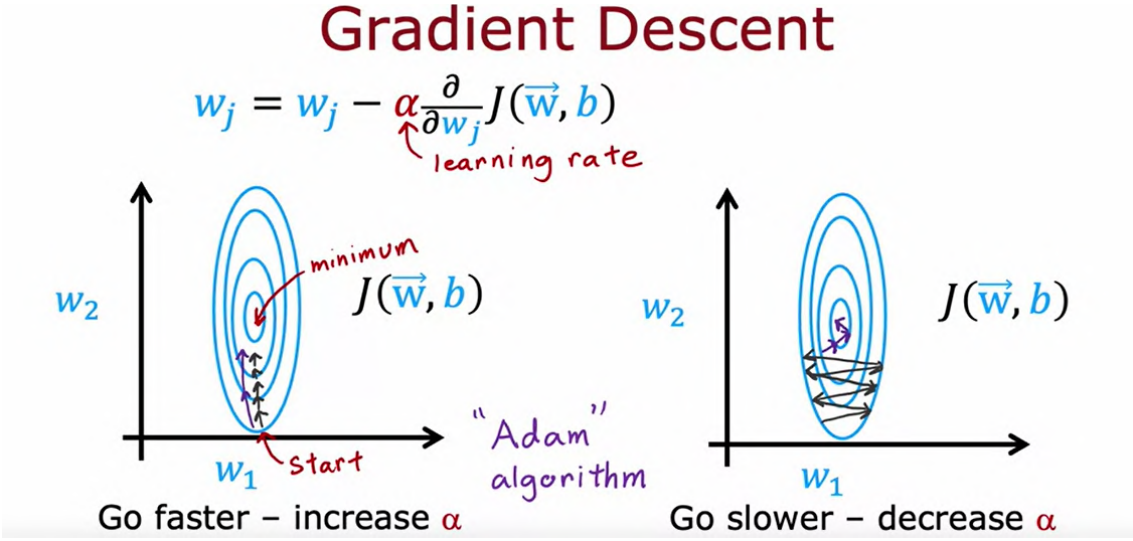
▼ What is multi - label classification ?

- In image recognition, sometime we predict more than one object: eg. Detecting traffic signal and pedestrians simultaneously. These kind of problems where the detection or outcome required is more than one : is called multi-label classification
-

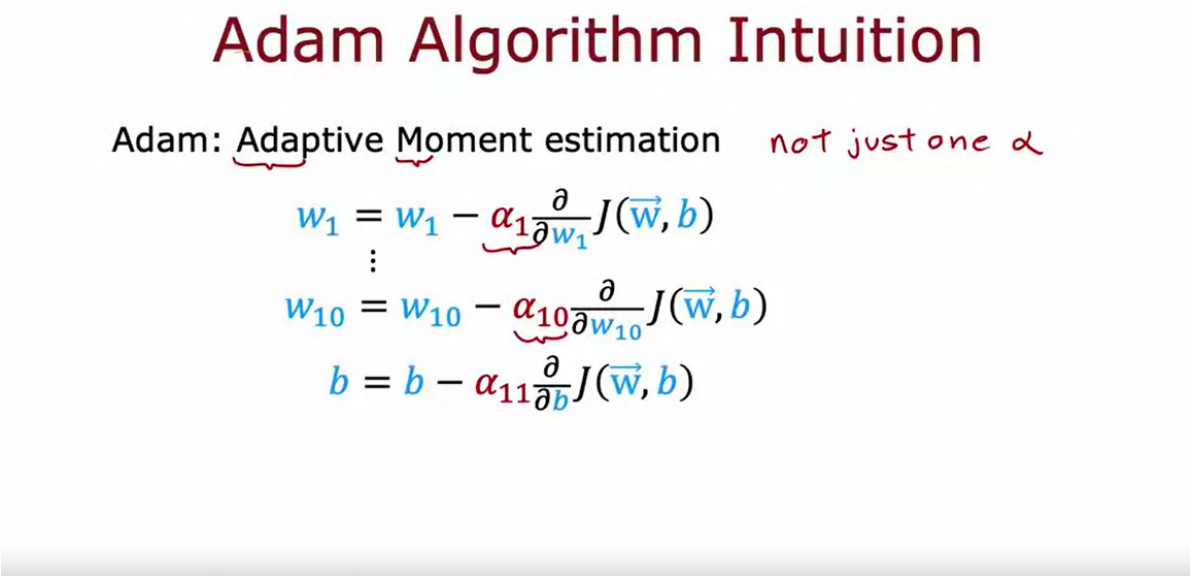


▼ What are the other methods than reducing cost function by gradient descent ?

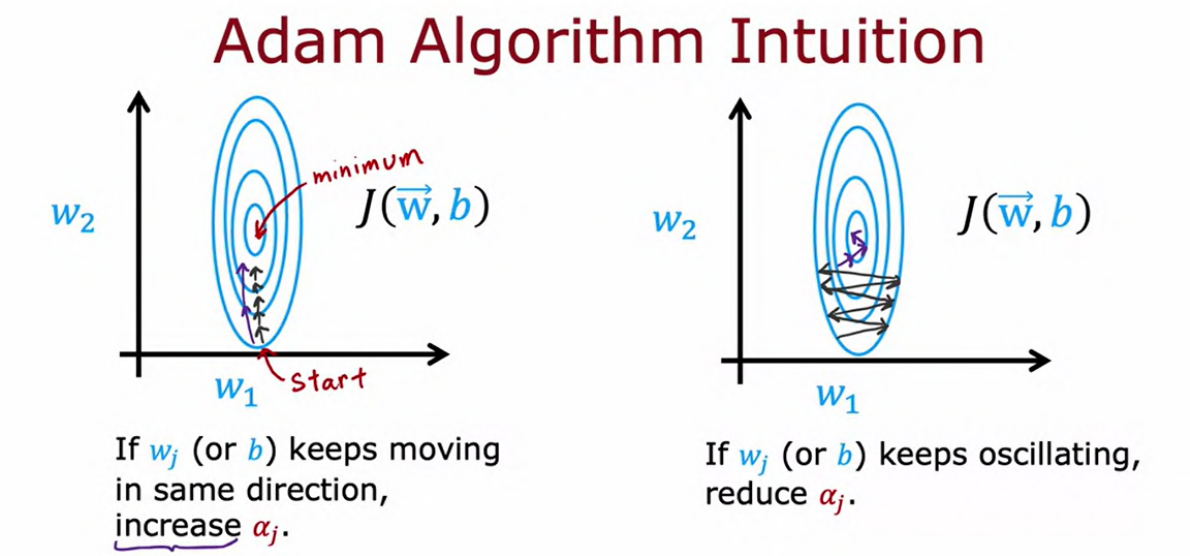
- When applying gradient descent the weights tend to converge when the cost function is minimum, if the learning rate is too low then this would take smaller steps though its converging → so to make this faster we are using an optimizer called "**Adam**"



- This Adam optimizer guides and changes the learning rate such that to converge faster bby manipulating the alpha.
- The learning is not global here.



- here the alpha changes with the nature of change of parameters w and b. Below picture depicts it :



- its robust with the learning rate optimization, the implementation code can be found below :

MNIST Adam

model

```
model = Sequential([
        tf.keras.layers.Dense(units=25, activation='sigmoid'),
        tf.keras.layers.Dense(units=15, activation='sigmoid'),
        tf.keras.layers.Dense(units=10, activation='linear')
])
```

compile

$\alpha = 10^{-3} = 0.001$

```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

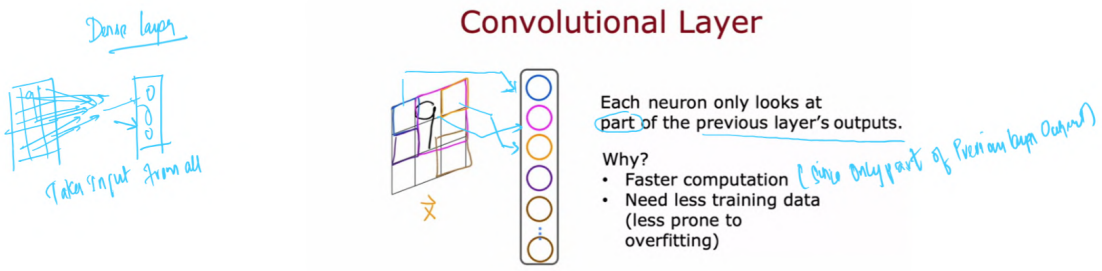
fit

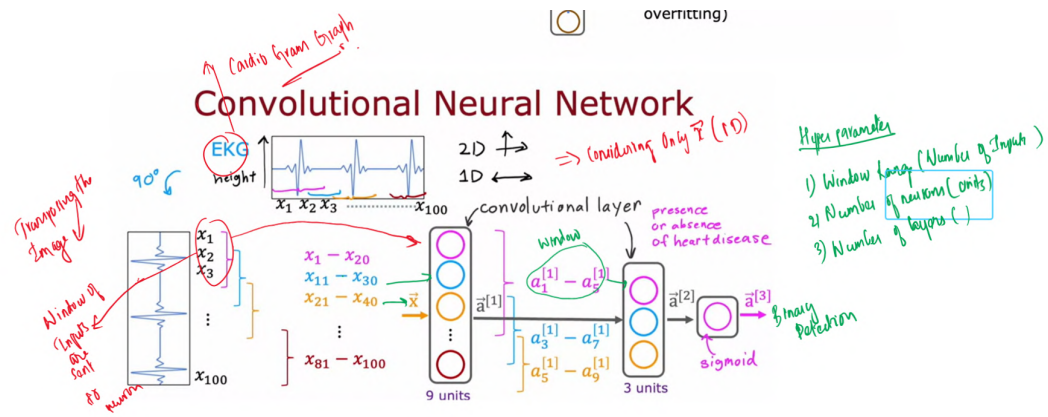
```
model.fit(X,Y,epochs=100)
```

▼ What are the alternative way or types ?

▼ Different type of neural network

- Convolutional layer :

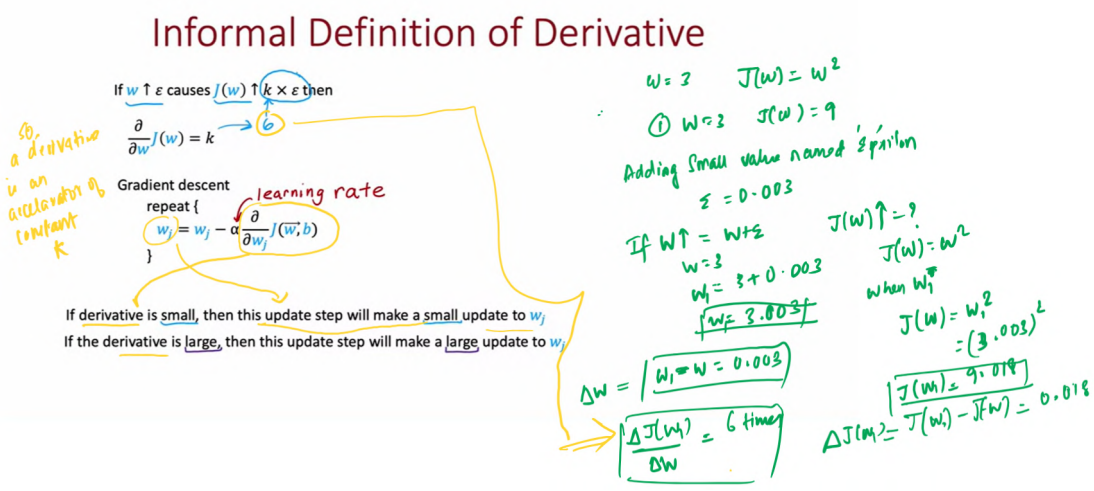




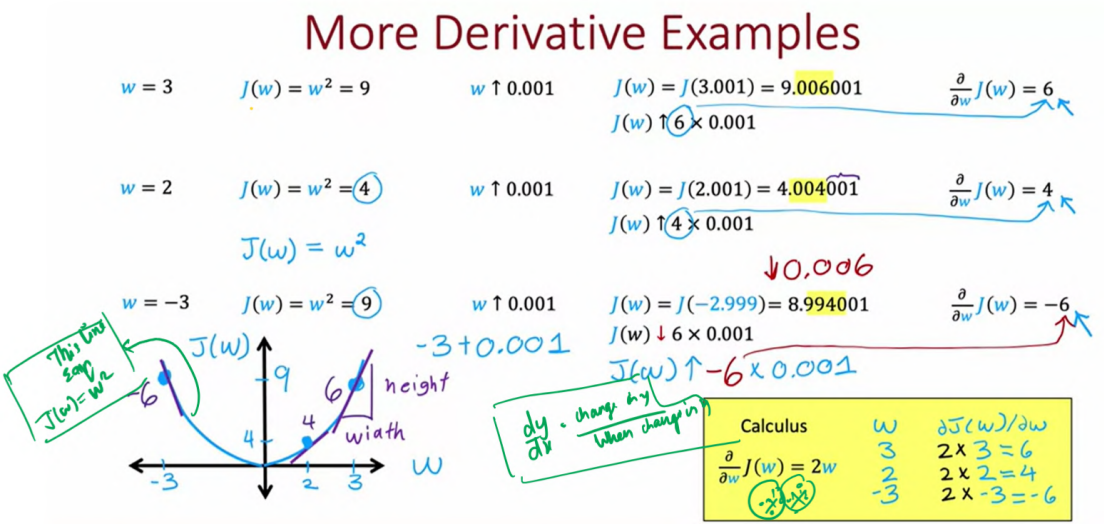
- you can see a sliding window which determines the number of inputs to be sent to other layer, instead of considering all the inputs.
-

▼ How the back propagation works ?

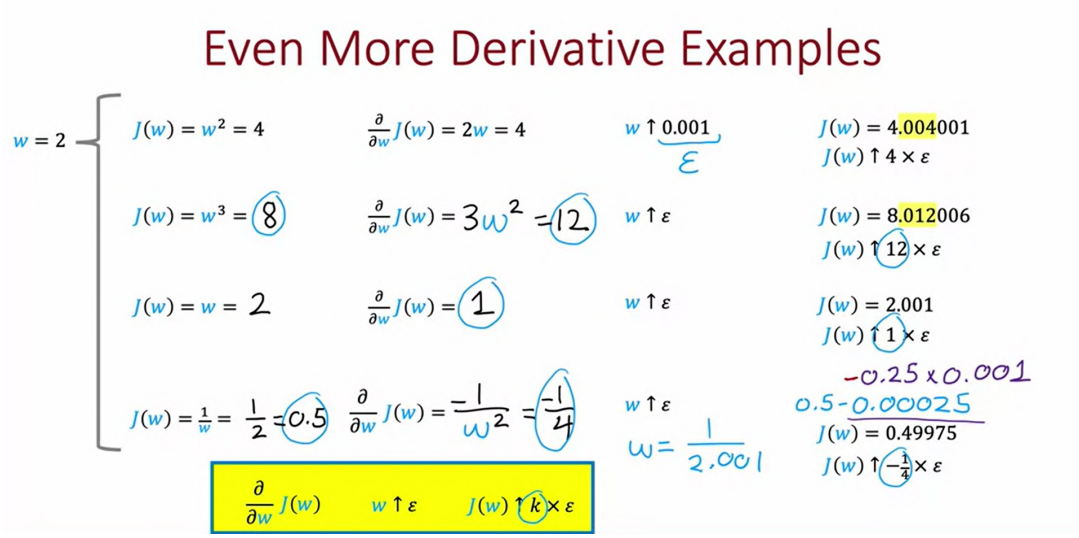
▼ Understanding derivative:

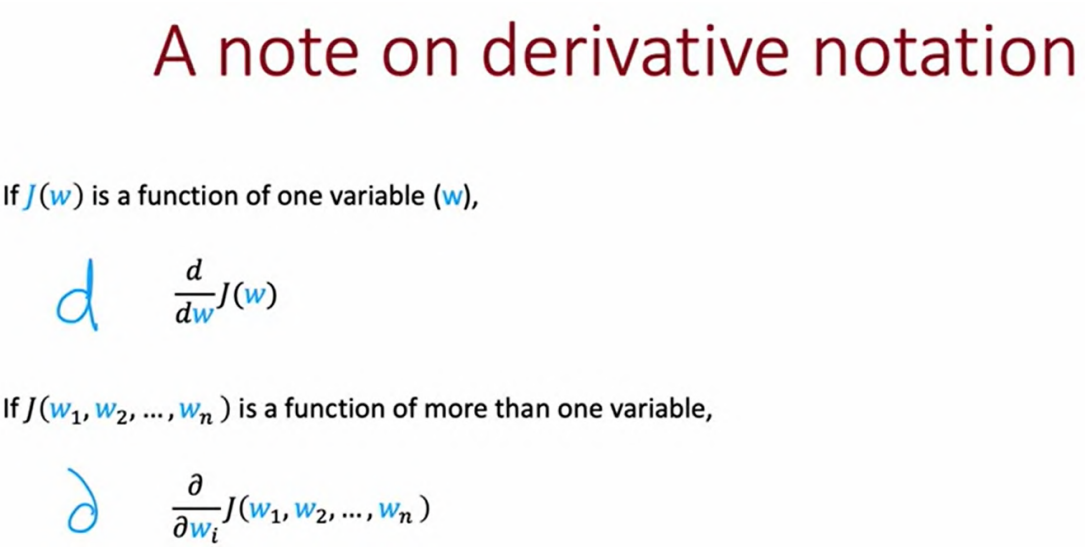


- The derivative is impacted by the weights of the parameter not Loss function :



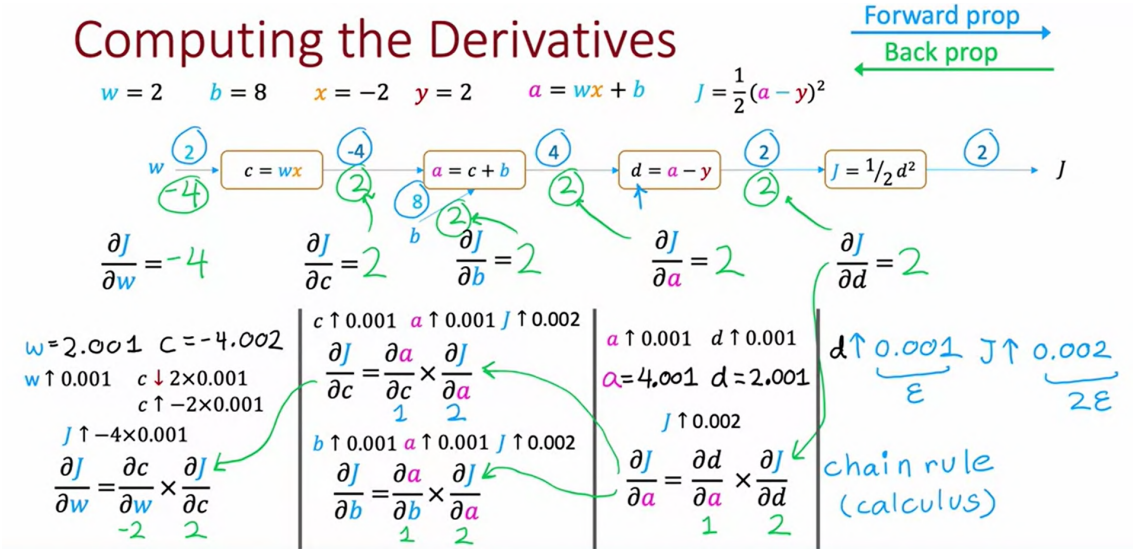
- sympy : library is used to implement the derivative of specific function.
  -



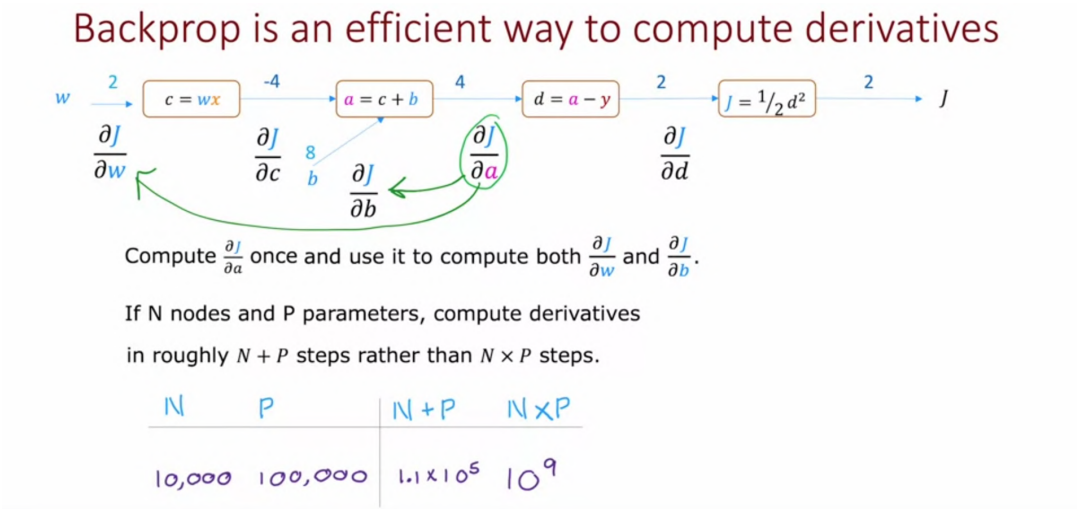


▼ How backpropagation works ?

- the application of partial derivative from left to right helps in normalizing the weights of the neuron. The partial derivative applies chain rule as well.



- why back propagation is essential ?
  - Right to left propagation essentially helps to understand, how the output is impacted by the input. How they are interlinked.
  - This back propagation helps in achieving highly efficient computing by minimizing the parameters mentioned in the image below :

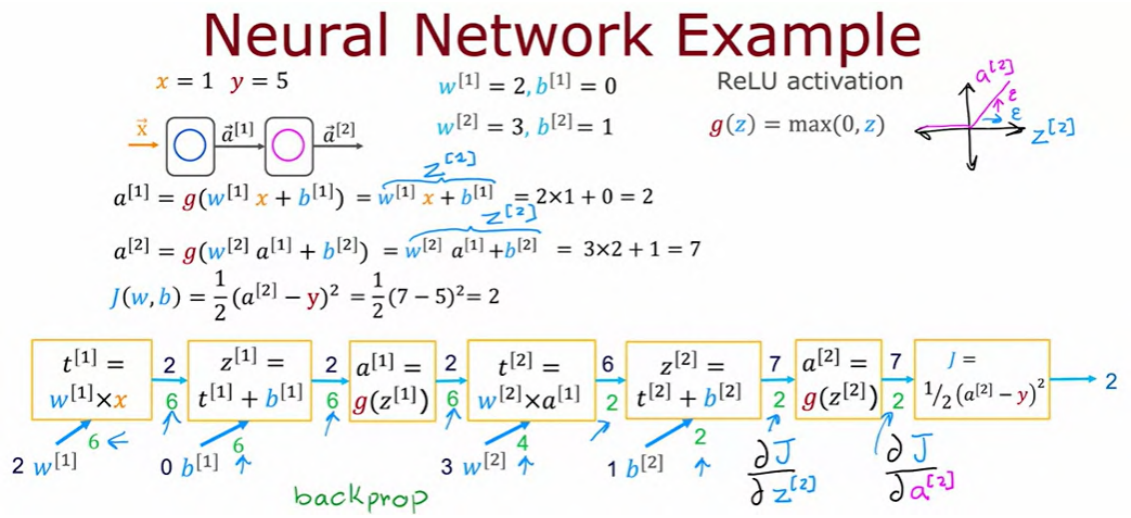


  - N : number of nodes ⟶ P : Number of parameters | N+P = total number of parameters used.

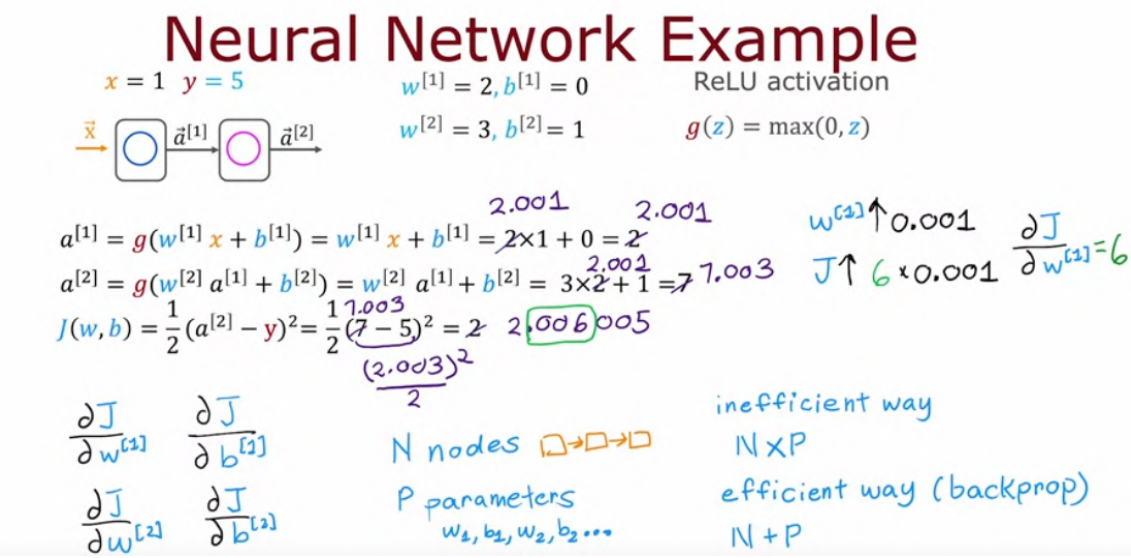
- Left → right : forword propagation is used for calculating the cost function and back propagation are used to calculate the derivatives (parameters of the weight.).

▼ How back propagation works in larger neural network ?

- implementation diagram :



- for N nodes and p parameters if only foreword propagations exists then the adjustments would take N times P, however the backpropagation helps it to reduce to N + P adjustments. Earlier times these derivatives are calculated manually.



▼ How to apply machine learning ? with diagnosis

▼ What if you incur with higher losses and undesirable results ?

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- adding polynomial features
- Increasing and decreasing Lambda

▼ How to evaluate and choose the model ?

- when we have one feature (meaning x1 and y) then we can plot and diagnose the model, if it is more than 2 then we invariable need a evaluation metric to diagnose the model. because now it has become a multidimensional model
- Splitting the data into train and test data set, fit the model on training data and analyze the prediction data on test data. Sometimes it makes sense to test the prediction on training dataset.

Train/test procedure for linear regression (with squared error cost)

Fit parameters by minimizing cost function $J(\vec{w}, b)$

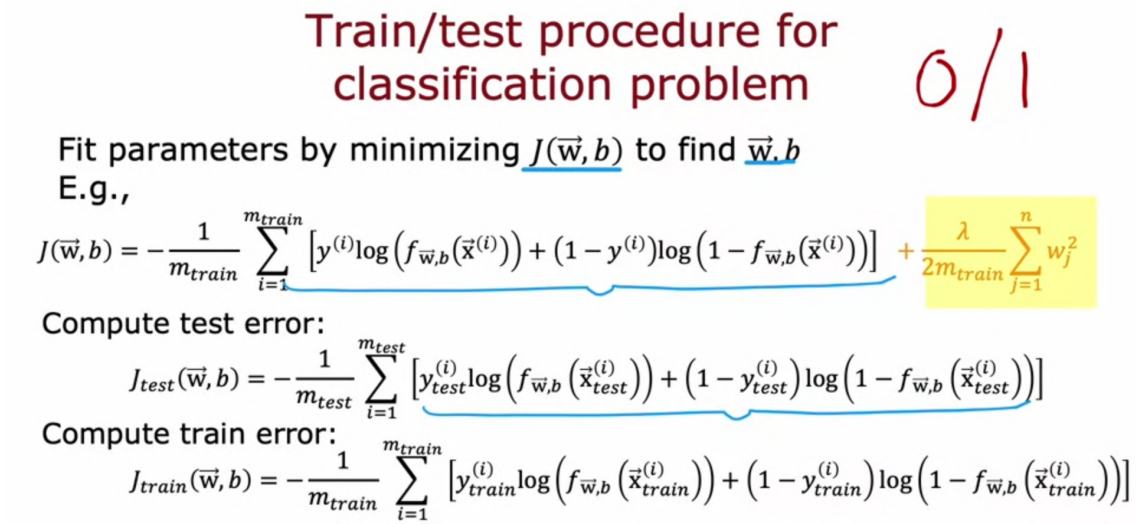
$$J(\vec{w}, b) = \left[\frac{1}{2m_{train}} \sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}(\vec{x}^{(i)}) - y^{(i)}\right)^2 + \frac{\lambda}{2m_{train}} \sum_{j=1}^{n} w_j^2\right]$$

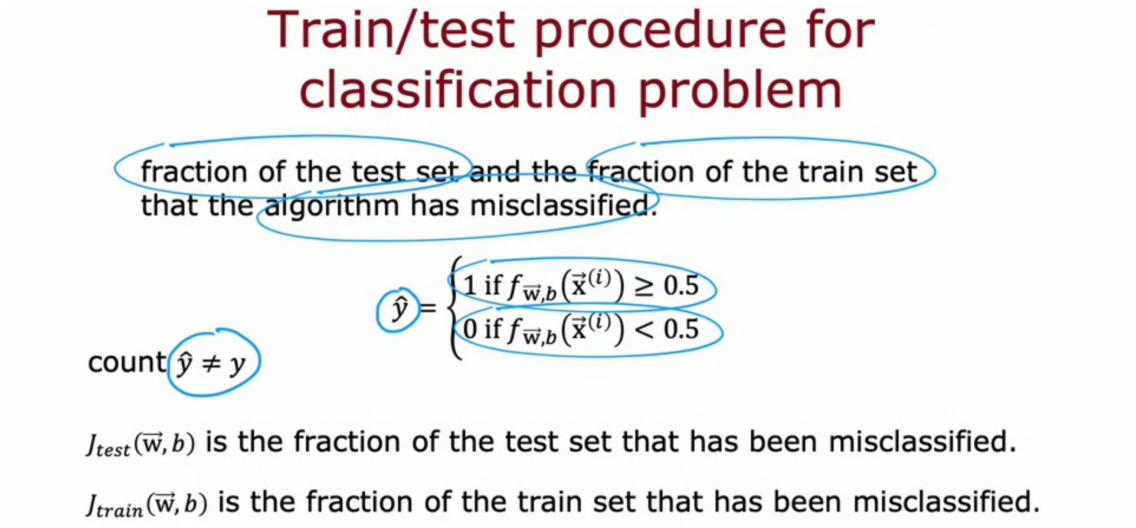
Compute test error:

$$J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[\sum_{i=1}^{m_{test}} \left(f_{\vec{w},b}\left(\vec{x}_{test}^{(i)}\right) - y_{test}^{(i)}\right)^2\right]$$

Compute training error:

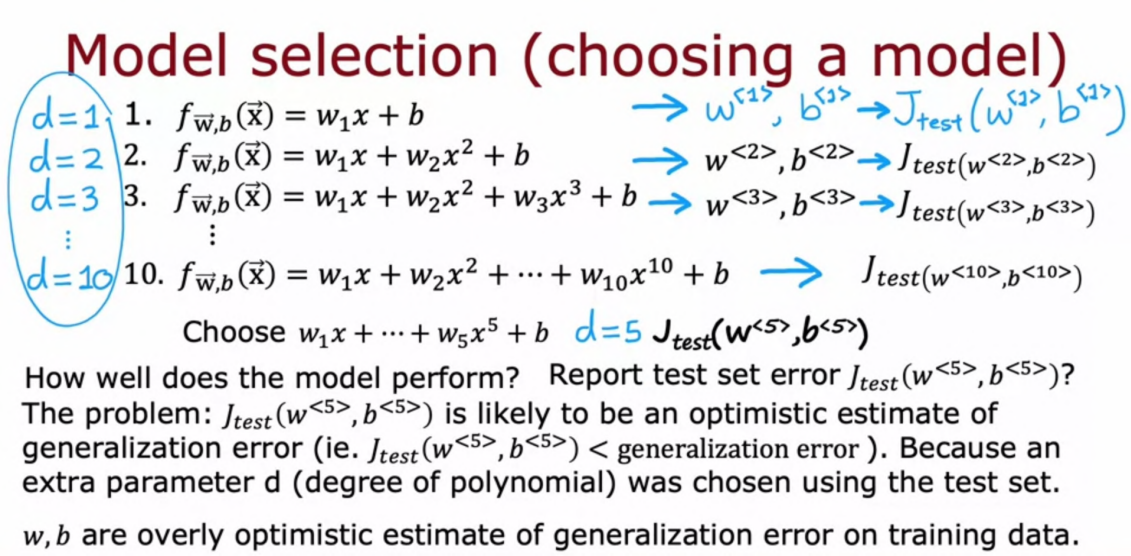
$$J_{train}(\vec{w}, b) = \frac{1}{2m_{train}} \left[\sum_{i=1}^{m_{train}} \left(f_{\vec{w},b}\left(\vec{x}_{train}^{(i)}\right) - y_{train}^{(i)}\right)^2\right]$$



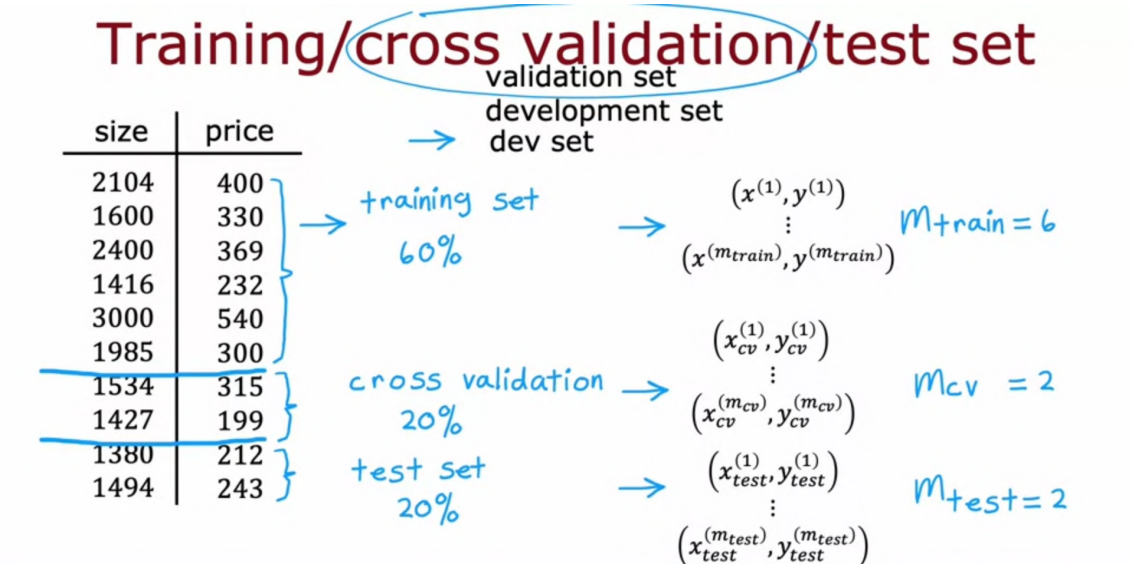


▼ How the training/test/validation dataset works ? why is it needed ?

- Features of good model
  - the test misclassification is lower and loss function is minimum
  - Difference between the train and test error will be lower.
  - They perform good on new dataset and
- There is flaw when you identify simply a model with minimum job cost function when you are iterating through multiple polynomials :



- So we are going to implement the validation dataset as a solution :

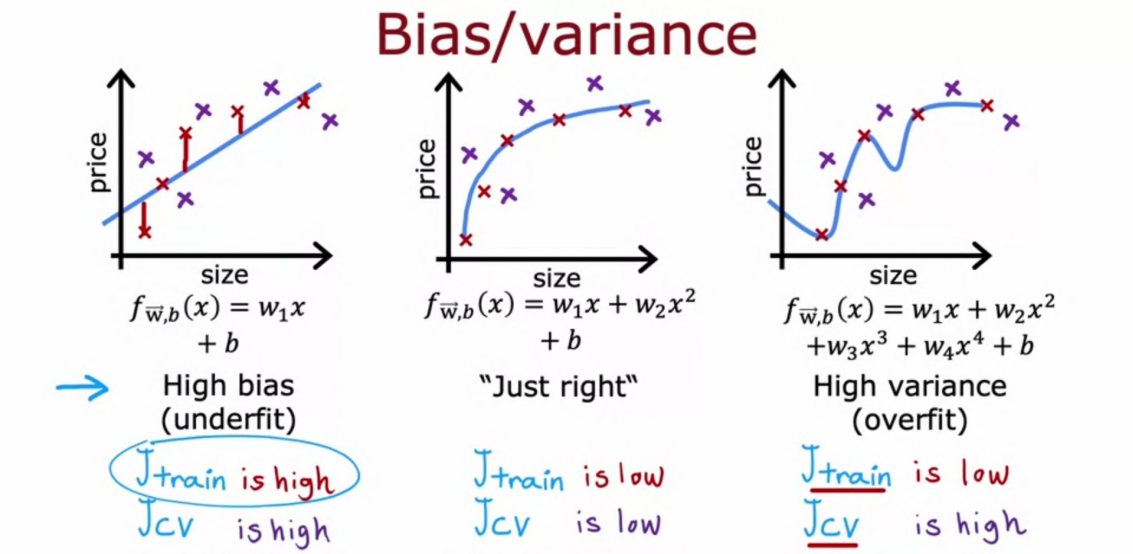


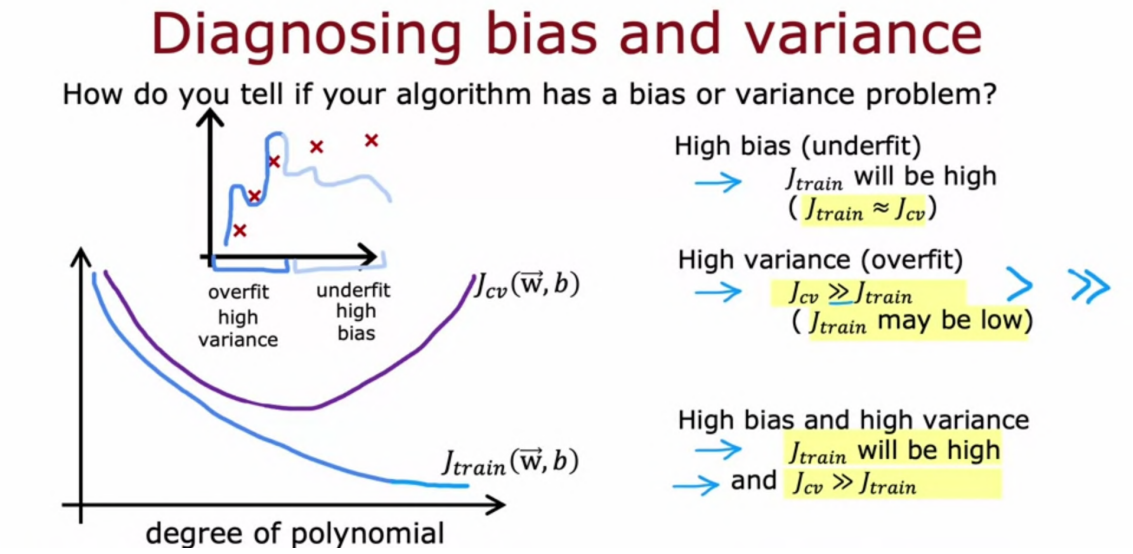
- **We will be choosing the model which has lower cost function in the validation dataset, then we apply the model to predict on testing data.**
- Why do we need validation dataset ?
  - Validation dataset is used to identify the fraction of misclassification that happens within the model.

- Model is not exposed to test dataset → so we can use test dataset to estimate the generalization power of the model.

▼ What is bias and variance ? and how to diagnose it

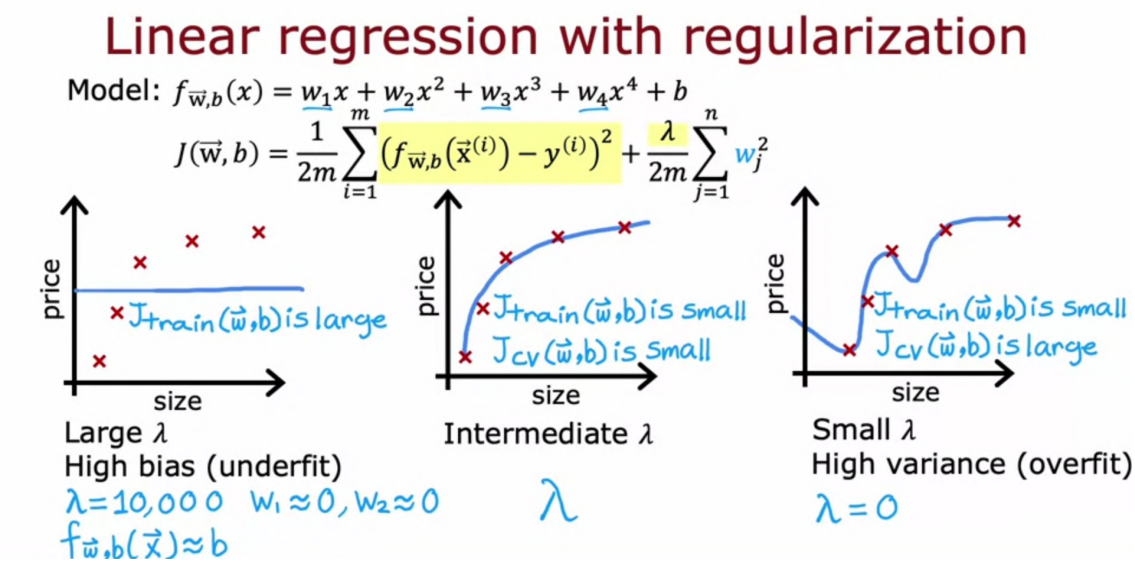
- Bias : does not even predict well on the training dataset, variance : even if the training dataset performs well → new dataset is unpredictable.

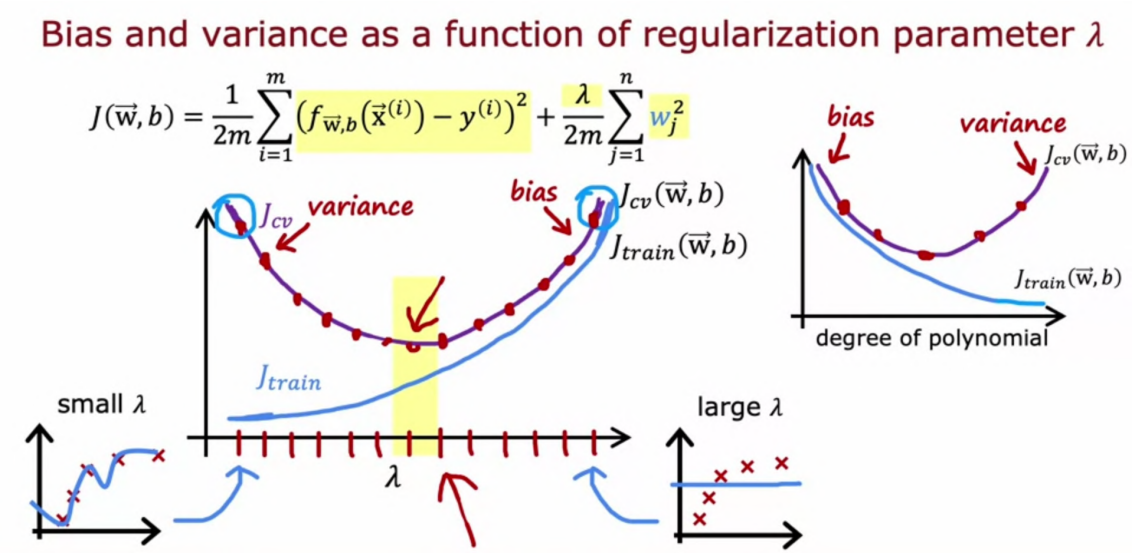




▼ How regularization helps in addressing the problems caused by bias and variance

- how regularization impacts the algorithm ? :





- Usually the training performance are expected to be lower than human level performance and we can use the human level performance as a benchmark for starting.
- Most of the cases, the training error should be less than the validation error. High training error would the bias is high.
- What is the level of error you can reasonably hope to get to ?

    - Human level performance : how much error % is to be expected from a human.
    - Competing with the algorithms performance
    - Guessing based on subject expertise.

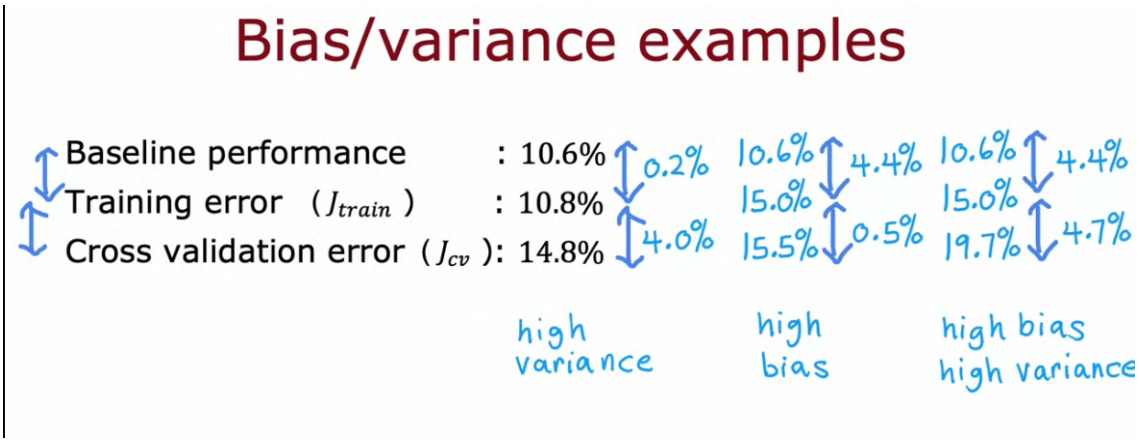
▼ How to set benchmark the performance of model ?

▼ Benchmark the model perfomance

- Check the human level performance
- prior experience
- maximum performance gotten by competing algorithm on same applications
- Getting to know others performance

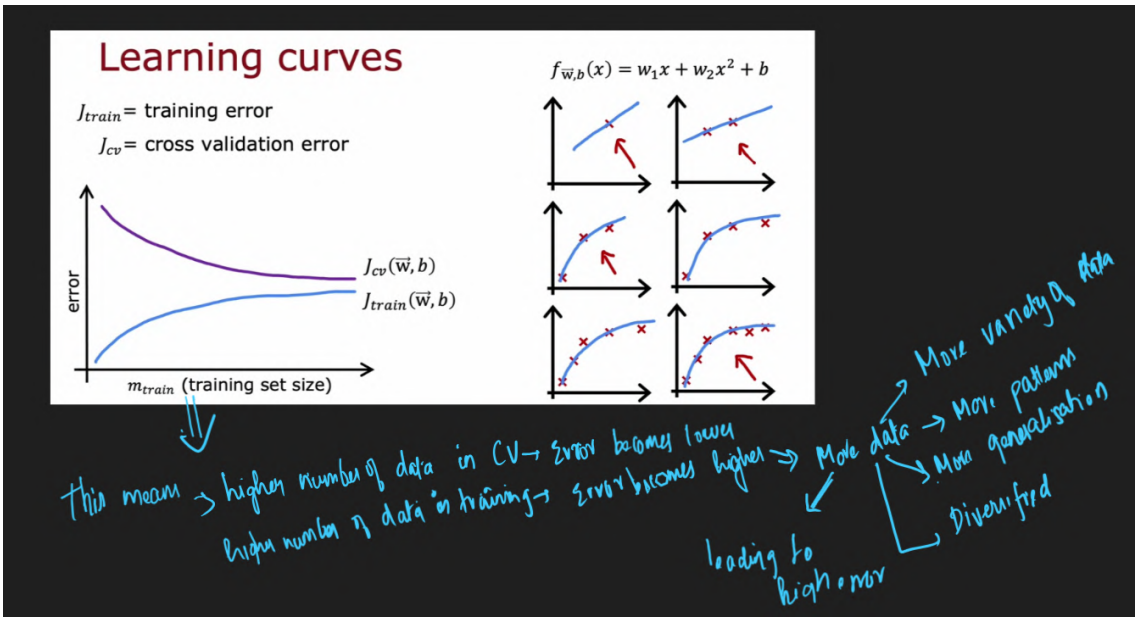
▼ How to evaluate the performance

- Difference between the baseline (benchmark : like human level performance or any competing algorithm) and validation error explains the bias
- Difference between the validation and test explains the variance
- Thus we can address the overfitting and underfitting issue.

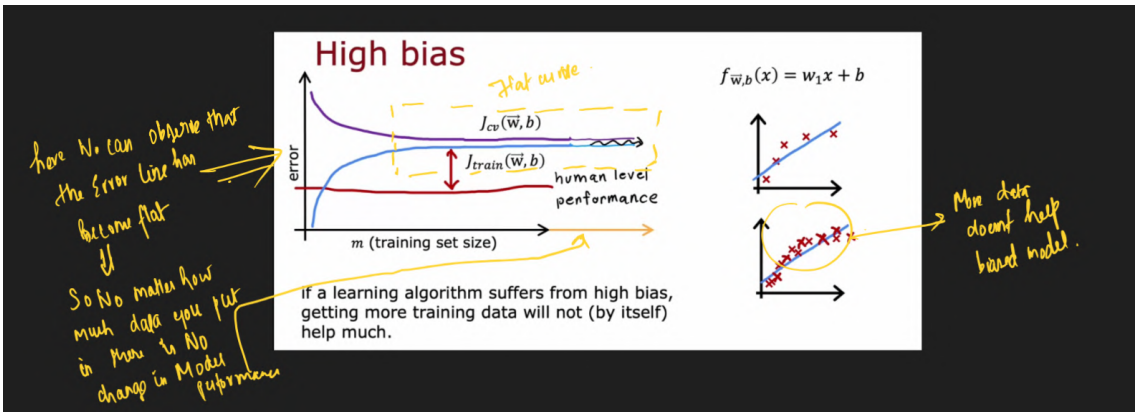


▼ How to learning curve to establish an baseline model ?

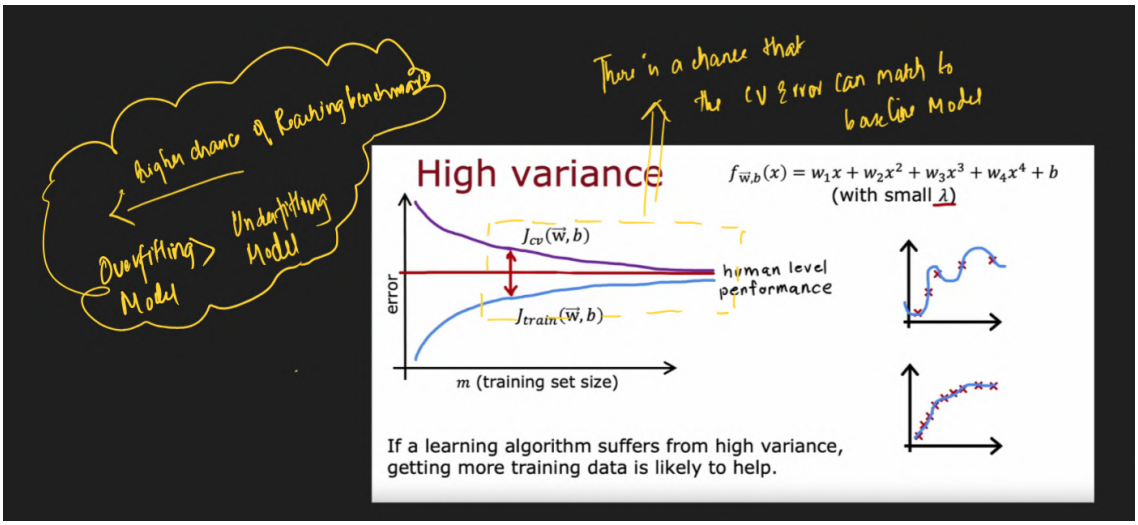
-



- When the model is biased as seen below : no matter how large data imputed into the model it is almost not possible to improve its performance, hence when we identify bias with baseline performance then it should be addressed first. Also ensure to include diversified dataset points into the model.



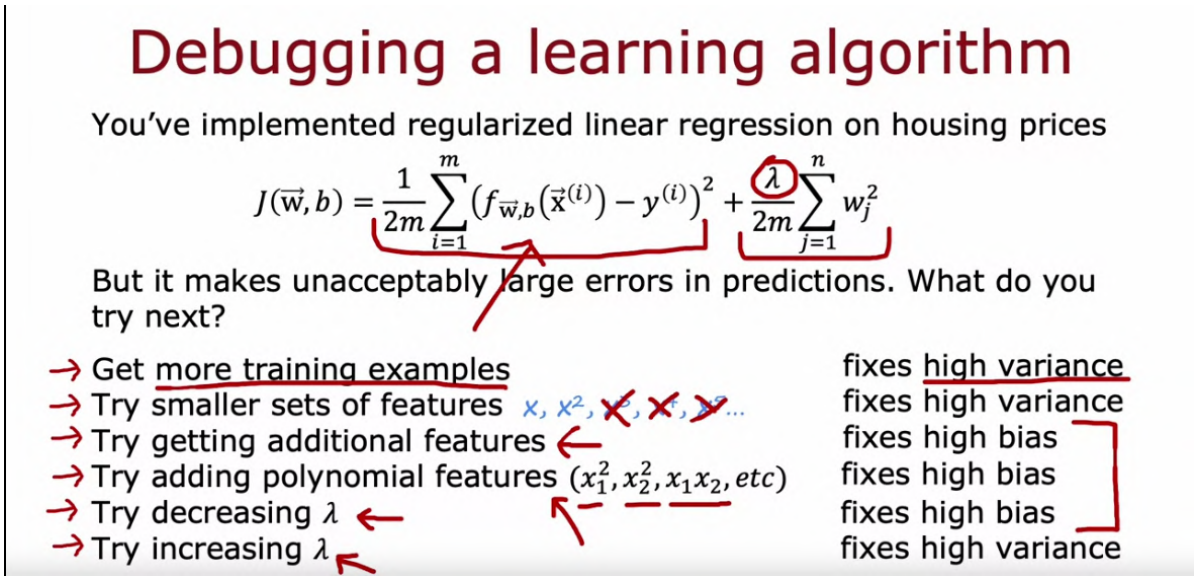
- High variance model (mentioned below): this model on exposing to more and more data, there is probability where can converge with baseline model. Hence the more prone model is the biased model where adding data is not sufficient to address the issue.



▼ How to proceed further while encountering the bias and variance ?

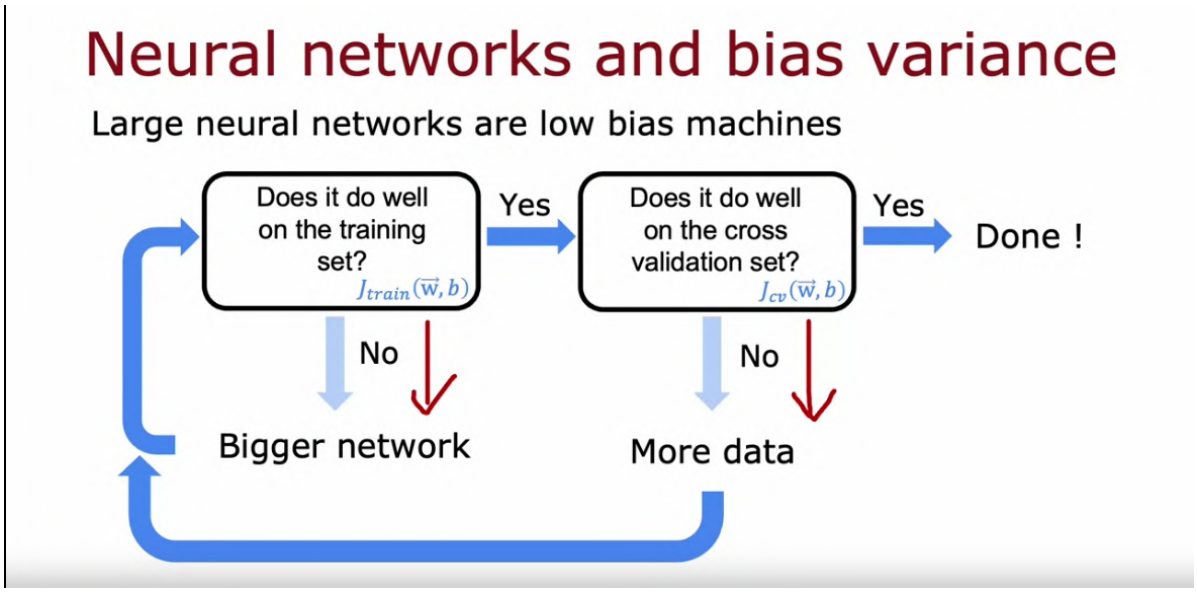
Overfitting problem	**Underfitting problem**
Get more training examples	Try adding more features
Try increasing lambda : more regularization	Try adding more polynomial features
Try smaller sets of features	Try decreasing the lambda : fewer regularization is achieved

•

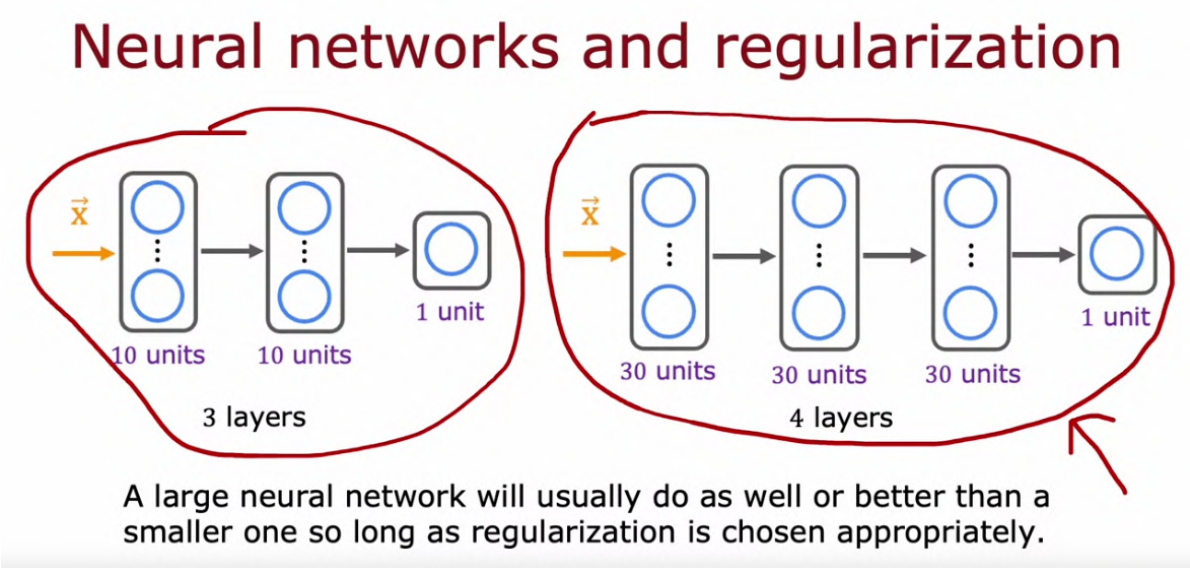


▼ How the bias and variance concept applied to neural network ?

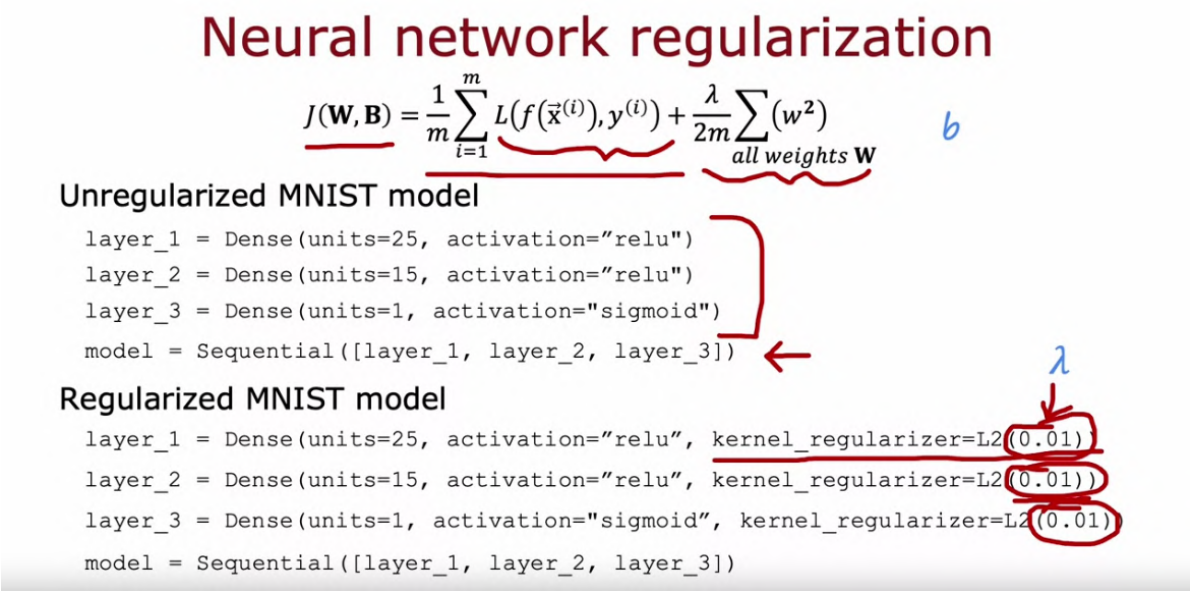
- Large neural network has lower bias as they may be prone to overfitting.
- Very large neural network could be computationally expensive.
    - Cons
        - More data can be a limiting factor
        - Needs GPU and other accelerators to run the larger neural network



- A idle process would be get a overfitting model with larger neural network and exposing them to new data as well as regularize them to achieve human level data.

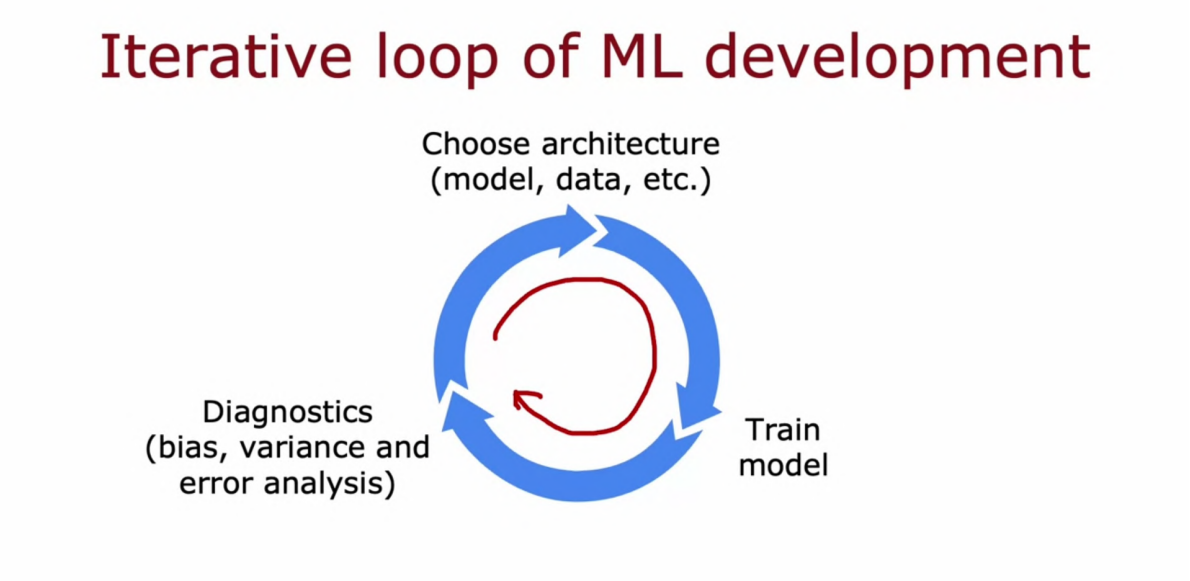


- How regularization applied to neural network applied below ?



▼ How to develop a machine learning system ?

- Iterative loop of development.

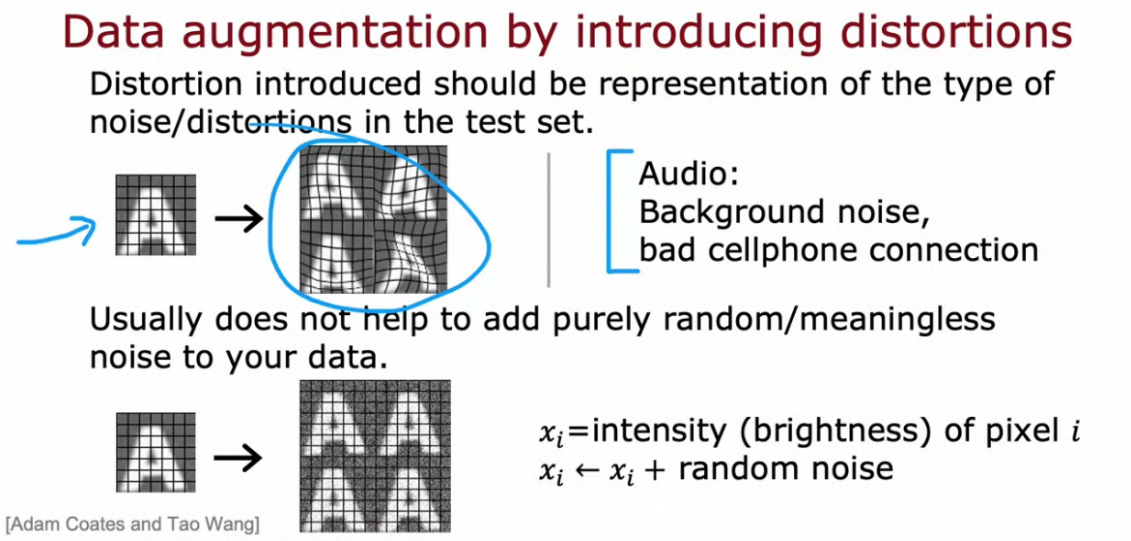


▼ Error Analysis : How does it look like ?

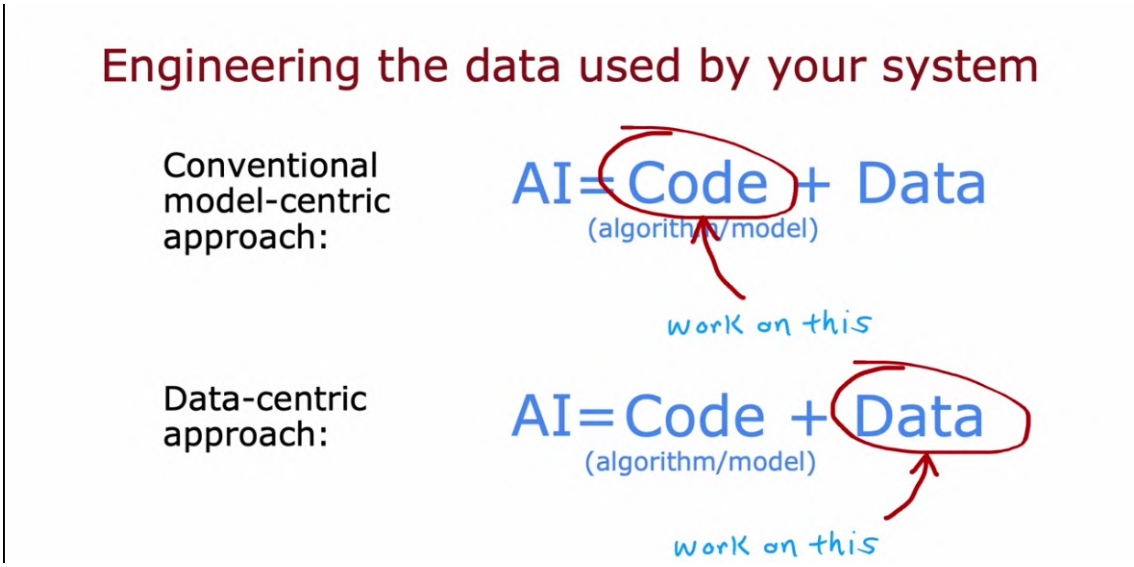
- Manual diagnosing the errors will be small and tiny impactful activity compared to automatic and computers way.
- Applicable where the bench mark is human level performance meaning : applicable to tasks that human can do better. Prediction may seem better with machine rather than human

▼ How to overcome the problems in machine learning :

- Adding data : on error analysis, if any specific kind of information is misclassified then the specific type of data can be added for betterment of the model.
- Data augmentation for image classification and audio recognition for adding data : new training examples are created by performing few minor modification(rotation , mirroring, introducing new lines within the image) to generate new training data set. Adding more background noise to the existing voice : Data augumentation.
- Never add random noise or meaningless noise.

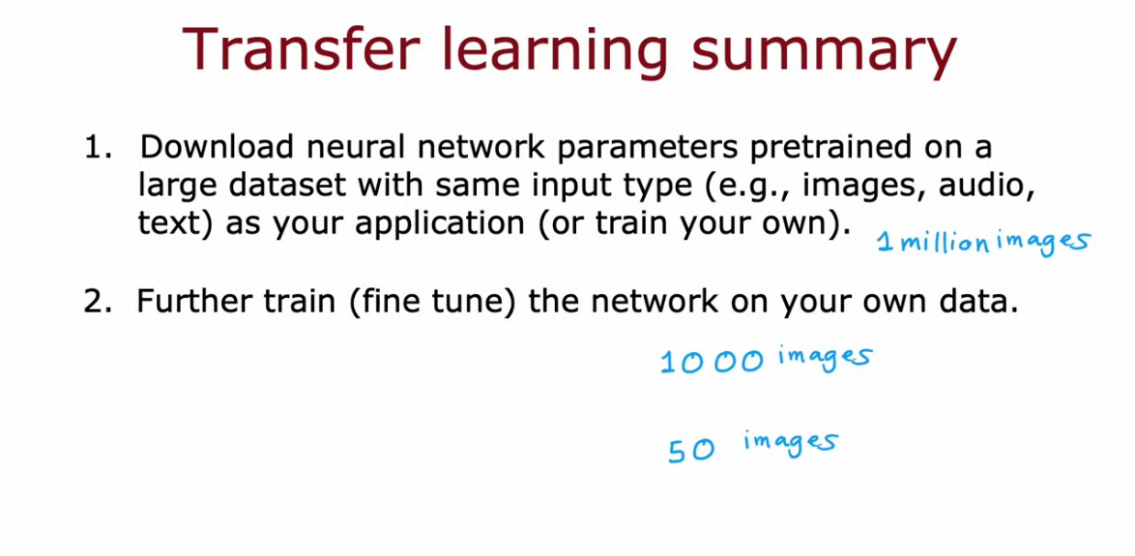


- Data Synthesis(higher application in computer vision) : Creating brand new examples or training data meaning existing training data is untouched. Manually can be created by work or Image generating LLM's could be employed here.
- **Our approach must foster : Data centric approach as the existing codes are well written and needs higher expertise in improving its approach**

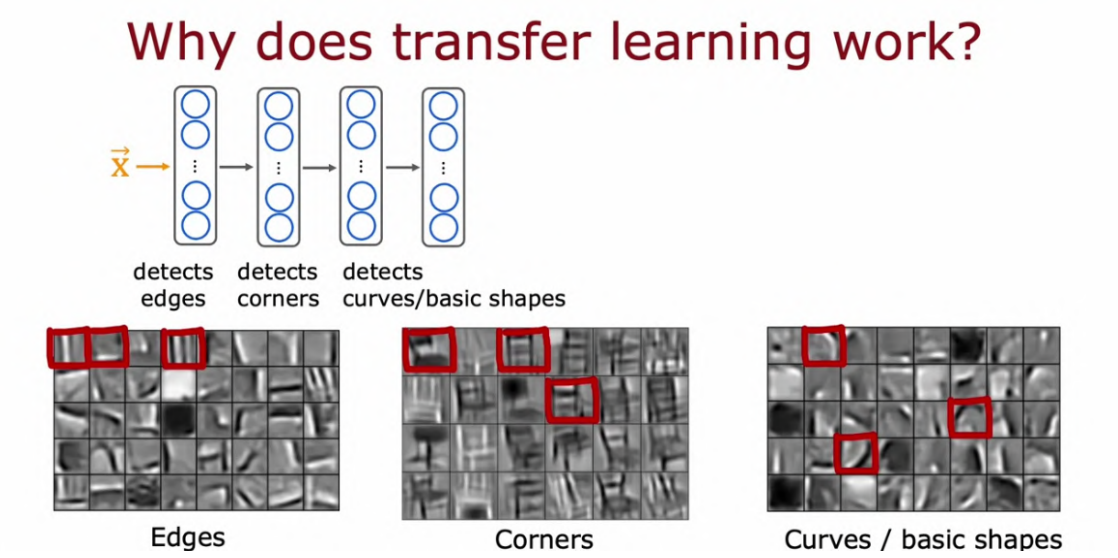


▼ What is transfer learning , why should be augmented into our existing approach ?

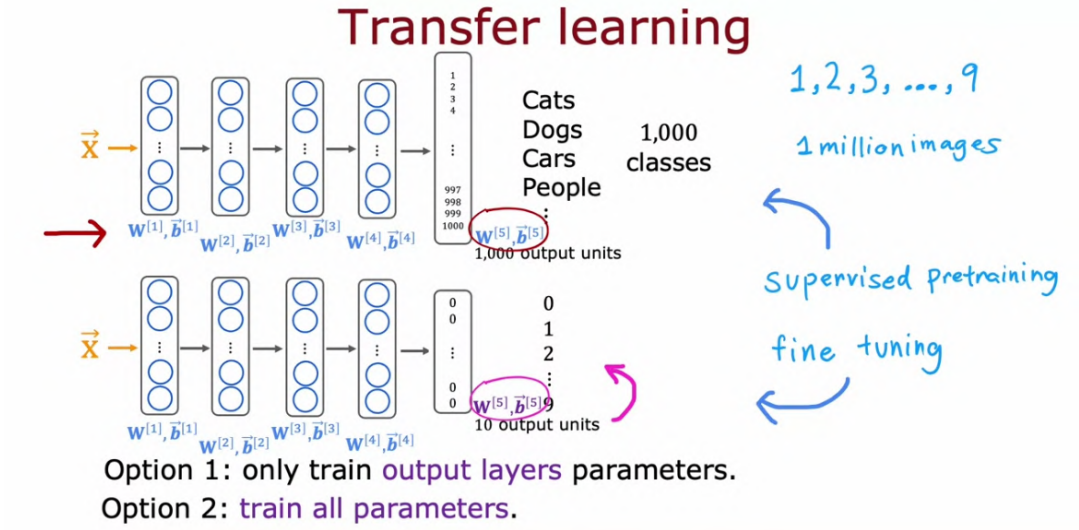
- Cannot be applied to all, however if possible then it could be powerful



- When we don't have much of training data or when we are in need of adding more data.
- Transfer learning allows to download and use the best model available on internet as they might have already researched and proved to provide better results for any application.
- The neurons with neural layer tries to learn the detection skills rather than detecting output hence, this allows the user to use the same model for different use cases of the same application(image NN on Image data not on audio data)

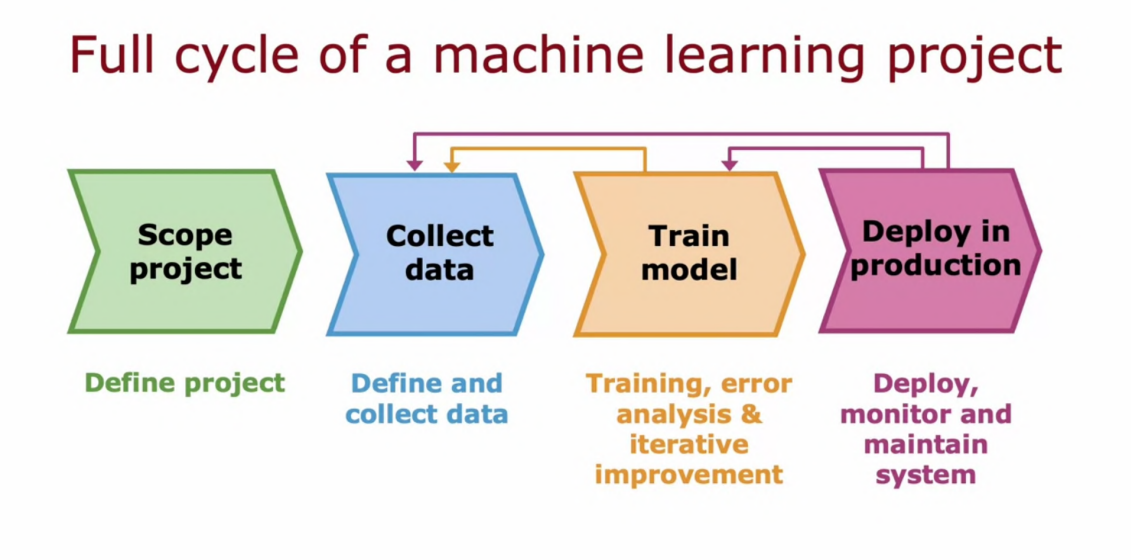


- **In transfer learning the output layer is replaced rather than adding a layer**
  - reason for not adding a layer
    - Computationally heavy
    - Net impact would be lower
- Options in transfer learning
  - option 1 (Supervised pre training): training only the output layer, meaning the weights and biases of previous layers remains untouched.
  - option 2 (fine-tuning): training all parameters > training all weights and biases of all layer.



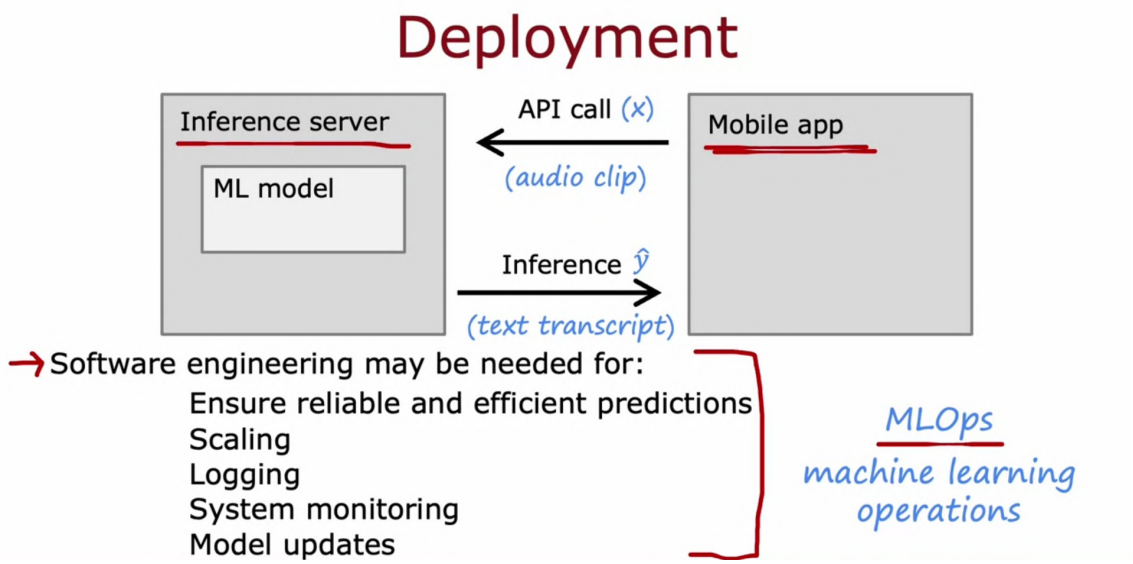
▼ How full cycle of a machine learning project would look like ?

•



→ Few software requirements for deployment mentioned below.

•



▼ Ethics in ML

-

Bias

Hiring tool that discriminates against women.

Facial recognition system matching dark skinned individuals to criminal mugshots.

Biased bank loan approvals.

Toxic effect of reinforcing negative stereotypes.

Adverse use cases

Deepfakes

Spreading toxic/incendiary speech through optimizing for engagement.

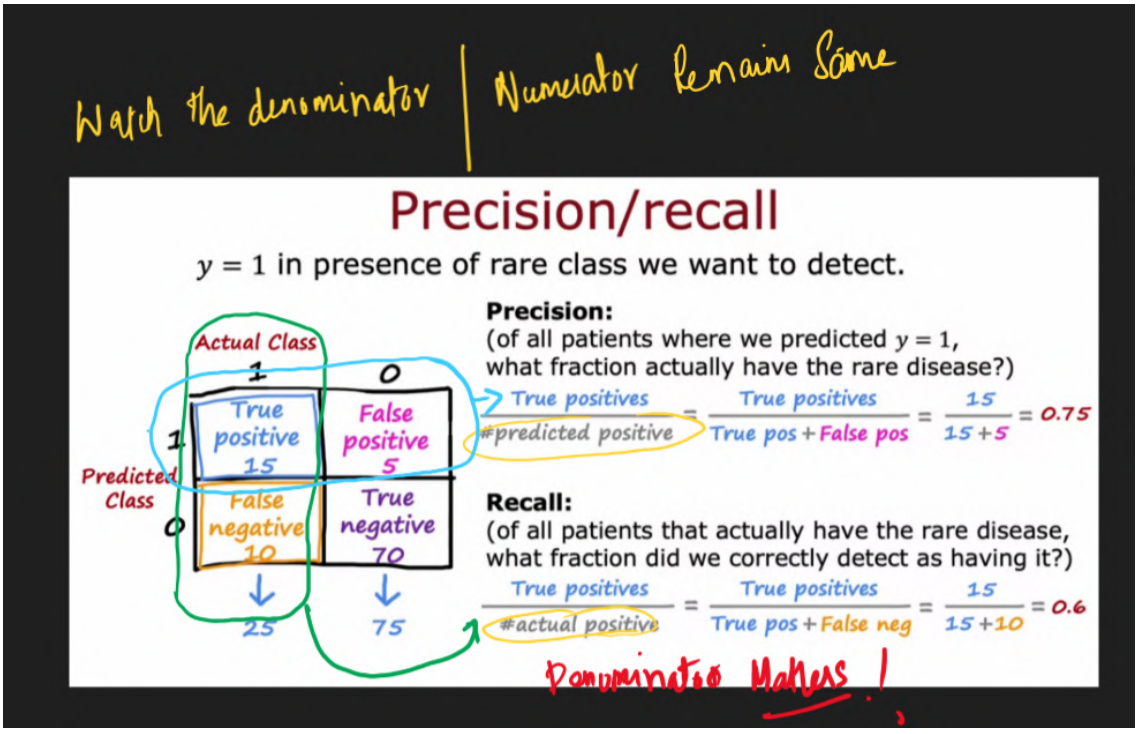
Generating fake content for commercial or political purposes.

Using ML to build harmful products, commit fraud etc.

Spam vs anti-spam : fraud vs anti-fraud.

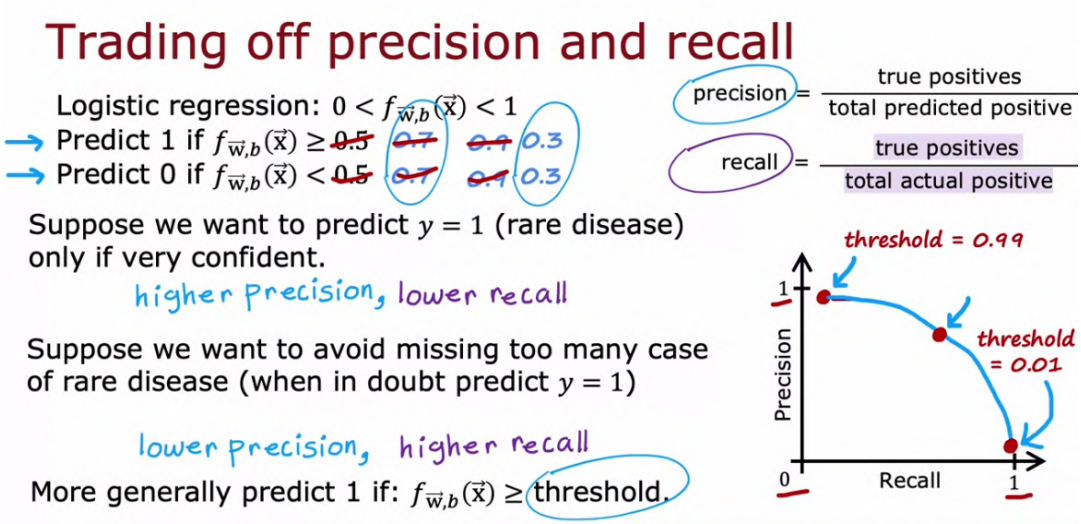
▼ How to measure Error Metrics for skewed datasets ?

- Precision/Recall : In precision(speaks about model), if a person is diagnosed as sick there 75% chance that he is sick. In recall(speaks about population), the model is capable to predict 60% of total sick persons.

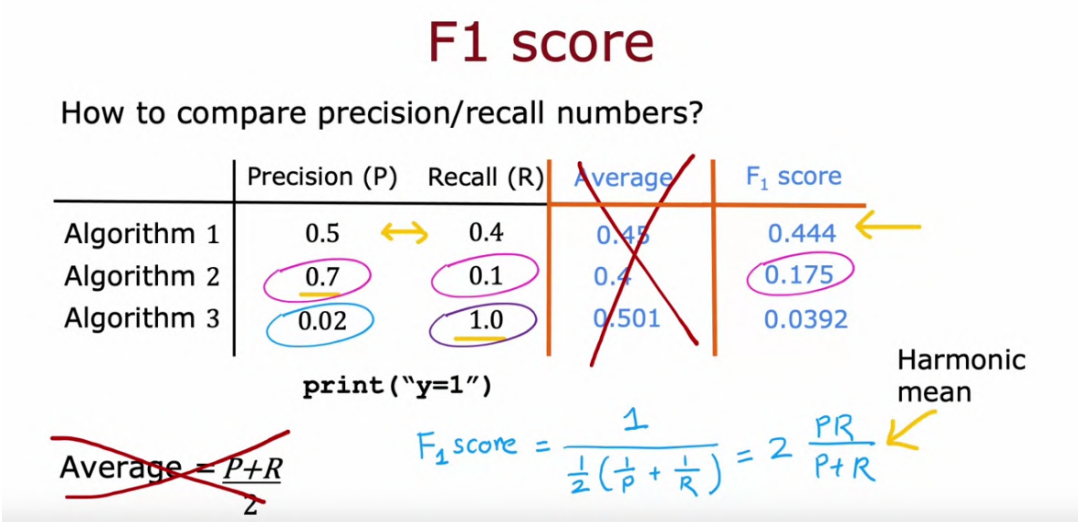


- High precision & High recall is not always possible.

- If the threshold for classification is increased which means that precision becomes higher, however this lowers the recall which means most of the patients are left out instead of identification.



- It is not always possible to manipulate the threshold as it becomes difficult during cross validations. Having two evaluation metrics for a model becomes uneasy to make decision if we multiple models to evaluate.
-
- F1 Score : automatically supports for selecting optimum precision and recall.



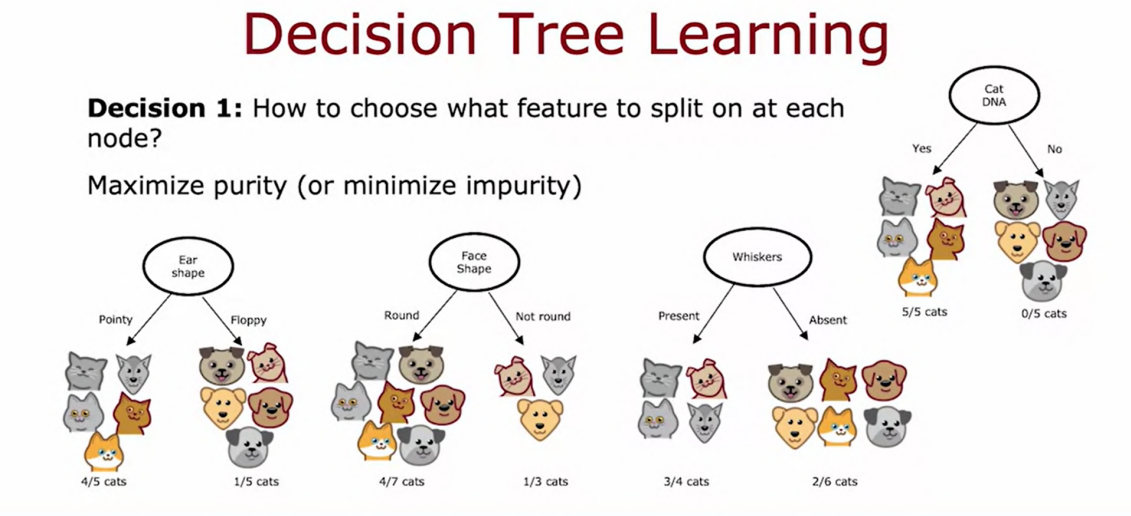
▼ Powerful algorithms implementation ?

▼ Decision tree

- root node : Top most node on the tree
- leaf node : node other than root node
-

▼ How to select the root node, how is it calculated?

- The root node feature is selected based on maximizing purity or minimizing purity.
  - what is purity here?
    - If number of classes within the subdivided group is 1 then it is high purity else it leads to impurity.
    - For a high purity leaf, leaf will have only one class.
- The right top corner explains what purity is →



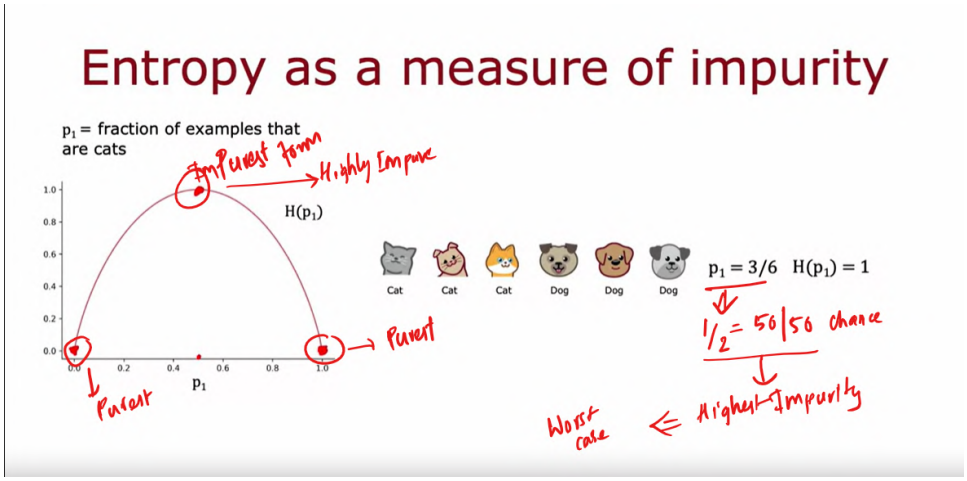
- How do we know when to stop the decision tree?
  - When purity is 100% meaning a leaf will have only one class
  - When tree exceeds the defined maximum depth it stops, as we know decision trees are greedy algorithm.

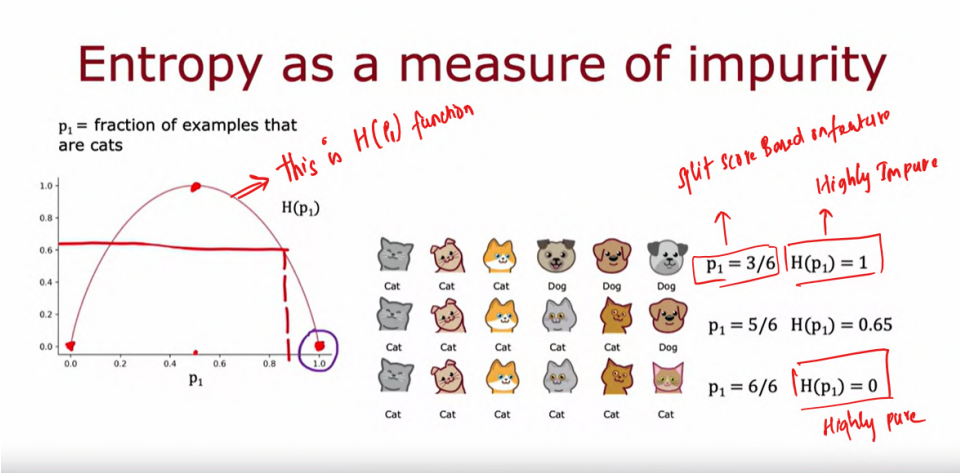
- When the improvements in purity of leaf is less than the defined purity on the code then the algorithm stops
- If the number of examples in any given node becomes less than the minimum defined example number then the algorithm stops.
-

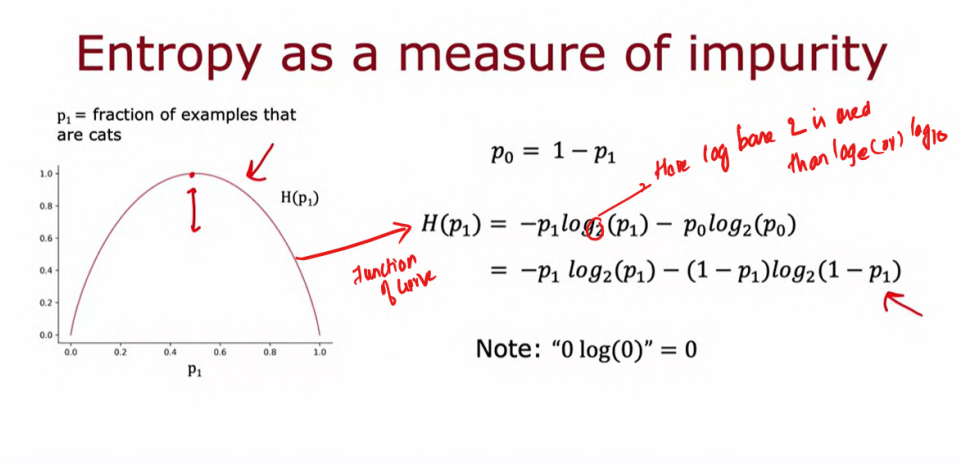
▼ How the split happens in the leaf node ?

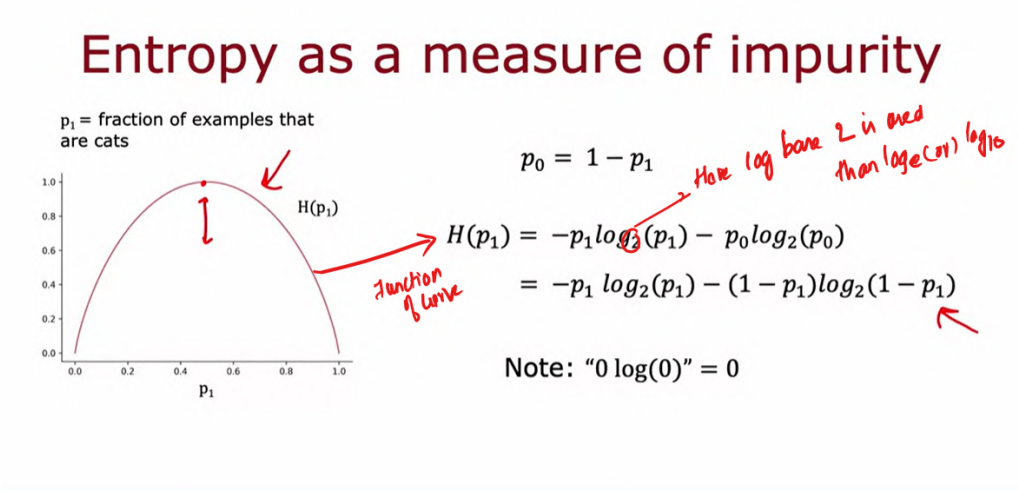
▼ how to measure impurity?

- entropy is a measure of impurity: if high entropy then high impurity, if low entropy then high purity
- Highest purity is when a leaf has only one class, highest impurity is when there are more than one class which are **equally distributed**
- what is entropy then?

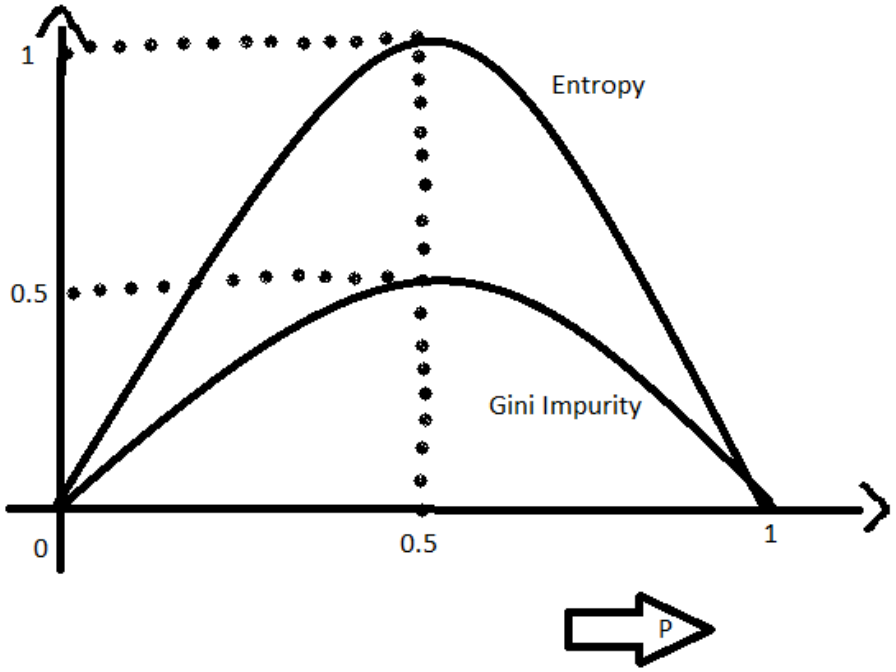








- Other functions for measuring impurity of decision tree :

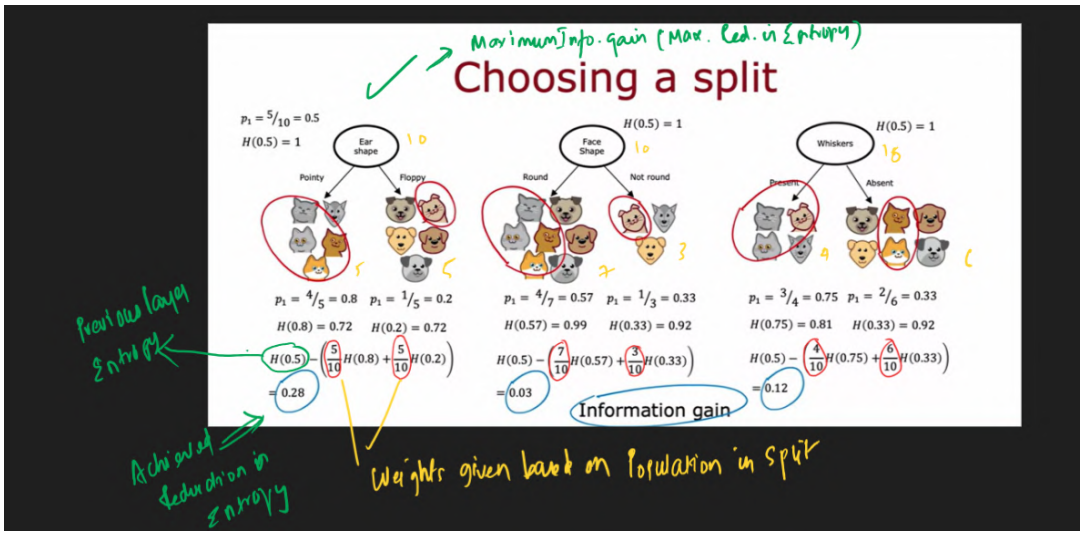


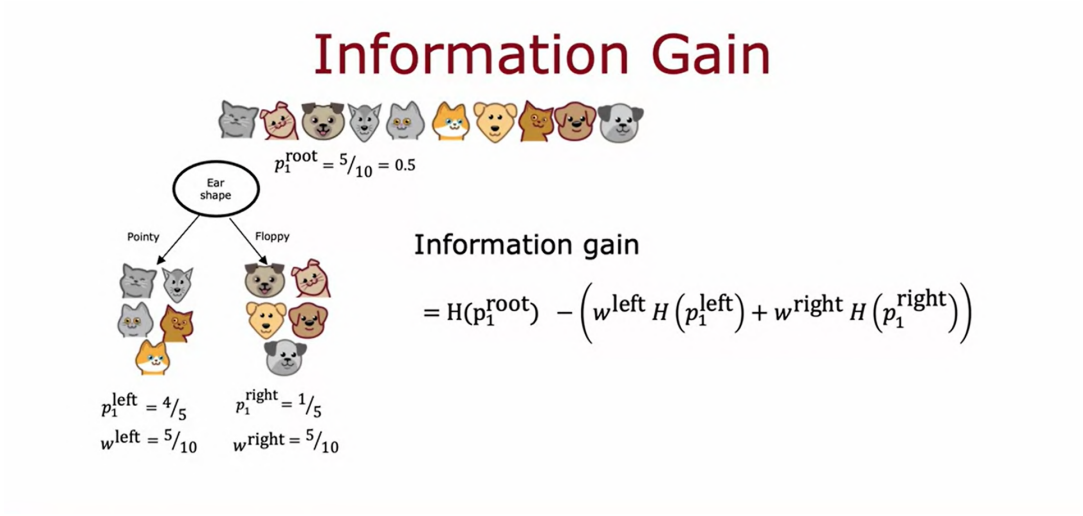
$$E(S) = \sum_{i=1}^{c} - p_i \log_2 p_i$$

$$Gini(E) = 1 - \sum_{j=1}^{c} p_j^2$$

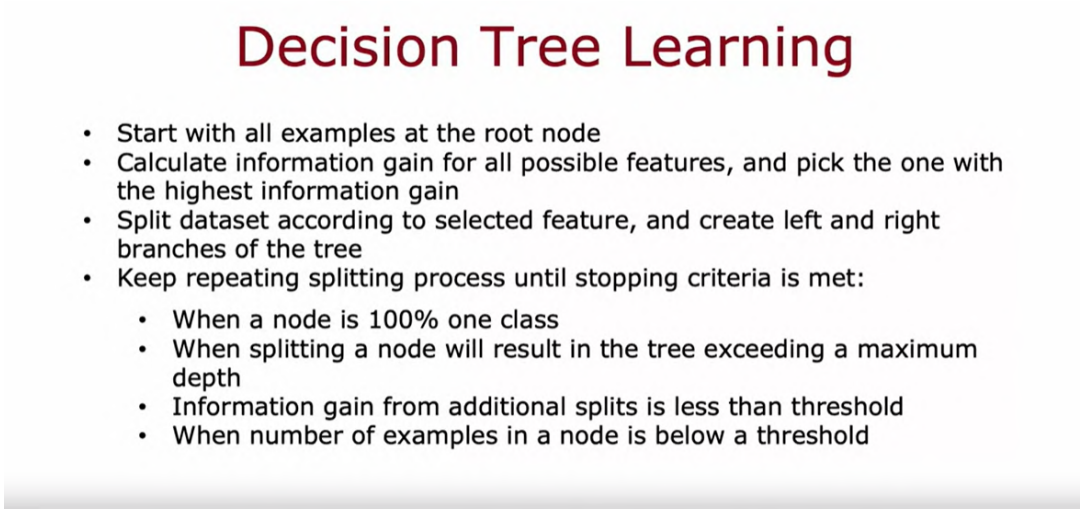
▼ How to choose the split based on information gain ?

- Information gain: The maximization or minimization of entropy (purity measuring function)
-

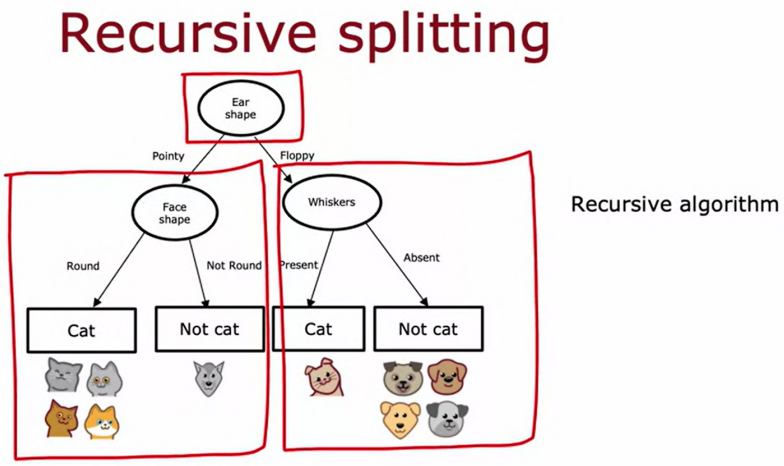




▼ How to build a decision tree ?



- Recursive Algorithm: A specific algorithm is used from top to bottom of a tree, repetition of algorithm at scale.

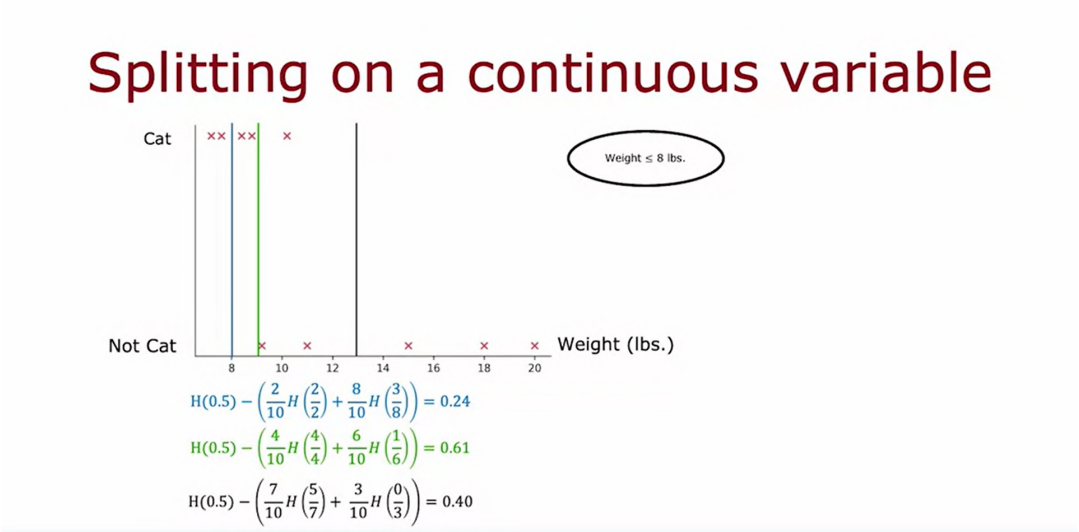


- How to select the depth of Decision tree
  - Select based on open source library
  - higher depths > larger tree > overfitting case might be.
  - Try validation techniques : for selecting the depth of tree, however open source libraries may be much better.
  - Use information gain to limit the growth of tree and number of examples within last leaf node.

▼ How to handle categorical features ?

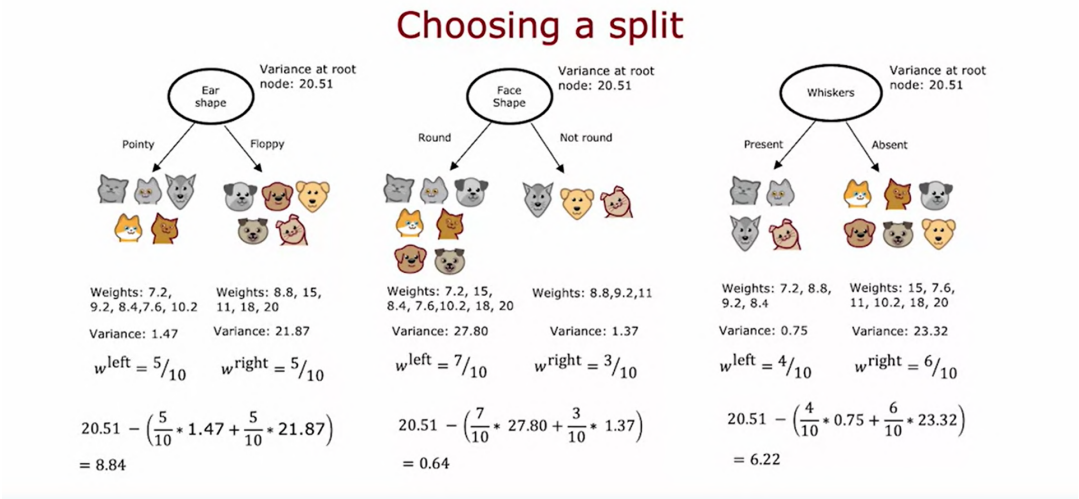
- What will happen, if it has more than 2 categories in the same column → seems challenging for decision tree.
- So, we will handle by using one-hot encoding.

▼ how decision tree works for continuous range for values ?



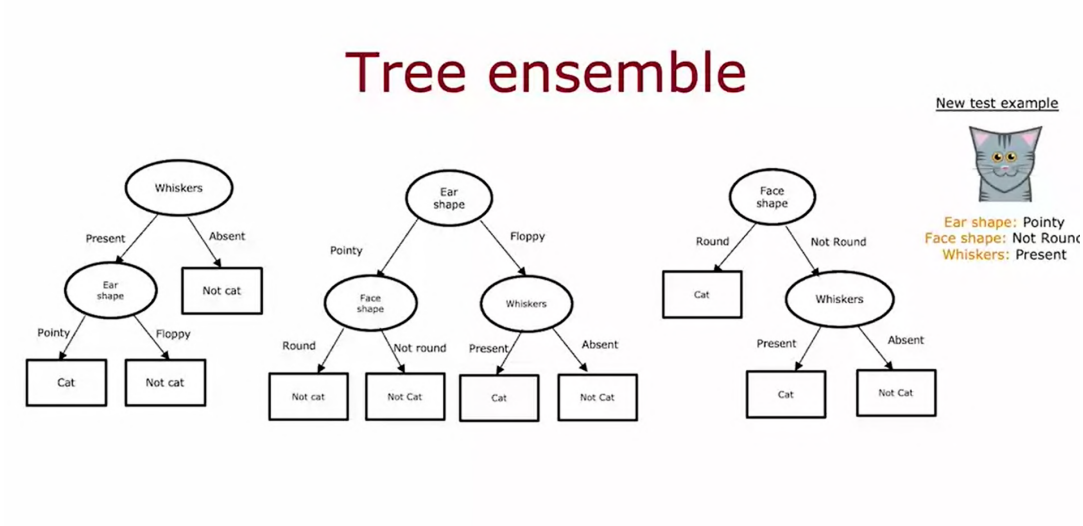
▼ how the generalization happens with decision tree ?

- we try to reduce the variance incase if it is a regression problem. Since output is continuous variable , we try to reduce the variance in each level of trees:



▼ What is ensemblance of decision tree

- combining multiple decision trees and consolidating the result.
-



▼ sampling with replacement

- example means rows with all the features in.
- higher number of tree makes model generalized and the performance remains same.
- This is also called bagged decision trees.
- For improving the variety of decision trees to be made :
  - We can take sample with replacements (subset of rows)
  - we can randomizing the feature selection (subset of columns)
    - for higher number of features(columns) :
      - if n number of features are available in total, we select k number of feature to create decision tree.
      - if the number of features are too high then : k = n^(1/2) → used generally. : this is called Random Forest algorithm.

Sampling with replacement

Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	1
Floppy	Not round	Absent	0
Pointy	Round	Absent	1
Pointy	Not round	Present	0
Floppy	Not round	Absent	0
Pointy	Round	Absent	1
Pointy	Round	Present	1
Floppy	Not round	Present	1
Floppy	Round	Absent	0
Pointy	Round	Absent	1

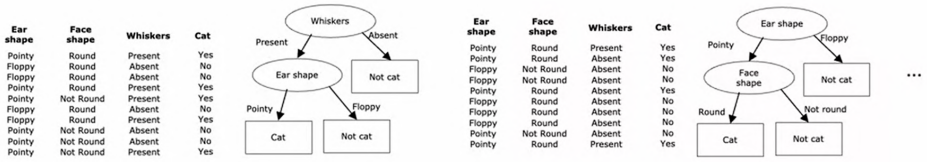
Generating a tree sample

Given training set of size m

For $b = 1$ to B:

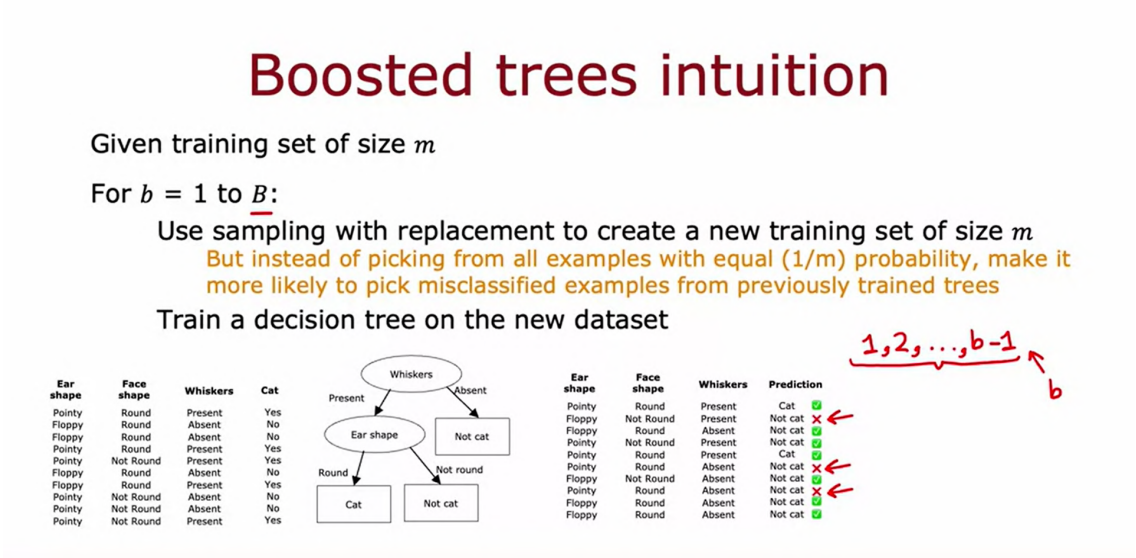
Use sampling with replacement to create a new training set of size m

Train a decision tree on the new dataset

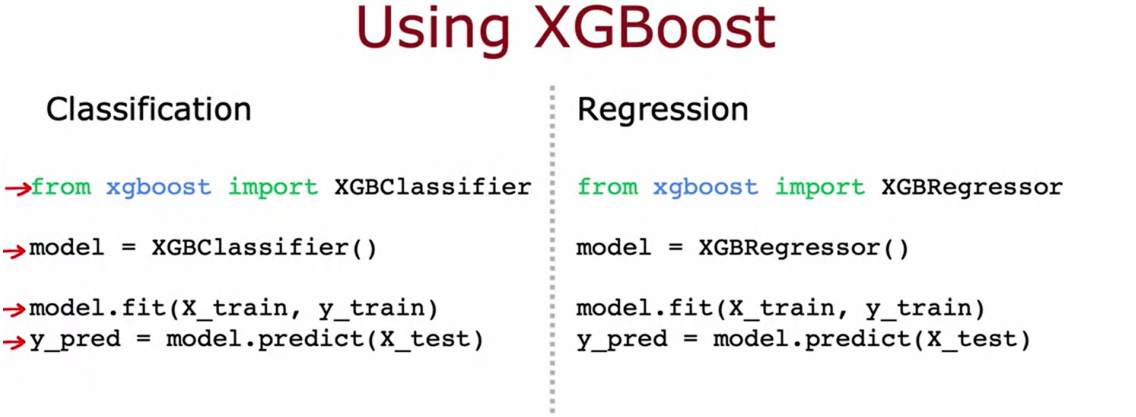


▼ How does the XGboost work ?

- Extreme gradient boosting
- Has built in regularisation to reduce the problem of overfitting
-



-



▼ Decision tree vs neural networks

-

Decision tree	Neural networks
Works best for structured data	Works well with all types of data and mixed data as well
Unsuited for unstructured data	Multiple models can be threaded together.
Faster training	May be slower based on the application

Smaller decision trees are interpretable, but does not work with ensemblances.	Has transfer learning, which is quite efficient.

▼