Types of Regularization Techniques

There are two main types of regularization techniques: L1(Lasso) and L2(Ridge) regularization

1) Lasso Regularization (L1 Regularization)

In L1 you add information to model equation to be the absolute sum of theta vector (θ) multiply by the regularization parameter (λ) which could be any large number over size of data (m), where (n) is the number of features.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) + \frac{\lambda}{m}\sum_{j=1}^{n} |\theta_j|$$

2) Ridge Regularization (L2 Regularization)

In L2, you add the information to model equation to be the sum of vector (θ) squared multiplied by the regularization parameter (λ) which can be any big number over size of data (m), which (n) is a number of features.

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} Cost(h_\theta(x^{(i)}), y^{(i)}) + \frac{\lambda}{2m}\sum_{j=1}^{n} \theta_j^2$$

Advanced Learning Algorithms - Course 2

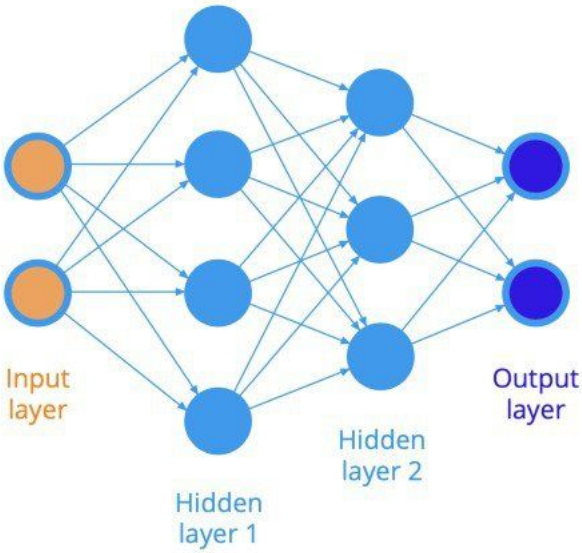
NEURAL NETWORKS

- Learning Algorithms that mimic the Human Brain.
- Speech Recognition, Computer Vision, NLP, Climate Change, Medical Image Detection, Product Recommendation.
- Just like human neuron, neural networks use many neurons to receive input and gives an output to next layer of network.

- As amount of data increased the performance of neural networks also increased.
- Layer is a grouping of neurons, which take input of similar features and output a few numbers together.
- Layer can have single or multiple neurons.
- If the layer is first, it is called Input Layer
- If the layer is last it is called Output Layer
- If the layer in middle, it is called Hidden Layer
- Each neurons have access to all other neurons in next layer.

Activations (a)

- Refers to the degree of high output value of neuron, given to the next neurons.
- Neural Networks learn it's own features. No need of manual Feature Engineering.



Demand Prediction of a Shirt?

- affordability, awareness, perceived quality are activations of each neurons.
- It is basically a Logistic Regression, trying to learn much by it's own.
- activation, a = g(z) = 1 / (1 + e ^ -z)
- Number of layers and neurons are called Neural Network Architecture.

- Multi-layer perception is also known as MLP. It is fully connected dense layers.

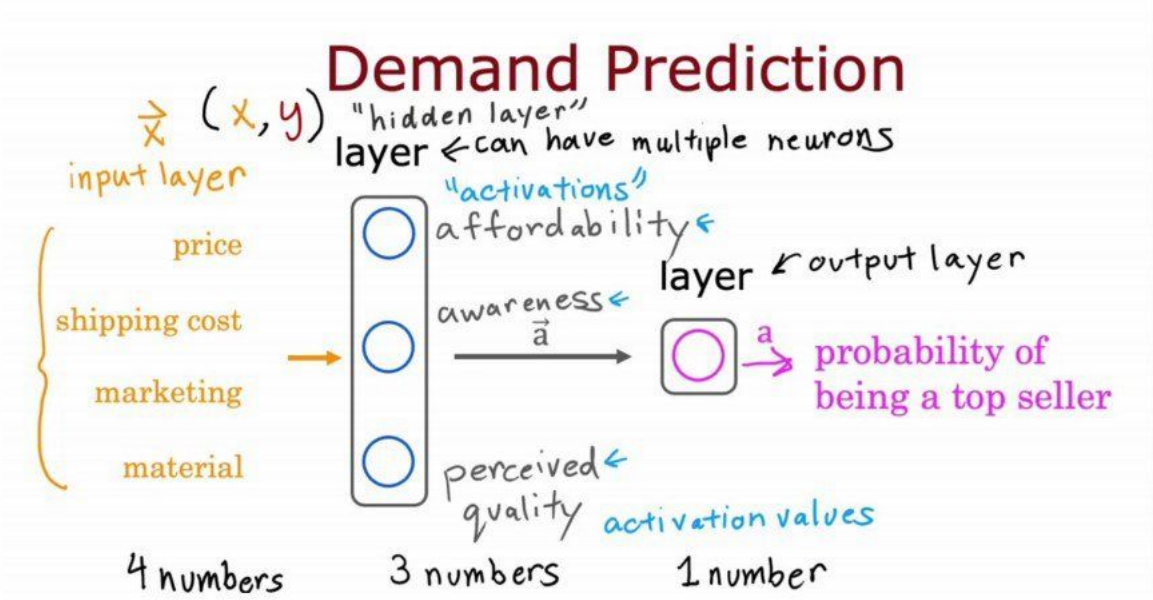
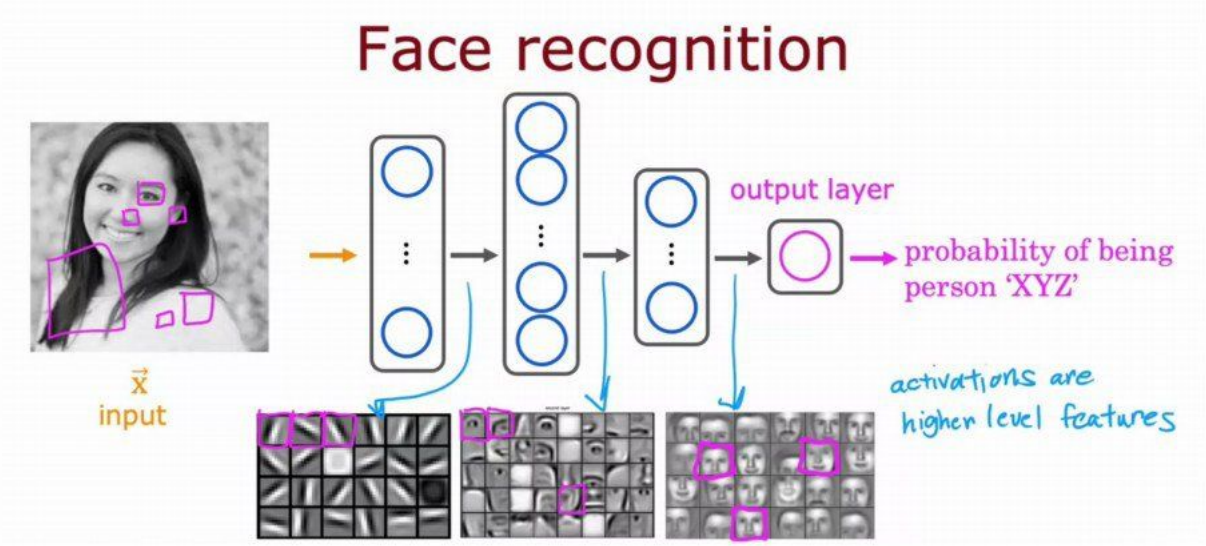


Image Recognition

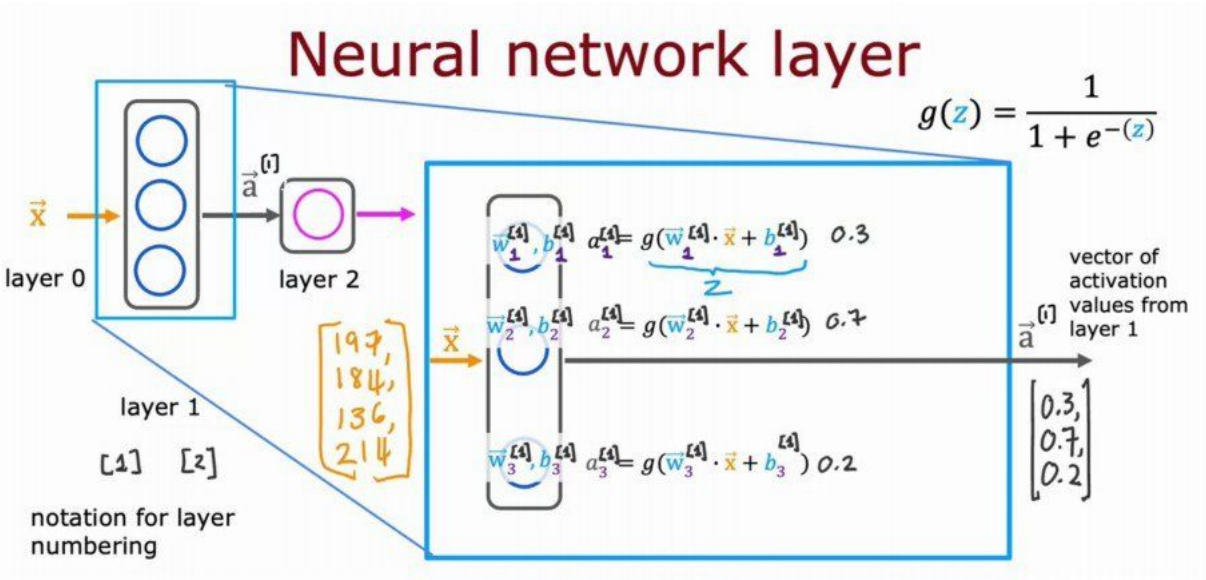
- For a 1000 * 1000 pixel image we have a vector of 1 million elements.
- We build a neural network by providing that as an input.
- In each hidden layer they are improving the recognition all by itself.

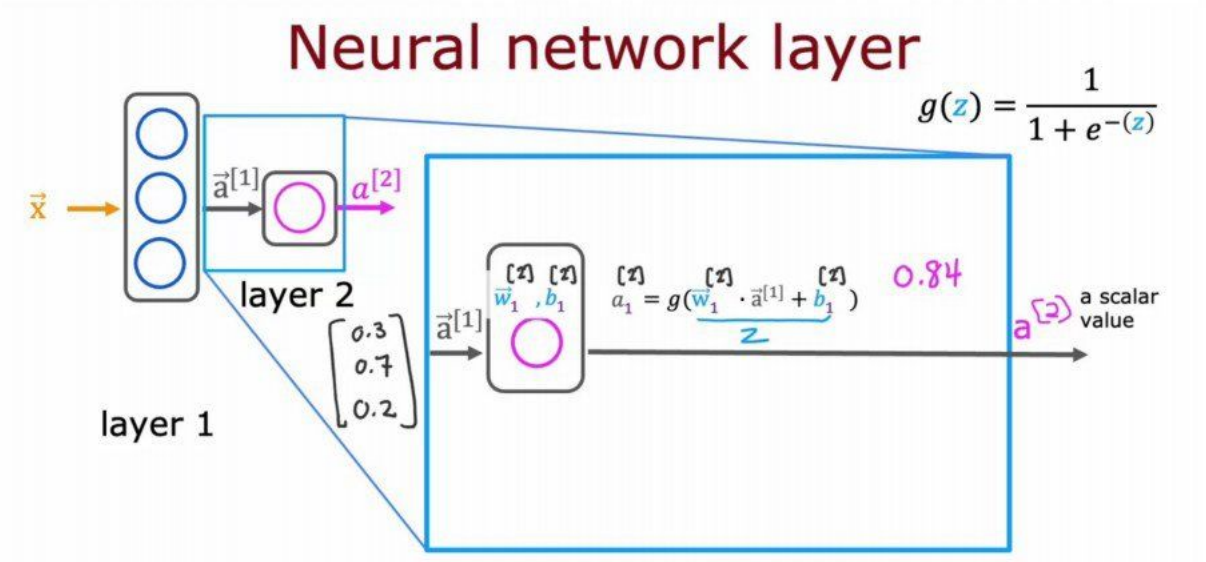


~~Neural Network Layer~~

- Each neuron is implementing the activation of Logistic Regression.

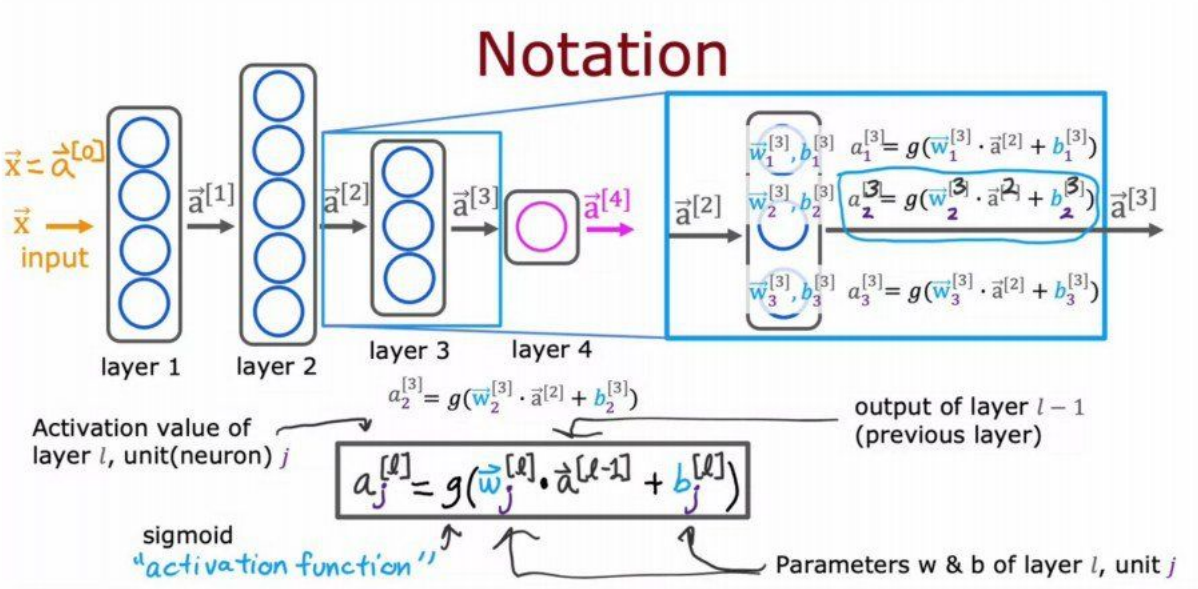
- Super script square bracket 1 means quantity that is associated with Neural Network 1 and so on.
- Input of layer 2 is the output of layer 1. It will be a set of vectors.
- Every layer input a vector of numbers and output a vector of numbers





Complex Neural Network

- aj[l] where j is the neuron/unit and l is the layer
- g(w.a+b) is the activation function(sigmoid function etc can be used here)
- input layer is a[0]

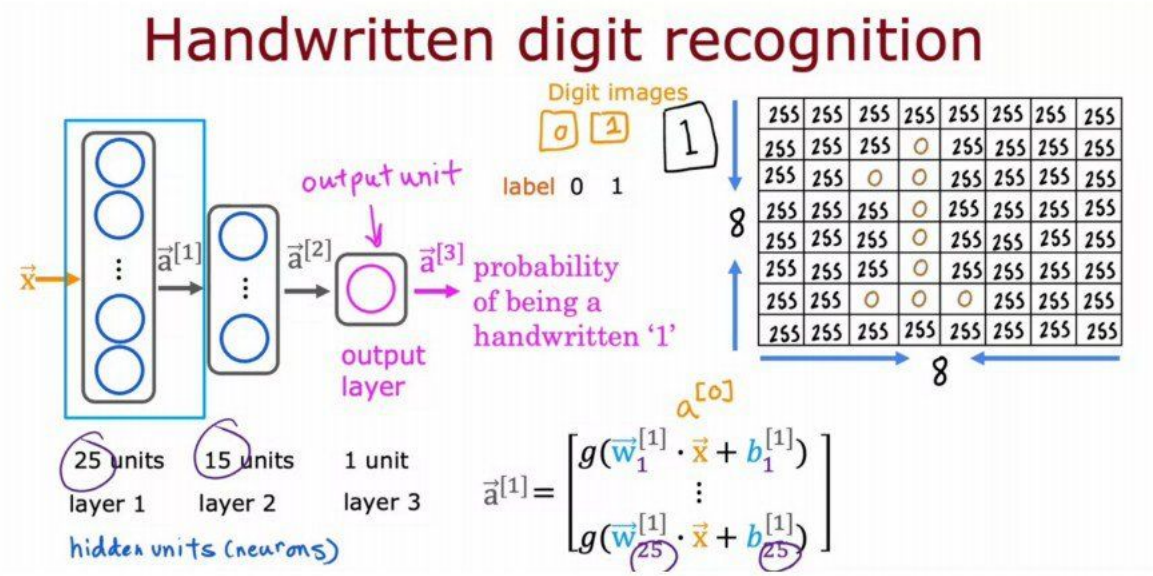


Inference/Make Prediction (Handwritten Digit Recognition)

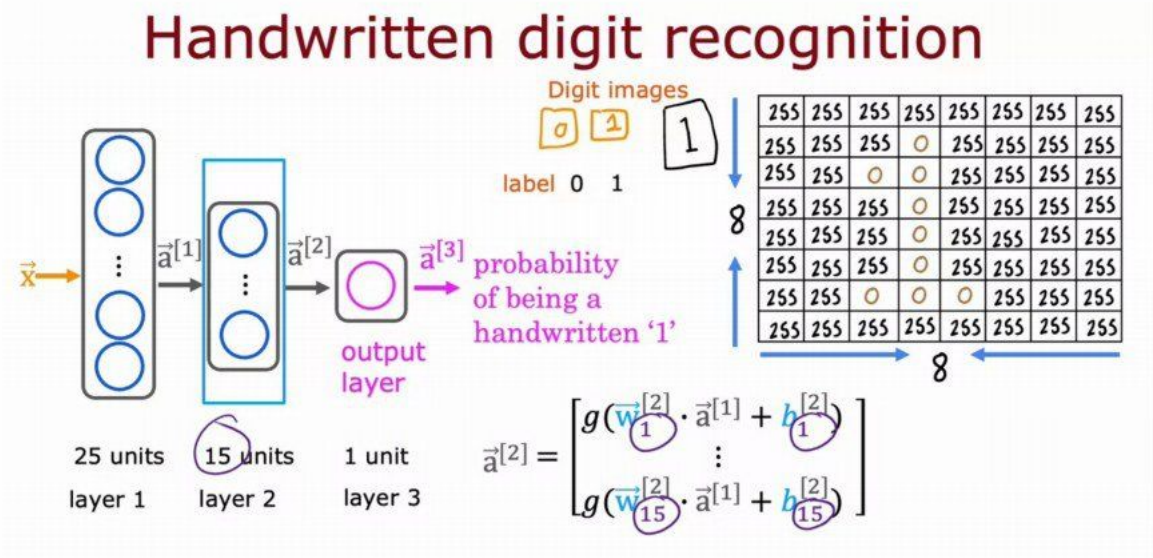
Forward Propagation

- If activation computation goes from left to right it is called, Forward Propagation.

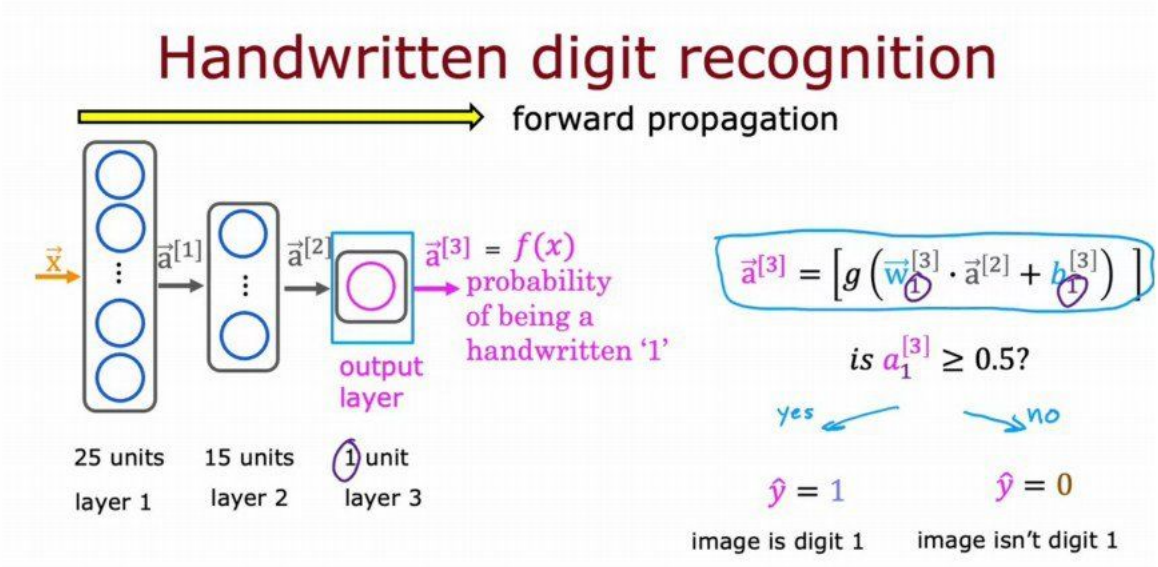
Calculation of first layer vector



Calculation of second layer vector

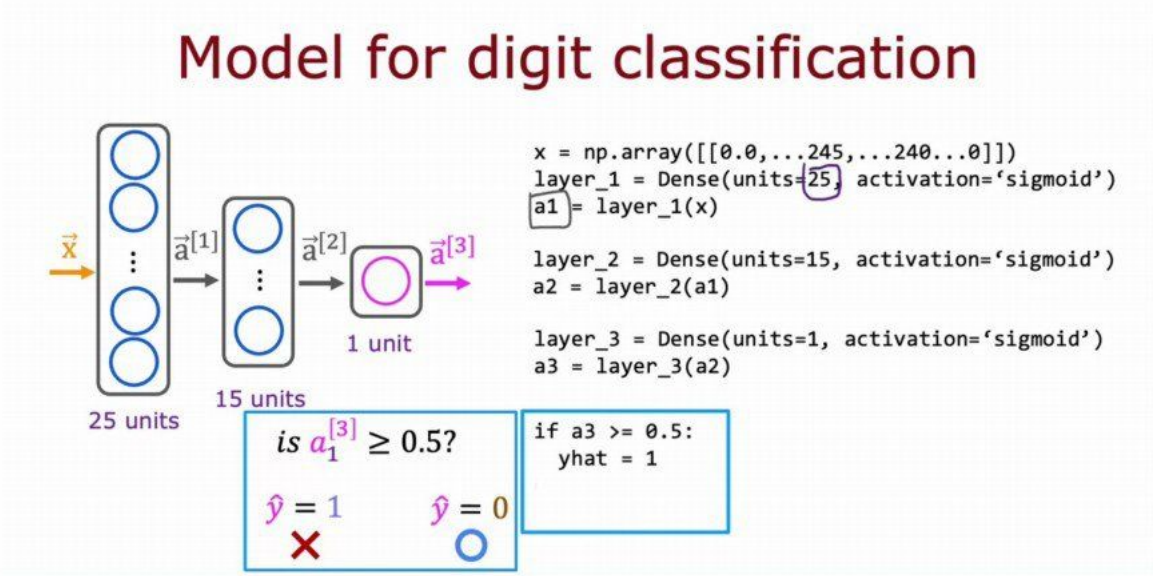


Calculation of last layer vector



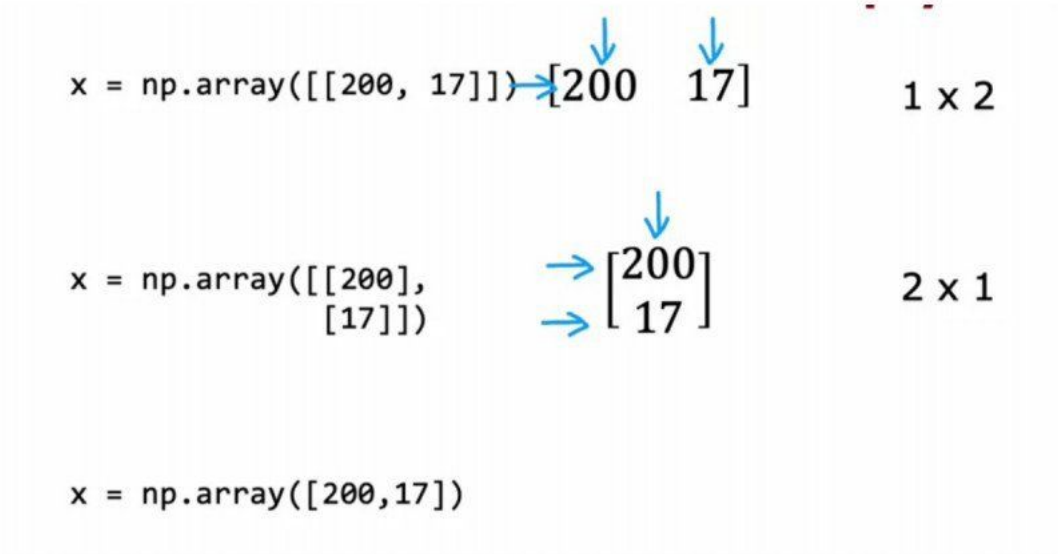
Tensorflow Implementation

- building a layer of NN in tensorflow below for handwritten digit 0/1 classification

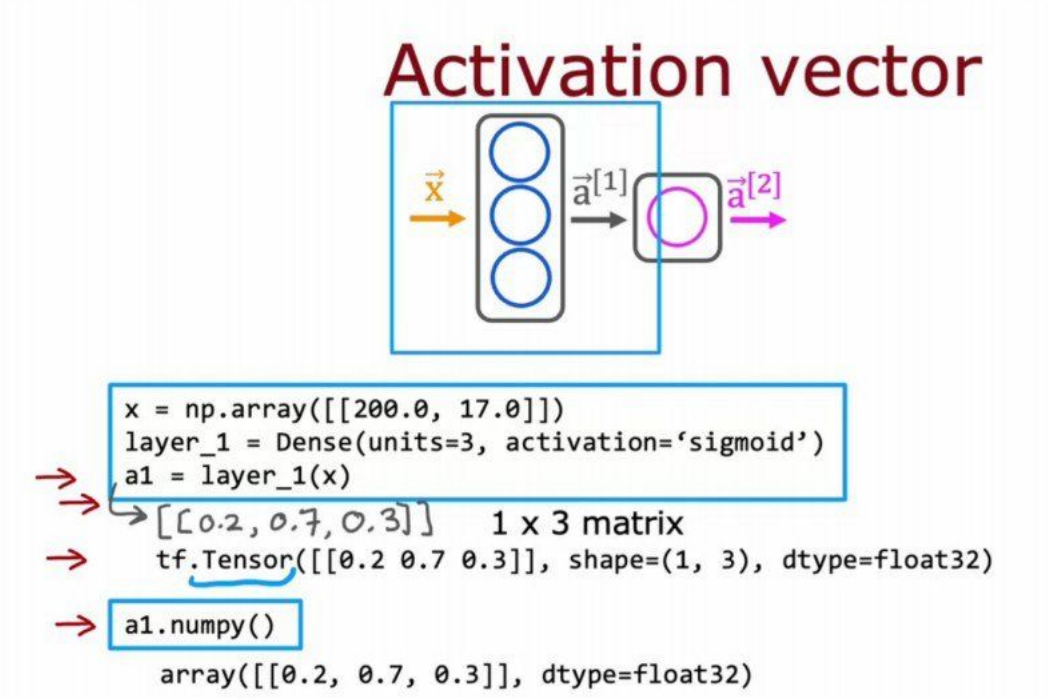


Data in Tensorflow

- Inconsistency between data in Numpy and Tensorflow
- Numpy we used 1D vector, np.array([200, 18])
- Tensorflow uses matrices, np.array([[200, 18]])
- Representing in Matrix instead of 1D array make tensorflow run faster.
- Tensor is something like matrix



Activation of Vector



Building a Neural Network in Tensorflow

Digit Classification Model

```
import tensorflow as tf
layer_1 = Dense( units=25, activation="sigmoid" )
layer_2 = Dense( units=15, activation="sigmoid" )
layer_3 = Dense( units=1, activation="sigmoid" )

model = Sequential ( [ layer_1, layer_2, layer_3 ] )

x = np.array( [ [ 0…..., 245, ….., 17 ],
                [ 0…..., 200, ….., 184 ]  ] )
y = np.array( [ 1, 0 ] )

model.compile(……)
model.fit( x, y )
model.predict( new_x )
```

Forward Prop in Single Layer (Major Parts)

```
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential

model = Sequential(
```

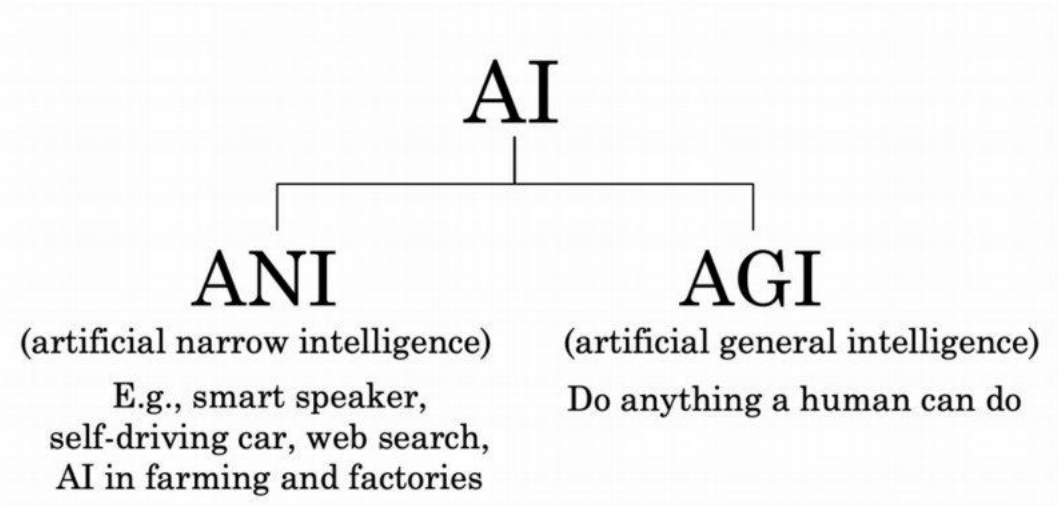
```
    [
        tf.keras.Input(shape=(2,)),
        Dense(3, activation='sigmoid', name = 'layer1'),
        Dense(1, activation='sigmoid', name = 'layer2')
     ]
)

model.compile(
    loss = tf.keras.losses.BinaryCrossentropy(),
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.01),
)

model.fit(
    Xt,Yt,
    epochs=10,
)
```

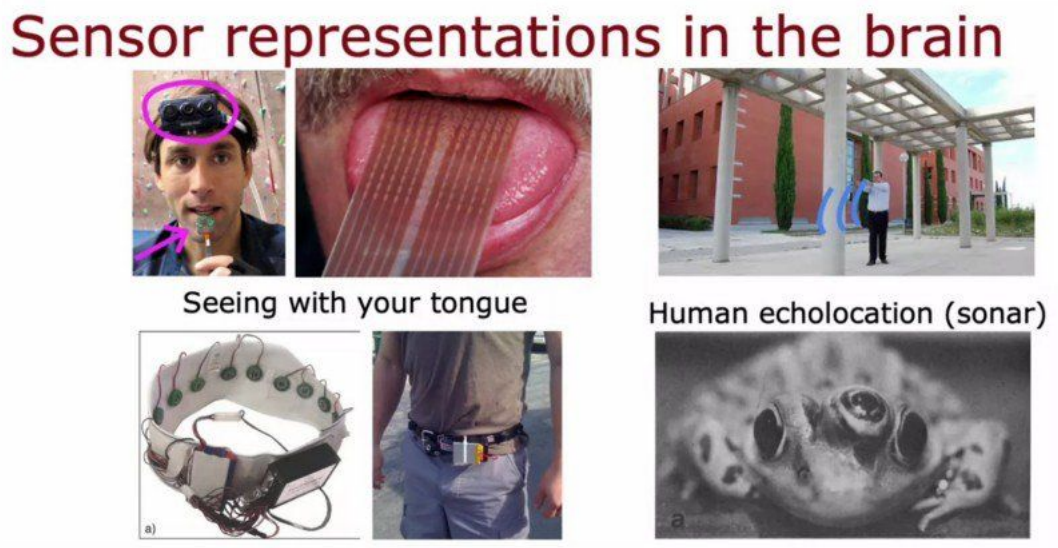
General Implementation of Forward Prop in Single Layer

- Uppercase for Matrix
- Lowercase for Vectors and Scalers

```
def my_dense(a_in, W, b, g):
    """
    Computes dense layer
    Args:
      a_in (ndarray (n, )) : Data, 1 example
      W    (ndarray (n,j)) : Weight matrix, n features per unit, j units
      b    (ndarray (j, )) : bias vector, j units
      g    activation function (e.g. sigmoid, relu..)
    Returns
      a_out (ndarray (j,))  : j units|
    """
    units = W.shape[1]
    a_out = np.zeros(units)
    for j in range(units):
        w = W[:,j]
        z = np.dot(w, a_in) + b[j]
        a_out[j] = g(z)
    return(a_out)

def my_sequential(x, W1, b1, W2, b2):
    a1 = my_dense(x,  W1, b1, sigmoid)
    a2 = my_dense(a1, W2, b2, sigmoid)
    return(a2)
```

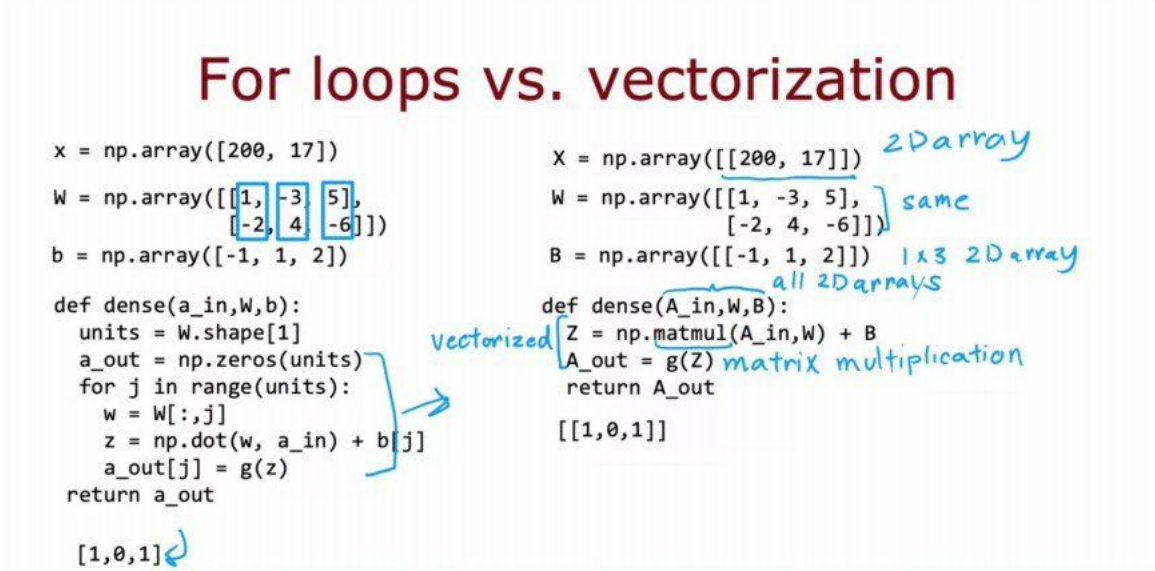
Artificial General Intelligence - AGI





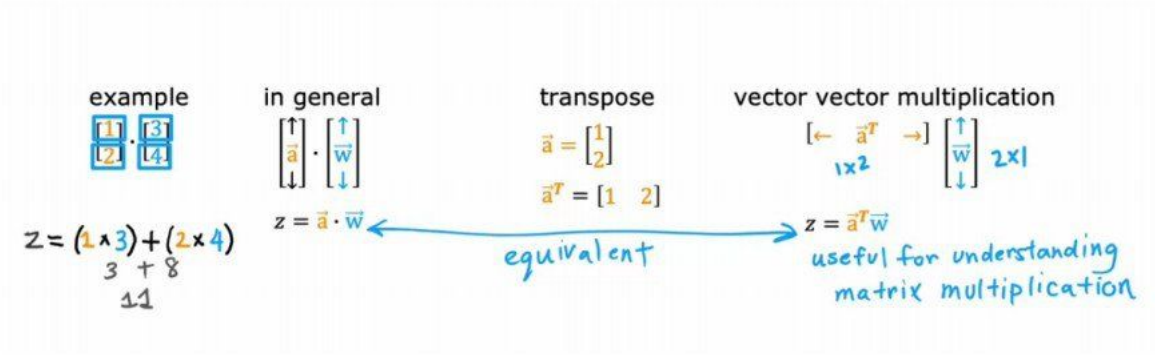
Vectorization

- Matrix Multiplication in Neural Networks using Parallel Computer Hardware
- Matrix Multiplication Replaces For Loops in speed comparison



Dot Product to Matrix Multiplication using Transpose

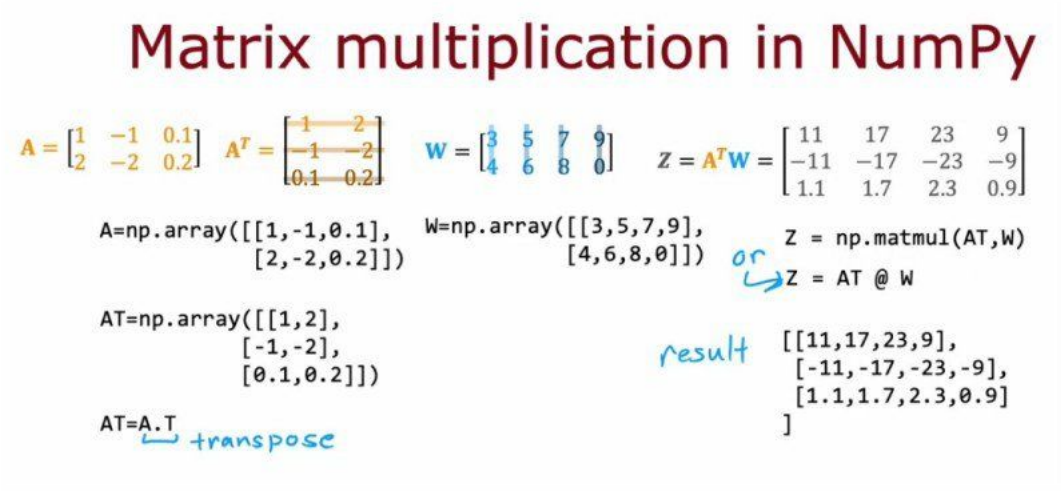
- a . w = a1*w1 + a2+w2 + ……. + an*wn
- a . w = a^T * w



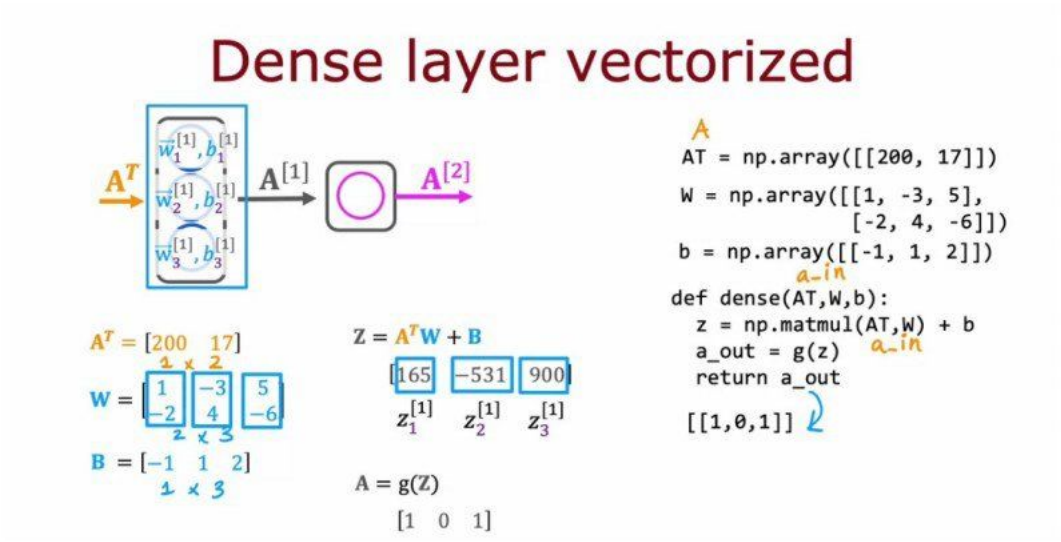
Matrix Multiplication in Neural Networks

- Dot Product of vectors that have same length
- Matrix Multiplication is valid if col of matrix 1 = row of matrix 2
- Output will have row of matrix 1 and col of matrix 2

In code for numpy array and vectors



Dense Layer Vectorized

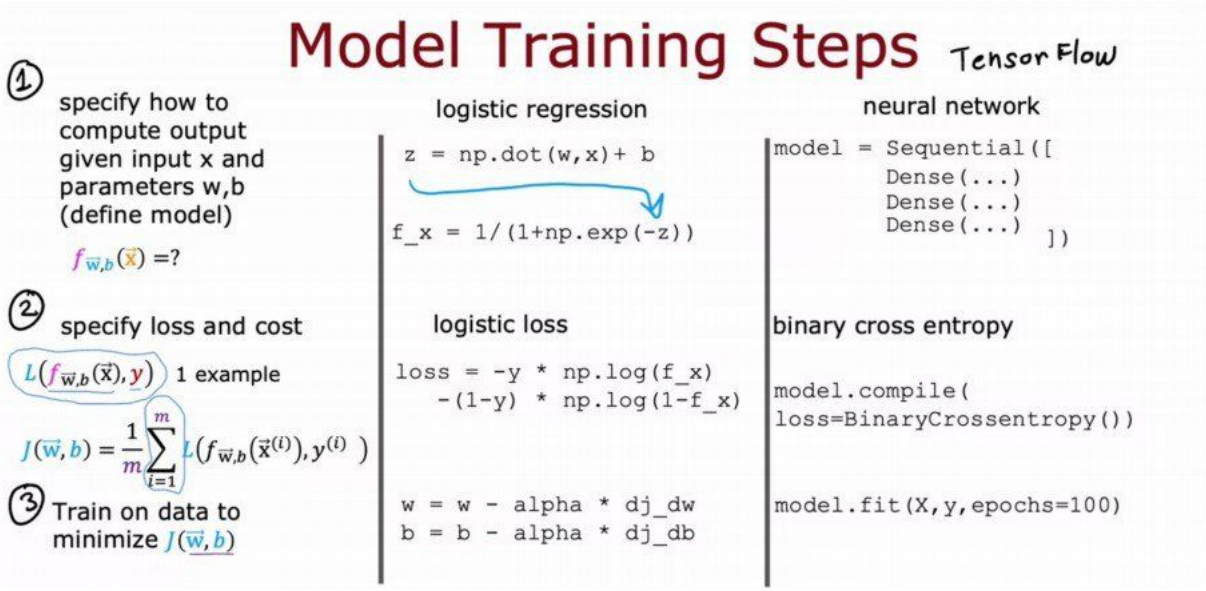


Tensorflow Training

- epoch is how many steps, the learning algorithm like gradient descent should run.
- BinaryCrossentropy is log loss function for logistic regression in classification problem
- if we have regression problem, use MeanSquaredError()
- loss = tf.keras.losses.MeanSquaredError()
- Derivatives of gradient descent has been calculated using backpropagation handled by model.fit

```
import tensorflow as tf
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import Sequential
```

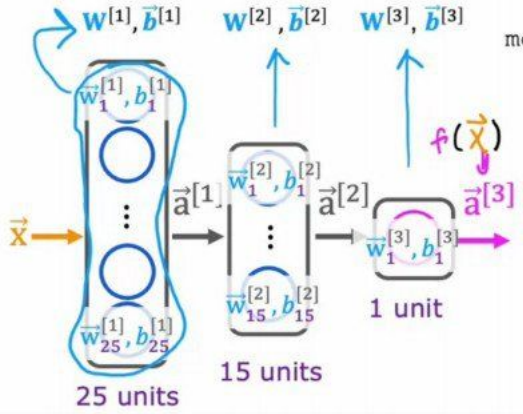
```
model = Sequential(
    [
        tf.keras.Input(shape=(2,)),
        Dense(3, activation='sigmoid', name = 'layer1'),
        Dense(1, activation='sigmoid', name = 'layer2')
     ]
)

model.compile(
    loss = tf.keras.losses.BinaryCrossentropy(),
    optimizer = tf.keras.optimizers.Adam(learning_rate=0.01),
)

model.fit(
    Xt,Yt,
    epochs=10,
)
```



1. Create the model

define the model

$f(\vec{x})$ =?




```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense

model = Sequential([
  Dense(units=25, activation='sigmoid'),
  Dense(units=15, activation='sigmoid'),
  Dense(units=1, activation='sigmoid'),
                ])
```

2. Loss and cost functions

handwritten digit classification problem — binary classification

$$L(f(\vec{x}), y) = -y\log(f(\vec{x})) - (1-y)\log(1-f(\vec{x}))$$

Compare prediction vs. target

logistic loss
also known as binary cross entropy

$$J(\mathbf{W}, \mathbf{B}) = \frac{1}{m}\sum_{i=1}^{m} L(f(\vec{x}^{(i)}), y^{(i)})$$

$\mathbf{W}^{[1]}, \mathbf{W}^{[2]}, \mathbf{W}^{[3]}$ $\vec{b}^{[1]}, \vec{b}^{[2]}, \vec{b}^{[3]}$

$f_{W,B}(\vec{x})$

```
model.compile(loss= BinaryCrossentropy())
```

```
from tensorflow.keras.losses import
  BinaryCrossentropy
```

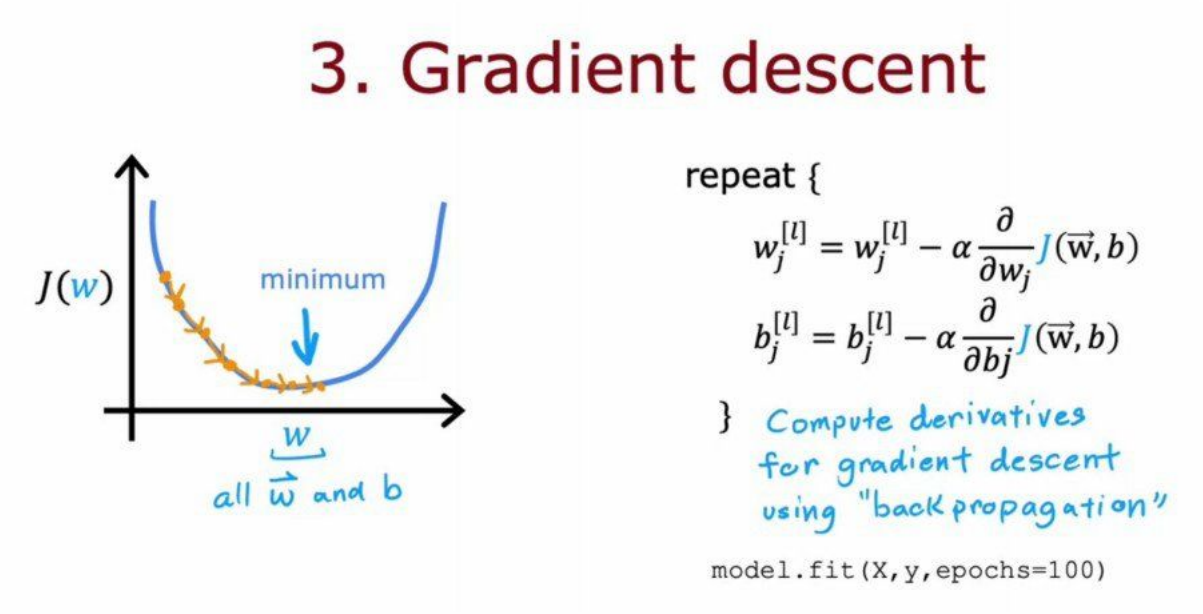
K Keras

regression
(predicting numbers and not categories)

mean squared error

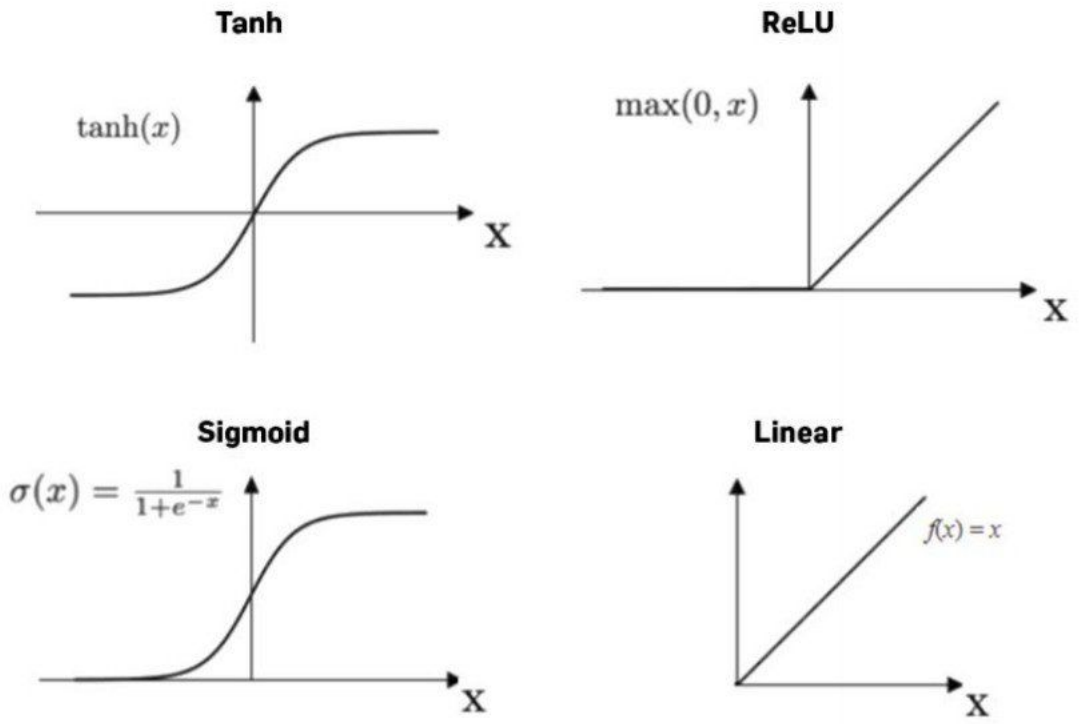
```
model.compile(loss= MeanSquaredError())
```

```
from tensorflow.keras.losses import
  MeanSquaredError
```



Activation Function Alternatives

- awareness can not be binary, It better to use like like non negative number from zero to large number
- So best activation here is ReLU, Rectifier Linear Unit
- g(z) = max(0, z)
- ReLU is common choice of training neural networks and now used more than Sigmoid
- ReLU is faster
- ReLU is flat in one side, and Sigmoid flat on both sides. So gradient descent perform faster in ReLU



Choose Activation Function

For Output Layer

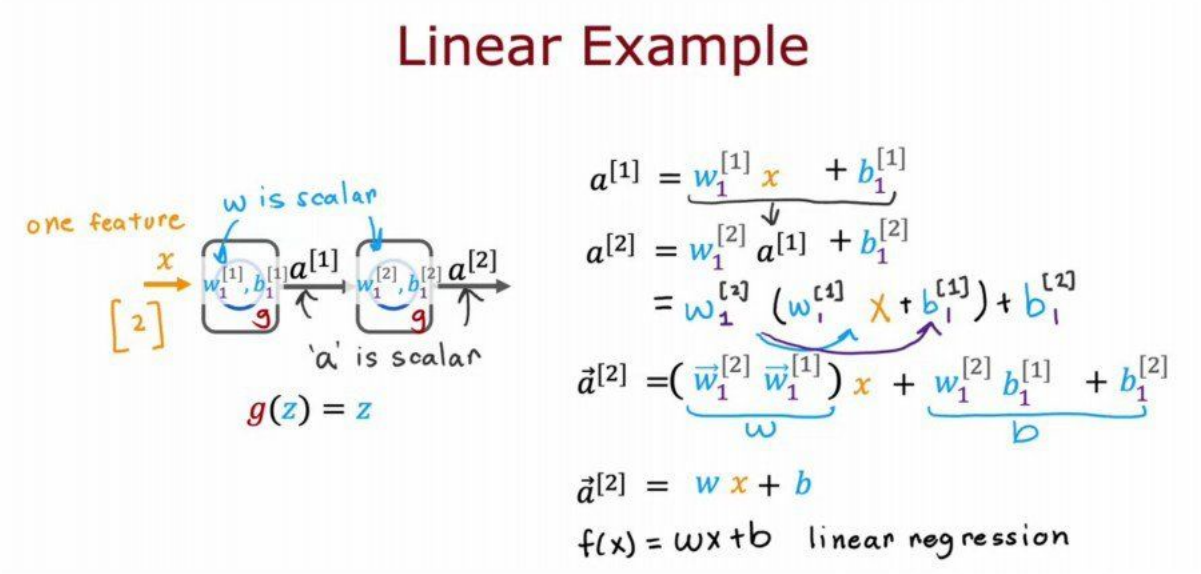
- Binary Classification - Sigmoid
- Regression - Linear
- Regression Positive Only - ReLU

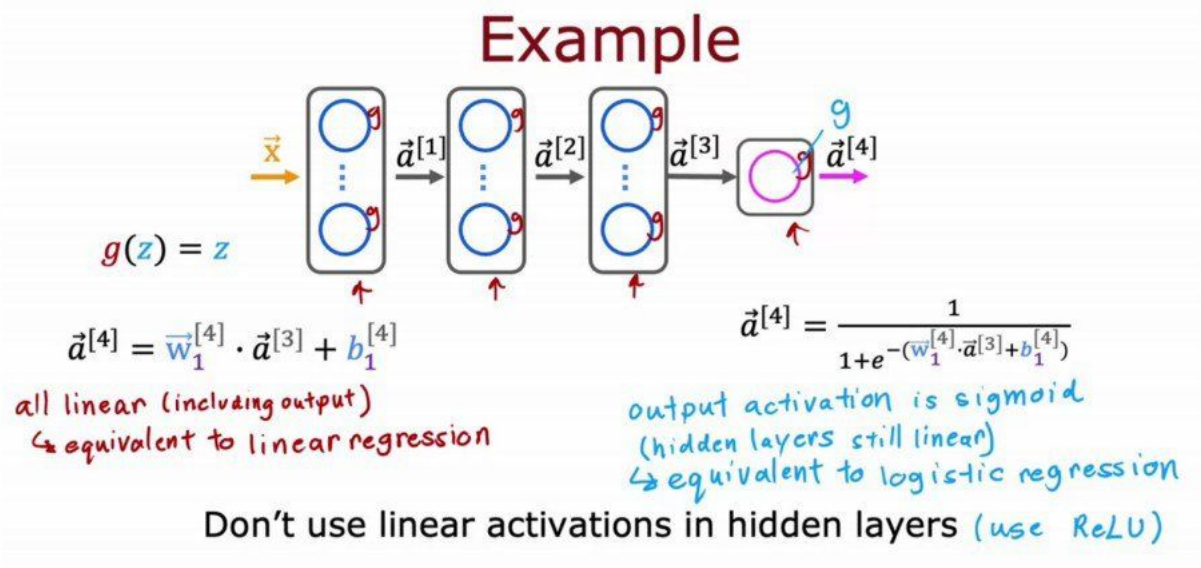
For Hidden Layer

- ReLU as standard activation

Why do we need activation functions?

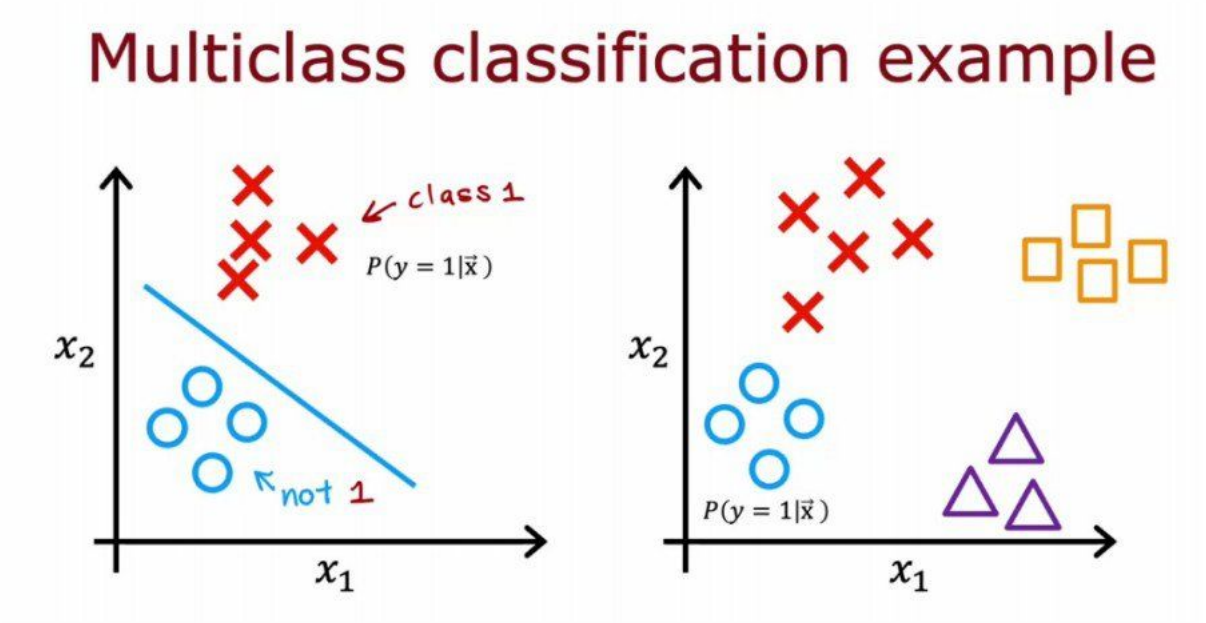
- If we use linear activation function across all the neurons the neural network is no different from Linear Regression.
- If we use linear in hidden and sigmoid in output it is similar to logistic regression.
- **Don’t use linear in hidden layers. Use ReLU**
- We wont be able to fit anything complex





Multiclass Classification

- classification with more than 2 class is called multiclass classification
- Identifying a number is multiclass since we want to classify 10 number classes (MNIST)
- for n = 2 softmax is basically logistic regression



Logistic regression
(2 possible output values)

$z = \vec{w} \cdot \vec{x} + b$

$a_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$ 0.71

$a_2 = 1 - a_1 = P(y=0|\vec{x})$ 0.29

Softmax regression
(N possible outputs) $y = 1, 2, 3, \ldots, N$

$z_j = \vec{w}_j \cdot \vec{x} + b_j \quad j = 1, \ldots, N$

parameters $w_1, w_2, \ldots, w_N$ $b_1, b_2, \ldots, b_N$

$a_j = \frac{e^{z_j}}{\sum_{k=1}^{N} e^{z_k}} = P(y=j|\vec{x})$

note: $a_1 + a_2 + \ldots + a_N = 1$

Softmax regression (4 possible outputs) $y = 1, 2, 3, 4$

$z_1 = \vec{w}_1 \cdot \vec{x} + b_1$

$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=1|\vec{x})$ 0.30

$z_2 = \vec{w}_2 \cdot \vec{x} + b_2$

$a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=2|\vec{x})$ 0.20

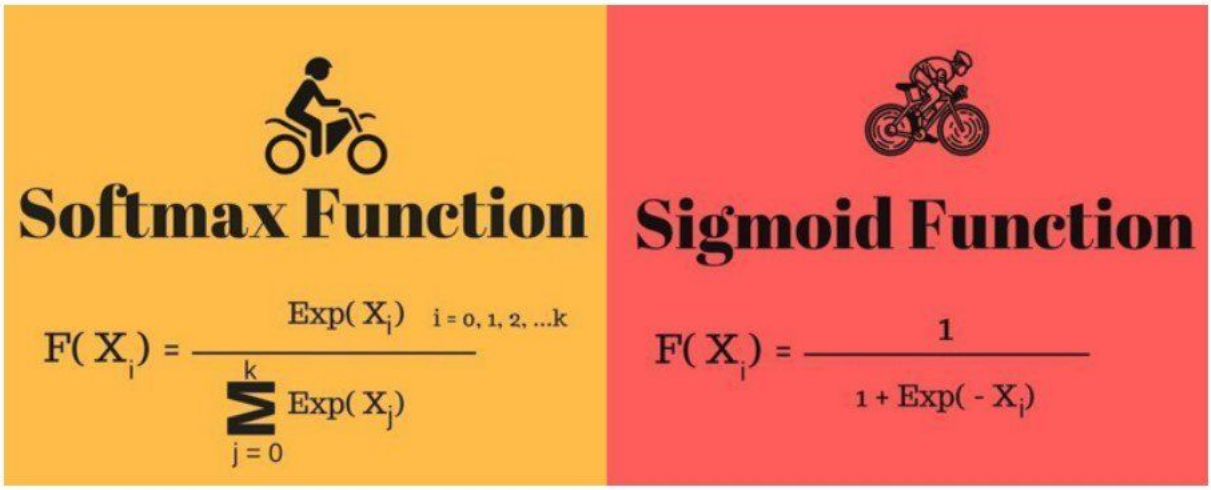
$z_3 = \vec{w}_3 \cdot \vec{x} + b_3$

$a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=3|\vec{x})$ 0.15

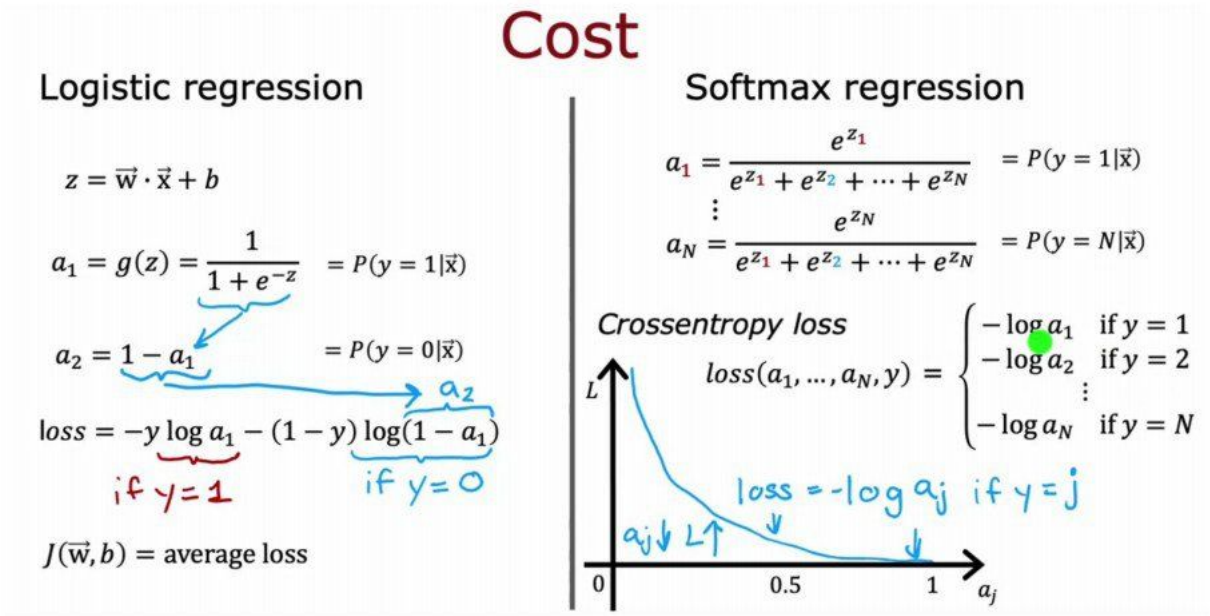
$z_4 = \vec{w}_4 \cdot \vec{x} + b_4$

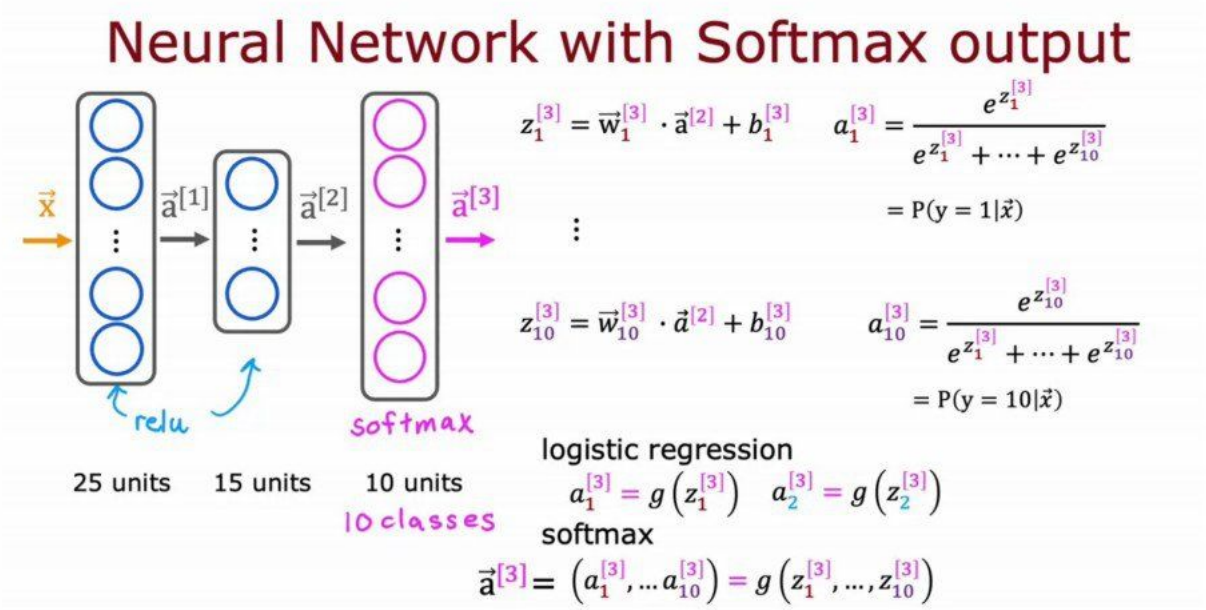
$a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=4|\vec{x})$ 0.35

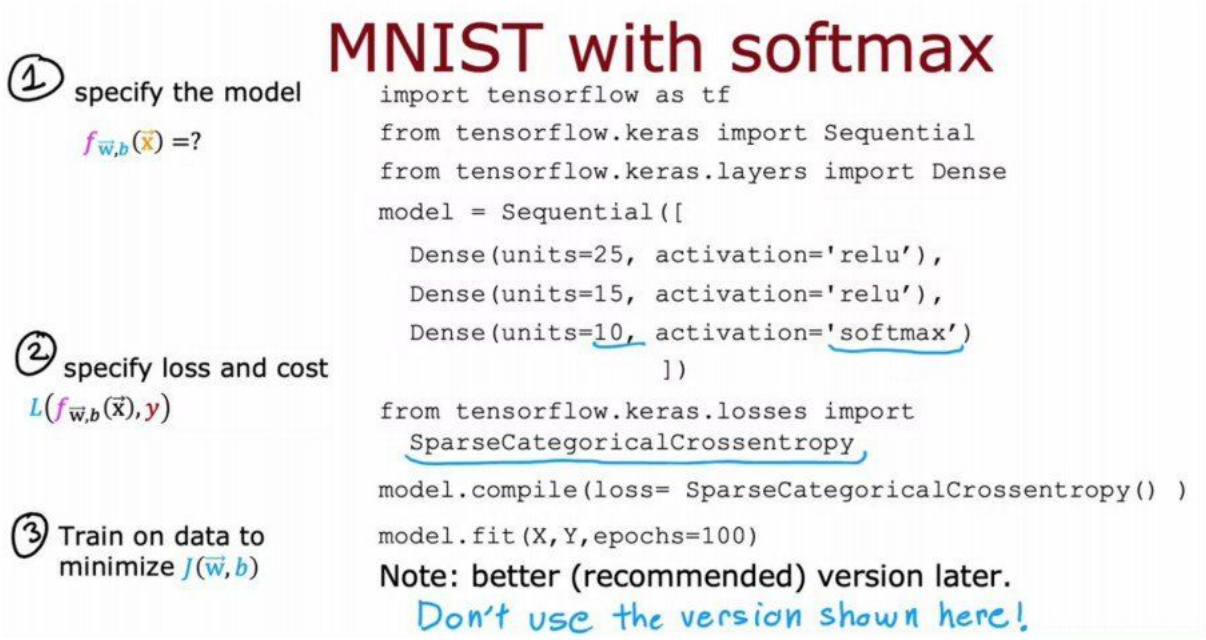
Cost Function of Softmax



- Cross entropy loss - if loss value is 1 smaller the loss.
- Cross entropy loss - if loss value is 0 larger the loss.
- **Sigmoid - BinaryCrossEntropy**
- **Softmax - SparseCategoricalCrossEntropy**







Don't use the above way of code for implementation. We can use alternative efficient method.

Improved Implementation of Softmax

```
x1 = 2.0 / 10000
0.00020000000000

x2 = (1 + 1/10000) - (1 - 1/10000)
0.000199999999978
```

```
# due to memory constraints in computer rounding happens.
# inorder to avoid/reduce rounding up in softmax we can use other implementation
```

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

Logistic regression:

$$a = g(z) = \frac{1}{1+e^{-z}}$$

Original loss

$$loss = -y\log(a) - (1-y)\log(1-a)$$

More accurate loss (in code)

$$loss = -y\log\left(\frac{1}{1+e^{-z}}\right) - (1-y)\log\left(1-\frac{1}{1+e^{-z}}\right)$$

logit: z

```
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=10, activation='sigmoid')
                  ])
model.compile(loss=BinaryCrossEntropy() )
```

'linear'

```
model.compile(loss=BinaryCrossEntropy(from_logits=True) )
```

More numerically accurate implementation of softmax

Softmax regression

$$(a_1, \ldots, a_{10}) = g(z_1, \ldots, z_{10})$$

$$\text{Loss} = L(\vec{a}, y) = \begin{cases} -\log a_1 & \text{if } y = 1 \\ \vdots & \\ -\log a_{10} & \text{if } y = 10 \end{cases}$$

```
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=10, activation='softmax')
                  ])
```

'linear'

```
model.compile(loss=SparseCategoricalCrossEntropy() )
```

More Accurate

$$L(\vec{a}, y) = \begin{cases} -\log \frac{e^{z_1}}{e^{z_1}+\cdots+e^{z_{10}}} & \text{if } y = 1 \\ \vdots & \\ -\log \frac{e^{z_{10}}}{e^{z_1}+\cdots+e^{z_{10}}} & \text{if } y = 10 \end{cases}$$

```
model.compile(loss=SparseCategoricalCrossEntropy(from_logits=True))
```

MNIST (more numerically accurate)

model

```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
  Dense(units=25, activation='relu'),
  Dense(units=15, activation='relu'),
  Dense(units=10, activation='linear') ])
```

loss

```
from tensorflow.keras.losses import
  SparseCategoricalCrossentropy
model.compile(...,loss=SparseCategoricalCrossentropy(from_logits=True) )
```

fit

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

predict

```
logits = model(X)
f_x = tf.nn.softmax(logits)
```

not is $a_1 \dots a_{10}$ $z_1 \dots z$

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)) )
```

fit

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

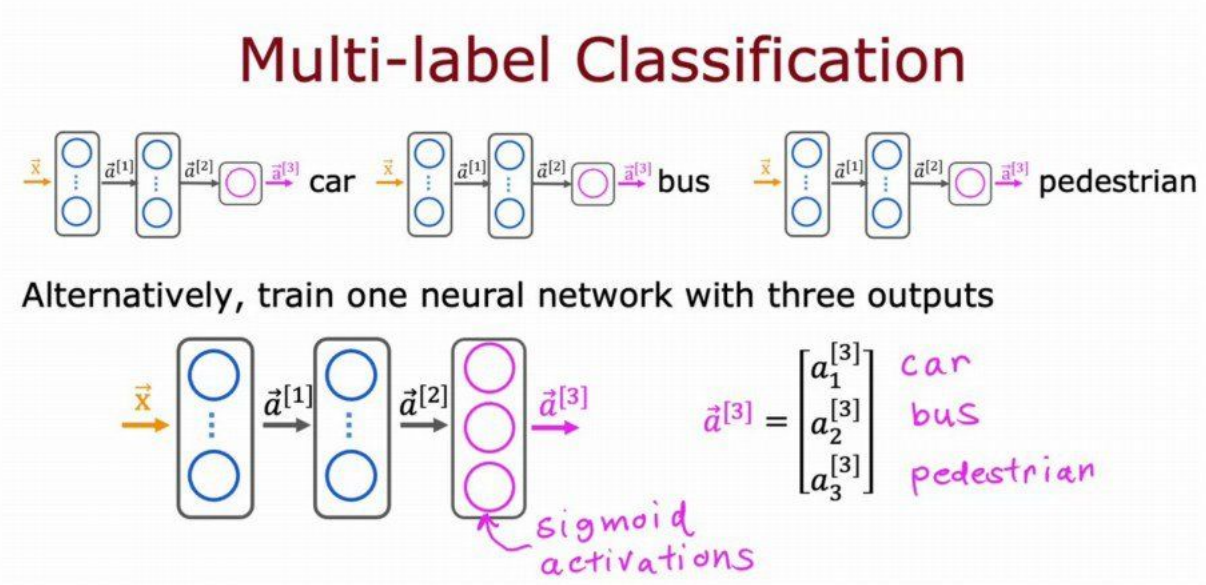
z

predict

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

Multi Label Classification

- In single output there are multiple class, like returning a vector of possibility
- For self driving car it have multiple labels like is there a car? is there a bus? is there a man?
- It output result as [1, 0, 1]
- We cannot make different NN for each we want it in single NN



Adam - Adaptive Moment Estimation

- Adam Optimizer is a alternative for Gradient Descent
- It is faster than GD
- learning rate is not a constant one but multiple learning rate is used
- different Alpha should be tried like so small and large even after using 0.001

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

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

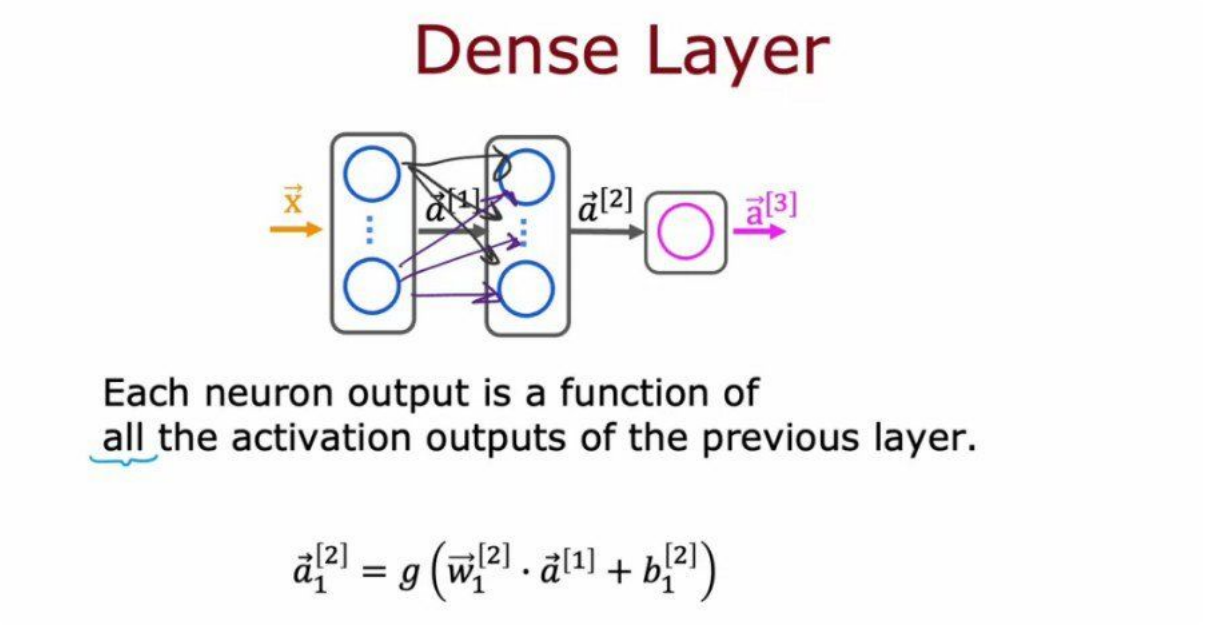
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

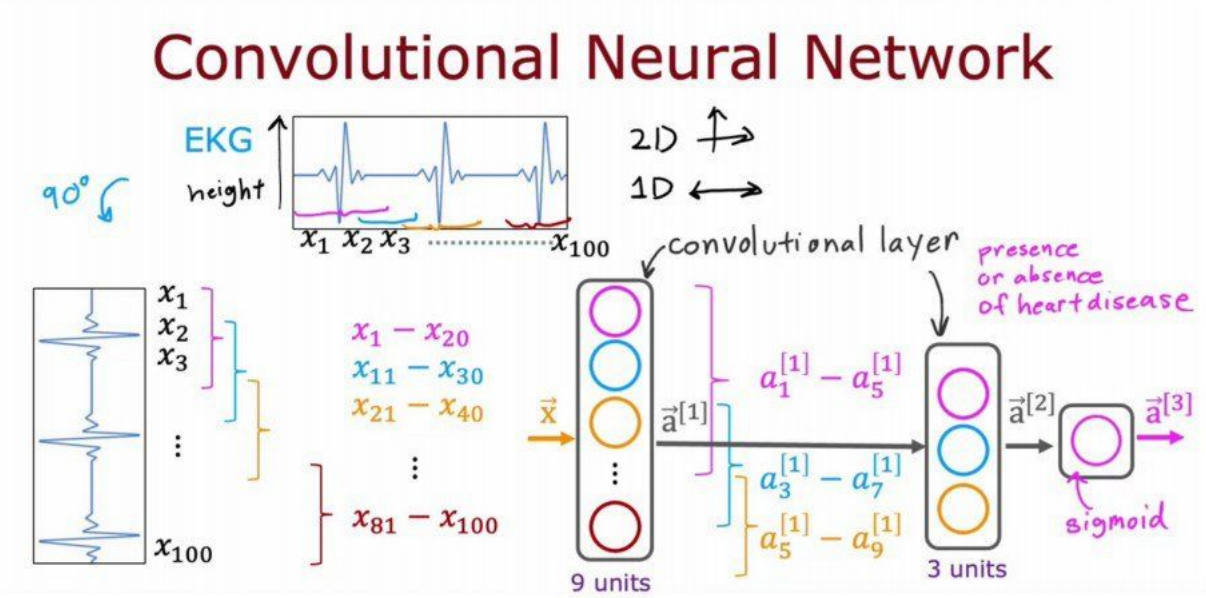
```
model.fit(X,Y,epochs=100)
```

Convolutional Neural Network - CNN

- It is a layer alternative for **Dense Layer**



- Hidden layer in CNN can use a set or part for creating a NN (MNIST)
- each neuron only look at part of the previous layer's input
- need less training data
- So due to this it can be fast
- we can avoid overfitting



Debugging a learning algorithm

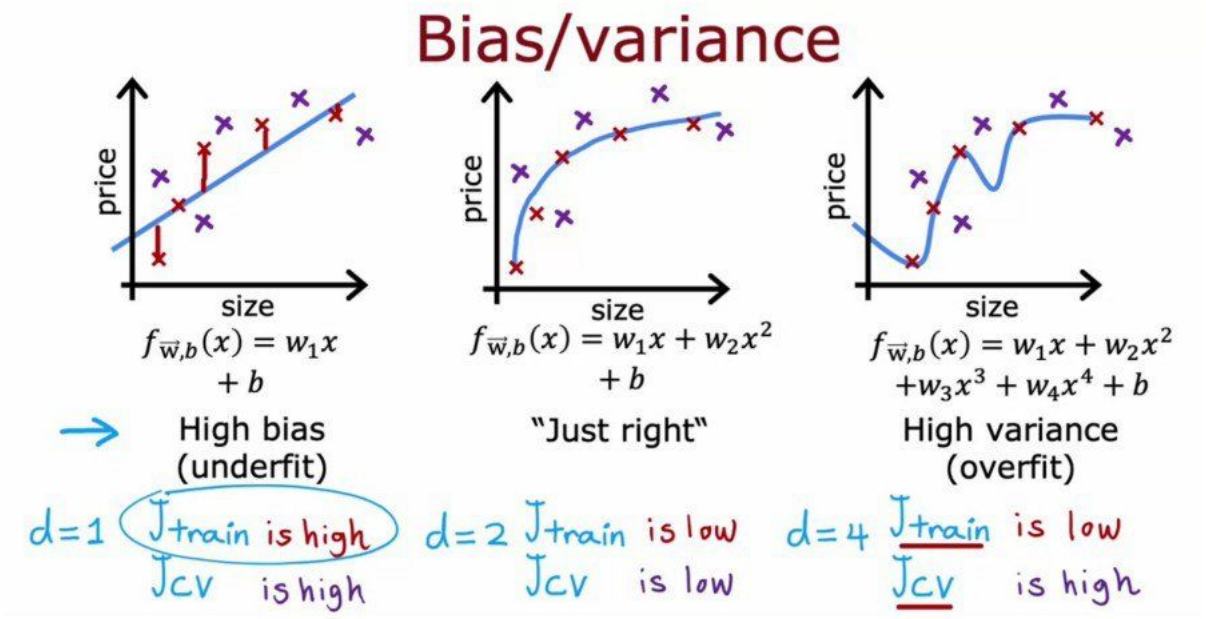
- Get more training data
- try small set of features
- try large set of features
- try adding polynomial features
- try decreasing and increasing learning rate and lambda (regularization parameter)
- we can use different diagnostic systems

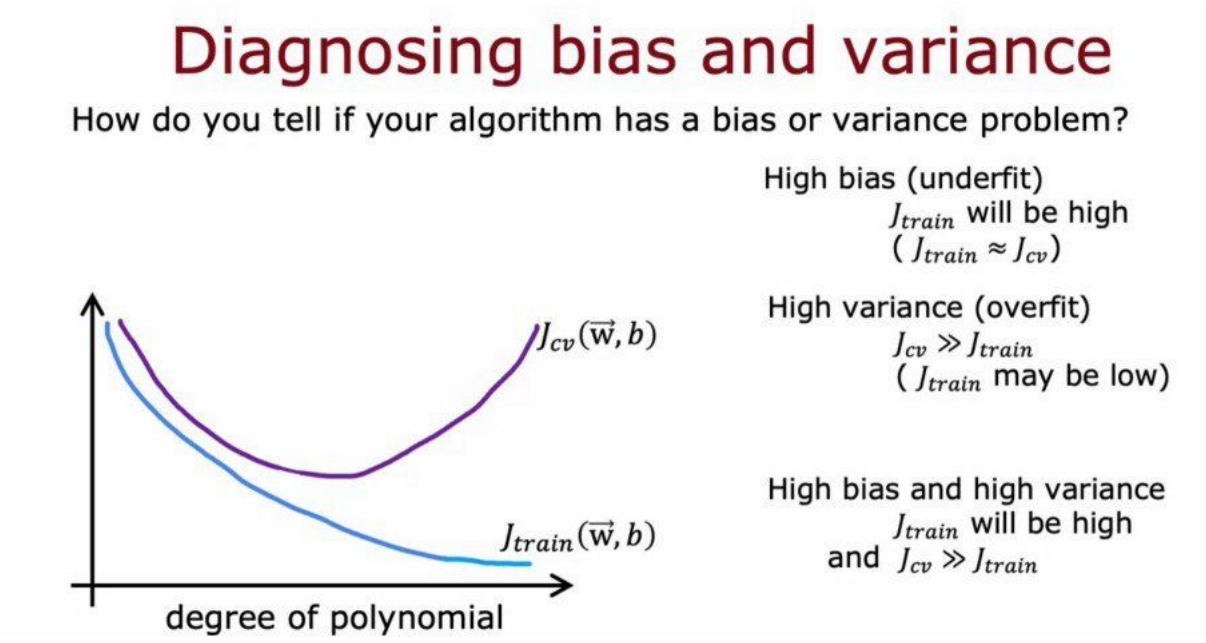
Evaluating your Model

- Split the data training and testing.
- we can find training error and testing error
- cost function should be minimized
- MSE for Linear Regression
- Log Loss for Logistic Regression - Classification
- **For classification problem we can find what percentage of training and test set model which has predicted false, which is better than log loss**
- we can use polynomial regression and test the data for various degree of polynomial
- But to improve testing we can split data into training, cross validation set and test set
- Even for NN we can use this type of model selection
- Then we can find the cost function for each set

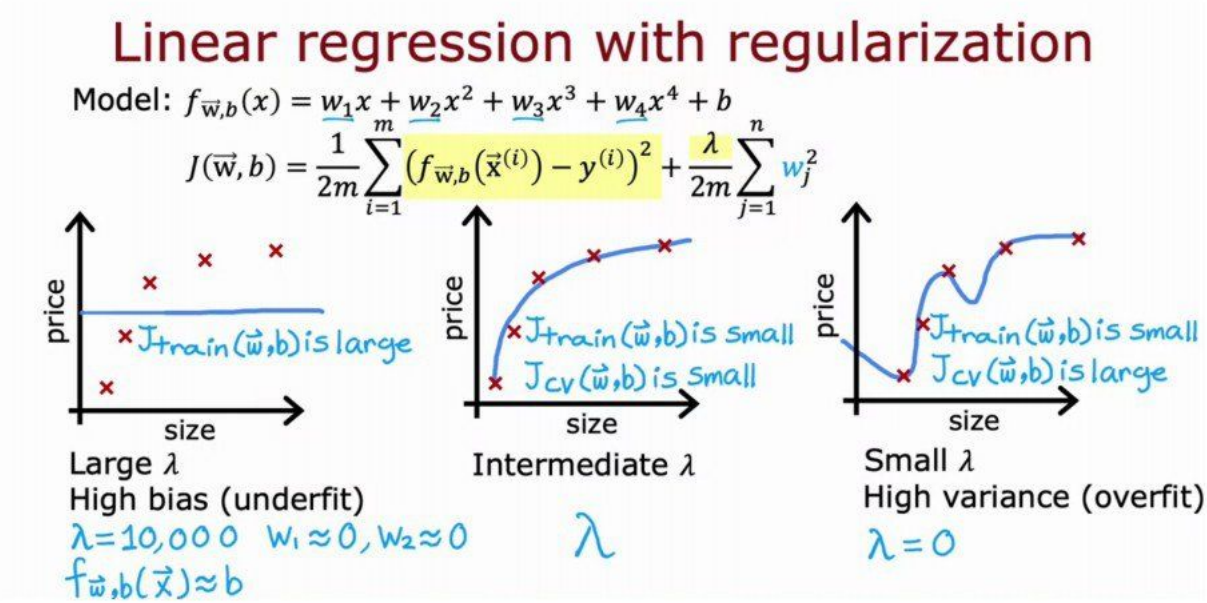
Bias and Variance - Model Complexity or High Degree Polynomial

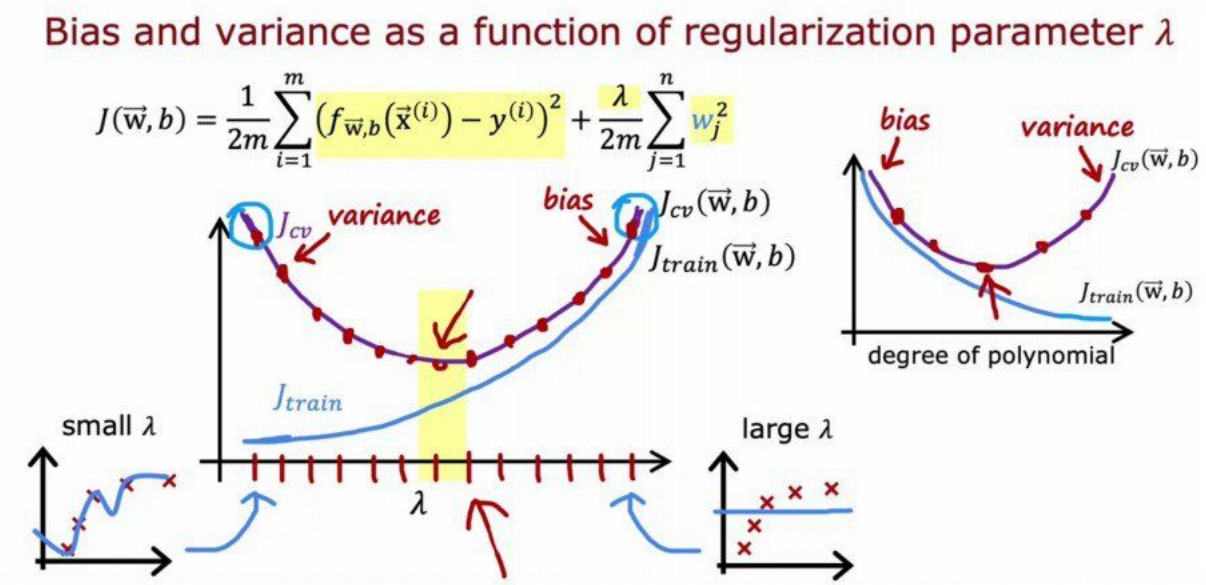
- We need low bias and low variance
- Bias - Difference between actual and predicted value
- Variance - Perform well on training set and worse on testing set
- degree of polynomial should not be small or large. It should be intermediate





Regularization - Bias and Variance





- Underfit - low Poly degree, high lambda (regularization parameter)
- Overfit - High Poly degree, Low lambda (regularization parameter)
- Intermediate lambda and Poly degree is best

Baseline

- First find out the baseline performance human can achieve or default baseline

Bias/variance examples

Baseline performance	: 10.6%	0.2%	10.6% → 4.4%	10.6% → 4.4%
Training error (J_{train})	: 10.8%		15.0% → 0.5%	15.0% → 4.7%
Cross validation error (J_{cv})	: 14.8%	4.0%	15.5%	19.7%
		high variance	high bias	high bias high variance

Learning Curve

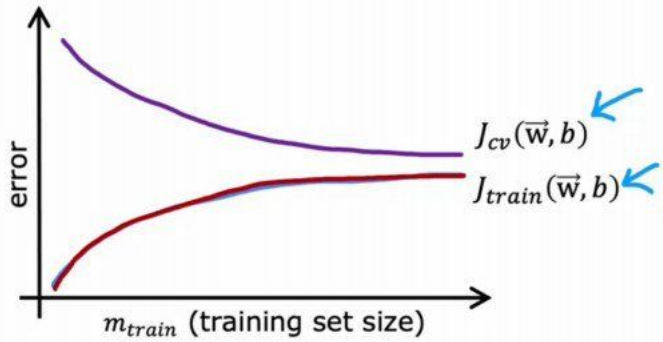
Bias/variance examples

Baseline performance	: 10.6%	0.2%	10.6% → 4.4%	10.6% → 4.4%
Training error (J_{train})	: 10.8%		15.0% → 0.5%	15.0% → 4.7%
Cross validation error (J_{cv})	: 14.8%	4.0%	15.5%	19.7%
		high variance	high bias	high bias high variance

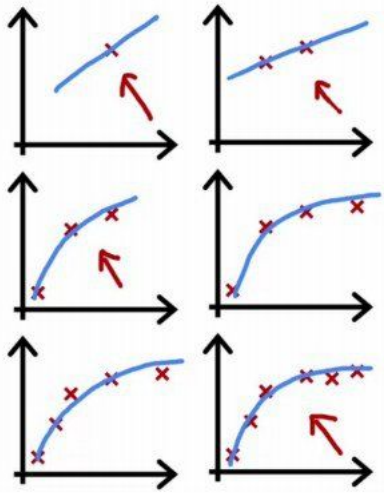
Learning curves

J_{train} = training error

J_{cv} = cross validation error

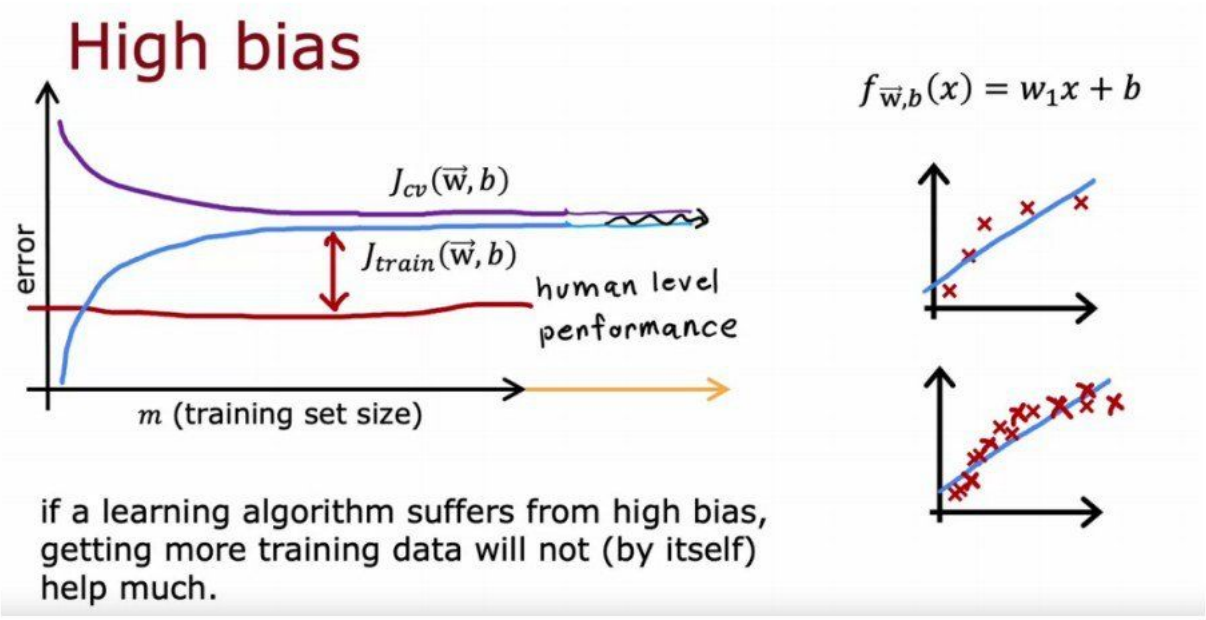


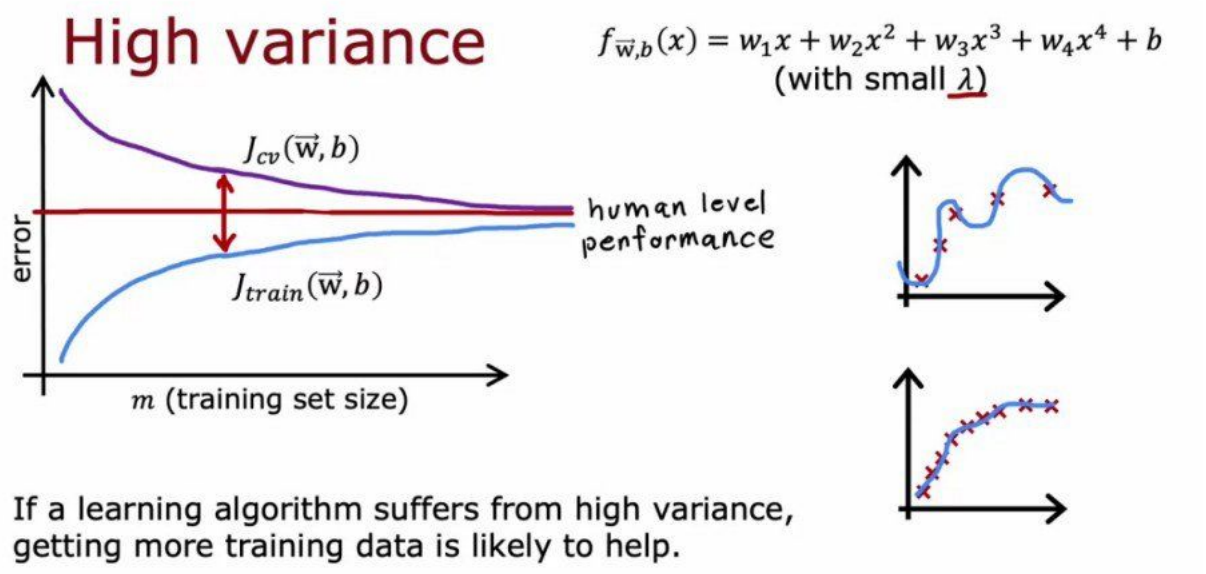
$f_{\vec{w},b}(x) = w_1x + w_2x^2 + b$



If we have high bias increasing training data is not going to help. It wont decrease the error

If we have high variance increasing training data is going to help. It will decrease the error





Deciding what to try next revisited

BIAS

- Bias is the difference between the average prediction of our model and the correct value.

- Model with high bias pays very little attention to the training data and oversimplifies the model.
- It always leads to high error on training and test data.

VARIANCE

- Variance is the variability of model prediction for a given data point or a value which tells us spread of our data.
- Model with high variance pays a lot of attention to training data and does not generalize on test data.
- As a result, such models perform very well on training data but has high error rates on test data.

Overfitting - High Variance and Low Bias

Underfitting - High/Low Variance and High Bias

High Variance

- Get more training data - High Variance
- try increasing lambda - Overfit High Variance
- try small set of features - Overfit High Variance

High Bias

- try large set of features - Underfit High Bias
- try adding polynomial features - Underfit High Bias
- try decreasing lambda - Underfit High Bias

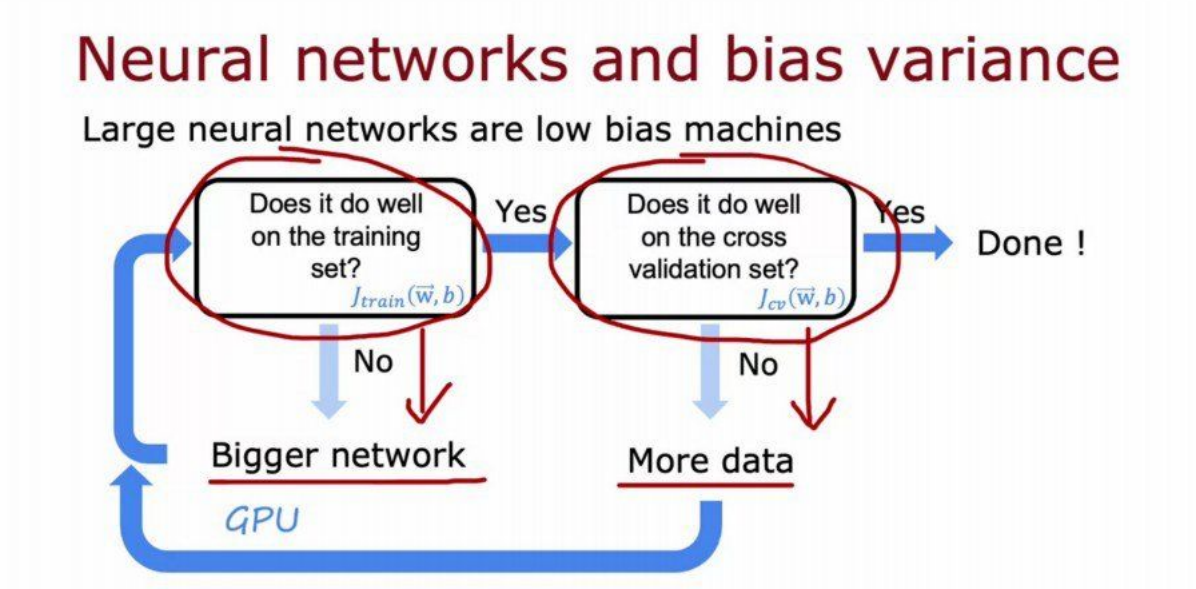
Bias and Variance Neural Networks

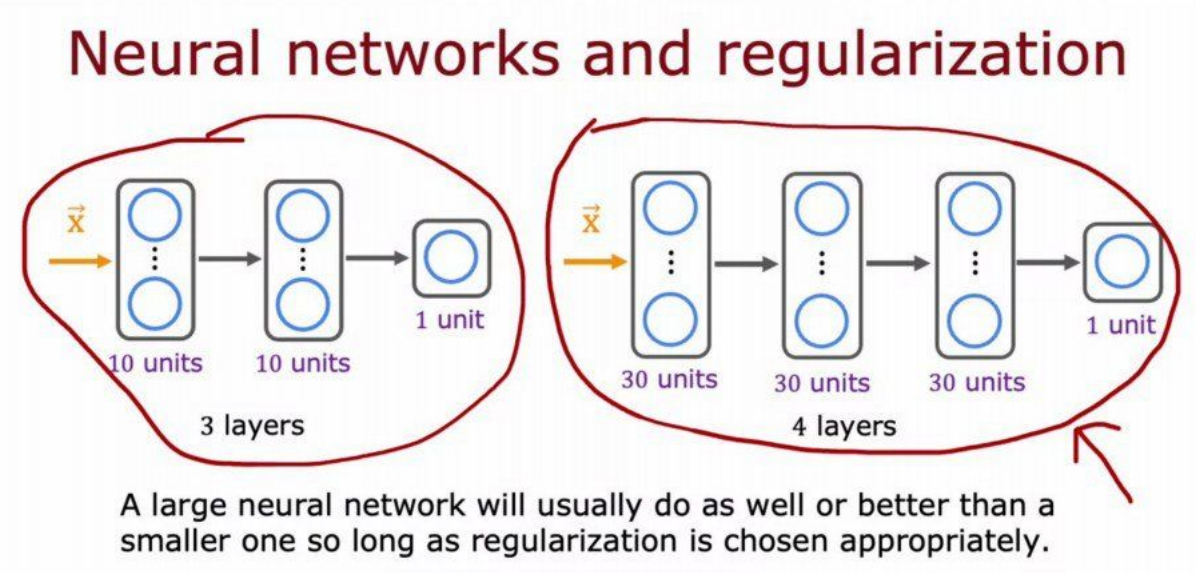
- Simple Model - High Bias
- Complex Model - High Variance
- We need a model between them, that is we need to find a trade off between them.

Neural Networks

- Large NN are having low bias machines

- GPU are there to speed up NN





- With best Regularization large NN will do better than smaller one.
- But large NN take time to run
- Implementing NN with regularization having lambda 0.01 (L2 regularization Ridge)

Neural network regularization

$$J(\mathbf{W},\mathbf{B}) = \frac{1}{m}\sum_{i=1}^{m} L\left(f(\vec{x}^{(i)}), y^{(i)}\right) + \frac{\lambda}{2m}\sum_{\text{all weights } \mathbf{W}} (w^2)$$

b

Unregularized MNIST model

```
layer_1 = Dense(units=25, activation="relu")
layer_2 = Dense(units=15, activation="relu")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
```

Regularized MNIST model

```
layer_1 = Dense(units=25, activation="relu", kernel_regularizer=L2(0.01))
layer_2 = Dense(units=15, activation="relu", kernel_regularizer=L2(0.01))
layer_3 = Dense(units=1, activation="sigmoid", kernel_regularizer=L2(0.01))
model = Sequential([layer_1, layer_2, layer_3])
```

λ

Iterative loop of ML development

- Choose architecture - model and data
- train model
- diagnostic - bias, variance, error analysis

Building a spam classifier

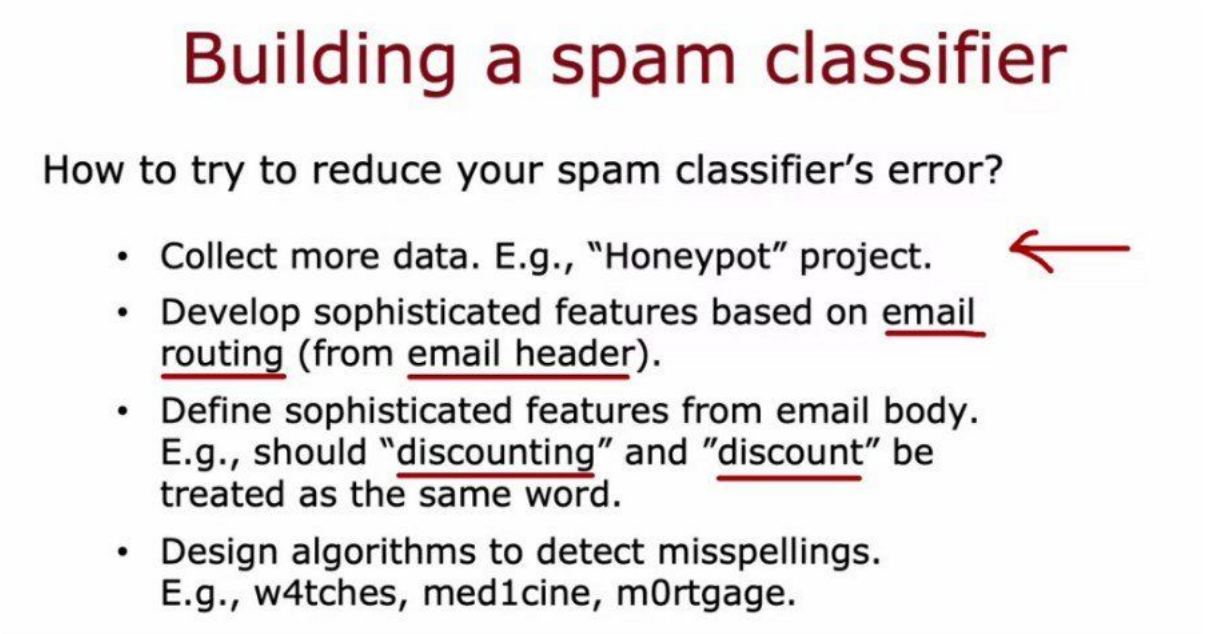
Supervised learning: $\vec{x}$ = features of email
y = spam (1) or not spam (0)

Features: list the top 10,000 words to compute $x_1, x_2, \cdots, x_{10,000}$

$$\vec{x} = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \begin{matrix} a \\ andrew \\ buy \\ deal \\ discount \\ \vdots \end{matrix}$$

From: cheapsales@buystufffromme.com
To: Andrew Ng
Subject: Buy now!

Deal of the week! Buy now!
Rolex w4tchs - $100
Med1cine (any kind) - £50
Also low cost M0rgages
available.



Error Analysis

- Manually (human) going through a misclassified small dataset and analyzing the data
- We get a idea about where to focus

Error analysis involves the iterative observation, isolating, and diagnosing erroneous Machine learning (ML) predictions.

In error analysis, ML engineers must then deal with the challenges of conducting thorough performance evaluation and testing for ML models to improve model ability and performance.

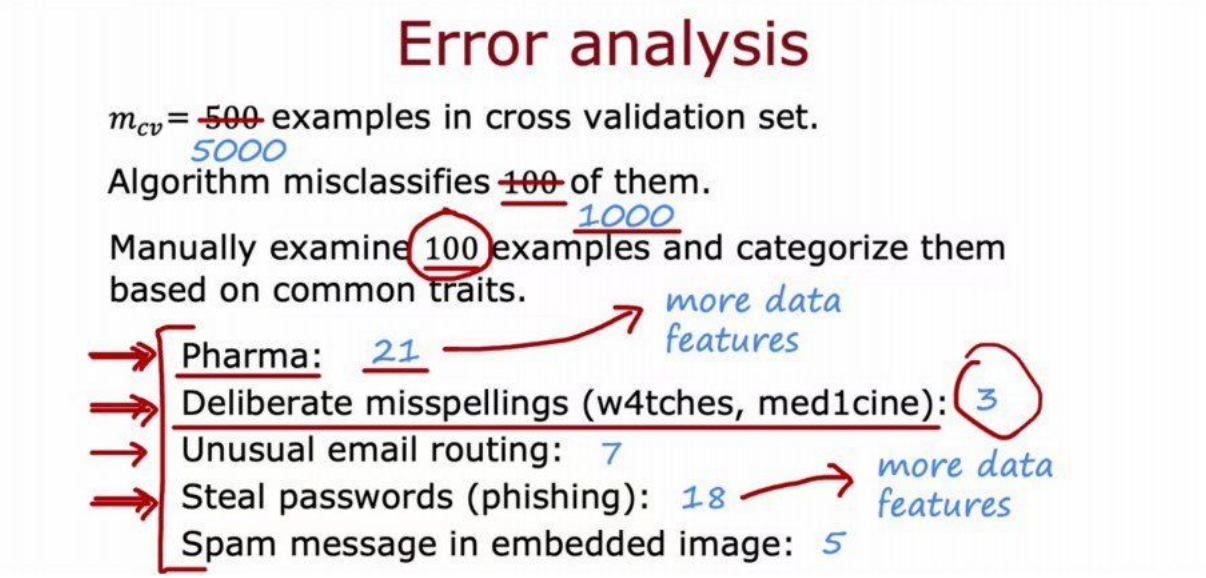
Error Analysis works by

- Identification - identify data with high error rates
- Diagnosis - enables debugging and exploring the datasets further for deeper analysis
- Model debugging

This deepcheck's model error analysis check helps identify errors and diagnose their distribution across certain features and values so that you can resolve them.

```
from deepchecks.tabular.datasets.classification import adult
from deepchecks.tabular.checks import ModelErrorAnalysis

train_ds, test_ds = adult.load_data(data_format='Dataset', as_train_test=True)
model = adult.load_fitted_model()

# We create the check with a slightly lower r squared threshold to ensure that
```

```
# the check can run on the example dataset.

check = ModelErrorAnalysis(min_error_model_score=0.3)
result = check.run(train_ds, test_ds, model)
result

# If you want to only have a look at model performance at pre-defined
# segments, you can use the segment performance check.

from deepchecks.tabular.checks import SegmentPerformance
SegmentPerformance(feature_1='workclass',
                   feature_2='hours-per-week').run(validation_ds, model)
```



Adding Data

- Include data where error analysis pointed

Data Augmentation

Kind of feature engineering where we make more data from existing features

- Image Recognition
  - rotated image
  - enlarged image
  - distorted image
  - compressed image.
- Speech Recognition

- Original audio
- noisy background - crowd
- noisy background - car
- audio on a bad cellphone connection

For Image Text Recognition we can make our own data by taking screenshot of different font at different color grade

- **Synthetic Data Creation** is also now a inevitable part of majority projects.
- Focus on Data not the code

What is Transfer Learning ?

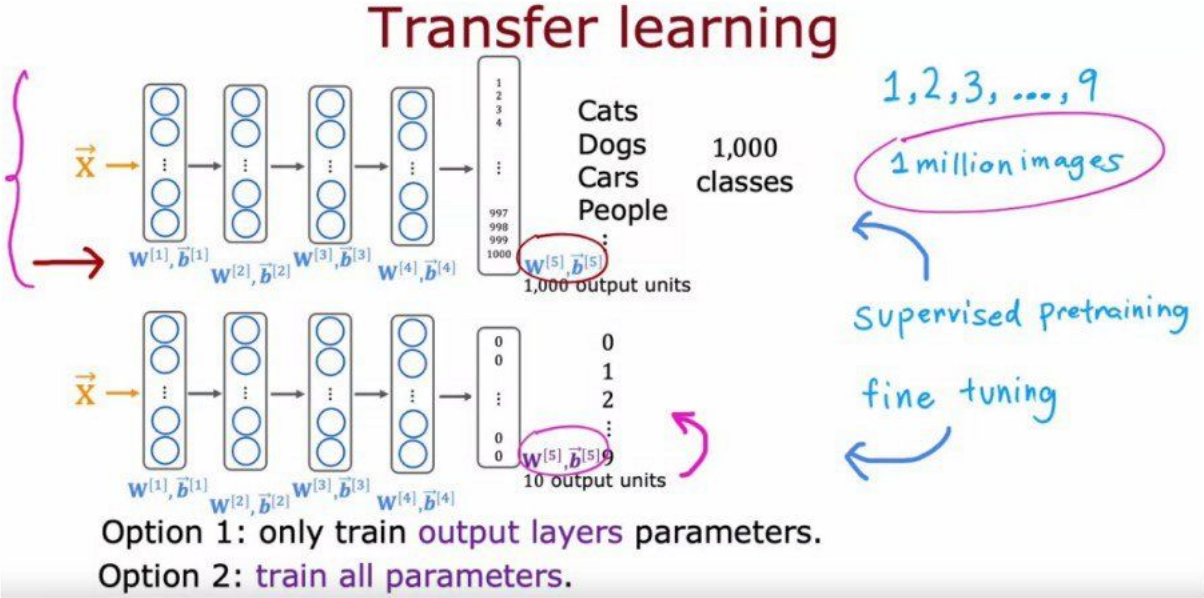
Transfer learning make use of the knowledge gained while solving one problem and applying it to a different but related problem (same type of input like, image model for image and audio model for audio.

For example, knowledge gained while learning to recognize cars can be used to some extent to recognize trucks.

- We want to recognize hand written digits from 0 to 9
- We change the last output layer we wanted from the large NN all ready pre build.

Pre Training

When we train the network on a **large dataset(for example: ImageNet)** , we train all the parameters of the neural network and therefore the model is learned. It may take hours on your GPU.

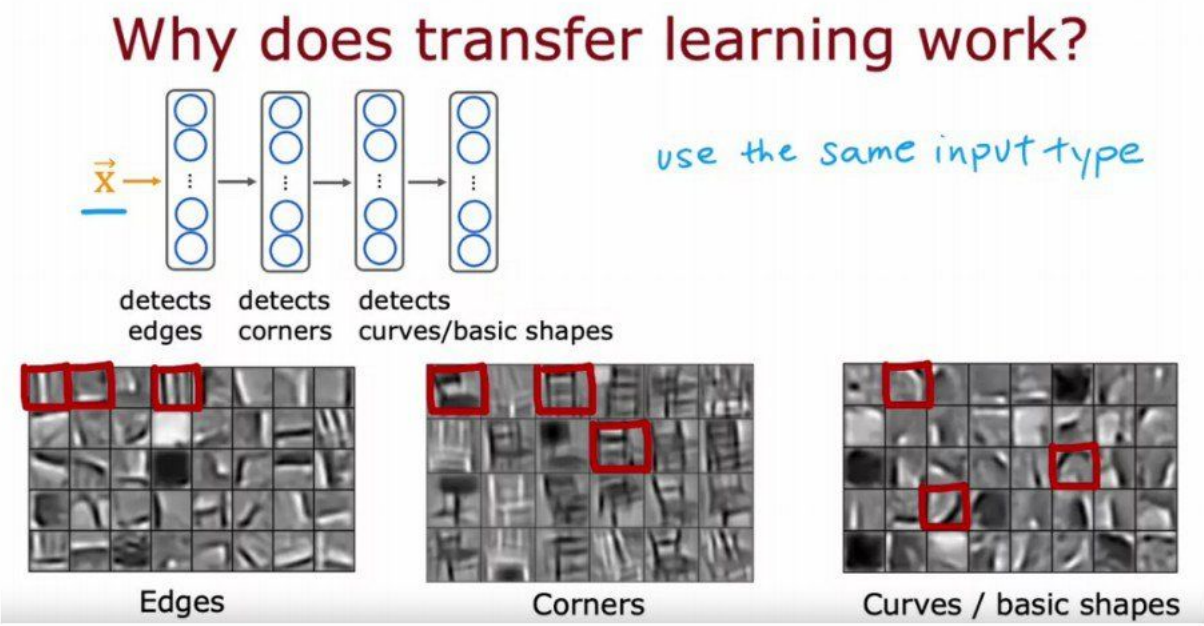


Fine Tuning

We can give the new dataset to fine tune the pre-trained CNN. Consider that the new dataset is almost similar to the orginal dataset used for pre-training. Since the new dataset is similar, the same weights can be used for extracting the features from the new dataset.

1. If the new dataset is very small, it's better to train only the final layers of the network to avoid overfitting, keeping all other layers fixed. So remove the final layers of the pre-trained network. Add new layers **Retrain only the new layers.**
2. **If the new dataset is very much large, retrain the whole network** with initial weights from the pretrained model.

How to fine tune if the new dataset is very different from the orginal dataset ?



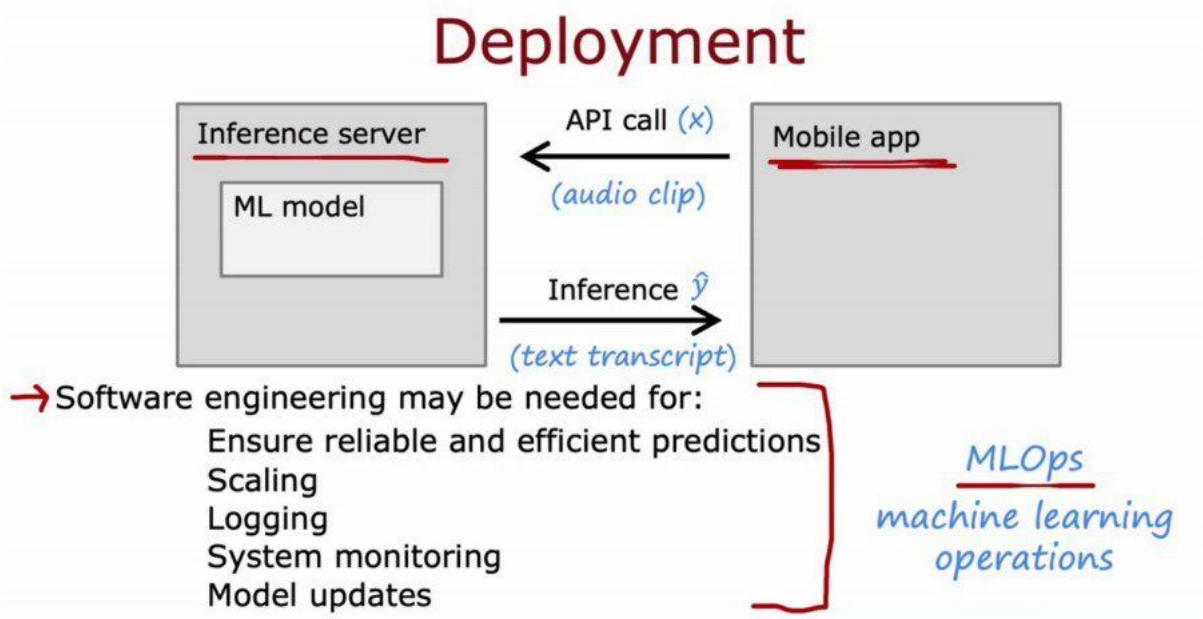
The earlier features of a ConvNet contain more **generic features (e.g. edge detectors or color blob detectors)**, but later layers of the ConvNet becomes progressively more specific to the details of the **classes contained in the original dataset.**

The earlier layers can help to extract the features of the new data. So it will be good if you fix the earlier layers and retrain the rest of the layers, if you got only small amount of data.

If you have large amount of data, you can retrain the whole network with weights initialized from the pre-trained network.

Full cycle of a machine learning project

- Define Project
- Define and Collect data
- Train/Error Analysis/Iterative Improvement
- Deploy Monitor and Maintain



Fairness, Bias, and Ethics

Bias

- Hiring tool to discriminate women.
- Facial Recognition system matching dark skinned individuals to criminal mugshot.
- Biased bank loan approval
- Toxic effect of reinforcing negative stereotypes
- Using DeepFakes to create national issues/political purpose
- Spreading toxic speech for user engagement
- Using ML to build harmful products, commit fraud etc

Guidelines

- Have a diverse team.
- Carry standard guidelines for your industry.
- Audit system against possible harm prior to deployment.
- Develop mitigation plan, monitor possible harm. (If self driving car get involved in accident)

Mitigation Plan - Reduces loss of life and property by minimizing the impact of disasters.

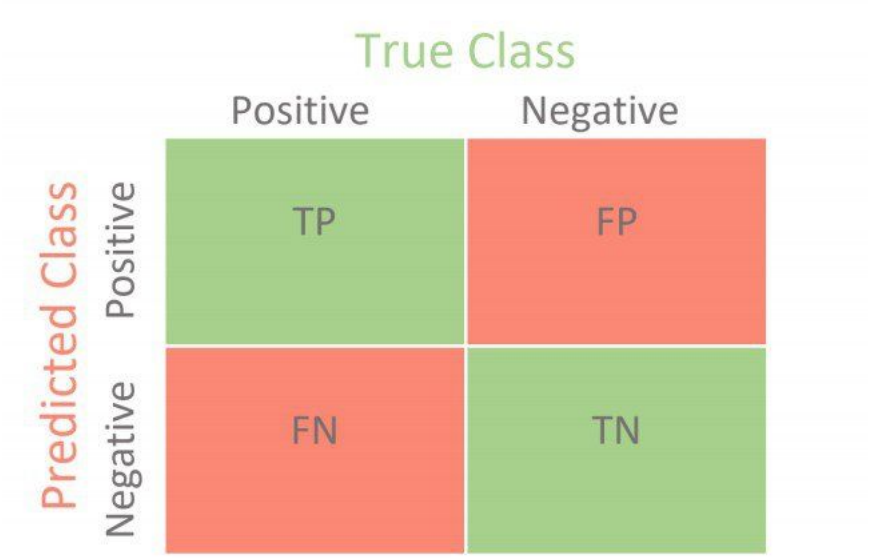
Error Metrics

- For classifying a rare disease, Accuracy is not best evaluation metrics
- Precision Recall and F1Score helps to measure the classification accuracy

Confusion Matrix in Machine Learning

Confusion Matrix helps us to display the performance of a model or how a model has made its prediction in Machine Learning.

Confusion Matrix helps us to visualize the point where our model gets confused in discriminating two classes. It can be understood well through a 2×2 matrix where the row represents the actual truth labels, and the column represents the predicted labels.



Accuracy

Simplest metrics of all, Accuracy. Accuracy is the ratio of the total number of correct predictions and the total number of predictions.

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision

Precision is the ratio between the True Positives and all the Positives. For our problem statement, that would be the measure of patients that we correctly identify having a heart/rare

disease out of all the patients actually having it.

Eg : Suppose I predicted 10 people in a class have heart disease. Out of those how many actually I predicted right.

Precision = TP / (TP + FP)

Recall

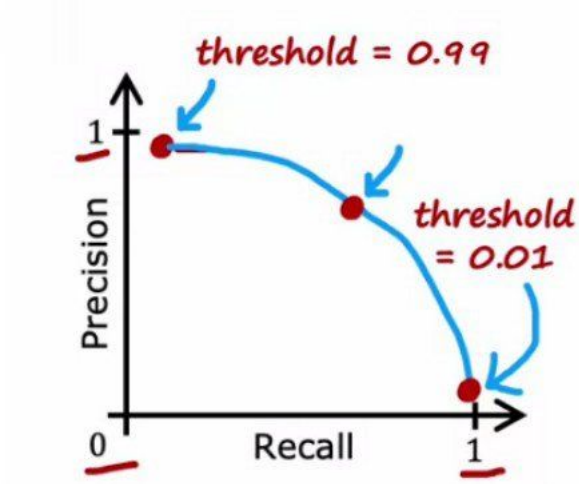
The recall is the measure of our model correctly identifying True Positives. Thus, for all the patients who actually have heart disease, recall tells us how many we correctly identified as having a heart disease.

Eg : Out of all people in a class having heart disease how many I got right prediction.

Recall = TP / (TP + FN)

Trading off Precision and Recall

- If we want to predict 1 (rare disease) only if we are really confident, then
  - High Precision and Low Recall
  - We set high threshold, predict 1 if $f(x) \geq 0.99$
- If we want to predict 1 (rare disease) when in small doubt, then
  - Low Precision and High Recall
  - We set small threshold, predict 1 if $f(x) \geq 0.01$



F1 Score

For some other models, like classifying whether a bank customer is a loan defaulter or not, it is desirable to have a high precision since the bank wouldn't want to lose customers who were denied a loan based on the model's prediction that they would be defaulters.

There are also a lot of situations where both precision and recall are equally important. For example, for our model, if the doctor informs us that the patients who were incorrectly classified as suffering from heart disease are equally important since they could be indicative of some other ailment, then we would aim for not only a high recall but a high precision as well.

In such cases, we use something called F1-score is used . F1-score is the Harmonic mean of the Precision and Recall

Precision, Recall and F1 Score should be close to one, if it is close to zero then model is not working well. (General case)

$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Decision Tree Model

- If features are categorical along with a target, for a classification problem decision tree is a good model
- Starting of DT is called **Root Node**
- Nodes at bottom is **Leaf Node**
- Nodes in between them is **Decision Node**
- The purpose of DT is to find best tree from the possible set of tree

Cat classification example

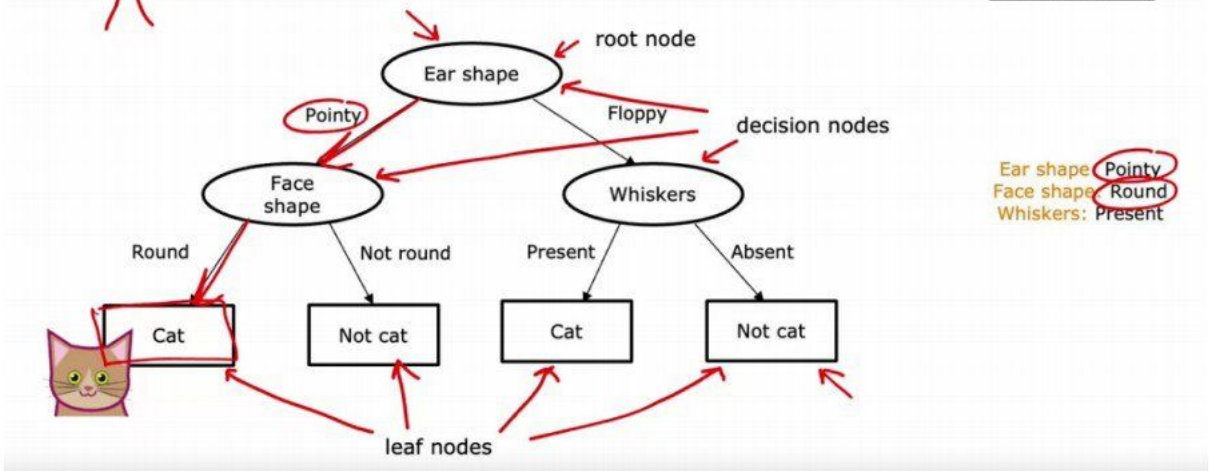
	Ear shape (x_1)	Face shape (x_2)	Whiskers (x_3)	Cat
	Pointy	Round	Present	1
	Floppy	Not round	Present	1
	Floppy	Round	Absent	0
	Pointy	Not round	Present	0
	Pointy	Round	Present	1
	Pointy	Round	Absent	1
	Floppy	Not round	Absent	0
	Pointy	Round	Absent	1
	Floppy	Round	Absent	0
	Floppy	Round	Absent	0

Categorical (discrete values)

X

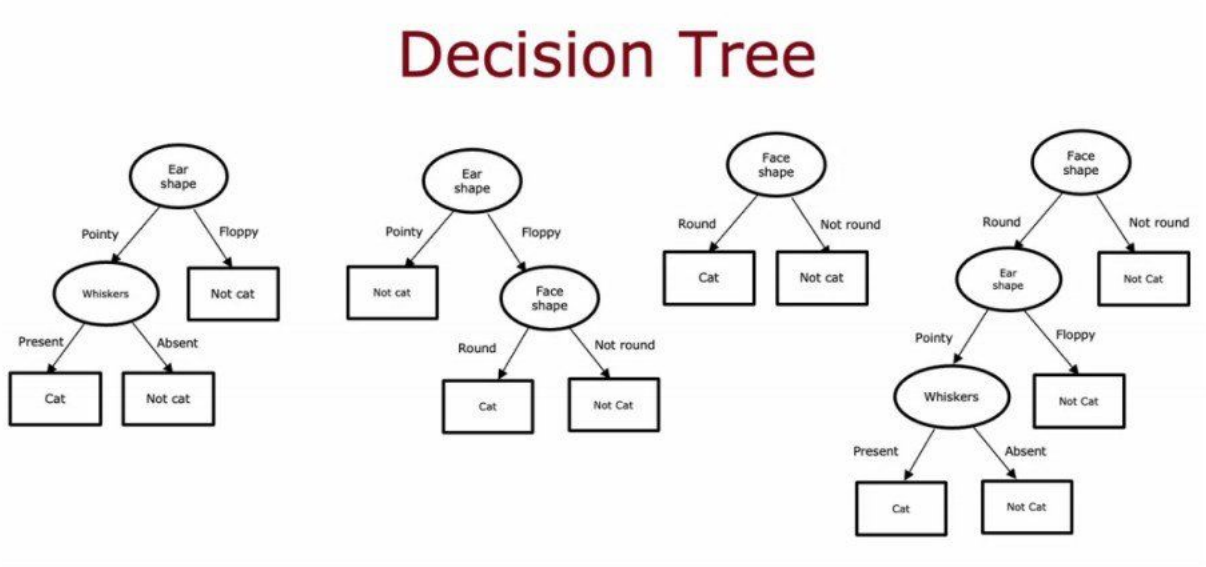
y

Decision Tree



New test example

Ear shape: Pointy
Face shape: Round
Whiskers: Present



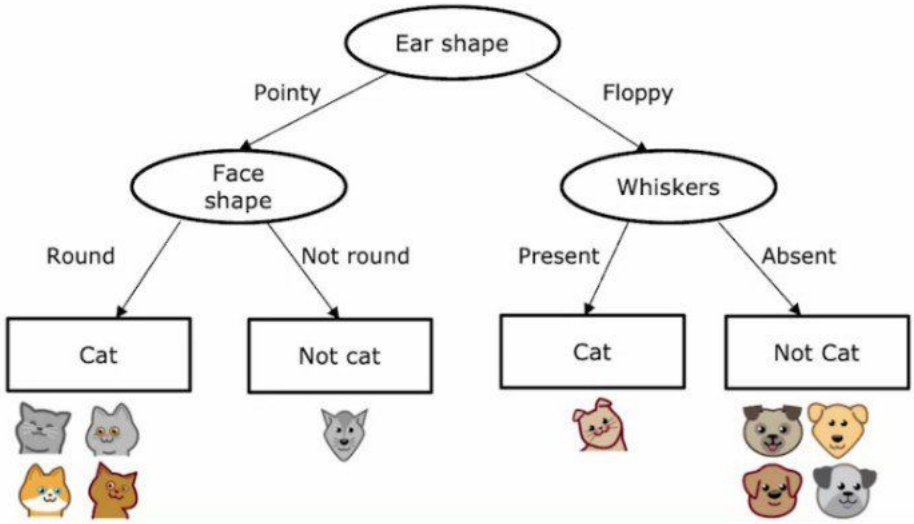
Decision Tree Learning Process

- If we reach at a node full of specific class, we stop there and set the node as leaf node.
- we want to achieve purity, that is class full of same type (Not completely possible all time).
- In order to maximize the purity, we need to decide where we need to split on at each node.

When to stop splitting?

- When node is 100% one class
- Reaching maximum depth of tree
- When depth increases - Overfitting
- When depth decreases - Underfitting
- when impurity score improvements are below a threshold
- Number of examples in a node is below a threshold

Decision Tree Learning

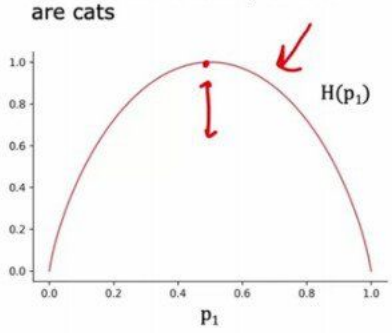


Measuring Purity

- **Entropy is measure of Impurity/randomness in data**
- It starts from 0 to 1 then goes back to 0
- P1 is the fraction of positive examples that are same class
- P0 is opposite class
- Entropy equation is as follows;

Entropy as a measure of impurity

p_1 = fraction of examples that are cats



$$p_0 = 1 - p_1$$

$$H(p_1) = -p_1 log_2(p_1) - p_0 log_2(p_0)$$
$$= -p_1 log_2(p_1) - (1 - p_1) log_2(1 - p_1)$$

Note: "0 log(0)" = 0

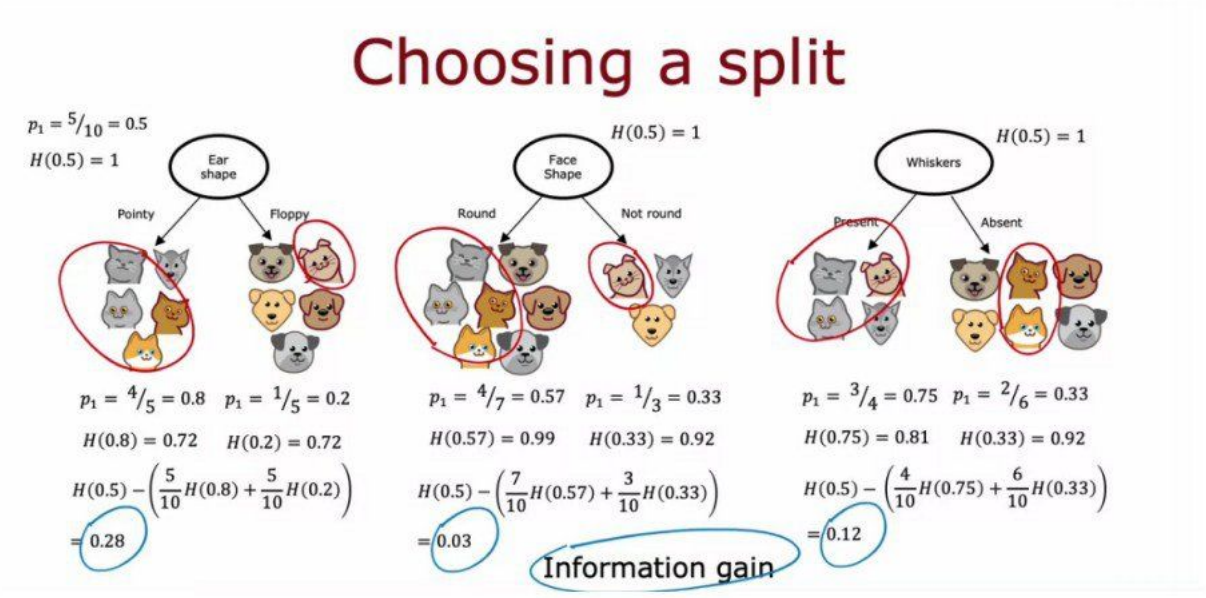
Choosing a split: Information Gain

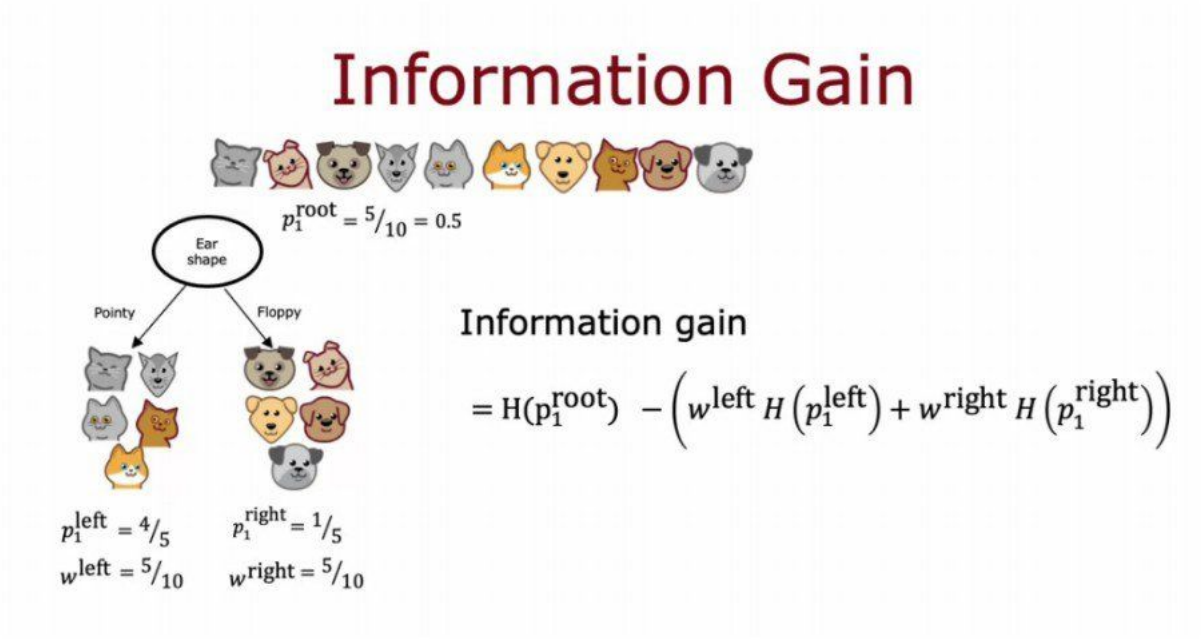
Reduction of Entropy is Information Gain

Some Initial Steps

- We calculate the Entropy in each decision node - Low Entropy is Good, Shows Less Impurity
- Also number of elements in sub branch is also important - High Entropy in Lot of Samples is Worse
- We need to combine the Entropy of two side somehow
- First calculate product of ratio of elements in node with Entropy. Take sum of both of that value from each side.
- Subtract from root node P1

By this we get reduction in entropy know as Information Gain. Then pick the largest Information gain for best output in decision tree.





Putting it together

- Start with all examples at root node
- Calculate Information Gain for all features and pick one with large IG
- Split data into left and right branch
- keep repeating until,
  - Node is 100% one class
  - When splitting result in tree exceeding depth. Because it result in Overfitting
  - If information gain is less than a threshold
  - if number of examples in a node is less than a threshold

Decision Tree uses Recursive Algorithm

Using one-hot encoding of categorical features

- One hot encoding is a process of converting categorical data variables to features with values 1 or 0.
- One hot encoding is a crucial part of feature engineering for machine learning.
- In case of binary class feature we can covert in just one column with either 0 or 1
- Can be used in Logistic/Linear Regression or NN

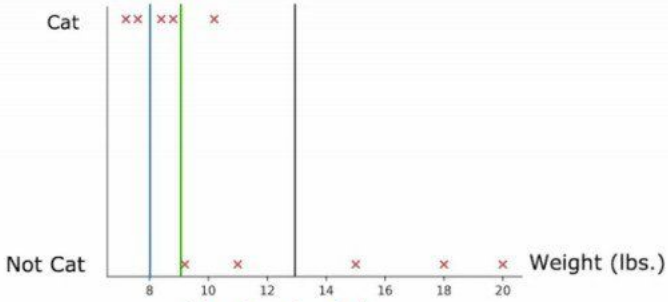
One hot encoding and neural networks

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

Decision Tree for Continuous valued features

- Below example weight is continuous
- We can consider weight ≤ x lbs, to split into 2 classes
- The value of x can be changed and calculate Information Gain for each case.
- General case is to use 3 cases but can be changed.

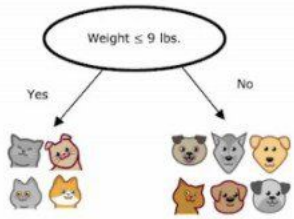
Splitting on a continuous variable



$$H(0.5) - \left(\frac{2}{10}H\left(\frac{2}{2}\right) + \frac{8}{10}H\left(\frac{3}{8}\right)\right) = 0.24$$

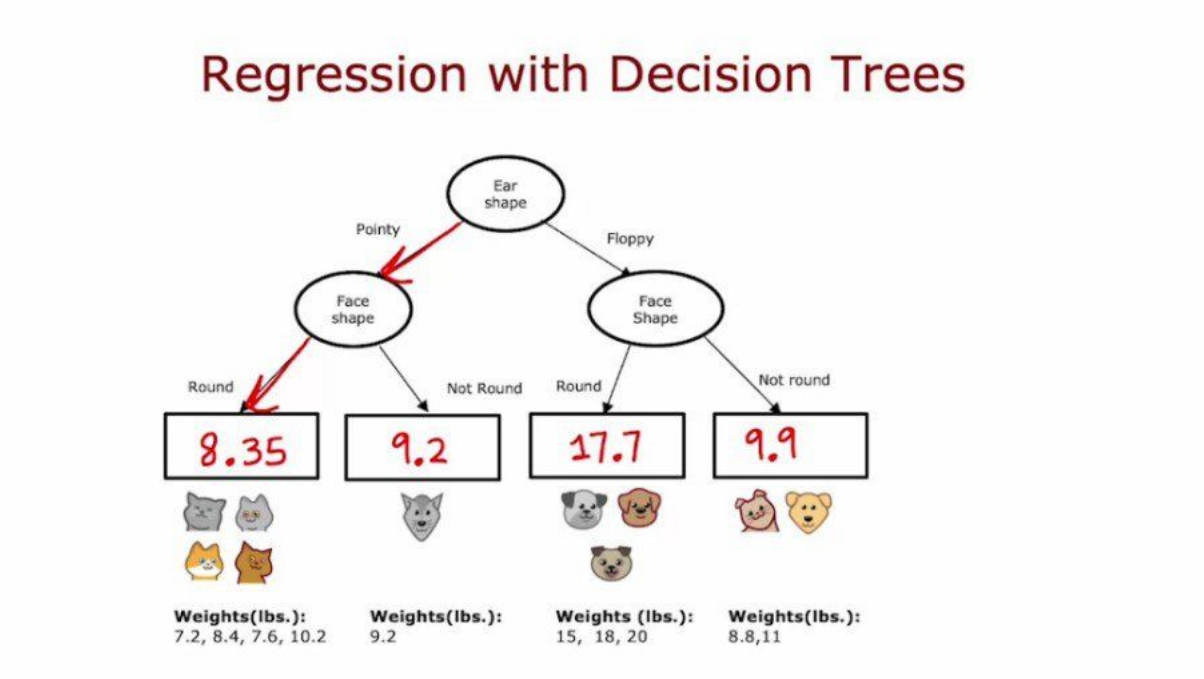
$$H(0.5) - \left(\frac{4}{10}H\left(\frac{4}{4}\right) + \frac{6}{10}H\left(\frac{1}{6}\right)\right) = 0.61$$

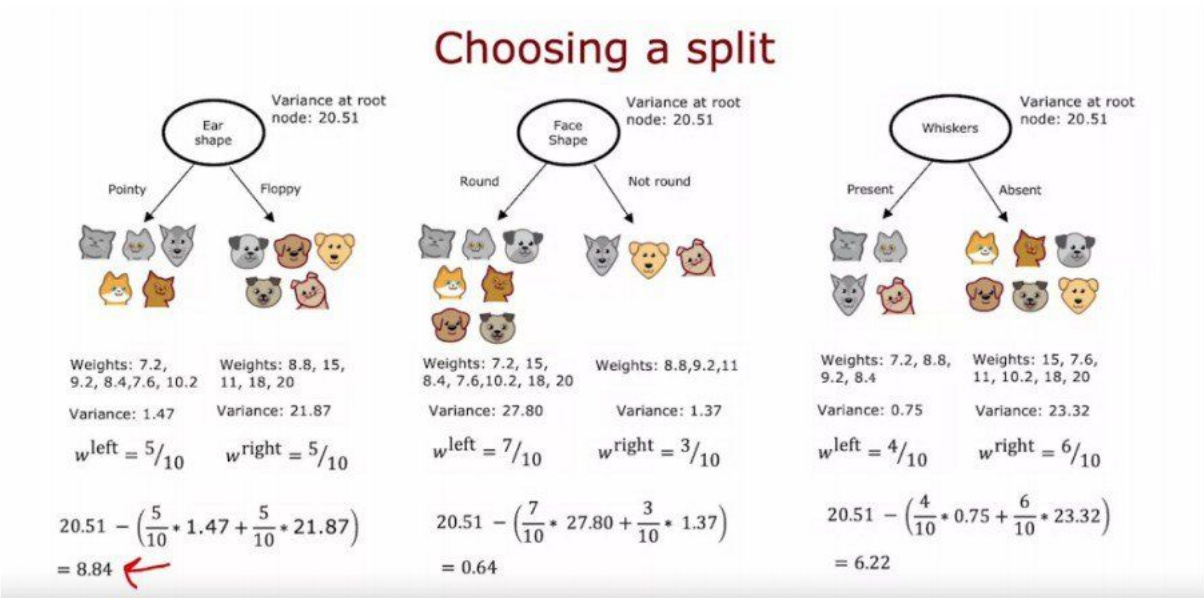
$$H(0.5) - \left(\frac{7}{10}H\left(\frac{5}{7}\right) + \frac{3}{10}H\left(\frac{0}{3}\right)\right) = 0.40$$



Regression Trees

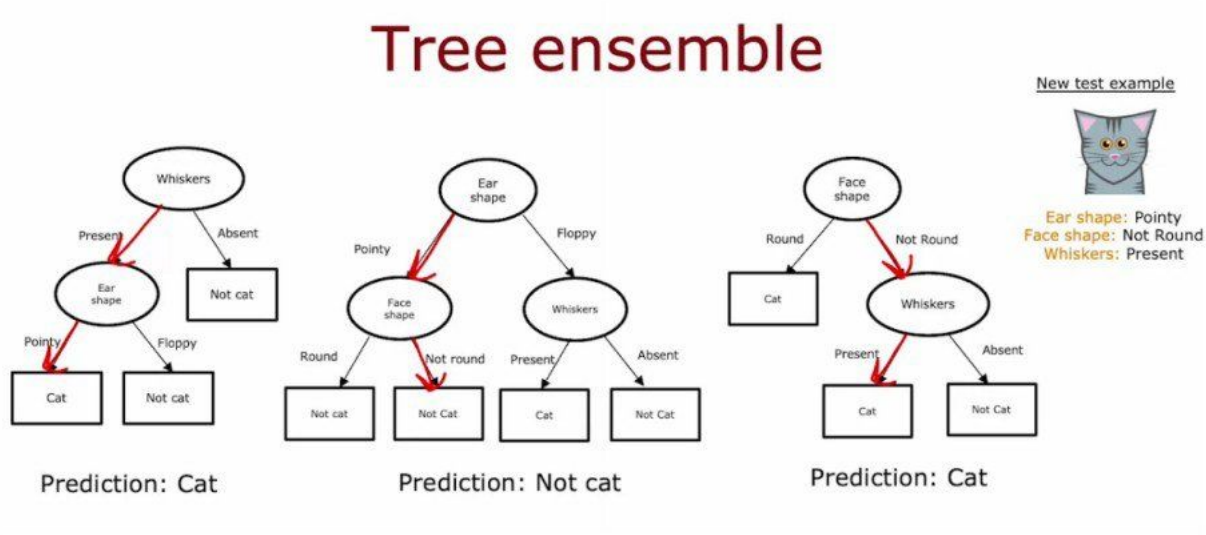
- It is same as Decision problem. At the end when we reach leaf node we calculate the average and predict
- But instead of Information Gain we calculate **Reduction in Variance**
- Calculate the variance of the leaf node then multiply by the ratio of number of elements, finally subtract it from variance of root node
- We choose split which give **Largest Reduction in Variance**





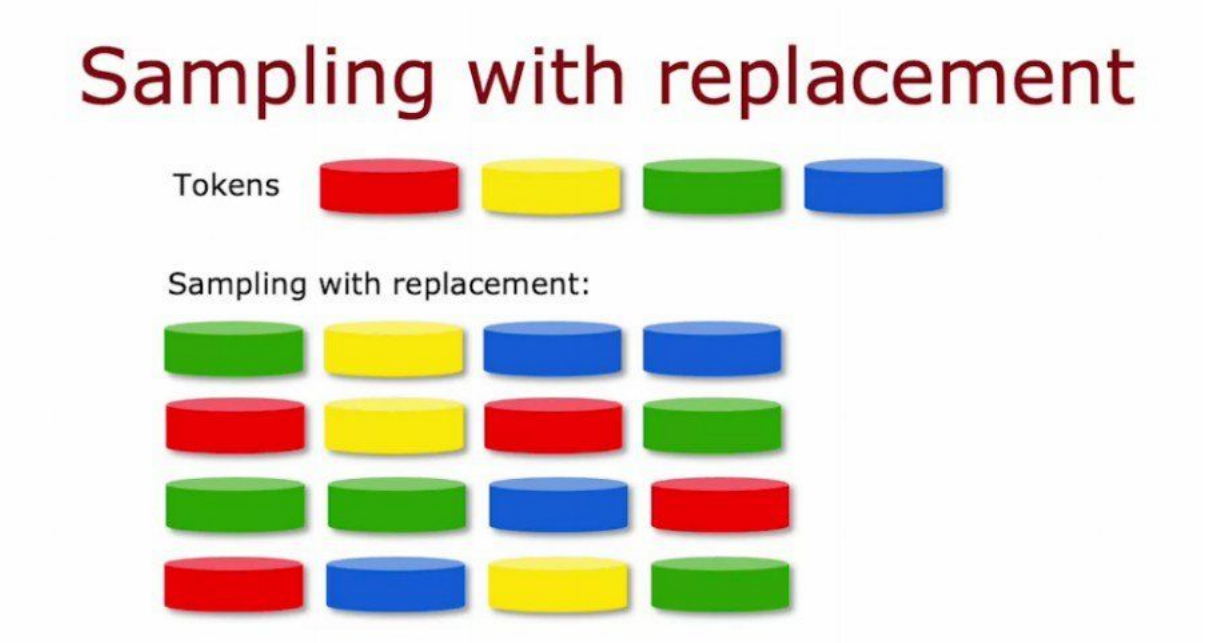
Using Multiple Decision Trees

- To avoid the sensitivity and make model robust we can use Multiple Decision Tree called **Tree ensembles (collection of multiple tree)**
- We take majority of the Ensemble prediction

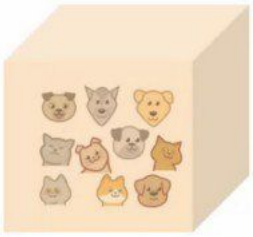


Sampling with replacement

- Help to construct new different but similar training set.



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

Random Forest Algorithm

- Pick a train data
- make duplicate of train data using sampling by replacement technique, Increasing this number of synthetic samples is ok. It do improve performance. But after a limit it is just a waste of GPU.
- Since we create new Decision Tree using this technique, it is also called **Bagged Decision Tree**
- Sampling helps Random Forest to understand small changes.
- Using different Decision Tree and averaging it improve robustness of Random Forest

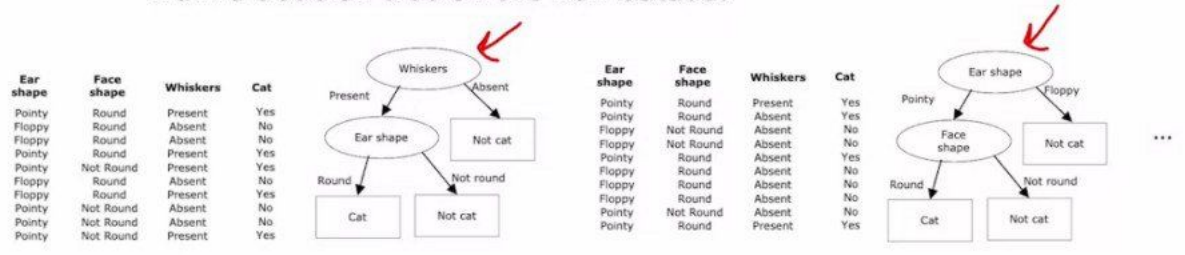
Generating a tree sample

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



Bagged decision tree

Suppose that in our example we had three features available rather than picking from all end features, we will instead pick a random subset of K less than N features. And allow the algorithm to choose only from that subset of K features. So in other words, you would pick K features as the allowed features and then out of those K features choose the one with the highest information gain as the choice of feature to use the split. When N is large, say n is Dozens or 10's or even hundreds. A typical choice for the value of K would be to choose it to be square root of N.

XGBoost

- Same like Random Forest, but in sampling while choosing next sample, we give more preference for picking the wrongly predicted one in the First made Decision Tree.

Boosted trees intuition

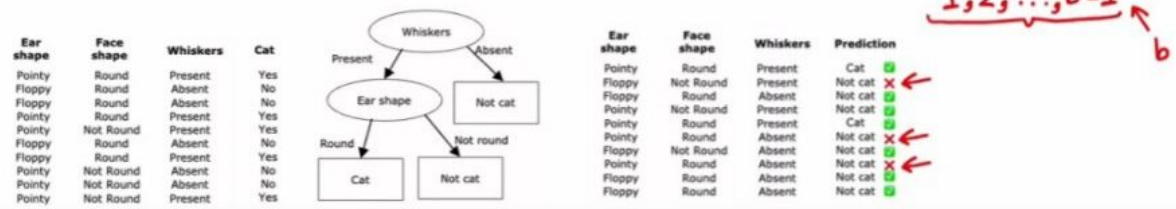
Given training set of size m

For $b = 1$ to B:

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

But instead of picking from all examples with equal (1/m) probability, make it more likely to pick misclassified examples from previously trained trees

Train a decision tree on the new dataset



Ear shape	Face shape	Whiskers	Cat
Pointy	Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Absent	No
Pointy	Round	Present	Yes
Pointy	Not Round	Present	Yes
Floppy	Round	Absent	No
Floppy	Round	Present	Yes
Pointy	Not Round	Absent	No
Pointy	Not Round	Absent	No
Pointy	Not Round	Present	Yes

Ear shape	Face shape	Whiskers	Prediction
Pointy	Round	Present	Cat
Floppy	Not Round	Present	Not cat
Floppy	Round	Absent	Not cat
Pointy	Not Round	Present	Not cat
Pointy	Round	Present	Cat
Pointy	Round	Absent	Not cat
Floppy	Not Round	Absent	Not cat
Pointy	Round	Absent	Not cat
Floppy	Round	Absent	Not cat
Floppy	Round	Absent	Not cat

XGBoost (eXtreme Gradient Boosting)

- Open source implementation of boosted trees
- Fast efficient implementation
- Good choice of default splitting criteria and criteria for when to stop splitting
- Built in regularization to prevent overfitting
- Highly competitive algorithm for machine learning competitions (eg: Kaggle competitions)

```
# Classification
from xgboost import XGBClassifier
model = XGBClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)

# Regression
from xgboost import XGBRegressor
model = XGBRegressor()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
```

When to use Decision Trees

Decision Tree and Tree Ensembles	Neural Networks
Works well on tabular data	Works well on Tabular (Structured and Unstructured data)
Not recommended for Images, audio and text	Recommended for Image, audio, and text
fast	slower than DT
Small Decision tree may be human interpretable	works with transfer learning
We can train one decision tree at a time.	when building a system of multiple models working together, multiple NN can be stringed together easily. We can train them all together using gradient descent.

Unsupervised Learning, Recommenders, Reinforcement Learning - Course 3

What is Clustering?

- Clustering is used to group unlabeled data
- There are various algorithms to perform the Clustering.