Coursera

20/7

Advanced Learning Algorithms

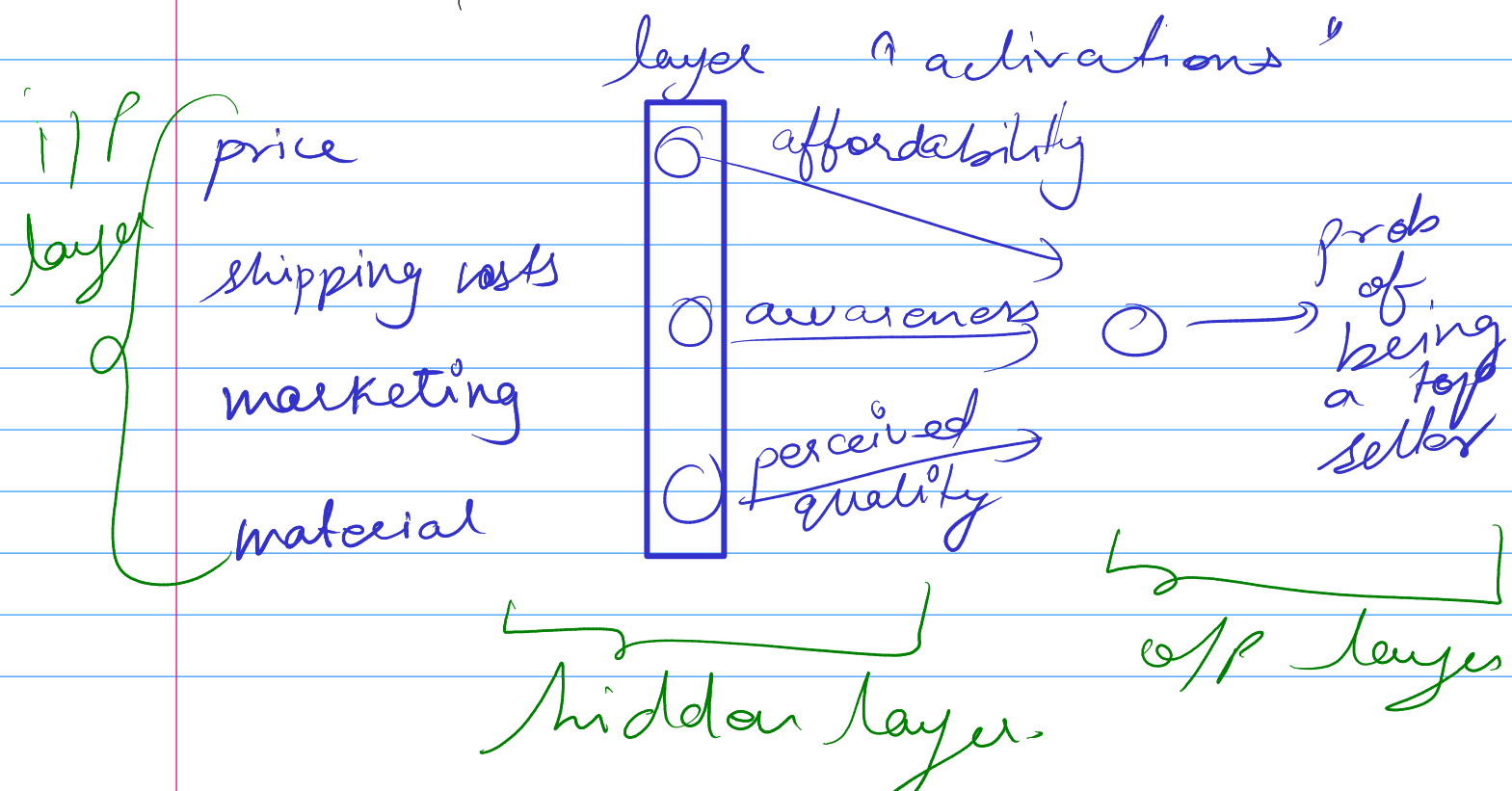
Week 1 — Neural Networks.
  ↳ Inference.
Week 2 → Training.
Week 3 → How to build ML systems.
Week 4 → Decision Trees.

Week 1

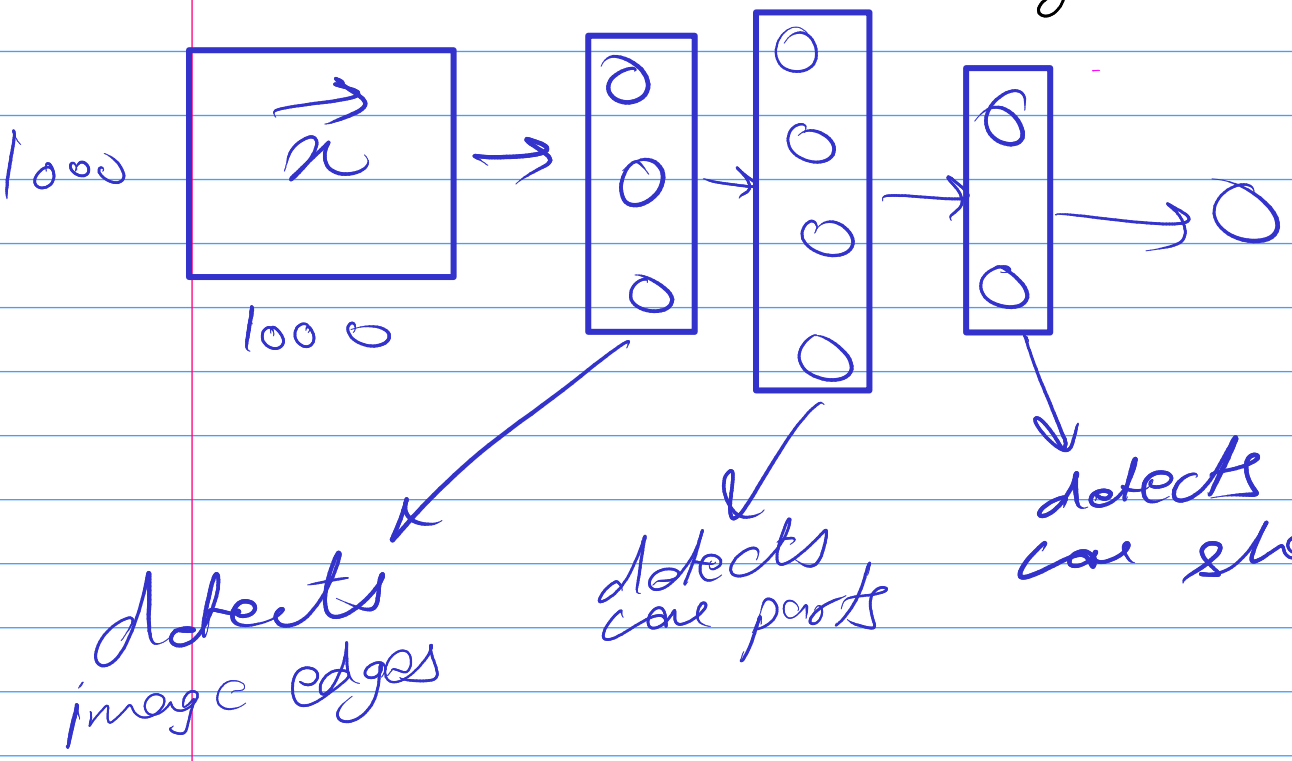
Demand Prediction

$$a = f(x) = \frac{1}{1 + e^{-(wx+b)}}$$

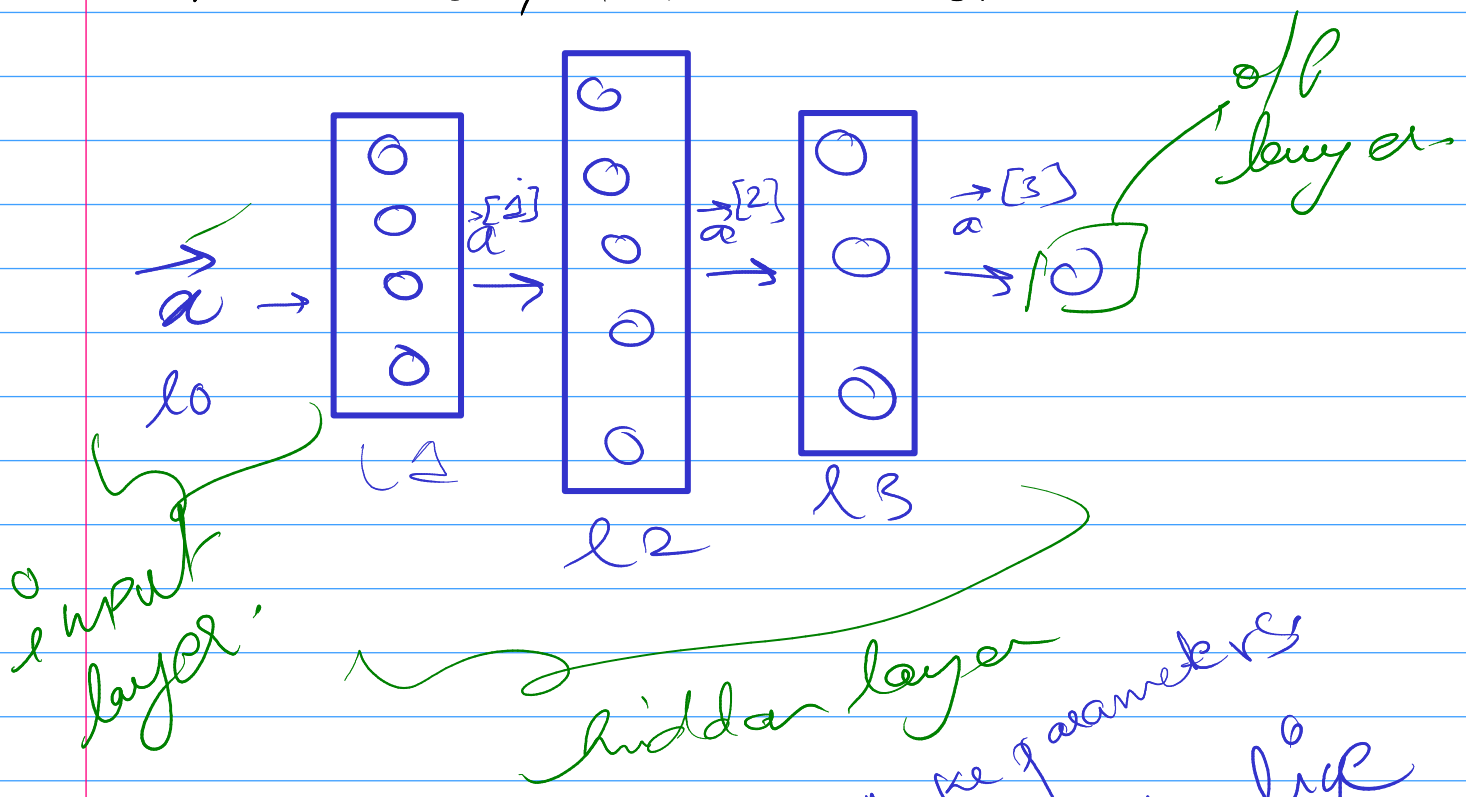
→ activation function.



Car Classification using NN



More Complex Neural Networks.



Layer 3 zoomed

layer 3 ke parameters

layer 3 ke liye

$w_1, b_1 \rightarrow a_1^{[3]} = g(w_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]})$

$w_2, b_2 \rightarrow a_2^{[3]} = g(w_2^{[3]} \cdot \vec{a}^{[3]} + b_2^{[3]})$

General formula $\rightarrow$ $a_j^{[l]} = g(\vec{w}_j^{[l]} \cdot \vec{a}^{[l-1]} + b_j^{[l]})$

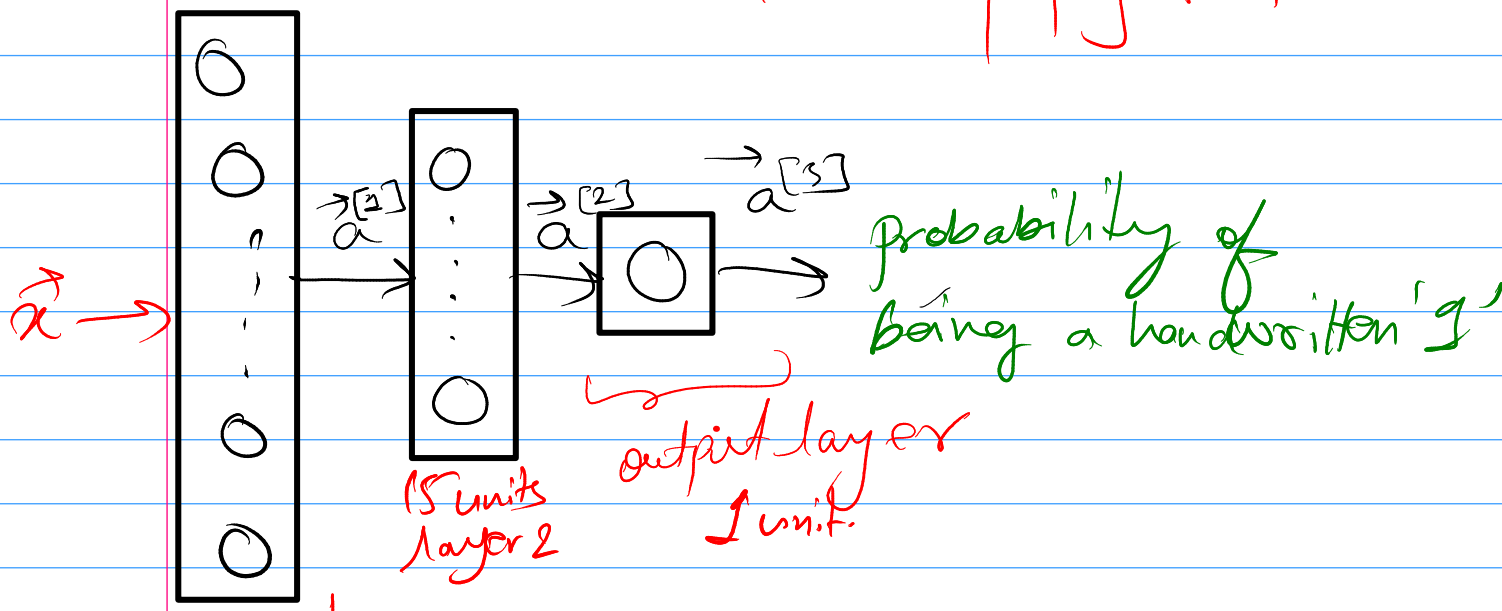
$l \rightarrow$ denotes layer l.

$j \rightarrow$ denotes unit j } Neuron no.

g → sigmoid function

Inference: making predictions.

→ Forward propagation



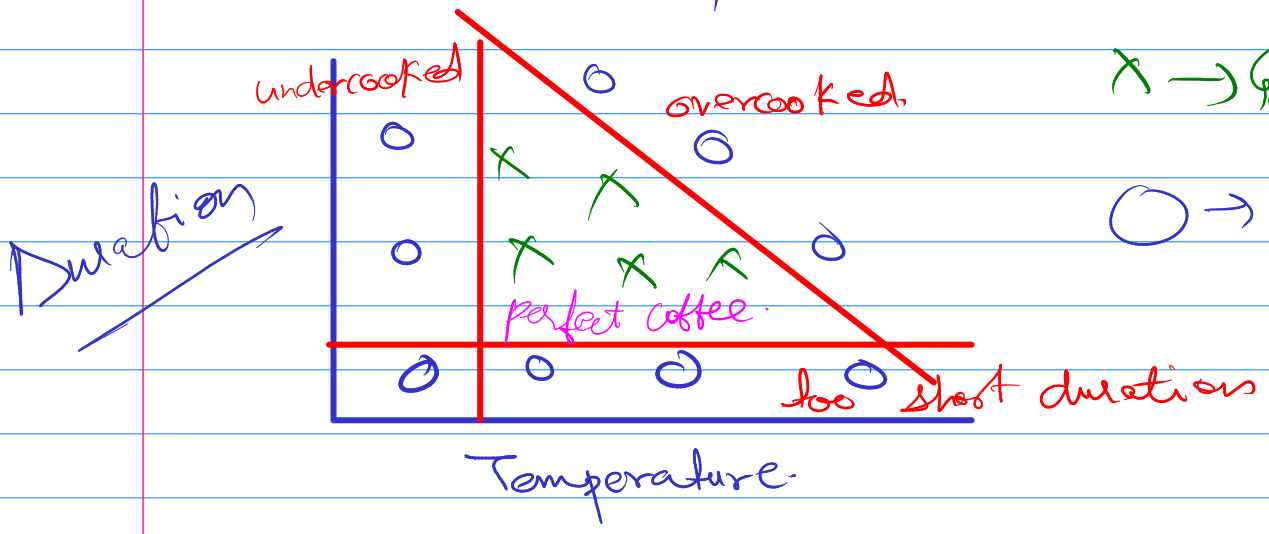
$$\vec{a}^{[1]} = \begin{bmatrix} g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]}) \\ \vdots \\ g(\vec{w}_{25}^{[1]} \cdot \vec{x} + b_{25}^{[1]}) \end{bmatrix}$$

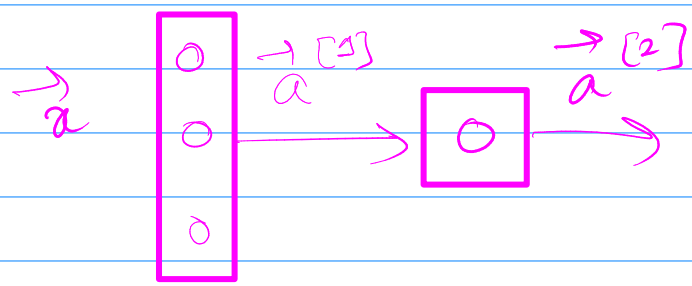
g ← sigmoid

Tensorflow Implementation

Coffee Roasting Example.

Imagine we have raw coffee beans and we want to roast it. The roasting params we can control are duration & temperature.





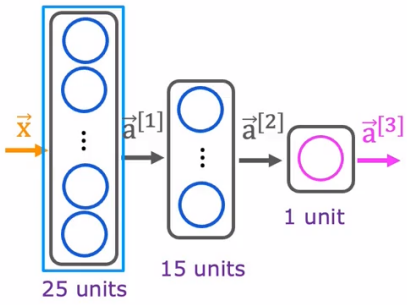
is $a_1^{[2]} \geq 0.5$?

yes → 1 no → 0

```
x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
```

```
layer_2 = Dense(units='1', activation="sigmoid")
a2 = layer_2(a1)

if a2 >= 0.5:
    yhat = 1
else:
    yhat = 0
```




```
x = np.array([[0.0 ..., 245.0 ...]])
l1 = Dense(units='25', activation='sigmoid')
a1 = l1(a0)

l2 = Dense(units='15', activation='sigmoid')
a2 = l2(a1)

l3 = Dense(units='1', activation='sigmoid')
a3 = l3(a2)

if a3 >= 0.5:
    yhat = 1
else:
    yhat = 0
```

How is data represented in Tensorflow.

Note about numpy arrays

3 columns

2 rows

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$

2 x 3 matrix

```
x = np.array([[1, 2, 3],
              [4, 5, 6]])
```

```
[[1, 2, 3],
 [4, 5, 6]]
```

$\begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -.5 & -.6 \\ 7 & 8 \end{bmatrix}$

```
x = np.array([[0.1, 0.2],
              [-3.0, -4.0,],
              [-0.5, -0.6,],
              [7.0, 8.0,]])
```

```
[[0.1, 0.2],
 [-3.0, -4.0,],
 [-0.5, -0.6,],
 [7.0, 8.0,]]
```

DeepLearning.AI Stanford ONLINE Andrew Ng

2, 3 is the first row of this matrix and 4, 5,

Note about numpy arrays

They are 2D matrix

```
x = np.array([[200, 17]])   → [200  17]      1 x 2 row vector
```

```
x = np.array([[200],        → [200]          2 x 1 column vector
              [17]])          [17]
```

This is only an array

```
x = np.array([200,17])      just a 1D array.
```

1D "vector"

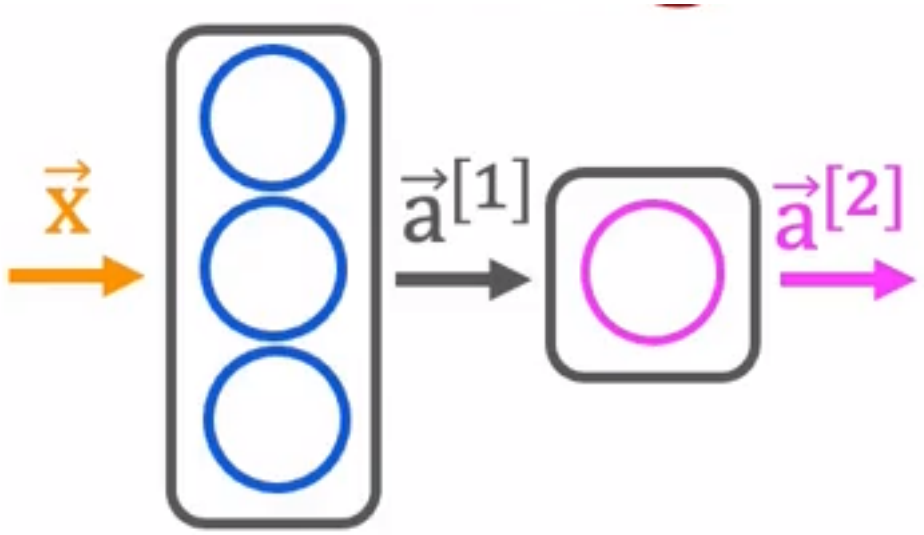
DeepLearning.AI Stanford ONLINE Andrew Ng

no columns, but it's just a list of numbers.

Note: In tensorflow the convention is to use matrices to represent data

—— x ——

Building a neural network.




```
layer_1 = Dense(units=3, activation='sigmoid')
layer_2 = Dense(units=1, activation='sigmoid')

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

Tells Tensorflow to sequentially string together these layers into a neural network.

```
x = np.array([[200.0, 17.0],
              [120.0, 5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
```

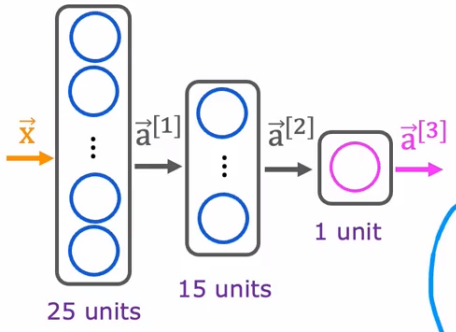
```
y = np.array([1,0,0,1])

model.compile(...)
model.fit(x,y)

model.predict(x_new)
```

Another example

Digit classification model



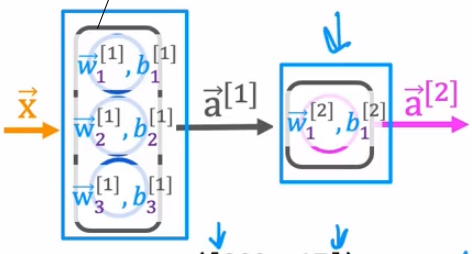

```
layer_1 = Dense(units=25, activation="sigmoid")
layer_2 = Dense(units=15, activation="sigmoid")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
model.compile(...)
x = np.array([[0..., 245, ..., 17],
              [0..., 200, ..., 184]])
y = np.array([1,0])
model.fit(x,y)
model.predict(x_new)
```

more about this next week!

X new and similar to what you saw before with the coffee classification network

Forward prop in a single layer.

forward prop (coffee roasting model)



$w_1^{[2]}$ w2_1

```
x = np.array([200, 17])
```

1D arrays

Break down to steps

$a_1^{[1]} = g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]})$

```
w1_1 = np.array([1, 2])
b1_1 = np.array([-1])
z1_1 = np.dot(w1_1,x)+b1_1
a1_1 = sigmoid(z1_1)
```

$a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]})$

```
w1_2 = np.array([-3, 4])
b1_2 = np.array([1])
z1_2 = np.dot(w1_2,x)+b1_2
a1_2 = sigmoid(z1_2)
```

$a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$

```
w1_3 = np.array([5, -6])
b1_3 = np.array([2])
z1_3 = np.dot(w1_3,x)+b1_3
a1_3 = sigmoid(z1_3)
```

```
a1 = np.array([a1_1, a1_2, a1_3])
```

Fed to layer 2 as input

DeepLearning.AI Stanford ONLINE Andrew Ng

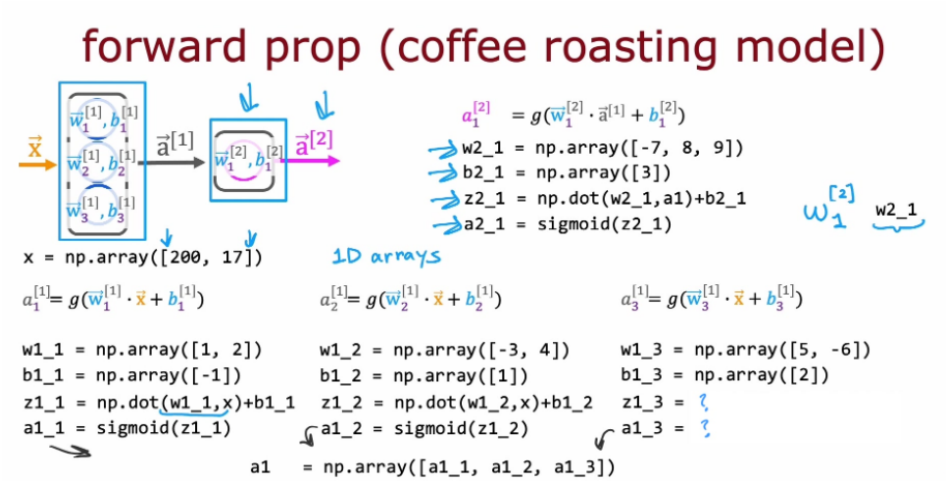
General Implementation of Fwd. propagation.

```
def dense(a_in, W, b):
    units = W.shape[0]          # all cols
    a_out = np.zeros(units)
    for j in range(units):      # traverse all rows of jth column.
        w = W[:, j]
        z = np.dot(w, a_in) + b[j]
        a_out = g(z)
    return a_out
```

Del params from prev layer, activations of new layer, and sends it to next layer.

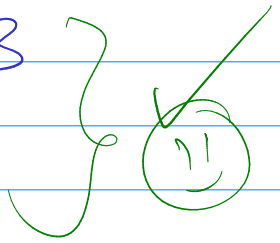
```
def sequential(x):
    a1 = dense(x, W1, b1)
    a2 = dense(a1, W2, b2)
    a3 = dense(a2, W3, b3)
    a4 = dense(a3, W4, b4)
    f_x = a4
    return f
```

1 point



1. According to the lecture, how do you calculate the activation of the third neuron in the first layer using NumPy?

z1_3 = np.dot(w1_3, x) + b1_3
a1_3 = sigmoid(z1_3)



Is there a path to AGI?

AI

ANI

Artificial Narrow Intelligence

↓

An AI system that does a particular task very well. Its narrowed down to do a task of a set of tasks.

AGI

Artificial General Intelligence.

↓

Do anything a human can do.

Vectorization

How neural networks are implemented efficiently.

Using for loops

```
x = np.array([200, 17])

W = np.array([[1, -3, 5],
              [-2, 4, -6]])
b = np.array([-1, 1, 2])

def dense(a_in,W,b):
  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
```

[1,0,1]

Vectorization

```
X = np.array([[200, 17]])
W = np.array([[1, -3, 5],
              [-2, 4, -6]])
B = np.array([[-1, 1, 2]])

def dense(A_in,W,B):
  Z = np.matmul(A_in,W) + B
  A_out = g(Z)
  return A_out

[[1,0,1]]
```

2D array

same

1 x 3 2D array

vectorized

matrix multiplication

matmul

v/s

Out short LOC's

Flow of using Tensorflow:

load_data()

↓

check the dimensions of the data

↓

model = Sequential([
Input Shape,
layers
- - -
- - -
], name = "my_model")

↓

model.summary()

↓

Examine the weights:

↓

W1, b1 = layer_1.get_weights()

↓

model.compile(
loss = tf.keras.losses.BinaryCrossentropy(),
optimizer = tf.keras.optimizers.Adam(0.001)
)

↓

model.fit(
X, y
epochs = 20
)

prediction = model.predict(X[0].reshape(1,400))

prediction = model.predict(X[500].reshape(1,400))

```
if prediction >= .5 :
    yhat = 1
else :
    yhat = 0
```

Note : Confusion b/w w and w[i]

Dot Product Requirement

`np.dot(a_in, w)` → dono ka shape `(n,)` hona chahiye.

- Agar tum `w[i]` likhoge → `w[i]` ek **scalar** ban jayega (sirf ek weight pick ho gaya) → pura dot product kharab ho jayega.
- Dot product hamesha **poora vector** ke saath hota hai → isliye `w = W[:, i]` lena zaroori hai.

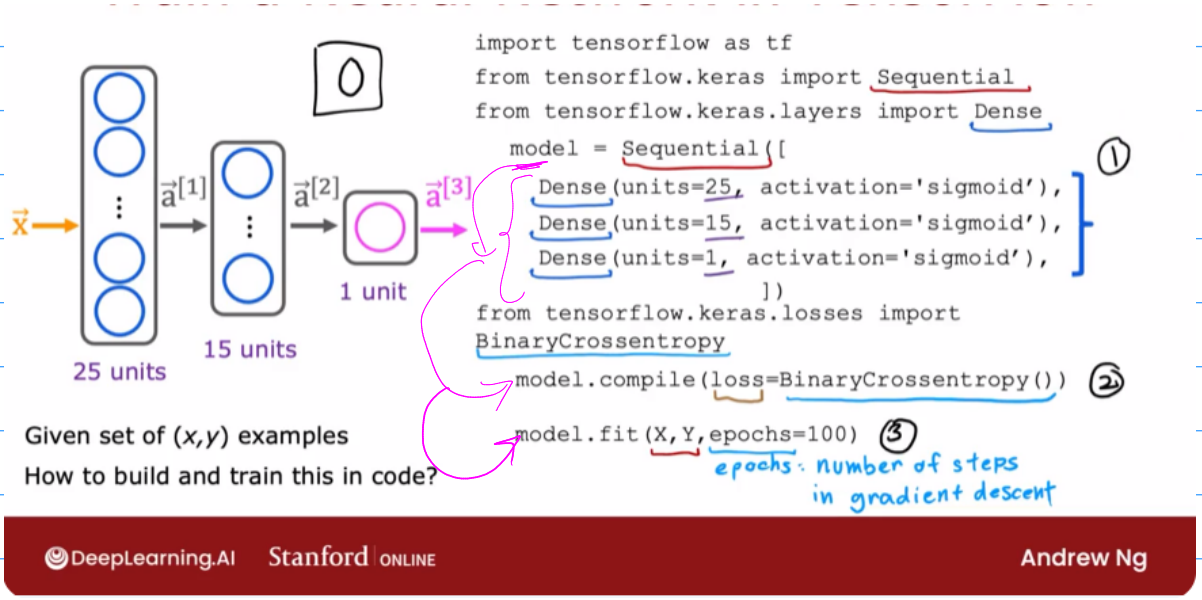
Rule of Thumb

- **Poora column of weights** (ek neuron ke liye) → `W[:, i]` (vector)
- **Ek single scalar weight** (sirf ek feature ka ek weight) → `W[i, j]` ya `w[i]` (scalar)

Week 2

Tensorflow Implementation

Train a NN in tf. (recap)



Model Training Steps

① Define the model given $\vec{x}$, $\vec{w}$ & b.

$f_{\vec{w},b}(\vec{x}) = ?$

In logistic regression:

```
z = np.dot(w,x) + b
f_x = 1/(1+np.exp(-z))
```

(2) Specify the loss and cost.

$$L(f_{\vec{w},b}(\vec{x}), y)$$ Loss

$$J(\vec{w}, b) = \frac{1}{m}\sum_{i=1}^{m} L(f_{\vec{w},b}(\vec{x}^{(i)}), y^{(i)})$$ Cost

In Logistic Regression

```
loss = -y * np.log(f_x)
   -(1-y) * np.log(1-f_x)
```

(3) Train on data to minimize $J(\vec{w}, b)$

```
w = w - alpha * dj_dw
b = b - alpha * dj_db
```

The 3 steps in model training in Neural Networks

Step 1: `model = Sequential([`
`Dense(---),`
`---`
`---`
`---])`

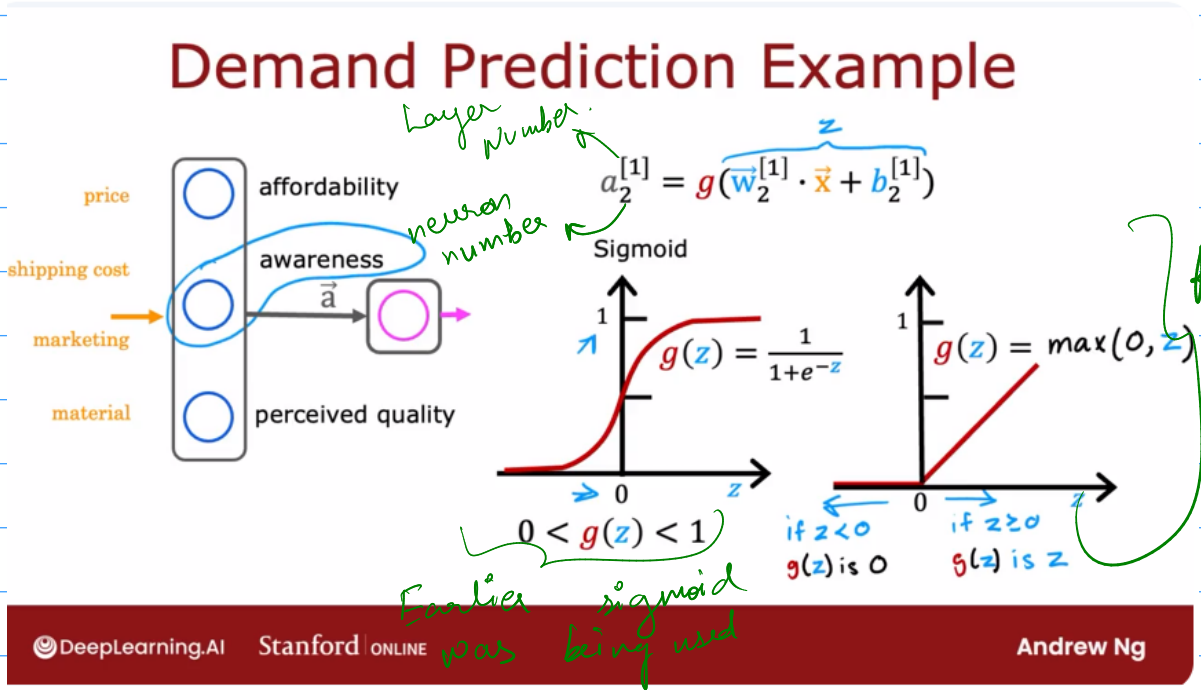
Describes the model architecture of NN

Step 2: `model.compile(loss = '----')`

Step 3: `model.fit(X, Y, epochs = 100)`

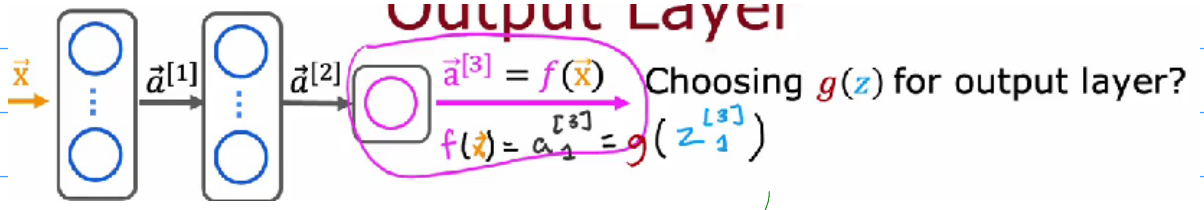
This performs backprop in background.

Alternatives to the Sigmoid Function.



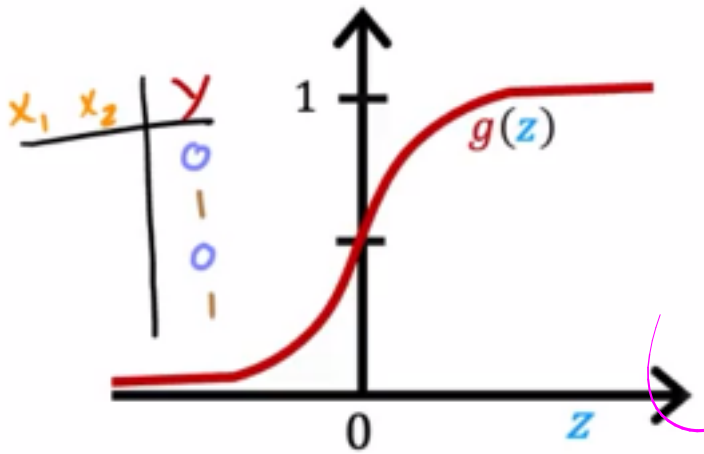
They are activation functions

Choosing an AF for Output Layer.



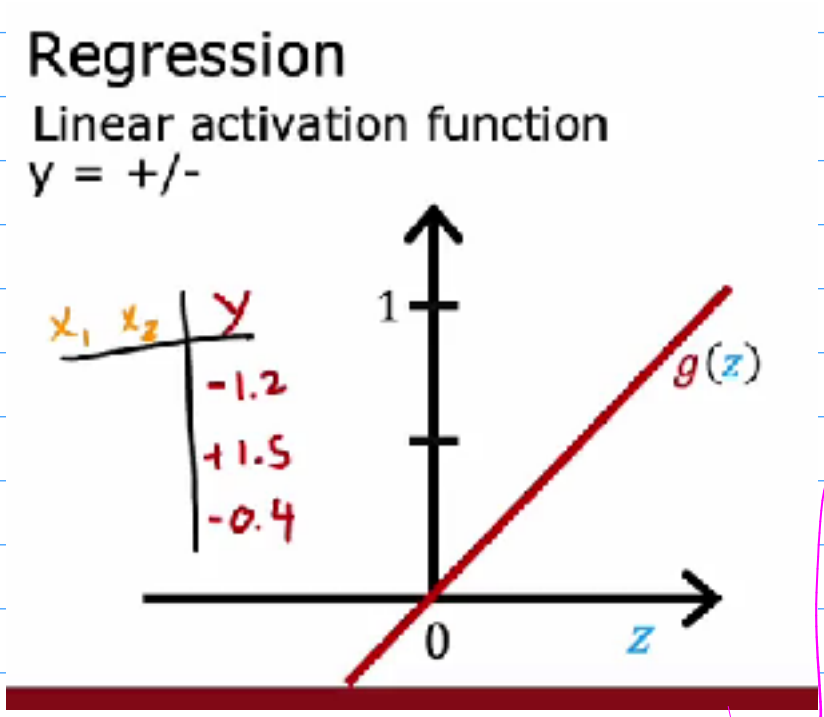
Output Layer,

if its a Binary classification (0/1) problem then we will use Sigmoid AF.



Sigmoid Activation Function,

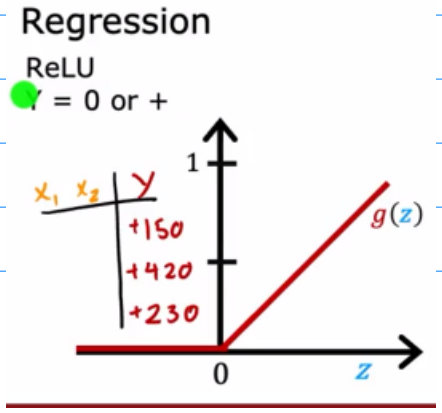
if its a regression task where the model has to prediction certain value(s) then we would use Linear Activation func?



In regression if the predicted values could be +ve or -ve then we will use Linear AF.

Regression case 2: Where the predicted values is only positive or 0, like for example predicting price of house - Price of a house can't be -ve -

In such cases we use ReLU AF,



ReLU = max(0, z)

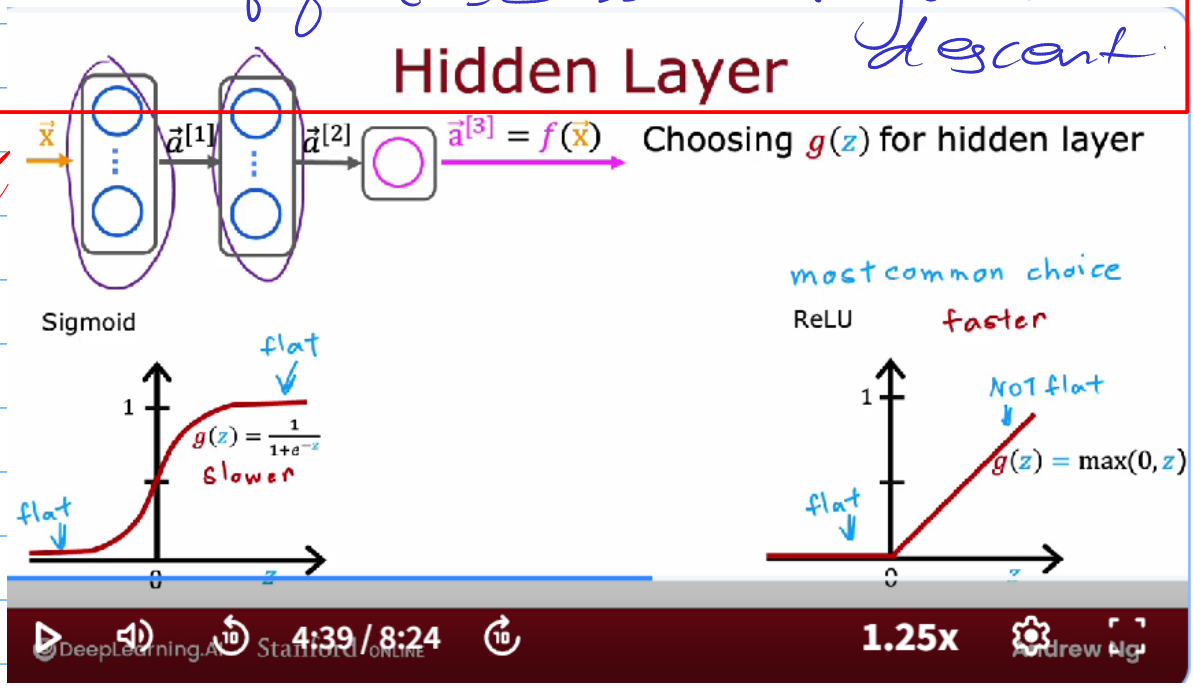
Activation function used in hidden layers.

The most commonly used AF in hidden layers is ReLU.

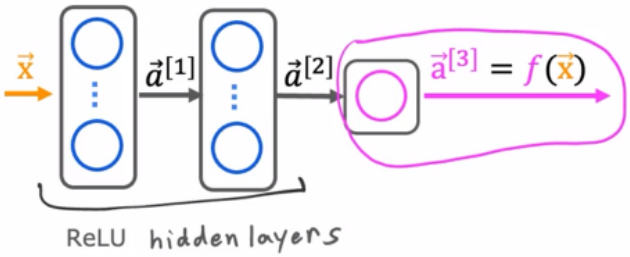
2 reasons to choose ReLU over Sigmoid for hidden layers: —

1. It's faster.
2. Sigmoid is flat on both ends while ReLU is only flat at one end.

More no. of flat == slower gradient descent.



Choosing Activation Summary



binary classification
activation='sigmoid'

regression y negative/positive
activation='linear'

regression $y \geq 0$
activation='relu'

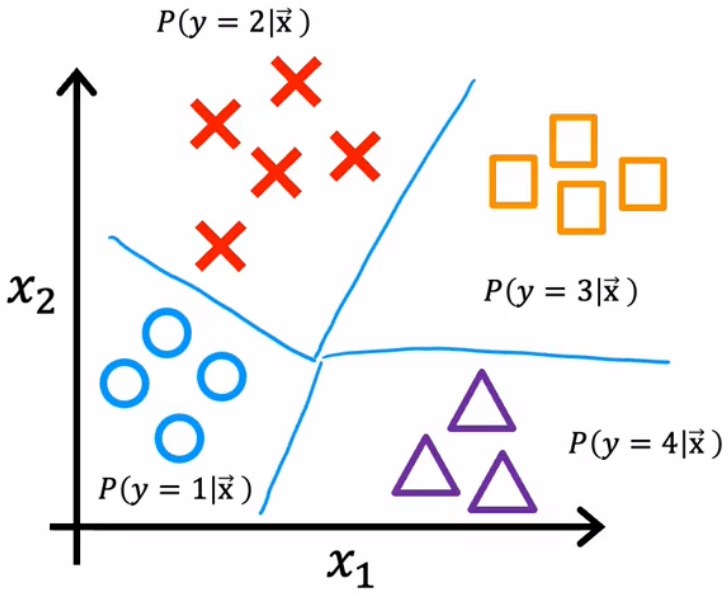
```
from tf.keras.layers import Dense
model = Sequential([
  Dense(units=25, activation='relu'),    layer1 ] hidden
  Dense(units=15, activation='relu'),    layer2
  Dense(units=1,  activation='sigmoid')  layer3 ] output
])
                  or 'linear'
                  or 'relu'
```

DeepLearning.AI Stanford ONLINE Andrew Ng

Note: Avoid using linear activation function in hidden layers, instead use ReLU.

Multi Class Classification.

It is a classification problem where the target 'y' can take on more than 2 possible values



Softmax

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

Softmax Formula.

$$z_1 = \vec{w}_1 \cdot \vec{x} + b_1 \qquad a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 1|\vec{x}) \quad 0.30$$

$$z_2 = \vec{w}_2 \cdot \vec{x} + b_2 \qquad a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 2|\vec{x}) \quad 0.20$$

$$z_3 = \vec{w}_3 \cdot \vec{x} + b_3 \qquad a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 3|\vec{x}) \quad 0.15$$

$$z_4 = \vec{w}_4 \cdot \vec{x} + b_4 \qquad a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y = 4|\vec{x}) \quad 0.35$$

Softmax Regression General Formula.

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

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

Imp

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

all probabilities sum up to 1.

Cost

Cost

Logistic regression

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

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

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

$loss = -y \log a_1 - (1-y) \log(1-a_1)$

if $y=1$ if $y=0$

$J(\vec{w}, b) =$ average loss

Softmax regression

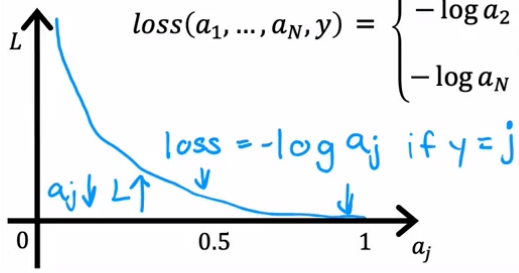
$a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=1|\vec{x})$

$\vdots$

$a_N = \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \cdots + e^{z_N}} = P(y=N|\vec{x})$

Crossentropy loss

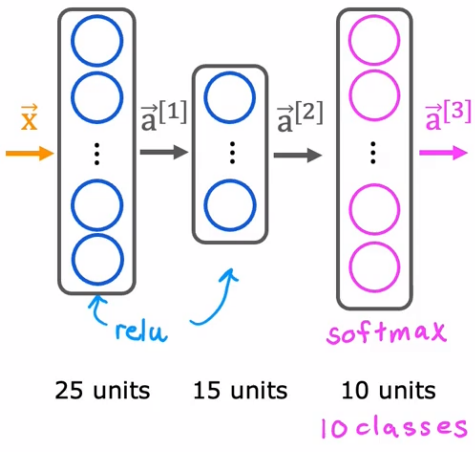
$$loss(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y=1 \\ -\log a_2 & \text{if } y=2 \\ \vdots & \\ -\log a_N & \text{if } y=N \end{cases}$$



DeepLearning.AI Stanford ONLINE Andrew Ng

Neural Network w/ Softmax Output.

Neural Network with Softmax output



$z_1^{[3]} = \vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]}$ $a_1^{[3]} = \frac{e^{z_1^{[3]}}}{e^{z_1^{[3]}} + \cdots + e^{z_{10}^{[3]}}} = P(y=1|\vec{x})$

$\vdots$

$z_{10}^{[3]} = \vec{w}_{10}^{[3]} \cdot \vec{a}^{[2]} + b_{10}^{[3]}$ $a_{10}^{[3]} = \frac{e^{z_{10}^{[3]}}}{e^{z_1^{[3]}} + \cdots + e^{z_{10}^{[3]}}} = P(y=10|\vec{x})$

gives the prob of y being any 1 of these 10 layers

MNIST with softmax

1. specify the model

$f_{\vec{w},b}(\vec{x})$ =?

```
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='softmax')
                ])
```

2. specify loss and cost

$L(f_{\vec{w},b}(\vec{x}), y)$

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

3. Train on data to minimize $J(\vec{w}, b)$

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

Note: better (recommended) version later.

Don't use the version shown here!

DeepLearning.AI Stanford ONLINE Andrew Ng

Sparse Categorical Cross Entropy

Fixed & sparse no. of categories to predict like 0 to 9.

There are more than 2 categories to predict

Improved implementation of Softmax.

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

$$\text{Loss} = L(\vec{a}, y) = \begin{cases} -\log \dfrac{e^{z_1}}{e^{z_1} + e^{z_2} + \cdots e^{z_{10}}} & \text{if } y=1 \\ \vdots & \\ -\log \dfrac{e^{z_{10}}}{e^{z_1} + e^{z_2} + \cdots e^{z_{10}}} & \text{if } y=10 \end{cases}$$

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

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

This implementation is equivalent to using Softmax in o/p layer. Why this? Coz this is more numerically accurate.

Why not to use Softmax + Cross Entropy Log-

1. Usual Softmax + Cross-Entropy Loss

- Normally:

$$\text{Softmax}(z)_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$

and cross-entropy loss:

$$L = -\log(\text{Softmax}(z)_y)$$

- This means you:
  1. Compute softmax → probabilities.
  2. Take log of the probability of the correct class.

2. Issue with Direct Softmax + Log

- If z_i is very large or very negative, e^{z_i} can overflow (∞) or underflow (0).
- Taking log of very small numbers → `-inf`, causing numerical instability.

This is not recommended and not a good industry practice.

---

3. Numerically Stable Version

- Instead of:

$$-\log\left(\frac{e^{z_y}}{\sum_j e^{z_j}}\right)$$

- We compute directly:

$$L = -z_y + \log\left(\sum_j e^{z_j}\right)$$

- And before exponentiating, we **subtract the max logit** to avoid overflow:

$$\log\left(\sum_j e^{z_j - \max z}\right) + \max z$$

- This is what libraries do internally when you set `from_logits=True`.

2nd approach: Recommended

This doesn't cause numerical instability, underflow, overflow.

This is the good practice to follow.

What is logits?

What is `from_logits=True`?

- **Logits** = raw outputs of the last layer (no activation).
- If you already applied `Softmax` before passing to the loss function, you pass `from_logits=False`.
- If your final layer is **linear** (no softmax), you tell the loss function `from_logits=True`, and it internally:
  1. Applies softmax in a numerically stable way.
  2. Calculates cross-entropy in one step.

Summary

- **Yes, more accurate**: Because it avoids overflow/underflow by combining softmax + log in one step.
- `from_logits=True`: Means you are passing **raw scores (logits)**, and the library will handle softmax internally.

Improved and Correct Code

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)
```

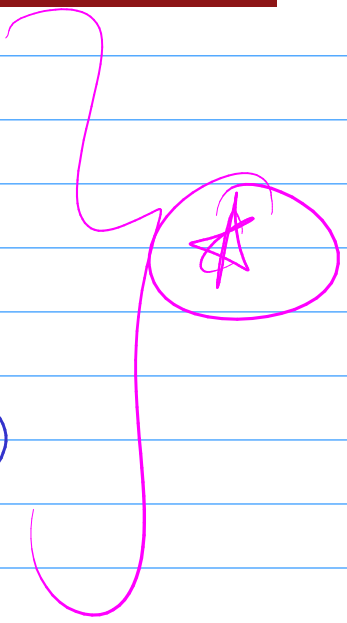
predict

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

its not outputting $a_1 \cdots a_{10}$
instead its outputting $z_1 \cdots z_{10}$

DeepLearning.AI Stanford ONLINE Andrew Ng

model = Sequential(layers) — ①
↓
model.compile(loss) — ②
↓
model.fit(X, Y, epochs) — ③
↓
logits = model(X) — ④
↓
f_x = tf.nn.softmax(logits) — ⑤



Multi Class v/s Multi Label Classification.

Multi Class : Only one output for a given i/p. Eg : Detect a digit.

Multi label : Multiple outputs for a given i/p.
Eg : Detect a Car, Bus, Person in a single image.

Imp:-

```
def softmax(z):
    ez = np.exp(z)
    sm = ez/np.sum(ez)
    return(sm)
```

The loss function associated with Softmax, the cross-entropy loss, is:

loss = -log(a_{true_label})

$$L(\mathbf{a}, y) = \begin{cases} -log(a_1), & \text{if } y = 1. \\ \vdots & \\ -log(a_N), & \text{if } y = N \end{cases} \tag{3}$$

Where y is the target category for this example and **a** is the output of a softmax function. In particular, the values in **a** are probabilities that sum to one.

> **Recall:** In this course, Loss is for one example while Cost covers all examples.

Note in (3) above, only the line that corresponds to the target contributes to the loss, other lines are zero. To write the cost equation we need an 'indicator function' that will be 1 when the index matches the target and zero otherwise.

$$\mathbf{1}\{y == n\} == \begin{cases} 1, & \text{if } y == n. \\ 0, & \text{otherwise} \end{cases}$$

Now the cost is:

$$J(\mathbf{w}, b) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{N} 1\{y^{(i)} == j\} \log \frac{e^{z_j^{(i)}}}{\sum_{k=1}^{N} e^{z_k^{(i)}}}\right] \tag{4}$$

yeh basically true label ka loss hai.

Where m is the number of examples, N is the number of outputs. This is the average of all the losses.

Indicator ka role

- Har example ke liye ek **one-hot** indicator hota hai:
  - Example 1 → [1,0,0]
  - Example 2 → [0,1,0]
  - Example 3 → [0,0,1]
- Formula mein:

$$\sum_j 1\{y = i\} \log(a_j) \rightarrow \text{sirf true label pick hota hai.}$$

Bottom line:

- Softmax se probability nikal
- True label ki probability lo
- $-\log(\text{true prob})$ ka average le lo

} Key Takeaway -

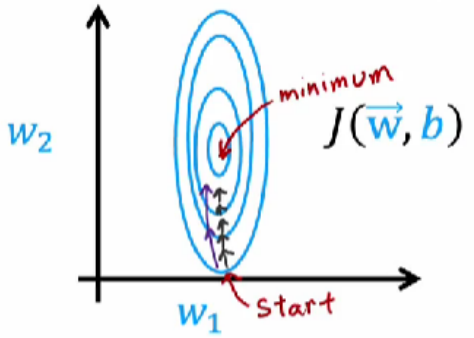
Advanced Optimization.

Adam: Adaptive Moment Estimation

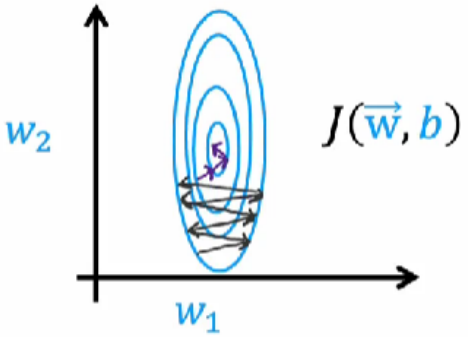
$$w_1 = w_1 - \alpha_1 \frac{\partial}{\partial w_1} J(\vec{w}, b)$$
$$\vdots$$
$$w_{10} = w_{10} - \alpha_{10} \frac{\partial}{\partial w_{10}} J(\vec{w}, b)$$
$$b = b - \alpha_{11} \frac{\partial}{\partial b} J(\vec{w}, b)$$

if there are 11 params including b then there would be 11 α's.

Adam Algorithm Intuition



If w_j (or b) keeps moving in same direction, increase α_j.



If w_j (or b) keeps oscillating, reduce α_j.

MNIST & Adam

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

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

$\alpha = 10^{-3}$ initial learning rate

fit

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

Adam is works than traditional Gradient Descent algorithm and is the defacto standard on how practitioners train their neural networks.

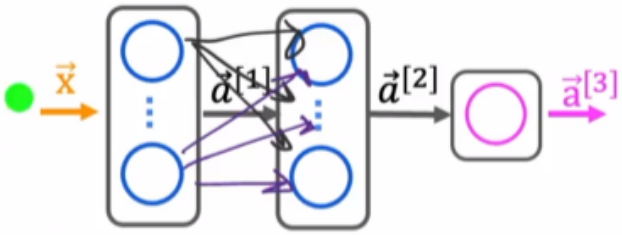
—X—

Additional Layer Types.

RECAP:

1. Dense.

Dense Layer



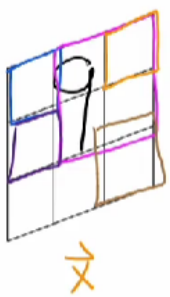
Each neuron output is a function of all the activation outputs of the previous layer.

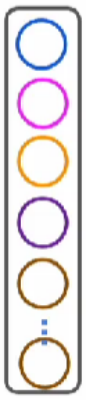
$$\vec{a}_1^{[2]} = g\left(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]}\right)$$

New Layer

→ Convolutional Layer.

Convolutional Layer





Each neuron only looks at part of the previous layer's outputs.

Why?
- Faster computation
- Need less training data (less prone to overfitting)

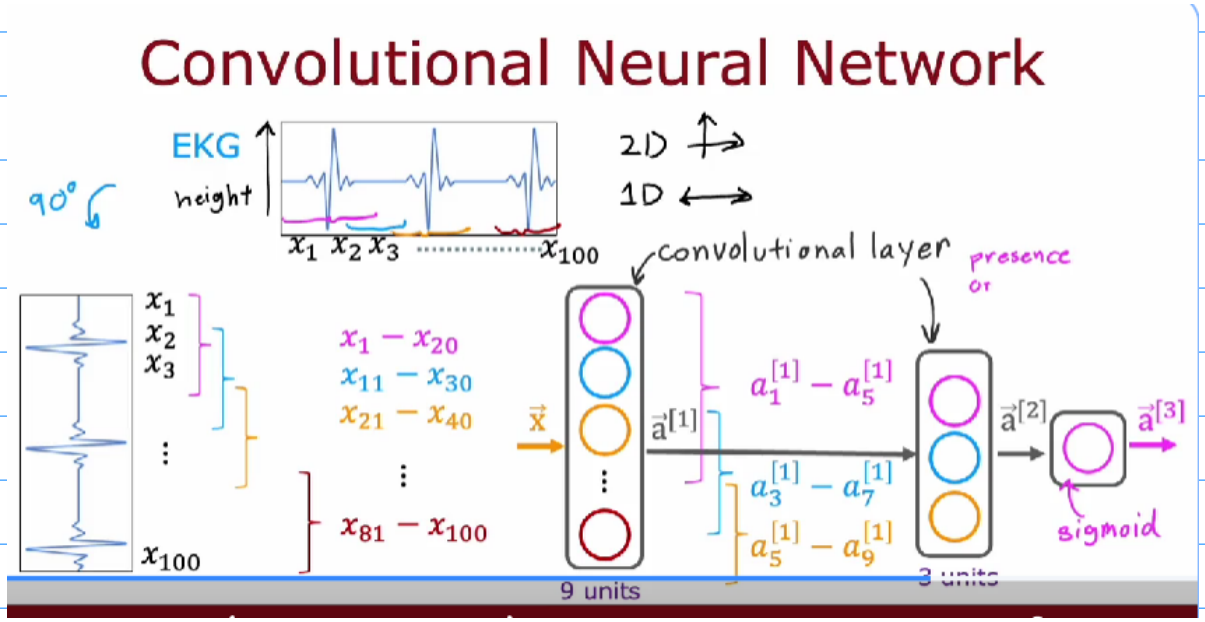
Conv Neural Network.

What's Conv Layer?

A layer that only looks at a part of the data and not the whole data

Combine Conv. layers.

You get a neural network, which is faster than Dense Neural Network, which consist of layers each that look at a part of data and then collectively look at the whole data. This is called a Convolutional Neural Network.



CNN's are good for spatial data recognition.

Backpropogation.

$J(w) = w^2$ let $w = 3$

$= 9$

if $w \uparrow$ by ε causes $J(w) \uparrow$ by $K * \varepsilon$

then $\frac{\partial}{\partial w} J(w) = K$

$\frac{\partial}{\partial w}(w^2) = 2w$

More Derivative Examples

$w = 3$ $J(w) = w^2 = 9$ $w \uparrow 0.001$ $J(w) = J(3.001) = 9.006001$ $\frac{\partial}{\partial w} J(w) = 6$

$J(w) \uparrow 6 \times 0.001$

$w = 2$ $J(w) = w^2 = 4$ $w \uparrow 0.001$ $J(w) = J(2.001) = 4.004001$ $\frac{\partial}{\partial w} J(w) = 4$

$J(w) \uparrow 4 \times 0.001$

$J(w) = w^2$

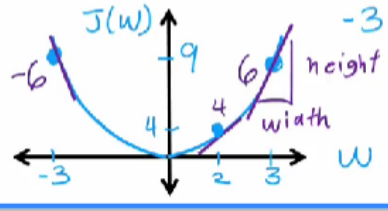
$\downarrow 0.006$

$w = -3$ $J(w) = w^2 = 9$ $w \uparrow 0.001$ $J(w) = J(-2.999) = 8.994001$ $\frac{\partial}{\partial w} J(w) = -6$

$J(w) \downarrow 6 \times 0.001$

$-3 + 0.001$

$J(w) \uparrow -6 \times 0.001$



Calculus	w	$\partial J(w)/\partial w$
$\frac{\partial}{\partial w} J(w) = 2w$	3	$2 \times 3 = 6$
	2	$2 \times 2 = 4$
	-3	$2 \times -3 = -6$

A note on derivative notation

If $J(w)$ is a function of one variable (w),

$\frac{d}{dw}J(w)$

If $J(w_1, w_2, \ldots, w_n)$ is a function of more than one variable,

$\frac{\partial}{\partial w_i}J(w_1, w_2, \ldots, w_n)$

DeepLearning.AI Stanford ONLINE Andrew Ng

Computation Graph

$\vec{x} \rightarrow \bigcirc \xrightarrow{a}$

let $w=2, b=8$

$x=-2 \quad y=2$

$a = wx + b$

$J(w,b) = \frac{1}{2}(a-y)^2$

$w \xrightarrow{2} c=wx \xrightarrow{-y} a=c+b \xrightarrow{y} d=a-y \xrightarrow{2} J=\frac{1}{2}d^2 \xrightarrow{2} J$

$8 \rightarrow$

Forward propagation.

Neural Network Example

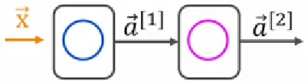
$x = 1 \quad y = 5$

$w^{[1]} = 2, b^{[1]} = 0$

$w^{[2]} = 3, b^{[2]} = 1$

ReLU activation

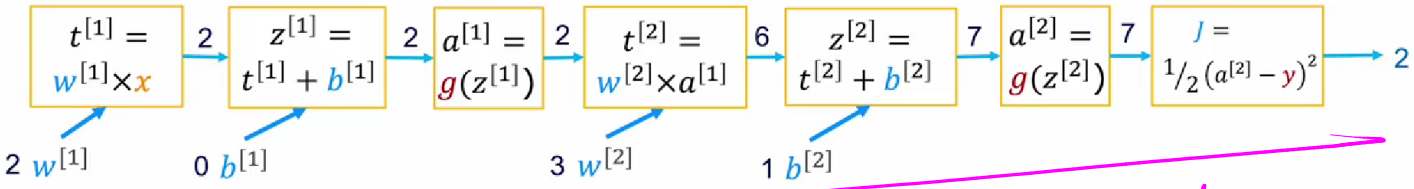
$g(z) = \max(0, z)$



$a^{[1]} = g(w^{[1]}\, x + b^{[1]}) = \overbrace{w^{[1]}\, x + b^{[1]}}^{z^{[1]}} = 2\times1 + 0 = 2$

$a^{[2]} = g(w^{[2]}\, a^{[1]} + b^{[2]}) = \overbrace{w^{[2]}\, a^{[1]} + b^{[2]}}^{z^{[2]}} = 3\times2 + 1 = 7$

$J(w, b) = \frac{1}{2}(a^{[2]} - y)^2 = \frac{1}{2}(7 - 5)^2 = 2$



Computation Graph for forward propogation.

Week 3

This week we will focus on.

Machine learning diagnostic

Diagnostic:

A test that you run to gain insight into what is/isn't working with a learning algorithm, to gain guidance into improving its performance.

Diagnostics can take time to implement
but doing so can be a very good use of your time.

m_{train} → no. of training examples
m_{test} → no. of test examples
} split dataset into Train & Test.

Train/Test procedure for Linear Regression

cost

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

pred. actual

no. of training example

regularization term.

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

Predicted o/p

True o/p

Compute test error:

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

Compute training error:

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

Train/test procedure for classification problem

0/1

Fit parameters by minimizing $J(\vec{w}, b)$ to find $\vec{w}, b$

main cost uses m_{train}

cost

E.g.,

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

loss

reg

Compute test error:

Test cost

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

Compute train error:

Train cost

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

A better way is calculate the fraction of test and train set that has been misclassified.

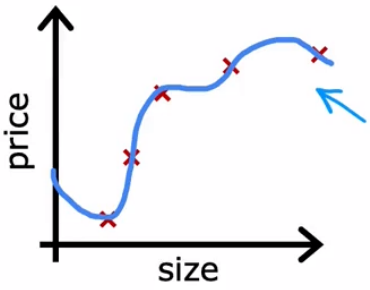
J_{test} → Fraction of test set that's misclassified.

J_{train} → Fraction of train set that's misclassified.

Evaluating & choosing models-

— Model Selection & training/cross validation /test sets

Model selection (choosing a model)



Once parameters $\vec{w}, b$ are fit to the training set, the training error $J_{train}(\vec{w}, b)$ is likely lower than the actual generalization error.

$J_{test}(\vec{w}, b)$ is better estimate of how well the model will generalize to new data compared to $J_{train}(\vec{w}, b)$.

$x = x_1$

$f_{\vec{w},b}(\vec{x}) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

Train Set v/s Validation Set v/s Test Set

"Final judgement ke liye use karo" ka matlab:

- **Training set:** Ispe tum model ko train karte ho (weights w, b learn karte ho).
- **Validation set:** Ispe tum **hyperparameters** (jaise polynomial degree d, learning rate, regularization strength, etc.) choose karte ho. Matlab model **kis shape/complexity ka hoga** yeh decide karte ho.
- **Test set:** Yeh ekdum alag rakha jata hai. Ispe **kuch bhi tune nahi karna**.
- Jab tum final model select kar chuke (hyperparameters fix ho gaye), tab **test set ek baar run karke** real-world performance estimate karte ho.

Matlab:

- Test set ek **final exam** hai.
- Agar tumne test set ka use **practice karne** (hyperparameter choose karne) ke liye bhi kar diya, toh exam ka answer tumhe pehle se pata hoga → aur tumhe lagega ki tum topper ho, but real life (new data) me utna score nahi aayega.

We will split our data into 3 parts-

Training/cross validation/test set

size	price
2104	400
1600	330
2400	369
1416	232
3000	540
1985	300
1534	315
1427	199
1380	212
1494	243

training set 60% → $(x^{(1)}, y^{(1)}) \dots (x^{(m_{train})}, y^{(m_{train})})$ → $m_{train} = 6$ } model train

cross validation 20% → $(x_{cv}^{(1)}, y_{cv}^{(1)}) \dots (x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$ → $m_{cv} = 2$ } hyper params

test set 20% → $(x_{test}^{(1)}, y_{test}^{(1)}) \dots (x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$ → $m_{test} = 2$ } Final exam

DeepLearning.AI Stanford ONLINE Andrew Ng

We'll see you on the next slide how to

Training/cross validation/test set

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

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

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

Procedure

Train the model on J_{train}

↓

Model Selection → validation set J_{cv}

↓

Est. generalization error using test set J_{test}

Final Exam

When to use what dataset.

Stage-wise use of datasets

1. Stage 1 → Model ko train karna (weights seekhna)

- **Dataset: Training set**
- **Purpose:** Model ke parameters (weights, biases) ko optimize karna.
- **Output:** Ek trained model (given architecture & hyperparameters ke saath).
- **Kya nahi karte:** architecture decide nahi karte is stage pe — sirf ek architecture ke liye train karte ho.

---

2. Stage 2 → Model design choose karna (architecture & hyperparameters tune karna)

- **Dataset: Validation (or Cross-validation) set**
- **Purpose:**
  - Compare karo: kaunsa **architecture** (e.g., layers/neurons) better perform karta hai?
  - Kaunsa **hyperparameter** (learning rate, regularization) best hai?
- **Why validation and not training?**
  - Training error sirf yeh batata hai ki model ne training data kitna yaad kiya.
  - Validation error batata hai ki unseen data pe model generalize kar raha ya nahi.
- **Modern ML (industry):** Architecture bhi mostly validation pe decide karte hain.

---

3. Stage 3 → Final unbiased performance check

- **Dataset: Test set**
- **Purpose:** Jab architecture + hyperparameters fix ho gaye, tab **ek baar** use karna to check **true unseen performance.**
- **Why single use?** Kyunki agar repeatedly test pe tune karte ho, woh bhi training jaise ho jayega → biased result.

Quick diagram idea

kotlin Copy Edit

```
Raw data
   |
Split -----> Training set (weights seekhna)
   |
   ---> Validation set (architecture & hyperparameter tuning)
   |
   ---> Test set (final unbiased check)
```

Stage Mapping in Code

1. **Stage 1 (Training weights):**

```python
model.fit(x_train, y_train)
```

→ Weights learn karte hain.

2. **Stage 2 (Architecture choice):**

```python
val_loss, val_acc = model.evaluate(x_val, y_val)
```

→ Kis architecture ka generalization best hai, yeh validation set se pata chalta hai.

3. **Stage 3 (Final unbiased check):**

```python
test_loss, test_acc = best_model.evaluate(x_test, y_test)
```

→ Sirf ek baar final check ke liye.

Bias and Variance.

Underfit Model

- It has high bias and is inaccurate
- J_{train} & J_{cv} is high.

If a model has significantly high bias & high variance, its the characteristic of an underfit model.

→ **Overfit Model**

- It has high variance and is highly accurate on training data but performs poorly on CV data & test data.
- J_{train} is low but J_{cv} is significantly high in comparison to J_{train}.

↳ This is the characteristic of an overfit model.

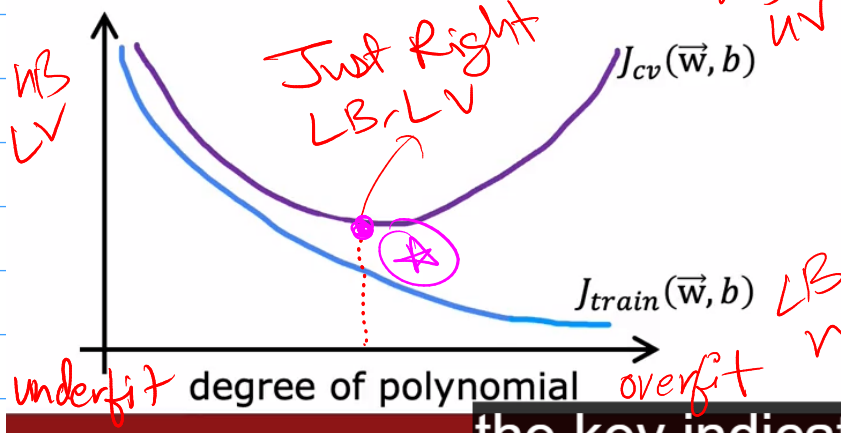
→ **Just right**

Both J_{train} & J_{cv} is low. Meaning the model is well fit and performs well.

J_{train} ↓ , J_{cv} ↓ → Characteristic of good model.

Diagnosing bias and variance

How do you tell if your algorithm has a bias or variance problem?



High bias (underfit)

→ J_{train} will be high ($J_{train} \approx J_{cv}$)

High variance (overfit)

$J_{cv} \gg J_{train}$

(J_{train} may be low)

High bias and high variance

J_{train} will be high

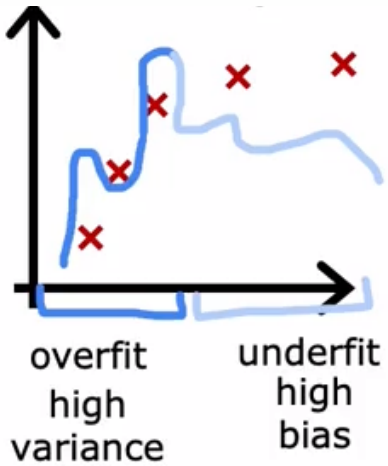
and $J_{cv} \gg J_{train}$

DeepLearning.AI Stanford ONLINE Andrew Ng

the key indicator will be if J train is high.

In what case do we get HB & HV?

When the model fits a part of the training data very well (overfits) and doesn't fit the other half of the training set (underfit) in such cases we get HB, HV getting HB, HV meaning your model is quite bad.

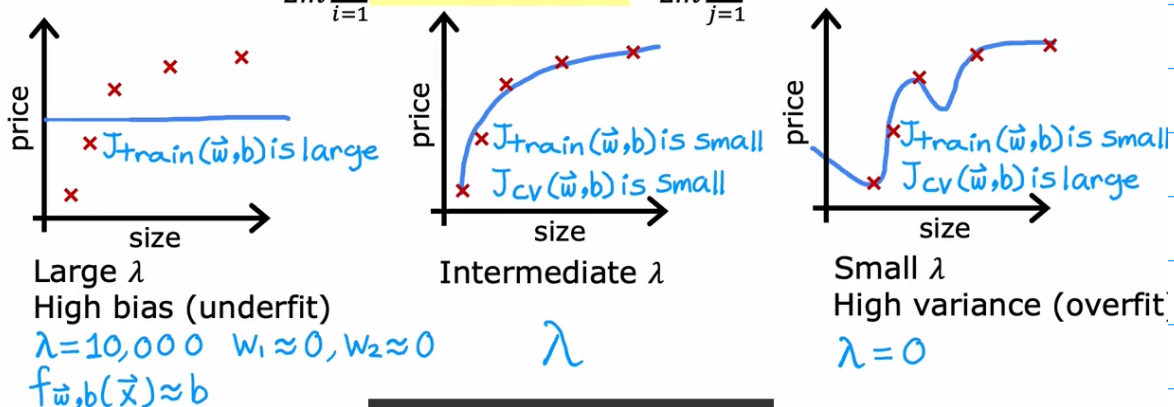


Regularization & Bias / Variance.

Linear regression with regularization

Model: $f_{\vec{w},b}(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$

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



Large λ
High bias (underfit)
$\lambda = 10,000$ $w_1 \approx 0, w_2 \approx 0$
$f_{\vec{w},b}(\vec{x}) \approx b$

Intermediate λ

Small λ
High variance (overfit)
$\lambda = 0$

DeepLearning.AI Stanford ONLINE Andrew Ng

Let's take a look at how we could do so.

How to choose the correct value of λ?

Let's say model is as follows :-

$$f(x) = w_1x + w_2x^2 + w_3x^3 + w_4x^4 + b$$

Try the following values of λ :-

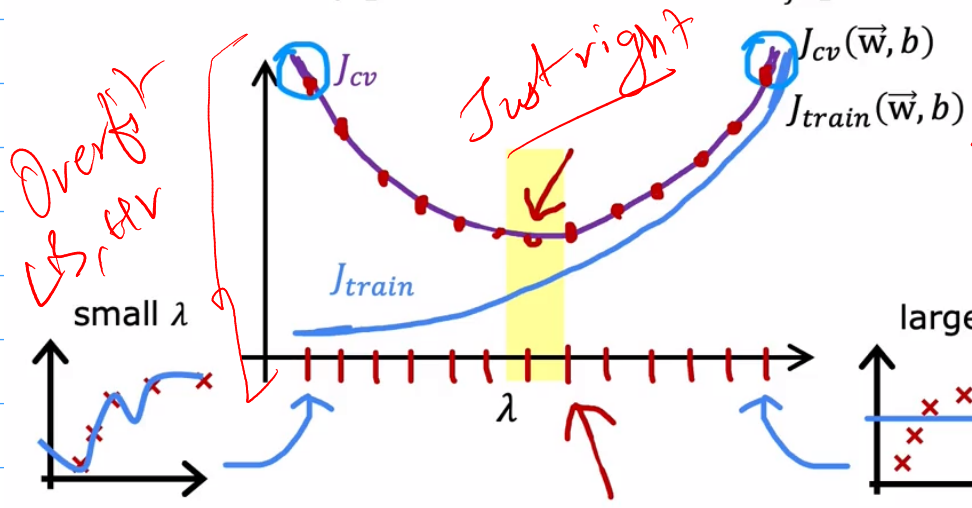
0, 0.01, 0.02, 0.04, 0.08, - - - - 10

Calculate the J_{cv} for each λ to figure out the suitable λ for your model.

pick λ with low $J_{cv}(w, b)$

Bias and variance as a function of regularization parameter λ

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



Overfit
Just right
High Bias
High Variance
Underfit

Establishing a Baseline level of Performance

By setting a Baseline we can judge if the learning algorithm is under or over by looking at J_{train} & J_{cv}

Speech recognition example

Human level performance : 10.6%
Training error J_{train} : 10.8%
Cross validation error J_{cv} : 14.8%

Human level performance is 10.6% & machine is 10.8%. This means quite accurate since if human is at 10.6% then you can't expect machine to be less or than human error right.
In this example it is just 2% more than human performance

higher than a human level of performance,

Tldr when we see J_{train} error we have to benchmark it against human level performance to decide if model is HB or LB

In above eg $J_{train} = 10.4$ & $J_{cv} = 14.8\%$

← 4% difference →

This 4% difference is quite high & this is inclined toward a variance problem.

Establishing a baseline level of performance

What is the level of error you can reasonably hope to get to?

- Human level performance
- Competing algorithms performance
- Guess based on experience

Bias/variance examples

Baseline performance : 10.6% ↕ 0.2% Low Bias
Training error (J_{train}) : 10.8% ↕ 4.0% High variance
Cross validation error (J_{cv}): 14.8%

high variance

Hence this is an overfitted model

Lets say:

Baseline Performance : 10.6%

J_{train} : 15.0%

J_{CV} : 15.5%

4.4% — High Bias

0.5% — Low Variance

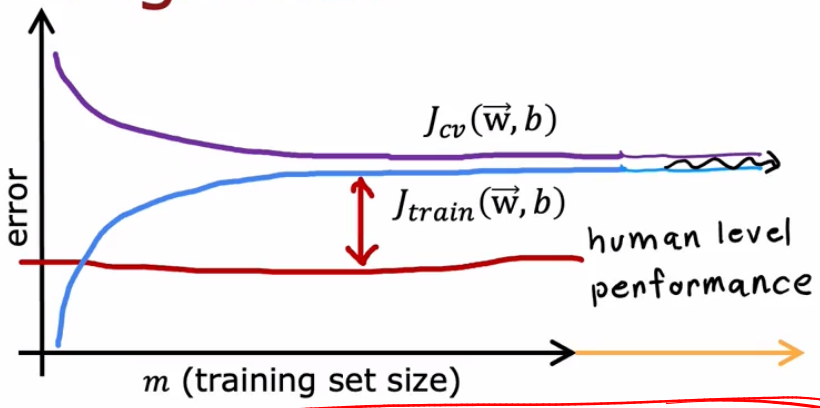
Hence this is an underfitted model.

> ⭐ Always compare to baseline performance

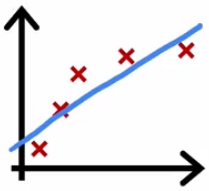
Learning Curve

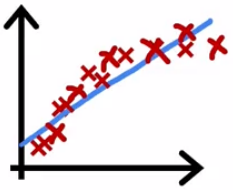
Learning Curve tells us about our learning algo as of what experience it has with the training examples.

High bias



$f_{\vec{w},b}(x) = w_1 x + b$

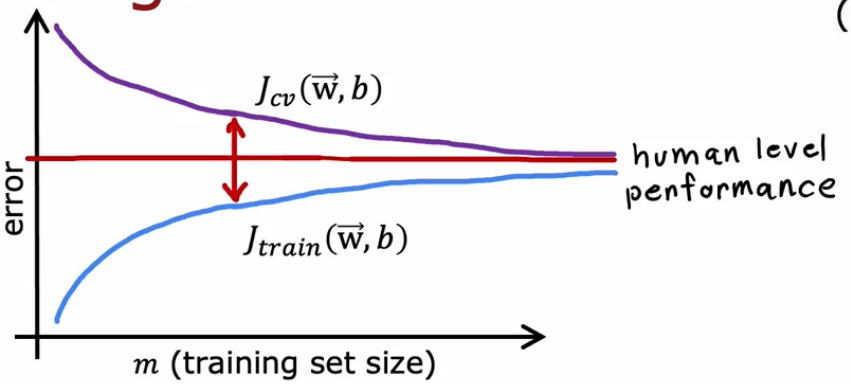


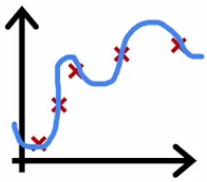


if a learning algorithm suffers from high bias, getting more training data will not (by itself) help much.

High variance

$f_{\vec{w},b}(x) = w_1 x + w_2 x^2 + w_3 x^3 + w_4 x^4 + b$
(with small λ)





If a learning algorithm suffers from high variance, getting more training data is likely to help.

then getting more training data is indeed likely to help.

Debugging a learning algorithm

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

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

But it makes unacceptably large errors in predictions. What do you try next?

→ Get more training examples	fixes high variance
→ Try smaller sets of features $x, x^2, x^3, x^4, x^5 \ldots$	fixes high variance
→ Try getting additional features	fixes high bias
→ Try adding polynomial features $(x_1^2, x_2^2, x_1x_2, etc)$	fixes high bias
→ Try decreasing λ	fixes high bias
→ Try increasing λ	fixes high variance

① If you find your algo has high variance then primarily you should get more training examples or dial down the complexity of the model. (By using lesser features or increase λ)

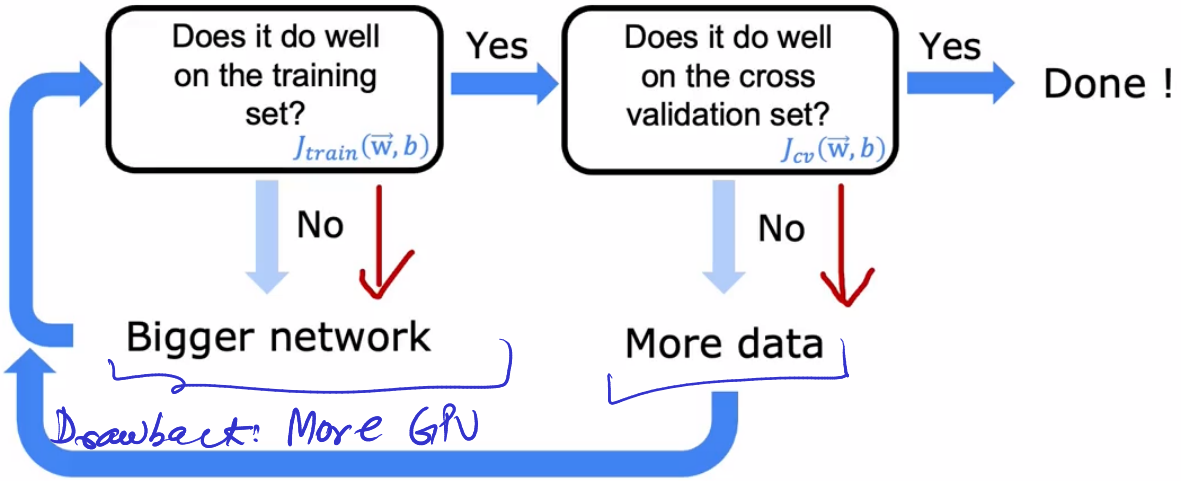
② If you find that your algorithm isn't even doing good on the training set, its HB, the primarily you should dial up the complexity (add more features or decrease λ).

Bias/Variance & Neural Networks

Large Neural Networks trained on medium to large datasets are low bias machines.

Neural networks and bias variance

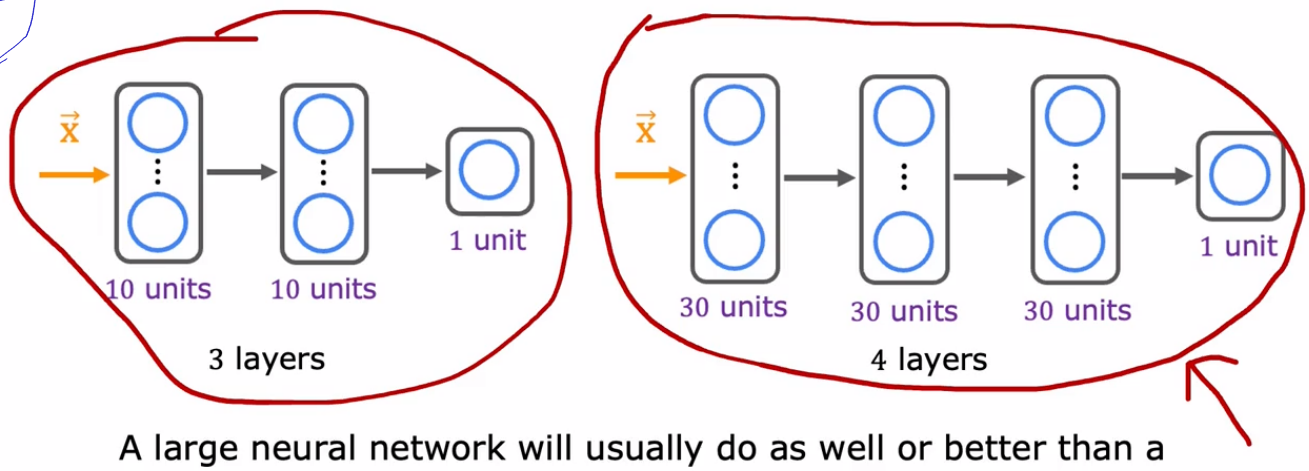
Large neural networks are low bias machines



validation set and hopefully will also generalize to new examples as well.

DeepLearning.AI Stanford ONLINE Andrew Ng

Neural networks and regularization



A large neural network will usually do as well or better than a smaller one so long as regularization is chosen appropriately.

So long as the regularization has chosen appropriately.

DeepLearning.AI Stanford ONLINE Andrew Ng

Tldr: Choose a large NN with an appropiate Regularization term.

Lasso (L1) v/s Ridge (L2) v/s Elastic Net ($L_1 + L_2$)

1) Use Lasso (L1)

When:

- **High-dimensional data** (features >> samples) → e.g. genomics, text, one-hot categorical encoding
- **Want feature selection automatically** → kuch features irrelevant hai, unko model khud **0** kar dega
- **Interpretability important hai** → chhota sparse model chahiye jo clearly bole "ye features important hain"
- **Storage ya latency constraint** → sparse weights = lightweight model

Tradeoff:

- Jab correlated features hain, Lasso randomly ek rakhega aur dusre ko zero karega (unstable selection).

---

2) Use Ridge (L2)

When:

- **Almost all features relevant hain** → aur unka magnitude control karna hai, par drop nahi karna
- **Multicollinearity issue hai** (features correlated hain) → Ridge sabko thoda shrink karega instead of dropping
- **Deep Learning & Neural Nets** → Ridge smoothness deta hai, optimization easier hai, isliye ye **default** hai
- **Numeric stability** chahiye (overfitting ka risk high hai, but interpretability important nahi hai)

Tradeoff:

- Sparse nahi banata → har feature ka contribution rahega.

---

3) Elastic Net (L1 + L2 mix)

When:

- Feature selection bhi chahiye **AND** correlated features ko saath handle karna hai
- Kaggle ke competitions ya applied ML me ye common hai because:
  - L1 → sparse bana deta hai
  - L2 → correlated features me stability deta hai

Summary Shortcut

- **Feature selection important? → Lasso (L1)**
- **Sab feature rakho but control karo? → Ridge (L2)**
- **Both chahiye → Elastic Net**

Okay, real-life relatable examples lete hain so tu yaad rakh sake:

---

1) Lasso (L1) – Push to 0

Real-life scenarios:

1. **Finance (Credit Scoring)**
   - Tumhare paas 200 variables hain (income, spending habits, location, 100 dummy features, etc.).
   - Tum chahte ho ki **kaunse features loan default predict karne me sabse zyada important hai.**
   - **Lasso** automatically irrelevant features (e.g., "favourite Netflix genre") ko **0** kar dega.
   - Output: sirf top 10–15 impactful variables bache → explain karna easy.
2. **Healthcare (Disease Prediction)**
   - Genomic data me 20,000+ gene expression features.
   - Tum chahte ho ki **kuch hi genes select ho jo cancer risk ke liye critical hain.**
   - Lasso → sparse output deta hai → interpret karna easy.
3. **IoT / Sensor Data**
   - 300 sensors ka data leke machine failure predict karna.
   - Lasso automatically useless sensors ignore karega → cheaper sensor system ban sakta hai.

---

2) Ridge (L2) – Near to 0

Real-life scenarios:

1. **Image Classification (Deep Learning)**
   - CNN me 1 million parameters.
   - Tum chahte ho ki **sab features contribute kare, par overfitting na ho.**
   - Ridge (L2) = weights ko shrink karega par kisi ko zero nahi karega → deep learning me default hai.
2. **Marketing Sales Prediction**
   - Tumhare paas 100 related marketing channels (TV, FB, Insta, etc.) ke spends ka data hai.
   - Sare correlated hain, tum kisi ko drop nahi kar sakte (sab important hai).
   - Ridge use karo → sabka thoda impact rakhta hai, par magnitude control kar ke stable predictions deta hai.
3. **Weather Forecasting**
   - Features like temperature, humidity, pressure, wind, etc. – sab naturally important hain.
   - Ridge ensures koi bhi feature hataya na jaye, bas unka effect smooth ho.

---

3) Elastic Net (L1 + L2)

Real-life scenarios:

1. **E-commerce Recommendations**
   - Bahut saare user features aur product features (kuch irrelevant bhi).
   - Elastic Net → irrelevant features drop karega (L1) **and** correlated features ko saath handle karega (L2).
2. **Stock Price Prediction (High Dimensional)**
   - 500+ technical indicators, mostly correlated.
   - Elastic Net feature selection + stability dono deta hai → robust model.

Quick Mental Model

- **Lasso** = jab tumhe "kaunse factors important hain" jaan'na hai (interpretability).
- **Ridge** = jab tumhe **sab features use karna hai** but stable prediction chahiye.
- **Elastic Net** = jab tumhe dono ka mix chahiye (real-world tabular data me common).

Regularized MNIST Model.

Neural network regularization

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

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

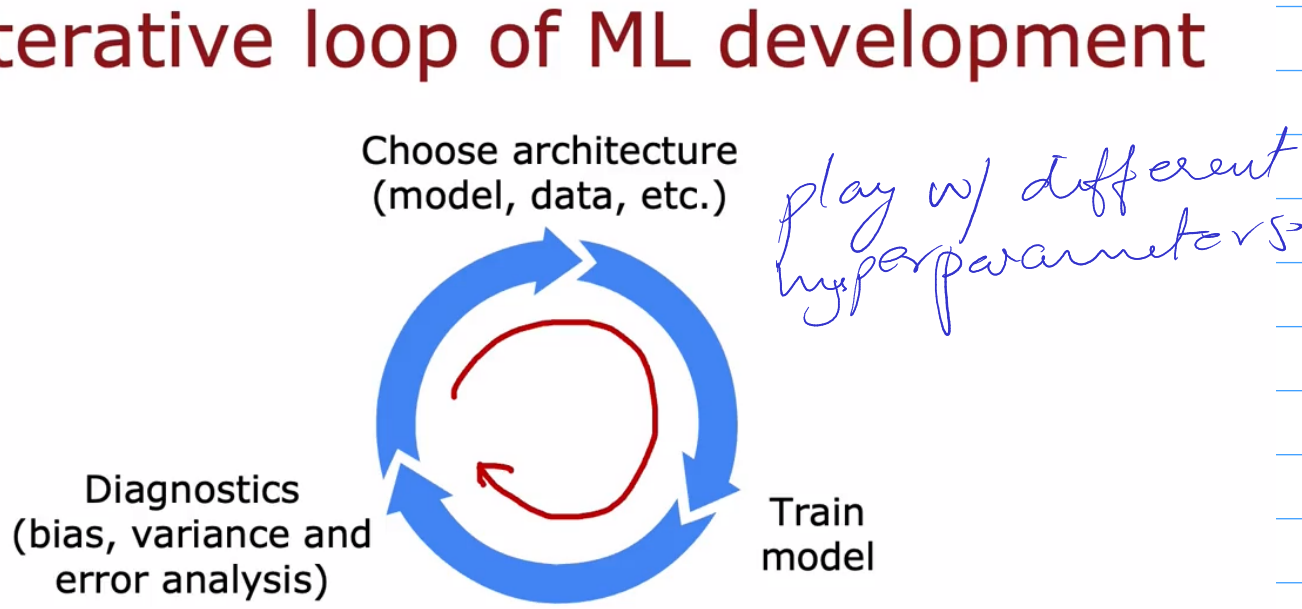
```
layer_1 = Dense(units=25, activation="relu", kernel_regularizer=L2(0.01))
layer_2 = Dense(units=15, activation="relu", kernel_regularizer=L2(0.01))
layer_3 = Dense(units=1, activation="sigmoid", kernel_regularizer=L2(0.01))
model = Sequential([layer_1, layer_2, layer_3])
```

DeepLearning.AI Stanford ONLINE Andrew Ng

Then the regularization term for a neural network looks like pretty much what

Machine Learning development process -

Iterative loop of ML Development -



play w/ different hyperparameters

In this lesson we will learn about error analysis which has a second key set of ideas for gaining insights about what architecture choices might be fruitful.

Error handling is one of the important ways to run error diagnostics -

Error Handling.

Error handling is the process of looking at your model's mistakes, figuring out what happened and using that information to improve the model.

Why Error Analysis?

- To **understand where your ML model is making mistakes.**
- Helps **decide what to improve** (data, model, features, etc.).
- **Data-driven approach** → avoid random guessing.

Steps in Error Analysis

1. **Collect Misclassified Examples**
   - Take a **random sample of errors** (e.g., 100–1000 misclassified points).
   - Look for patterns.
2. **Manually Examine Errors**
   - Check **what kind of errors are common.**
   - Categorize them (e.g., blurry image, wrong label, unusual angle, etc.).
3. **Quantify Error Types**
   - Example:
     - 43% = bad lighting
     - 29% = incorrect labels
     - 15% = unusual objects
   - This tells where to focus.
4. **Decide Priorities**
   - If 50% errors are due to mislabeled data → fix labeling.
   - If 40% due to unusual backgrounds → collect more such data.

Important Tips

- **Look at actual errors, not overall accuracy.**
- **Use % of error categories to decide next steps** (data cleaning, data augmentation, model architecture).
- **Revisit error analysis regularly** when model improves.

Outcome?

→ Clear actionable insights.

→ Faster model improvements with less random trial & error.

Error Analysis in a nutshell?

Look at where your NN model is going wrong, find patterns in mistakes, and use that info to decide what to fix first.

Collect some of the wrong predictions

↓

Manually check why they went wrong

↓

Group and count error types

↓

Prioritize fixes based on which error type happens the most.

Error Analysis = MANUALLY checking your model's wrong predictions.

Adding data

Add more data of everything. E.g., "Honeypot" project.

Add more data of the types where error analysis has indicated it might help.

Pharma spam

E.g., Go to unlabeled data and find more examples of Pharma related spam.

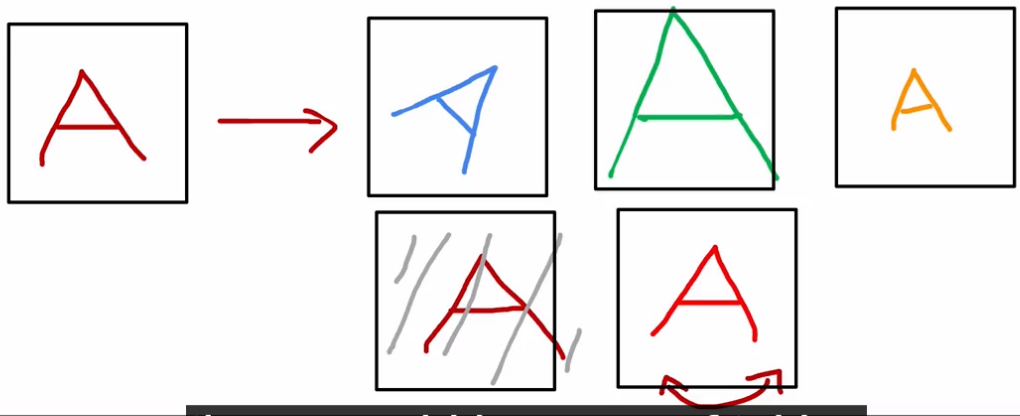
Beyond getting brand new training examples (x,y), another technique: Data augmentation

DeepLearning.AI Stanford Online Andrew Ng

audio data that can increase your training set size significantly.

Data augmentation

Augmentation: modifying an existing training example to create a new training example.



these would be ways of taking a training example X, Y.

DeepLearning.AI Andrew Ng

Data augmentation for speech

Speech recognition example

- Original audio (voice search: "What is today's weather?")
- + Noisy background: Crowd
- + Noisy background: Car
- + Audio on bad cellphone connection

examples here, one with crowd background noise, one with car background noise and

DeepLearning.AI Andrew Ng

Transfer Learning

Transfer Learning is when you take a model that has already learnt something useful in one task and then reuse it to help with another task — usually when you dont have much data for your own task.

Nutshell Definition:

> **Transfer Learning** is the technique of using a **pre-trained model** (on a large dataset) as a starting point for a **new but related task**, especially when you have **limited data**.

Apne kaam ke liye kam data hai ?

↓ Solution

Dusre task pe trained model utha aur apne kaam ke liye tweak kar.

Core Idea:

1. **Train a model on a BIG dataset** (like 1 million images of cats, dogs, cars, etc.)
2. Use the **learned layers** (especially earlier ones) as a **starting point** for a **new task** (like recognizing handwritten digits 0-9).
3. Just change the **final output layer** and **fine-tune** the rest — that's transfer learning.

✅ Solution: TRANSFER LEARNING

Step 1️⃣: Use a Pre-trained Model

Someone trained a deep neural net on **1 million images** with **1000 classes** (like cats, cars, etc.).

That model learned:

- Layer 1 → Detect edges
- Layer 2 → Corners
- Layer 3 → Shapes
- Layer 4 → Combinations of shapes
- Layer 5 → Final classification to 1000 classes

These **early layers** learn **generic features** that are useful for *ANY* image task (even digit recognition).

Step 2️⃣: Copy Layers (Except Final One)

You copy:

- W^1, b^1
- W^2, b^2
- W^3, b^3
- W^4, b^4

But throw away the final layer (W^5, b^5) and replace it with a **new output layer** of **10 units** for digits 0–9.

🛠️ Two Fine-Tuning Options:

🔹 Option 1: Freeze base, train only final layer

- Early layers (W^1 to W^4) → **Keep fixed**
- Train only new layer (W^5, b^5)
- Good when **you have very little data**
  (Prevents overfitting, since fewer paran ↓ rs are trained)

🧠 But Why Does This Even Work?

> Bro, cats and digits are totally different! 🤔

Valid point. But...

- The **early layers** of CNNs don't care about the *specific object*
- They just learn to detect **edges, textures, patterns**
- These low-level features are **common** across many visual tasks
- So, when you re-use these layers, you're starting from a **smart place** instead of random weights

---

🔍 Terminology:

Term	Meaning
Pre-training	Training on big dataset (like ImageNet)
Fine-tuning	Adjusting the model on your small dataset
Transfer Learning	The whole process: reusing & tuning

okay sure
so lets imagine you are building an image classification algo
but ohhhh nooooooooooo, you dont have much training examples

here comes TL to the rescueeeeeeeeeeeeeee🥰🥰🥰🥰

you: ehhh, TL whats that, and how would that help meeee???

well TL is a technique where you can use a pre-trained model,
a model who is trained on extensive amount of similar kind of data and hence has its
foundation layers solid for lets say image classifcation in your case.

you: but how would the model know about my images???

well you directly dont copy pste the algo and model. you take its foundational
or early layers, the foundational layers of the algo all learn the similar things
or propeties of a general image like edges, corners, texture, basic shapes and its the
later layers that hand pick the objects. so you can use the foundation layers of a
pre trained model and add layers of your model that fine tune it to you dataset and tweak
the foundational layers if you want to further double down and improve its performance

you: ohhhhhh wowwwww rlyyyyyyyyyy???🤯🤯🤯

yessssirrr, just take the heavy work already done by a pre trained model
and attch your work to it and tweak, fine-tune and polish it according to you needs
and boom you have achived what you wanted without you doing much heavy lifting

NOTE: The input types must be same for both

the networks for transfer learning to work

So:

- If your model is pre-trained on **images**, it expects **images** again
- If it's pre-trained on **text**, it only works for **text-based tasks**
- You **can't** use an image-trained model for audio or vice versa (unless you want your model to hallucinate 😵)

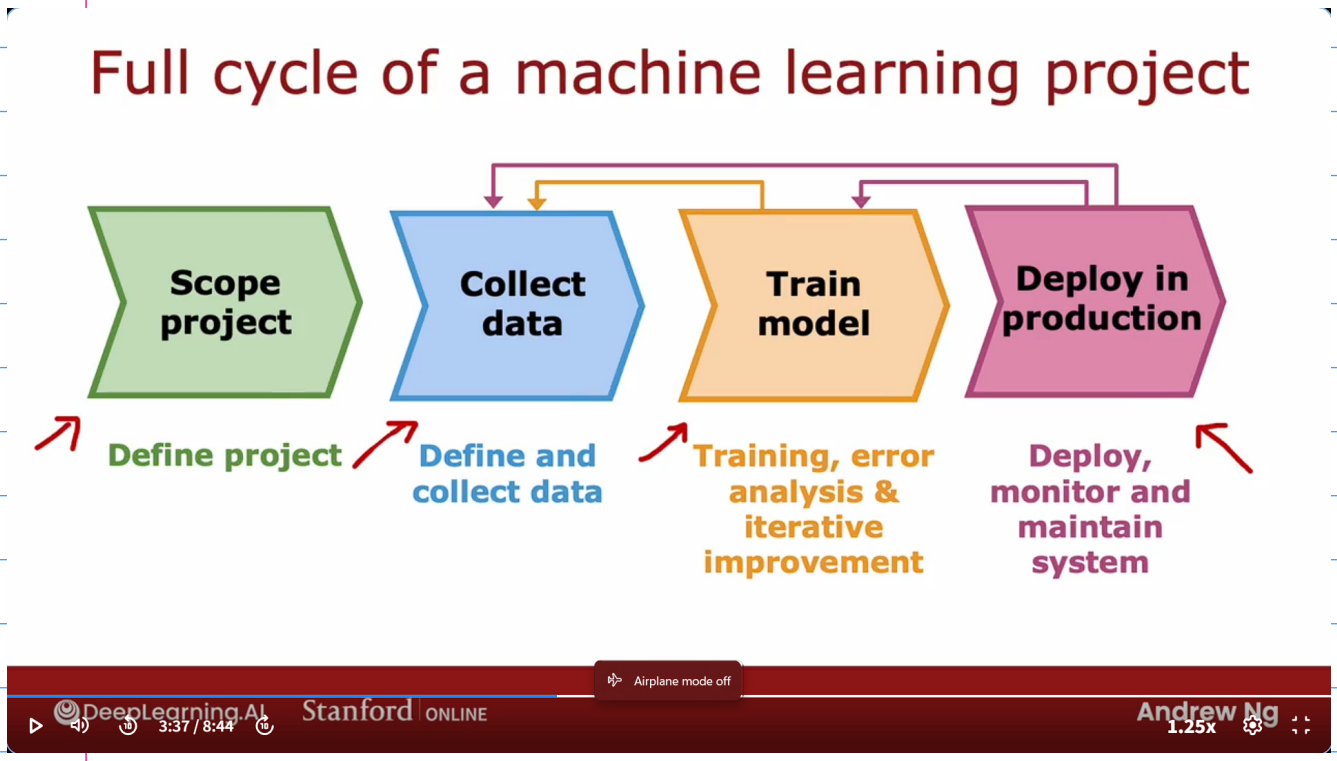
Transfer learning summary

1. Download neural network parameters pretrained on a large dataset with same input type (e.g., images, audio, text) as your application (or train your own). 1 million images
2. Further train (fine tune) the network on your own data.

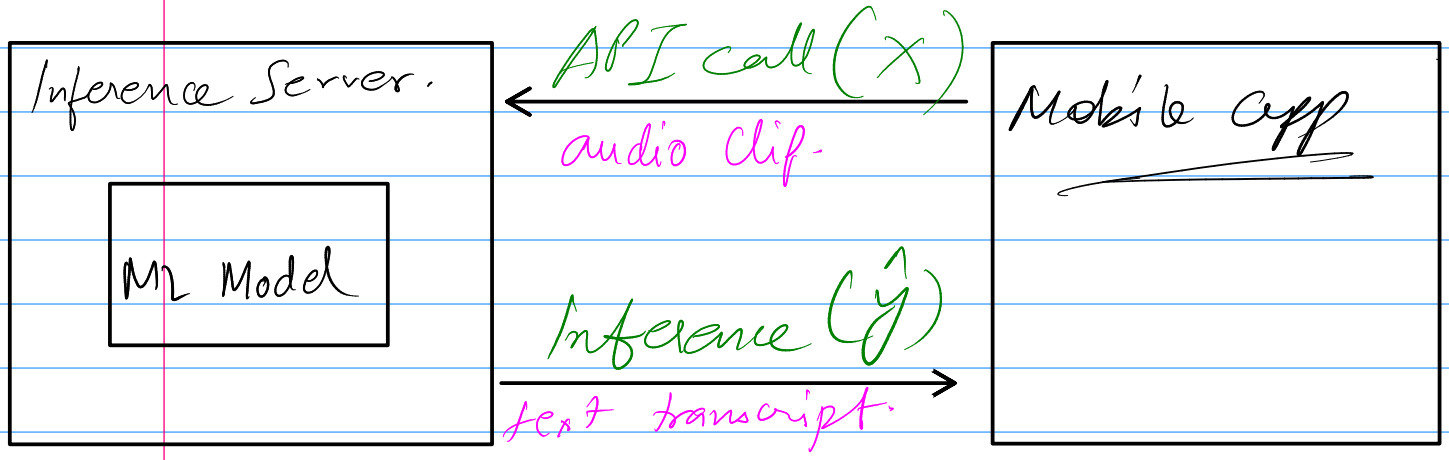
1000 images

50 images

Full Cycle of a Machine Learning project



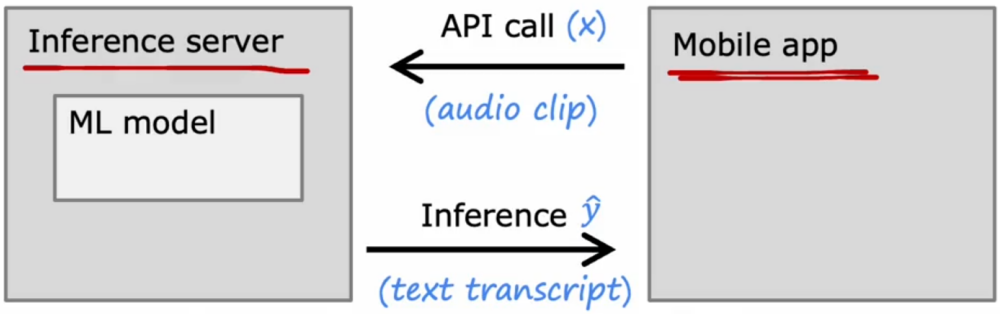
Deployment



Tumhara model at inference server may hoga. Mobile app via api tumhare server se communicate karega and model ko invoke karega with the input you gave.

model evaluate karega and apna prediction bhejega via the server to your mobile application as an output

Deployment



→ Software engineering may be needed for:
- Ensure reliable and efficient predictions
- Scaling
- Logging
- System monitoring
- Model updates

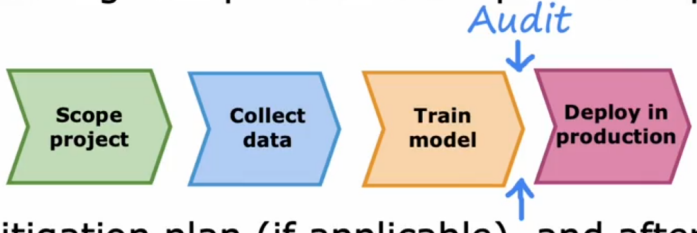
MLOps
machine learning operations

Guidelines

Get a diverse team to brainstorm things that might go wrong, with emphasis on possible harm to vulnerable groups.

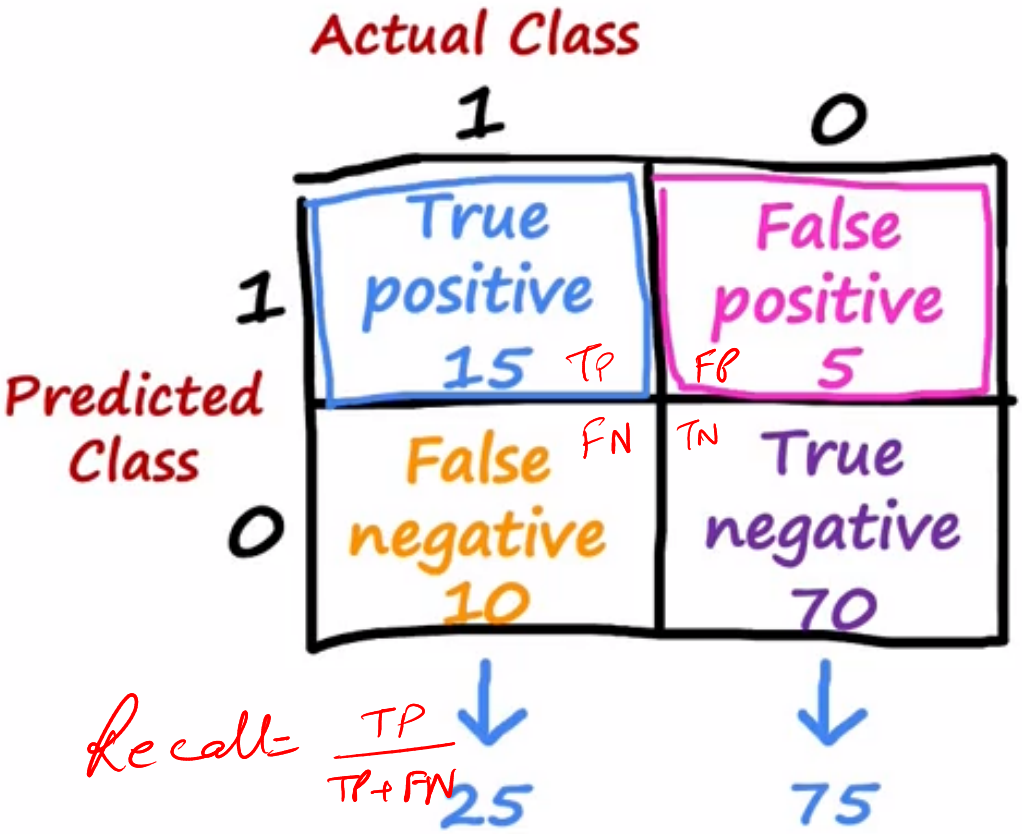
Carry out literature search on standards/guidelines for your industry.

Audit systems against possible harm prior to deployment.



Develop mitigation plan (if applicable), and after deployment, monitor for possible harm.

Confusion Matrix



Precision = $\frac{TP}{TP+FP}$

Recall = $\frac{TP}{TP+FN}$

Precision:
(of all patients where we predicted $y = 1$, what fraction actually have the rare disease?)

$$\frac{\text{True positives}}{\#\text{predicted positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False pos}} = \frac{15}{15+5} = 0.75$$

Recall:
(of all patients that actually have the rare disease, what fraction did we correctly detect as having it?)

$$\frac{\text{True positives}}{\#\text{actual positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False neg}}$$

1️⃣ Precision — *model ka "clean shooter" hone ka scale* 🎯

- Tells you: **When the model says "Yes/Positive", can you trust it?**
- High Precision → Model doesn't raise false alarms often.
- Low Precision → Model says "Positive" too often, even when it's wrong.

Example:

Spam detection

- Model predicts an email is spam.
- **Precision** = Of all emails marked as spam, how many were actually spam?
- High precision → Hardly any important mail got sent to spam.
- Low precision → Important mails also land in spam → user angry.

2️⃣ Recall — *model ka "catch them all" ability* 🔍

- Tells you: **Does the model miss real positives?**
- High Recall → Catches almost every positive case.
- Low Recall → Misses a lot of positives.

Example:

Cancer detection

- Model predicts whether someone has cancer.
- **Recall** = Of all patients who actually have cancer, how many did the model correctly flag?
- High recall → Almost every sick person is detected.
- Low recall → Some sick people go undetected → dangerous.

3️⃣ What they say about the learning algorithm

- **High Precision, Low Recall** → Model is *very picky*. It only says "Positive" when it's super sure → fewer false alarms, but misses many real positives. (*Like a strict teacher who rarely gives full marks.*)
- **Low Precision, High Recall** → Model is *very generous*. Flags a lot of positives → catches most true positives but also many false alarms. (*Like a CCTV motion sensor that triggers even if a leaf moves.*)
- **High Precision, High Recall** → Ideal → Model is both accurate when it predicts and catches almost everything.
- **Low Precision, Low Recall** → Trash → Can't predict correctly and misses a lot.

Precision v/s Recall Tradeoff

Alright, boss — simple tareeke se samajh le precision–recall tradeoff ka pura scene:

---

1 Tradeoff ka funda (simple)

- **Precision ↑, Recall ↓** → Model becomes **picky** → predicts "Positive" only when super sure → fewer false alarms, but more actual positives get missed.
- **Recall ↑, Precision ↓** → Model becomes **generous** → predicts "Positive" more often → catches more real positives, but also creates more false alarms.

Matlab:

- Agar tu **tight filter** lagata hai → kam galat hits, but miss zyada.
- Agar tu **loose filter** lagata hai → zyada catch, but galat bhi zyada.

---

2 Real-life me ka scene

In practice, tu **business needs** ke hisaab se decide karta hai balance:

- **Critical safety / health** → Recall ko priority dete hain (e.g., cancer detection, security breaches)
  → Thoda galat alarm chale toh chalega, bas koi important case miss na ho.
- **User experience / trust** → Precision ko priority dete hain (e.g., spam filter, face unlock)
  → Galat alarm bohot irritating hota hai, so model should be sure before saying "Positive."

---

3 Example table to visualize tradeoff

Precision	Recall	Model Behavior	Risk
High	Low	Picky sniper 🎯	Misses many true positives
Low	High	Over-enthusiastic guard 🚨	Too many false alarms
High	High	Perfect utopia 😇	Rare in real life
Low	Low	Trash model 🗑️	Useless

Case A — Balanced (reasonable tradeoff)
Predictions: model finds a lot of true positives and some false positives.

Confusion matrix:

Actual \ Predicted	Positive	Negative	Total (actual)
Positive	TP = 80	FN = 20	100
Negative	FP = 40	TN = 860	900
Total (pred)	120	880	1000

Precision = TP / (TP + FP) = 80 / (80 + 40)

- Step 1: 80 + 40 = **120**
- Step 2: 80 / 120 = reduce by 40 → 2 / 3
- Long division: 2 ÷ 3 = 0.666666... → round = **0.6667**
- As percent: **66.67%**

Recall = TP / (TP + FN) = 80 / (80 + 20)

- Step 1: 80 + 20 = **100**
- Step 2: 80 / 100 = **0.80 → 80.00%**

F1 = 2 * (precision * recall) / (precision + recall)
Use fractions: precision = 2/3, recall = 4/5.

- Multiply: (2/3) * (4/5) = 8/15
- Numerator: 2 * (8/15) = 16/15
- Denominator: (2/3 + 4/5) = (10/15 + 12/15) = 22/15
- Division: (16/15) ÷ (22/15) = 16/22 = 8/11
- Decimal: 8 ÷ 11 = 0.7272727... → **0.7273 → 72.73%**

Takeaway (A): decent recall (80%) and okay precision (66.7%). Good if you want a balanced model.

Maximize F1 score if you want a balanced model.

Case B — Picky (high precision, low recall)
Model only predicts "Positive" when very sure.

Confusion matrix:

Actual \ Predicted	Positive	Negative	Total (actual)
Positive	TP = 40	FN = 60	100
Negative	FP = 5	TN = 895	900
Total (pred)	45	955	1000

Precision = 40 / (40 + 5)

- Step 1: 40 + 5 = **45**
- Step 2: 40 / 45 = reduce by 5 → 8 / 9
- Long division: 8 ÷ 9 = 0.888888... → round = **0.8889 → 88.89%**

Recall = 40 / (40 + 60)

- Step 1: 40 + 60 = **100**
- Step 2: 40 / 100 = **0.40 → 40.00%**

F1 using fractions: precision = 8/9, recall = 2/5.

- Multiply: (8/9) * (2/5) = 16/45
- Numerator: 2 * 16/45 = 32/45
- Denominator: 8/9 + 2/5 = (40/45 + 18/45) = 58/45
- Division: (32/45) ÷ (58/45) = 32/58 = reduce → 16/29
- Decimal: 16 ÷ 29 = 0.55172413... → **55.17%**

Takeaway (B): Very reliable positive predictions (88.9%), but misses a lot of real positives (only 40% recall).
Good when *false positives are costly*.

Saved memory full

Case C — Generous (high recall, low precision)

Model flags almost everything as suspicious.

Confusion matrix:

Actual \ Predicted	Positive	Negative	Total (actual)
Positive	TP = 95	FN = 5	100
Negative	FP = 300	TN = 600	900
Total (pred)	395	605	1000

Precision = 95 / (95 + 300)

- Step 1: 95 + 300 = **395**
- Step 2: 95 / 395 = divide numerator & denominator by 5 → 19 / 79
- Long division (19 ÷ 79) — digit by digit to 6 decimals:
  - 79 into 19 → 0, remainder 19 → decimal: 0.
  - 190 → 79×2 = 158 → remainder 32 → decimal digit 2 → 0.2
  - 320 → 79×4 = 316 → remainder 4 → decimal digit 4 → 0.24
  - 40 → 79×0 = 0 → remainder 40 → digit 0 → 0.240
  - 400 → 79×5 = 395 → remainder 5 → digit 5 → 0.2405
  - 50 → 79×0 = 0 → remainder 50 → digit 0 → 0.24050
  - Continue → pattern yields ≈ **0.240506...**
- As percent: **24.05%** (approx)

Recall = 95 / (95 + 5)

- Step 1: 95 + 5 = **100**
- Step 2: 95 / 100 = **0.95 → 95.00%**

F1 (use fractions): precision = 19/79, recall = 19/20.

- Multiply: (19/79) * (19/20) = 361 / 1580
- Numerator: 2 * 361/1580 = 722 / 1580
- Sum: 19/79 + 19/20 = (19×20)/(79×20) + (19×79)/(79×20) = (380 + 1501) / 1580 = 1881 / 1580
- Division: (722/1580) ÷ (1881/1580) = 722 / 1881
- Long division 722 ÷ 1881 = 0.3838383838... → **0.3838 → 38.38%**

Takeaway (C): You catch almost every real positive (95% recall) but most positive predictions are false (precision ≈ 24%). Good when *missing a positive is ve… costly*, and you can afford many false alarms (e.g., triage with human review).

1 What is F1?

The **F1 score** is the **harmonic mean** of **precision** and **recall**:

$$F1 = 2 \times \frac{(\text{Precision} \times \text{Recall})}{(\text{Precision} + \text{Recall})}$$

- It's **high** only when **both** precision and recall are high.
- If either precision or recall is low, F1 drops sharply.
- It's a **single number** to judge a model when **you care about both equally.**

2 Why harmonic mean?

The harmonic mean punishes **imbalances** more than the normal average:

- **Example:**
  - Avg of 90% and 10% (arithmetic mean) = 50%
  - But F1 = ~18% → tells you "nah, this model sucks for balance."
    This stops you from being fooled by one high number.

3 Example values & comparison

Let's compare four models:

Model	Precision	Recall	F1 Score	What it tells us
A	0.90	0.90	0.90	Excellent balance — catches most positives, and rarely wrong.
B	0.95	0.50	0.655	Very picky → high precision but misses half of actual positives.
C	0.50	0.95	0.655	Very generous → catches almost all positives but half are false alarms.
D	0.60	0.60	0.60	Meh performance — average in both, nothing stands out.

Relative interpretation

- **A vs B/C:** A is clearly better *overall* (balanced high numbers).
  B & C have the same F1 (0.655) even though one favors precision, the other recall → F1 ignores the "direction" of imbalance, just punishes the imbalance itself.
- **D:** Lower than B/C means it's weaker overall — both precision and recall are mediocre.
- **B vs C:** F1 says they're equally "balanced" in the *mathematical* sense, but *use case decides which is better* (e.g., spam filter → pick B, cancer detection → pick C).

---

4 What F1 really tells you

- Think of it as the **final combined score** when precision and recall are **team members** — if one is lazy, the team's score tanks.
- **High F1 (≥0.8)** → Good in both areas.
- **Medium (0.6–0.8)** → Tradeoffs exist; okay but not perfect.
- **Low (<0.6)** → Either precision, recall, or both are too weak.

Golden Rule to Follow (Must Practice)

Always calculate the F1 score of your model.

Exactly 💯 — that's the perfect TL;DR:

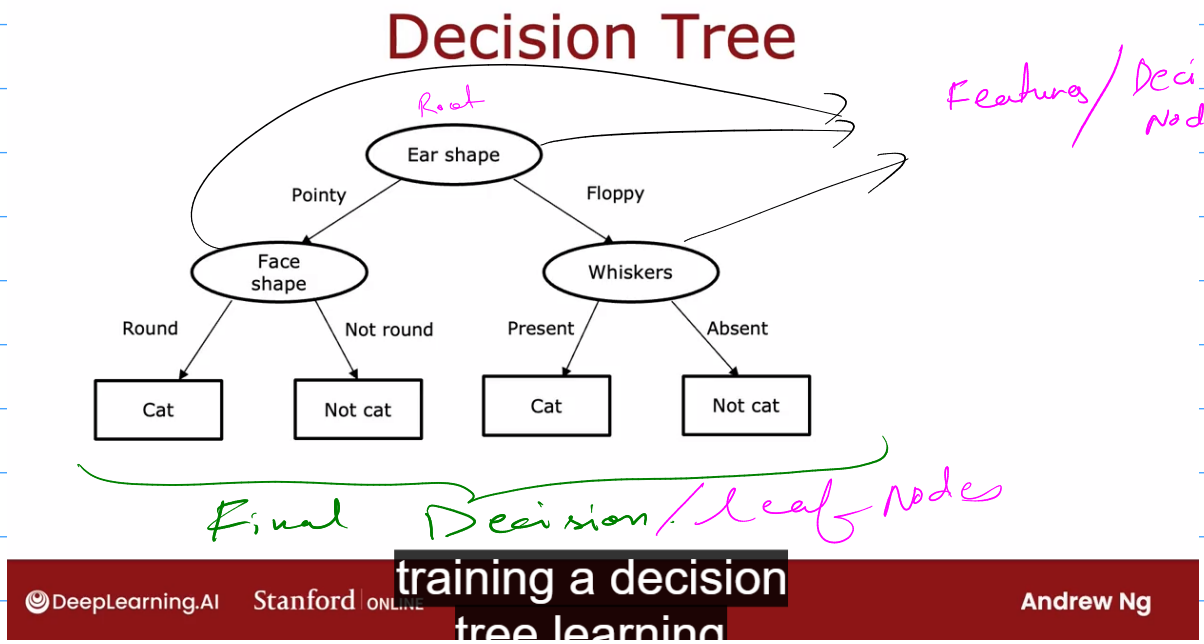
1. **Check F1** → Quick health check.
2. **If F1 is low** →
   - Look at **precision vs recall.**
   - If **both low** → model is trash, needs major work.
   - If **one high, one low** → could still be good depending on the business case (precision-heavy vs recall-heavy use case).
3. **If both high** → You've got a well-balanced, strong model. 🚀

That's literally the **precision–recall–F1 decision tree** in one line.

Note: Precision, Recall, F1 all these are for classification use cases!

Week -

Decision Tree Models.

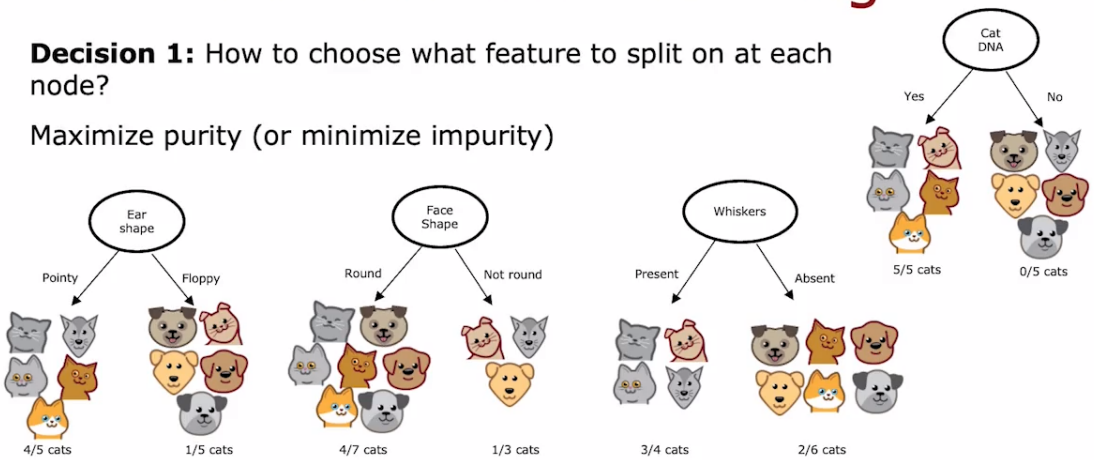


How to develop Decision Trees

Decision Tree Learning

Decision 1: How to choose what feature to split on at each node?

Maximize purity (or minimize impurity)



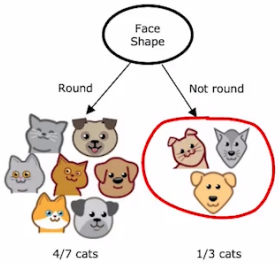
DeepLearning.AI Stanford ONLINE Andrew Ng

the greatest purity of

Decision Tree Learning

Decision 2: When do you stop splitting?

- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold

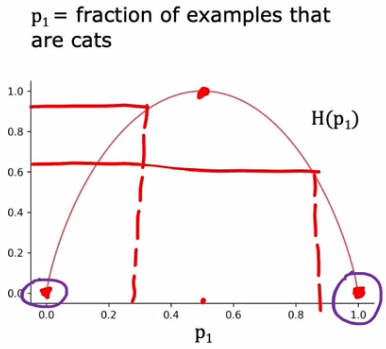


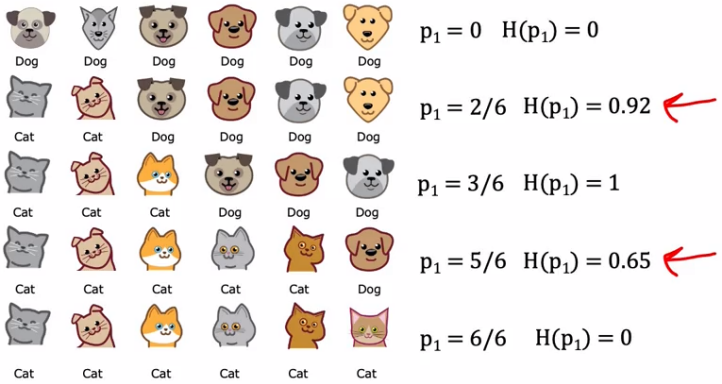
DeepLearning.AI Stanford ONLINE Andrew Ng

if you decided not to split

Measuring Purity

Entropy as a measure of impurity



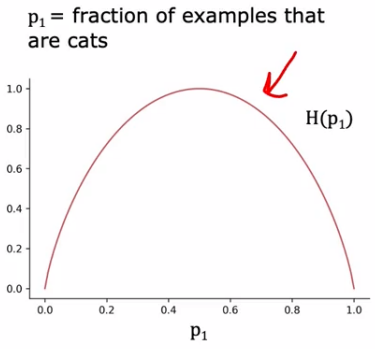


DeepLearning.AI Stanford ONLINE Andrew Ng

zero impurity or this would be

Entropy Formula

Entropy as a measure of impurity



$$p_0 = 1 - p_1$$

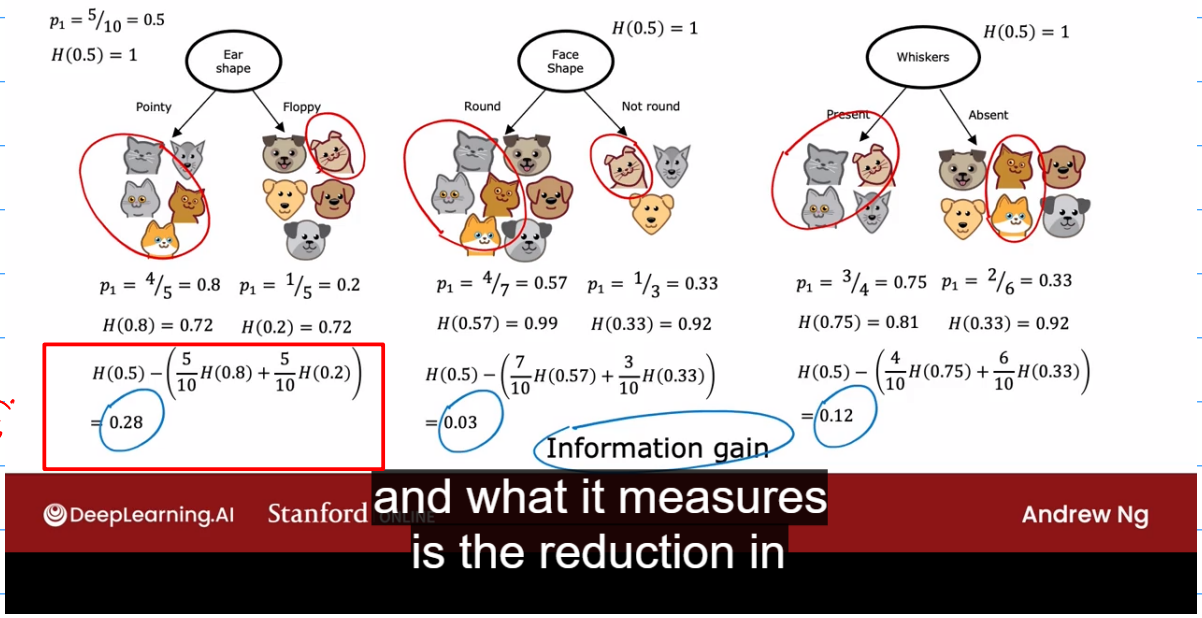
$$H(p_1) = -p_1 log_2(p_1) - p_0 log_2(p_0)$$
$$= -p_1 log_2(p_1) - (1 - p_1) log_2(1 - p_1)$$

DeepLearning.AI Stanford ONLINE Andrew Ng

that it will be exactly this function on the left.

Choosing the correct splitting Feature.

Information Gain

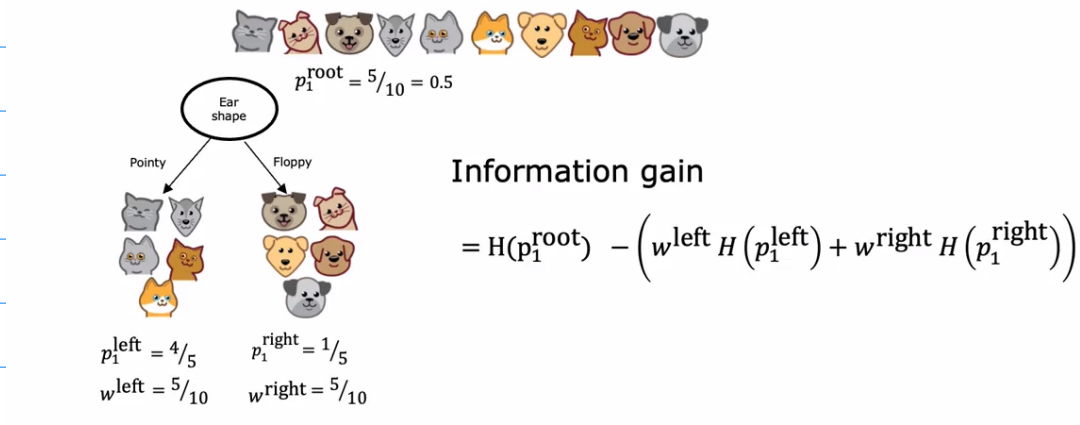


and what it measures is the reduction in

Reduction in info. gain.

Information gain

Maximum Information Gain is chosen as Splitting Node



on that increases the purity of

Decision Tree Learning

- Start with all examples at the root node
- Calculate information gain for all possible features, and pick the one with the highest information gain
- Split dataset according to selected feature, and create left and right branches of the tree
- Keep repeating splitting process until stopping criteria is met:
  - When a node is 100% one class
  - When splitting a node will result in the tree exceeding a maximum depth
  - Information gain from additional splits is less than threshold
  - When number of examples in a node is below a threshold

DeepLearning.AI Stanford Online Andrew Ng

You will keep on repeating the splitting process until

One hot encoding of Categorical Features

One hot encoding

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

DeepLearning.AI Stanford Online Andrew Ng

on only one of two possible values, either 0 or 1.

If a categorical feature can take on k values, then we have to create k binary features that take on values 0 & 1 as seen in the screenshot above.

One hot encoding and neural networks

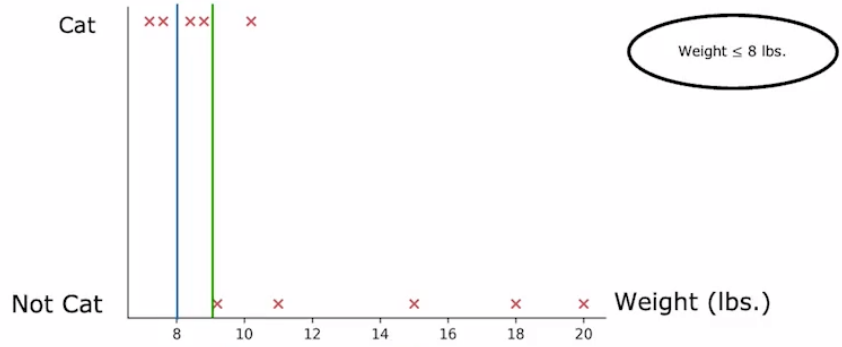
Pointy ears	Floppy ears	Round ears	Face shape	Whiskers	Cat
1	0	0	~~Round~~ 1	~~Present~~ 1	1
0	0	1	~~Not round~~ 0	~~Present~~ 1	1
0	0	1	~~Round~~ 1	~~Absent~~ 0	0
1	0	0	~~Not round~~ 0	~~Present~~ 1	0
0	0	1	~~Round~~ 1	~~Present~~ 1	1
1	0	0	~~Round~~ 1	~~Absent~~ 0	1
0	1	0	~~Not round~~ 0	~~Absent~~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

DeepLearning.AI Stanford Online Andrew Ng

one for whiskers and encoded as a list of these five features.

Continued Valued Features

Splitting on a continuous variable

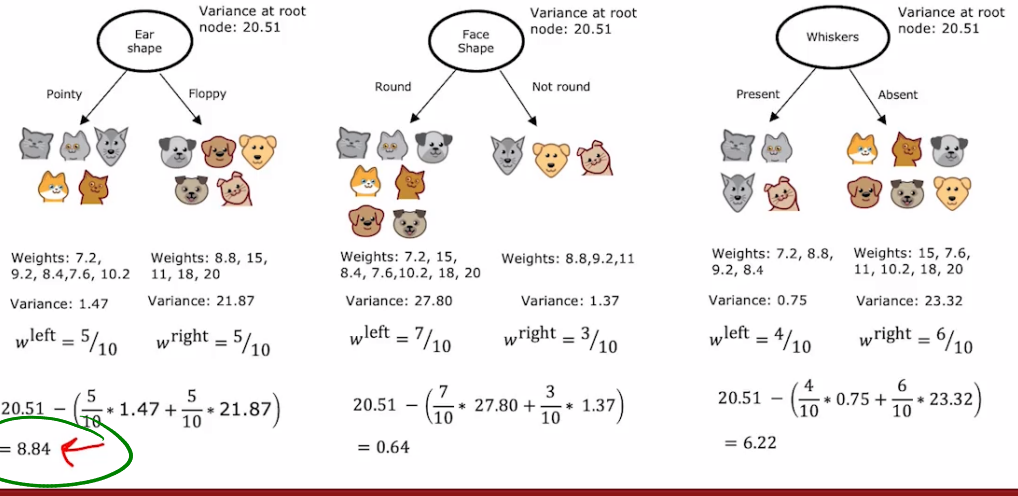


$$H(0.5) - \left(\frac{2}{10}H\left(\frac{2}{2}\right) + \frac{8}{10}H\left(\frac{3}{8}\right)\right) = 0.24$$ IG when threshold is wt ≤ 8 lbs

$$H(0.5) - \left(\frac{4}{10}H\left(\frac{4}{4}\right) + \frac{6}{10}H\left(\frac{1}{6}\right)\right) = 0.61$$ IG when threshold is wt ≤ 9 lbs.

Regression Trees

Choosing a split



DeepLearning.AI Stanford ONLINE Andrew Ng

Largest reduction in variance would give you the splitting feature.