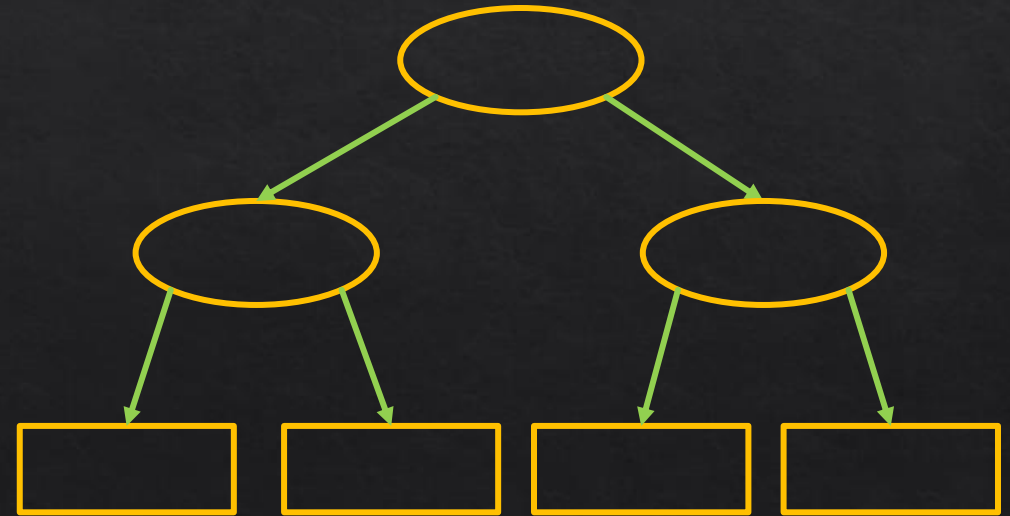
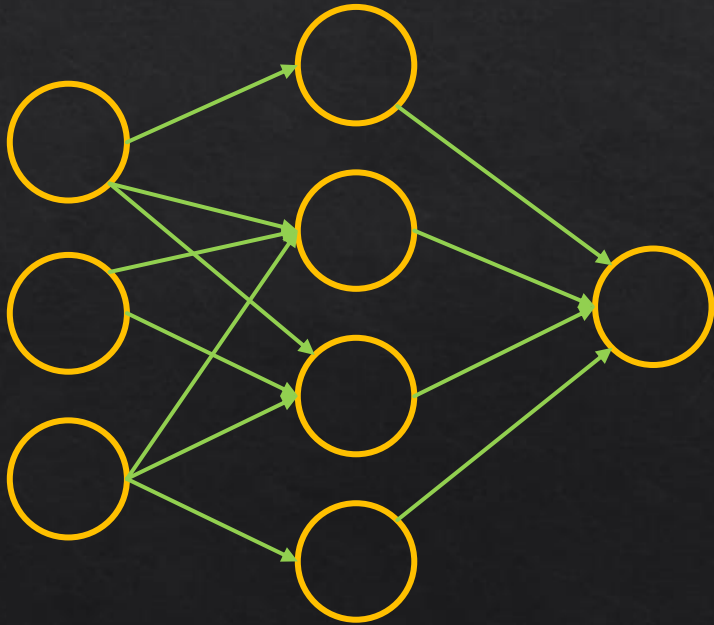


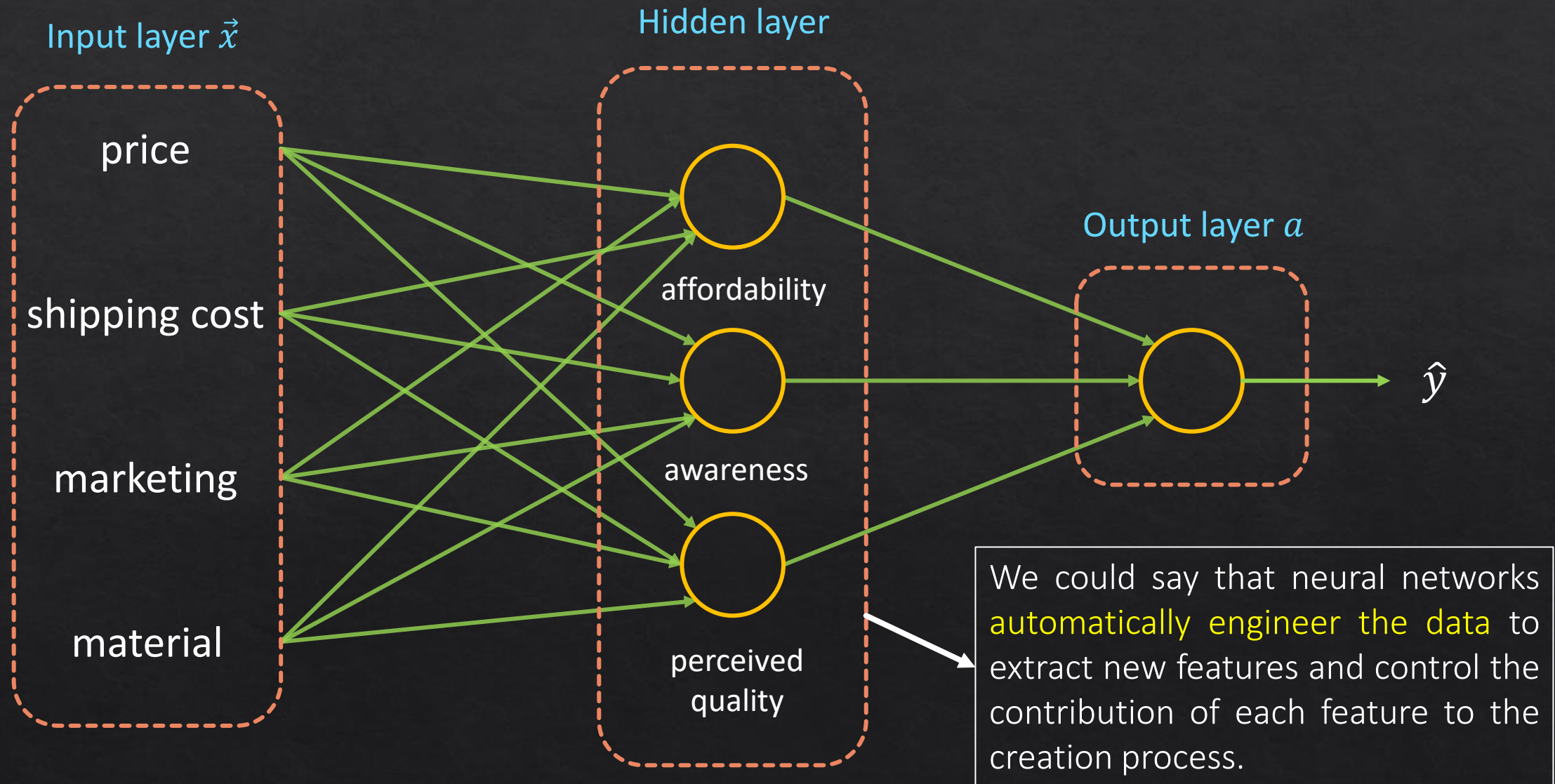
Advanced Machine Learning

Neural Networks – Decision Trees – Random Forest – XGBoost



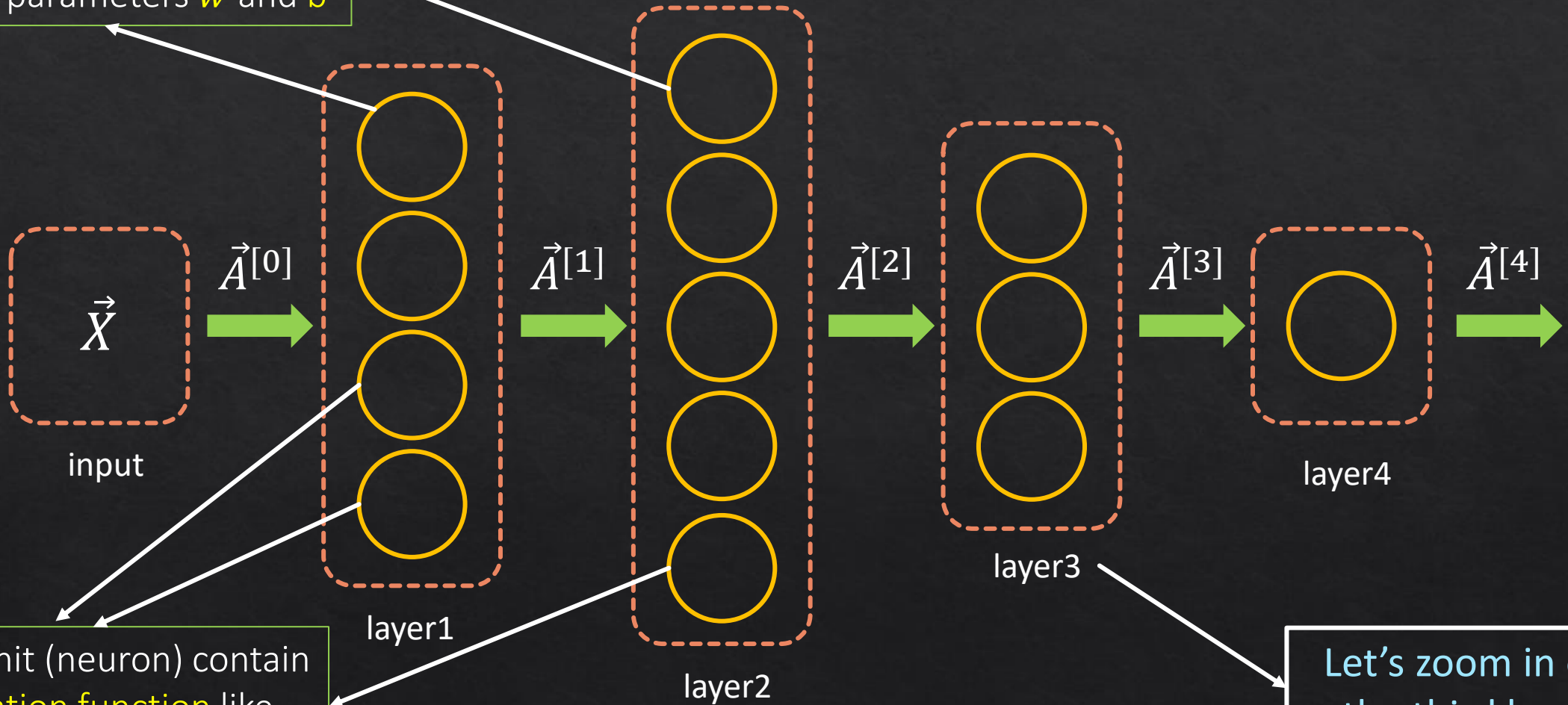
Neural Networks

Neural Networks



Neural Networks Notation

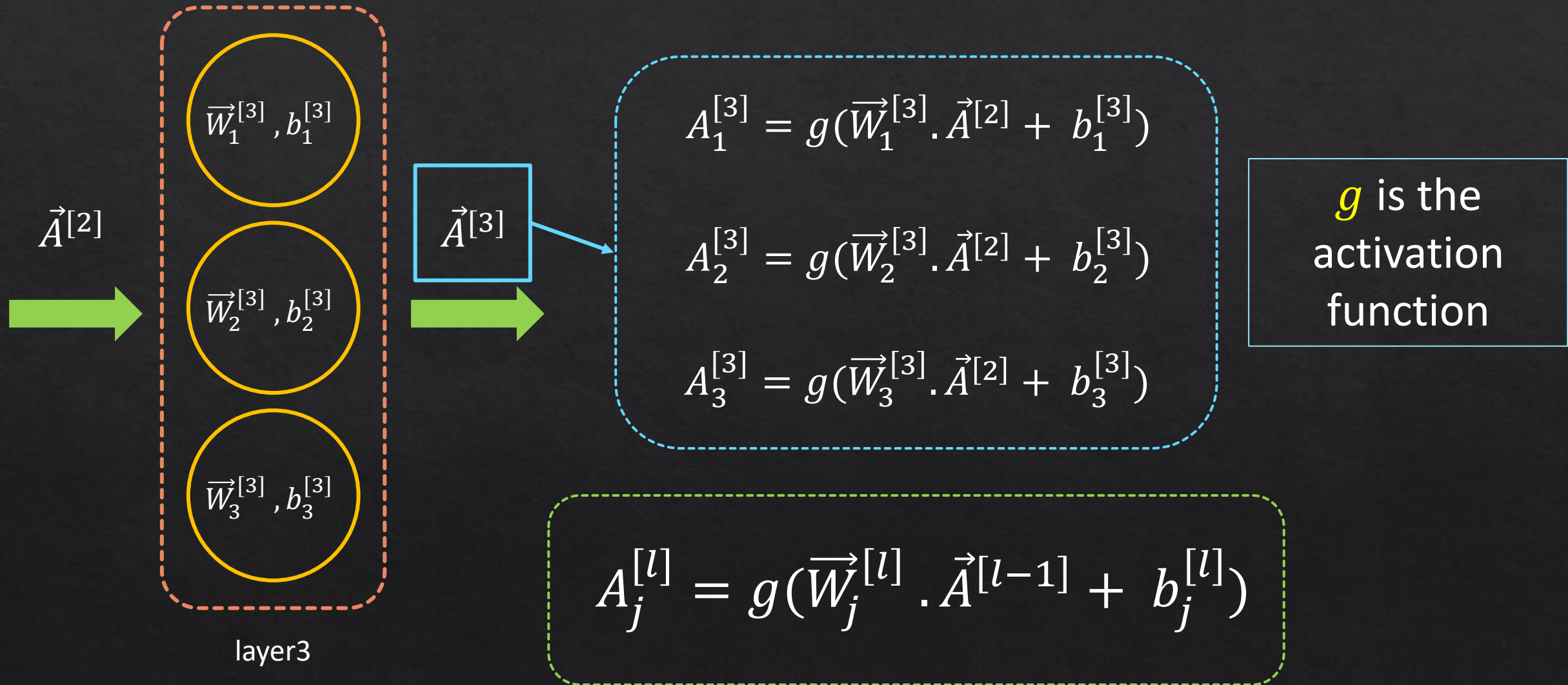
every unit (neuron) contains
its own parameters \vec{w} and b



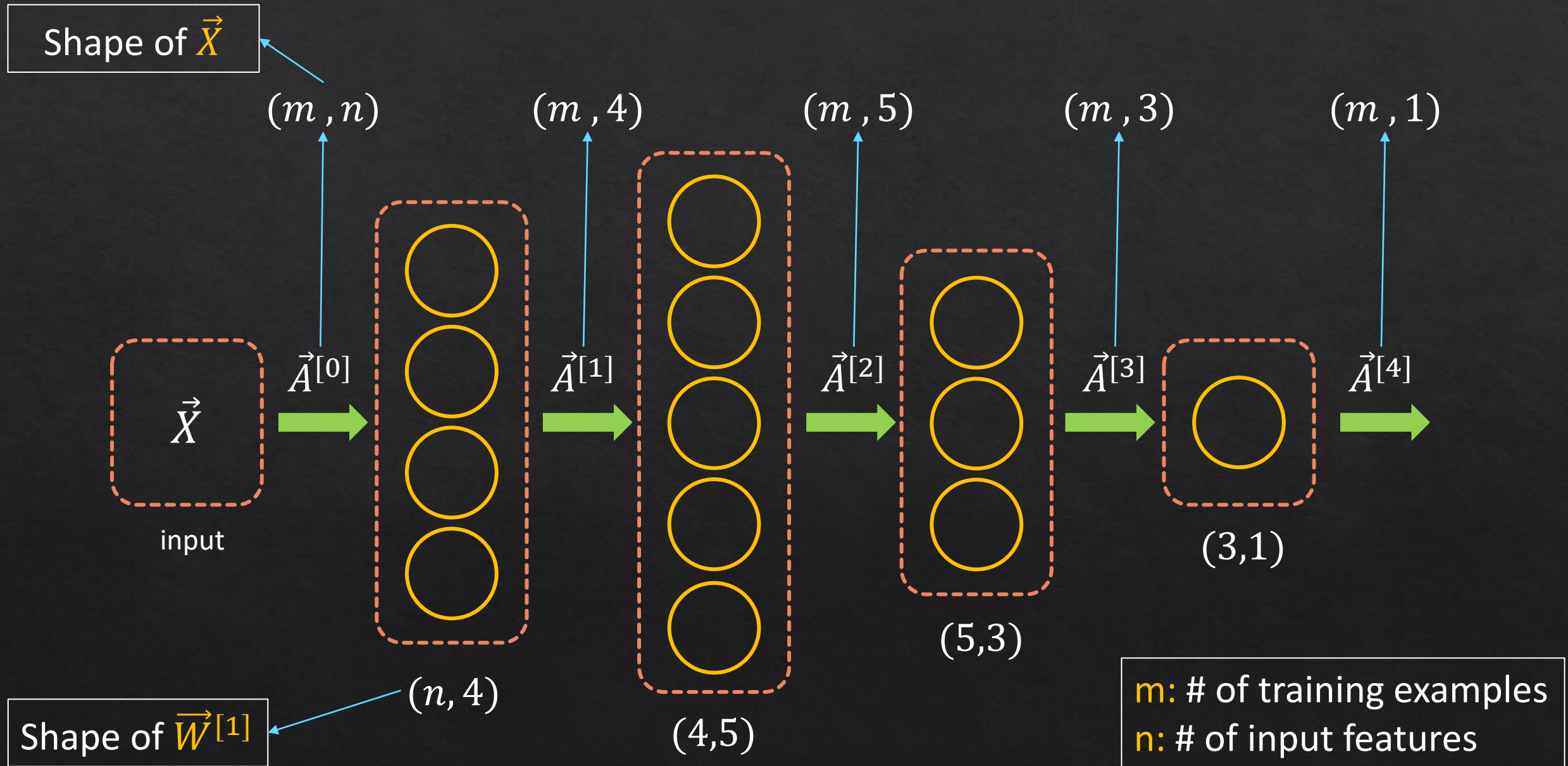
every unit (neuron) contain
activation function like
sigmoid, tanh, Relu ...

Let's zoom in on
the third layer

Neural Networks Notation



Neural Networks Notation (Matrices Shapes)

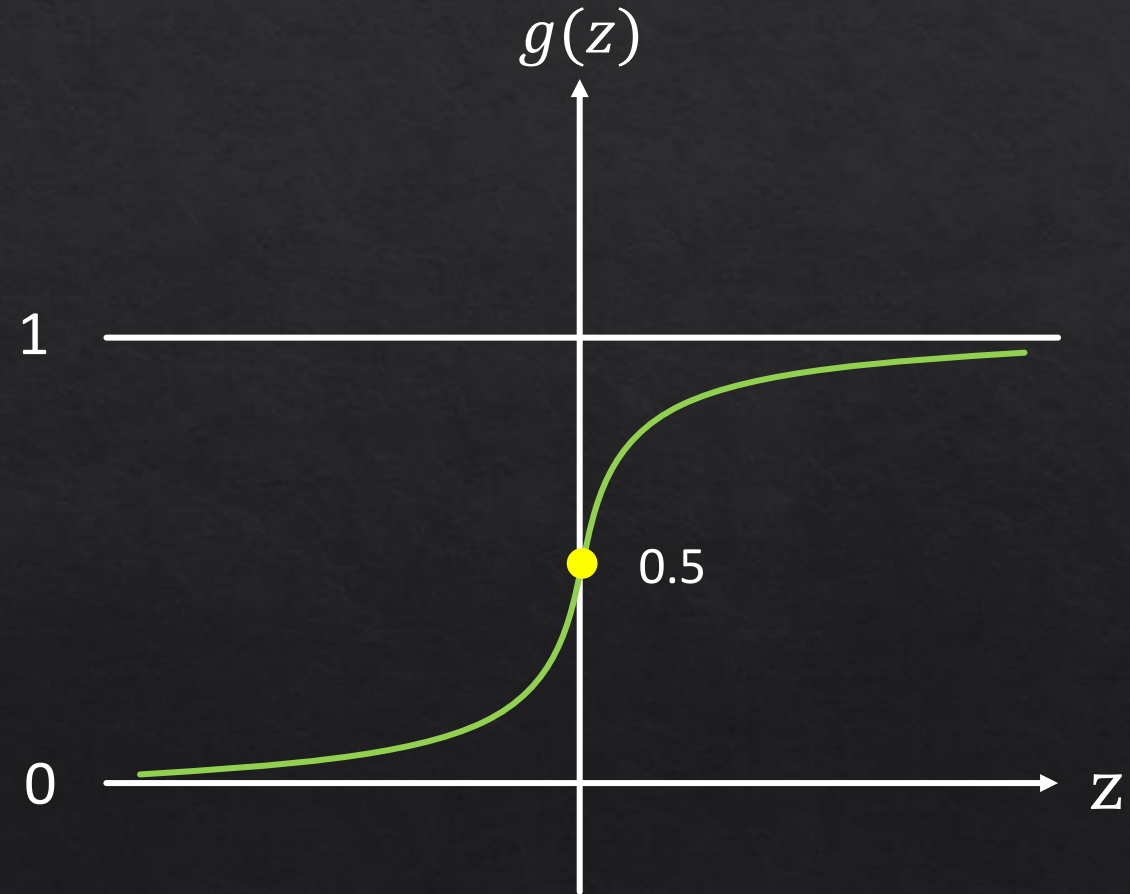


Activation Function Types (Sigmoid)

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

Binary classification

It is commonly used for models where we have to predict the probability as an output. Since the probability of anything exists only between the range of 0 and 1, sigmoid is the right choice because of its range.

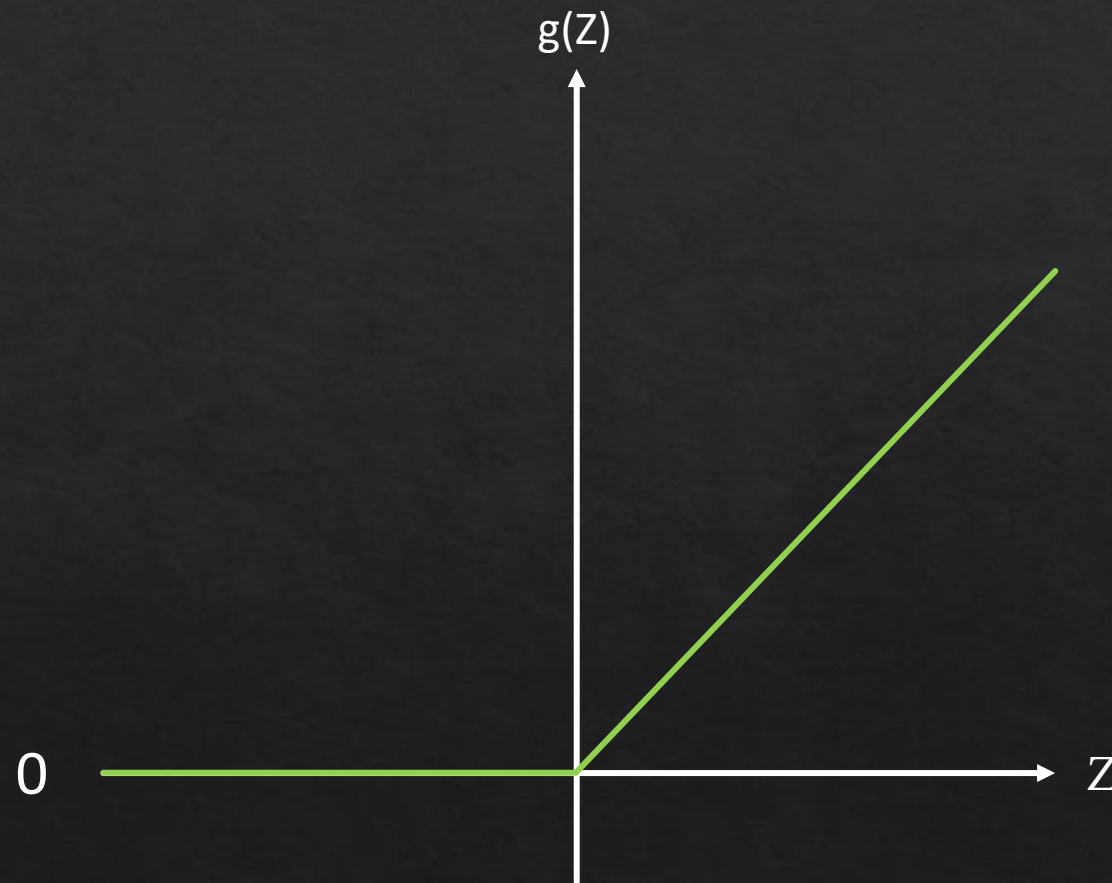


Activation Function Types (ReLU)

$$g(z) = \text{ReLU}(z) = \max(0, z)$$

Regression

It is commonly used for models where we have to predict any **positive** numeric value as an **output** (as in the case of the regression problem like house price), and ReLU is the most common choice for **hidden layers** because of its **low compute** time and its **non-flat part**.



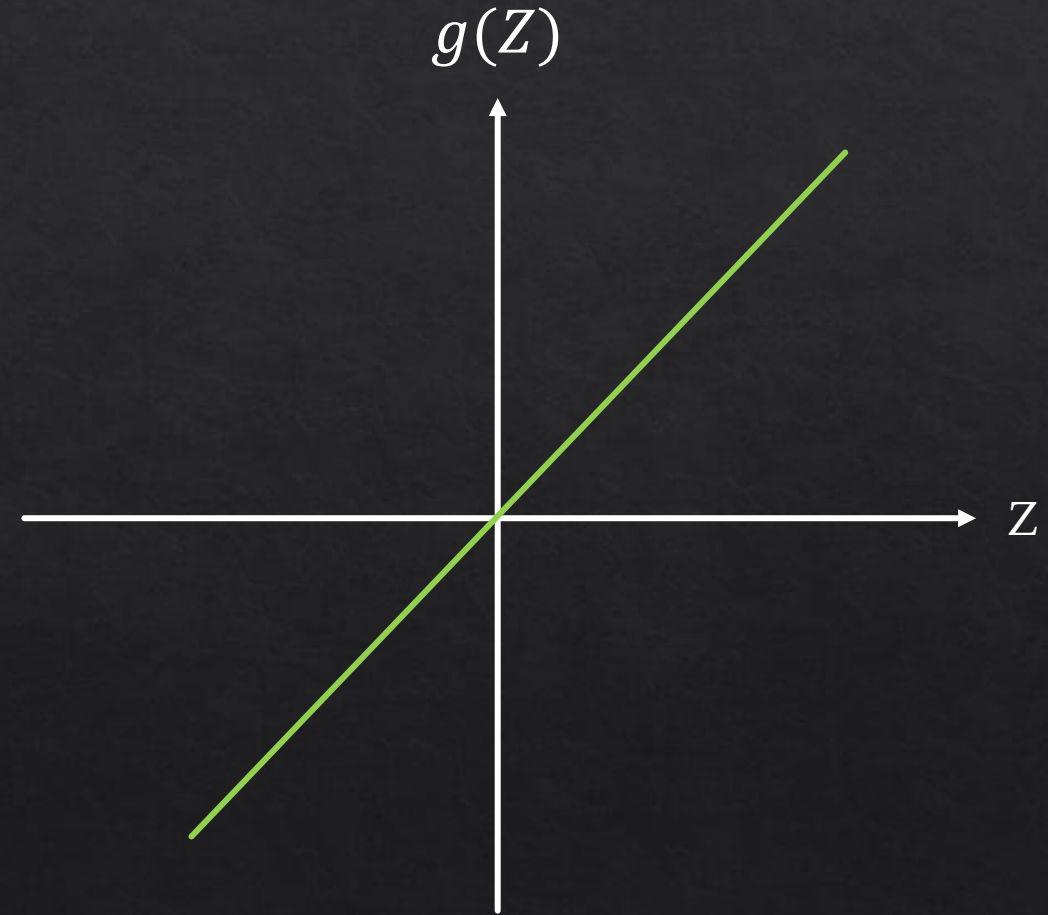
Activation Function Types (Linear Activation Function)

$$g(z) = z$$

Regression

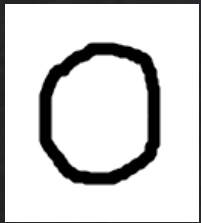
It is commonly used for models where we have to predict **any** numeric value as an output (as in the case of the regression problem like total profit)

Note: There are many other activation function types like: **tanh**, **leaky ReLU**, and **SoftMax** ... and we will talk about them later.



Binary Classification NN Example

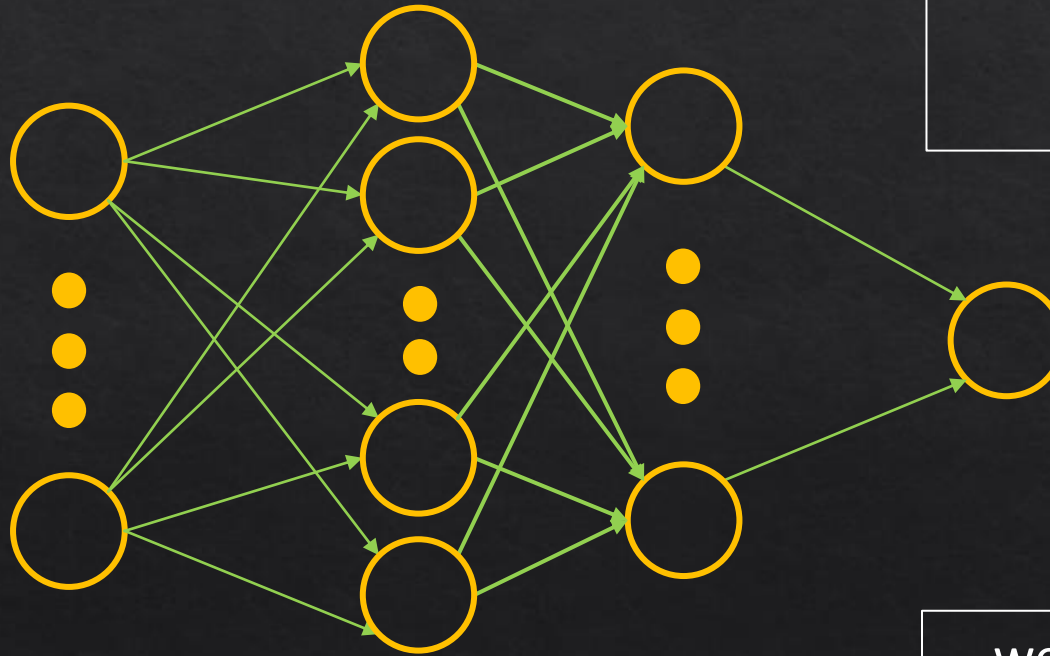
We want to build a neural network to predict if a **handwritten image number is 0 or 1**.



OR



The **input** is the image pixels



The activation function for the output neuron is **sigmoid** to predict **probability from 0 to 1**

threshold: $\hat{y} \geq 0.5$

0 Or 1

we will use the **cross-entropy loss function** because we are trying to predict one of two classes

Binary Classification NN Example

We will build it using Tensorflow:

```
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")
    ]
)
```

The first step is building and stringing the NN layers

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

Then we will compile the model and select the loss function

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

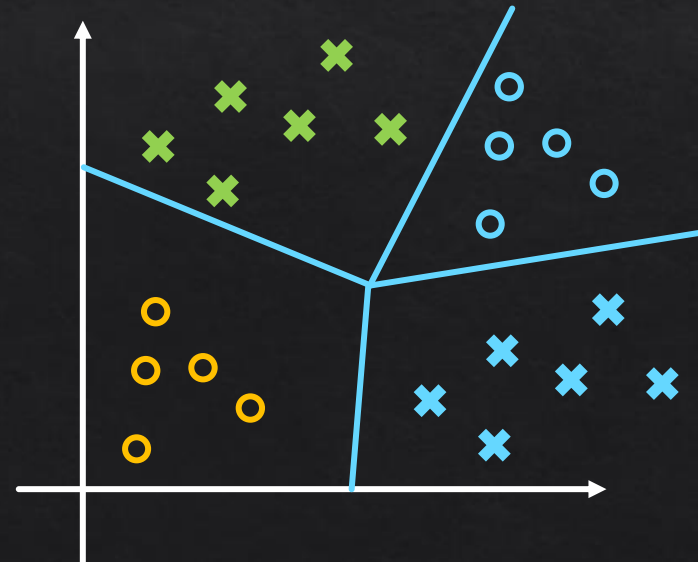
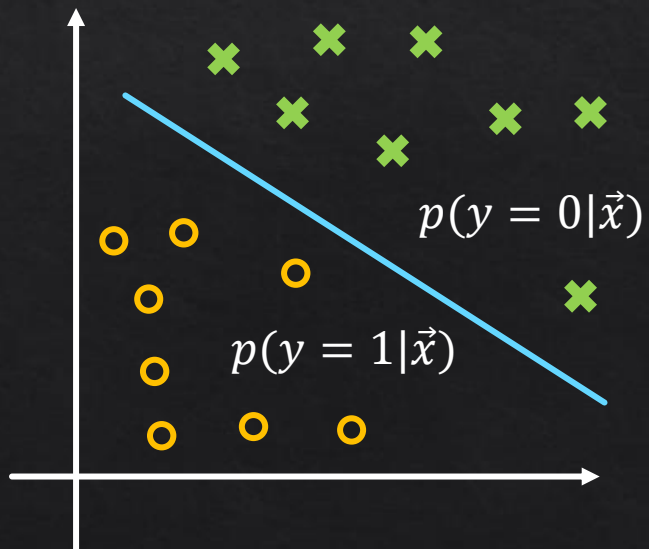
The last step is to train the NN

Multiclass Classification

What if we have more than two classes?

- Recognize all numbers
- Recognize if a patient has one of three or five different diseases.
- Recognize animal categories.

2 Classes



Multiclass

How to apply it?

SoftMax (4 Possible outputs)

Logistic Regression

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

○ $a1 = g(z) = \frac{w}{1 + e^{-z}} = p(y = 1 | \vec{x})$

✕ $a2 = 1 - a1 = p(y = 0 | \vec{x})$

SoftMax Regression

○ $z_1 = \vec{w}_1 \cdot \vec{x} + b_1$ $a_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

✕ $z_2 = \vec{w}_2 \cdot \vec{x} + b_2$ $a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

○ $z_3 = \vec{w}_3 \cdot \vec{x} + b_3$ $a_3 = \frac{e^{z_3}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

✕ $z_4 = \vec{w}_4 \cdot \vec{x} + b_4$ $a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$

SoftMax (N Possible outputs)

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

$$z_j = \vec{w}_j \cdot \vec{x} + b_j : j = 1, 2, \dots, n$$

note: $a_1 + a_2 + \dots + a_n = 1$

Cost

Logistic Regression

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

$$a1 = g(z) = \frac{w}{1 + e^{-z}} = p(y = 1 | \vec{x})$$

$$a2 = 1 - a1 = p(y = 0 | \vec{x})$$

$$loss = -y \log(a_1) - (1 - y) \log(1 - a1)$$

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

SoftMax Regression

$$a_1 = \frac{e^{z1}}{e^{z1} + e^{z2} + e^{z3} + e^{z4}} = p(y = 1 | \vec{x})$$



$$a_N = \frac{e^{zN}}{e^{z1} + e^{z2} + e^{z3} + e^{z4}} = p(y = N | \vec{x})$$

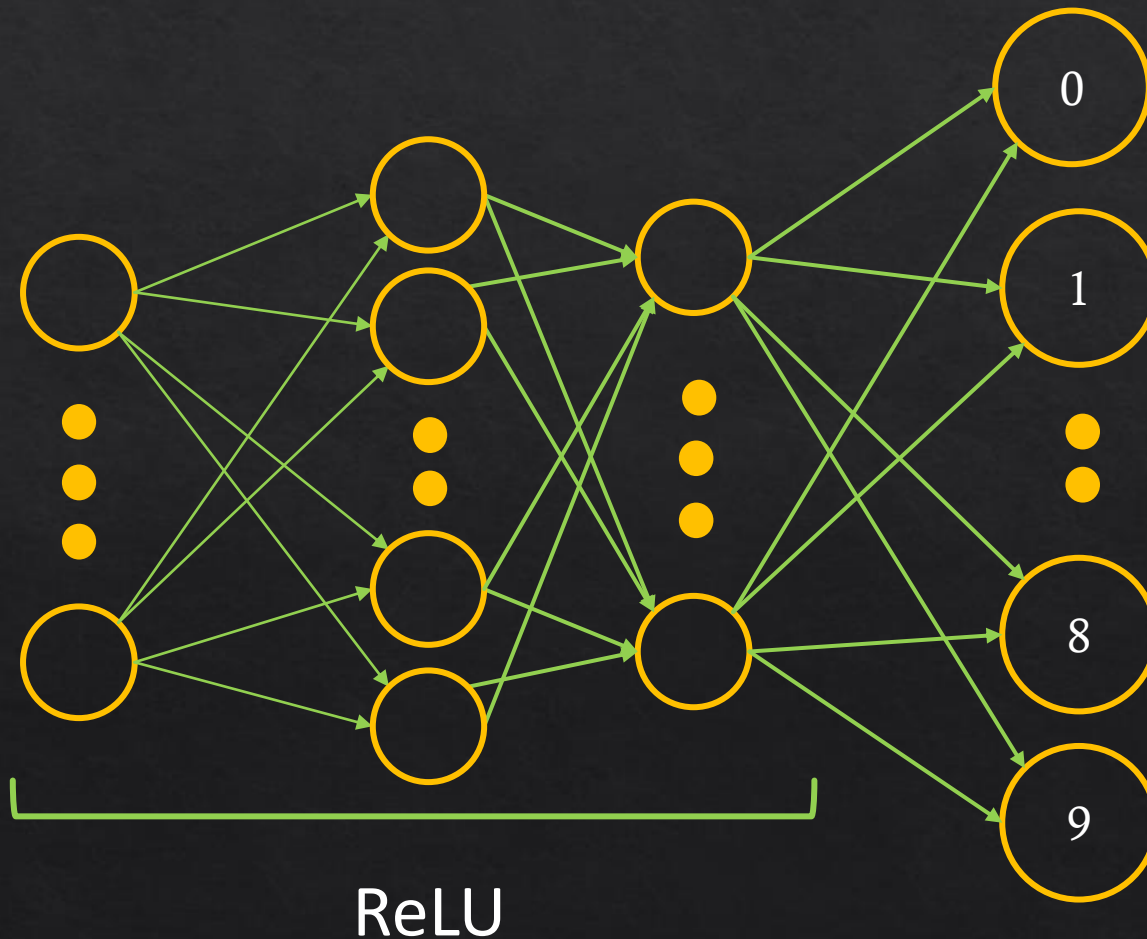
$$loss(a_1, a_2, \dots, a_N, y) = \begin{cases} -\log(a_1) & \text{if } y = 1 \\ -\log(a_2) & \text{if } y = 2 \\ \vdots & \\ -\log(a_N) & \text{if } y = N \end{cases}$$

Multiclass Classification NN Example

We want to build a neural network to recognize a **handwritten image number**.



The **input** is the
image pixels



The activation
function for the
output neuron is
SoftMax to predict
probability from 0 to
1 for each class.

Multiclass Classification NN Example

We will build it using Tensorflow:

```
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")
    ]
)
```

The first step is building and stringing the NN layers

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

Then we will compile the model and select the loss function

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

The last step is to train the NN

Multiclass Classification NN Example

More numerically accurate implementation of logistic loss:

```
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")
    ]
)

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

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


Advanced Optimization Algorithm

Adam(Adaptive moment estimation)

w_2



Every step is going in the same
direction
Let's go faster (increase α)

w_1

w_2



Every step is going in a different
direction
Let's fix it (decrease α)

w_1

So instead of having a single
learning rate for all parameters,
Adam uses a **different one for
every** parameter and **changes it
automatically** through training.

Machine Learning Advices

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

$$J(w, b) = \frac{1}{2m} \sum_i^m (f_{w,b}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_j^n w_j^2$$

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

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try a more complex model
- Try increasing λ
- Try decreasing λ

Evaluating a Model

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

70% Train Data

We need to know how our model fits **training** and **testing** data.

So we compute **cost function** for **train data** and for **test data**

30% Test Data

Sometimes Model **fits the training data** **very** well but **fails to generalize to new examples** in the training set.

Model Selection

(choosing the right model)

Suppose we have to select one of these models:

$$d = 1 \Rightarrow f_{\vec{w},b}(\vec{x}) = w_1x_1 + b$$

$$d = 2 \Rightarrow f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_1^2 + b$$



$$d = 10 \Rightarrow f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_1^2 + \dots + w_{10}x_1^{10} + b$$

$$w^{<1>}, b^{<1>} \Rightarrow j_{test}(w^{<1>}, b^{<1>})$$

$$w^{<2>}, b^{<2>} \Rightarrow j_{test}(w^{<2>}, b^{<2>})$$

$$w^{<10>}, b^{<10>} \Rightarrow j_{test}(w^{<10>}, b^{<10>})$$

After calculating test costs for every model, we found that $j_{test}(w^{<5>}, b^{<5>})$ has the lowest value, Should we select the fifth model?

Model Selection

(choosing the right model)

The Problem is $j_{test}(w^{<5>}, b^{<5>})$ is likely to be an **optimistic estimate of generalization error**. I.e: An extra parameter d (degree of polynomial) was chosen using the test set.

So we need an extra dependent set for model selection ... What is its name?

Cross Validation Set (validation set, development set, dev set)

Model Selection

(choosing the right model)

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

60% Train Data

20% Cross-Validation Data

20% Test Data

Model Selection

(choosing the right model)

$$d = 1 \Rightarrow f_{\vec{w},b}(\vec{x}) = w_1x_1 + b$$

$$w^{<1>}, b^{<1>} \Rightarrow j_{cv}(w^{<1>}, b^{<1>})$$

$$d = 2 \Rightarrow f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_1^2 + b$$

$$w^{<2>}, b^{<2>} \Rightarrow j_{cv}(w^{<2>}, b^{<2>})$$



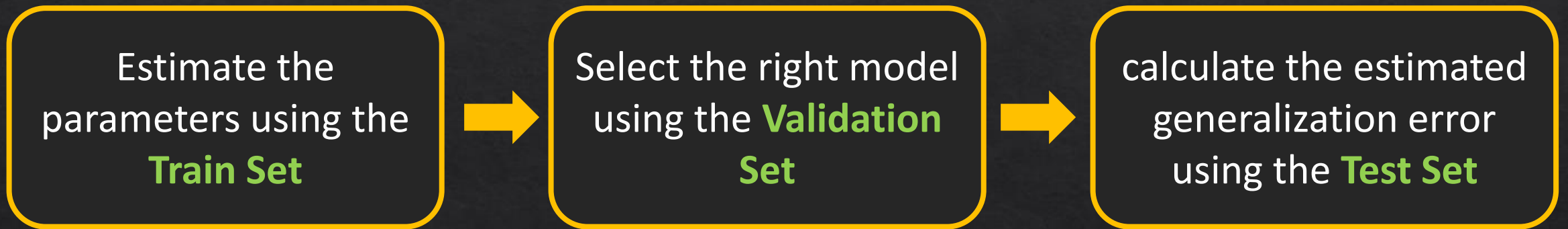
$$d = 10 \Rightarrow f_{\vec{w},b}(\vec{x}) = w_1x_1 + w_2x_1^2 + \dots + w_{10}x_1^{10} + b$$

$$w^{<10>}, b^{<10>} \Rightarrow j_{cv}(w^{<10>}, b^{<10>})$$

We will select the model that has the least **validation error**, Then calculate the estimated **generalization error** using the test set.

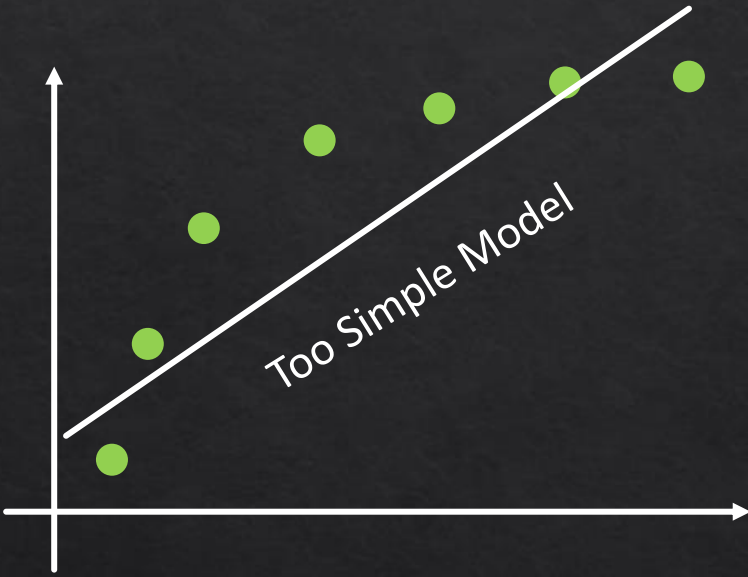
Model Selection

(choosing the right model)



Bias / Variance (Model Complexity)

Underfitting

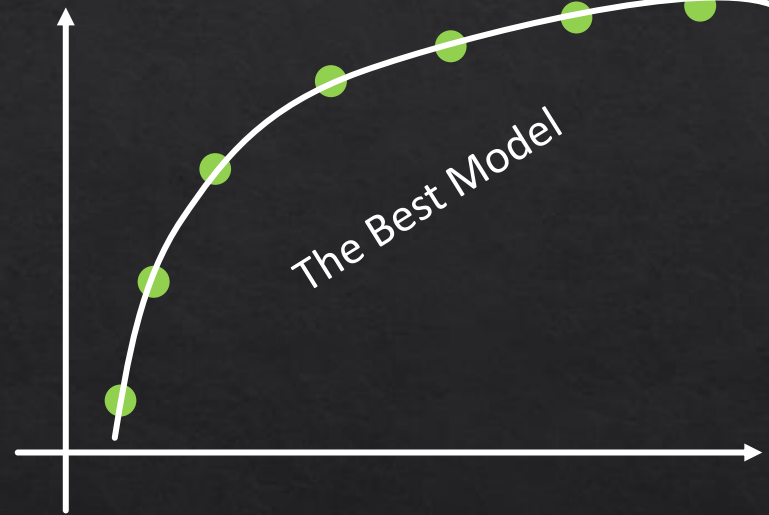


High Bias

J_{train} is high

J_{cv} is high

Sweet spot – Sugar place
“just right”

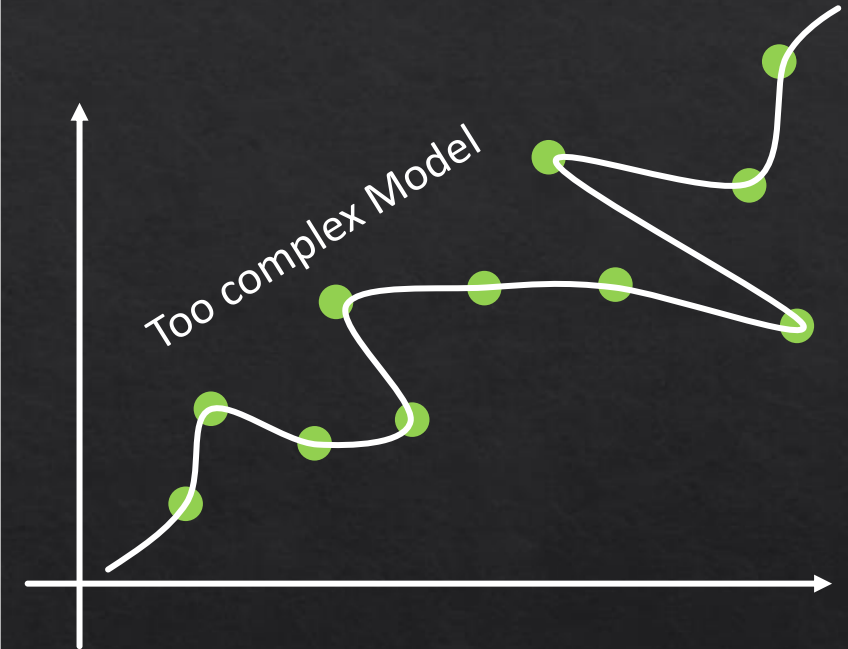


Just Right

J_{train} is low

J_{cv} is low

Overfitting

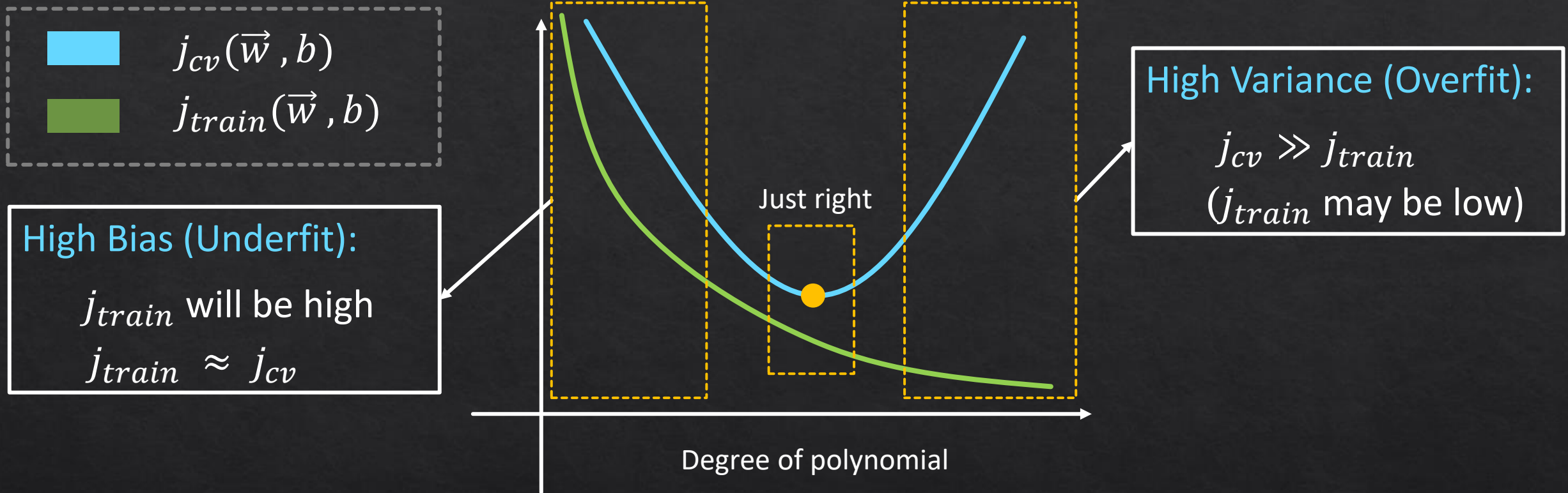


High Variance

J_{train} is low

J_{cv} is high

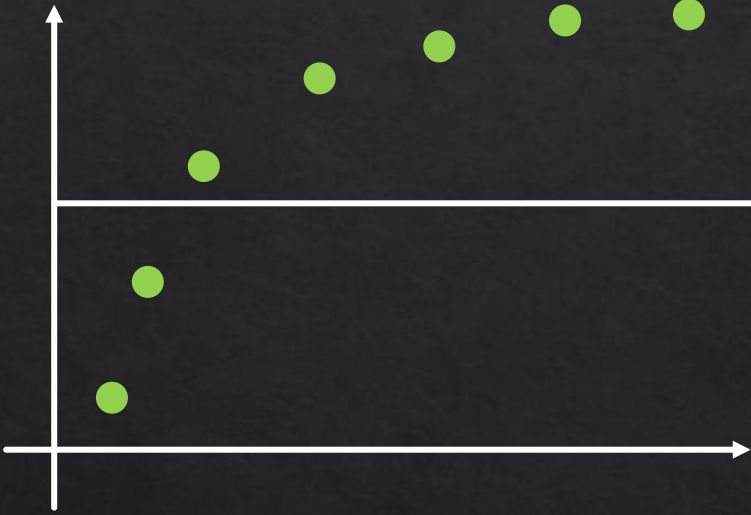
Bias / Variance with Model Complexity



Note: Sometimes maybe we have High Bias and High Variance: (j_{train} will be high **AND** $j_{cv} \gg j_{train}$) this happened when the model overfits for part of the input (very complicated model), And for the other part of the input it does not fit the data will (underfit)

Bias / Variance (with regularization)

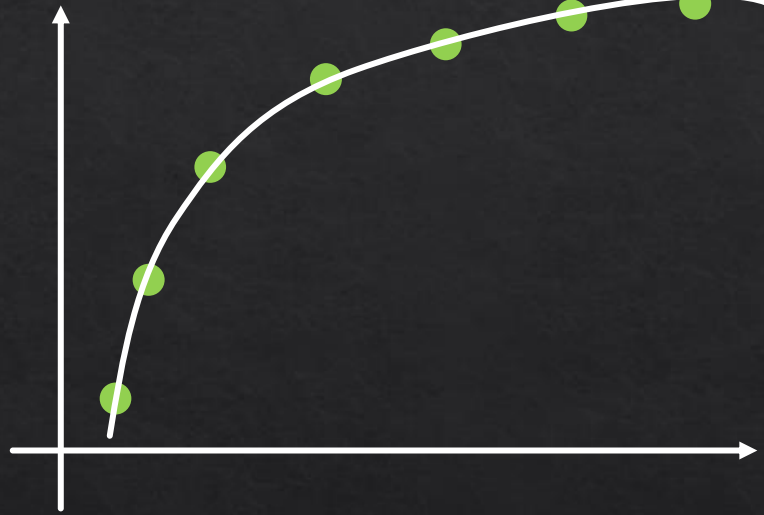
Underfitting



High Bias

λ is too high $\Rightarrow \vec{w}$ will be zero

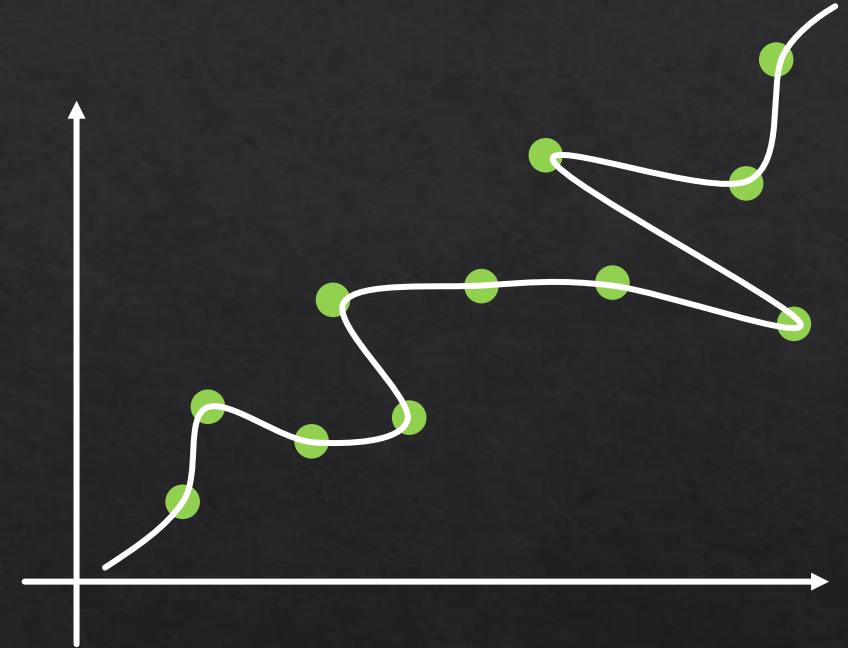
Sweet spot – Sugar place
“just right”



Just Right

λ is right

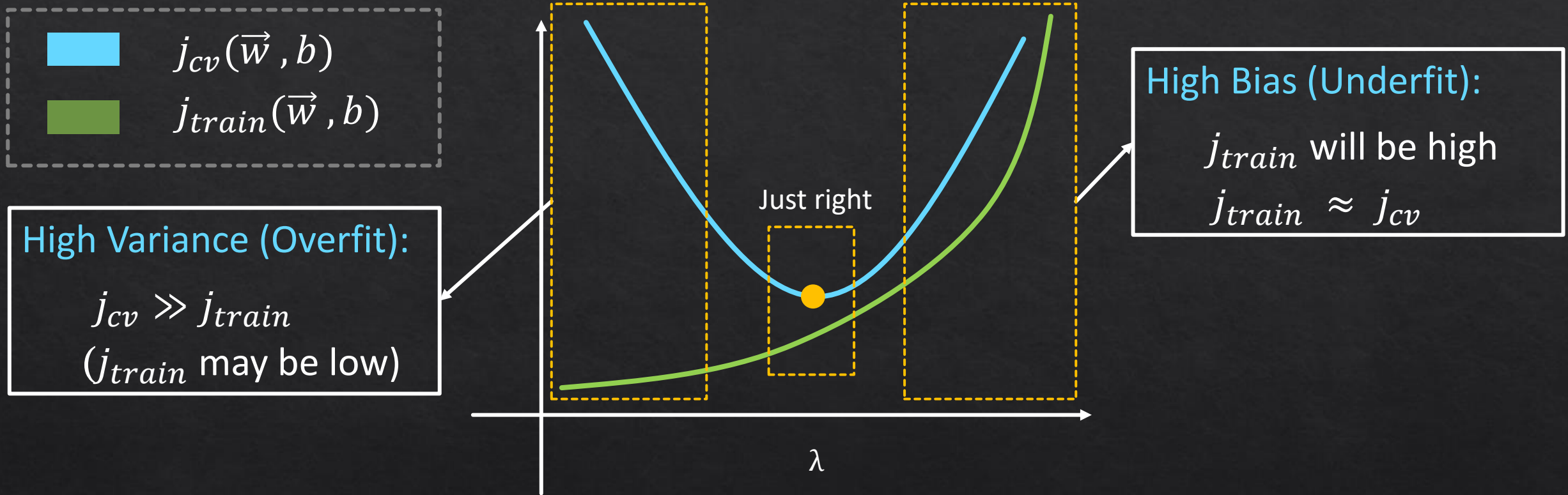
Overfitting



High Variance

λ is too low \Rightarrow no regularization

Bias / Variance with Regularization



Note: The regularization curve is the horizontal flip of the polynomial degree curve.

Establishing a baseline level of performance

Suppose we want to build a speech recognition application (Audio to Text):

After building the system we got these errors:

Training Error $j_{train} = 10.8\%$

Cross Validation Error $j_{cv} = 14.3\%$

How to know if these error values are good or bad?

We need to establish a baseline level of performance to compare with.

Human level performance = 10.6%

Training Error $j_{train} = 10.8\%$

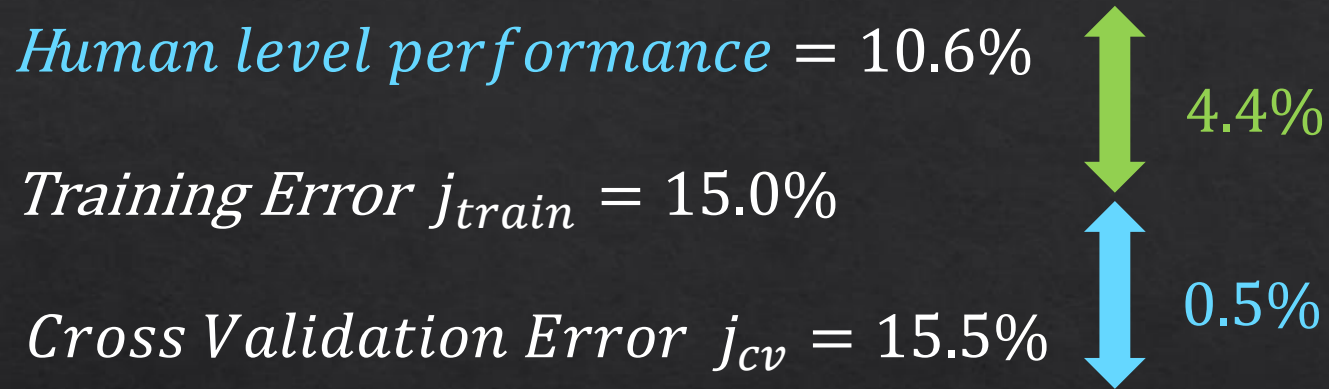
Cross Validation Error $j_{cv} = 14.8\%$

0.2%

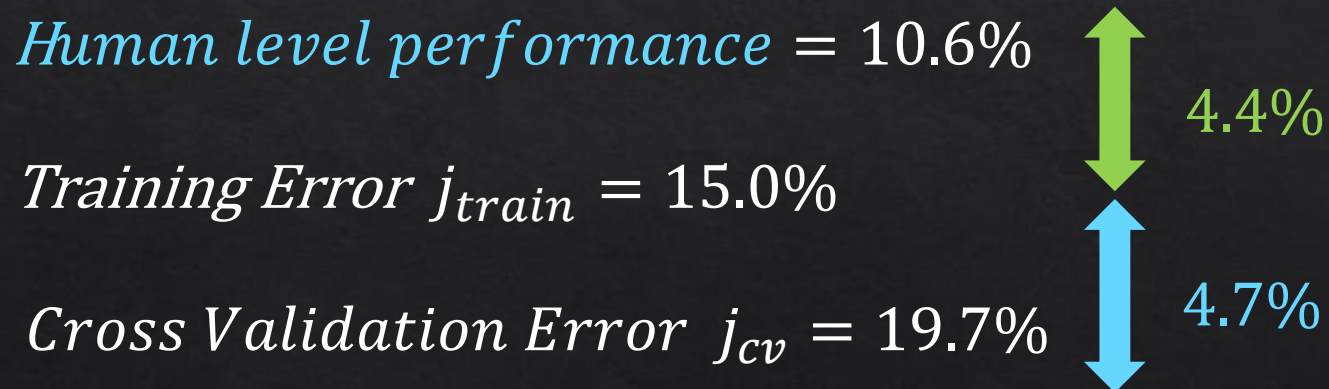
4.0%

Now we can see that training error is close to humans error,
But cross-validation error is much higher than training error
=> high variance (Overfitting)

Establishing a baseline level of performance



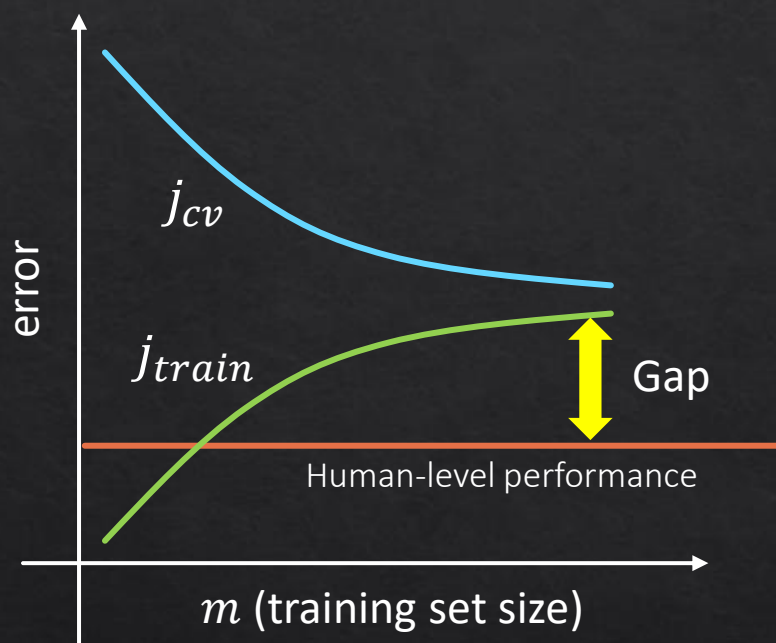
training error and cross-validation error are much higher than human error => **high bias (Underfitting)**



training error is much higher than humans error, And cross-validation error is much higher than training error => **high bias, high variance**

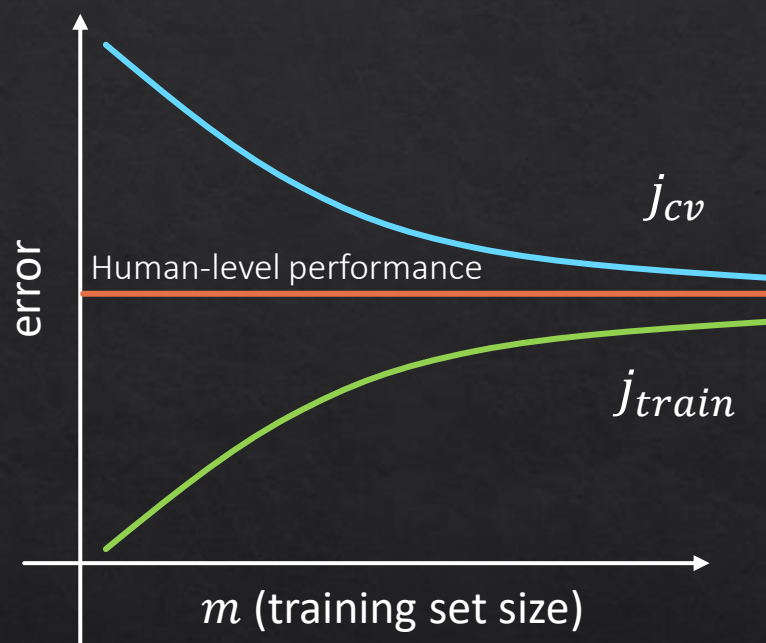
Learning Curves

Underfitting

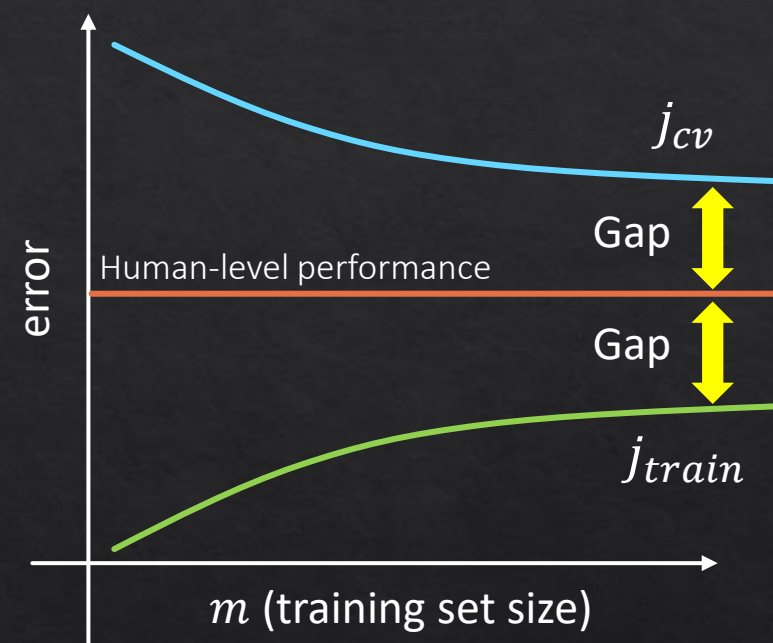


If a learning algorithm suffers from **high bias**, getting more training data **will not** (by itself) help much

Sweet spot – Sugar place “just right”



Overfitting



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

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

$$J(w, b) = \frac{1}{2m} \sum_i^m (f_{w,b}(x^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_j^n w_j^2$$

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

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try a more complex model
- Try increasing λ
- Try decreasing λ

Fixes high variance

Fixes high variance

Fixes high bias

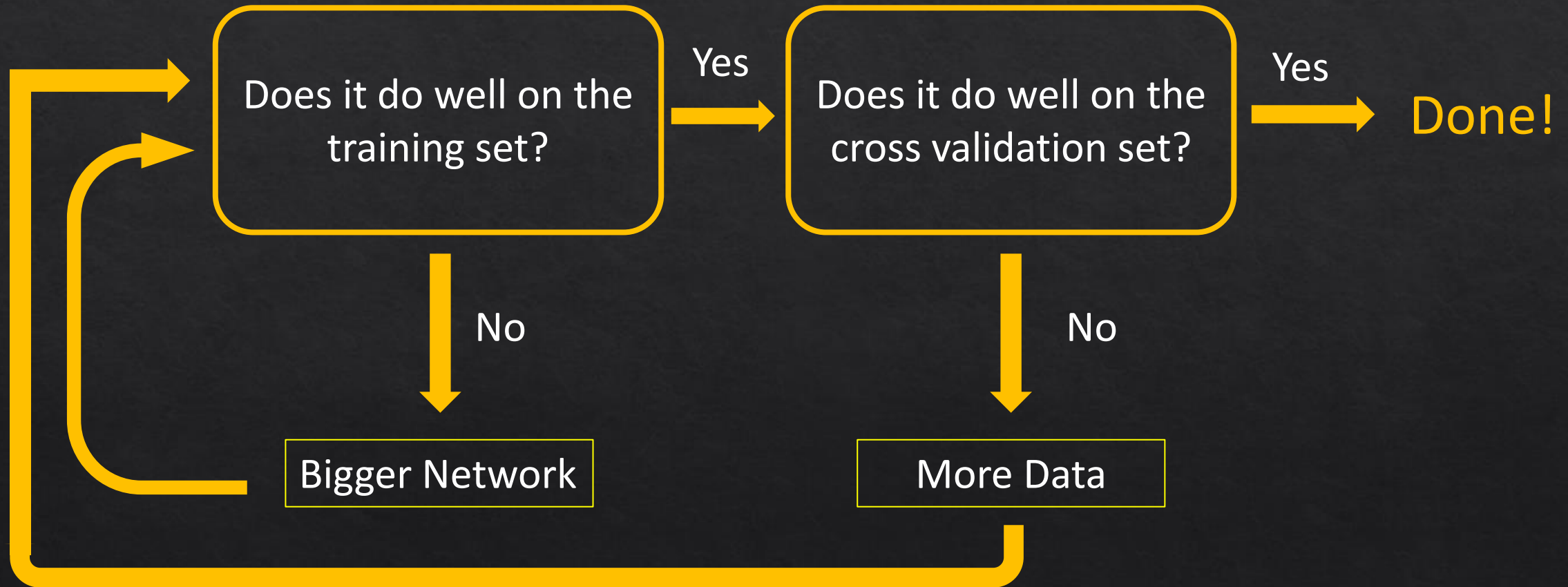
Fixes high bias

Fixes high bias

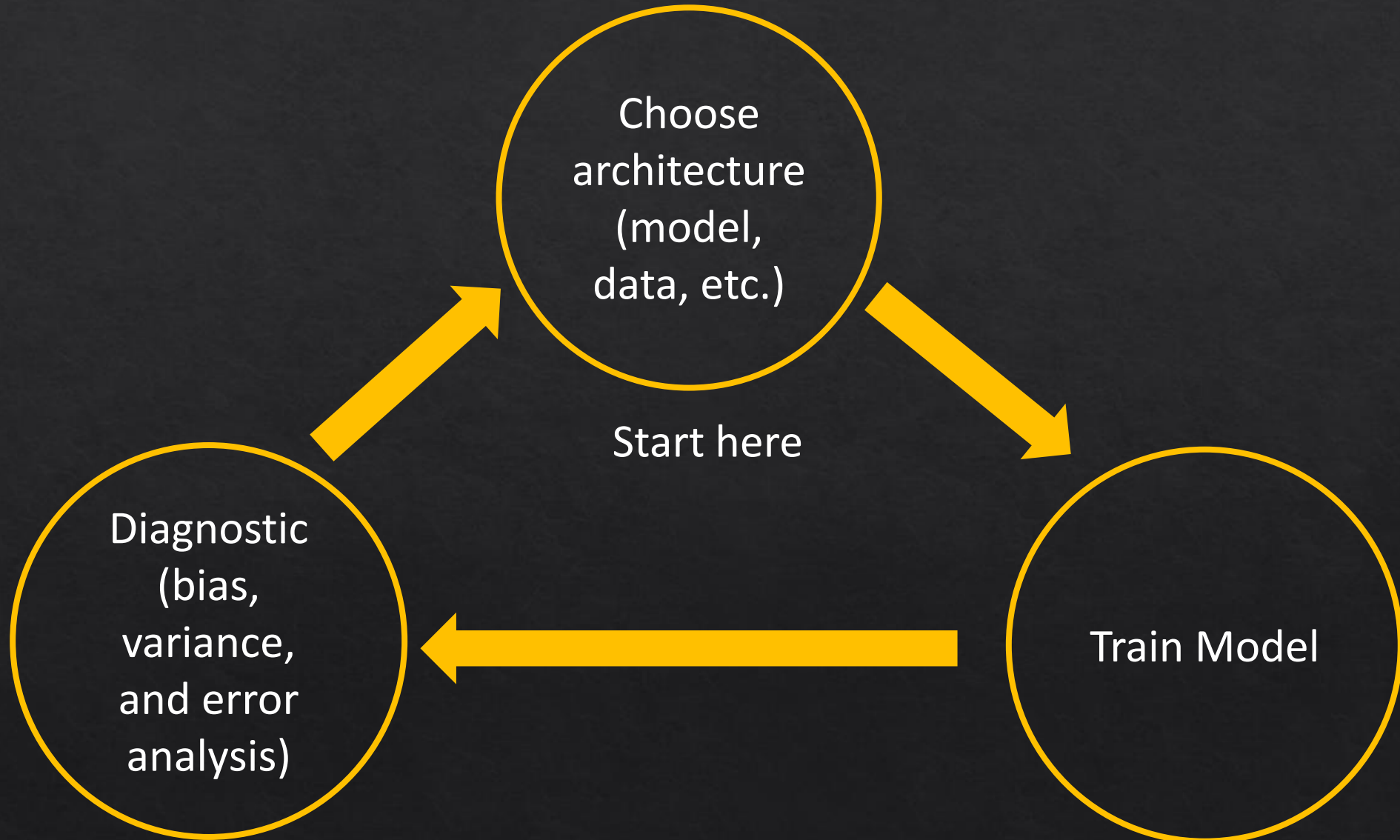
Fixes high variance

Neural Networks Bias / Variance

Neural Networks are low bias models.



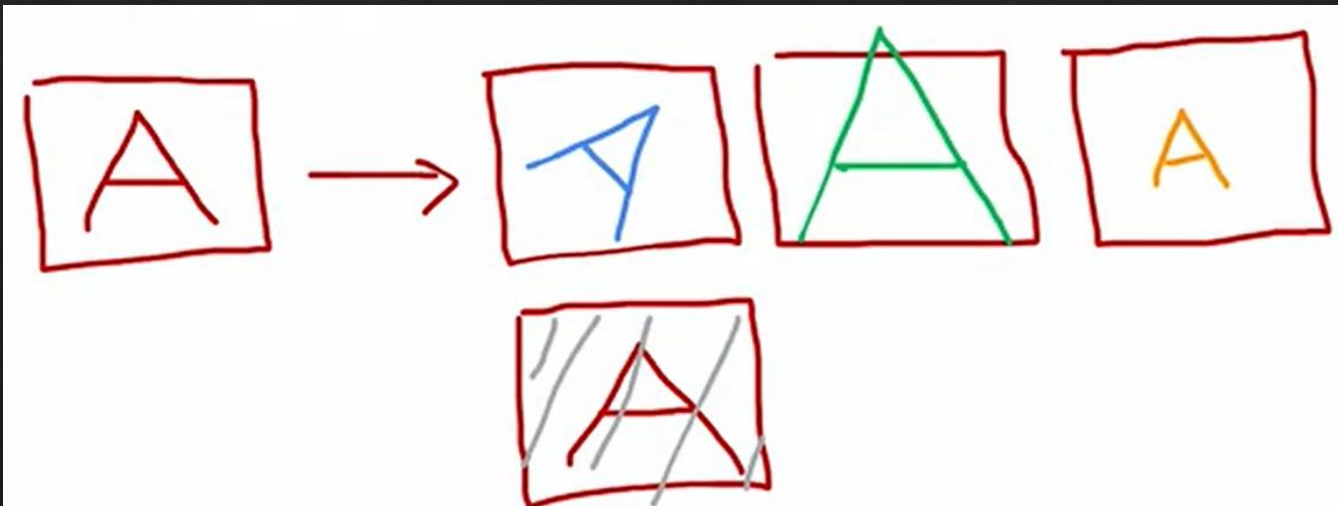
Iterative Loop Of ML Development



Data Augmentation

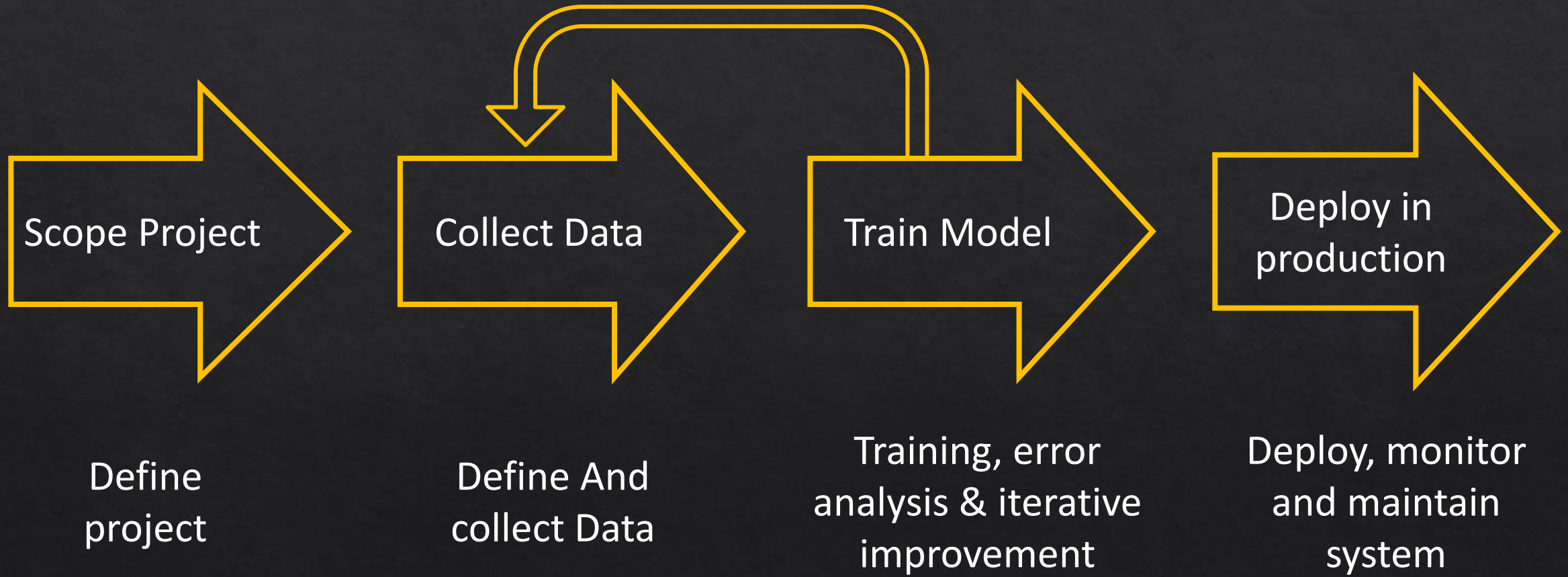
Augmentation: modifying existing training examples to create new training examples

This technique is used especially for **images** and **audio** data that can increase your training set size significantly



The distortion introduced should be a representation of the type of noise/distortion in the **test set**

Full Cycle Of Machine Learning project

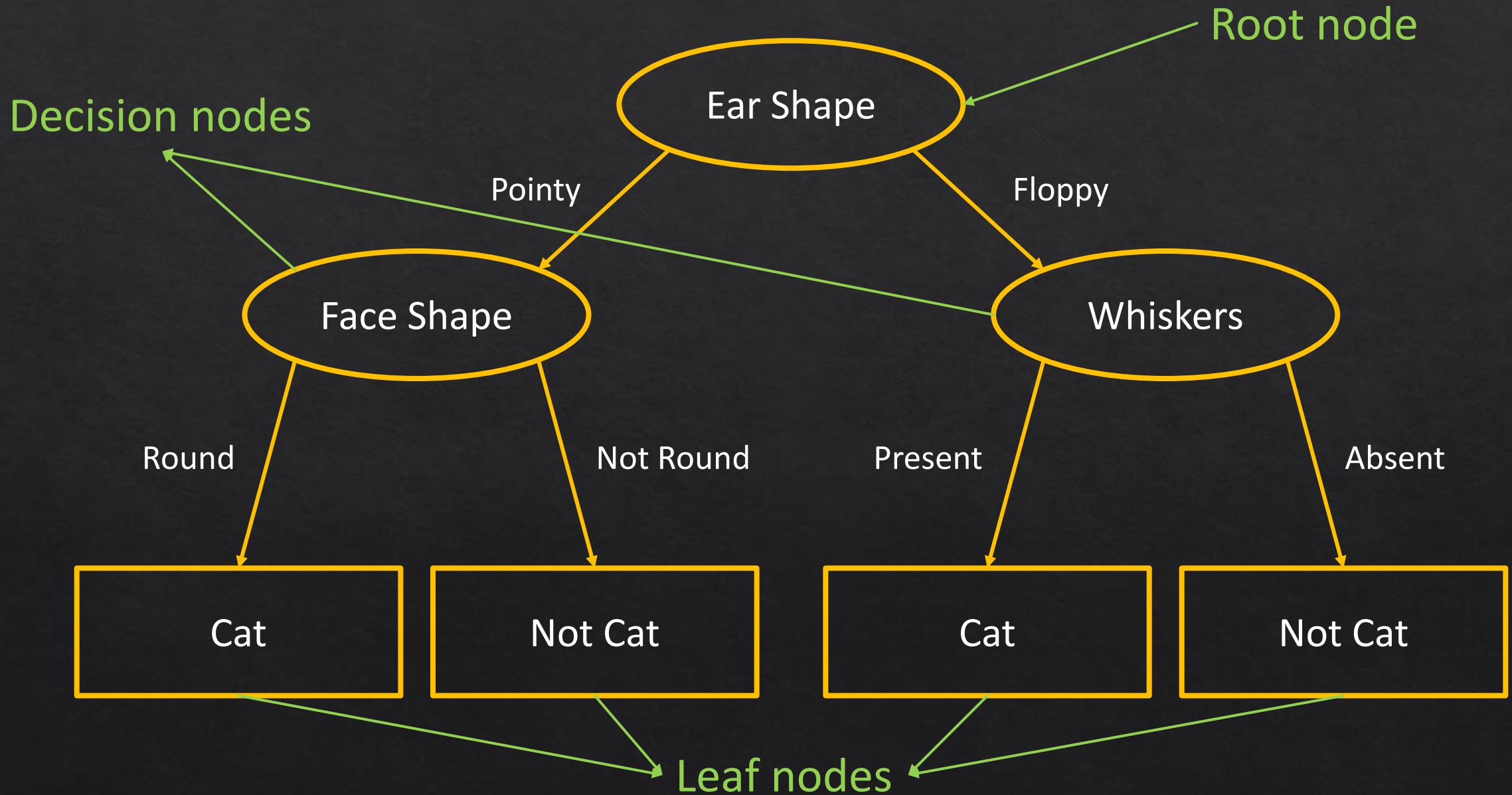


Decision Tree Model

Cat Classification Example

Ear Shape	Face Shape	Whiskers	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

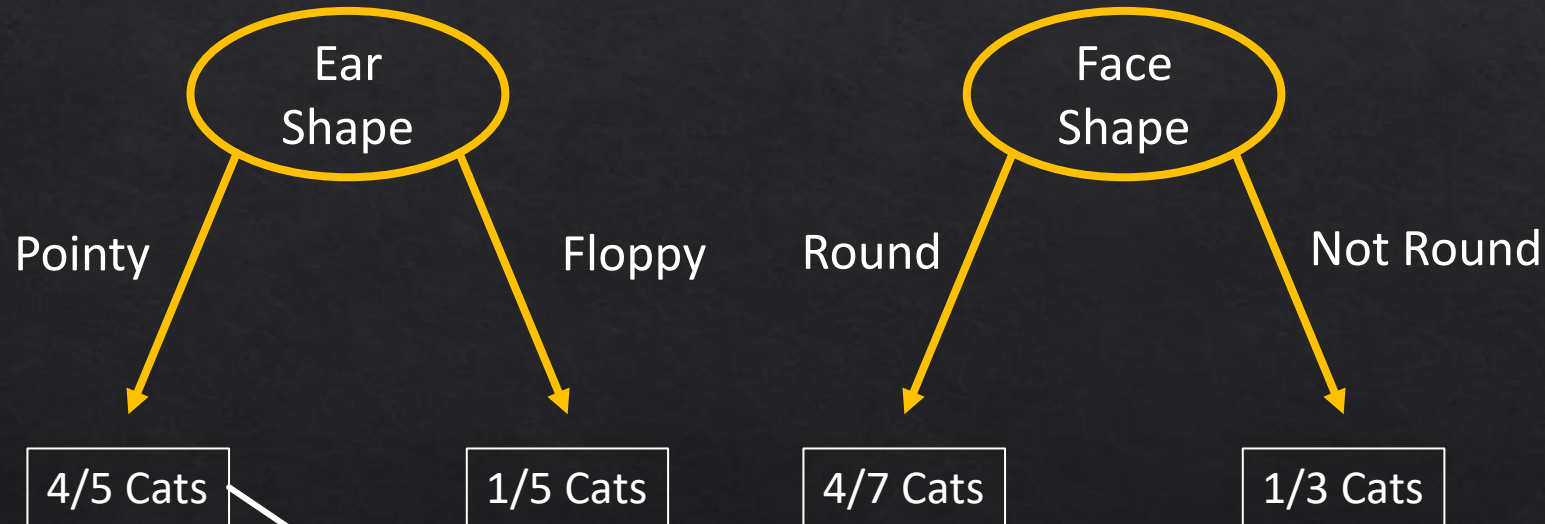
Cat Classification Example



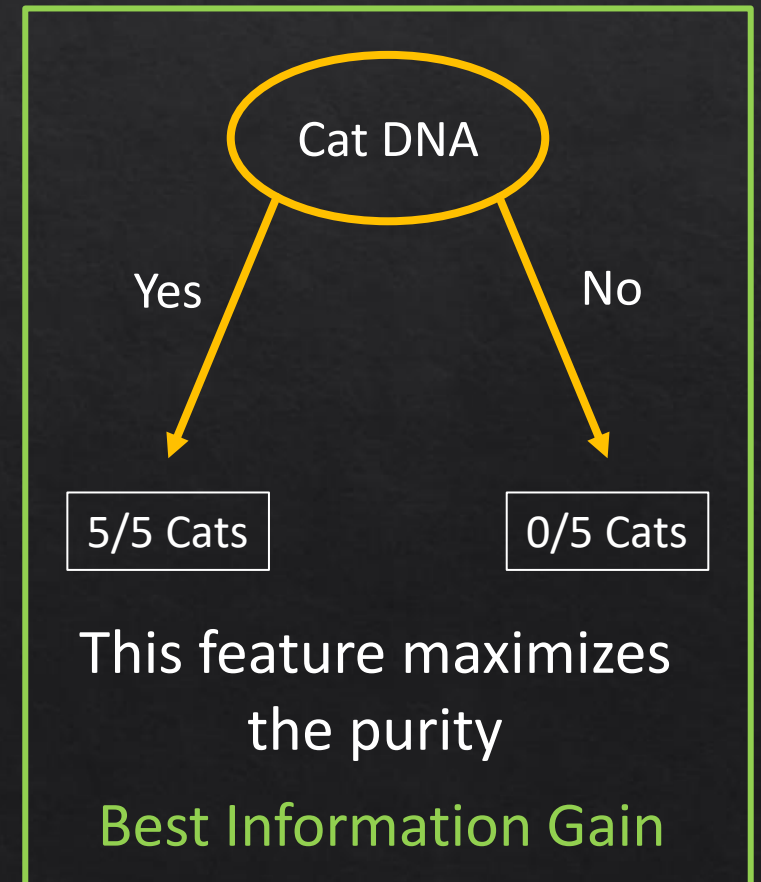
Decision Tree Learning

Decision1: How to choose what feature to split on at each node?

Maximize purity (or minimize impurity)



This means there are 5 examples with **Pointy** ear shapes, 4 of them are **Cats**

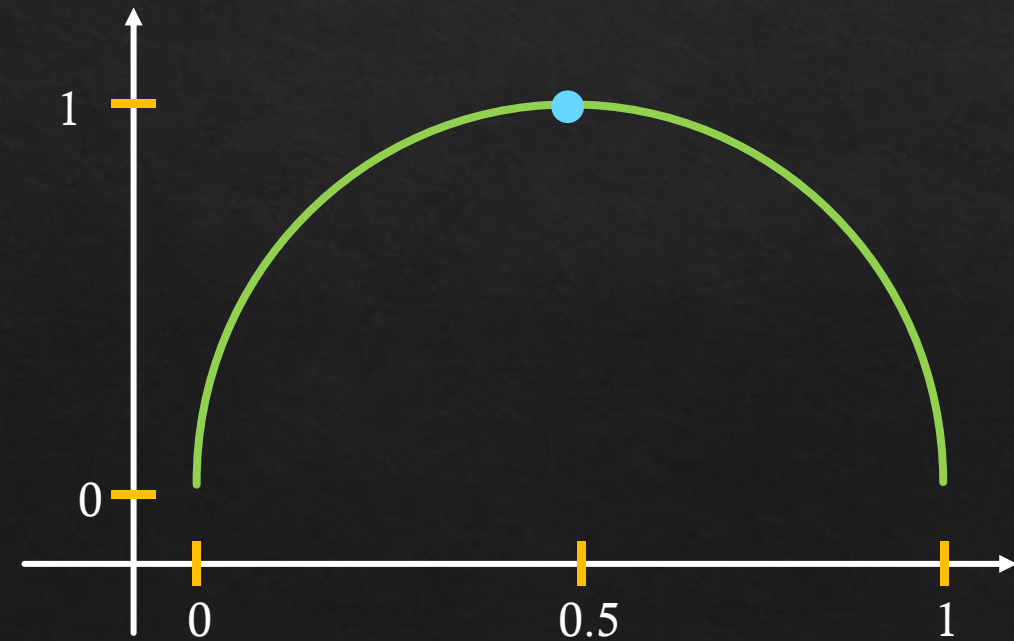


Decision Tree Learning

(But how to calculate the impurity?)

Entropy as a measure of Impurity:

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



$$p_1 = \frac{3}{6} \Rightarrow -0.5 \log_2(0.5) - (1 - 0.5) \log_2(1 - 0.5) = 1$$

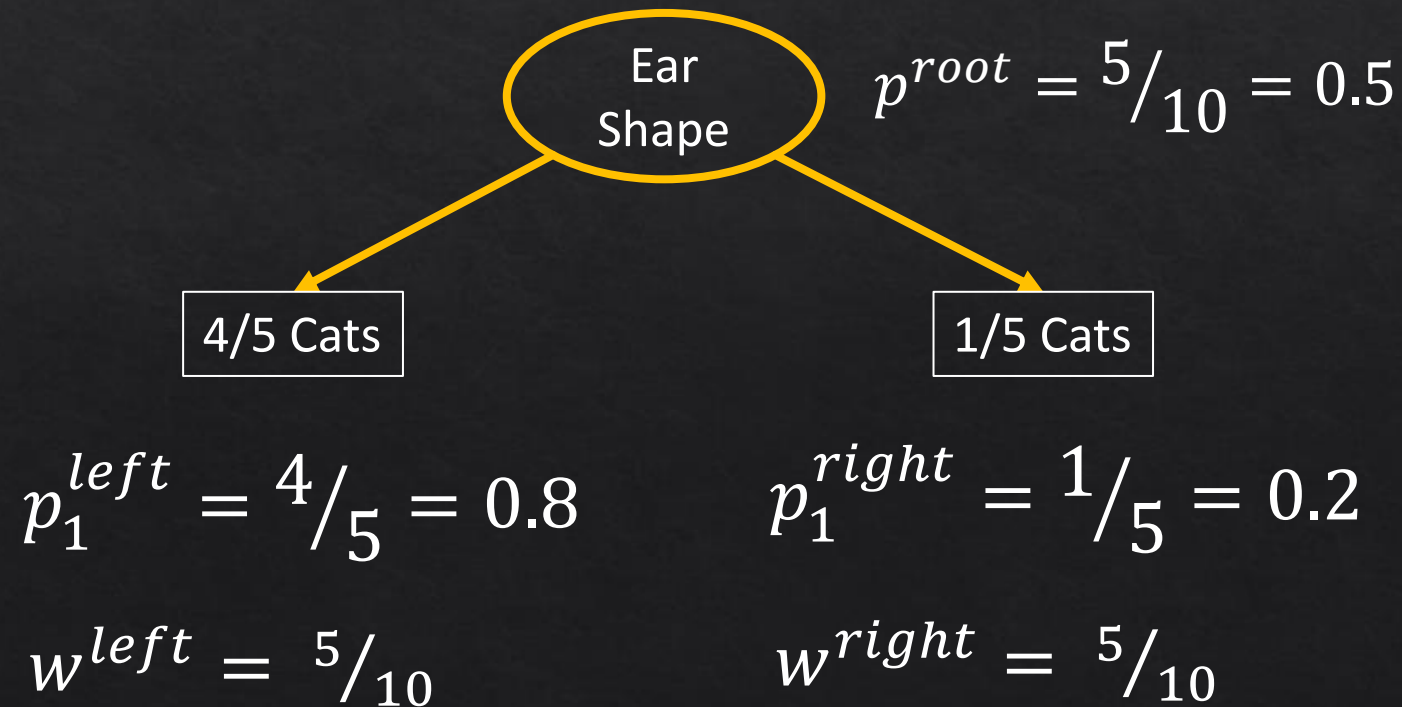
$$p_1 = \frac{5}{6} \Rightarrow -0.83 \log_2(0.83) - (1 - 0.83) \log_2(1 - 0.83) = 0.66$$

$$p_1 = \frac{6}{6} \Rightarrow -1 \log_2(1) - (1 - 1) \log_2(1 - 1) = 0$$

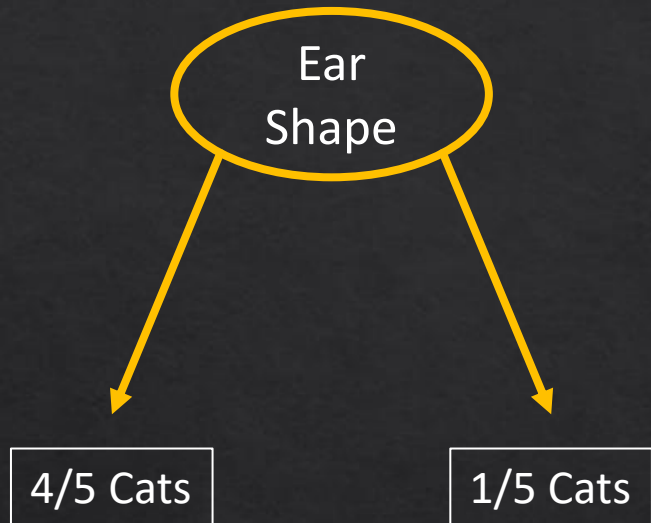
Information Gain

Information Gain: The reduction of entropy in compared to if we hadn't split at all

$$\text{Information Gain} = H(p_1^{\text{root}}) - (w^{\text{left}} H(p_1^{\text{left}}) + w^{\text{right}} H(p_1^{\text{right}}))$$



$$p_1 = 5/10 = 0.5 \quad H(0.5) = 1$$



$$p_1 = 4/5 = 0.8 \quad p_1 = 1/5 = 0.2$$

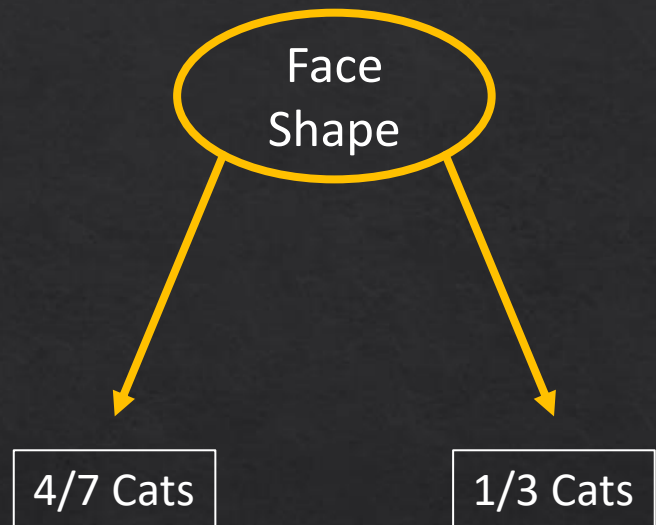
$$H(0.8) = 0.72 \quad H(0.2) = 0.72$$

$$H(0.5) - \left(\frac{5}{10} H(0.8) + \frac{5}{10} H(0.2) \right)$$

$$= 0.28$$

Best information gain

$$p_1 = 5/10 = 0.5 \quad H(0.5) = 1$$



$$p_1 = 4/7 = 0.57 \quad p_1 = 1/3 = 0.33$$

$$H(0.57) = 0.99 \quad H(0.33) = 0.92$$

$$H(0.5) - \left(\frac{7}{10} H(0.87) + \frac{3}{10} H(0.33) \right)$$

$$= 0.03$$

$$p_1 = 5/10 = 0.5 \quad H(0.5) = 1$$



$$p_1 = 3/4 = 0.75 \quad p_1 = 2/6 = 0.33$$

$$H(0.75) = 0.81 \quad H(0.33) = 0.92$$

$$H(0.5) - \left(\frac{4}{10} H(0.75) + \frac{6}{10} H(0.33) \right)$$

$$= 0.12$$

Decision Tree Learning

Decision2: 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 a number of examples in a node is below a threshold.

Decision Tree Learning

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

One Hot Encoding Of Categorical Feature

We saw that all features in the decision tree have only 2 classes (Ear shape: pointy or floppy
– Face Shape: Round Or Not Round - ...)

What if we have a feature that has more than 2 classes? We will apply One Hot Encoding

3 classes 2 classes 2 classes

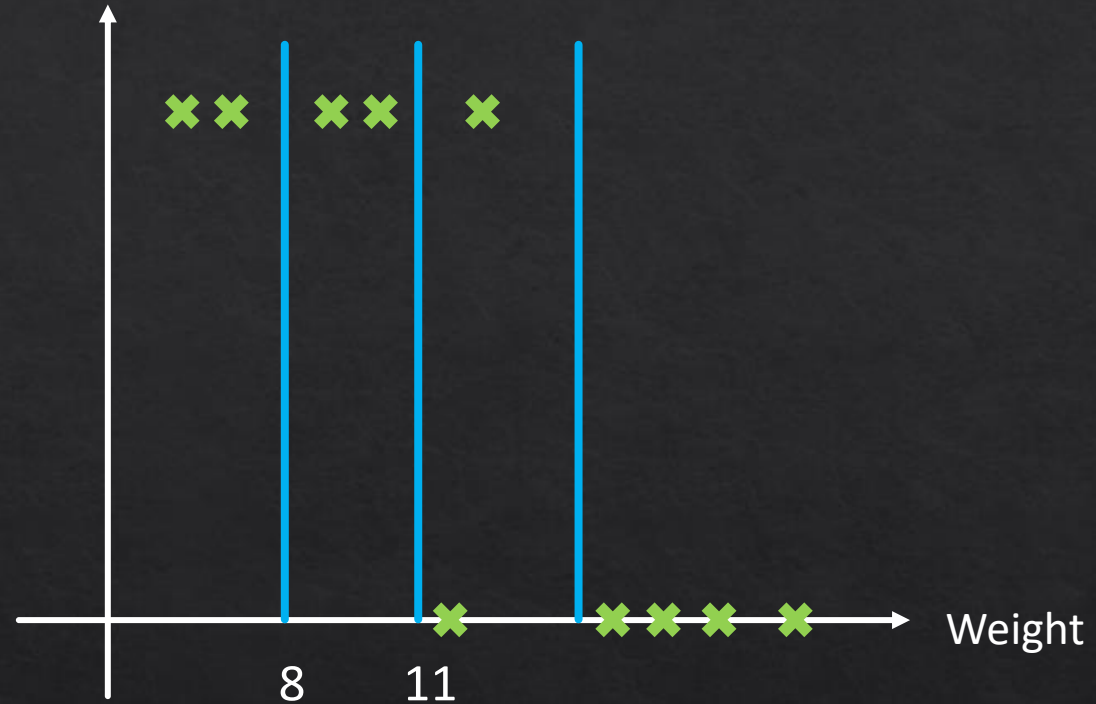
Ear	Face	Whisker	Is Cat
Pointy	Round	1	1
Floppy	Not Round	1	1
Round	Round	0	0
Pointy	Not Round	0	0



Ear			Face	Whisker	Is Cat
Pointy	Floppy	Round	Face	Whisker	Is Cat
1	0	0	Round	1	1
0	1	0	Not Round	1	1
0	0	1	Round	0	0
1	0	0	Not Round	0	0

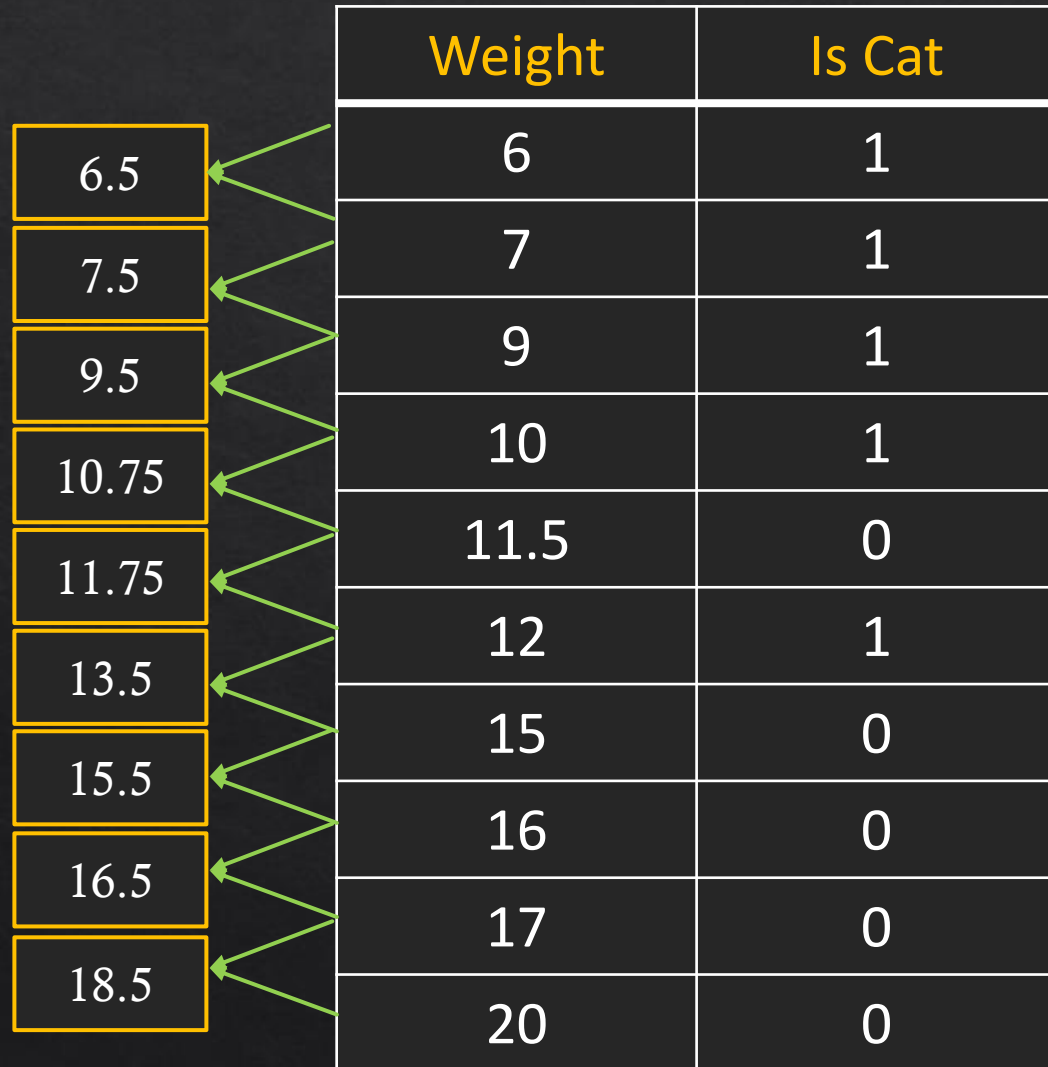
Continuous Feature

Weight	Is Cat
6	1
20	0
9	1
16	0
12	1
11.5	0
15	0
10	1
17	0
7	1



We will try many thresholds to split the range on the continuous feature, And take the threshold that **minimizes the impurity**

Continuous Feature



	Weight	Is Cat
6.5	6	1
7.5	7	1
9.5	9	1
10.75	10	1
11.75	11.5	0
13.5	12	1
15.5	15	0
16.5	16	0
18.5	17	0
	20	0

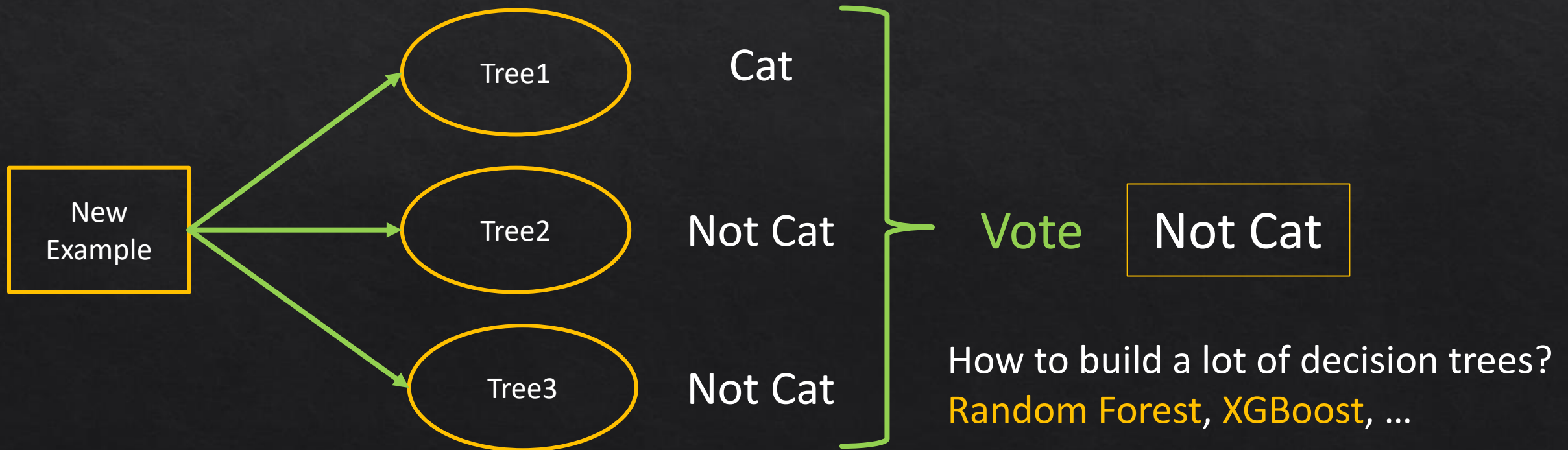
One way to do this is:

1. sort the values in **ascending order**
2. calculate the **averages** for every **adjacent values**
3. calculate the **impurity** if we select this average as a threshold
4. Set the average with the **minimum impurity** as a threshold

Multiple Decision Trees

One of the weaknesses of using a single decision tree is that maybe this decision tree is sensitive to small changes in the data. **What is the solution?**

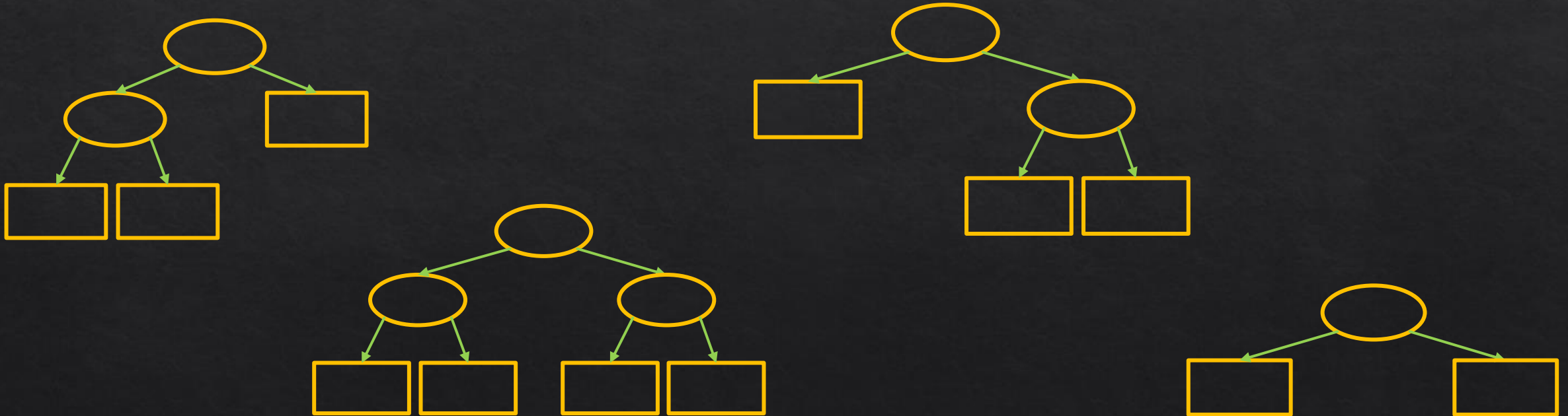
The solution is to build a lot of decision trees (tree ensemble)



Random Forest - XGBoost

Random Forest Algorithm

Random forest combines the simplicity of decision trees with flexibility resulting in a vast improvement in accuracy



These awesome Random Forest slides were taken from StateQuest Youtube Channel, The link is in the description.

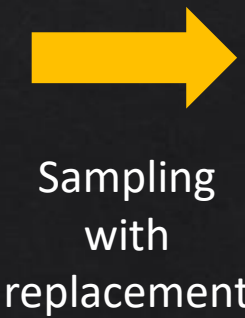
Step1: Create a “bootstrapped” dataset

To create a bootstrapped dataset that is the same size as the original, We randomly select samples from the original dataset.

The important detail is that we can pick the same sample more than once.

Original Dataset

Chest Pain	Good Blood Circ	Blocked Arteries	Weight	Heart Disease
0	0	0	125	0
1	1	1	180	1
1	1	0	210	0
1	0	1	167	1



Bootstrapped Dataset

Chest Pain	Good Blood Circ	Blocked Arteries	Weight	Heart Disease
1	1	1	180	1
0	0	0	125	0
1	0	1	167	1
1	0	1	167	1

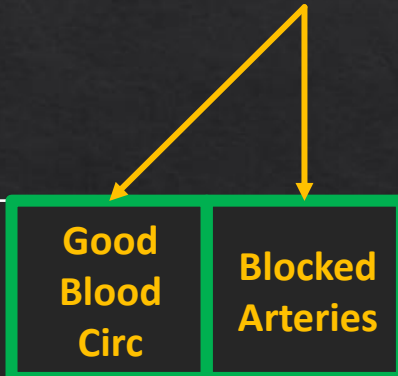
Duplicated

Step2: Create a decision tree using the bootstrapped dataset, but only use a **random subset** of variables (or columns) at each step (in this example we will only consider 2 variables at each step)

Choose the variable that did the best job separating the samples (assume that Good Blood Circ did)

that Good
Blood Circ

Randomly select 2 variables



Chest Pain	Good Blood Circ	Blocked Arteries	Weight	Heart Disease
1	1	1	180	1
0	0	0	125	0
1	0	1	167	1
1	0	1	167	1

Step2: Create a decision tree using the bootstrapped dataset, but only use a **random subset** of variables (or columns) at each step (in this example we will only consider 2 variables at each step)

Just like for the root, we randomly select 2 variables from the **remaining** features and choose the best.

that Good
Blood Circ

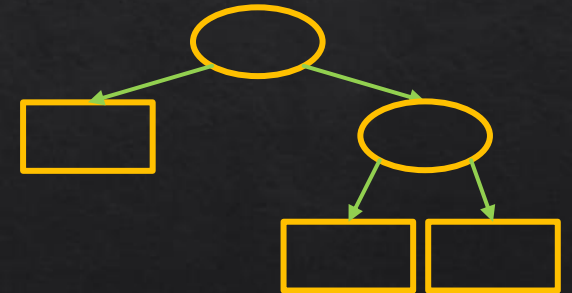
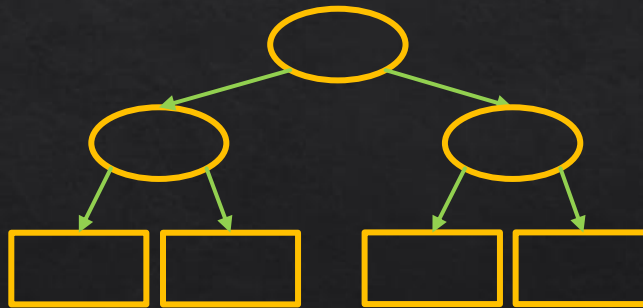
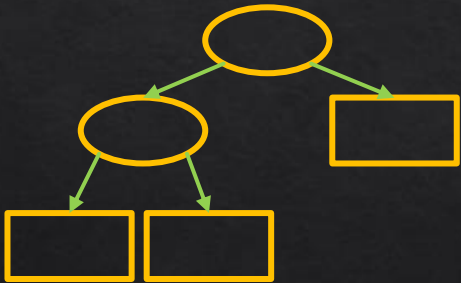
Randomly select 2 variables

Chest Pain	Good Blood Circ	Blocked Arteries	Weight	Heart Disease
1	1	1	180	1
0	0	0	125	0
1	0	1	167	1
1	0	1	167	1

We built a tree:

1. Using a bootstrapped dataset.
2. Only considering a random subset of variables at each step.

Now go back to Step1 and repeat: Make a new bootstrapped dataset and build a tree considering a subset of variables at each step.



Note: Bootstrapping the data plus using aggregate to make a decision is called **Bagging**

Note: After generating the tree, we vote for a new example and consider the **most votes**.

XGBoost

Given a training set of size m .

For $b=1$ to B :

1. 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 examples that the previously trained trees misclassify
2. Train a decision tree on the new dataset.

Decision Trees Vs Neural Networks

Decision trees and tree ensembles:

- Works well on tabular (structured) data
- Not recommended for unstructured data (images, audio, text)
- Fast
- Small decision trees may be human interpretable

Neural Networks:

- Works well on all types of data, including tabular (structured) and unstructured data.
- May be slower than decision trees.
- Works with transfer learning
- When building a system of multiple models working together, it might be easier to string together multiple neural networks



Oday Mourad

I am happy to share Data Science and Machine
Learning basics with my wonderful Linked In
friends, please follow me and let's learn together.