

[Part \(Week\) 1 : Neural Networks](#)

1

2.1 - Neural networks intuition

- 2.1.1 - Welcome!
- 2.1.2 - Neurons and the brain
- 2.1.3 - Demand Prediction
- 2.1.4 - Example: Recognizing Images

Ungraded App Item[IMPORTANT] Have questions, issues or ideas? Join our Community!

Practice quiz: Neural networks intuition

2.2 - Neural network model

- 2.2.1 - Neural network layer
 - 2.2.2 - More complex neural networks
 - 2.2.3 - Inference: making predictions (forward propagation)
- Lab: Neurons and Layers
- Practice quiz: Neural network model

2.3 - TensorFlow implementation

- 2.3.1 - Inference in Code
 - 2.3.2 - Data in TensorFlow
 - 2.3.3 - Building a neural network
- Lab: Coffee Roasting in Tensorflow
- Practice quiz: TensorFlow implementation

2.4 - Neural network implementation in Python

- 2.4.1 - Forward prop in a single layer
 - 2.4.2 - General implementation of forward propagation
- Lab: CoffeeRoastingNumPy
- Practice quiz: Neural network implementation in Python

2.5 - Speculations on artificial general intelligence (AGI)

Is there a path to AGI?

2.6 - Vectorization (optional)

- 2.6.1 - How neural networks are implemented efficiently
 - 2.6.2 - Matrix multiplication
 - 2.6.3 - Matrix multiplication rules
 - 2.6.4 - Matrix multiplication code
- Practice Lab: Neural networks

[Part \(Week\) 2 : Neural Network Training](#)

16

2.1 - Neural Network Training

- 2.1.1 - TensorFlow implementation. Duration: 3 minutes3 min
 - 2.1.2 - Training Details. Duration: 13 minutes13 min
- Practice quiz: Neural Network Training

2.2 - Activation Functions

- 2.2.1 - Alternatives to the sigmoid activation. Duration: 5 minutes5 min
 - 2.2.2 - Choosing activation functions. Duration: 8 minutes8 min
 - 2.2.3 - Why do we need activation functions?. Duration: 5 minutes5 min
- Lab: ReLU activation . Duration: 1 hour1h
- Practice quiz: Activation Functions

2.3 - Multiclass Classification

- 2.3.1 - Multiclass. Duration: 3 minutes3 min
 - 2.3.2 - Softmax. Duration: 11 minutes11 min
 - 2.3.3 - Neural Network with Softmax output . Duration: 7 minutes7 min
 - 2.3.4 - Improved implementation of softmax. Duration: 9 minutes9 min
 - 2.3.5 - Classification with multiple outputs (Optional). Duration: 4 minutes4 min
- Lab: Softmax. Duration: 1 hour1h
- Lab: Multiclass. Duration: 15 minutes15 min
- Practice quiz: Multiclass Classification

2.4 - Additional Neural Network Concepts

- 2.4.1 - Advanced Optimization. Duration: 6 minutes6 min
 - 2.4.2 - Additional Layer Types. Duration: 8 minutes8 min
- Practice quiz: Additional Neural Network Concepts

2.5 - Back Propagation (Optional)

- 2.5.1 - What is a derivative? (Optional). Duration: 22 minutes22 min
- 2.5.2 - Computation graph (Optional). Duration: 19 minutes19 min

2.5.3 - Larger neural network example (Optional). Duration: 9 minutes9 min

Lab: Optional Lab: Derivatives. Duration: 30 minutes30 min

Lab: Optional Lab: Back propagation. Duration: 30 minutes30 min

Practice Lab: Neural network training

Part (Week) 3 : Machine Learning

34

3.1 - Advice for applying machine learning

3.1.1 - Deciding what to try next. Duration: 3 minutes3 min

3.1.2 - Evaluating a model. Duration: 10 minutes10 min

3.1.3 - Model selection and training/cross validation/test sets. Duration: 13 minutes13 min

Lab: Optional Lab: Model Evaluation and Selection. Duration: 30 minutes30 min

Practice quiz: Advice for applying machine learning

3.2 - Bias and variance

3.2.1 - Diagnosing bias and variance. Duration: 11 minutes11 min

3.2.2 - Regularization and bias/variance. Duration: 10 minutes10 min

3.2.3 - Establishing a baseline level of performance. Duration: 9 minutes9 min

3.2.4 - Learning curves. Duration: 11 minutes11 min

3.2.5 - Deciding what to try next revisited. Duration: 8 minutes8 min

3.2.6 - Bias/variance and neural networks. Duration: 10 minutes10 min

Lab: Optional Lab: Diagnosing Bias and Variance. Duration: 30 minutes30 min

Practice quiz: Bias and variance

3.3 - Machine learning development process

3.3.1 - Iterative loop of ML development. Duration: 7 minutes7 min

3.3.2 - Error analysis. Duration: 8 minutes8 min

3.3.3 - Adding data. Duration: 14 minutes14 min

3.3.4 - Transfer learning: using data from a different task. Duration: 11 minutes11 min

3.3.5 - Full cycle of a machine learning project. Duration: 8 minutes8 min

3.3.6 - Fairness, bias, and ethics. Duration: 9 minutes9 min

Practice quiz: Machine learning development process

3.4 - Skewed datasets (optional)

3.4.1 - Error metrics for skewed datasets. Duration: 11 minutes11 min

3.4.2 - Trading off precision and recall. Duration: 11 minutes11 min

Practice Lab: Advice for applying machine learning

Part (Week) 4 : Decision trees

56

4.1 - Decision trees

4.1.1 - Decision tree model. Duration: 7 minutes7 min

4.1.2 - Learning Process. Duration: 11 minutes11 min

Practice quiz: Decision trees

4.2 - Decision tree learning

4.2.1 - Measuring purity. Duration: 7 minutes7 min

4.2.2 - Choosing a split: Information Gain. Duration: 11 minutes11 min

4.2.3 - Putting it together. Duration: 9 minutes9 min

4.2.4 - Using one-hot encoding of categorical features. Duration: 5 minutes5 min

4.2.5 - Continuous valued features. Duration: 6 minutes6 min

4.2.6 - Regression Trees (optional). Duration: 9 minutes9 min

Lab: Optional Lab: Decision Trees. Duration: 30 minutes30 min

Practice quiz: Decision tree learning

4.3 - Tree ensembles

4.3.1 - Using multiple decision trees. Duration: 3 minutes3 min

4.3.2 - Sampling with replacement. Duration: 3 minutes3 min

4.3.3 - Random forest algorithm. Duration: 6 minutes6 min

4.3.4 - XGBoost. Duration: 6 minutes6 min

4.3.5 - When to use decision trees. Duration: 6 minutes6 min

Lab: Optional Lab: Tree Ensembles. Duration: 30 minutes30 min

Practice quiz: Tree ensembles

Practice Lab: Decision Trees

4.4 - Conversations with Andrew (Optional)

Video: VideoAndrew Ng and Chris Manning on Natural Language Processing. Duration: 47 minutes47 min

Part (Week) 1 : Neural Networks

2.1 - Neural networks intuition

2.1.1 - Welcome!

2.1.2 - Neurons and the brain

2.1.3 - Demand Prediction

2.1.4 - Example: Recognizing Images

Ungraded App Item[IMPORTANT] Have questions, issues or ideas? Join our Community!

Practice quiz: Neural networks intuition

2.2 - Neural network model

2.2.1 - Neural network layer

2.2.2 - More complex neural networks

2.2.3 - Inference: making predictions (forward propagation)

Lab: Neurons and Layers

Practice quiz: Neural network model

2.3 - TensorFlow implementation

2.3.1 - Inference in Code

2.3.2 - Data in TensorFlow

2.3.3 - Building a neural network

Lab: Coffee Roasting in Tensorflow

Practice quiz: TensorFlow implementation

2.4 - Neural network implementation in Python

2.4.1 - Forward prop in a single layer

2.4.2 - General implementation of forward propagation

Lab: CoffeeRoastingNumPy

Practice quiz: Neural network implementation in Python

2.5 - Speculations on artificial general intelligence (AGI)

Is there a path to AGI?

2.6 - Vectorization (optional)

2.6.1 - How neural networks are implemented efficiently

2.6.2 - Matrix multiplication

2.6.3 - Matrix multiplication rules

2.6.4 - Matrix multiplication code

Practice Lab: Neural networks

2.1 - Neural networks intuition : 2.1.1 - Welcome :

Advanced learning algorithms

Neural Networks 

inference (prediction)

training

Practical advice for building machine learning systems Decision Trees 

2.1 - Neural networks intuition : 2.1.2 - Neurons and the brain :

Neural networks

Origins: Algorithms that try to mimic the brain.

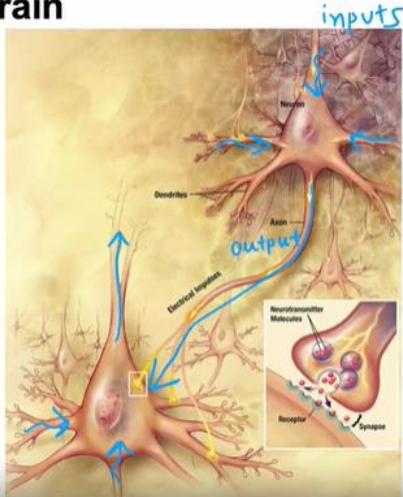


Used in the 1980's and early 1990's.
Fell out of favor in the late 1990's.

Resurgence from around 2005.

speech → images → text (NLP) → ...

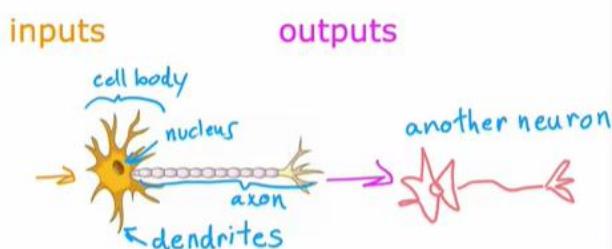
Neurons in the brain



4:01 / 10:51 National Institutes of Health, National Institute on Aging]



Biological neuron



Simplified mathematical model of a neuron

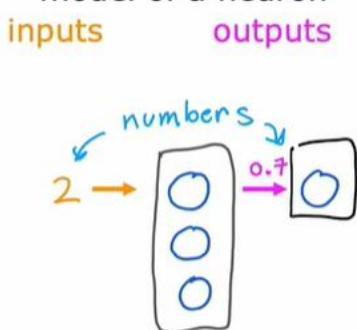
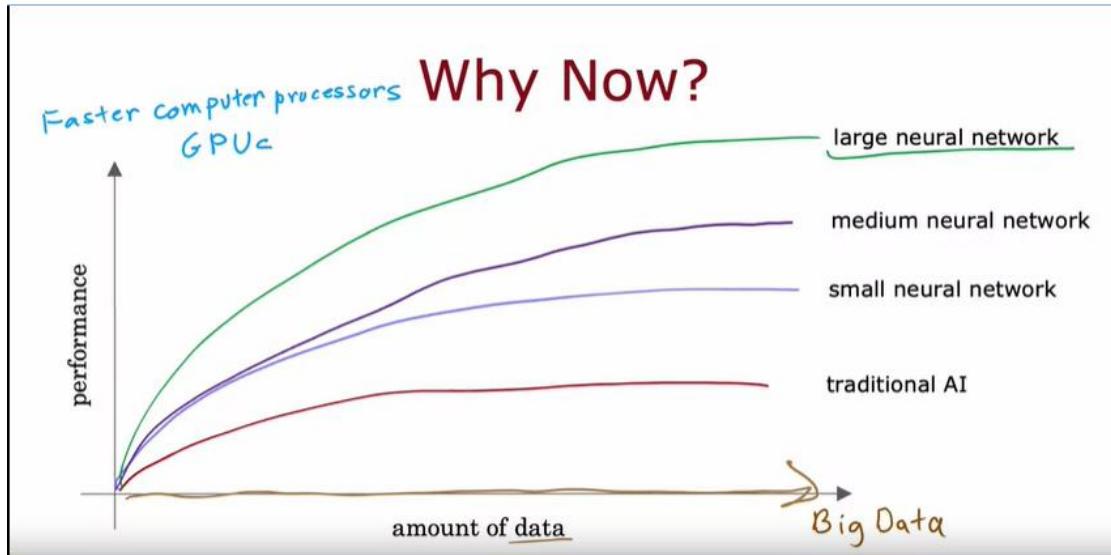
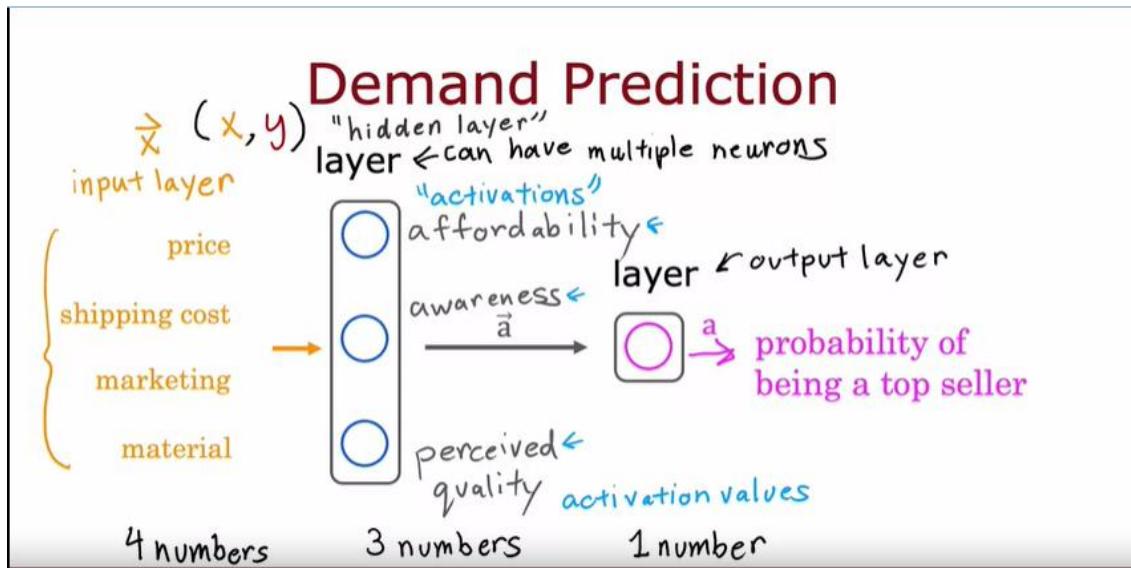
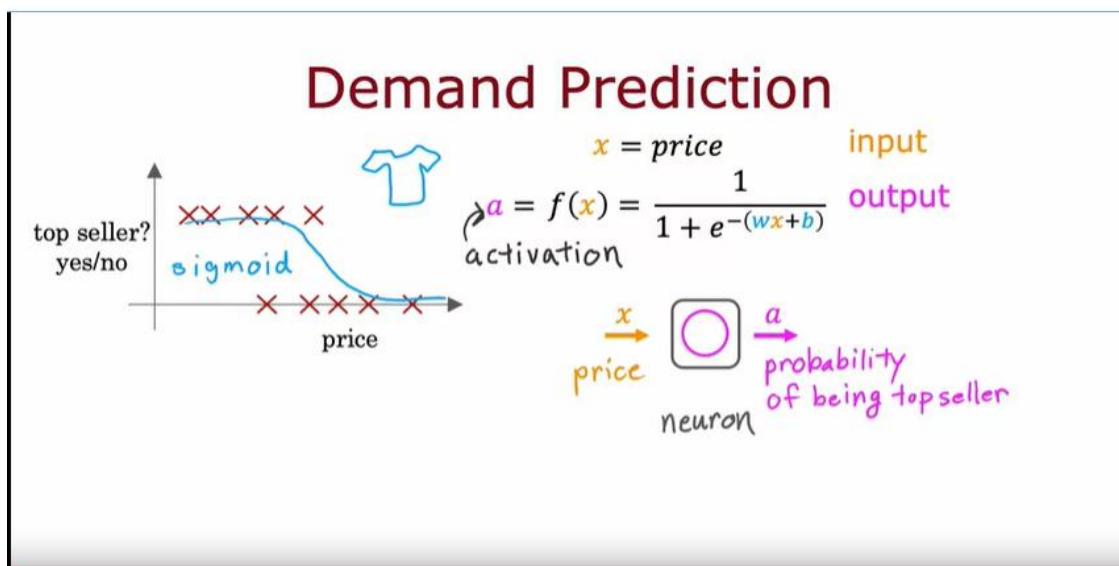
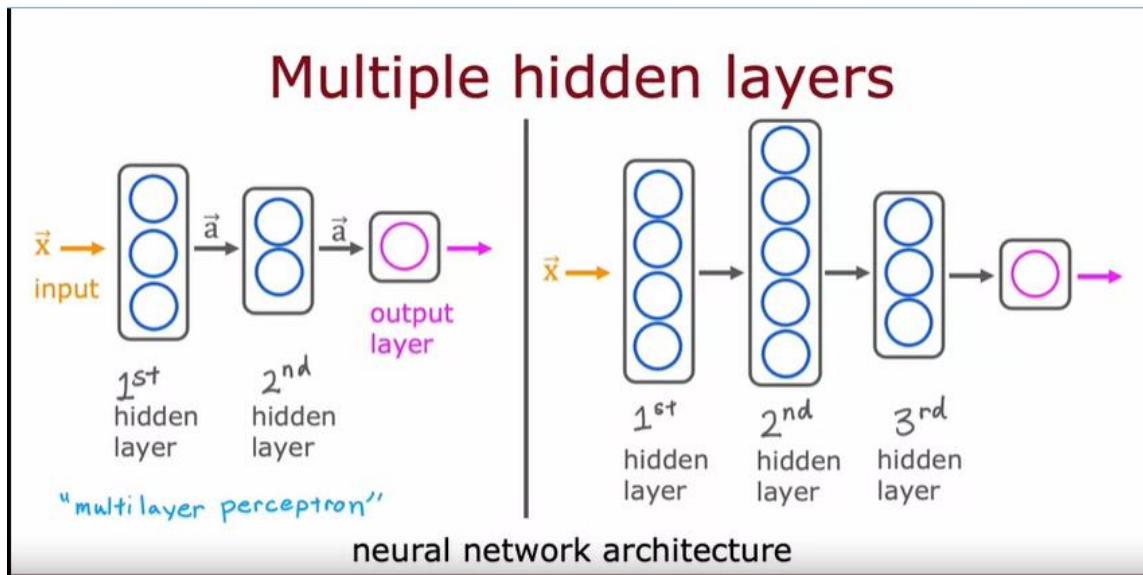
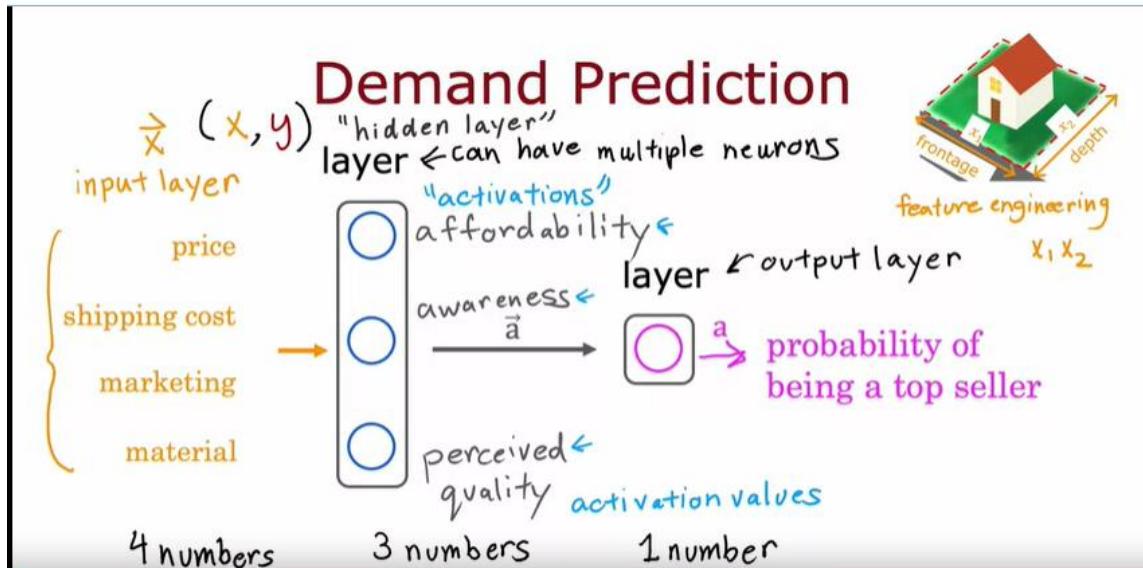


image source: <https://biologydictionary.net/sensory-neuron/>



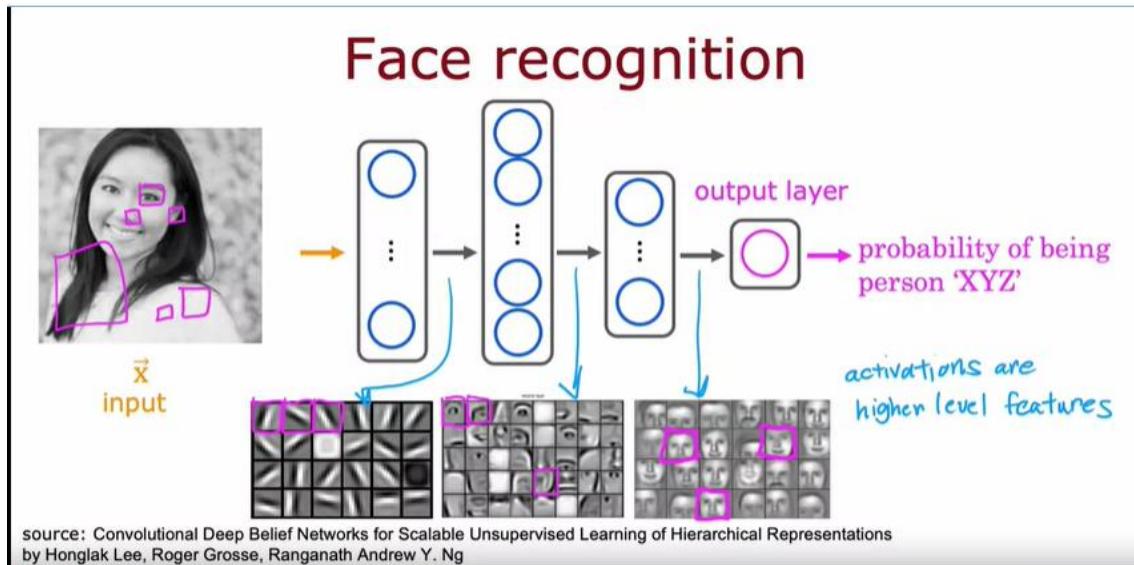
2.1 - Neural networks intuition : 2.1.3 - Demand Prediction :

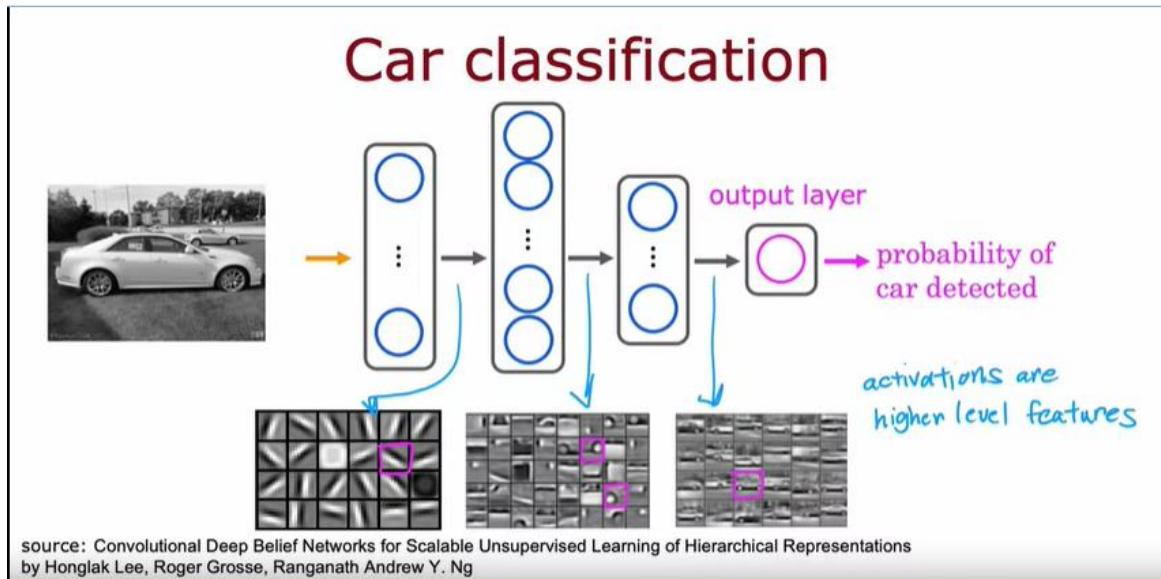




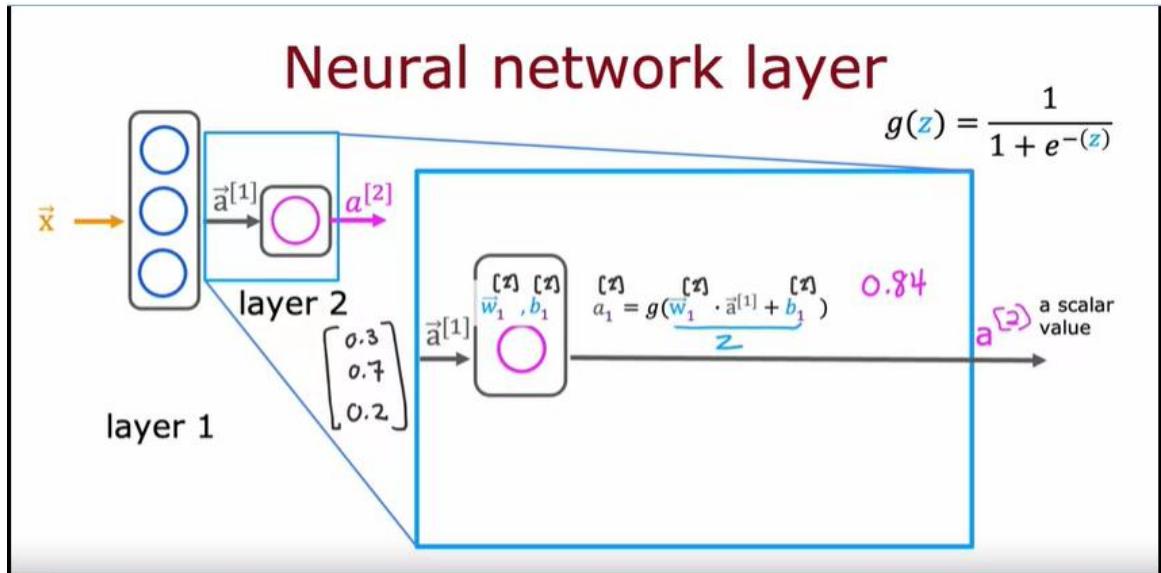
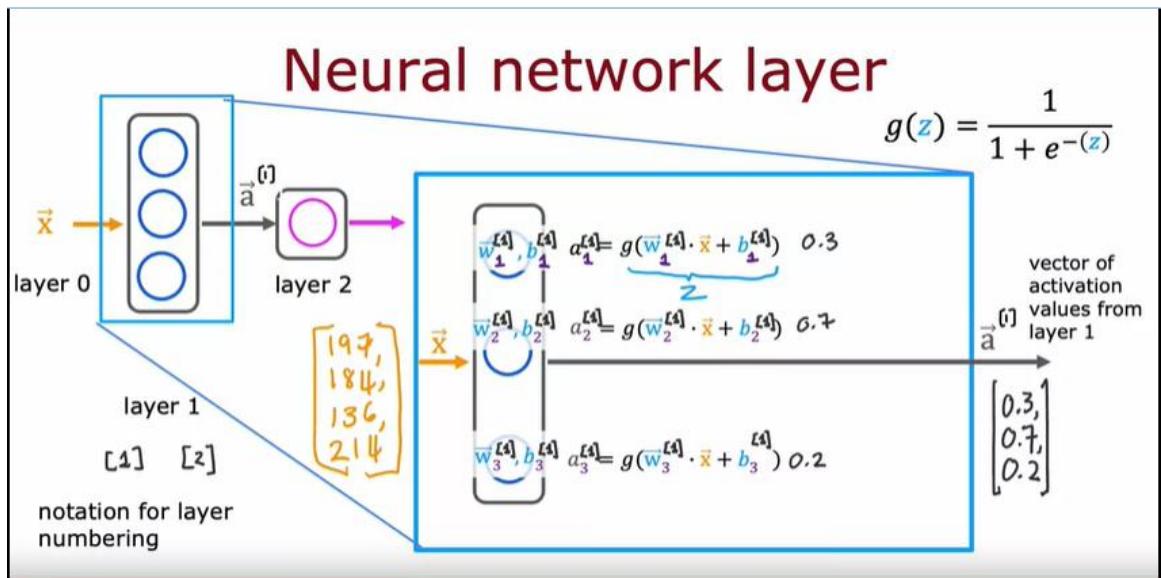
2.1 - Neural networks intuition : 2.1.4 - Example: Recognizing Images :

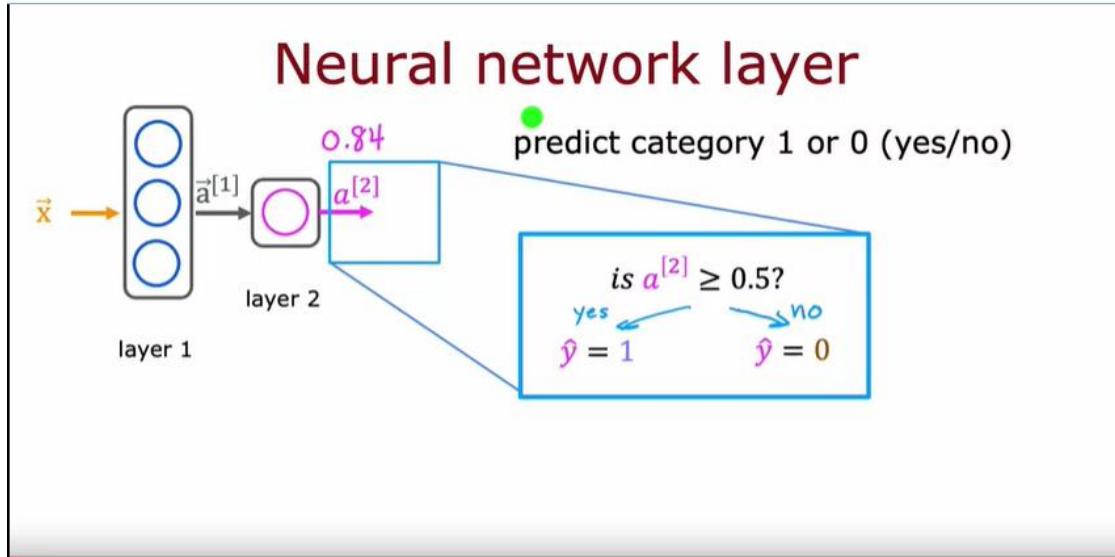
Pixel intensity --> 0 - 255



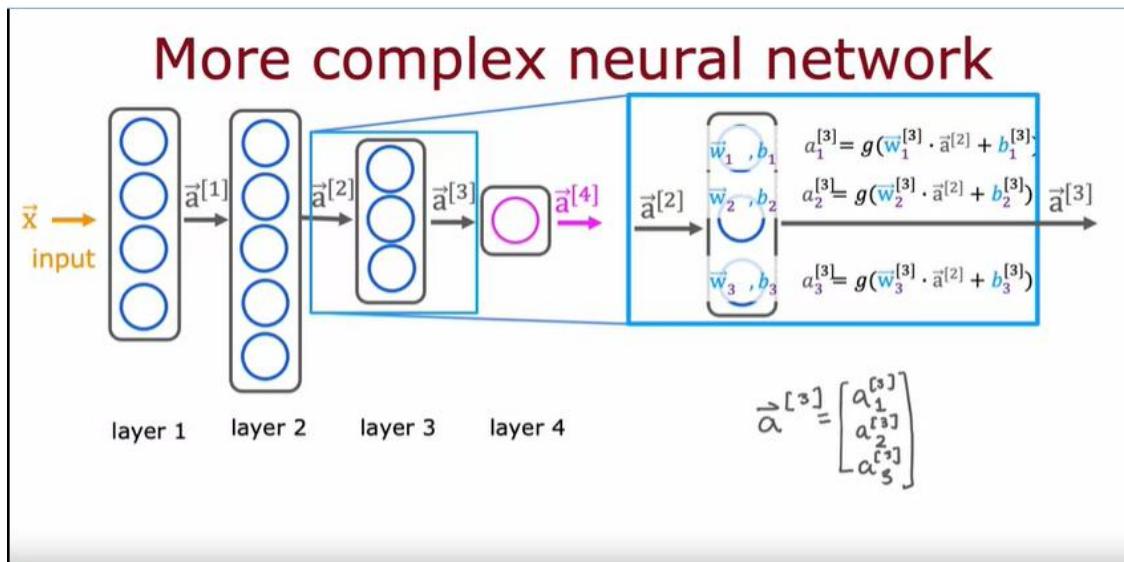
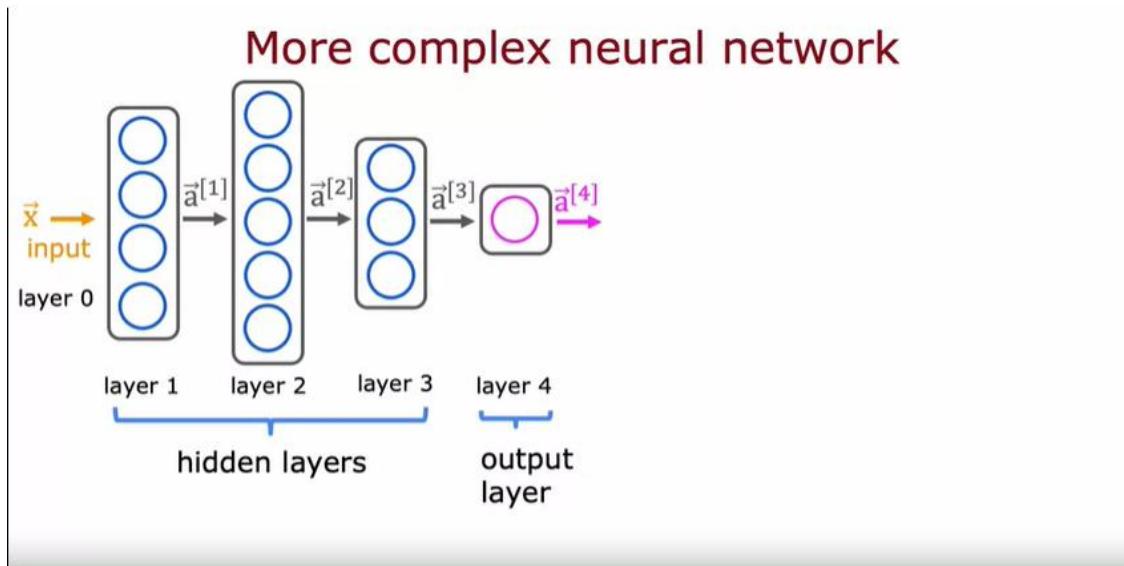


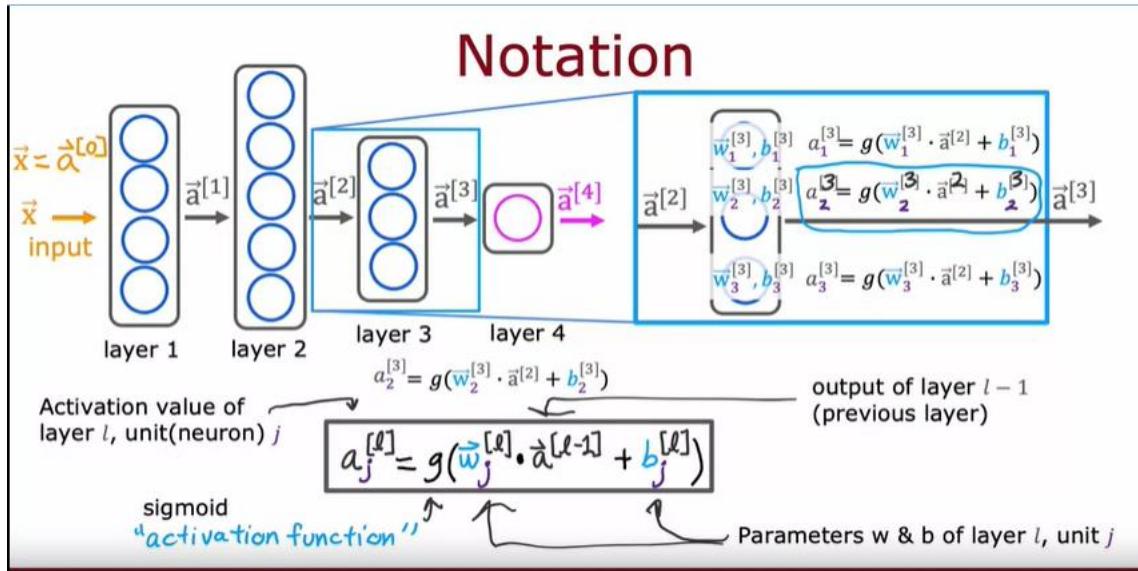
2.2 - Neural network model : 2.2.1 - Neural network layer :



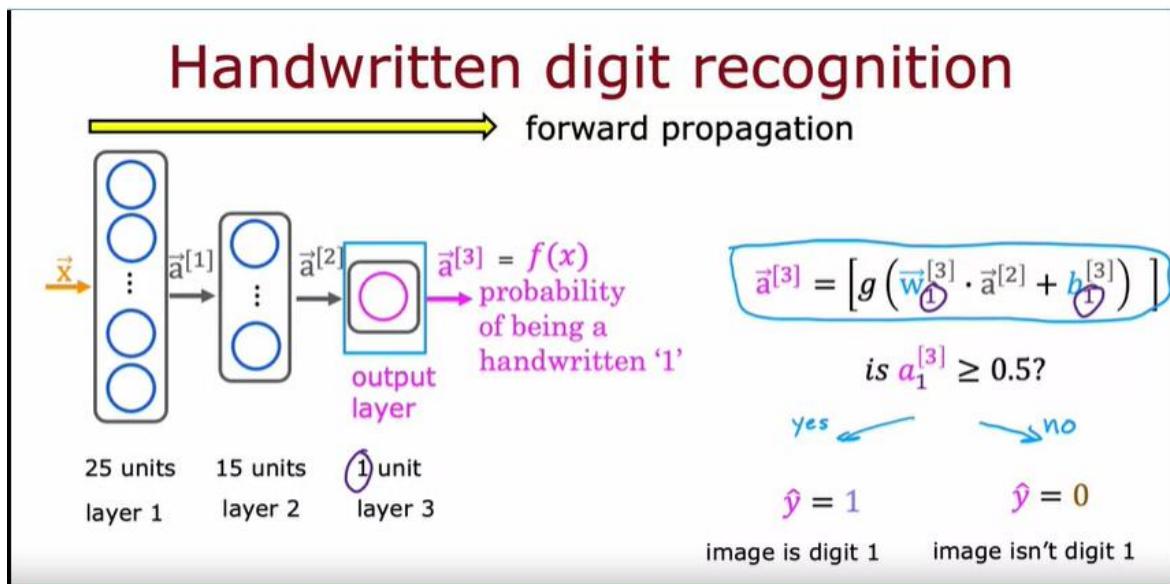
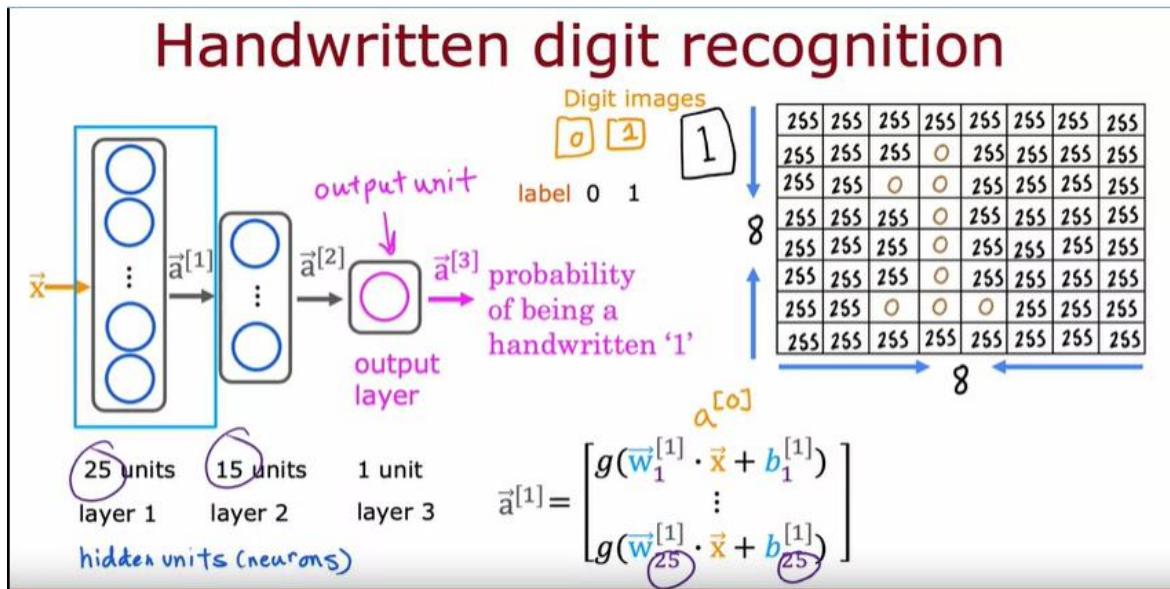


2.2 - Neural network model : 2.2.2 - More complex neural networks :



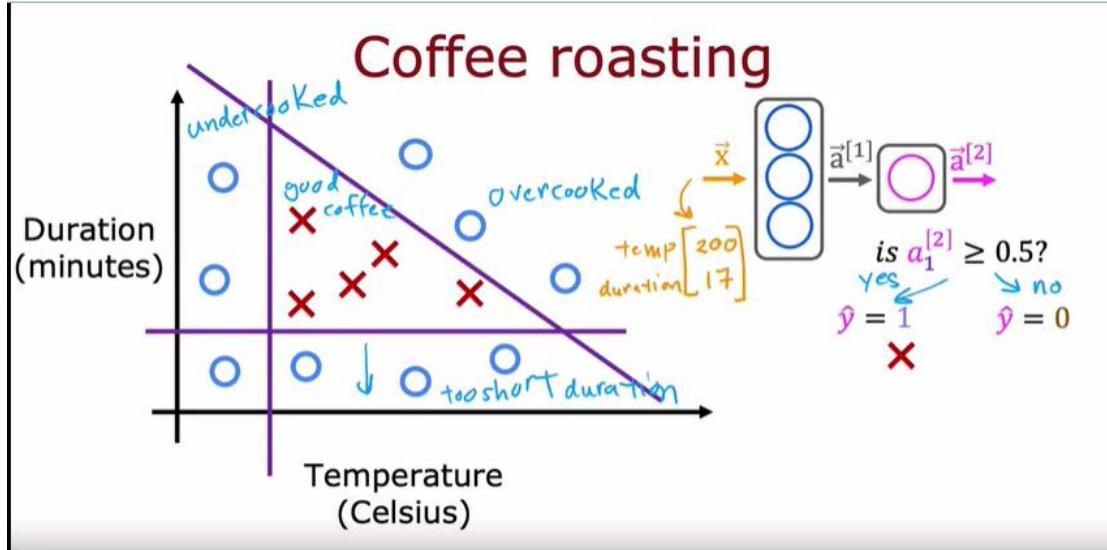


2.2 - Neural network model : 2.2.3 - Inference: making predictions (forward propagation) :



2.3 - TensorFlow implementation : 2.3.1 - Inference in Code :

- TensorFlow and PyTorch are the leading frameworks to implementing deep learning algorithms.
- One of the remarkable things about neural networks is the same algorithm can be applied to so many different applications.



Build the model using TensorFlow

Model for digit classification

2.3 - TensorFlow implementation : 2.3.2 - Data in TensorFlow :

Feature vectors

temperature (Celsius)	duration (minutes)	Good coffee? (1/0)
200.0	17.0	1
425.0	18.5	0
...

$x = \text{np.array}([[200.0, 17.0]])$ ←
 ↗
 [[200.0, 17.0]]
 why?

Note about numpy arrays

$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$x = \text{np.array}([[1, 2, 3], [4, 5, 6]])$	2D array
2 rows	$[[1, 2, 3], [4, 5, 6]]$	2×3
$\begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -0.5 & -0.6 \\ 7 & 8 \end{bmatrix}$	$x = \text{np.array}([[0.1, 0.2], [-3.0, -4.0], [-0.5, -0.6], [7.0, 8.0]])$	4×2
4 rows	$[[0.1, 0.2], [-3.0, -4.0], [-0.5, -0.6], [7.0, 8.0]]$	1×2
2 columns		2×1
$\begin{bmatrix} 200 & 17 \end{bmatrix}$		

Note about numpy arrays

$x = \text{np.array}([[200, 17]])$ → [200 17] 1×2

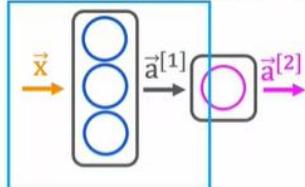
$x = \text{np.array}([[200], [17]])$ → [200]
 ↓
 [17] 2×1

→ $x = \text{np.array}([200, 17])$
 ↑
 "Vector"

what is the tensor?

A tensor here is a data type that the TensorFlow team had created in order to store and carry out computations on matrices efficiently.

Activation vector

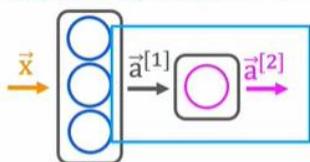


```

x = np.array([[200.0, 17.0]])
layer_1 = Dense(units=3, activation='sigmoid')
a1 = layer_1(x)
→ [0.2, 0.7, 0.3] 1 x 3 matrix
→ tf.Tensor([[0.2 0.7 0.3]], shape=(1, 3), dtype=float32)
→ a1.numpy()
array([[0.2, 0.7, 0.3]], dtype=float32)

```

Activation vector



```

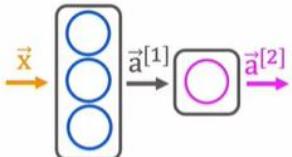
→ layer_2 = Dense(units=1, activation='sigmoid')
→ a2 = layer_2(a1)
→ [[0.8]] ← 1 x 1
→ tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
→ a2.numpy()
→ array([[0.8]], dtype=float32)

```

When you pass a NumPy array into TensorFlow, TensorFlow likes to convert it to its own internal format : The tensor and then operate efficiently using tensors. And when you read the data back out you can keep it as a tensor or convert it back to a NumPy array.

2.3 - TensorFlow implementation : 2.3.3 - Building a neural network :

What you saw earlier



```

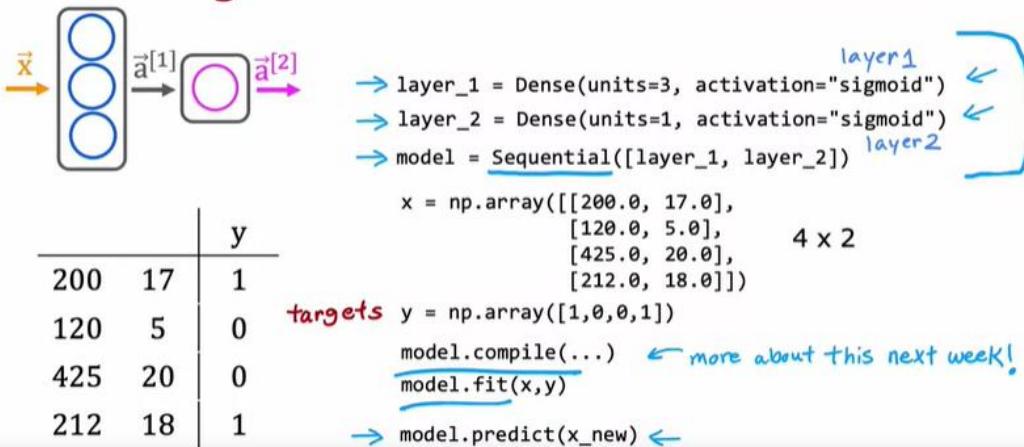
→ x = np.array([[200.0, 17.0]])
→ layer_1 = Dense(units=3, activation="sigmoid")
→ a1 = layer_1(x)

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

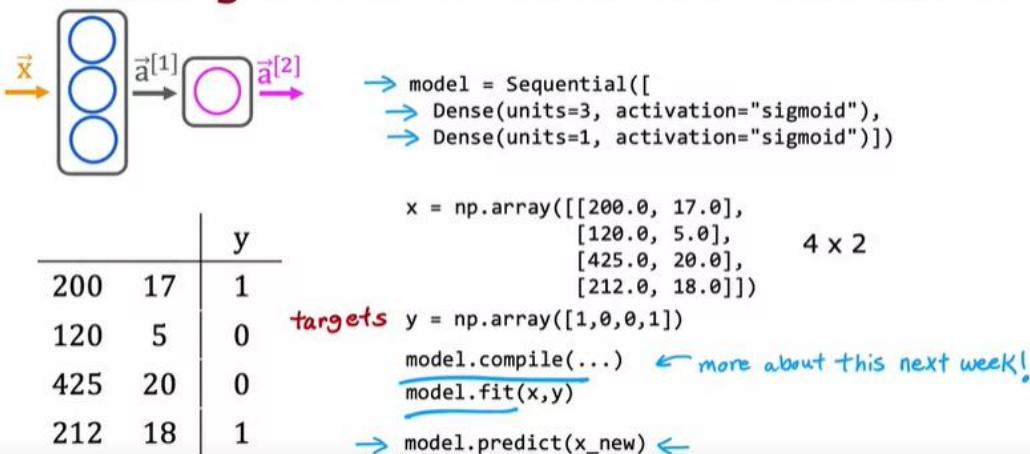
```

prop one layer of
computation at the time.

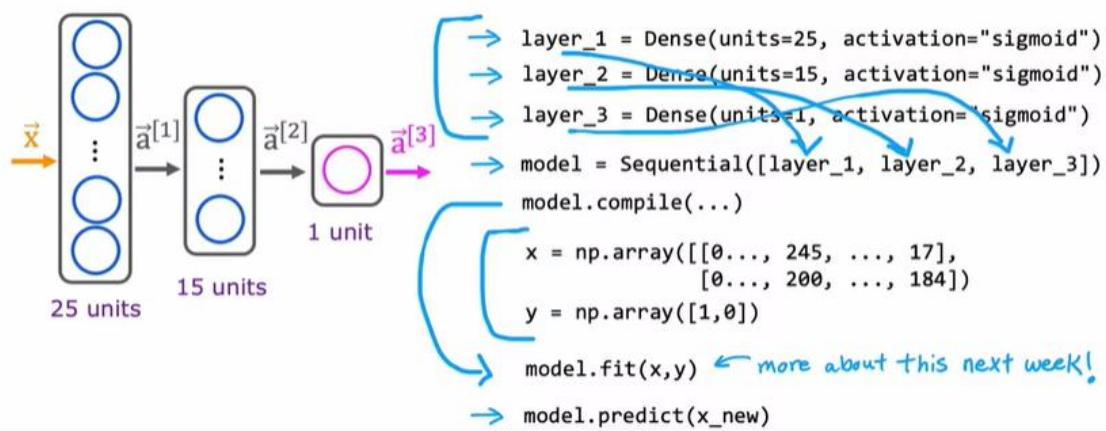
Building a neural network architecture



Building a neural network architecture

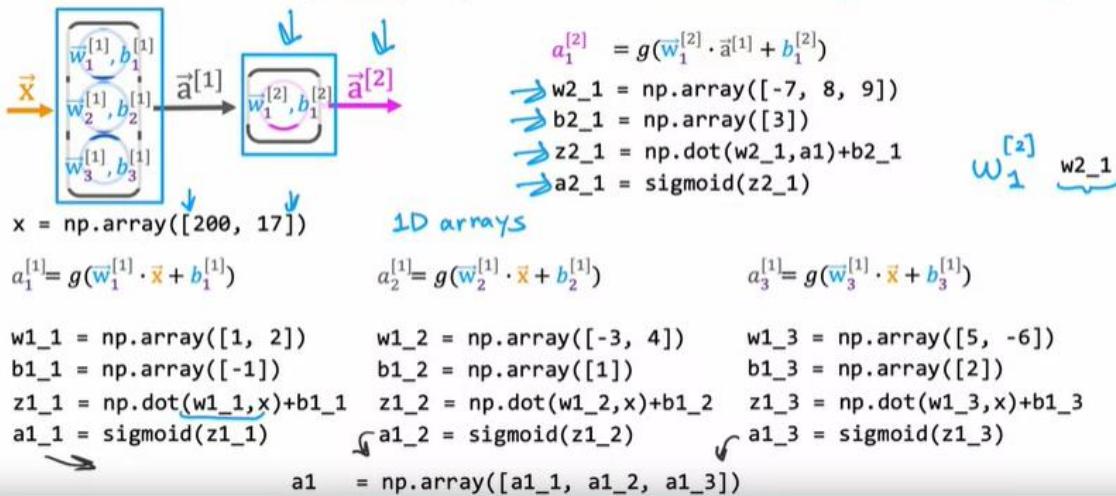


Digit classification model



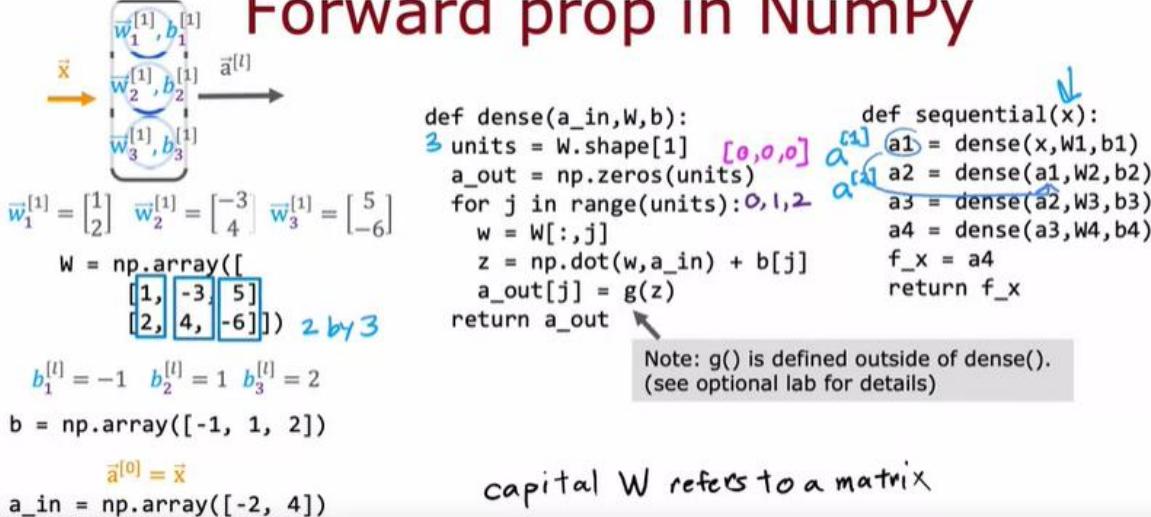
2.4 - Neural network implementation in Python : 2.4.1 - Forward prop in a single layer :

forward prop (coffee roasting model)



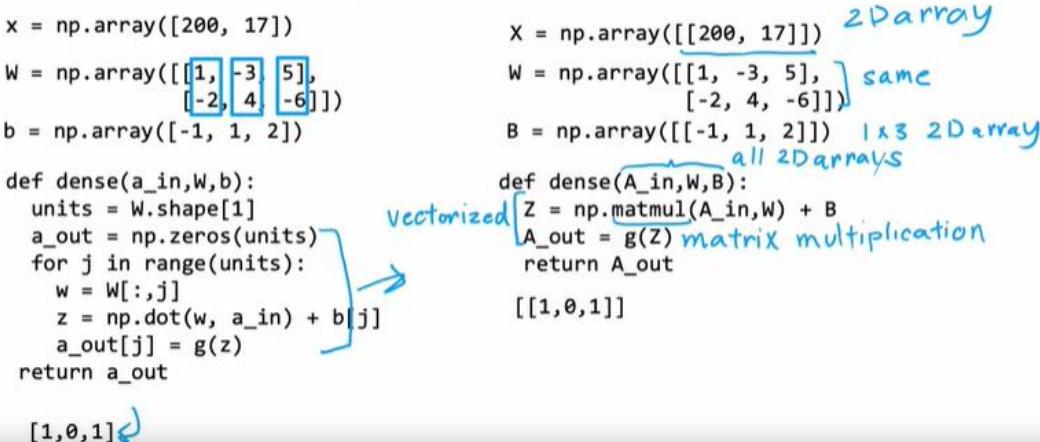
2.4 - Neural network implementation in Python : 2.4.2 - General implementation of forward propagation :

Forward prop in NumPy



2.6 - Vectorization (optional) : 2.6.1 - How neural networks are implemented efficiently :

For loops vs. vectorization



2.6 - Vectorization (optional) : 2.6.2 - Matrix multiplication :

Dot products

example $\begin{bmatrix} 1 \\ 2 \end{bmatrix} \cdot \begin{bmatrix} 3 \\ 4 \end{bmatrix}$ $z = (1 \times 3) + (2 \times 4)$ $3 + 8$ $\underline{\underline{11}}$	in general $\begin{bmatrix} \uparrow \\ \vec{a} \\ \downarrow \end{bmatrix} \cdot \begin{bmatrix} \uparrow \\ \vec{w} \\ \downarrow \end{bmatrix}$ $z = \vec{a} \cdot \vec{w}$	transpose $\vec{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$ $\vec{a}^T = [1 \ 2]$	vector vector multiplication $\begin{bmatrix} \leftarrow \vec{a}^T \rightarrow \\ 1 \times 2 \end{bmatrix} \begin{bmatrix} \uparrow \\ \vec{w} \\ \downarrow \end{bmatrix} 2 \times 1$ $z = \vec{a}^T \vec{w}$ <i>useful for understanding matrix multiplication</i>
---	---	---	--

equivalent

Vector matrix multiplication

$$\vec{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

$$\vec{a}^T = \begin{bmatrix} 1 & 2 \end{bmatrix}$$

$$W = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$$

$$Z = \vec{a}^T W \quad [\leftarrow \vec{a}^T \rightarrow] \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$$

1 by 2

$$Z = [\vec{a}^T \vec{w}_1 \quad \vec{a}^T \vec{w}_2]$$

$$(1 * 3) + (2 * 4)$$

$$3 + 8$$

$$\underline{\underline{11}}$$

$$(1 * 5) + (2 * 6)$$

$$5 + 12$$

$$\underline{\underline{17}}$$

$$Z = [11 \quad 17]$$

matrix matrix multiplication

$$A = \begin{bmatrix} 1 & -1 \\ 2 & 2 \end{bmatrix}$$

$$A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \end{bmatrix}$$

rows

$$W = \begin{bmatrix} 3 & 5 \\ 4 & 6 \end{bmatrix}$$

columns

$$Z = A^T W = \begin{bmatrix} \leftarrow \vec{a}_1^T \rightarrow \\ \leftarrow \vec{a}_2^T \rightarrow \end{bmatrix} \begin{bmatrix} \uparrow & \uparrow \\ \vec{w}_1 & \vec{w}_2 \\ \downarrow & \downarrow \end{bmatrix}$$

$$\begin{array}{c} \text{row1 col1} \\ \text{row2 col1} \end{array} = \begin{bmatrix} \vec{a}_1^T \vec{w}_1 & \vec{a}_1^T \vec{w}_2 \\ \vec{a}_2^T \vec{w}_1 & \vec{a}_2^T \vec{w}_2 \end{bmatrix} \begin{array}{c} \text{row1 col2} \\ \text{row2 col2} \end{array}$$

$$\begin{array}{c} (-1 \times 3) + (-2 \times 4) \\ -3 + -8 \\ -11 \end{array} = \begin{bmatrix} 11 & 17 \\ -11 & -17 \end{bmatrix} \quad \begin{array}{c} (-1 \times 5) + (-2 \times 6) \\ -5 + -12 \\ -17 \end{array}$$

general rules for matrix multiplication ↳ next video!

2.6 - Vectorization (optional) : 2.6.3 - Matrix multiplication rules :

Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

$$\vec{a}_1^T \vec{w}_1 = (1 \times 3) + (2 \times 4) = 11$$

3 by 4 matrix

$$\vec{a}_3^T \vec{w}_2 = (0.1 \times 5) + (0.2 \times 6) = 1.7$$

0.5 + 1.2

$$\vec{a}_2^T \vec{w}_3 = (-1 \times 7) + (-2 \times 8) = -23$$

-7 + -16

row 2 column 3?

Matrix multiplication rules

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

3 x 2 2 x 4

can only take dot products
of vectors that are same length

$$\begin{bmatrix} 0.1 & 0.2 \end{bmatrix} \quad \begin{bmatrix} 5 \\ 6 \end{bmatrix}$$

length 2 length 2

3 by 4 matrix
↳ same # rows as A^T
same # columns as W

2.6 - Vectorization (optional) : 2.6.4 - Matrix multiplication code :

Matrix multiplication in NumPy

$$A = \begin{bmatrix} 1 & -1 & 0.1 \\ 2 & -2 & 0.2 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 2 \\ -1 & -2 \\ 0.1 & 0.2 \end{bmatrix} \quad W = \begin{bmatrix} 3 & 5 & 7 & 9 \\ 4 & 6 & 8 & 0 \end{bmatrix} \quad Z = A^T W = \begin{bmatrix} 11 & 17 & 23 & 9 \\ -11 & -17 & -23 & -9 \\ 1.1 & 1.7 & 2.3 & 0.9 \end{bmatrix}$$

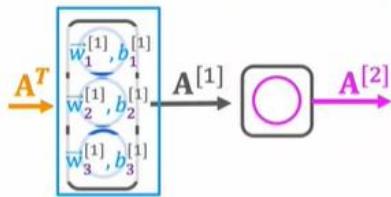
$A = np.array([[1, -1, 0.1], [2, -2, 0.2]])$ $W = np.array([[3, 5, 7, 9], [4, 6, 8, 0]])$ $Z = np.matmul(A^T, W)$
or
 $Z = A^T @ W$

$A^T = np.array([[1, 2], [-1, -2], [0.1, 0.2]])$

$A^T = A.T$ transpose

result
 $\begin{bmatrix} [11, 17, 23, 9], [-11, -17, -23, -9], [1.1, 1.7, 2.3, 0.9] \end{bmatrix}$

Dense layer vectorized



$$A^T = [200 \ 17]$$

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

$$B = [-1 \ 1 \ 2]$$

$$Z = A^T W + B$$

$$\begin{bmatrix} 165 \\ -531 \\ 900 \end{bmatrix}$$

$$z_1^{[1]} \quad z_2^{[1]} \quad z_3^{[1]}$$

$$A = g(Z)$$

$$\begin{bmatrix} 1 & 0 & 1 \end{bmatrix}$$

```
A
AT = np.array([[200, 17]])
W = np.array([[1, -3, 5],
[-2, 4, -6]])
b = np.array([-1, 1, 2])
a-in
def dense(AT,W,b):
    z = np.matmul(AT,W) + b
    a_out = g(z) a-in
    return a_out
[[1,0,1]]
```

Part (Week) 2 : Neural Network Training

2.1 - Neural Network Training

2.1.1 - TensorFlow implementation. Duration: 3 minutes3 min

2.1.2 - Training Details. Duration: 13 minutes13 min

Practice quiz: Neural Network Training

2.2 - Activation Functions

2.2.1 - Alternatives to the sigmoid activation. Duration: 5 minutes5 min

2.2.2 - Choosing activation functions. Duration: 8 minutes8 min

2.2.3 - Why do we need activation functions?. Duration: 5 minutes5 min

Lab: ReLU activation . Duration: 1 hour1h

Practice quiz: Activation Functions

2.3 - Multiclass Classification

2.3.1 - Multiclass. Duration: 3 minutes3 min

2.3.2 - Softmax. Duration: 11 minutes11 min

2.3.3 - Neural Network with Softmax output . Duration: 7 minutes7 min

2.3.4 - Improved implementation of softmax. Duration: 9 minutes9 min

2.3.5 - Classification with multiple outputs (Optional). Duration: 4 minutes4 min

Lab: Softmax. Duration: 1 hour1h

Lab: Multiclass. Duration: 15 minutes15 min

Practice quiz: Multiclass Classification

2.4 - Additional Neural Network Concepts

2.4.1 - Advanced Optimization. Duration: 6 minutes6 min

2.4.2 - Additional Layer Types. Duration: 8 minutes8 min

Practice quiz: Additional Neural Network Concepts

2.5 - Back Propagation (Optional)

2.5.1 - What is a derivative? (Optional). Duration: 22 minutes22 min

2.5.2 - Computation graph (Optional). Duration: 19 minutes19 min

2.5.3 - Larger neural network example (Optional). Duration: 9 minutes9 min

Lab: Optional Lab: Derivatives. Duration: 30 minutes30 min

Lab: Optional Lab: Back propagation. Duration: 30 minutes30 min

Practice Lab: Neural network training

2.1 - Neural Network Training : 2.1.1 - TensorFlow implementation :

Train a Neural Network in TensorFlow

Given set of (x, y) examples
How to build and train this in code?

```

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')
])
from tensorflow.keras.losses import BinaryCrossentropy
model.compile(loss=BinaryCrossentropy())
model.fit(X, Y, epochs=100) ③
    epochs: number of steps
    in gradient descent
  
```

①
②
③

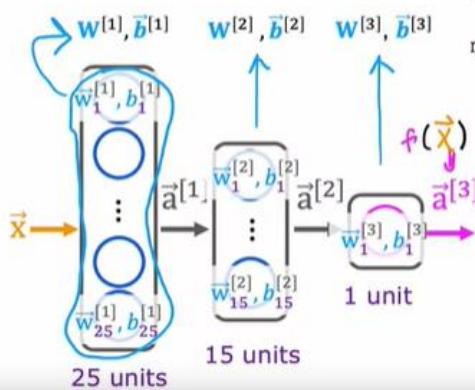
2.1 - Neural Network Training : 2.1.2 - Training Details. Duration:

Model Training Steps		
① specify how to compute output given input x and parameters w, b (define model) $f_{\vec{w}, \vec{b}}(\vec{x}) = ?$	logistic regression $\begin{aligned} z &= \text{np.dot}(w, x) + b \\ f_x &= 1 / (1 + \text{np.exp}(-z)) \end{aligned}$	TensorFlow neural network <pre>model = Sequential([Dense(...), Dense(...), Dense(...)])</pre>
② specify loss and cost $L(f_{\vec{w}, \vec{b}}(\vec{x}), y)$ 1 example $J(\vec{w}, \vec{b}) = \frac{1}{m} \sum_{i=1}^m L(f_{\vec{w}, \vec{b}}(\vec{x}^{(i)}), y^{(i)})$	logistic loss $\begin{aligned} \text{loss} &= -y * \text{np.log}(f_x) \\ &- (1-y) * \text{np.log}(1-f_x) \end{aligned}$	binary cross entropy <pre>model.compile(loss=BinaryCrossentropy())</pre>
③ Train on data to minimize $J(\vec{w}, \vec{b})$	$w = w - \text{alpha} * dj_{dw}$ $b = b - \text{alpha} * dj_{db}$	<pre>model.fit(X, y, epochs=100)</pre>

1. Create the model

define the model

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



```
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
    Dense(units=25, activation='sigmoid'),
    Dense(units=15, activation='sigmoid'),
    Dense(units=1, activation='sigmoid'),
])
```

By the way, I don't always remember the names of all the loss functions in the TensorFlow, but I just do a quick web search myself to find the right name and then I plug that into my code.

2. Loss and cost functions

handwritten digit classification problem

binary classification

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

Compare prediction vs. target

logistic loss

also known as binary cross entropy

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

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

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

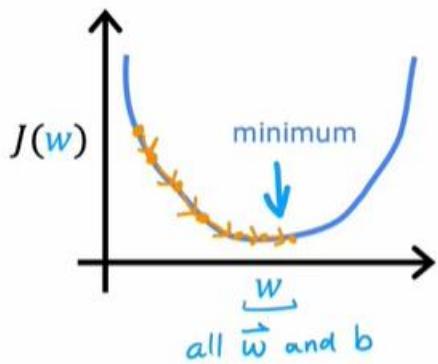
regression
(predicting numbers and not categories)

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

from tensorflow.keras.losses import
BinaryCrossentropy
Keras

from tensorflow.keras.losses import
MeanSquaredError

3. Gradient descent



repeat {

$$w_j^{[l]} = w_j^{[l]} - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$$

$$b_j^{[l]} = b_j^{[l]} - \alpha \frac{\partial}{\partial b_j} J(\vec{w}, b)$$

} Compute derivatives
for gradient descent
using "backpropagation"

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

Neural network libraries

Use code libraries instead of coding "from scratch"



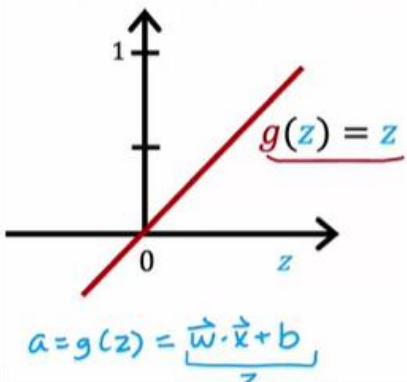
Good to understand the implementation
(for tuning and debugging).

2.2 - Activation Functions : 2.2.1 - Alternatives to the sigmoid activation:

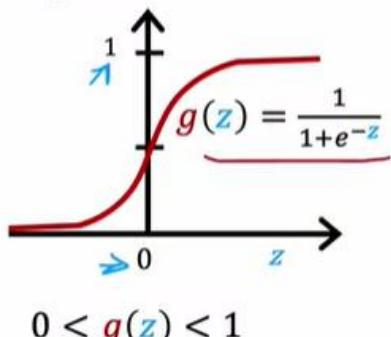
Examples of Activation Functions

"No activation function"

Linear activation function

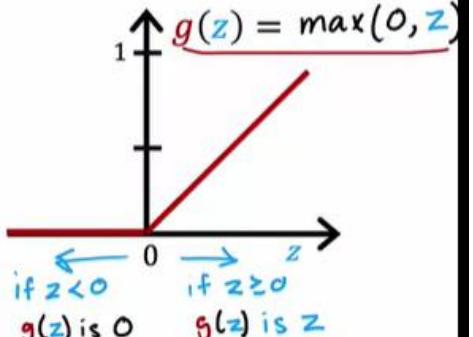


Sigmoid

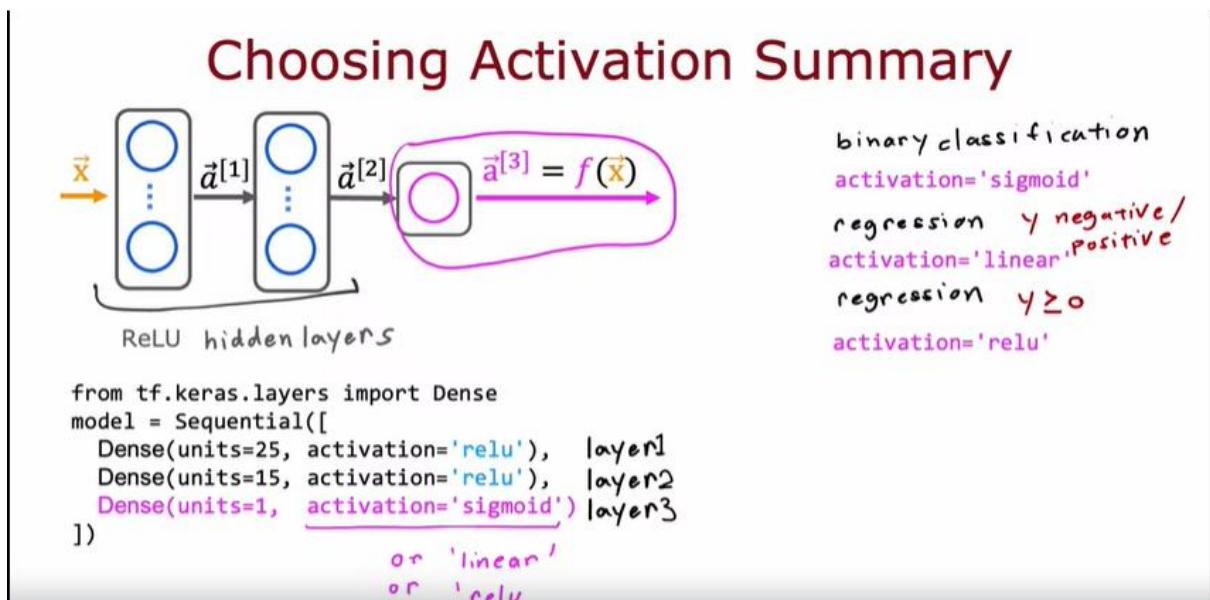
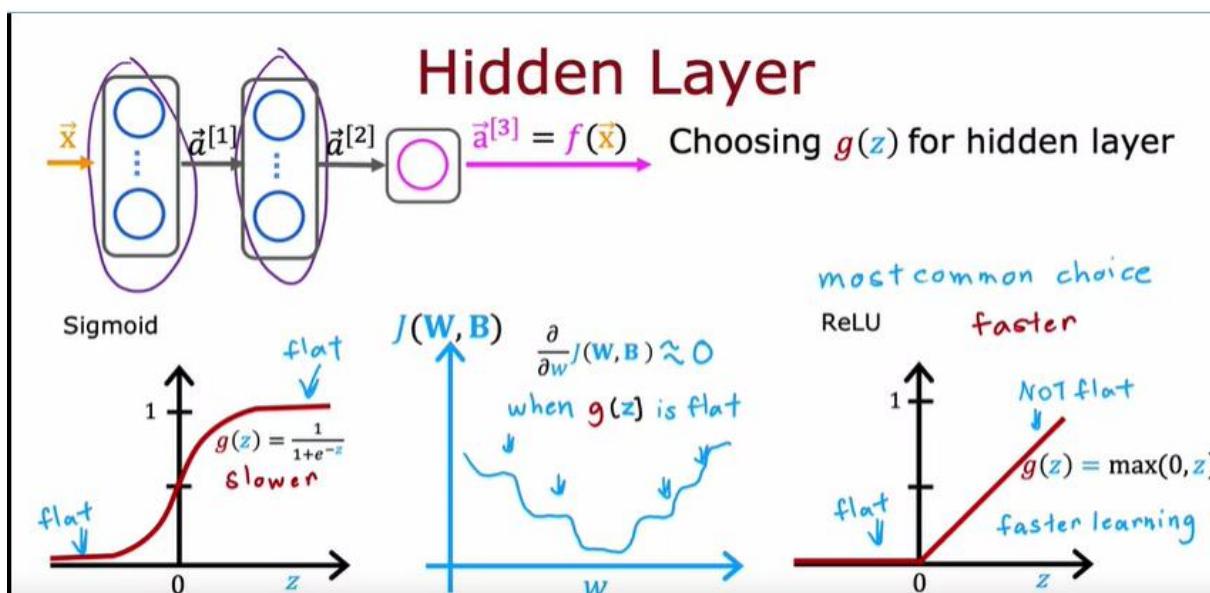
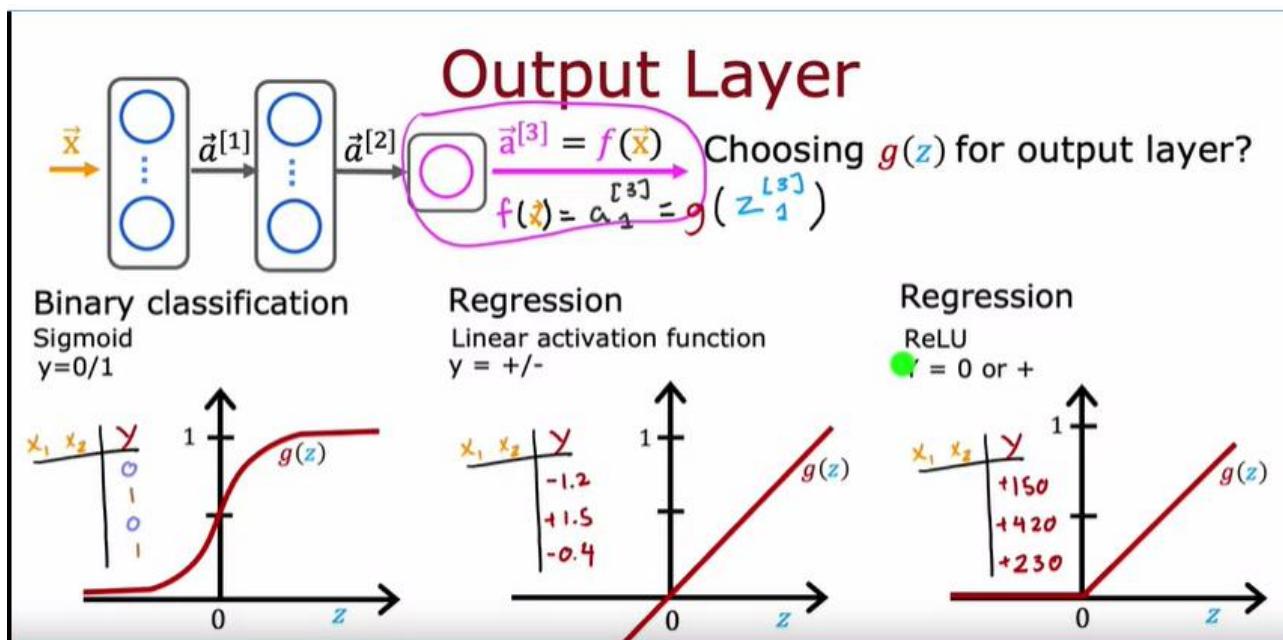
$$a_2^{[1]} = g(\overbrace{\vec{w}_2^{[1]} \cdot \vec{x}}^z + b_2^{[1]})$$


Later: softmax activation

ReLU Rectified Linear Unit

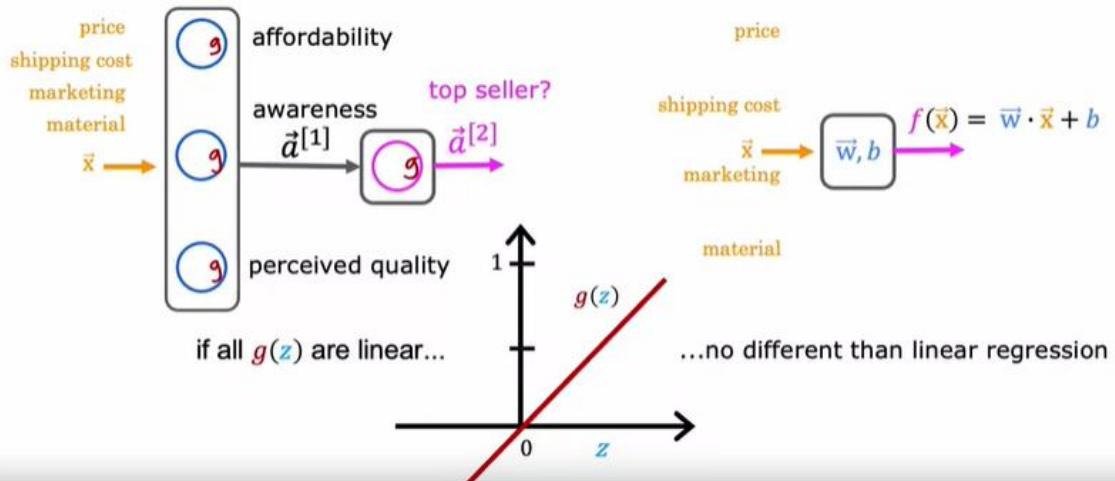


2.2 - Activation Functions : 2.2.2 - Choosing activation functions:



2.2 - Activation Functions : 2.2.3 - Why do we need activation functions? :

Why do we need activation functions?



Linear Example

one feature
 x

w is scalar
 $a^{[1]}$
 $a^{[2]}$

' a ' is scalar
 $g(z) = z$

$$\begin{aligned}
 a^{[1]} &= \underbrace{w_1^{[1]} x}_{w} + b_1^{[1]} \\
 a^{[2]} &= \underbrace{w_1^{[2]} a^{[1]}}_{w} + b_1^{[2]} \\
 &= w_1^{[2]} (w_1^{[1]} x + b_1^{[1]}) + b_1^{[2]} \\
 \vec{a}^{[2]} &= (\underbrace{\vec{w}_1^{[2]} \vec{w}_1^{[1]}}_{w}) x + \underbrace{w_1^{[2]} b_1^{[1]} + b_1^{[2]}}_{b} \\
 \vec{a}^{[2]} &= w x + b \\
 f(x) &= wx + b \text{ linear regression}
 \end{aligned}$$

Example

\vec{x}

$g(z) = z$

$\vec{a}^{[4]} = \vec{w}_1^{[4]} \cdot \vec{a}^{[3]} + b_1^{[4]}$

all linear (including output)
↳ equivalent to linear regression

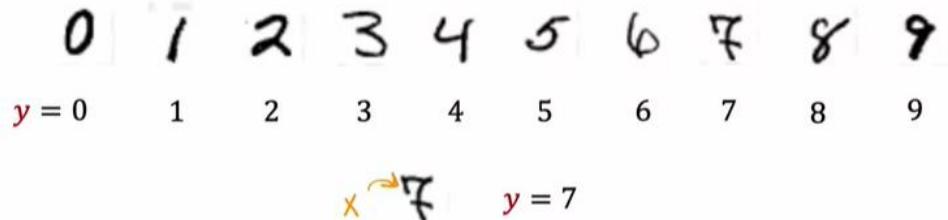
$\vec{a}^{[4]} = \frac{1}{1+e^{-(\vec{w}_1^{[4]} \cdot \vec{a}^{[3]} + b_1^{[4]})}}$

output activation is sigmoid
(hidden layers still linear)
↳ equivalent to logistic regression

Don't use linear activations in hidden layers (use ReLU)

2.3 - Multiclass Classification : 2.3.1 - Multiclass :

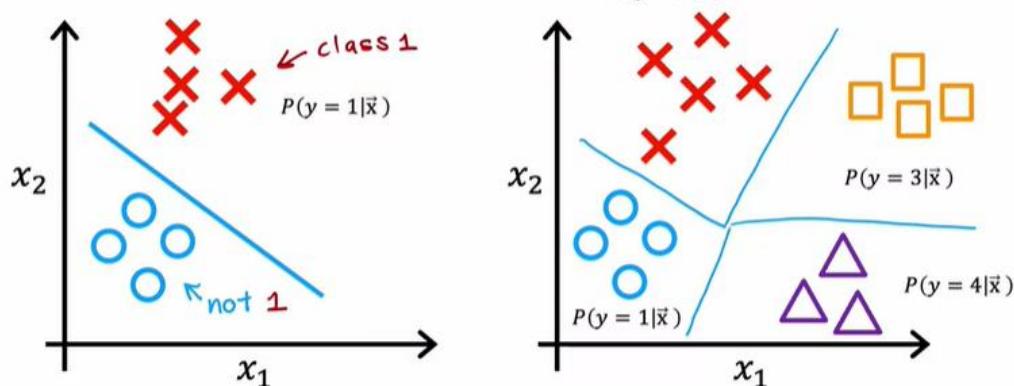
MNIST example



multiclass classification problem:
target y can take on more than two possible values

2.3 - Multiclass Classification : 2.3.2 - Softmax :

Multiclass classification example



Logistic regression
(2 possible output values)

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

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

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

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

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

parameters w_1, w_2, \dots, w_N
 b_1, b_2, \dots, b_N

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

Note: $a_1 + a_2 + \dots + a_N = 1$

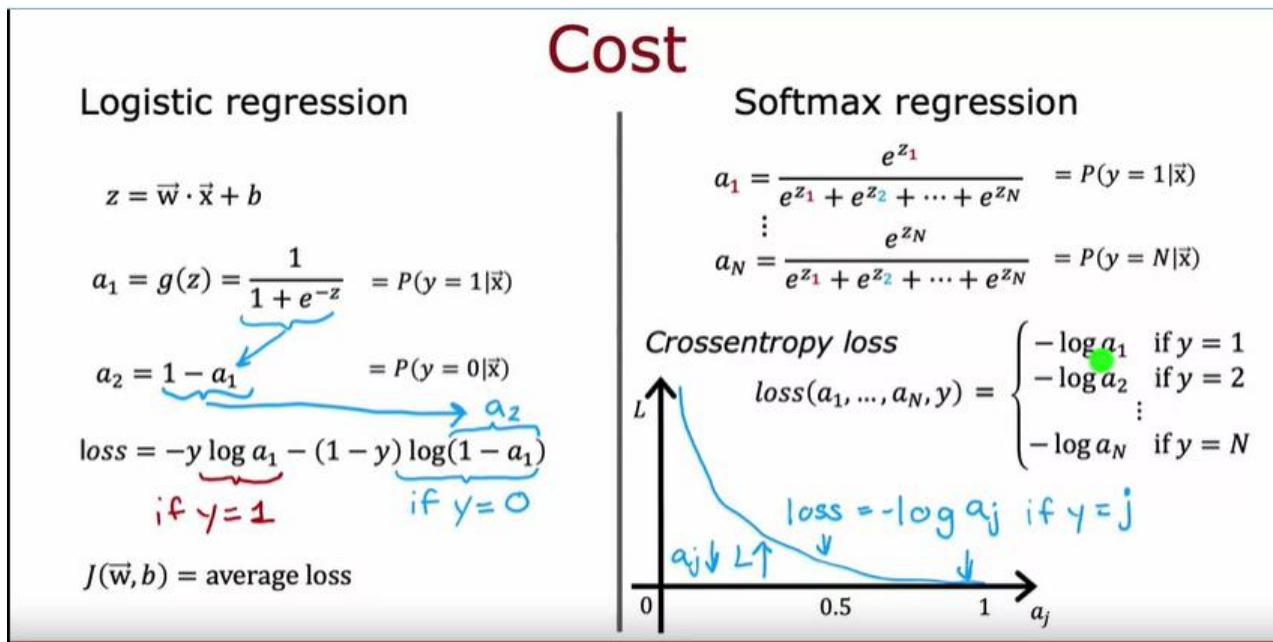
Softmax regression (4 possible outputs) $y=1, 2, 3, 4$

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

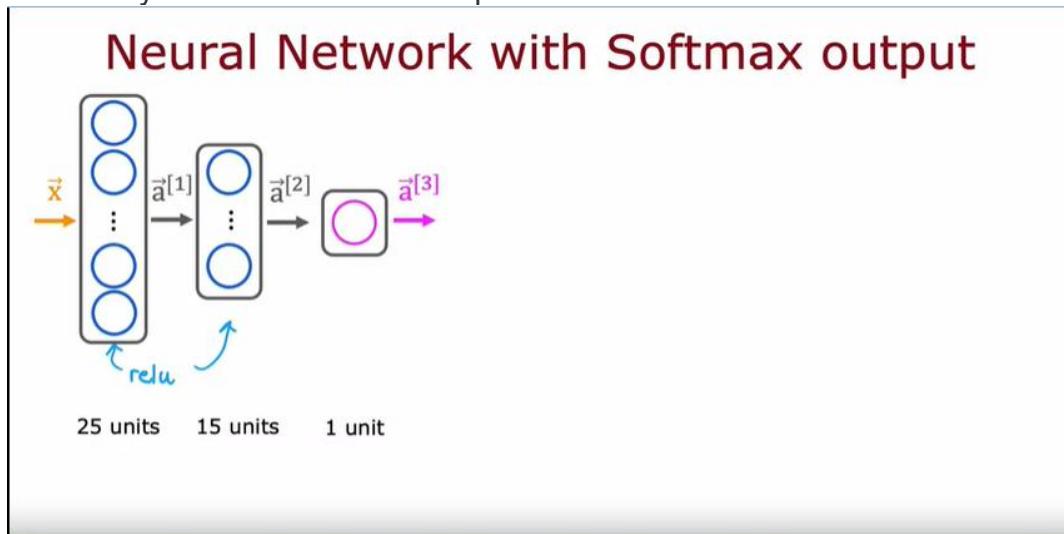
$$\textcolor{blue}{\circ} \quad z_2 = \vec{w}_2 \cdot \vec{x} + b_2 \quad a_2 = \frac{e^{z_2}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} \\ = P(y=2|\vec{x}) \quad 0.20$$

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

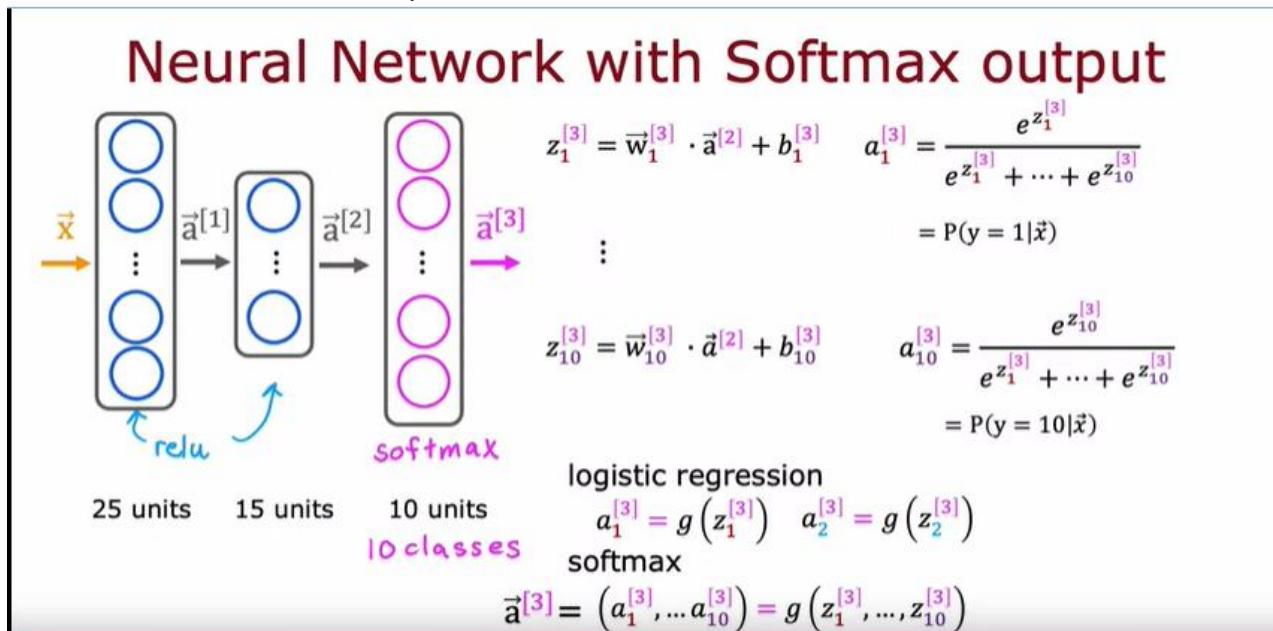
$$\textcolor{purple}{\triangle} \quad z_4 = \vec{w}_4 \cdot \vec{x} + b_4 \quad a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} \\ = P(y=4|\vec{x}) \quad 0.35$$



2.3 - Multiclass Classification : 2.3.3 - Neural Network with Softmax output :
Previously : For 2 classes --> output 0 or 1



Now : For 10 classes --> output 0 or 9



whereas for logistic regression we had the BinaryCrossentropy function, here we're using the SparseCategoricalCrossentropy function. And what sparse categorical refers to is that you're still classified y into categories. So it's categorical. This takes on values from 1 to 10. And sparse refers to that y can only take on one of these 10 values. So each image is either 0 or 1 or 2 or so on up to 9. You're not going to see a picture that is simultaneously the number two and the number seven so sparse refers to that each digit is only one of these categories.

MNIST with softmax

① specify the model

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

```
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')
])
from tensorflow.keras.losses import
    SparseCategoricalCrossentropy
model.compile(loss= SparseCategoricalCrossentropy() )
model.fit(X, Y, epochs=100)
```

② specify loss and cost

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

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

Note: better (recommended) version later.

Don't use the version shown here!

2.3 - Multiclass Classification : 2.3.4 - Improved implementation of softmax :

Numerical Roundoff Errors

option 1

$$x = \frac{2}{10,000}$$

option 2

$$x = \underbrace{\left(1 + \frac{1}{10,000}\right)}_{\text{underbrace}} - \underbrace{\left(1 - \frac{1}{10,000}\right)}_{\text{underbrace}}$$

```
In [1]: x1 = 2.0 / 10000
print(f"{x1:.18f}") # print 18 digits to the right of decimal point
0.00020000000000000000

In [2]: x2 = 1 + (1/10000) - (1 - 1/10000)
print(f"{x2: .18f}")

0.00019999999999978
```

Both the codes work well for Logistic Regression. Numerical Round Off errors play vital role in softmax regression

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

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

Logistic regression:

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

Original loss

$$\text{loss} = -y \log(a) - (1-y)\log(1-a)$$

More accurate loss (in code)

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

logit: z

Numerically More Accurate :

More numerically accurate implementation of softmax

Softmax regression

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

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

More Accurate

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

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

'linear'

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

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) ← not  $a_1 \dots a_{10}$ 
         f_x = tf.nn.softmax(logits) is  $z_1 \dots z_{10}$ 

```

logistic regression (more numerically accurate)

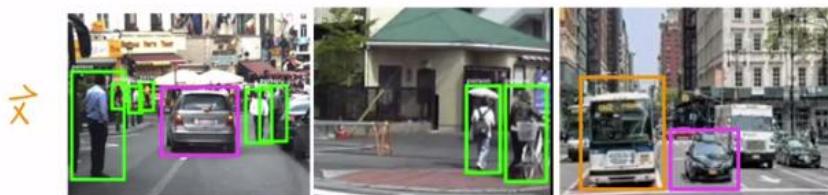
```

model   model = Sequential([
          Dense(units=25, activation='sigmoid'),
          Dense(units=15, activation='sigmoid'),
          Dense(units=1, activation='linear')
        ])
         from tensorflow.keras.losses import
         BinaryCrossentropy
loss    model.compile(..., BinaryCrossentropy(from_logits=True))
         model.fit(X, Y, epochs=100)
fit     logit = model(X)  $\downarrow z$ 
predict f_x = tf.nn.sigmoid(logit)

```

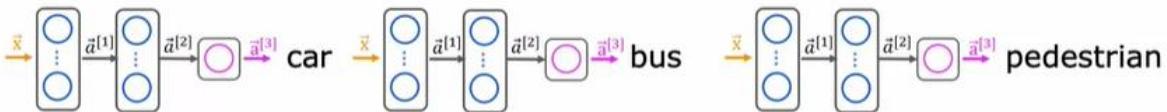
2.3 - Multiclass Classification : 2.3.5 - Classification with multiple outputs (Optional) :
Multi-Label Classification :

Multi-label Classification

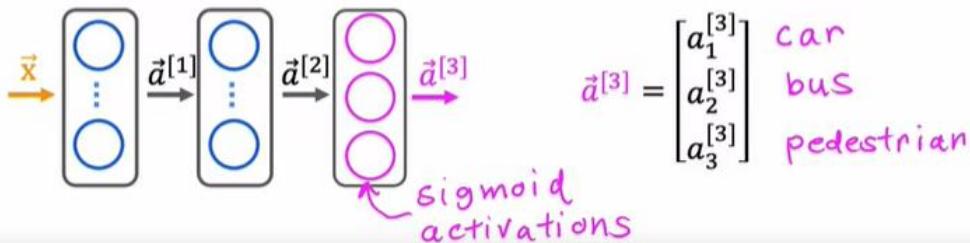


Is there a car?	<i>yes</i>	$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$	<i>no</i>	$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$	<i>yes</i>	$y = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
Is there a bus?	<i>no</i>	$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$	<i>no</i>	$y = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$	<i>yes</i>	$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$
Is there a pedestrian?	<i>yes</i>	$y = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$	<i>yes</i>	$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$	<i>no</i>	$y = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

Multi-label Classification

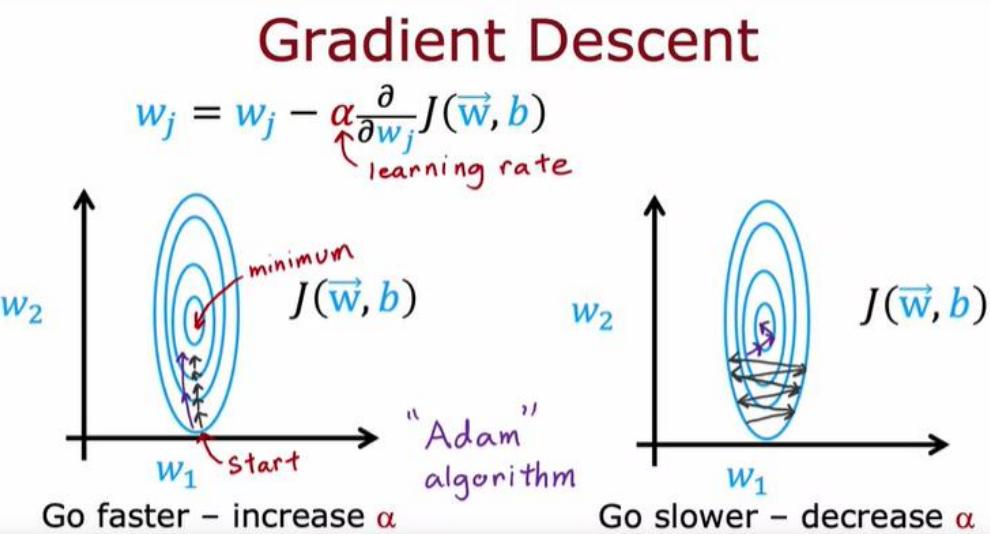


Alternatively, train one neural network with three outputs



2.4 - Additional Neural Network Concepts :
(Adam Optimization Algorithm)

2.4.1 - Advanced Optimization :

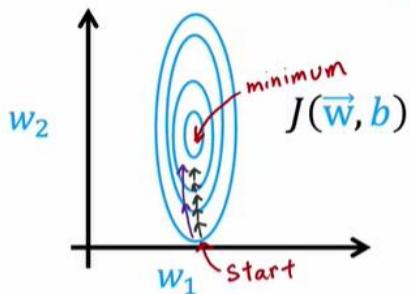


Adam Algorithm Intuition

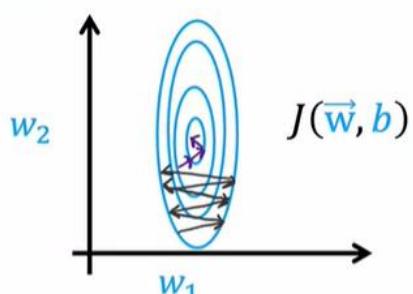
Adam: Adaptive Moment estimation not just one α

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

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

model

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

compile

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

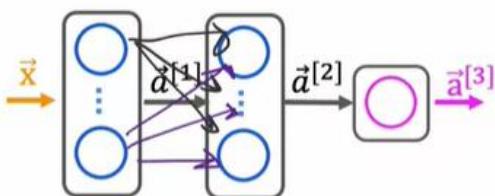
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

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

2.4 - Additional Neural Network Concepts : 2.4.2 - Additional Layer Types :

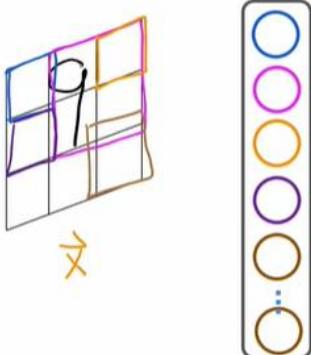
Dense Layer



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

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

Convolutional Layer

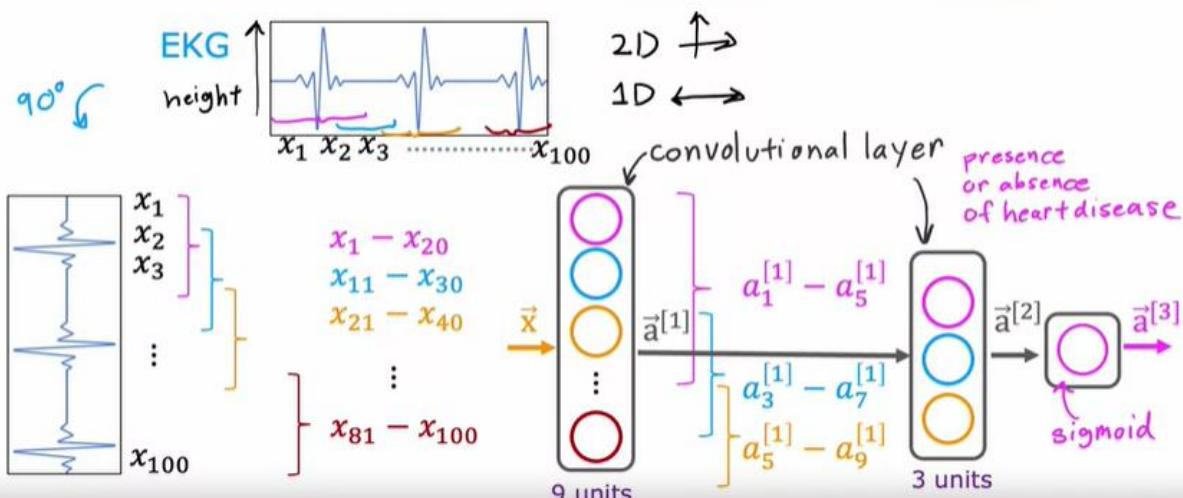


Each Neuron only looks at part of the previous layer's inputs.

Why?

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

Convolutional Neural Network



2.5 - Back Propagation (Optional) : 2.5.1 - What is a derivative? :

Derivative Example

Cost function

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

Say $w = 3$

$$J(w) = 3^2 = 9$$

$$\epsilon = 0.002$$

If we increase w by a tiny amount $\epsilon = 0.001$ how does $J(w)$ change?

$$w = 3 + 0.001 \quad 0.002$$

$$J(w) = w^2 = 9.006001$$

$$\underbrace{9.012}_{9.012} \underbrace{004}_{004}$$

$$\text{If } w \uparrow 0.001 \quad \epsilon \leftarrow 0.002 \\ J(w) \uparrow 6 \times 0.001 \quad 6 \times \epsilon \quad 6 \times 0.002 = 0.012 \\ \frac{\partial}{\partial w} J(w) = 6$$

Informal Definition of Derivative

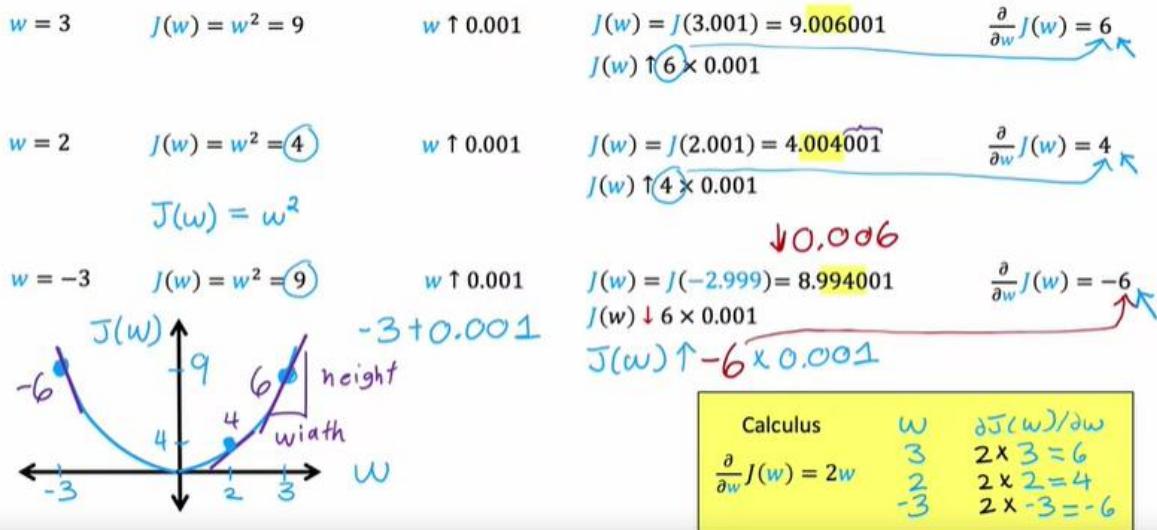
If $w \uparrow \varepsilon$ causes $J(w) \uparrow k \times \varepsilon$ then
 $\frac{\partial}{\partial w} J(w) = k$

Gradient descent
repeat {
 $w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(w, b)$
}

If derivative is small, then this update step will make a small update to w_j
If the derivative is large, then this update step will make a large update to w_j

you can use Python to compute these derivatives yourself using a nifty Python package called SymPy

More Derivative Examples



Even More Derivative Examples

$$\begin{aligned}
w = 2 & \left[\begin{array}{lll} J(w) = w^2 = 4 & \frac{\partial}{\partial w} J(w) = 2w = 4 & w \uparrow \underbrace{0.001}_{\varepsilon} \\ J(w) = w^3 = 8 & & J(w) = 4.004001 \\ J(w) = w = 2 & & J(w) \uparrow 4 \times \varepsilon \\ J(w) = \frac{1}{w} = \frac{1}{2} = 0.5 & & \end{array} \right]
\end{aligned}$$

```
In [2]: J, w = sympy.symbols('J,w')
In [3]: J = w**2
Out[3]: w2
In [5]: dJ_dw = sympy.diff(J,w)
dJ_dw
Out[5]: 2w
In [6]: dJ_dw.subs([(w,2)])
Out[6]: 4
In [ ]:
```

```
In [2]: J, w = sympy.symbols('J,w')
In [10]: J=w #J=w**3 #J = w**2
Out[10]: w
In [11]: dJ_dw = sympy.diff(J,w)
dJ_dw
Out[11]: 1
In [12]: dJ_dw.subs([(w,2)])
Out[12]: 1
In [ ]:
```

```
In [2]: J, w = sympy.symbols('J,w')
In [7]: J=w**3 #J = w**2
J
Out[7]: w3
In [8]: dJ_dw = sympy.diff(J,w)
dJ_dw
Out[8]: 3w2
In [9]: dJ_dw.subs([(w,2)])
Out[9]: 12
In [ ]:
```

```
In [1]: import sympy
In [2]: J, w = sympy.symbols('J,w')
In [13]: J=1/w #J=w #J=w**3 #J = w**2
J
Out[13]: 1
w
In [14]: dJ_dw = sympy.diff(J,w)
dJ_dw
Out[14]: -1
w2
In [15]: dJ_dw.subs([(w,2)])
Out[15]: -1
4
In [ ]:
```

Even More Derivative Examples

$w = 2$

$J(w) = w^2 = 4$	$\frac{\partial}{\partial w} J(w) = 2w = 4$	$w \uparrow \underbrace{0.001}_{\varepsilon}$	$J(w) = 4.004001$ $J(w) \uparrow 4 \times \varepsilon$
$J(w) = w^3 = 8$	$\frac{\partial}{\partial w} J(w) = 3w^2 = 12$	$w \uparrow \varepsilon$	$J(w) = 8.012006$ $J(w) \uparrow 12 \times \varepsilon$
$J(w) = w = 2$	$\frac{\partial}{\partial w} J(w) = 1$	$w \uparrow \varepsilon$	$J(w) = 2.001$ $J(w) \uparrow 1 \times \varepsilon$
$J(w) = \frac{1}{w} = 0.5$	$\frac{\partial}{\partial w} J(w) = -\frac{1}{w^2} = -\frac{1}{4}$	$w \uparrow \varepsilon$ $w = \frac{1}{2.001}$	-0.25×0.001 $0.5 - 0.00025$ $J(w) = 0.49975$ $J(w) \uparrow -\frac{1}{4} \times \varepsilon$
$\frac{\partial}{\partial w} J(w)$ $w \uparrow \varepsilon$ $J(w) \uparrow k \times \varepsilon$			

A note on derivative notation

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

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

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

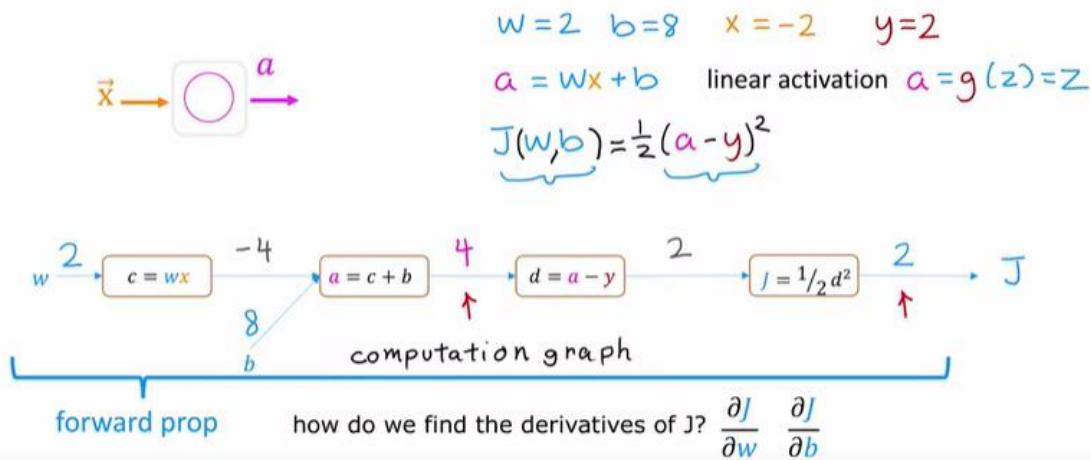
$$\partial \quad \frac{\partial}{\partial w_i} J(w_1, w_2, \dots, w_n) \quad \frac{\partial J}{\partial w_i} \quad \text{or} \quad \frac{\partial}{\partial w_i} J$$

"partial derivative"

notation used
in these courses

2.5 - Back Propagation (Optional) : 2.5.2 - Computation graph :

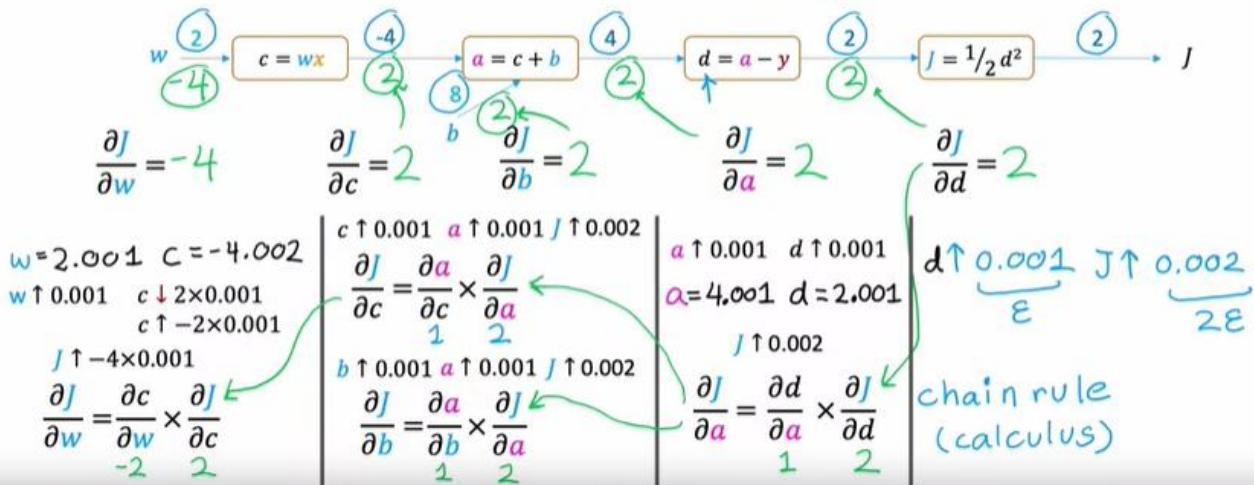
Small Neural Network Example



Computing the Derivatives

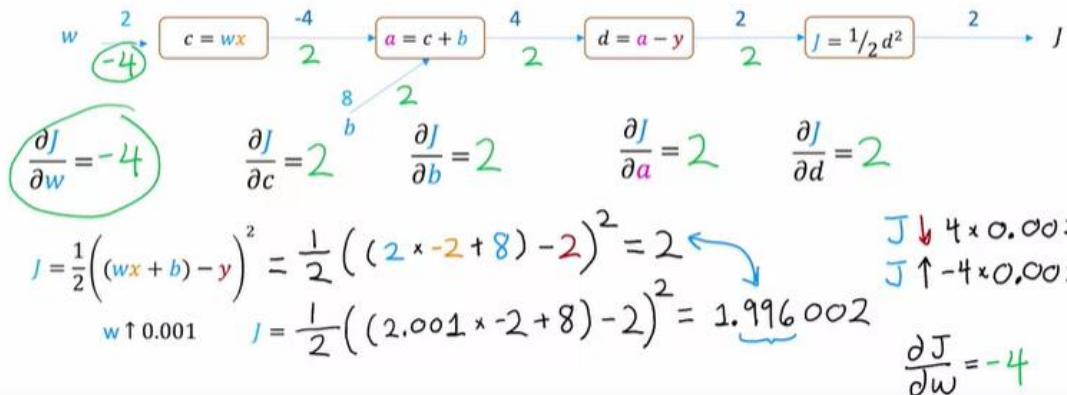
$$w = 2 \quad b = 8 \quad x = -2 \quad y = 2 \quad a = wx + b \quad J = \frac{1}{2}(a-y)^2$$

Forward prop
Back prop

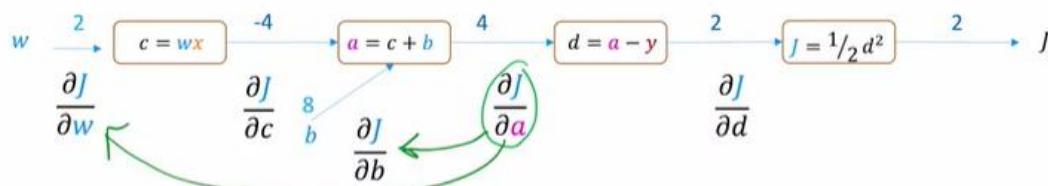


Computing the Derivatives

$$w = 2 \quad b = 8 \quad x = -2 \quad y = 2 \quad a = wx + b \quad J = \frac{1}{2}(a - y)^2$$



Backprop is an efficient way to compute derivatives



Compute $\frac{\partial J}{\partial a}$ once and use it to compute both $\frac{\partial J}{\partial w}$ and $\frac{\partial J}{\partial b}$.

If N nodes and P parameters, compute derivatives

in roughly $N + P$ steps rather than $N \times P$ steps.

N	P	N+P	N×P
10,000	100,000	1.1×10^5	10^9

2.5 - Back Propagation (Optional) : 2.5.3 - Larger neural network example :

Neural Network Example

$x = 1 \ y = 5$

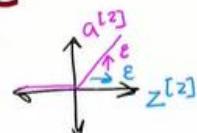
$w^{[1]} = 2, b^{[1]} = 0$

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

$a^{[1]} = g(w^{[1]}x + b^{[1]}) = w^{[1]}x + b^{[1]}$

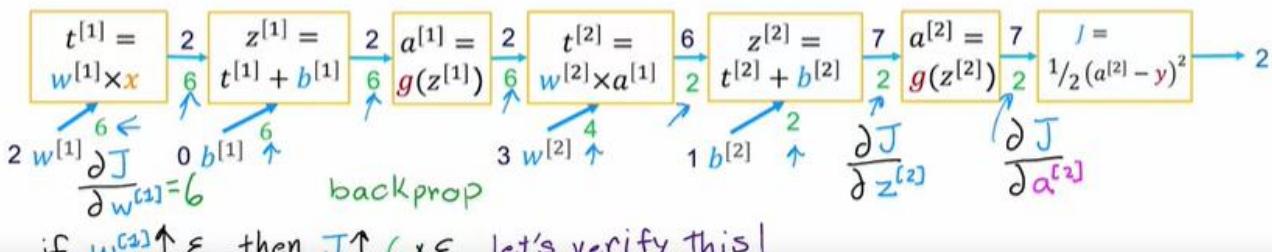
$\underline{z_1^{[1]}}$ $\underline{z_2^{[1]}}$

ReLU activation



$$a^{[2]} = \textcolor{red}{g}(\textcolor{blue}{w}^{[2]} a^{[1]} + \textcolor{teal}{b}^{[2]}) = \underbrace{\textcolor{blue}{w}^{[2]} a^{[1]} + \textcolor{teal}{b}^{[2]}}_{\text{Z-3}} = 3 \times 2 + 1 = 7$$

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

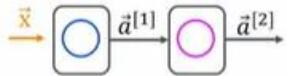


Neural Network Example

$$x = 1 \quad y = 5$$

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

ReLU activation



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

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

$$a^{[1]} = g(w^{[1]} x + b^{[1]}) = w^{[1]} x + b^{[1]} = 2 \times 1 + 0 = 2$$

$$a^{[2]} = g(w^{[2]} a^{[1]} + b^{[2]}) = w^{[2]} a^{[1]} + b^{[2]} = 3 \times 2 + 1 = 7$$

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

$\frac{2.001}{2}$

$$\frac{2.001}{2} \uparrow 0.001 \quad J \uparrow 6 \times 0.001 \quad \frac{\partial J}{\partial w^{[1]}} = 6$$

$$\frac{\partial J}{\partial w^{[1]}} \quad \frac{\partial J}{\partial b^{[1]}}$$

N nodes $\square \rightarrow \square \rightarrow \square$

P parameters
 $w_1, b_1, w_2, b_2, \dots$

inefficient way

$N \times P$

efficient way (backprop)

$N + P$

Part (Week) 3 : Machine Learning**3.1 - Advice for applying machine learning**

3.1.1 - Deciding what to try next. Duration: 3 minutes3 min

3.1.2 - Evaluating a model. Duration: 10 minutes10 min

3.1.3 - Model selection and training/cross validation/test sets. Duration: 13 minutes13 min

Lab: Optional Lab: Model Evaluation and Selection. Duration: 30 minutes30 min

Practice quiz: Advice for applying machine learning

3.2 - Bias and variance

3.2.1 - Diagnosing bias and variance. Duration: 11 minutes11 min

3.2.2 - Regularization and bias/variance. Duration: 10 minutes10 min

3.2.3 - Establishing a baseline level of performance. Duration: 9 minutes9 min

3.2.4 - Learning curves. Duration: 11 minutes11 min

3.2.5 - Deciding what to try next revisited. Duration: 8 minutes8 min

3.2.6 - Bias/variance and neural networks. Duration: 10 minutes10 min

Lab: Optional Lab: Diagnosing Bias and Variance. Duration: 30 minutes30 min

Practice quiz: Bias and variance

3.3 - Machine learning development process

3.3.1 - Iterative loop of ML development. Duration: 7 minutes7 min

3.3.2 - Error analysis. Duration: 8 minutes8 min

3.3.3 - Adding data. Duration: 14 minutes14 min

3.3.4 - Transfer learning: using data from a different task. Duration: 11 minutes11 min

3.3.5 - Full cycle of a machine learning project. Duration: 8 minutes8 min

3.3.6 - Fairness, bias, and ethics. Duration: 9 minutes9 min

Practice quiz: Machine learning development process

3.4 - Skewed datasets (optional)

3.4.1 - Error metrics for skewed datasets. Duration: 11 minutes11 min

3.4.2 - Trading off precision and recall. Duration: 11 minutes11 min

Practice Lab: Advice for applying machine learning

3.4 - Skewed datasets (optional) : 3.4.2 - Trading off precision and recall :**3.1 - Advice for applying machine learning : 3.1.1 - Deciding what to try next ::**

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

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

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

- Get more training examples
- Try smaller sets of features
- Try getting additional features
- Try adding polynomial features ($x_1^2, x_2^2, x_1x_2, \text{etc}$)
- Try decreasing λ
- Try increasing λ



3.1 - Advice for applying machine learning : 3.1.1 - Deciding what to try next :

Machine learning diagnostic

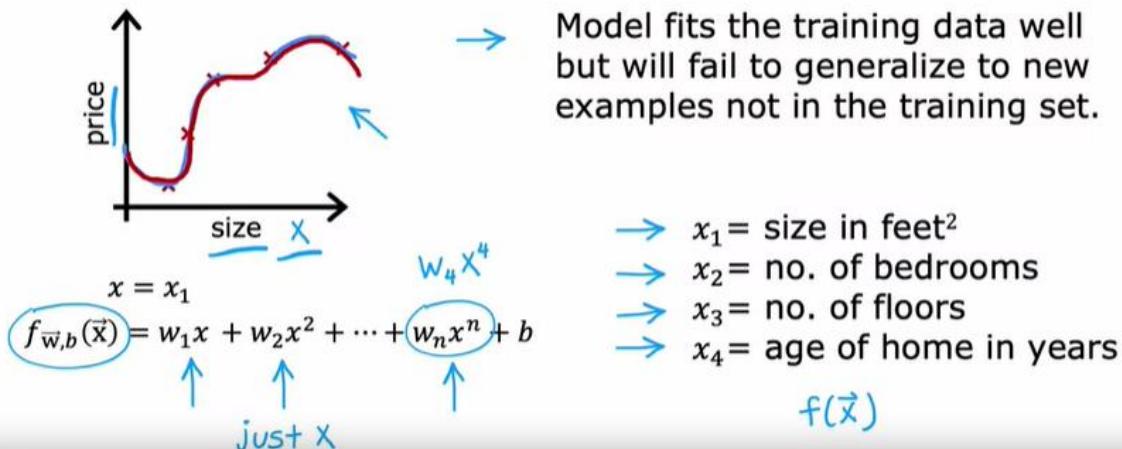
Diagnostic:

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.

3.1 - Advice for applying machine learning : 3.1.2 - Evaluating a model :

Evaluating your model



Evaluating your model

Dataset:

size	price	
2104	400	70%
1600	330	
2400	369	
1416	232	
3000	540	
1985	300	
1534	315	
1427	199	30%
1380	212	
1494	243	

training set $\rightarrow (x^{(1)}, y^{(1)})$
 $(x^{(2)}, y^{(2)})$
 \vdots
 $(x^{(m_{train})}, y^{(m_{train})})$

$m_{train} = \text{no. training examples} = 7$

test set $\rightarrow (x_{test}^{(1)}, y_{test}^{(1)})$
 \vdots
 $(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

$m_{test} = \text{no. test examples} = 3$

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

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

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

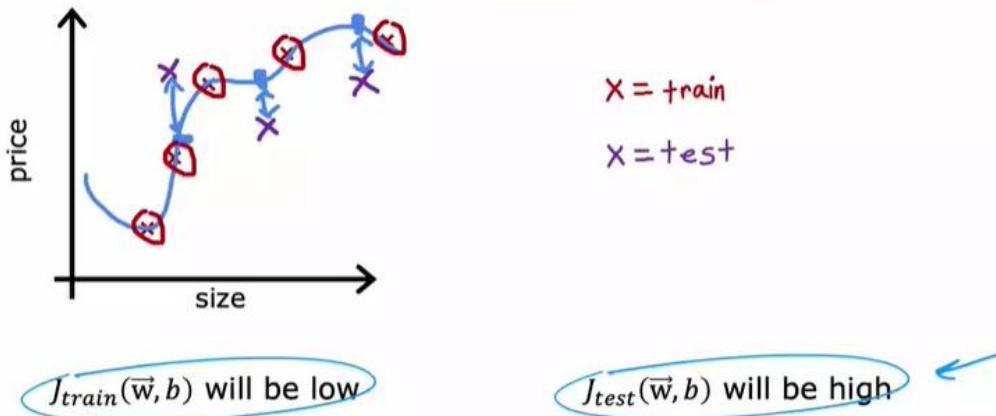
Compute test error:

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

Compute training error:

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

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



Train/test procedure for classification problem

fraction of the test set and the fraction of the train set that the algorithm has misclassified.

count $\hat{y} \neq y$

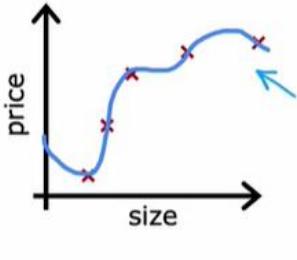
$$\hat{y} = \begin{cases} 1 & \text{if } f_{\vec{w}, b}(\vec{x}^{(i)}) \geq 0.5 \\ 0 & \text{if } f_{\vec{w}, b}(\vec{x}^{(i)}) < 0.5 \end{cases}$$

$J_{test}(\vec{w}, b)$ is the fraction of the test set that has been misclassified.

$J_{train}(\vec{w}, b)$ is the fraction of the train set that has been misclassified.

3.1 - Advice for applying machine learning : 3.1.3 - Model selection and 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 $J_{train}(\vec{w}, b)$.

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

Model selection (choosing a model)

- $d=1$ 1. $f_{\vec{w}, b}(\vec{x}) = w_1 x + b \rightarrow w^{<1>} , b^{<1>} \rightarrow J_{test}(w^{<1>} , b^{<1>})$
- $d=2$ 2. $f_{\vec{w}, b}(\vec{x}) = w_1 x + w_2 x^2 + b \rightarrow w^{<2>} , b^{<2>} \rightarrow J_{test}(w^{<2>} , b^{<2>})$
- $d=3$ 3. $f_{\vec{w}, b}(\vec{x}) = w_1 x + w_2 x^2 + w_3 x^3 + b \rightarrow w^{<3>} , b^{<3>} \rightarrow J_{test}(w^{<3>} , b^{<3>})$
- ⋮
- $d=10$ 10. $f_{\vec{w}, b}(\vec{x}) = w_1 x + w_2 x^2 + \dots + w_{10} x^{10} + b \rightarrow J_{test}(w^{<10>} , b^{<10>})$

Choose $w_1 x + \dots + w_5 x^5 + b \quad d=5 \quad J_{test}(w^{<5>} , b^{<5>})$

How well does the model perform? Report test set error $J_{test}(w^{<5>} , b^{<5>})$?

The problem: $J_{test}(w^{<5>} , b^{<5>})$ is likely to be an optimistic estimate of generalization error (ie. $J_{test}(w^{<5>} , b^{<5>}) <$ generalization error). Because an extra parameter d (degree of polynomial) was chosen using the test set.

w, b are overly optimistic estimate of generalization error on training data.

Training/cross validation/test set

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

→ training set → $(x^{(1)}, y^{(1)})$ $M_{train} = 6$

60% \vdots

$(x^{(m_{train})}, y^{(m_{train})})$

cross validation → $(x_{cv}^{(1)}, y_{cv}^{(1)})$ $M_{cv} = 2$

20% \vdots

$(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$

test set → $(x_{test}^{(1)}, y_{test}^{(1)})$ $M_{test} = 2$

20% \vdots

$(x_{test}^{(m_{test})}, y_{test}^{(m_{test})})$

Training/cross validation/test set

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

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

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

Model selection

$$\begin{aligned} d=1 \quad & 1. \quad f_{\vec{w}, b}(\vec{x}) = w_1 x + b \quad w^{(1)}, b^{(1)} \rightarrow J_{cv}(w^{(1)}, b^{(1)}) \\ d=2 \quad & 2. \quad f_{\vec{w}, b}(\vec{x}) = w_1 x + w_2 x^2 + b \quad \rightarrow J_{cv}(w^{(2)}, b^{(2)}) \\ d=3 \quad & 3. \quad f_{\vec{w}, b}(\vec{x}) = w_1 x + w_2 x^2 + w_3 x^3 + b \\ & \vdots \\ d=10 \quad & 10. \quad f_{\vec{w}, b}(\vec{x}) = w_1 x + w_2 x^2 + \dots + w_{10} x^{10} + b \quad J_{cv}(w^{(10)}, b^{(10)}) \end{aligned}$$

→ Pick $w_1 x + \dots + w_4 x^4 + b$ ($J_{cv}(w^{(4)}, b^{(4)})$)

Estimate generalization error using test the set: $J_{test}(w^{(4)}, b^{(4)})$

It's considered best practice in machine learning that if you have to make decisions about your model, such as fitting parameters or choosing the model architecture, such as neural network architecture or degree of polynomial if you're fitting a linear regression, to make all those decisions only using your training set and your cross-validation set, and to not look at the test set at all while you're still making decisions regarding your learning algorithm. It's only after you've come up with one model as your final model to only then evaluate it on the test set and because you haven't made any decisions using the test set, that ensures that your test set is a fair and not overly optimistic estimate of how well your model will generalize to new data.

Model selection – choosing a neural network architecture

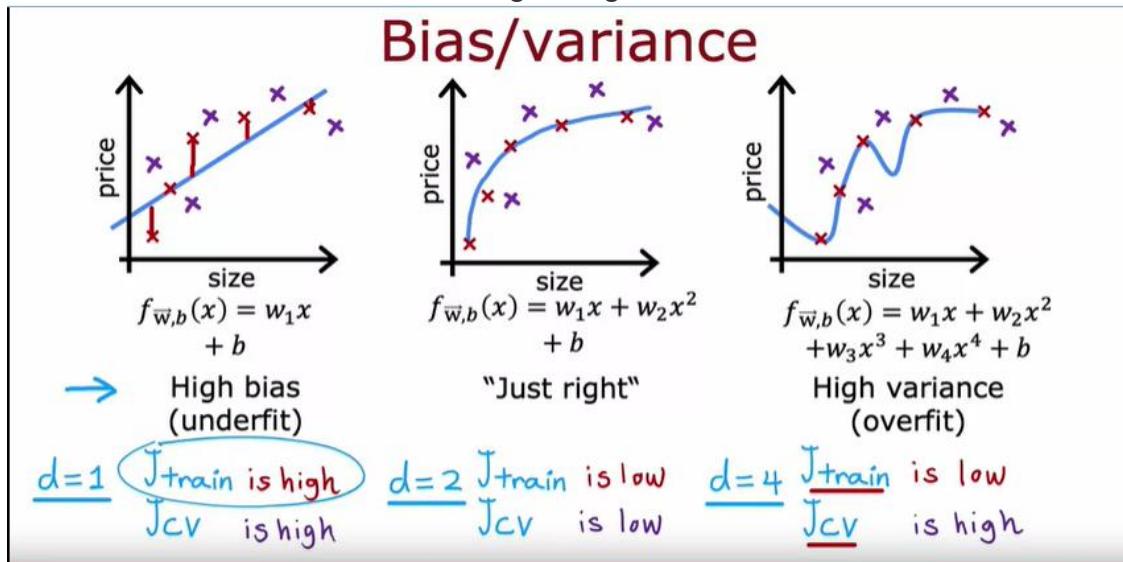


Pick $w^{(2)}, b^{(2)}$

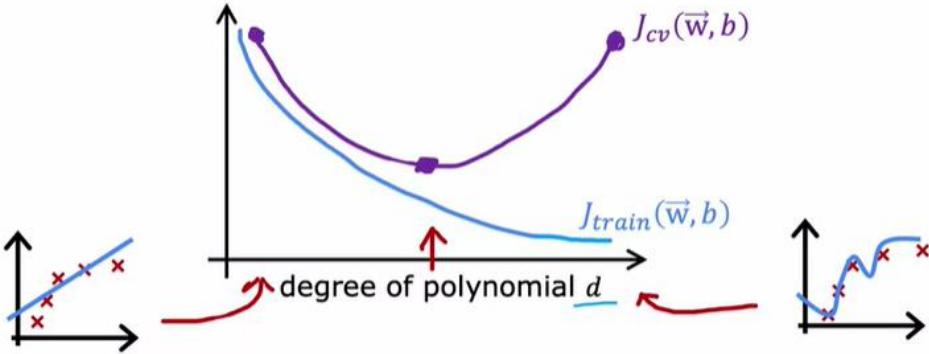
train, CV

Estimate generalization error using the test set: $J_{test}(w^{(2)}, b^{(2)})$

3.2 - Bias and variance : 3.2.1 - Diagnosing bias and variance :



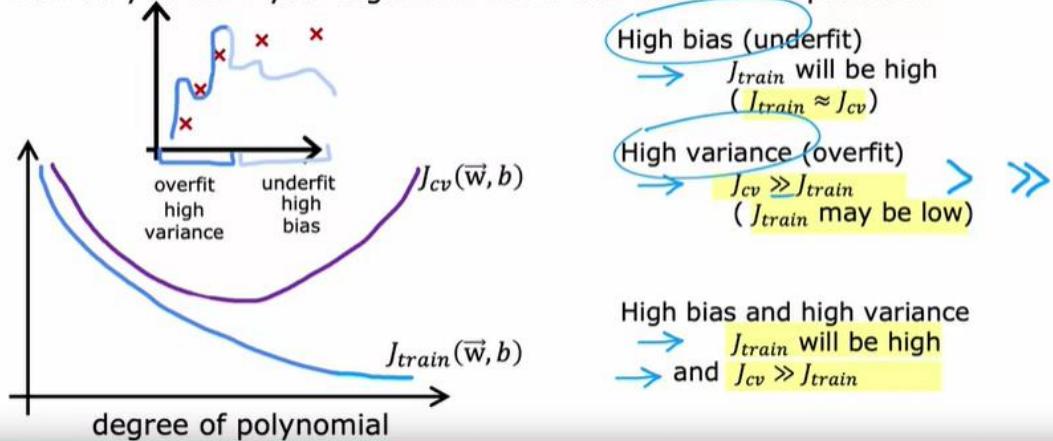
Understanding bias and variance



In some cases, it is possible to simultaneously have high bias and have high-variance. You won't see this happen that much for linear regression, but it turns out that if you're training a neural network, there are some applications where unfortunately you have high bias and high variance.

Diagnosing bias and variance

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

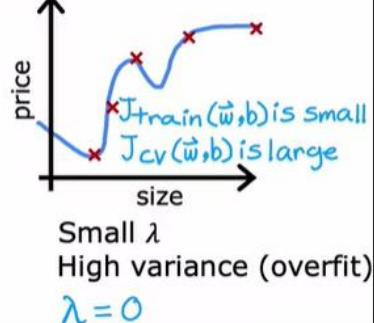
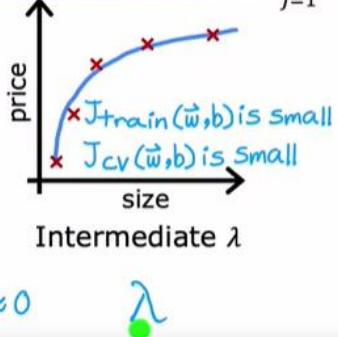
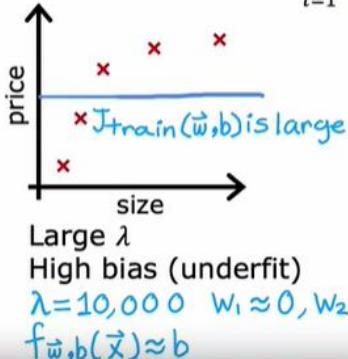


3.2 - Bias and variance : 3.2.2 - Regularization and bias/variance :

Linear regression with regularization

Model: $f_{\vec{w}, b}(x) = \underbrace{w_1 x}_{m} + \underbrace{w_2 x^2}_{n} + \underbrace{w_3 x^3}_{m} + \underbrace{w_4 x^4}_{n} + b$

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



The problem we're addressing is if you're fitting a fourth-order polynomial, so that's the model and you're using regularization, how can you choose a good value of Lambda?

Choosing the regularization parameter λ

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

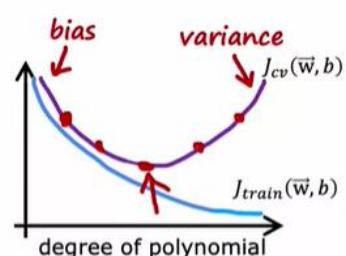
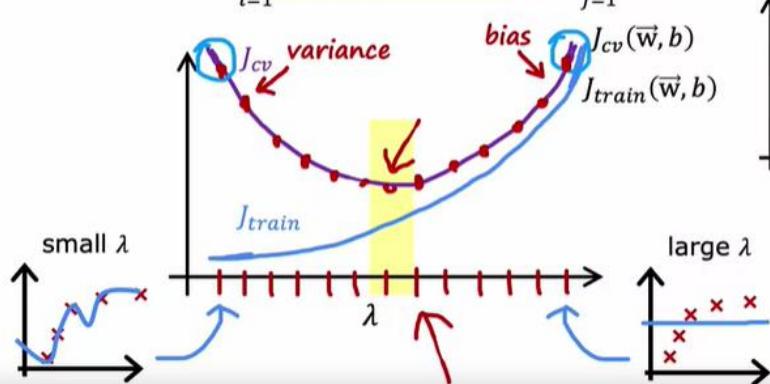
- 1. Try $\lambda = 0$ → $\min_{\vec{w}, b} J(\vec{w}, b)$ → $w^{<1>} b^{<1>}$ → $J_{cv}(w^{<1>}, b^{<1>})$
- 2. Try $\lambda = 0.01$ → $w^{<2>} b^{<2>}$ → $J_{cv}(w^{<2>}, b^{<2>})$
- 3. Try $\lambda = 0.02$ → $w^{<3>} b^{<3>}$ → $J_{cv}(w^{<3>}, b^{<3>})$
- 4. Try $\lambda = 0.04$
- 5. Try $\lambda = 0.08$
- ⋮
- 12. Try $\lambda \approx 10$ → $w^{<12>} b^{<12>}$ → $J_{cv}(w^{<12>}, b^{<12>})$

Pick $w^{<5>} b^{<5>}$

Report test error: $J_{test}(w^{<5>} b^{<5>})$

Bias and variance as a function of regularization parameter λ

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



3.2 - Bias and variance : 3.2.3 - Establishing a baseline level of performance :

Speech recognition example



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

\downarrow 0.2%
 \uparrow 4.0%



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%
 Training error (J_{train}) : 10.8%
 Cross validation error (J_{cv}) : 14.8%

\downarrow 0.2% 10.6% \uparrow 4.4% 10.6% \uparrow 4.4%
 \downarrow 4.0% 15.0% \downarrow 0.5% 15.0% \downarrow 4.7%

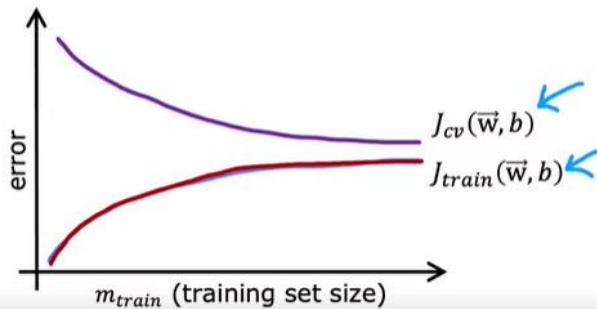
high variance high bias high bias
 high variance

3.2 - Bias and variance : 3.2.4 - Learning curves :

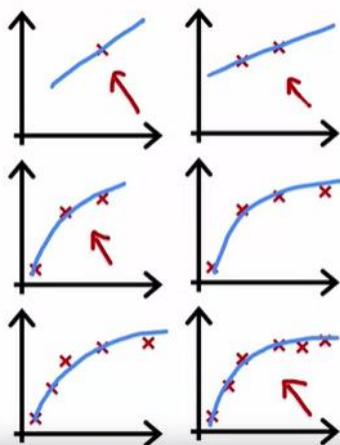
Learning curves

J_{train} = training error

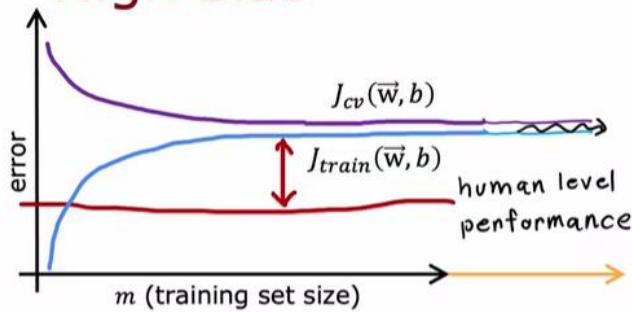
J_{cv} = cross validation error



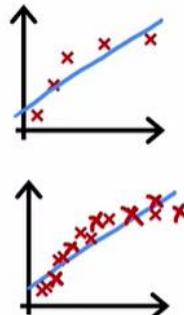
$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + b$$



High bias

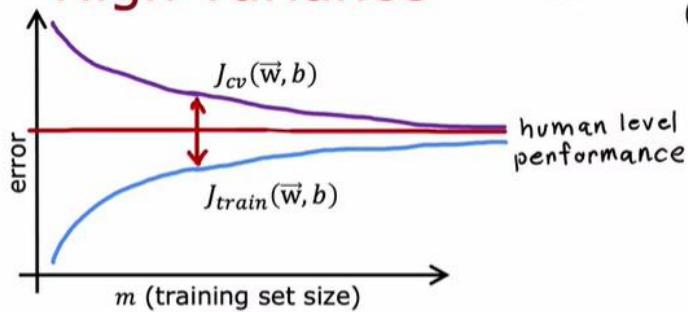


$$f_{\vec{w}, b}(x) = w_1 x + b$$



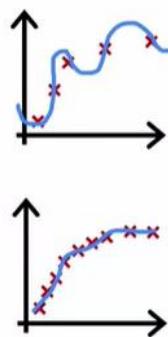
if a learning algorithm suffers from high bias,
getting more training data will not (by itself)
help much.

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.

3.2 - Bias and variance : 3.2.5 - Deciding what to try next revisited :

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

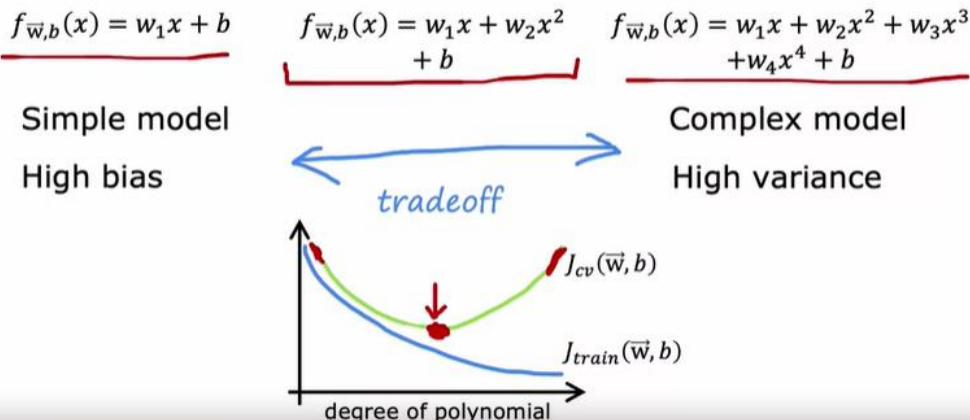
$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

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

- Get more training examples
 - Try smaller sets of features x, x^2, \dots
 - Try getting additional features
 - Try adding polynomial features ($x_1^2, x_2^2, x_1x_2, \dots$)
 - Try decreasing λ
 - Try increasing λ
- fixes high variance
fixes high variance
- fixes high bias
fixes high bias
- fixes high bias
fixes high bias
- fixes high variance
fixes high variance

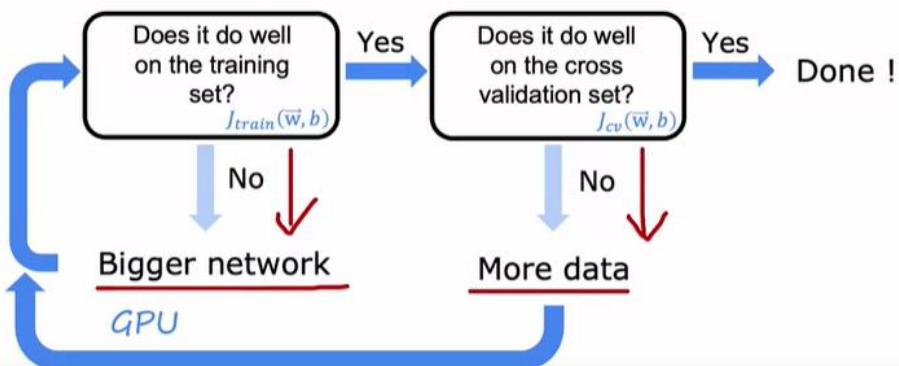
3.2 - Bias and variance : 3.2.6 - Bias/variance and neural networks :

The bias variance tradeoff

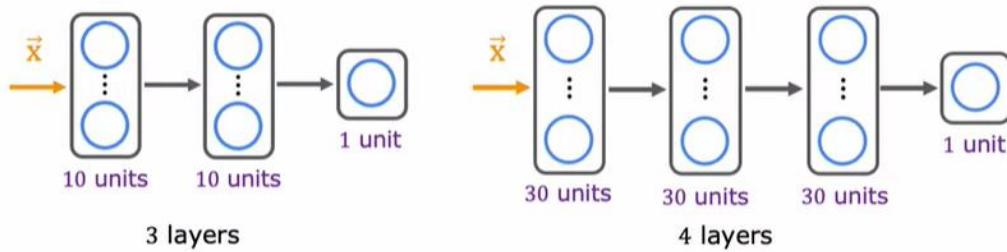


Neural networks and bias variance

Large neural networks are low bias machines



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.

Neural network regularization

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

Unregularized MNIST model

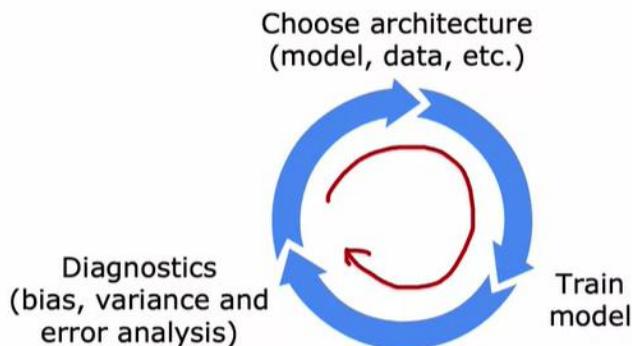
```
layer_1 = Dense(units=25, activation="relu")
layer_2 = Dense(units=15, activation="relu")
layer_3 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2, layer_3])
```

Regularized MNIST model

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

3.3 - Machine learning development process : 3.3.1 - Iterative loop of ML development :

Iterative loop of ML development



Spam classification example

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

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

From: Alfred Ng
 To: Andrew Ng
 Subject: Christmas dates?

Hey Andrew,
 Was talking to Mom about plans for Xmas. When do you get off work. Meet Dec 22?
 Alf

Building a spam classifier

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

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

$$\vec{x} = \begin{bmatrix} 0 \\ 1 \\ 2 \\ 1 \\ 1 \\ 0 \\ \vdots \end{bmatrix} \quad \begin{array}{l} a \\ andrew \\ buy \\ deal \\ discount \\ \vdots \end{array}$$

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

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

Honeypot projects: These are very large-scale projects that create a large number of fake email addresses and tries to deliberately to get these fake email addresses into the hands of spammers so that when they send spam email to these fake emails well we know these are spam email messages and so this is a way to get a lot of spam data.

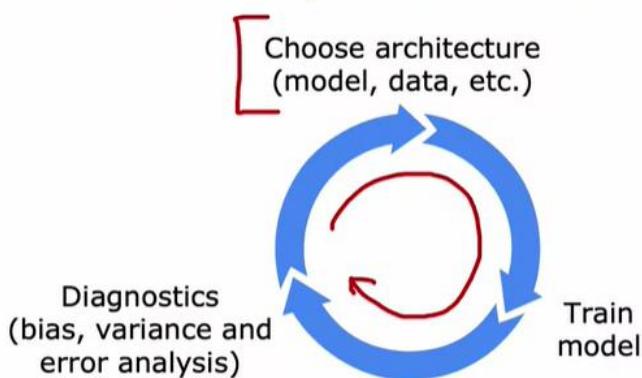
- Choosing the more promising path forward can speed up your project easily 10 times compared to if you were to somehow choose some of the less promising directions.
- For example, we've already seen that if your algorithm has high bias rather than high variance, then spending months and months on a honeypot project may not be the most fruitful direction.
- But if your algorithm has high variance, then collecting more data could help a lot.

Building a spam classifier

How to try to reduce your spam classifier's error?

- Collect more data. E.g., "Honeypot" project. 
- Develop sophisticated features based on email routing (from email header).
- Define sophisticated features from email body. E.g., should "discounting" and "discount" be treated as the same word.
- Design algorithms to detect misspellings. E.g., w4tches, med1cine, m0rtgage.

Iterative loop of ML development



3.3 - Machine learning development process : 3.3.2 - Error analysis :

- Spam is sometimes also, instead of writing the spam message in the email body they instead create an image and then writes to spam the message inside an image that appears in the email. This makes it a little bit harder for learning algorithm to figure out what's going on. Maybe some of those emails are these embedded image spam.
- The categories for Error Analysis** can be overlapping or in other words they're not mutually exclusive. For example, there can be a pharmaceutical spam email that also has unusual routing or a password that has deliberate misspellings and is also trying to carry out the phishing attack. One email can be counted in multiple categories.

Error analysis

$m_{cv} = \frac{500}{5000}$ examples in cross validation set.

Algorithm misclassifies 100 of them.

Manually examine 100 examples and categorize them based on common traits.

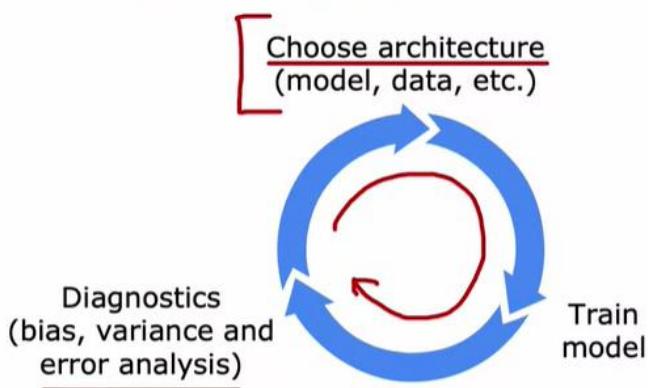
- | | |
|--|---------------------------|
| <u>Pharma:</u> <u>21</u> | <u>more data features</u> |
| <u>Deliberate misspellings (w4tches, med1cine):</u> <u>3</u> | |
| <u>Unusual email routing:</u> <u>7</u> | |
| <u>Steal passwords (phishing):</u> <u>18</u> | <u>more data features</u> |
| <u>Spam message in embedded image:</u> <u>5</u> | |

Building a spam classifier

How to try to reduce your spam classifier's error?

- Collect more data. E.g., "Honeypot" project.
- Develop sophisticated features based on email routing (from email header).
- Define sophisticated features from email body.
E.g., should "discounting" and "discount" be treated as the same word.
- Design algorithms to detect misspellings.
E.g., w4tches, med1cine, m0rtgage.

Iterative loop of ML development



3.3 - Machine learning development process : 3.3.3 - Adding data :

Adding data

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

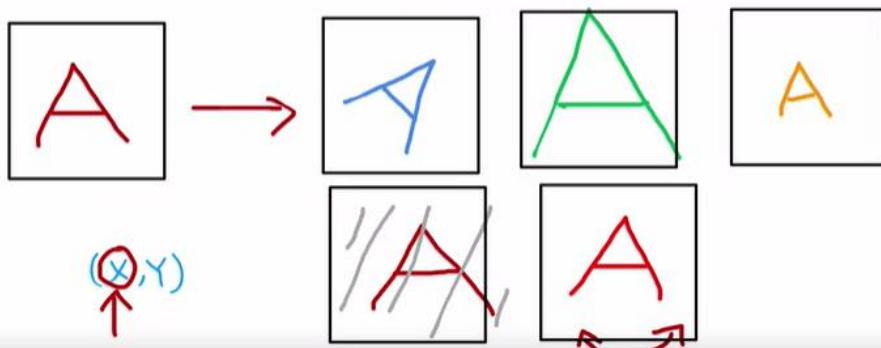
Add more data of the types where error analysis has indicated it might help.

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

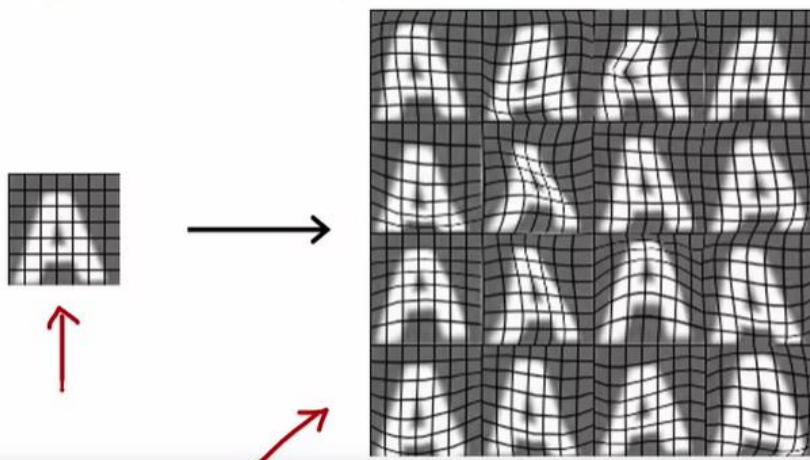
Beyond getting brand new training examples (x, y), another technique: Data augmentation

Data augmentation

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



Data augmentation by introducing distortions



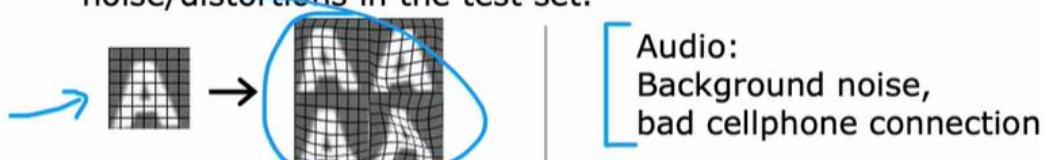
Data augmentation for speech

Speech recognition example

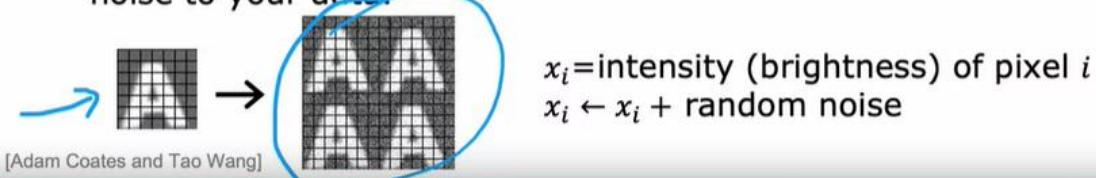
-  Original audio (voice search: "What is today's weather?")
-  + Noisy background: Crowd
-  + Noisy background: Car
-  + Audio on bad cellphone connection

Data augmentation by introducing distortions

Distortion introduced should be representation of the type of noise/distortions in the test set.



Usually does not help to add purely random/meaningless noise to your data.



Data synthesis

Synthesis: using artificial data inputs to create a new training example.

Artificial data synthesis for photo OCR



[<http://www.publicdomainpictures.net/>]

Artificial data synthesis for photo OCR



Real data

[Adam Coates and Tao Wang]

Abcdefg

Abcdefg

Abcdefg

Abcdefg

Abcdefg

Artificial data synthesis for photo OCR



Real data

[Adam Coates and Tao Wang]



Synthetic data

- Sometimes it can be more fruitful to spend more of your time taking a **data centric approach** in which you focus on engineering the data used by your algorithm.
- And this can be anything from collecting more data just on pharmaceutical spam. If that's what error analysis told you to do. To using **data augmentation** to generate more images or more audio or using **data synthesis** to just create more training examples.
- And sometimes that focus on the data can be an efficient way to help your learning algorithm improve its performance

Engineering the data used by your system

Conventional
model-centric
approach:

$$\text{AI} = \text{Code} + \text{Data}$$

(algorithm/model)

work on this

Data-centric
approach:

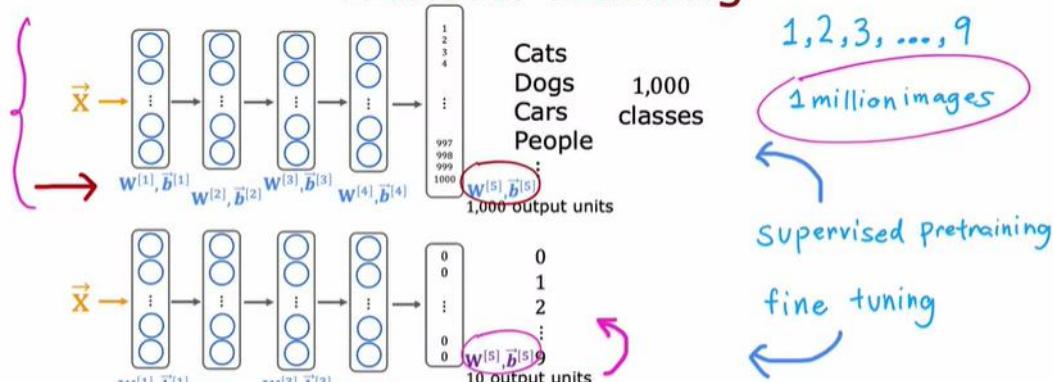
$$\text{AI} = \text{Code} + \text{Data}$$

(algorithm/model)

work on this

3.3 - Machine learning development process : 3.3.4 - Transfer learning: using data from a different task :

Transfer learning

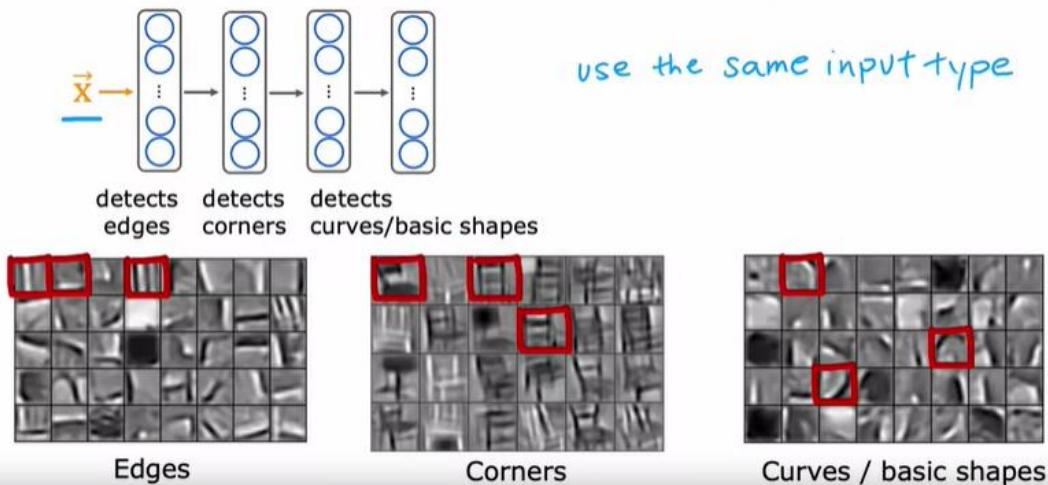


Option 1: only train **output layers** parameters.

Option 2: **train all parameters**.

- If the final task you want to solve is a computer vision tasks, then the pre-training step also has been a neural network trained on the same type of input, namely an image of the desired dimensions.
- Conversely, if your goal is to build a speech recognition system to process audio, then a neural network pre-trained on images probably won't do much good on audio. Instead, you want a neural network pre-trained on audio data, there you then fine tune on your own audio dataset and the same for other types of applications.

Why does transfer learning work?



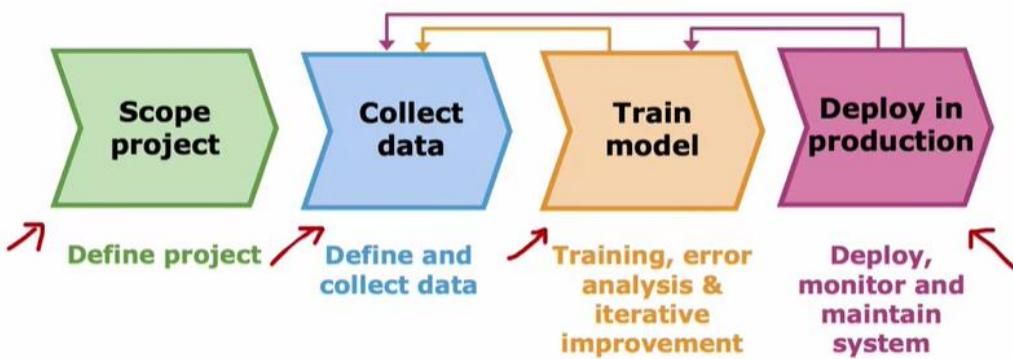
Transfer learning summary

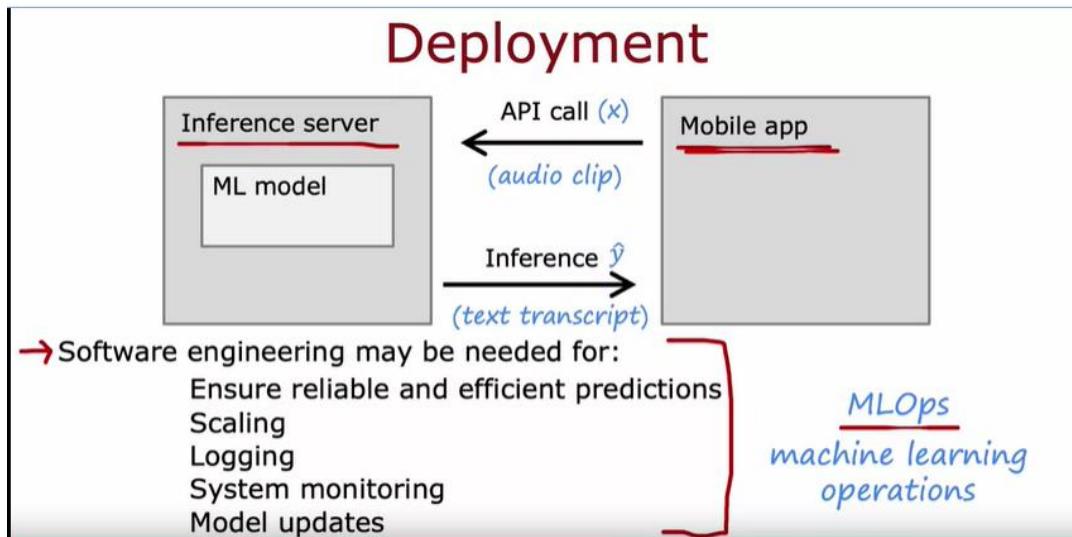
1. Download neural network parameters pretrained on a large dataset with same input type (e.g., images, audio, text) as your application (or train your own). *1 million images*
2. Further train (fine tune) the network on your own data. *1000 images*
50 images

3.3 - Machine learning development process : 3.3.5 - Full cycle of a machine learning project

- Project Scope : Eg. speech recognition for voice search

Full cycle of a machine learning project





3.3 - Machine learning development process : 3.3.6 - Fairness, bias, and ethics :

Bias

Hiring tool that discriminates against women.

Facial recognition system matching dark skinned individuals to criminal mugshots.

Biased bank loan approvals.

Toxic effect of reinforcing negative stereotypes.

Adverse use cases

Deepfakes



By the company buzzfeed of a deepfake of former US President Barack Obama and

Adverse use cases

Deepfakes

Spreading toxic/incendiary speech through optimizing for engagement.

Generating fake content for commercial or political purposes.

Using ML to build harmful products, commit fraud etc.

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

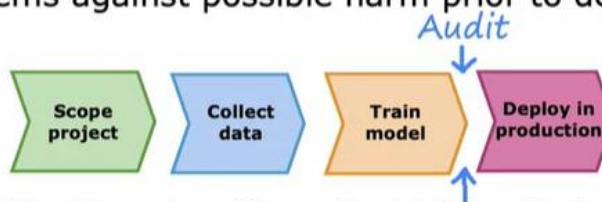
- Even after deployment to continue to monitor harm so that you can then trigger a mitigation plan and act quickly in case there is a problem that needs to be addressed.
- For example, all of the self driving car teams prior to rolling out self driving cars on the road had developed mitigation plans for what to do in case the car ever gets involved in an accident so that if the car was ever in an accident, there was already a mitigation plan that they could execute immediately rather than have a car get into an accident and then only scramble after the fact to figure out what to do.

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.

3.4 - Skewed datasets (optional) : 3.4.1 - Error metrics for skewed datasets :

Rare disease classification example

Train classifier $f_{\bar{w}, b}(\vec{x})$

($y = 1$ if disease present,
 $y = 0$ otherwise)

Find that you've got 1% error on test set
(99% correct diagnoses)

Only 0.5% of patients have the disease

`print("y=0")`

99.5% accuracy, 0.5% error

1%
1.2%

less

Usefulness

more

Precision/recall

$y = 1$ in presence of rare class we want to detect.

		Actual Class	
		1	0
Predicted Class	1	True positive 15	False positive 5
	0	False negative 10	True negative 70
	↓ 25	↓ 75	

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

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

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

$$\text{Recall} = \frac{\text{True positives}}{\#\text{actual positive}} = \frac{\text{True positives}}{\text{True pos} + \text{False neg}} = \frac{15}{15+10} = 0.6$$

print("y=0")

- When you have a rare class, looking at precision and recall and making sure that both numbers are decently high, that hopefully helps reassure you that your learning algorithm is actually useful.

3.4 - Skewed datasets (optional) : 3.4.2 - Trading off precision and recall :

- In the ideal case, we like for learning algorithms that have **high precision and high recall**.
- High precision would mean that if a diagnosis of patients have that rare disease, probably the patient does have it and it's an accurate diagnosis.
- High recall means that if there's a patient with that rare disease, probably the algorithm will correctly identify that they do have that disease.
- But it turns out that in practice there's often a trade-off between precision and recall.
- By plotting the **Precision-Recall curve**, you can then try to pick a **threshold** which corresponds to picking a point on this curve that balances the cost of false positives and false negatives or that balances the **benefits** of high precision and high recall.

Trading off precision and recall

Logistic regression: $0 < f_{\vec{w}, b}(\vec{x}) < 1$

→ Predict 1 if $f_{\vec{w}, b}(\vec{x}) \geq 0.5$ 0.7 0.3

→ Predict 0 if $f_{\vec{w}, b}(\vec{x}) < 0.5$ 0.1 0.3

Suppose we want to predict $y = 1$ (rare disease) only if very confident.
higher precision, lower recall

Suppose we want to avoid missing too many cases of rare disease (when in doubt predict $y = 1$)
lower precision, higher recall

More generally predict 1 if: $f_{\vec{w}, b}(\vec{x}) \geq \text{threshold}$.

precision = $\frac{\text{true positives}}{\text{total predicted positive}}$

recall = $\frac{\text{true positives}}{\text{total actual positive}}$

A graph showing the Precision-Recall curve. The vertical axis is labeled 'Precision' and ranges from 0 to 1. The horizontal axis is labeled 'Recall' and ranges from 0 to 1. A blue curve starts at a point on the y-axis (representing 100% precision) and curves down towards the bottom right. Two specific points on the curve are highlighted with red dots and labeled with their respective threshold values: 'threshold = 0.99' near the top left and 'threshold = 0.01' near the bottom right. Arrows point from these labels to their corresponding points on the curve.

- The metric called the **F1 score** is sometimes used to automatically combine precision recall to help you pick the best value or the **best trade-off** between the two.
- F1 score is a way of combining P and R precision and recall but that gives more **emphasis to whichever of these values is lower**.

F1 score

How to compare precision/recall numbers?

	Precision (P)	Recall (R)	Average	F_1 score
Algorithm 1	0.5	0.4	0.45	0.444
Algorithm 2	0.7	0.1	0.4	0.175
Algorithm 3	0.02	1.0	0.501	0.0392

print("y=1")

~~Average = $\frac{P+R}{2}$~~

$F_1 \text{ score} = \frac{1}{\frac{1}{2}(\frac{1}{P} + \frac{1}{R})} = 2 \frac{PR}{P+R}$

Harmonic mean

Part (Week) 4 : Decision trees**4.1 - Decision trees**

4.1.1 - Decision tree model. Duration: 7 minutes7 min

4.1.2 - Learning Process. Duration: 11 minutes11 min

Practice quiz: Decision trees

4.2 - Decision tree learning

4.2.1 - Measuring purity. Duration: 7 minutes7 min

4.2.2 - Choosing a split: Information Gain. Duration: 11 minutes11 min

4.2.3 - Putting it together. Duration: 9 minutes9 min

4.2.4 - Using one-hot encoding of categorical features. Duration: 5 minutes5 min

4.2.5 - Continuous valued features. Duration: 6 minutes6 min

4.2.6 - Regression Trees (optional). Duration: 9 minutes9 min

Lab: Optional Lab: Decision Trees. Duration: 30 minutes30 min

Practice quiz: Decision tree learning

4.3 - Tree ensembles

4.3.1 - Using multiple decision trees. Duration: 3 minutes3 min

4.3.2 - Sampling with replacement. Duration: 3 minutes3 min

4.3.3 - Random forest algorithm. Duration: 6 minutes6 min

4.3.4 - XGBoost. Duration: 6 minutes6 min

4.3.5 - When to use decision trees. Duration: 6 minutes6 min

Lab: Optional Lab: Tree Ensembles. Duration: 30 minutes30 min

Practice quiz: Tree ensembles

Practice Lab: Decision Trees

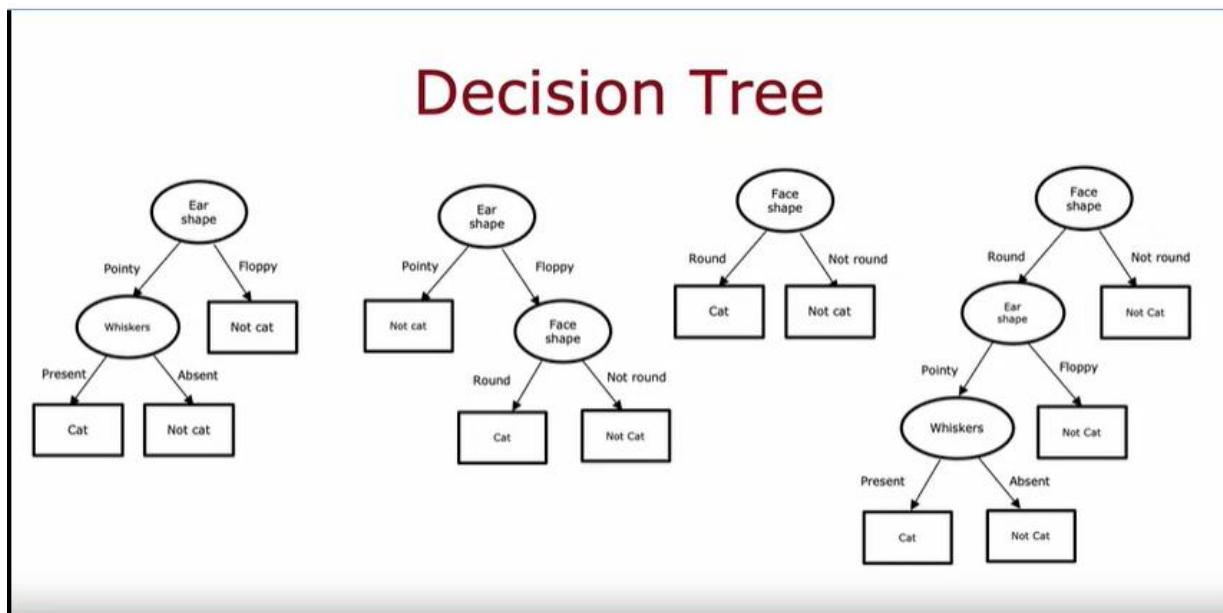
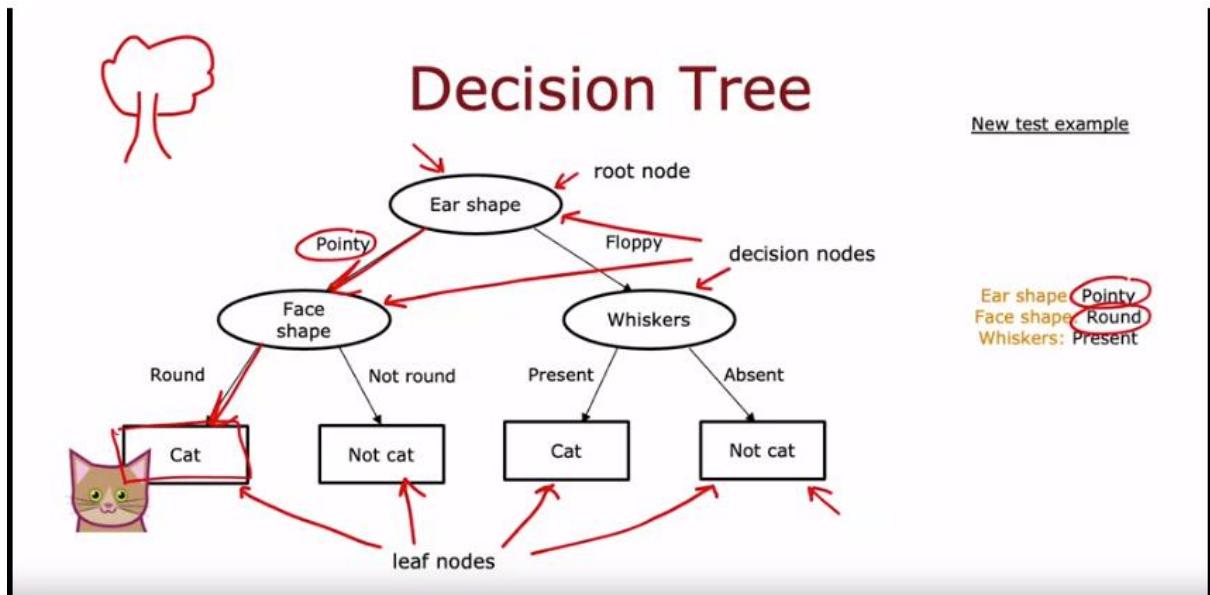
4.4 - Conversations with Andrew (Optional)

Video: VideoAndrew Ng and Chris Manning on Natural Language Processing. Duration: 47 minutes47 min

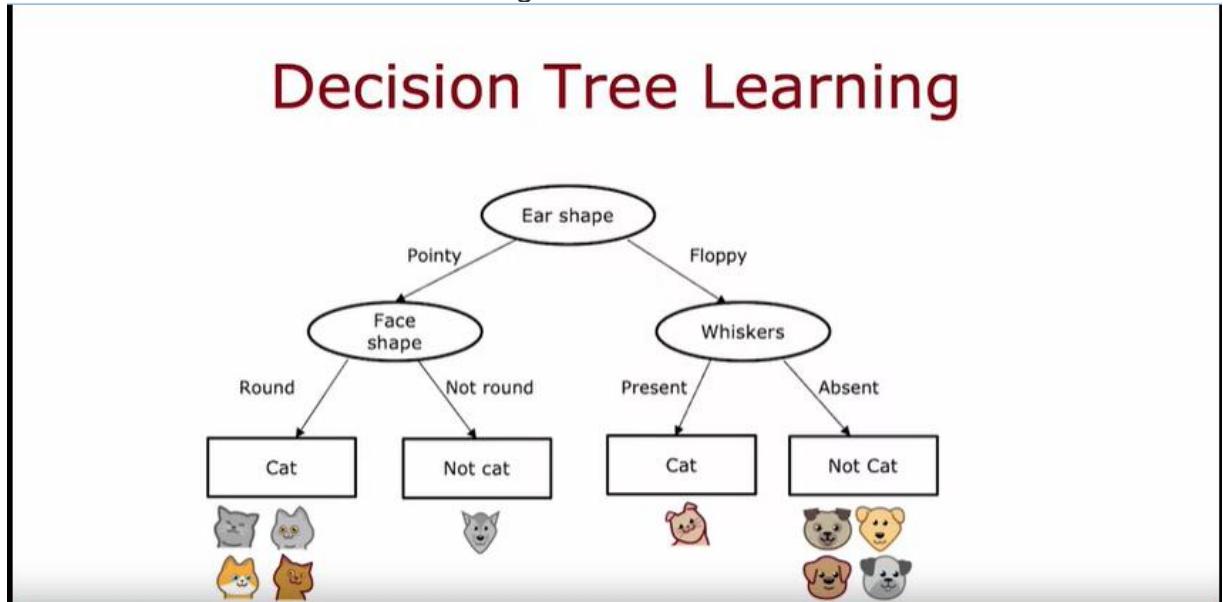
4.3 - Tree ensembles : 4.3.1 - Using multiple decision trees :**4.1 - Decision trees : 4.1.1 - Decision tree model :**

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

Categorical (discrete values) X Y



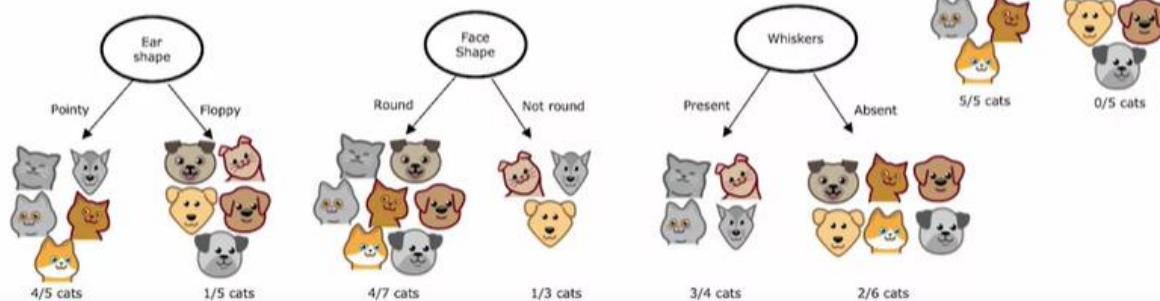
4.1 - Decision trees : 4.1.2 - Learning Process :



Decision Tree Learning

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

Maximize purity (or minimize impurity)

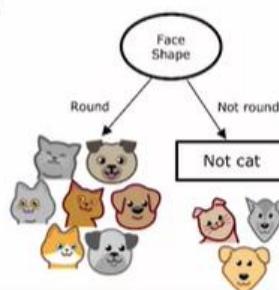


- One reason you might want to limit the depth of the decision tree is to make sure for us to tree doesn't get too big and unwieldy and second, by keeping the tree small, it makes it less prone to overfitting.

Decision Tree Learning

Decision 2: When do you stop splitting?

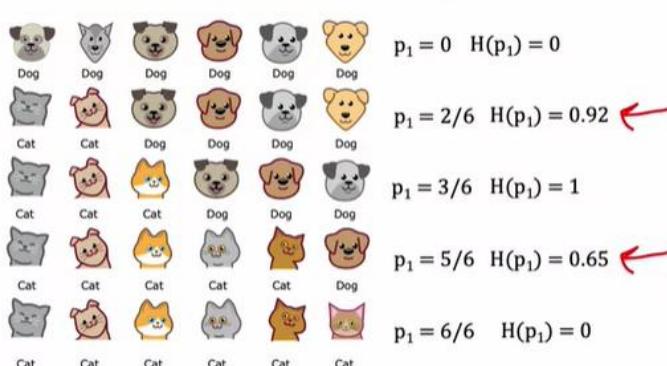
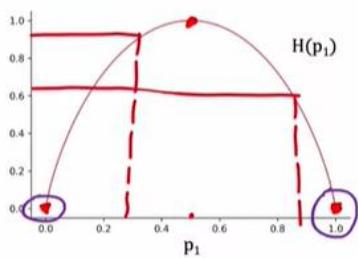
- When a node is 100% one class
- When splitting a node will result in the tree exceeding a maximum depth
- When improvements in purity score are below a threshold
- When number of examples in a node is below a threshold



4.2 - Decision tree learning : 4.2.1 - Measuring purity :

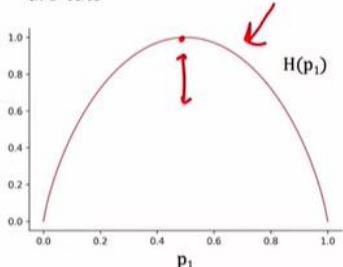
Entropy as a measure of impurity

p_1 = fraction of examples that are cats



Entropy as a measure of impurity

p_1 = fraction of examples that are cats



$$p_0 = 1 - p_1$$

$$\begin{aligned} H(p_1) &= -p_1 \log_2(p_1) - p_0 \log_2(p_0) \\ &= -p_1 \log_2(p_1) - (1 - p_1) \log_2(1 - p_1) \end{aligned}$$

Note: "0 log(0)" = 0

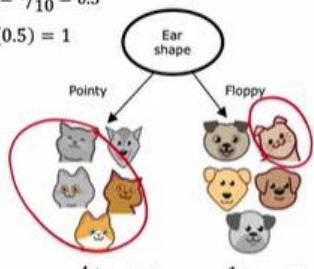
4.2 - Decision tree learning : 4.2.2 - Choosing a split :

- **Information gain** measures the reduction in entropy that you get in your tree resulting from making a split.
- Avoid to carry out the split any further if the reduction in entropy is too small or below a threshold. Otherwise, you're just increasing the size of the tree unnecessarily and risking overfitting by splitting.

Choosing a split

$$p_1 = 5/10 = 0.5$$

$$H(0.5) = 1$$



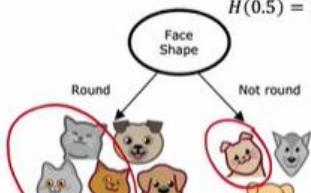
$$p_1 = 4/5 = 0.8 \quad p_1 = 1/5 = 0.2$$

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

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

$$= 0.28$$

$$H(0.5) = 1$$



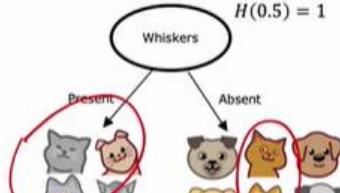
$$p_1 = 4/7 = 0.57 \quad p_1 = 1/3 = 0.33$$

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

$$H(0.5) - \left(\frac{7}{10} H(0.57) + \frac{3}{10} H(0.33) \right)$$

$$= 0.03$$

$$H(0.5) = 1$$



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

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

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

$$= 0.12$$

Information gain

- weightage

$$w_{lef}= 5/10 \quad w_{rig}=5/10$$

$$w_{lef}= 7/10 \quad w_{rig}=3/10$$

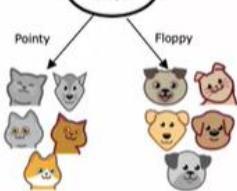
$$w_{lef}= 4/10 \quad w_{rig}=6/10$$

- pick the one that gives you the highest information gain

Information Gain



$$p_1^{\text{root}} = 5/10 = 0.5$$



$$\begin{aligned} p_1^{\text{left}} &= 4/5 & p_1^{\text{right}} &= 1/5 \\ w^{\text{left}} &= 5/10 & w^{\text{right}} &= 5/10 \end{aligned}$$

Information gain

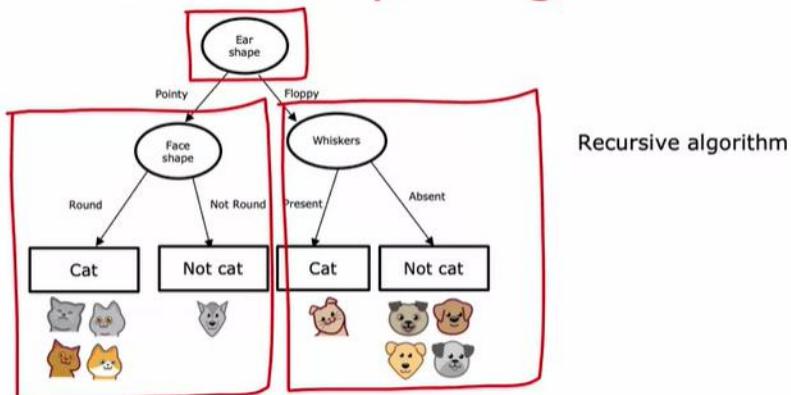
$$= H(p_1^{\text{root}}) - \left(w^{\text{left}} H(p_1^{\text{left}}) + w^{\text{right}} H(p_1^{\text{right}}) \right)$$

4.2 - Decision tree learning : 4.2.3 - Putting it together :

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

Recursive splitting

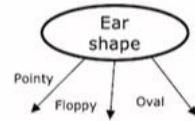


4.2 - Decision tree learning : 4.2.4 - Using one-hot encoding of categorical features :

Features with three possible values

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

3 possible values



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

One hot encoding

If a categorical feature can take on k values, create k binary features (0 or 1 valued).

One hot encoding and neural networks

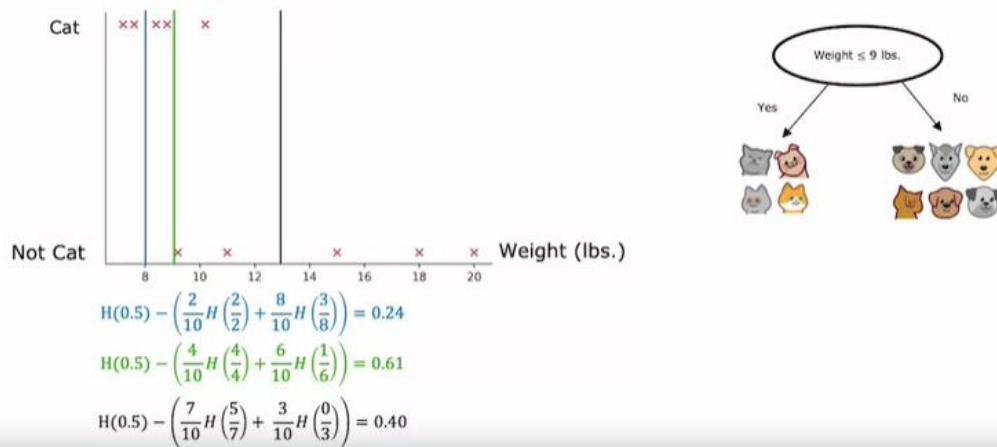
Pointy ears	Floppy ears	Round ears	Face shape	Whiskers	Cat
1	0	0	Round 1	Present 1	1
0	0	1	Not round 0	Present 1	1
0	0	1	Round 1	Absent 0	0
1	0	0	Not round 0	Present 1	0
0	0	1	Round 1	Present 1	1
1	0	0	Round 1	Absent 0	1
0	1	0	Not round 0	Absent 0	1
0	0	1	Round 1	Absent 0	1
0	1	0	Round 1	Absent 0	1
0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

4.2 - Decision tree learning : 4.2.5 - Continuous valued features :

Continuous features ↴

Ear shape	Face shape	Whiskers	Weight (lbs.)	Cat
Pointy	Round	Present	7.2	1
Floppy	Not round	Present	8.8	1
Floppy	Round	Absent	15	0
Pointy	Not round	Present	9.2	0
Pointy	Round	Present	8.4	1
Pointy	Round	Absent	7.6	1
Floppy	Not round	Absent	11	0
Pointy	Round	Absent	10.2	1
Floppy	Round	Absent	18	0
Floppy	Round	Absent	20	0

Splitting on a continuous variable



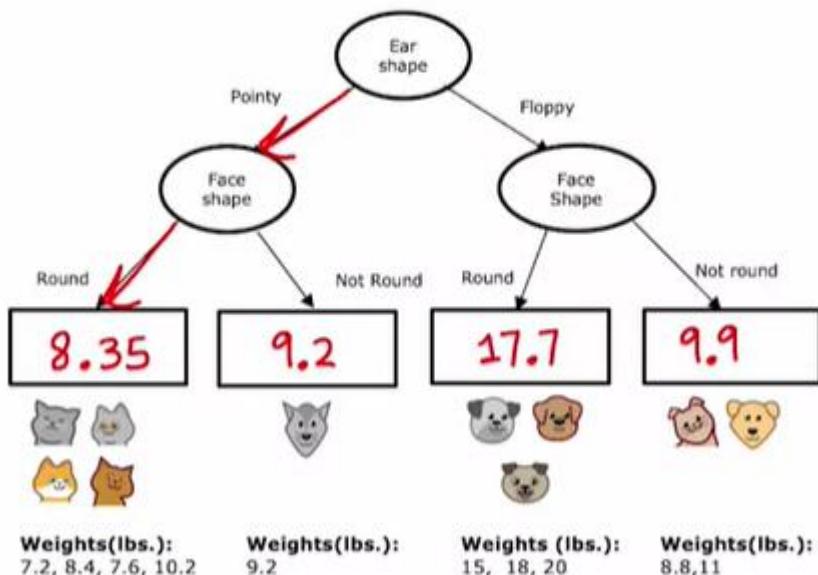
4.2 - Decision tree learning : 4.2.6 - Regression Trees (optional) :

Regression with Decision Trees: Predicting a number

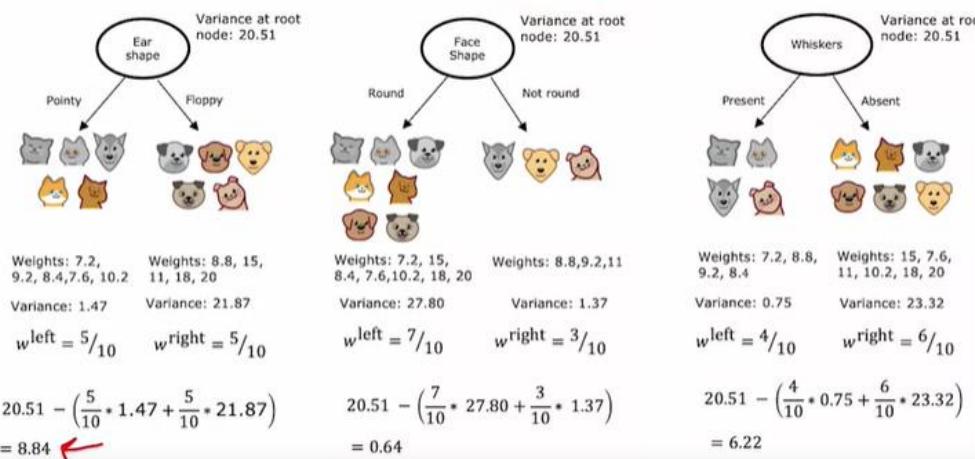
Ear shape	Face shape	Whiskers	Weight (lbs.)
Pointy	Round	Present	7.2
Floppy	Not round	Present	8.8
Floppy	Round	Absent	15
Pointy	Not round	Present	9.2
Pointy	Round	Present	8.4
Pointy	Round	Absent	7.6
Floppy	Not round	Absent	11
Pointy	Round	Absent	10.2
Floppy	Round	Absent	18
Floppy	Round	Absent	20

X Y

Regression with Decision Trees



Choosing a split



4.3 - Tree ensembles : 4.3.1 - Using multiple decision trees :

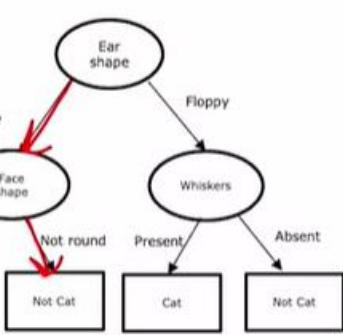
Trees are highly sensitive to small changes of the data



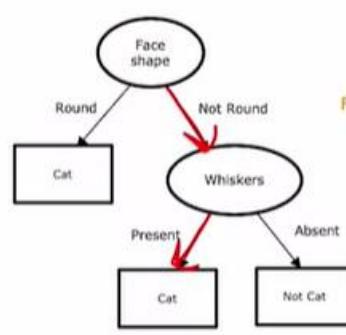
Tree ensemble



Prediction: Cat



Prediction: Not cat



Prediction: Cat

Final prediction: Cat

New test example

Ear shape: Pointy
Face shape: Not Round
Whiskers: Present