

Advanced Learning Algorithms

week1

Neural networks intuition

Neurons and the brain

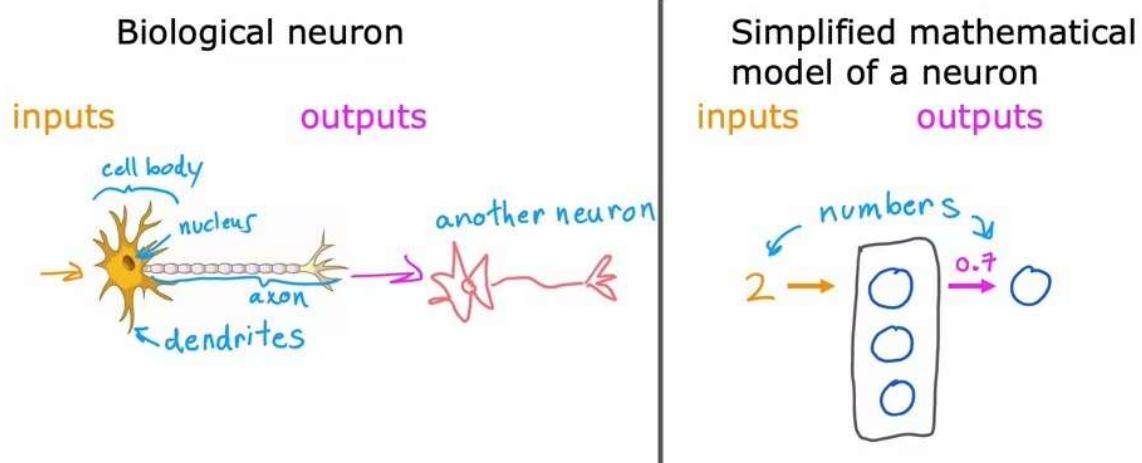
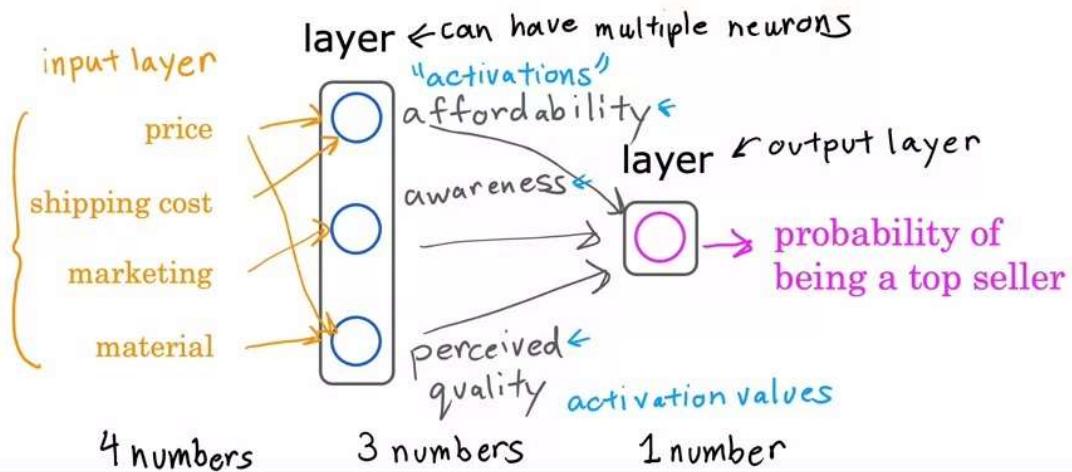


image source: <https://biologydictionary.net/sensory-neuron/>

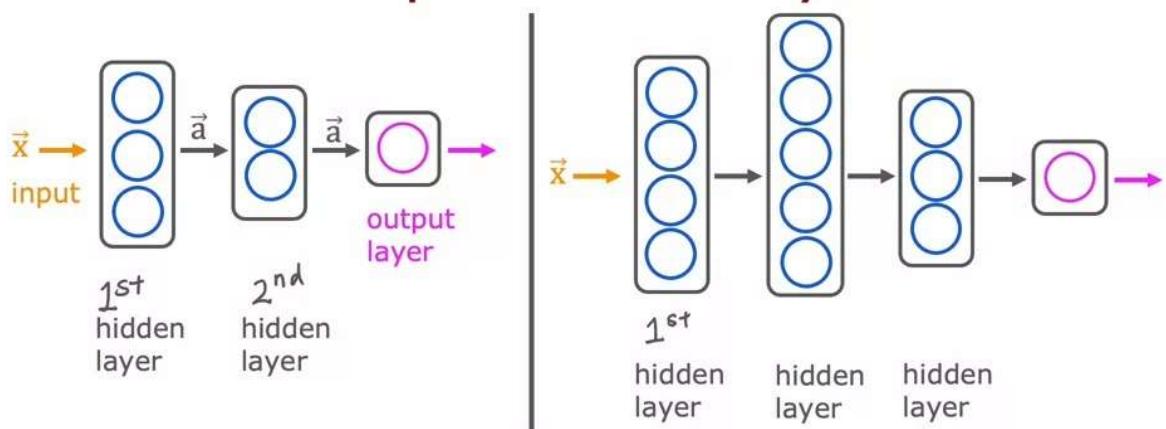


Demand Prediction

Demand Prediction

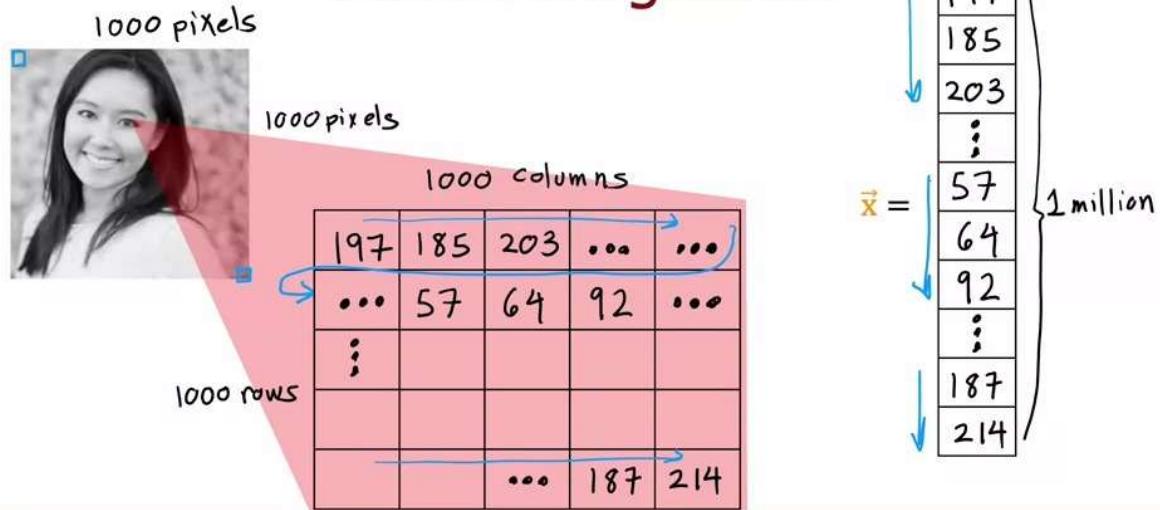


Multiple hidden layers

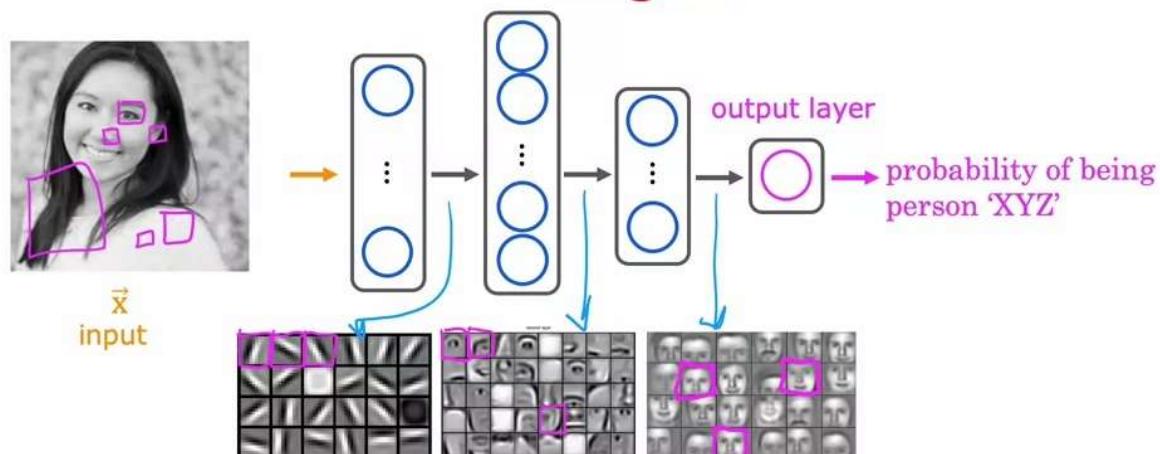


Recognizing Images

Face recognition

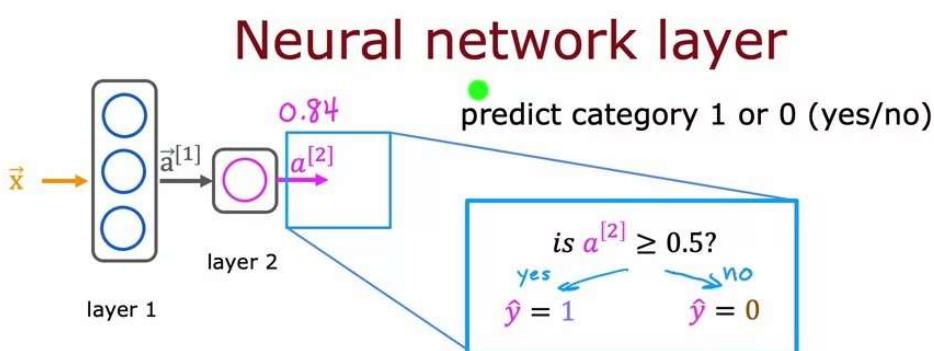
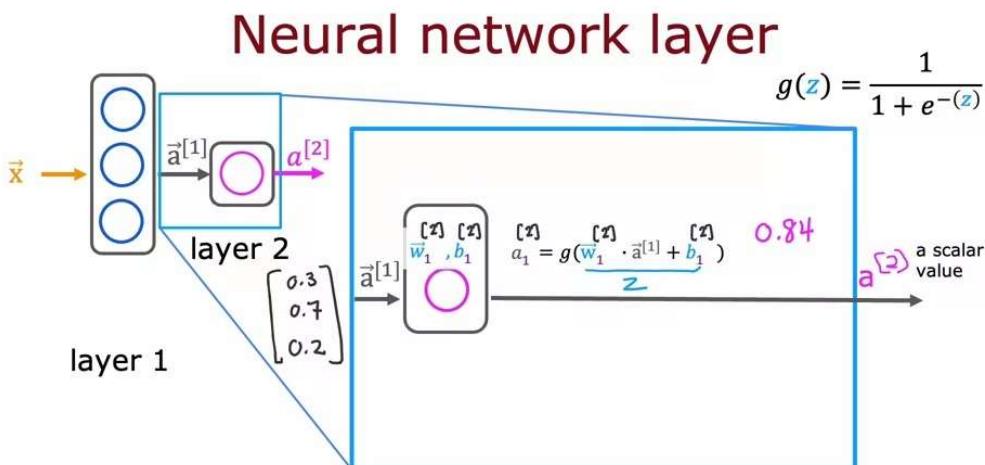
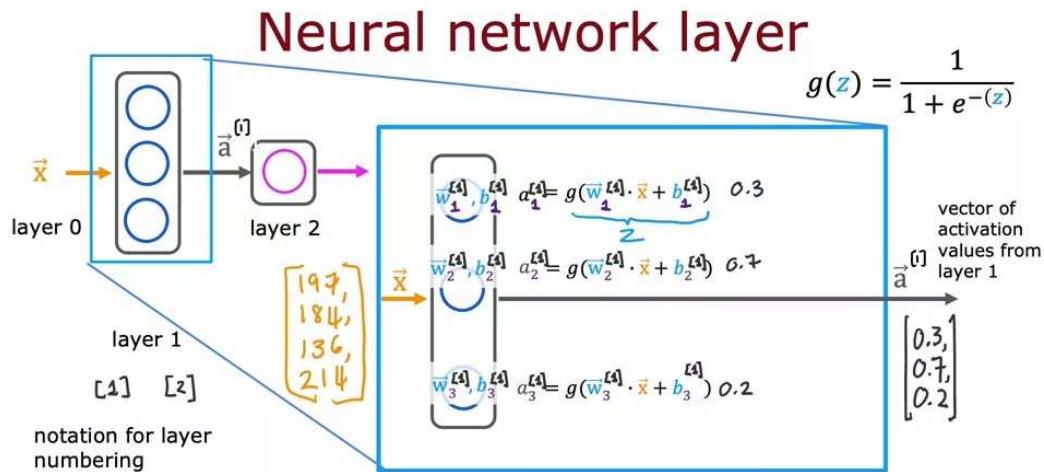


Face recognition



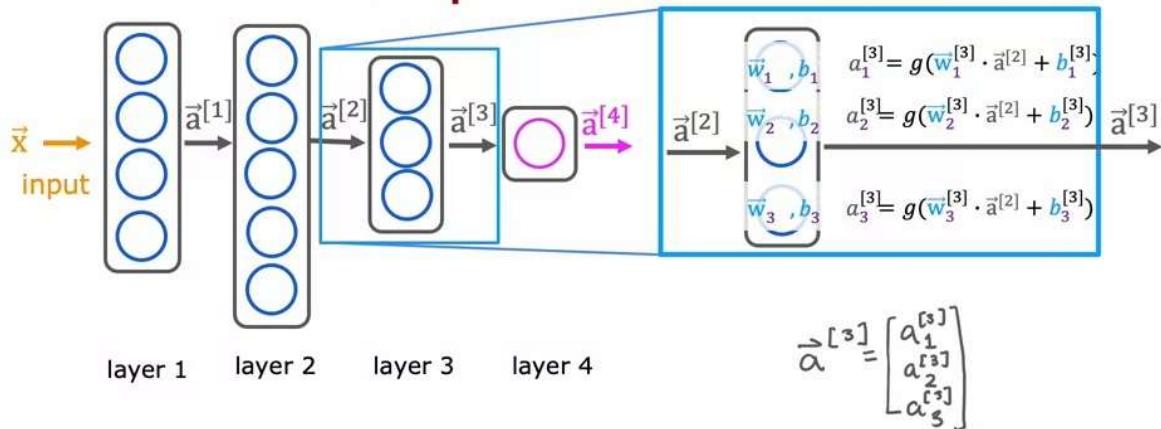
source: Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations
by Honglak Lee, Roger Grosse, Ranganath Andrew Y. Ng

Neural network layer

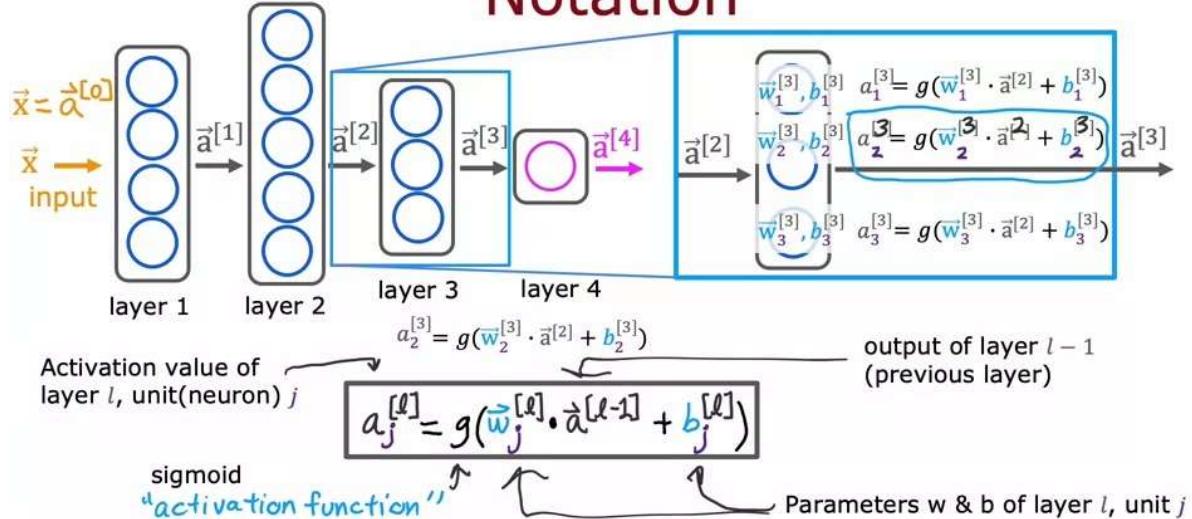


More complex neural networks

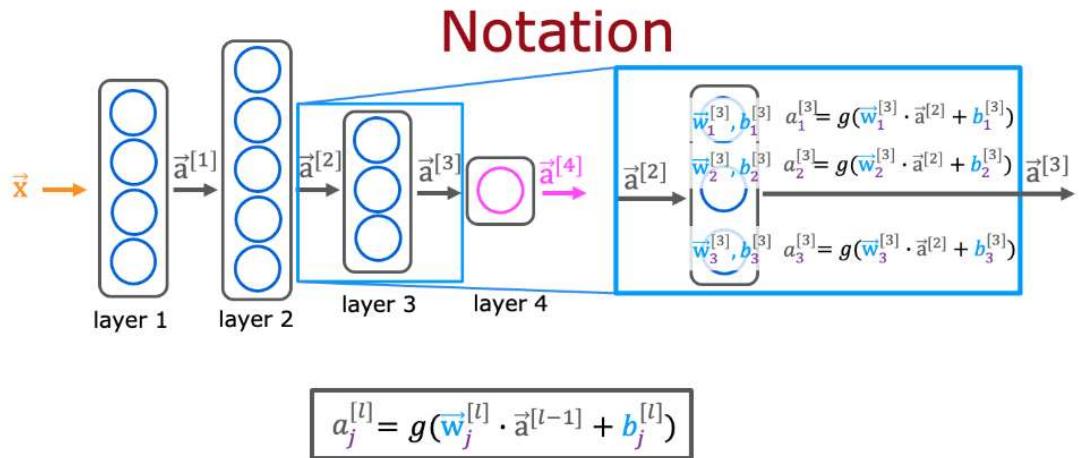
More complex neural network



Notation



1.

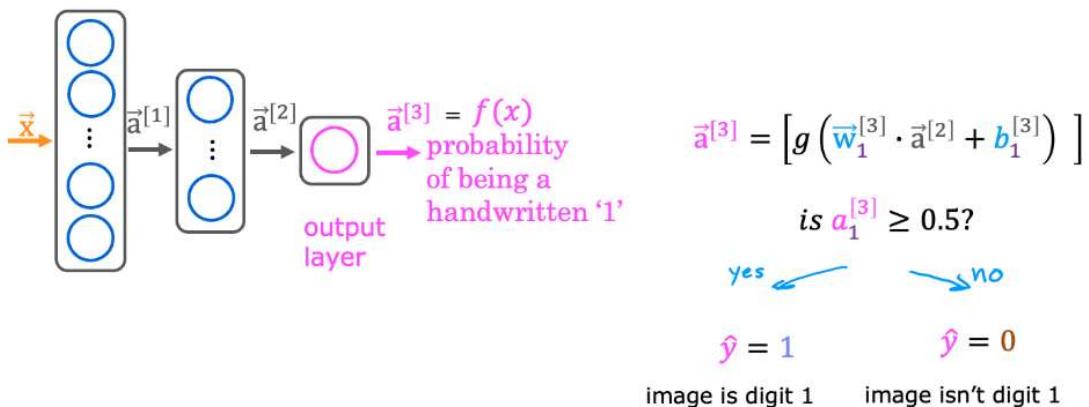


For a neural network, what is the expression for calculating the activation of the third neuron in layer 2? Note, this is different from the question that you saw in the lecture video.

- $a_3^{[2]} = g(\vec{w}_3^{[2]} \cdot \vec{a}^{[2]} + b_3^{[2]})$
- $a_3^{[2]} = g(\vec{w}_3^{[2]} \cdot \vec{a}^{[1]} + b_3^{[2]})$

2.

Handwritten digit recognition

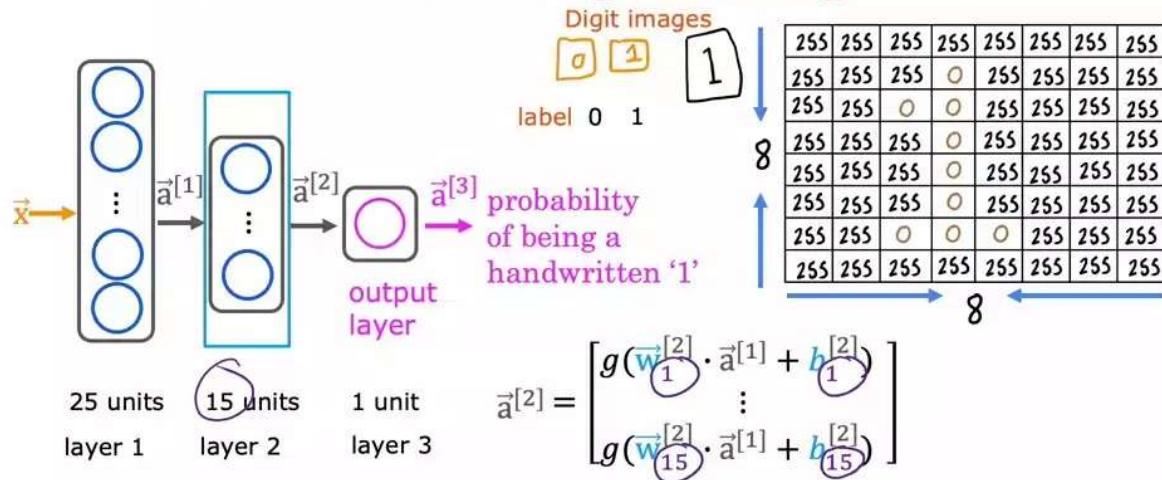


For the handwriting recognition task discussed in lecture, what is the output $a_1^{[3]}$?

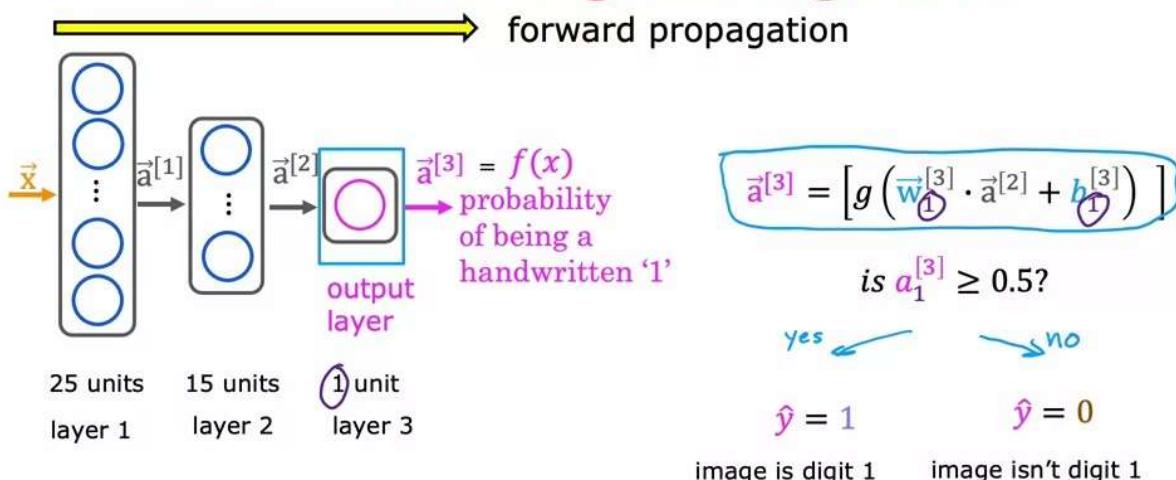
- A number that is either exactly 0 or 1, comprising the network's prediction
- A vector of several numbers, each of which is either exactly 0 or 1
- A vector of several numbers that take values between 0 and 1
- The estimated probability that the input image is of a number 1, a number that ranges from 0 to 1.

Making predictions (forward propagation)

Handwritten digit recognition



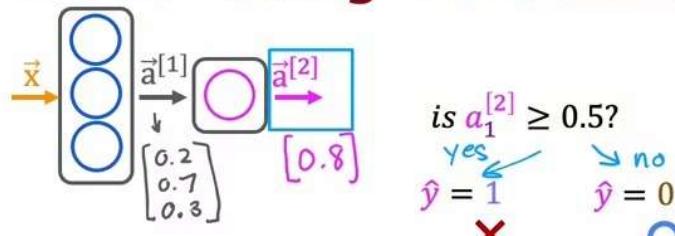
Handwritten digit recognition



TensorFlow implementation

Inference in Code

Build the model using TensorFlow



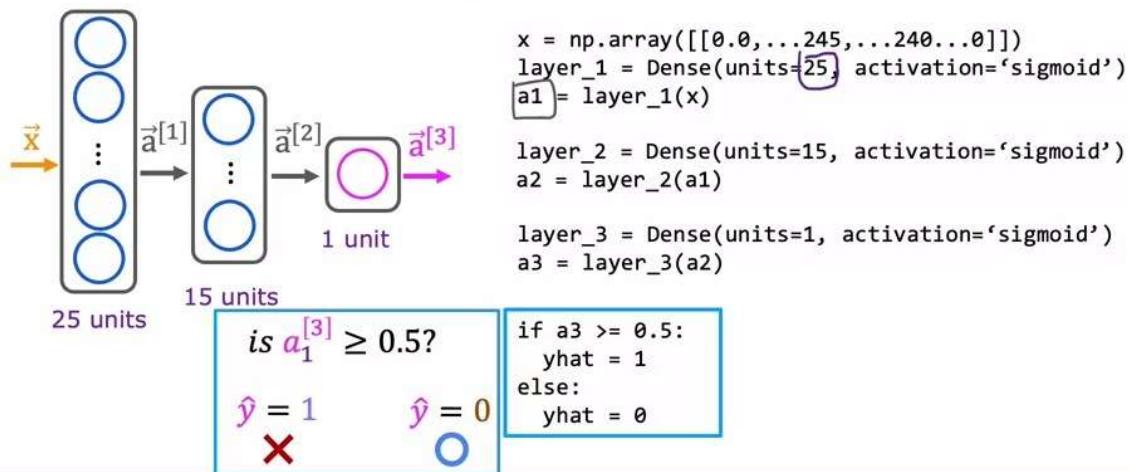
is $a_1^{[2]} \geq 0.5$?
yes $\hat{y} = 1$ no $\hat{y} = 0$

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

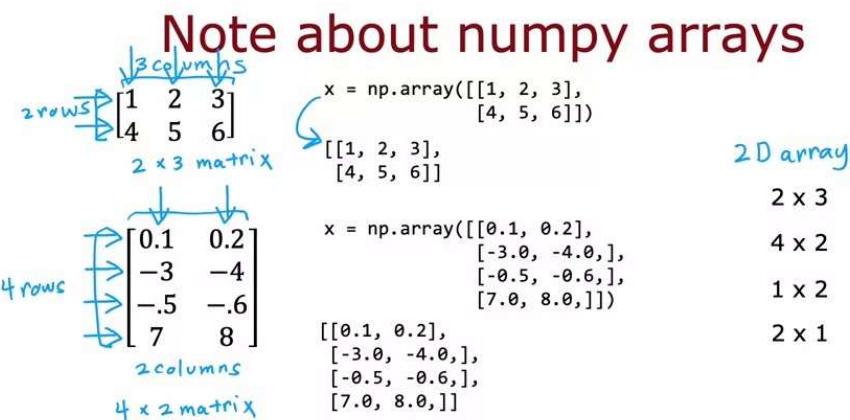
layer_2 = Dense(units=1, activation='sigmoid')
a2 = layer_2(a1)
```

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

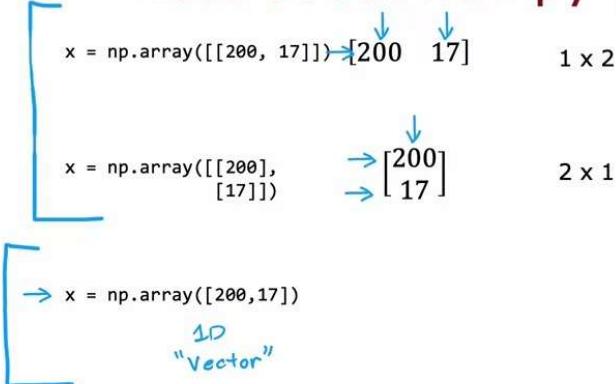
Model for digit classification



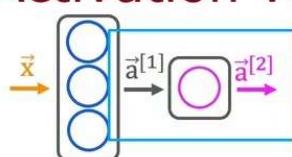
Data in TensorFlow



Note about numpy arrays



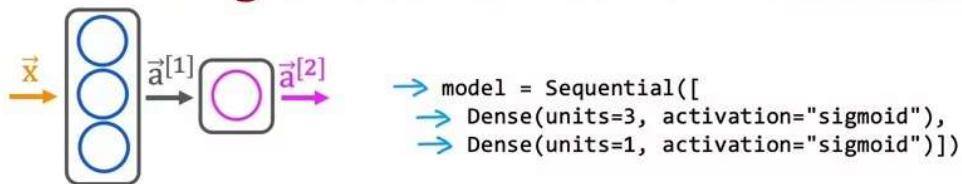
Activation vector



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

Building a neural network

Building a neural network architecture

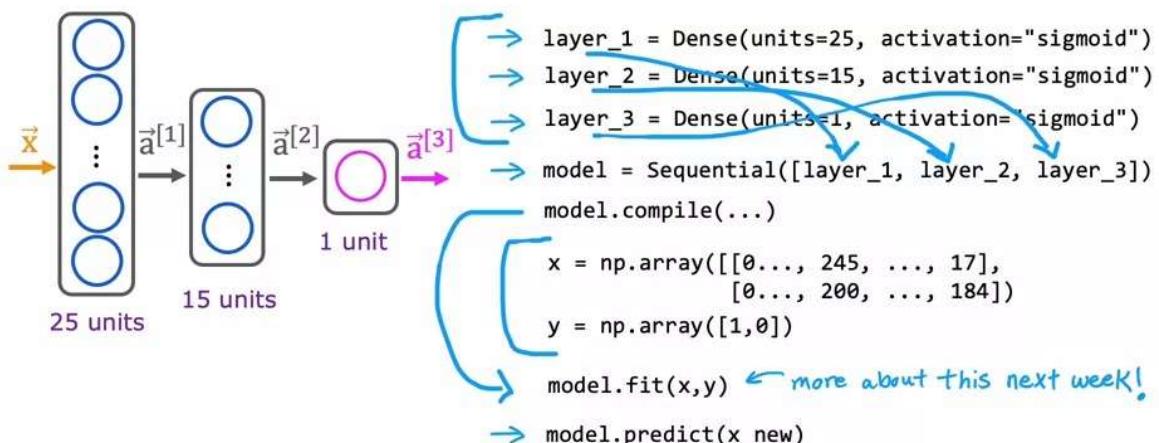


```
→ model = Sequential([
→   Dense(units=3, activation="sigmoid"),
→   Dense(units=1, activation="sigmoid")])
```

		y	
200	17	1	
120	5	0	targets y = np.array([1,0,0,1])
425	20	0	model.compile(...)
212	18	1	model.fit(x,y) ← more about this next week!

```
→ model.predict(x_new) ←
```

Digit classification model



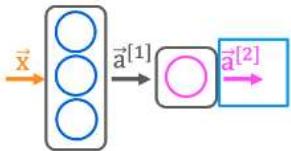
```
→ layer_1 = Dense(units=25, activation="sigmoid")
→ layer_2 = Dense(units=15, activation="sigmoid")
→ layer_3 = Dense(units=1, activation="sigmoid")
→ model = Sequential([layer_1, layer_2, layer_3])
model.compile(...)
x = np.array([[0..., 245, ..., 17],
[0..., 200, ..., 184]])
y = np.array([1,0])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

1. For the the following code:

```
model = Sequential([
    Dense(units=25, activation="sigmoid"),
    Dense(units=15, activation="sigmoid"),
    Dense(units=10, activation="sigmoid"),
    Dense(units=1, activation="sigmoid")])
```

This code will define a neural network with how many layers?

- 3
- 5
- 25
- 4



```
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)
```

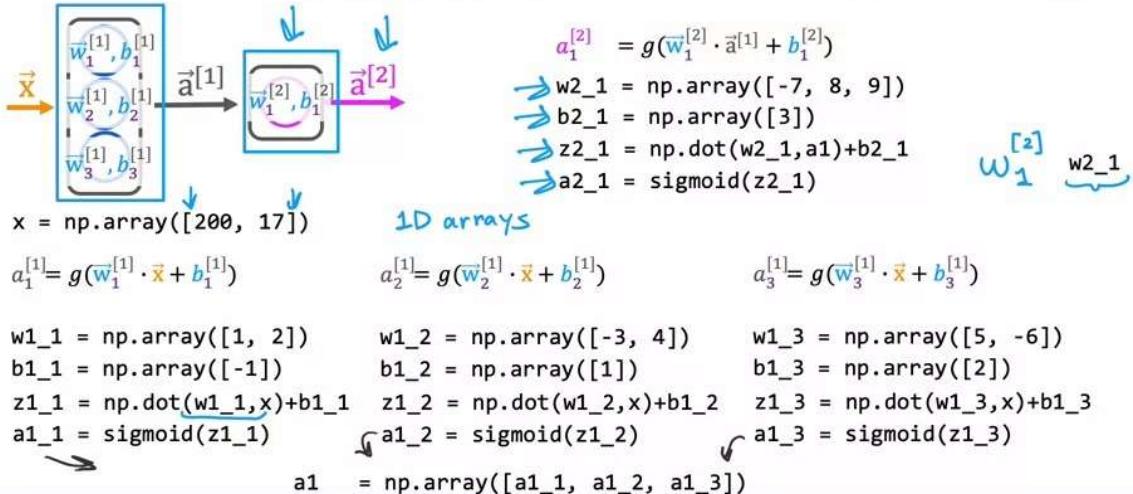
How do you define the second layer of a neural network that has 4 neurons and a sigmoid activation?

- `Dense(layer=2, units=4, activation = 'sigmoid')`
- `Dense(units=4, activation='sigmoid')`
- `Dense(units=4)`
- `Dense(units=[4], activation=['sigmoid'])`

Neural network implementation in Python

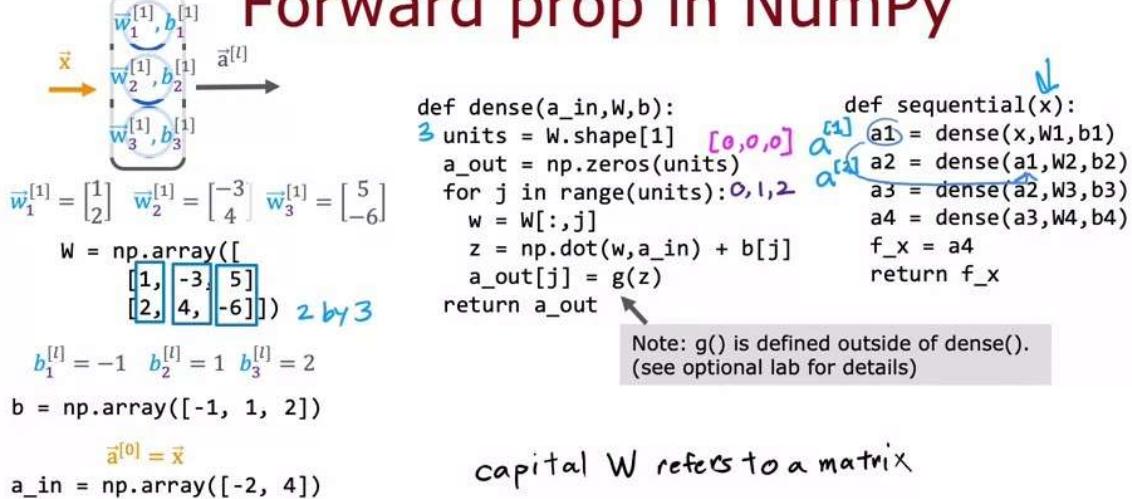
Forward prop in a single layer

forward prop (coffee roasting model)

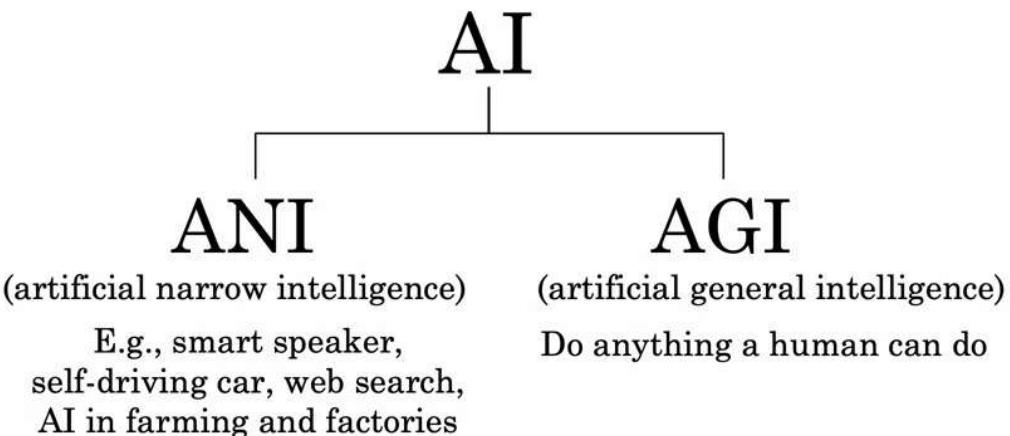


General implementation of forward propagation

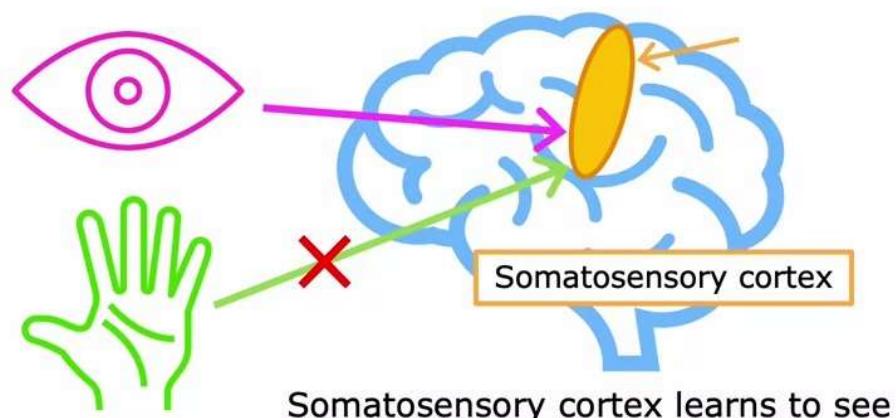
Forward prop in NumPy



Artificial general intelligence (AGI)



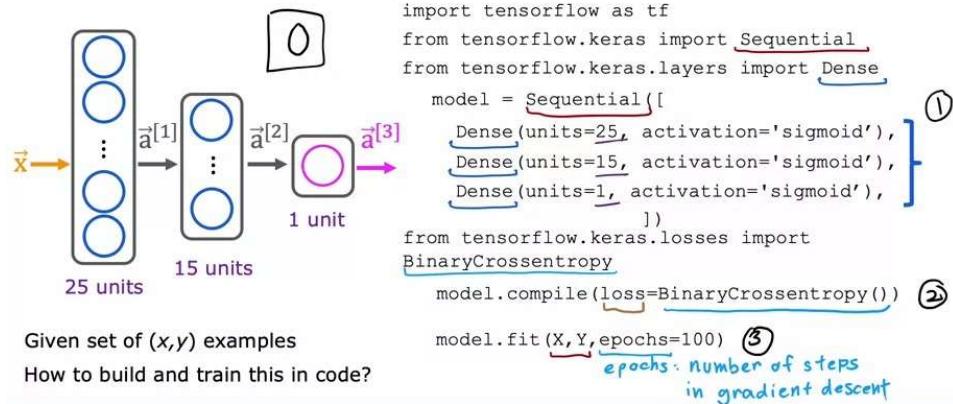
The "one learning algorithm" hypothesis



Week 2

Neural Network Training TensorFlow implementation

Train a Neural Network in TensorFlow



DeepLearning.AI Stanford ONLINE

Andrew Ng

Step1 - specify the model

Step2 - compile model

Step3 - train model

Model Training Steps

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

DeepLearning.AI Stanford ONLINE

Andrew Ng

Neural network libraries

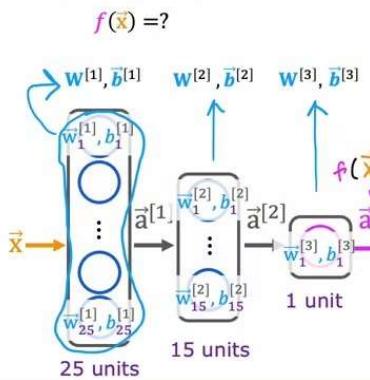
Use code libraries instead of coding "from scratch"



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

1. Create the model

define the model



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

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

2. Loss and cost functions

handwritten digit classification problem

binary classification

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

Compare prediction vs. target

logistic loss
also Known as binary cross entropy

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

$\mathbf{w}^{[1]}, \mathbf{w}^{[2]}, \mathbf{w}^{[3]}$

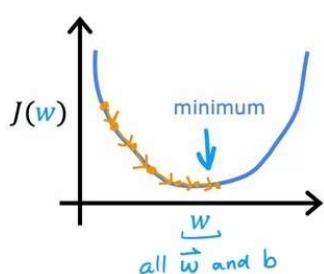
$\mathbf{b}^{[1]}, \mathbf{b}^{[2]}, \mathbf{b}^{[3]}$

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

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

regression
(predicting numbers mean squared error
and not categories)
model.compile(loss= MeanSquaredError())
from tensorflow.keras.losses import
MeanSquaredError
```

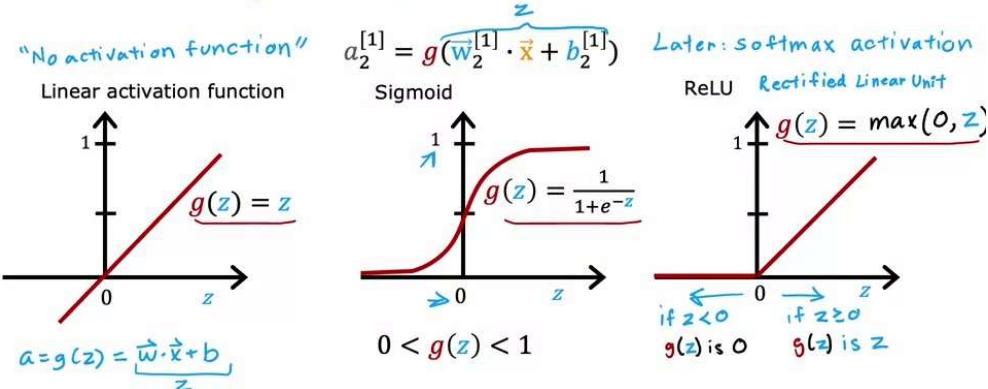
3. Gradient descent



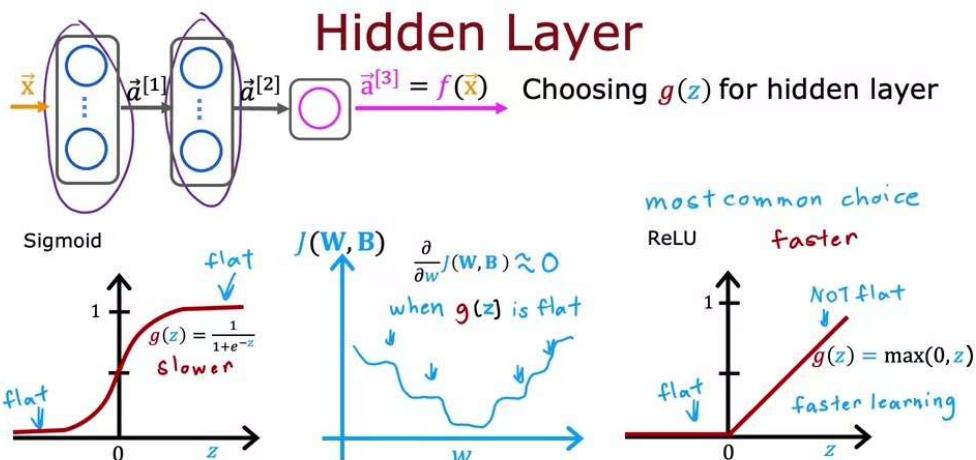
```
repeat {
     $w_j^{[l]} = w_j^{[l]} - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$ 
     $b_j^{[l]} = b_j^{[l]} - \alpha \frac{\partial}{\partial b_j} J(\vec{w}, b)$ 
} Compute derivatives
for gradient descent
using "backpropagation"
model.fit(X, y, epochs=100)
```

Activation Functions

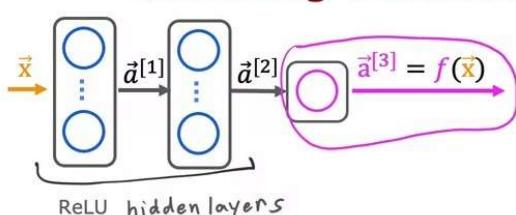
Examples of Activation Functions



Hidden Layer



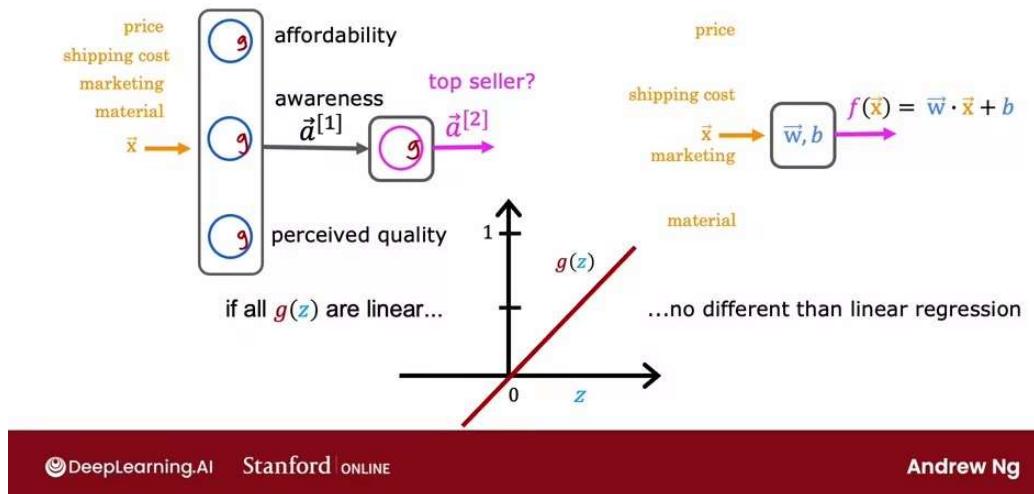
Choosing Activation Summary



binary classification
activation='sigmoid'
regression y negative/
activation='linear' positive
regression y ≥ 0
activation='relu'

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

Why do we need activation functions?



Linear Example

one feature
 x

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

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

$a^{[1]} = \underbrace{w_1^{[1]} x}_{w} + \underbrace{b_1^{[1]}}_{b}$

$a^{[2]} = \underbrace{w_1^{[2]} a^{[1]}}_{w} + \underbrace{b_1^{[2]}}_{b}$
= $w_1^{[2]} (\underbrace{w_1^{[1]} x + b_1^{[1]}}_{w}) + b_1^{[2]}$

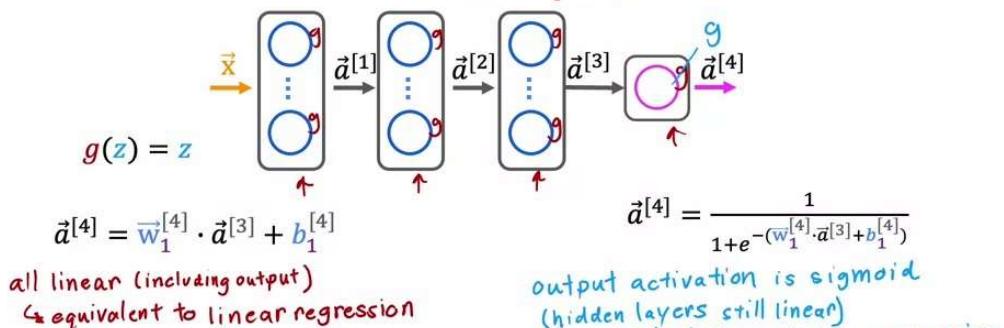
$\vec{a}^{[2]} = (\underbrace{\vec{w}_1^{[2]} \vec{w}_1^{[1]}}_{w}) x + \underbrace{\vec{w}_1^{[2]} b_1^{[1]} + b_1^{[2]}}_{b}$

$\vec{a}^{[2]} = w x + b$

$f(x) = wx + b$ linear regression

DeepLearning.AI Stanford ONLINE Andrew Ng

Example



Multiclass Classification

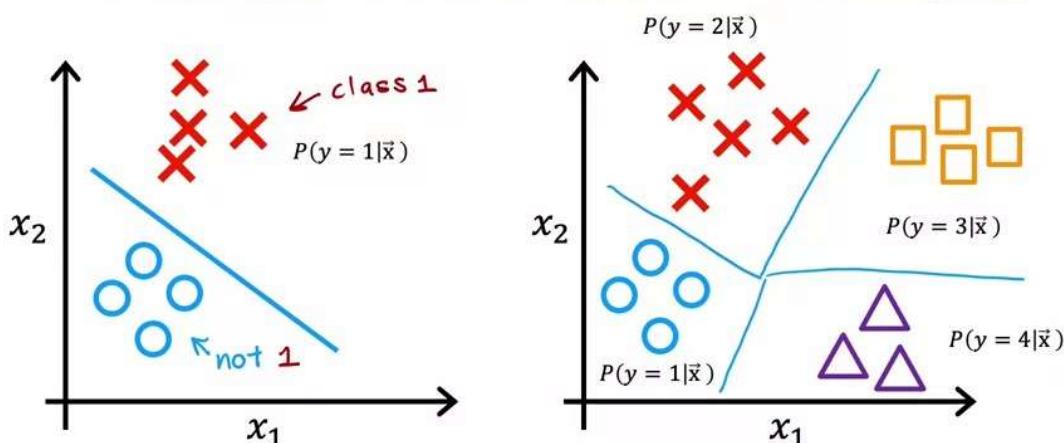
MNIST example

0 1 2 3 4 5 6 7 8 9
 $y = 0$ 1 2 3 4 5 6 7 8 9

 $y = 7$

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

Multiclass classification example



Softmax

Logistic regression
(2 possible output values)
 $z = \vec{w} \cdot \vec{x} + b$

$$\text{X } \alpha_1 = g(z) = \frac{1}{1+e^{-z}} = P(y=1|\vec{x})$$

$$\text{O } \alpha_2 = 1 - \alpha_1 = P(y=0|\vec{x})$$

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) $y=1, 2, 3, 4$

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

X O □ △

$$= P(y=1|\vec{x}) \text{ 0.30}$$

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

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

$$\Delta z_4 = \vec{w}_4 \cdot \vec{x} + b_4 \quad a_4 = \frac{e^{z_4}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}} = P(y=4|\vec{x}) \approx 35$$

Cost

Logistic regression

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

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

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

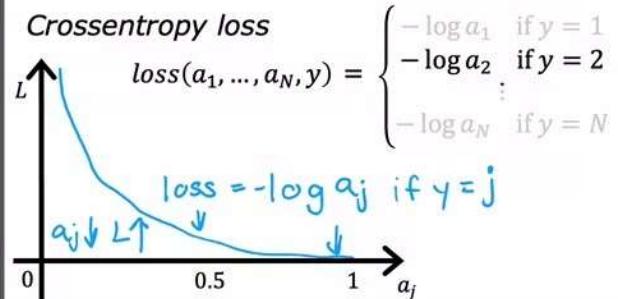
$$J(\vec{w}, b) = \text{average loss}$$

Softmax regression

$$\begin{aligned} a_1 &= \frac{e^{z_1}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} &= P(y = 1 | \bar{x}) \\ &\vdots \\ a_N &= \frac{e^{z_N}}{e^{z_1} + e^{z_2} + \dots + e^{z_N}} &= P(y = N | \bar{x}) \end{aligned}$$

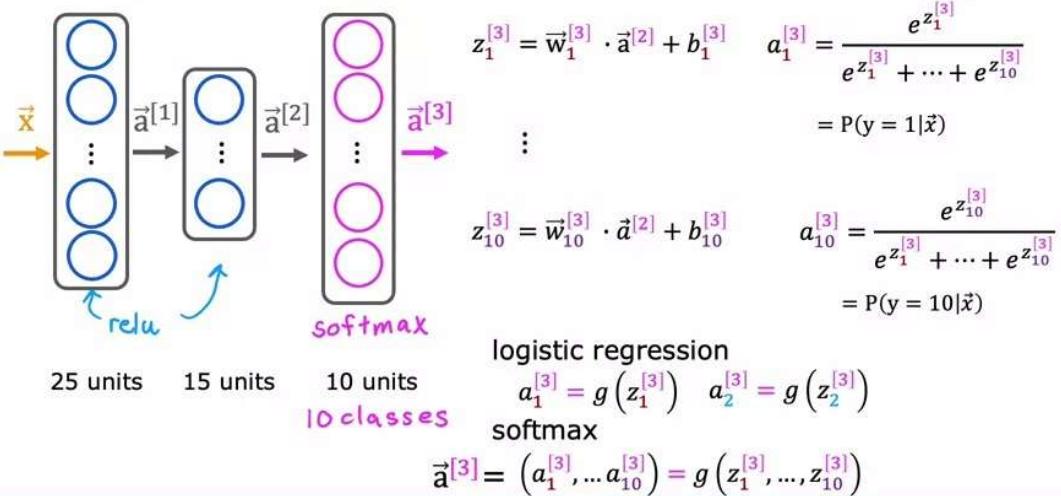
Crossentropy loss

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



Neural Network with Softmax output

Neural Network with Softmax output



① specify the model

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

MNIST with softmax

```
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

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

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

Note: better (recommended) version later.
 Don't use the version shown here!

Improved implementation of softmax

Numerical Roundoff Errors

option 1

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

option 2

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

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

Logistic regression:

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

Original loss

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

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

More accurate loss (in code)

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

$$1 + \frac{1}{10,000} \quad 1 - \frac{1}{10,000}$$

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

logit: z

More numerically accurate implementation of softmax

Softmax regression

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

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

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

More Accurate

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

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

MNIST (more numerically accurate)

```
model    import tensorflow as tf
         from tensorflow.keras import Sequential
         from tensorflow.keras.layers import Dense
         model = Sequential([
             Dense(units=25, activation='relu'),
             Dense(units=15, activation='relu'),
             Dense(units=10, activation='linear') ])
loss     from tensorflow.keras.losses import
         SparseCategoricalCrossentropy
         model.compile(..., loss=SparseCategoricalCrossentropy(from_logits=True) )
fit      model.fit(X, Y, epochs=100)
predict  logits = model(X) ← not  $a_1 \dots a_{10}$ 
         is  $z_1 \dots z_{10}$ 
         f_x = tf.nn.softmax(logits)
```

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
         model.compile(..., BinaryCrossentropy(from_logits=True))
         model.fit(X, Y, epochs=100)
fit      logit = model(X)  $\rightarrow$ 
predict  f_x = tf.nn.sigmoid(logit)
```

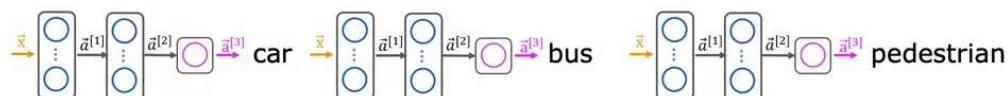
Multilabel classification

Multi-label Classification

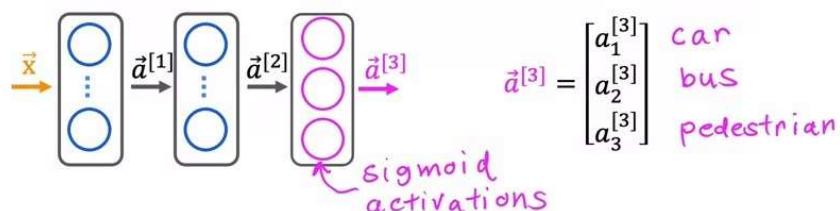


Is there a car?	yes	$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$	no	$y = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$	yes	$y = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$
Is there a bus?	no	$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$	no	$y = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$	yes	$y = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$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

Multi-label Classification



Alternatively, train one neural network with three outputs

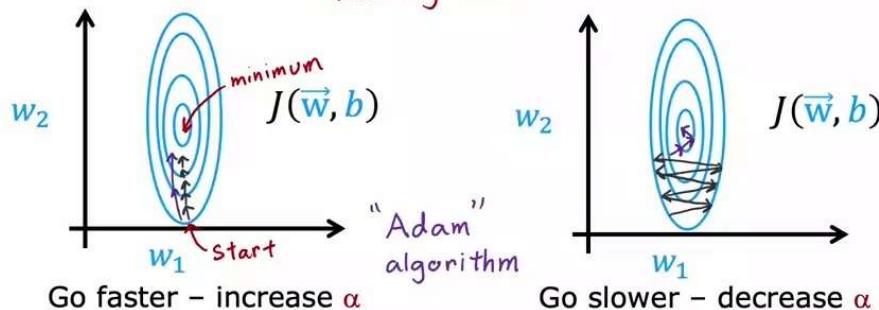


Advanced Optimization

Gradient Descent

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

learning rate

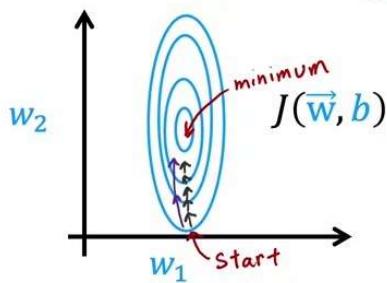


Adam Algorithm Intuition

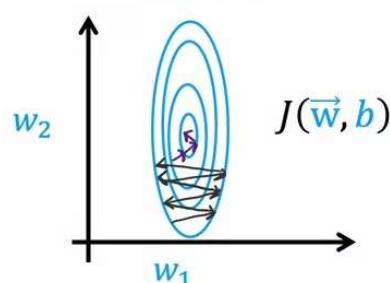
Adam: Adaptive Moment estimation *not just one α*

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

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')
])

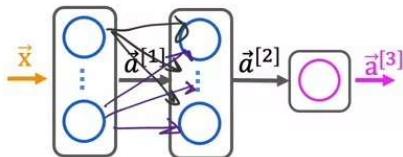
compile
         $\alpha = 10^{-3} = 0.001$ 
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

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

Additional Layer Types

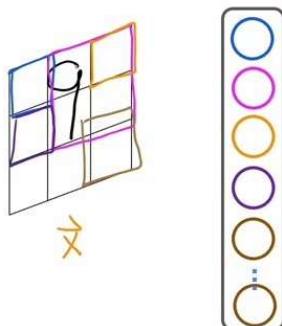
Dense Layer



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

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

Convolutional Layer

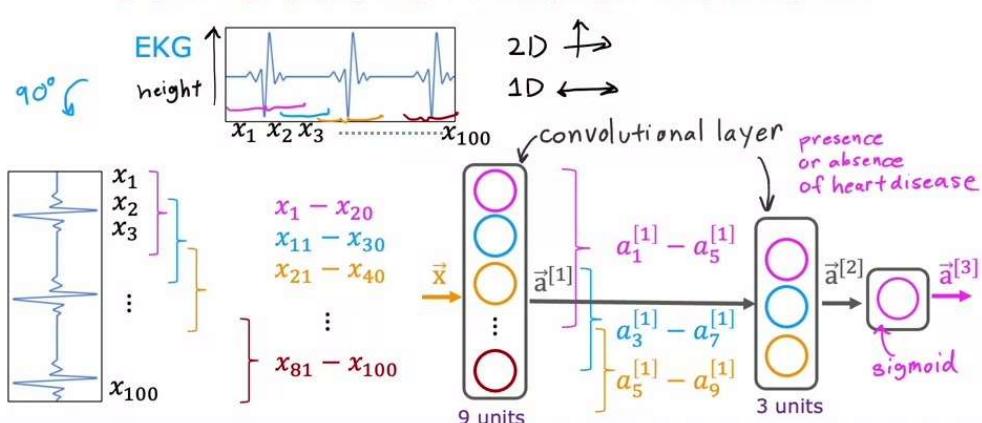


Each neuron only looks at part of the previous layer's outputs.

Why?

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

Convolutional Neural Network



Week 3

Advice for applying machine Learning

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 ($x_1^2, x_2^2, x_1x_2, etc$)
- Try decreasing λ
- Try increasing λ



Machine learning diagnostic

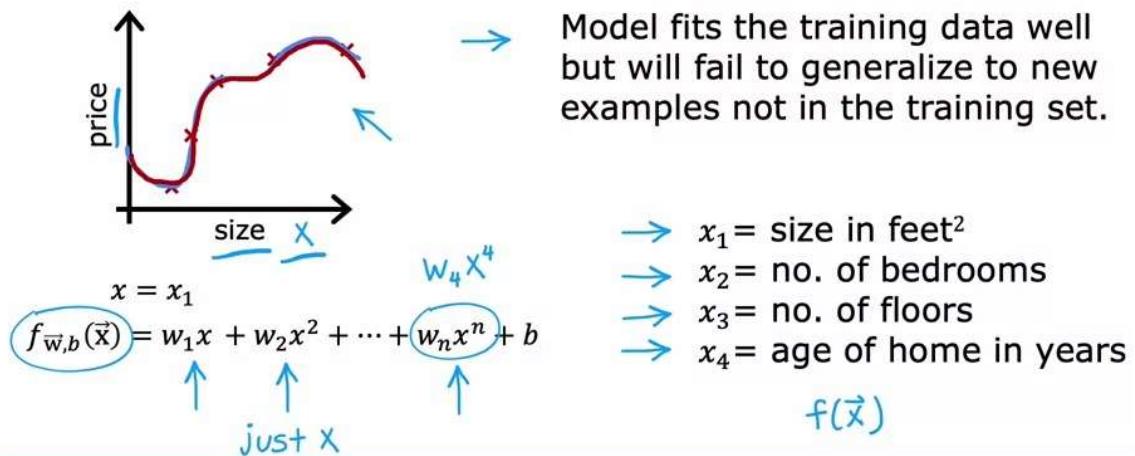
Diagnostic:

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.

Evaluating a model

Evaluating your model



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

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

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

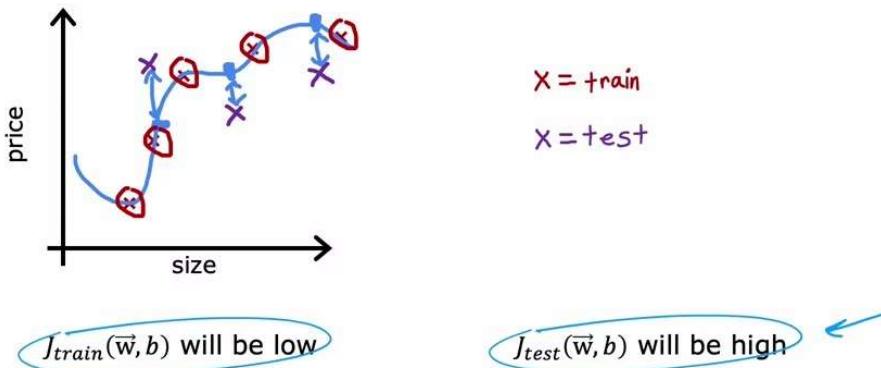
Compute test error:

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

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

0 / 1

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

E.g.,

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

Compute test error:

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

Compute train error:

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

Train/test procedure for classification problem

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

count $\hat{y} \neq y$

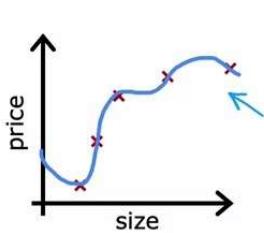
$$\hat{y} = \begin{cases} 1 & \text{if } f_{\vec{w}, b}(\vec{x}^{(i)}) \geq 0.5 \\ 0 & \text{if } f_{\vec{w}, b}(\vec{x}^{(i)}) < 0.5 \end{cases}$$

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

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

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.

$$x = x_1$$

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

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. $f_{\vec{w}, b}(\vec{x}) = w_1 x + w_2 x^2 + w_3 x^3 + b \rightarrow w^{<3>} , b^{<3>} \rightarrow J_{test}(w^{<3>} , b^{<3>})$
 - \vdots
 - $d=10$ 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 $w_1 x + \dots + w_5 x^5 + b$ $d=5 J_{test}(w^{<5>} , b^{<5>})$

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		$M_{train} = 6$
		development set	dev set	
2104	400			
1600	330			
2400	369			
1416	232			
3000	540			
1985	300			
1534	315	+ training set 60%		$(x^{(1)}, y^{(1)})$ \vdots $(x^{(m_{train})}, y^{(m_{train})})$
1427	199			
1380	212	cross validation 20%		$(x_{cv}^{(1)}, y_{cv}^{(1)})$ \vdots $(x_{cv}^{(m_{cv})}, y_{cv}^{(m_{cv})})$
1494	243	+ test set 20%		$(x_{test}^{(1)}, y_{test}^{(1)})$ \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: $J_{test}(\vec{w}, b) = \frac{1}{2m_{test}} \left[\sum_{i=1}^{m_{test}} (f_{\vec{w}, b}(\vec{x}_{test}^{(i)}) - y_{test}^{(i)})^2 \right]$

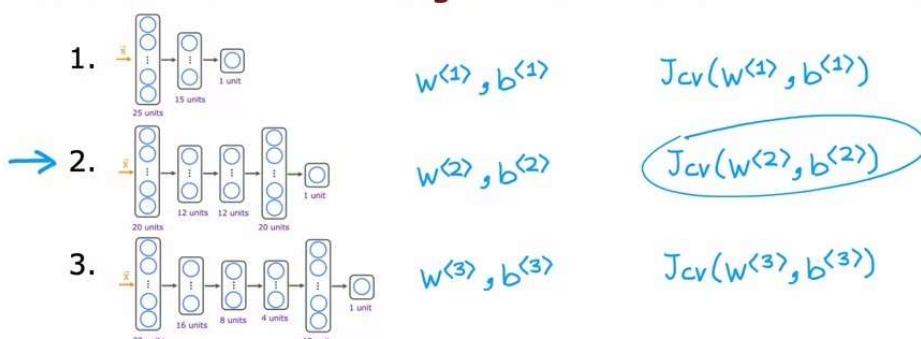
Model selection

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

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

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

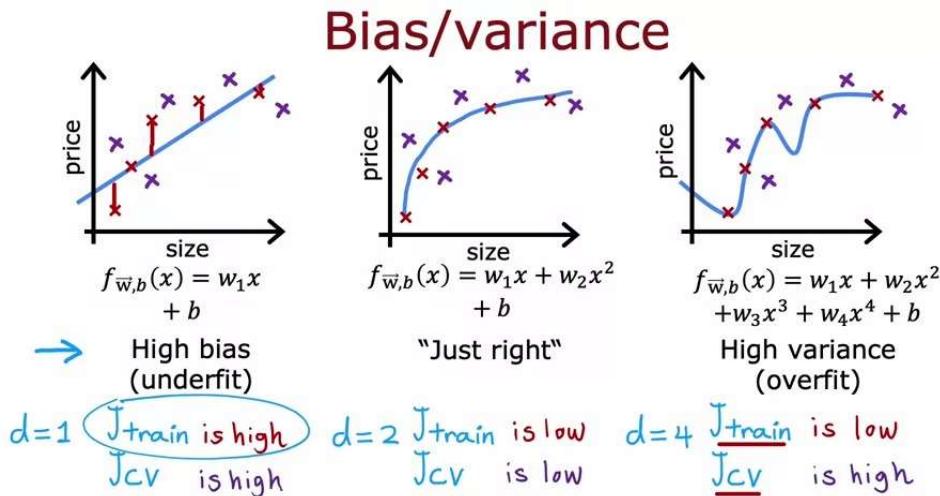
Model selection – choosing a neural network architecture



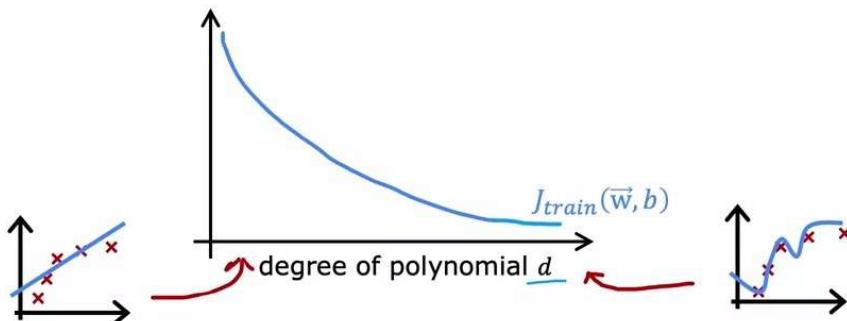
Pick $w^{<2>} , b^{<2>}$

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

Bias and variance

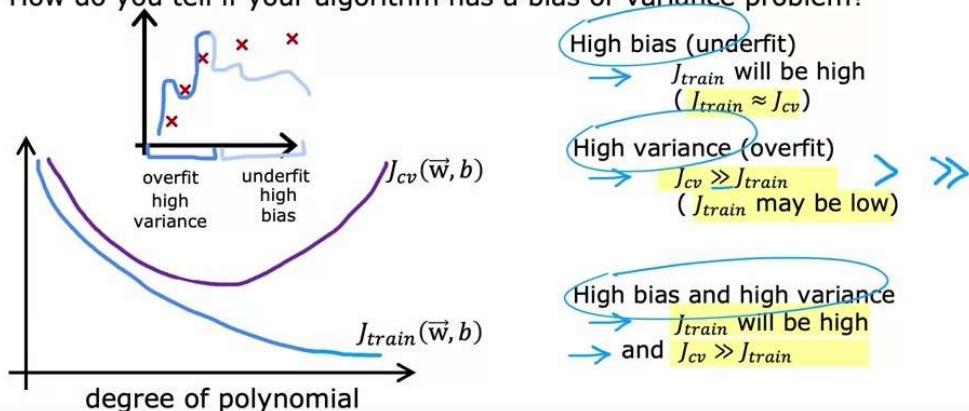


Understanding bias and variance



Diagnosing bias and variance

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

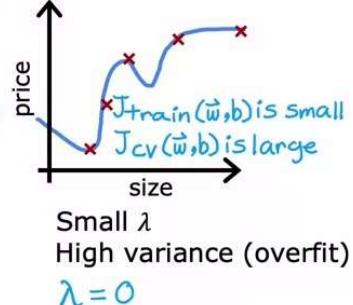
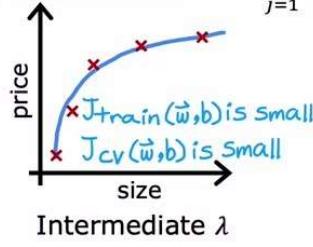
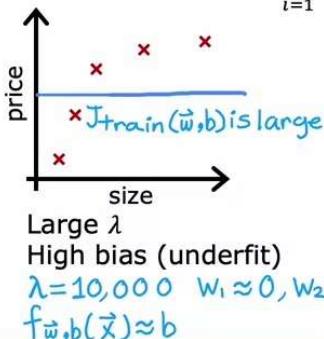


Regularization and bias/variance

Linear regression with regularization

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

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



Choosing the regularization parameter λ

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

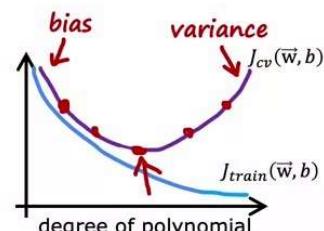
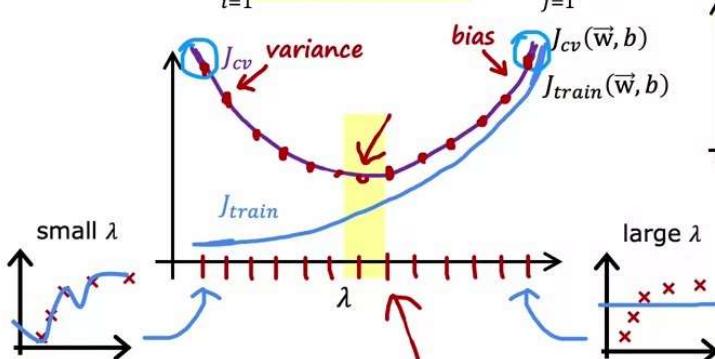
- 1. Try $\lambda = 0$
 - 2. Try $\lambda = 0.01$
 - 3. Try $\lambda = 0.02$
 - 4. Try $\lambda = 0.04$
 - 5. Try $\lambda = 0.08$
 - ⋮
 - 12. Try $\lambda \approx 10$
- $\min_{\vec{w}, b} J(\vec{w}, b)$ → $w^{<1>} b^{<1>} \quad J_{cv}(w^{<1>} b^{<1>})$
 $w^{<2>} b^{<2>} \quad J_{cv}(w^{<2>} b^{<2>})$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

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

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

Bias and variance as a function of regularization parameter λ

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



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%



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%

high
variance

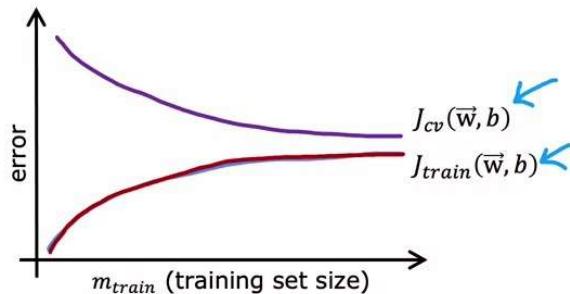
high
bias

Learning curves

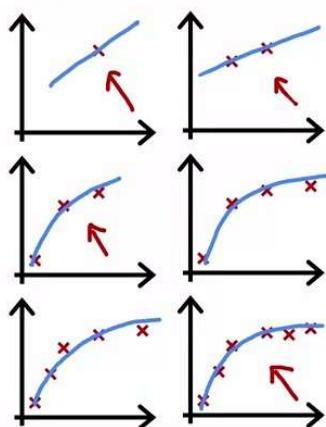
Learning curves

J_{train} = training error

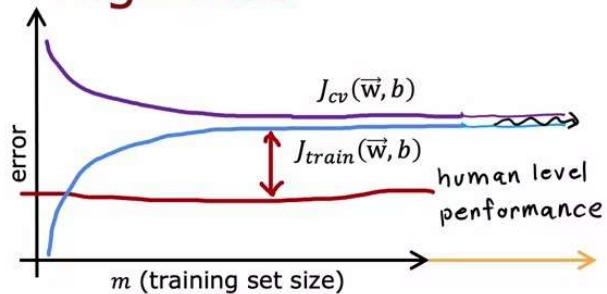
J_{cv} = cross validation error



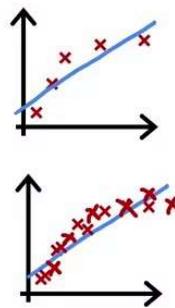
$$f_{\vec{w}, b}(x) = w_1 x + w_2 x^2 + b$$



High bias

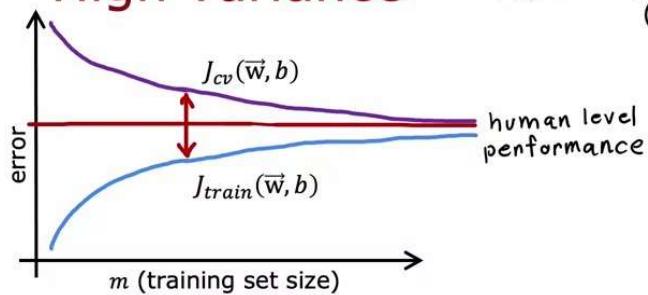


$$f_{\vec{w}, b}(x) = w_1 x + b$$

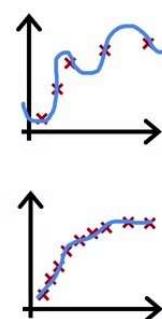


if a learning algorithm suffers from high bias, getting more training data will not (by itself) help much.

High variance



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



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

Debugging a learning algorithm

You've implemented regularized linear regression on housing prices

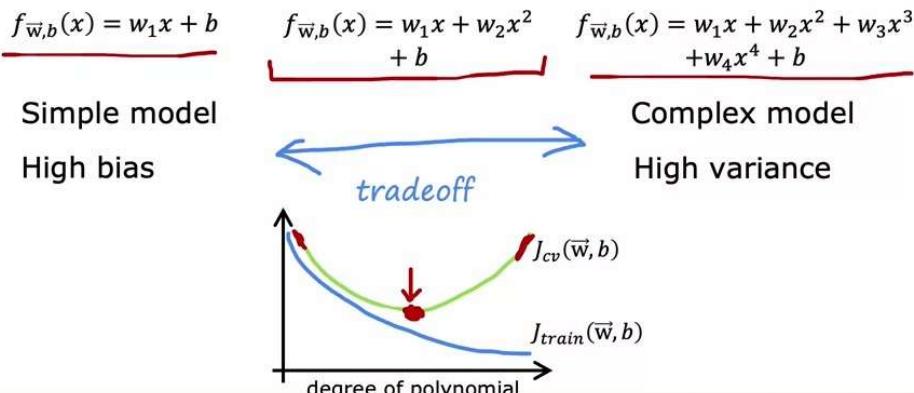
$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

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

- Get more training examples
 - Try smaller sets of features $x, x^2, \cancel{x}, \cancel{x^2}, \cancel{x^3}, \cancel{x^4}, \dots$
 - Try getting additional features \leftarrow
 - Try adding polynomial features $(x_1^2, x_2^2, x_1 x_2, \text{etc})$
 - Try decreasing $\lambda \leftarrow$
 - Try increasing $\lambda \leftarrow$
- fixes high variance
fixes high variance
fixes high bias
fixes high bias
fixes high bias
fixes high variance

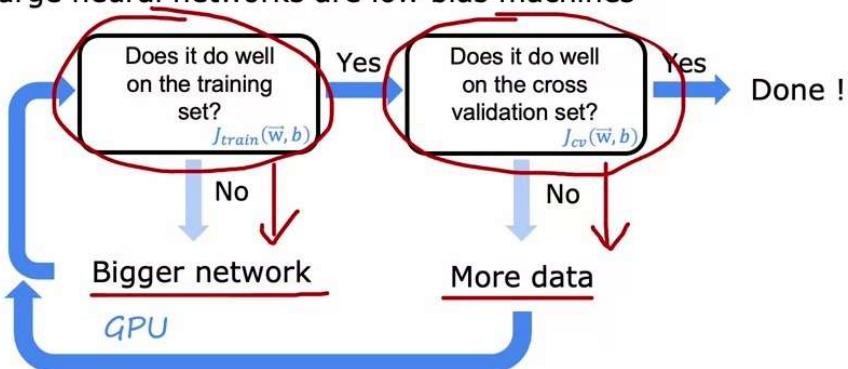
Neural networks

The bias variance tradeoff

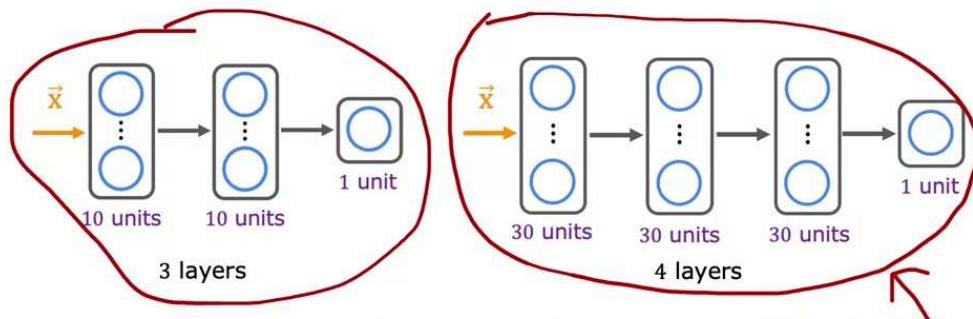


Neural networks and bias variance

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

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

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])
```

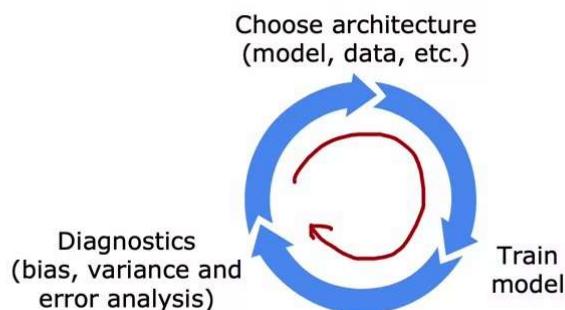
Regularized MNIST model

```
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])
```

Machine learning development process

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
Medcine (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 $x_1, x_2, \dots, x_{10,000}$

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

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

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

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.

Error analysis

Error analysis

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

Algorithm misclassifies $\frac{100}{1000}$ of them.

Manually examine $\frac{100}{1000}$ examples and categorize them based on common traits.

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

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.

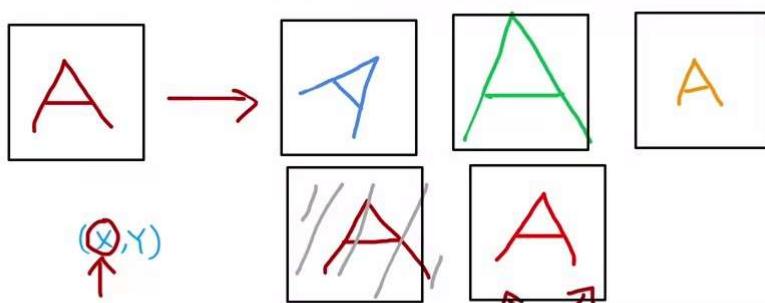
Pharma spam

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

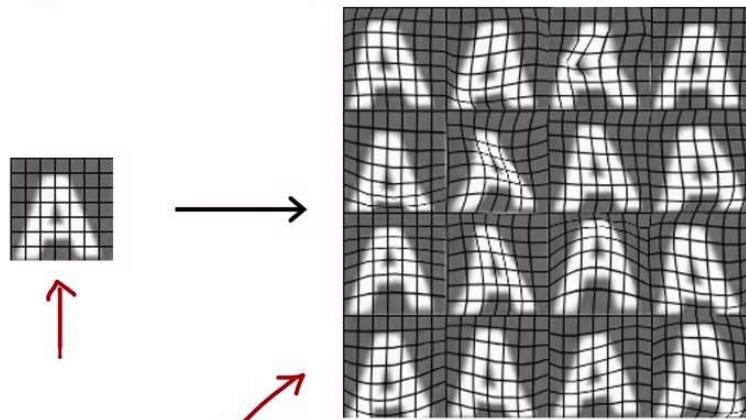
Beyond getting brand new training examples (x, y), another technique: Data augmentation

Data augmentation

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



Data augmentation by introducing distortions



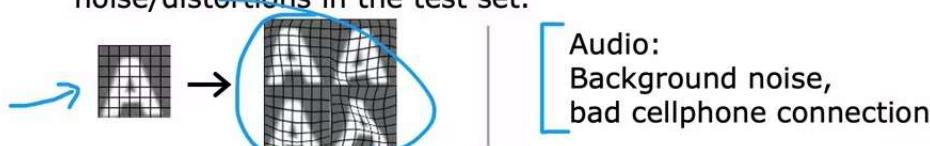
Data augmentation for speech

Speech recognition example

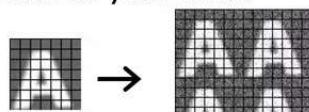
-  Original audio (voice search: "What is today's weather?")
-  + Noisy background: Crowd
-  + Noisy background: Car
-  + Audio on bad cellphone connection

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.



x_i =intensity (brightness) of pixel i
 $x_i \leftarrow x_i + \text{random noise}$

Artificial data synthesis for photo OCR



Real data

[Adam Coates and Tao Wang]



Synthetic data

Engineering the data used by your system

Conventional
model-centric
approach:

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

(algorithm/model)

work on this

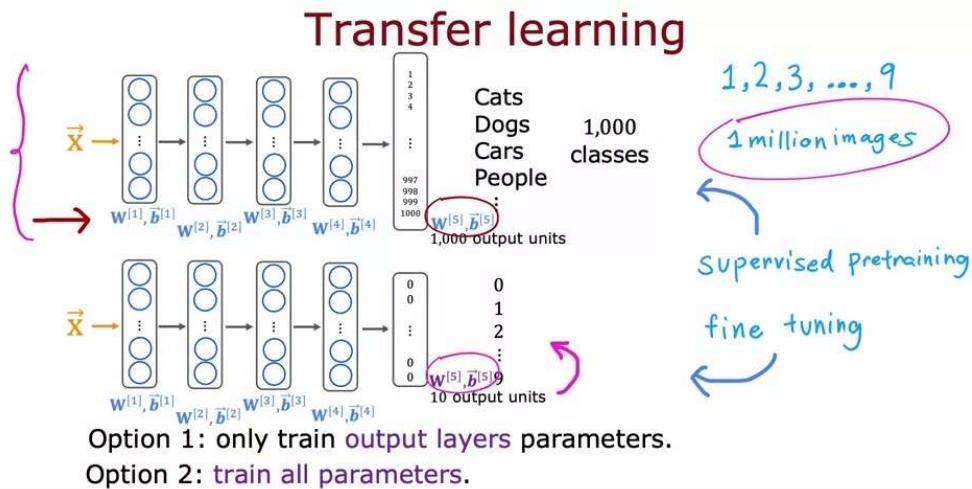
Data-centric
approach:

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

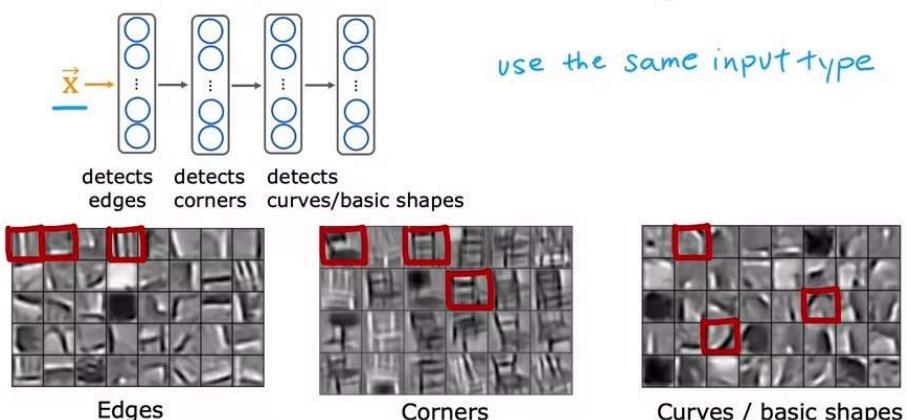
(algorithm/model)

work on this

Transfer learning



Why does transfer learning work?

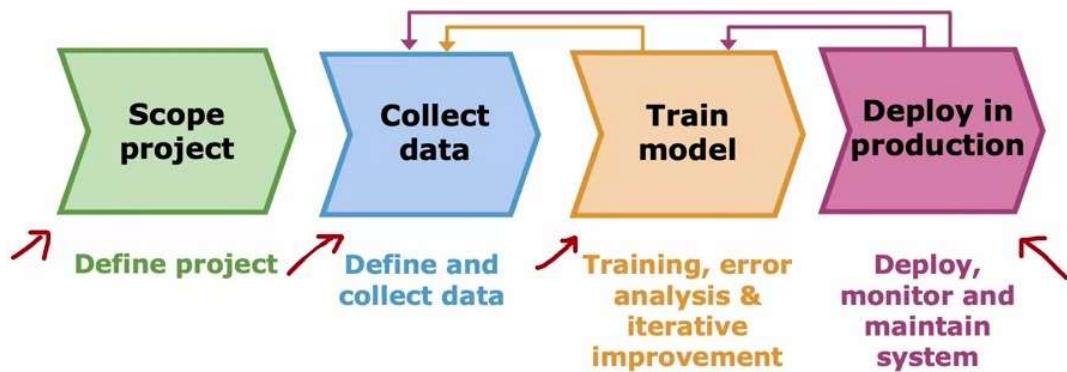


Transfer learning summary

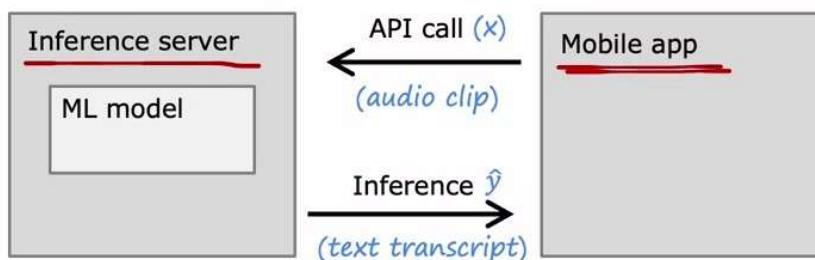
1. Download neural network parameters pretrained on a large dataset with same input type (e.g., images, audio, text) as your application (or train your own). *1 million images*
2. Further train (fine tune) the network on your own data. *1000 images*
50 images

Full cycle of a machine learning project

Full cycle of a machine learning project



Deployment



→ Software engineering may be needed for:

- Ensure reliable and efficient predictions
- Scaling
- Logging
- System monitoring
- Model updates

MLOps
machine learning operations

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

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.

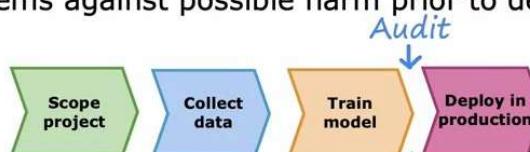
Spam vs anti-spam : fraud vs anti-fraud.

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.

Week 4

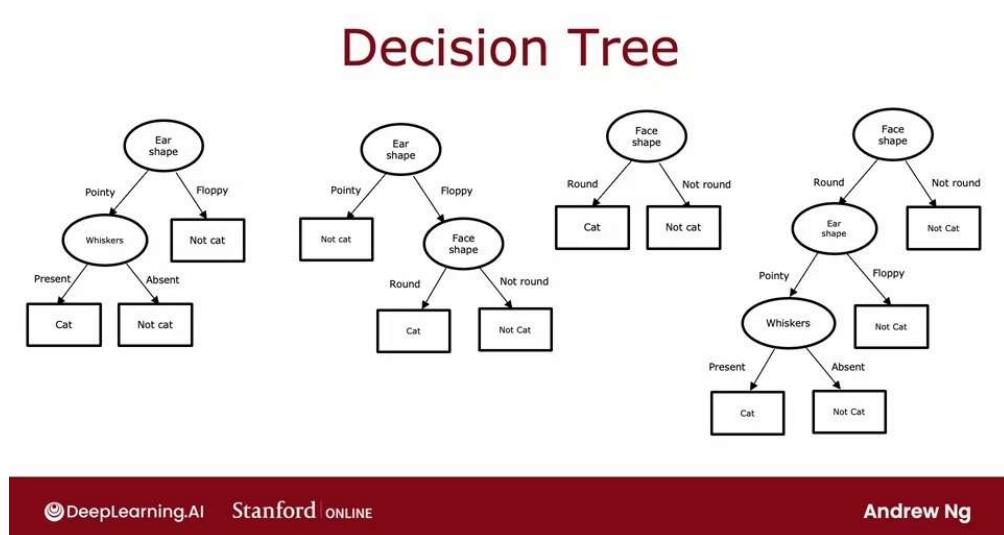
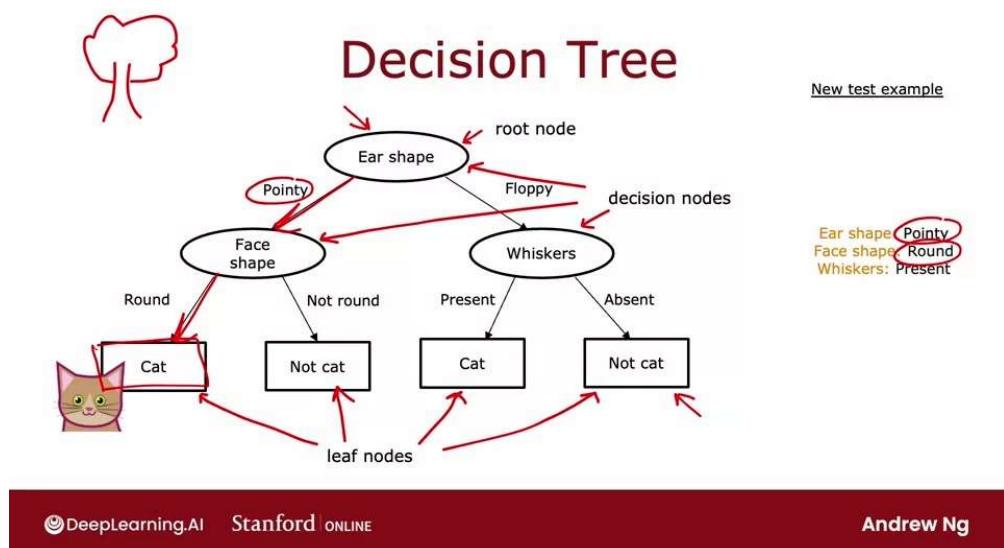
Decision trees

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

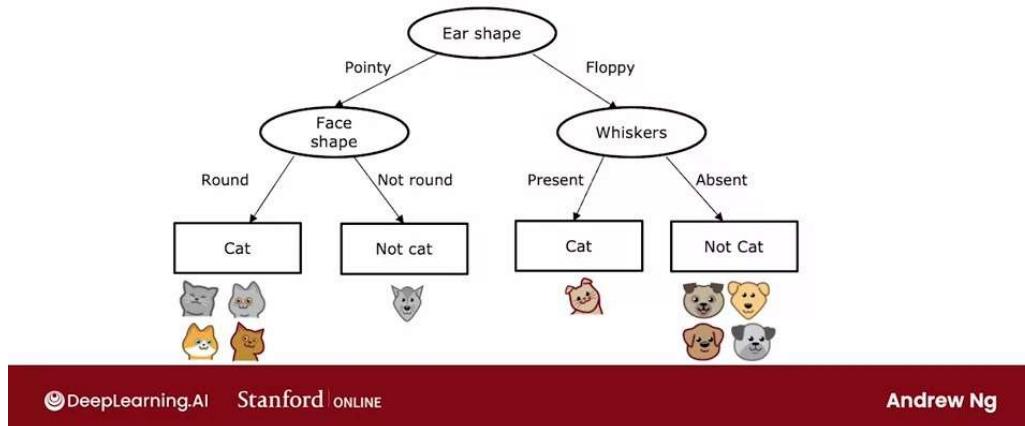
Categorical (discrete values) X Y

DeepLearning.AI Stanford | ONLINE Andrew Ng



Learning Process

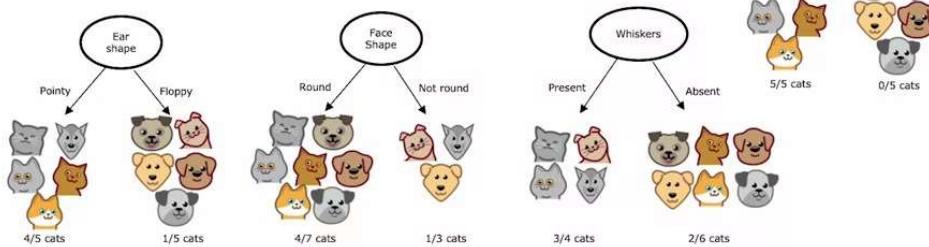
Decision Tree Learning



Decision Tree Learning

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

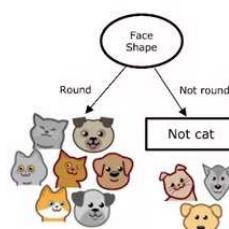
Maximize purity (or minimize impurity)



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



Decision tree learning

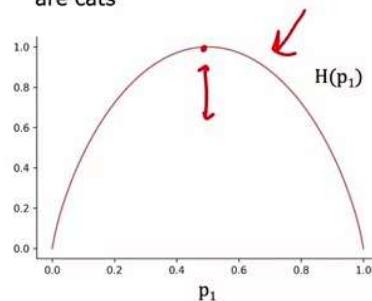
Measuring purity

Entropy as a measure of impurity



Entropy as a measure of impurity

p_1 = fraction of examples that are cats

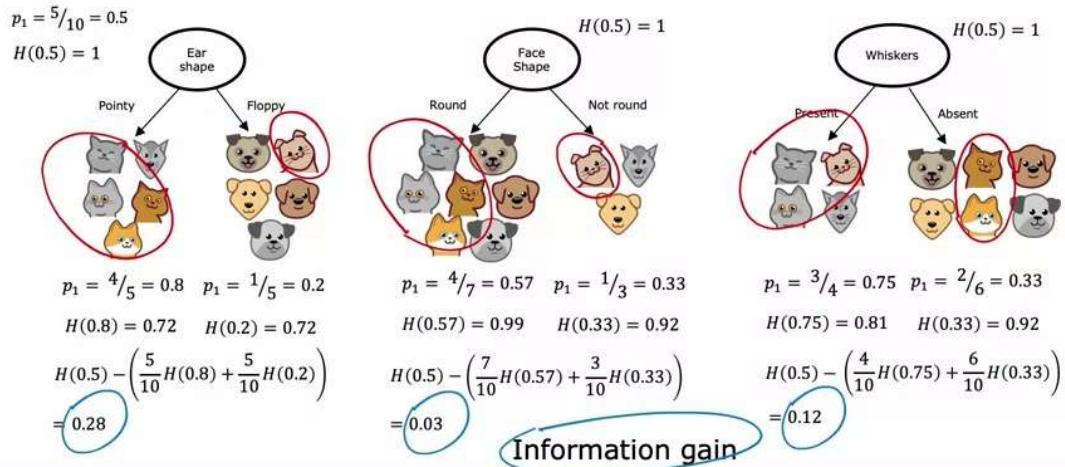


$$\begin{aligned} H(p_1) &= -p_1 \log_2(p_1) - p_0 \log_2(p_0) \\ &= -p_1 \log_2(p_1) - (1-p_1) \log_2(1-p_1) \end{aligned}$$

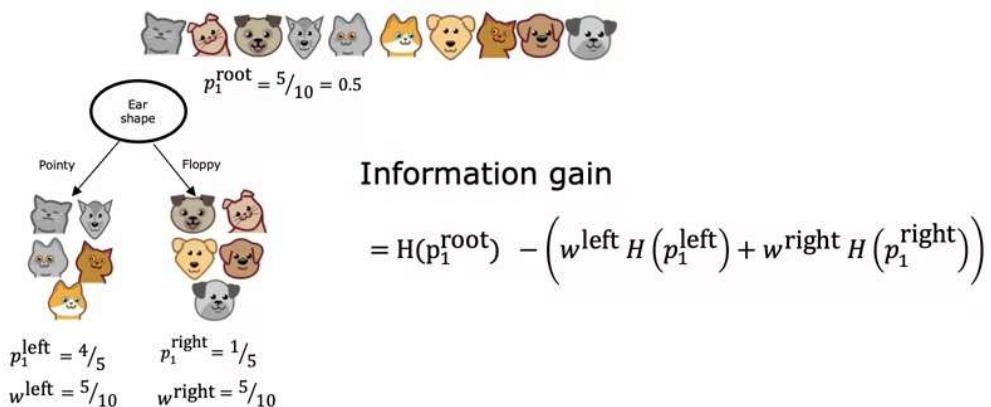
Note: "0 log(0)" = 0

Information Gain

Choosing a split



Information Gain

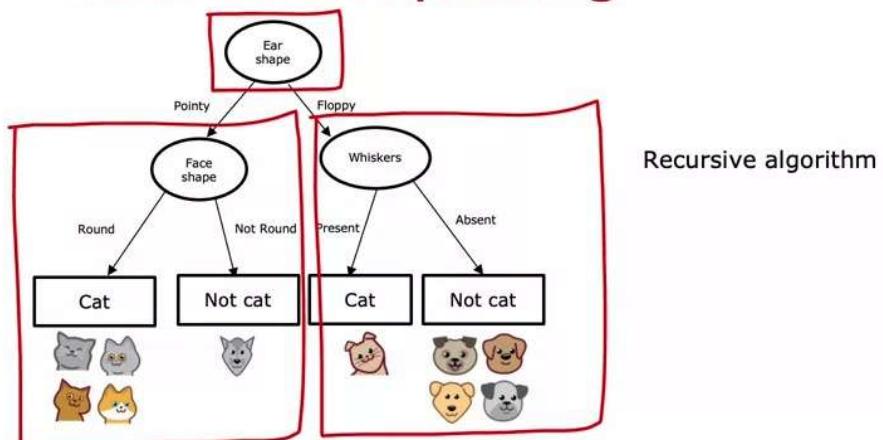


Putting it together

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

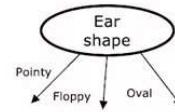


Recursive algorithm

one-hot encoding

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

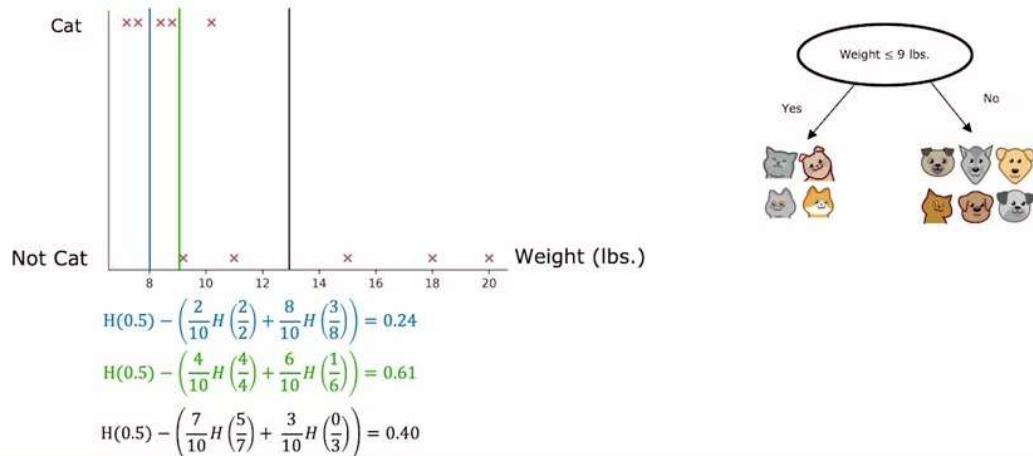
Ear shape	Pointy ears	Floppy ears	Oval ears	Face shape	Whiskers	Cat
	1	0	0	Round	Present	1
	0	0	1	Not round	Present	1
	0	0	1	Round	Absent	0
	1	0	0	Not round	Present	0
	0	0	1	Round	Present	1
	1	0	0	Round	Absent	1
	0	1	0	Not round	Absent	0
	0	0	1	Round	Absent	1
	0	1	0	Round	Absent	0
	0	1	0	Round	Absent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

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

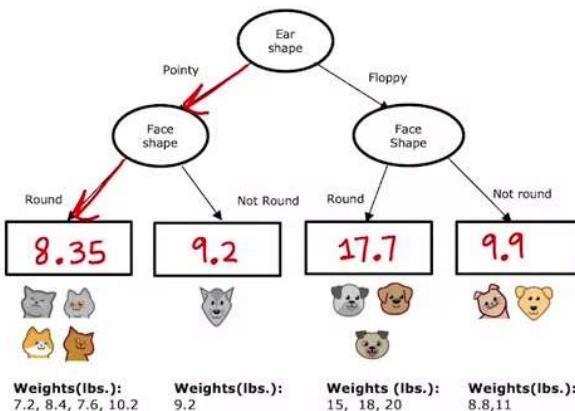
Continuous valued features

Splitting on a continuous variable



Regression Trees

Regression with Decision Trees



Tree ensembles
multiple decision trees

Trees are highly sensitive to small changes of the data

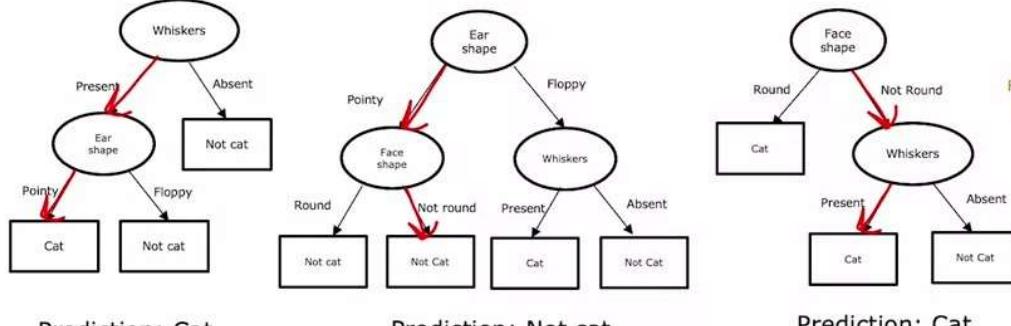


Tree ensemble

New test example



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



Prediction: Cat

Prediction: Not cat

Prediction: Cat

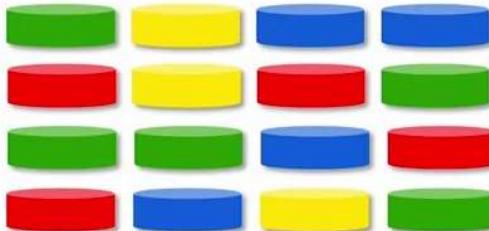
Final prediction: Cat

Sampling with replacement

Sampling with replacement

Tokens 

Sampling with replacement:



Sampling with replacement



	Ear shape	Face shape	Whiskers	Cat
1	Pointy	Round	Present	1
2	Floppy	Not round	Absent	0
3	Pointy	Round	Absent	1
4	Pointy	Not round	Present	0
5	Floppy	Not round	Absent	0
6	Pointy	Round	Absent	1
7	Pointy	Round	Present	1
8	Floppy	Not round	Present	1
9	Floppy	Round	Absent	0
10	Pointy	Round	Absent	1

Random forest algorithm

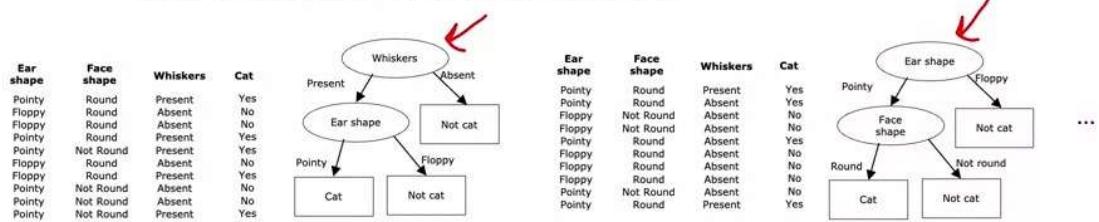
Generating a tree sample

Given training set of size m

For $b = 1$ to B

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

 Train a decision tree on the new dataset



Bagged decision tree

Randomizing the feature choice

At each node, when choosing a feature to use to split, if n features are available, pick a random subset of $k < n$ features and allow the algorithm to only choose from that subset of features.

$$K = \sqrt{n}$$

Random forest algorithm

XGBoost

Boosted trees intuition

Given training set of size m

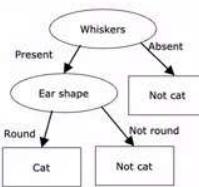
For $b = 1$ to B :

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

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

Train a decision tree on the new dataset

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



Boosted trees intuition

Given training set of size m

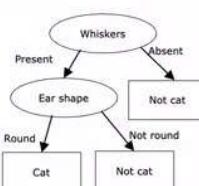
For $b = 1$ to B :

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

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

Train a decision tree on the new dataset

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



$1, 2, \dots, b-1$

b

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

XGBoost (eXtreme Gradient Boosting)

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

Using XGBoost

Classification

```
→from xgboost import XGBClassifier  
→model = XGBClassifier()  
→model.fit(X_train, y_train)  
→y_pred = model.predict(X_test)
```

Regression

```
from xgboost import XGBRegressor  
model = XGBRegressor()  
model.fit(X_train, y_train)  
y_pred = model.predict(X_test)
```

When to use decision trees

Decision Trees vs Neural Networks

Decision Trees and Tree ensembles

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

Neural Networks

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