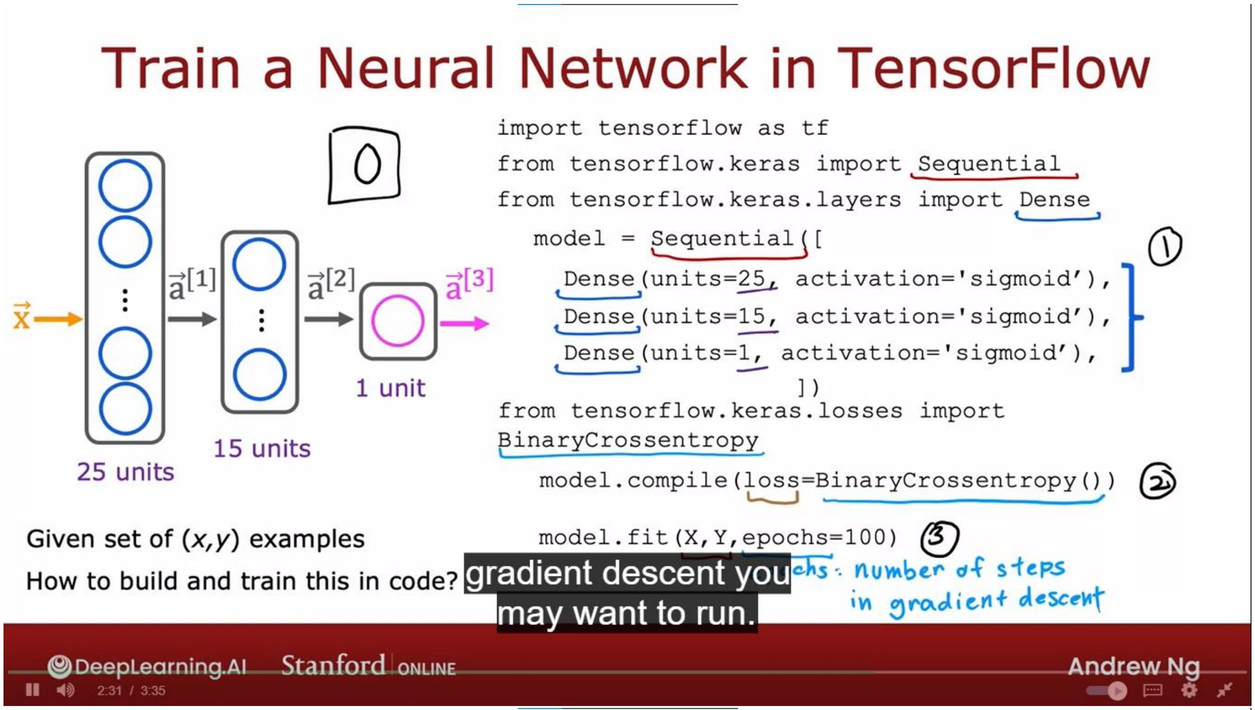
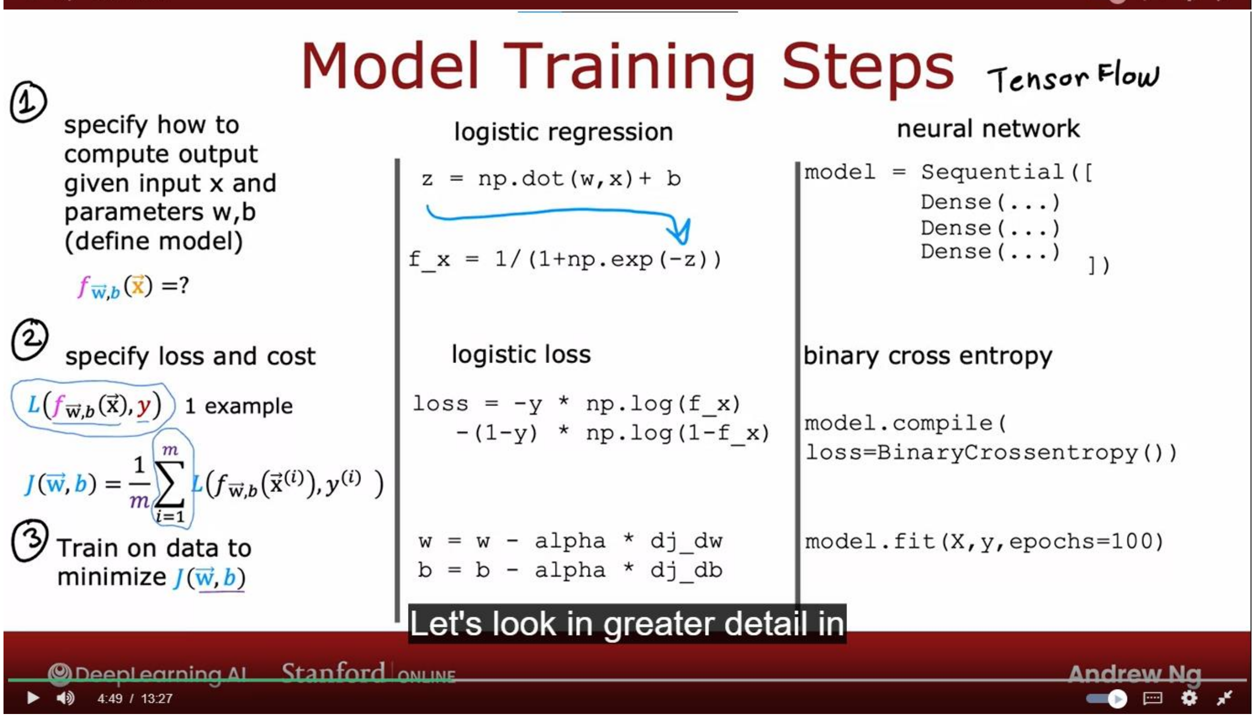
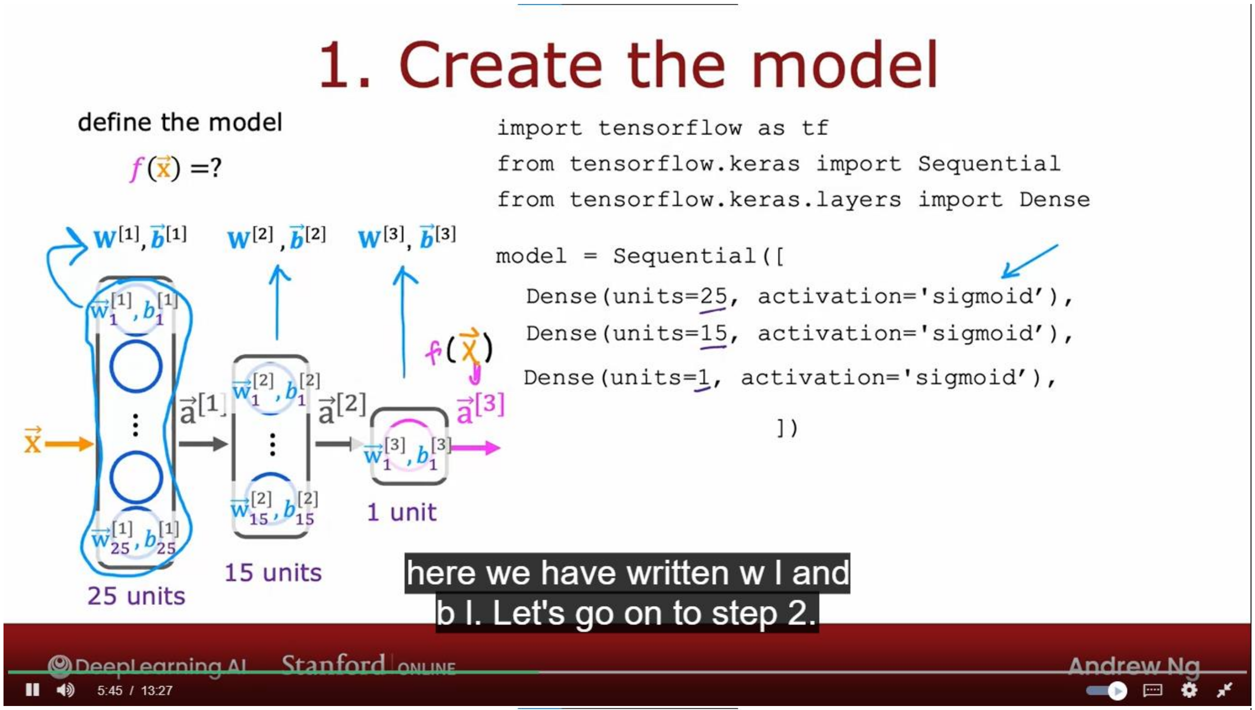
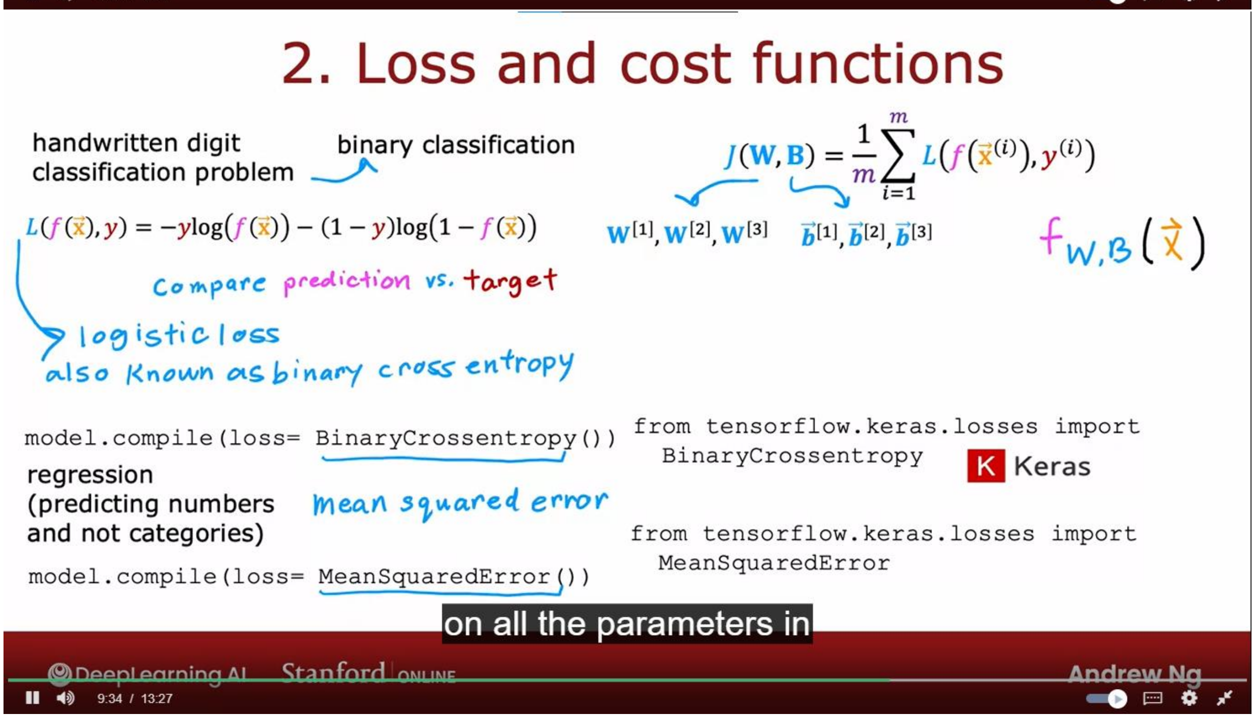
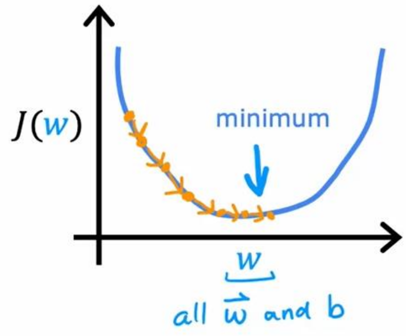

Train a Neural Network in TensorFlow
0
x
a[1]
a[2]
a[3]
25 units
15 units
1 unit
import tensorflow as tf
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
model = Sequential([
Dense(units=25, activation='sigmoid'),
Dense(units=15, activation='sigmoid'),
Dense(units=1, activation='sigmoid'),
])
1
from tensorflow.keras.losses import
BinaryCrossentropy
model.compile(loss=BinaryCrossentropy())
2
model.fit(X,Y,epochs=100)
3
epochs: number of steps
in gradient descent
Given set of (x,y) examples
How to build and train this in code?
gradient descent you
may want to run.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
2:31 / 3:35


Model Training Steps
TensorFlow
1
specify how to
compute output
given input x and
parameters w,b
(define model)
f_w,b(x) =?
2
specify loss and cost
L(f_w,b(x), y) 1 example
J(w,b) = 1/m Σ_{i=1}^{m} L(f_w,b(x^(i)), y^(i))
3
Train on data to
minimize J(w,b)
logistic regression
z = np.dot(w,x)+ b
f_x = 1/(1+np.exp(-z))
logistic loss
loss = -y * np.log(f_x)
-(1-y) * np.log(1-f_x)
w = w - alpha * dj_dw
b = b - alpha * dj_db
neural network
model = Sequential([
Dense(...)
Dense(...)
Dense(...)
])
binary cross entropy
model.compile(
loss=BinaryCrossentropy())
model.fit(X,y,epochs=100)
Let's look in greater detail in
DeepLearning.AI
Stanford ONLINE
Andrew Ng
4:49 / 13:27


1. Create the model
define the model
f(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'),
])
W[1], b[1]
W[2], b[2]
W[3], b[3]
f(x)
x
a[1]
a[2]
a[3]
25 units
15 units
1 unit
here we have written w l and
b l. Let's go on to step 2.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
5:45 / 13:27


2. Loss and cost functions
handwritten digit
classification problem
binary classification
J(W,B) = 1/m Σ L(f(x(i)), y(i))
W[1], W[2], W[3]
b[1], b[2], b[3]
f W,B (x)
L(f(x),y) = −ylog(f(x)) − (1 − y)log(1 − f(x))
compare prediction vs. target
logistic loss
also known as binary cross entropy
model.compile(loss= BinaryCrossentropy())
from tensorflow.keras.losses import
BinaryCrossentropy
Keras
regression
(predicting numbers
and not categories)
mean squared error
from tensorflow.keras.losses import
MeanSquaredError
model.compile(loss= MeanSquaredError())
on all the parameters in
DeepLearning.AI
Stanford ONLINE
Andrew Ng
9:34 / 13:27

3. Gradient descent



repeat {

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

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

} Compute derivatives for gradient descent using "back propagation"

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

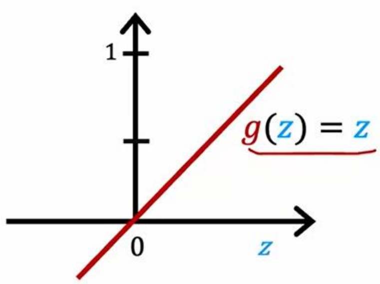
In fact, what you see later
is that TensorFlow can use

DeepLearning.AI Stanford ONLINE Andrew Ng

Examples of Activation Functions

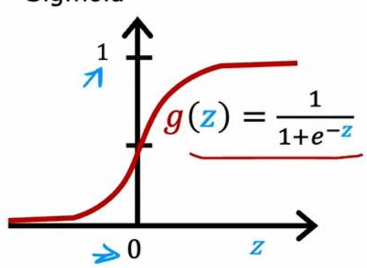
$$a_2^{[1]} = g(\overbrace{\vec{\mathrm{w}}_2^{[1]} \cdot \vec{\mathrm{x}} + b_2^{[1]}}^{z})$$

"No activation function" — Linear activation function



$a = g(z) = \underbrace{\vec{w} \cdot \vec{x} + b}_{z}$

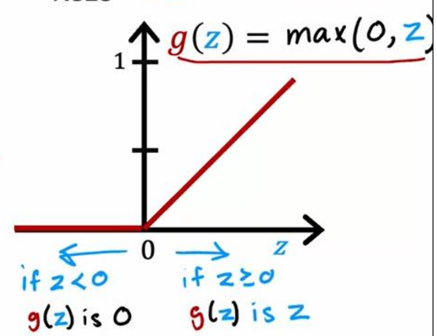
Sigmoid



$0 < g(z) < 1$

Later: softmax activation

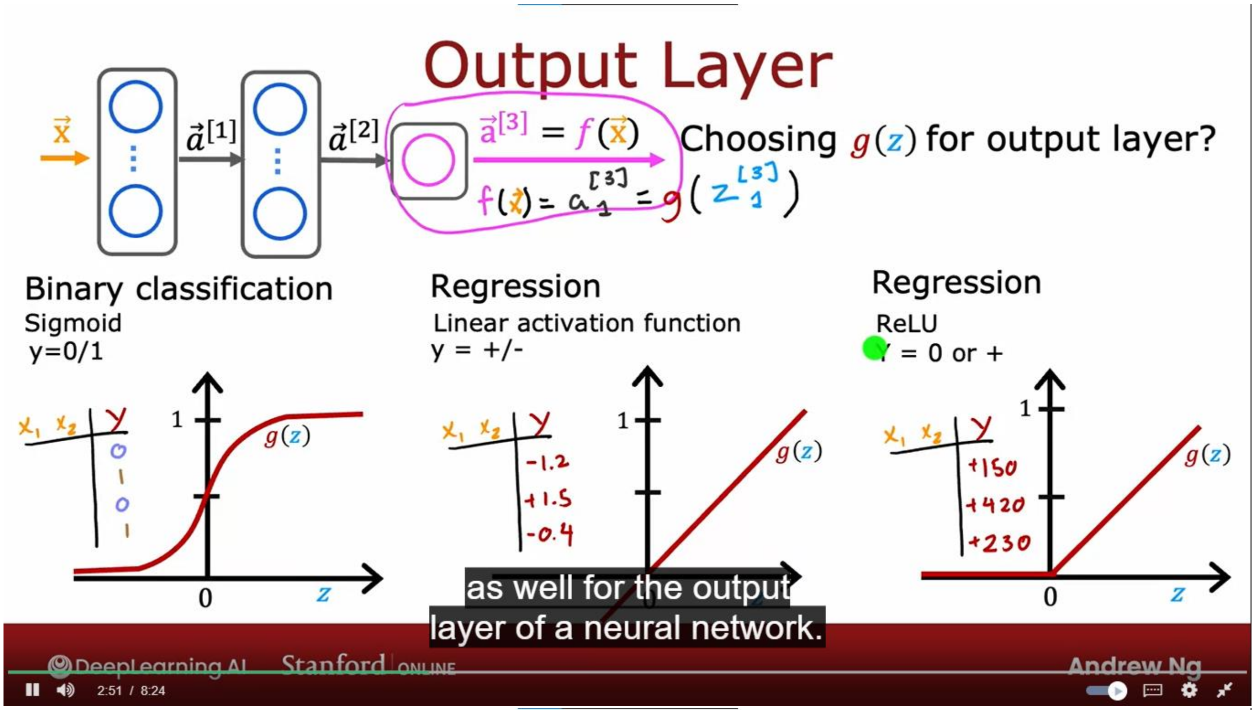
ReLU Rectified Linear Unit

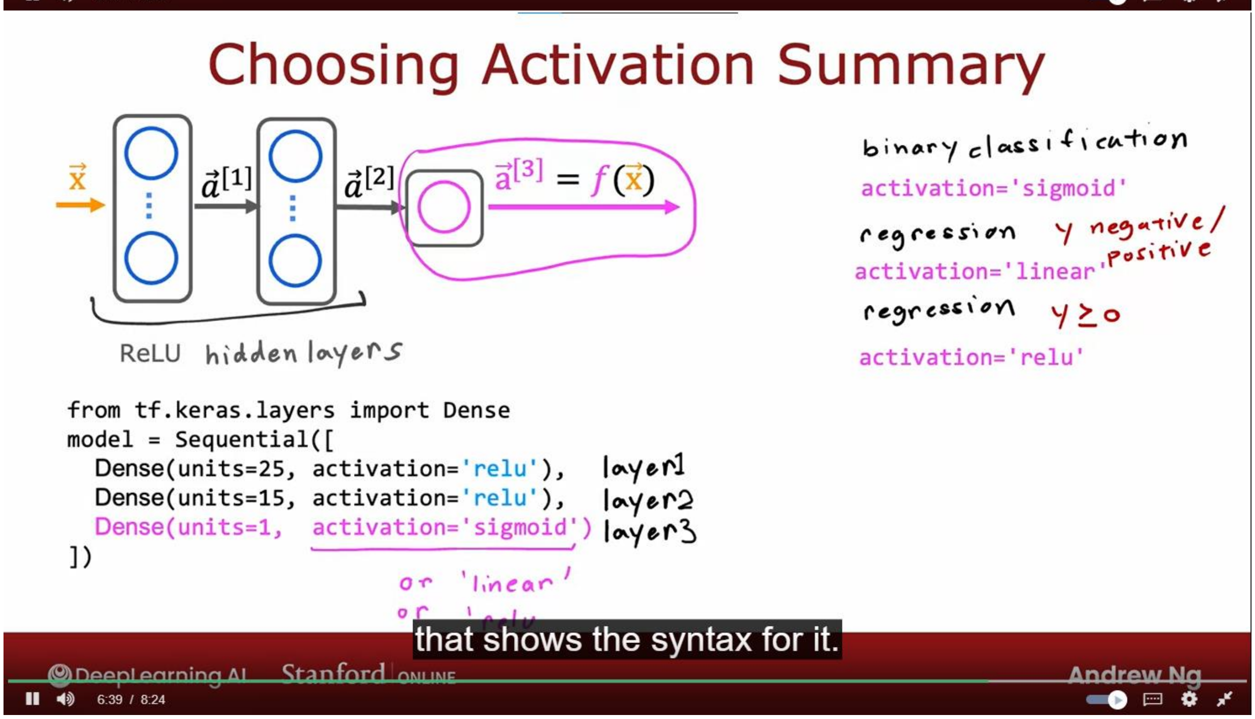


a rich variety of
powerful neural networks.

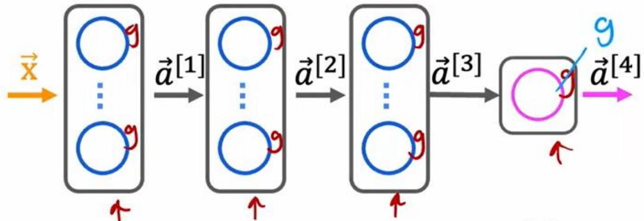
DeepLearning.AI Stanford ONLINE Andrew Ng

5:10 / 5:29


Output Layer
$\vec{x}$
$\vec{a}^{[1]}$
$\vec{a}^{[2]}$
$\vec{a}^{[3]} = f(\vec{x})$
$f(\vec{x}) = a_1^{[3]} = g(z_1^{[3]})$
Choosing $g(z)$ for output layer?
Binary classification
Sigmoid
y=0/1
Regression
Linear activation function
y = +/-
Regression
ReLU
y = 0 or +
x_1 x_2 y
0
1
0
1
-1.2
+1.5
-0.4
+150
+420
+230
$g(z)$
z
as well for the output
layer of a neural network.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
2:51 / 8:24


Choosing Activation Summary
$\vec{x}$
$\vec{a}^{[1]}$
$\vec{a}^{[2]}$
$\vec{a}^{[3]} = f(\vec{x})$
ReLU hidden layers
binary classification
activation='sigmoid'
regression y negative/positive
activation='linear'
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'
that shows the syntax for it.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
6:39 / 8:24

Example



$g(z) = z$

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

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

in the hidden layers
of the neural network.

Logistic regression
(2 possible output values)

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

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

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

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

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

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

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

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

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

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

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

× ○ □ △

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

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

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

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

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

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

thing as logistic regression.

Cost

Logistic regression

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

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

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

$\text{loss} = -y \underbrace{\log a_1}_{\text{if } y=1} - (1-y) \underbrace{\overbrace{\log(1 - a_1)}^{a_2}}_{\text{if } y=0}$

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

Softmax regression

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

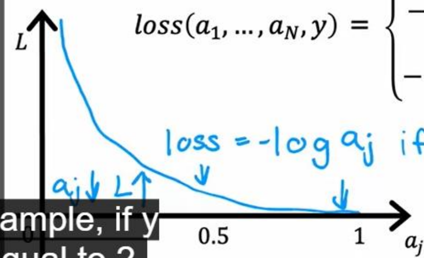
$\vdots$

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

Crossentropy loss

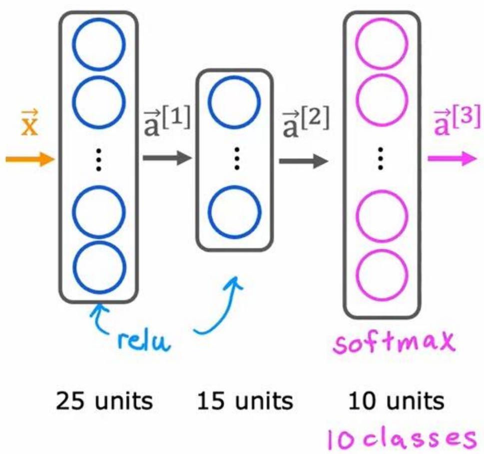
$$loss(a_1, \ldots, a_N, y) = \begin{cases} -\log a_1 & \text{if } y = 1 \\ -\log a_2 & \text{if } y = 2 \\ \vdots & \\ -\log a_N & \text{if } y = N \end{cases}$$

loss = $-\log a_j$ if $y = j$



For example, if y was equal to 2,

Neural Network with Softmax output



$z_1^{[3]} = \vec{w}_1^{[3]} \cdot \vec{a}^{[2]} + b_1^{[3]} \qquad a_1^{[3]} = \dfrac{e^{z_1^{[3]}}}{e^{z_1^{[3]}} + \cdots + e^{z_{10}^{[3]}}}$

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

$\vdots$

$z_{10}^{[3]} = \vec{w}_{10}^{[3]} \cdot \vec{a}^{[2]} + b_{10}^{[3]} \qquad a_{10}^{[3]} = \dfrac{e^{z_{10}^{[3]}}}{e^{z_1^{[3]}} + \cdots + e^{z_{10}^{[3]}}}$

$= P(y = 10|\vec{x})$

logistic regression

$a_1^{[3]} = g\left(z_1^{[3]}\right) \quad a_2^{[3]} = g\left(z_2^{[3]}\right)$

softmax

$\vec{a}^{[3]} = \left(a_1^{[3]}, \ldots a_{10}^{[3]}\right) = g\left(z_1^{[3]}, \ldots, z_{10}^{[3]}\right)$

MNIST with softmax

① specify the model

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

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

② specify loss and cost

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

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

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

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

you can train a neural network on
a multi class classification problem.

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

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

Logistic regression:

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

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

'linear'

Original loss

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

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

More accurate loss (in code)

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

logit: z

worse when it comes to softmax.

More numerically accurate implementation of softmax

Softmax regression

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

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

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

'linear'

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

More Accurate

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

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

and this whole computation of

DeepLearning.AI Stanford ONLINE Andrew Ng

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.softma
```

It is instead of putting
z_1 through z_10.

DeepLearning.AI Stanford ONLINE Andrew Ng

8:27 / 9:11

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)
```
z

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

to actually get the probability.

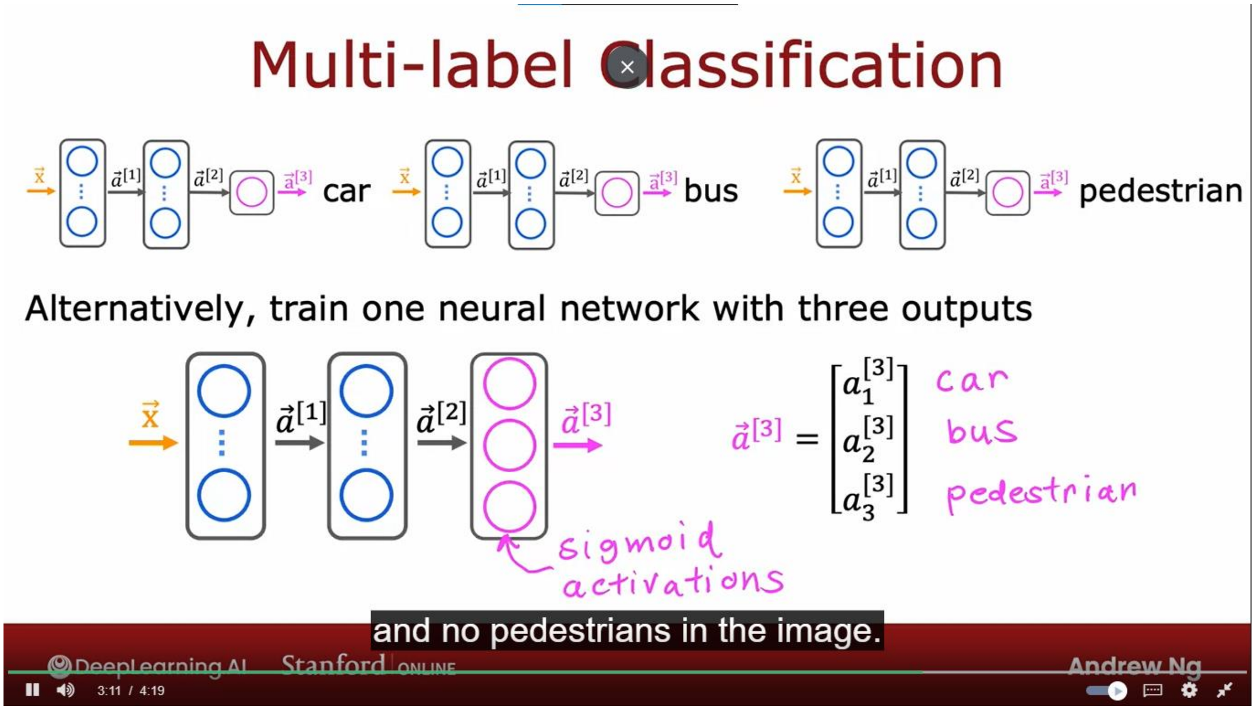
Multi-label Classification

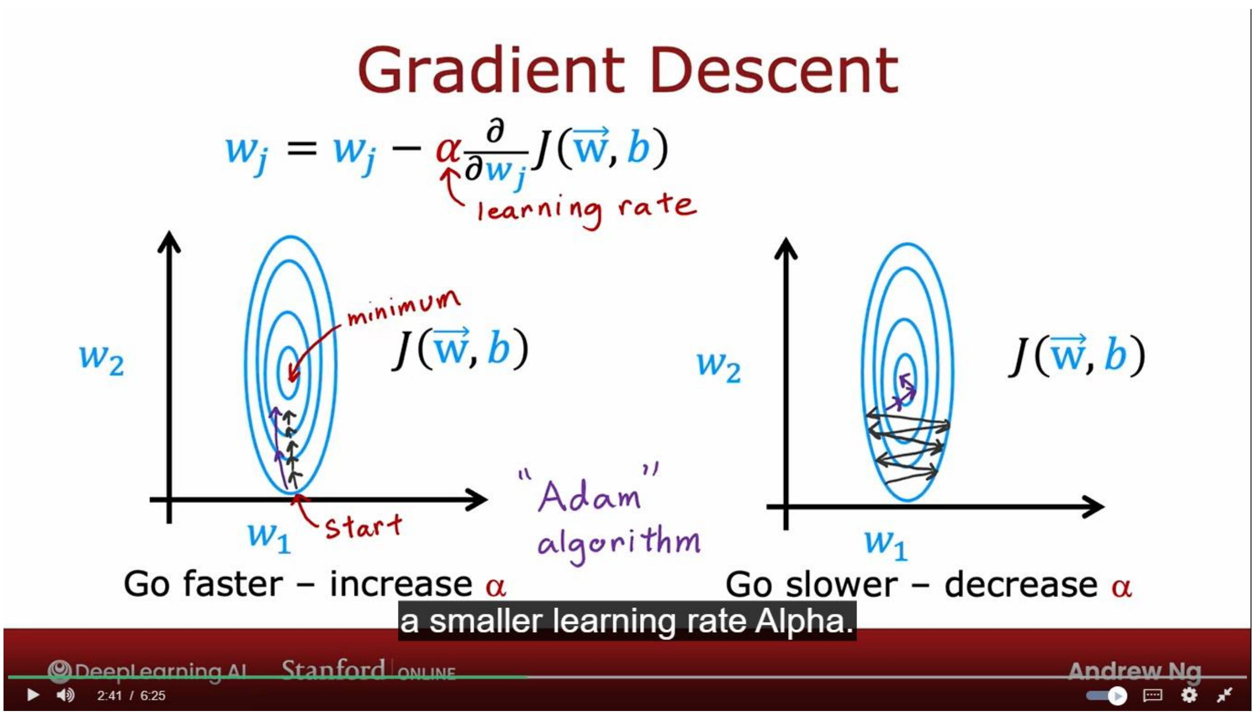
$\vec{x}$

Is there a car?	yes	no	yes
Is there a bus?	no	no	yes
Is there a pedestrian	yes	yes	no

$y=\begin{bmatrix}1\\0\\1\end{bmatrix}$ $y=\begin{bmatrix}0\\0\\1\end{bmatrix}$ $y=\begin{bmatrix}1\\1\\0\end{bmatrix}$

The second one to detect buses and


Multi-label Classification
car
bus
pedestrian
Alternatively, train one neural network with three outputs
$\vec{a}^{[3]} = \begin{bmatrix} a_1^{[3]} \\ a_2^{[3]} \\ a_3^{[3]} \end{bmatrix}$
car
bus
pedestrian
sigmoid activations
and no pedestrians in the image.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
3:11 / 4:19


Gradient Descent
$w_j = w_j - \alpha \frac{\partial}{\partial w_j} J(\vec{w}, b)$
learning rate
minimum
$J(\vec{w}, b)$
w_2
w_1
start
"Adam" algorithm
Go faster – increase α
Go slower – decrease α
a smaller learning rate Alpha.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
2:41 / 6:25

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

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

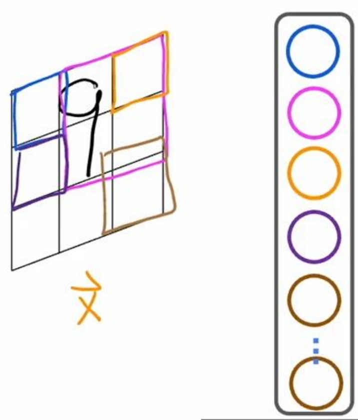
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
  loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

fit

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

see what gives you the
fastest learning performance.

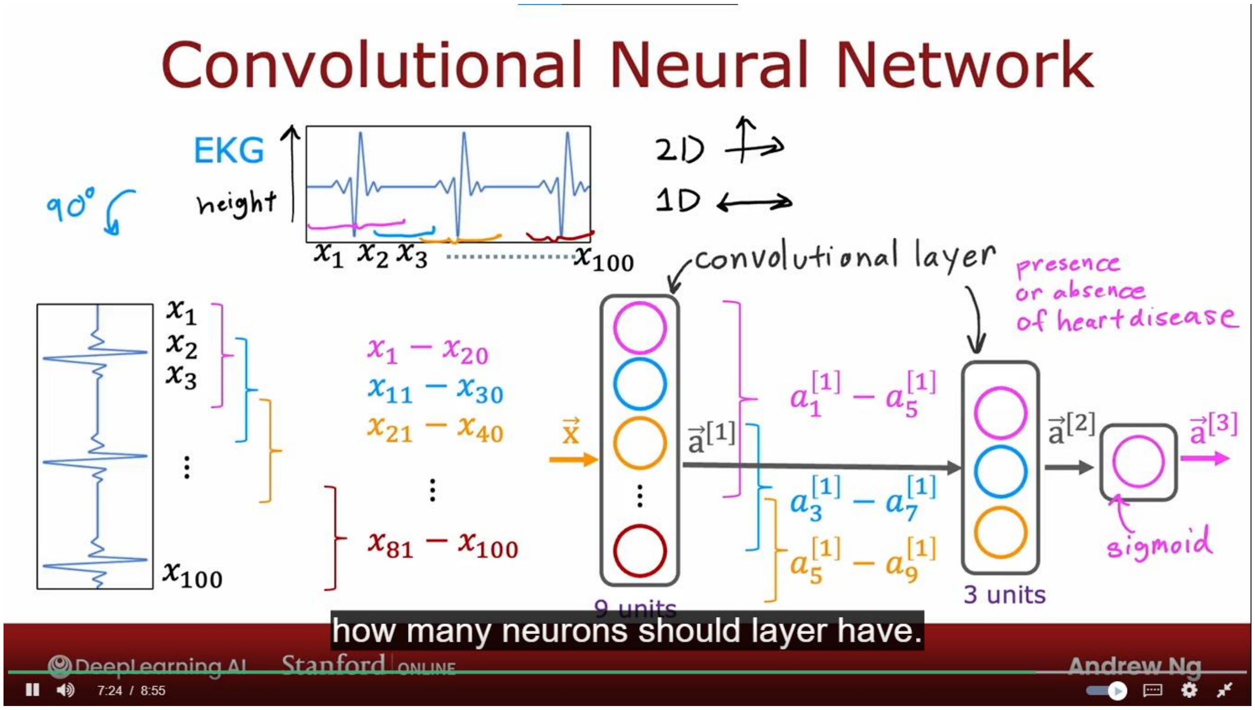
Convolutional Layer

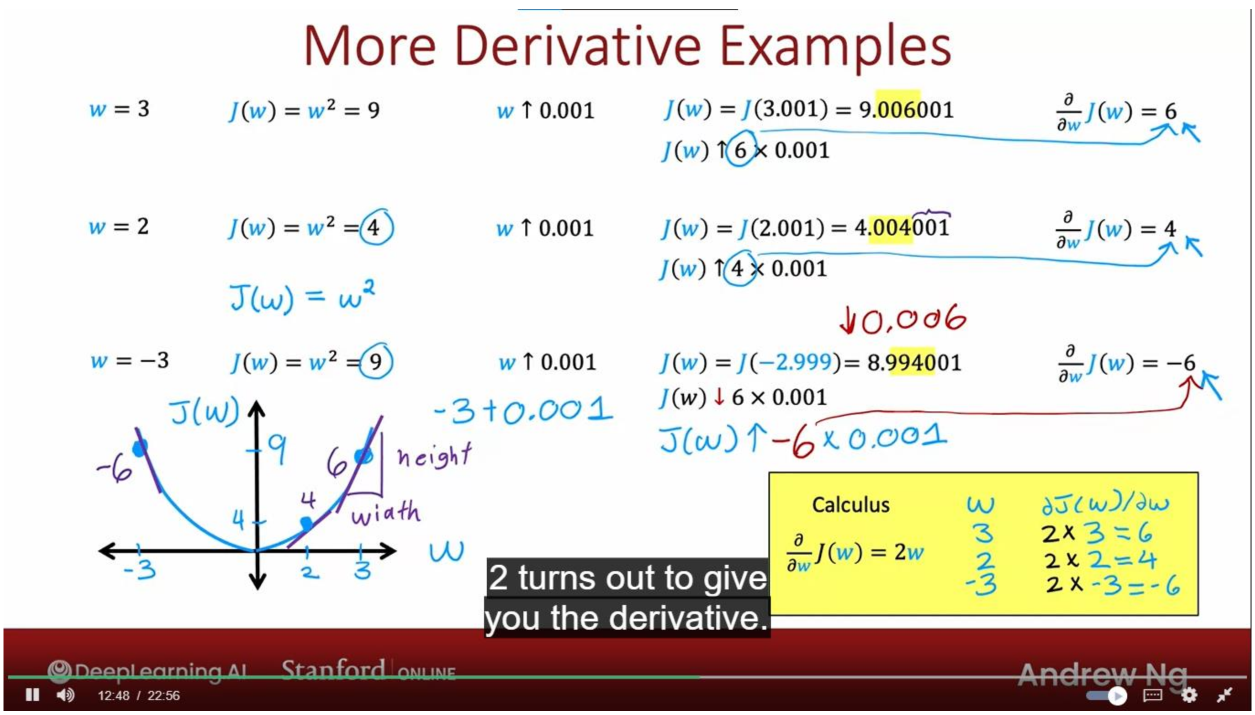


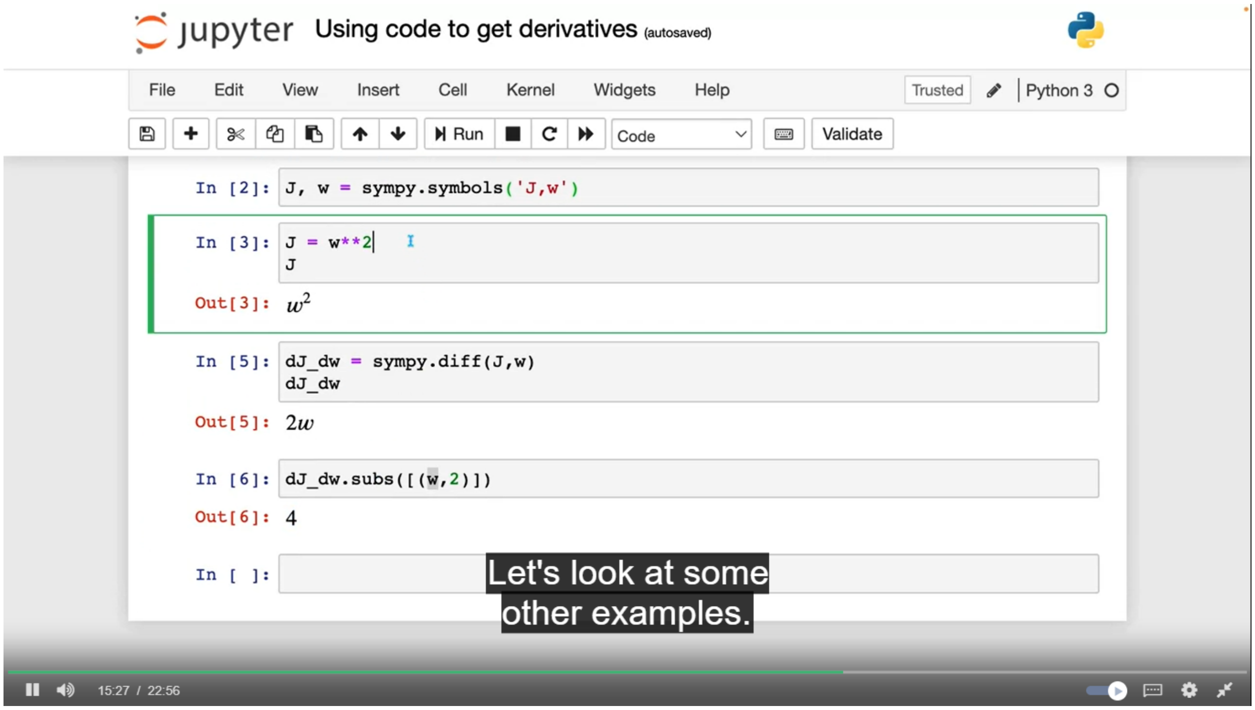
Each neuron only looks at
part of the previous layer's outputs.

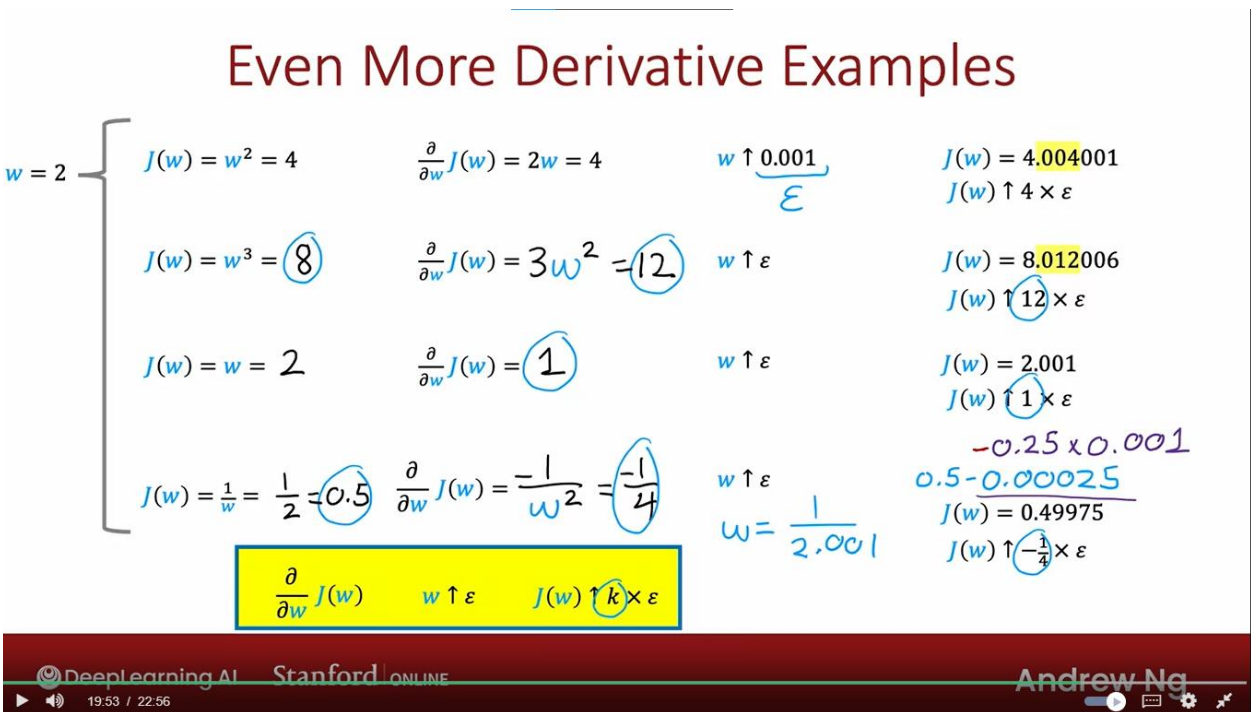
Why?
- Faster computation
- Need less training data
  (less prone to
  overfitting)

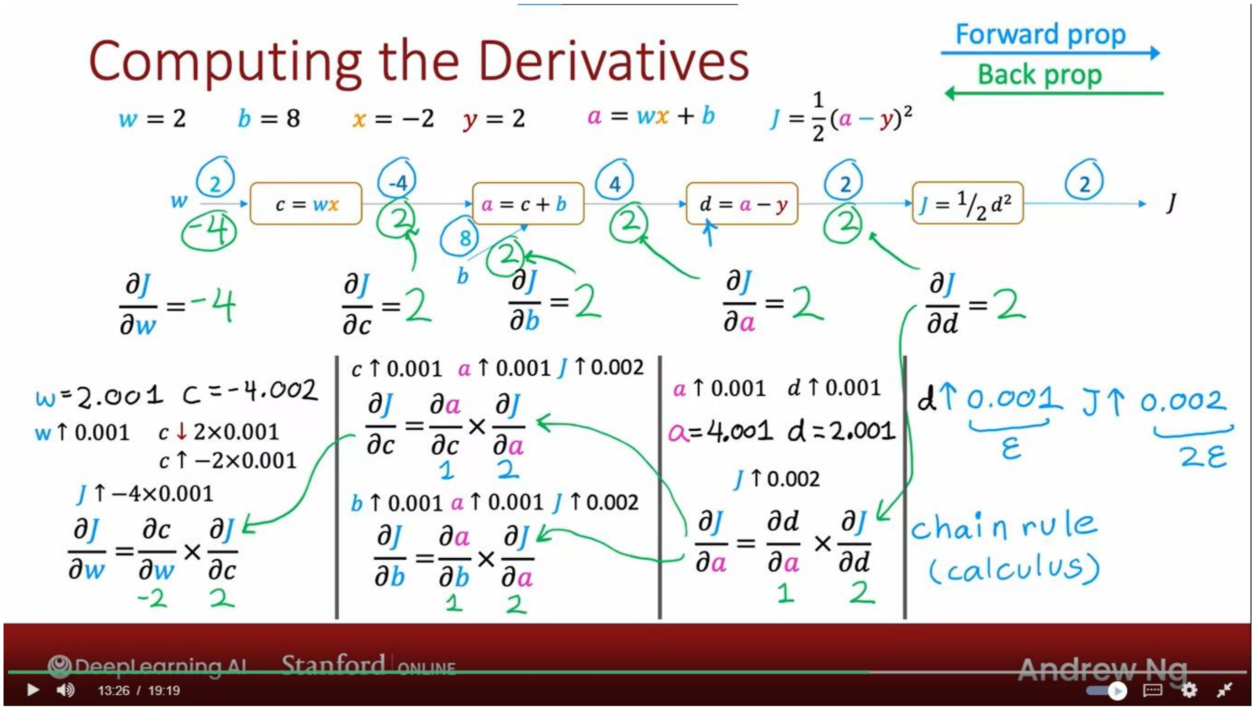
of how to get convolutional layers
to work and popularized their use.

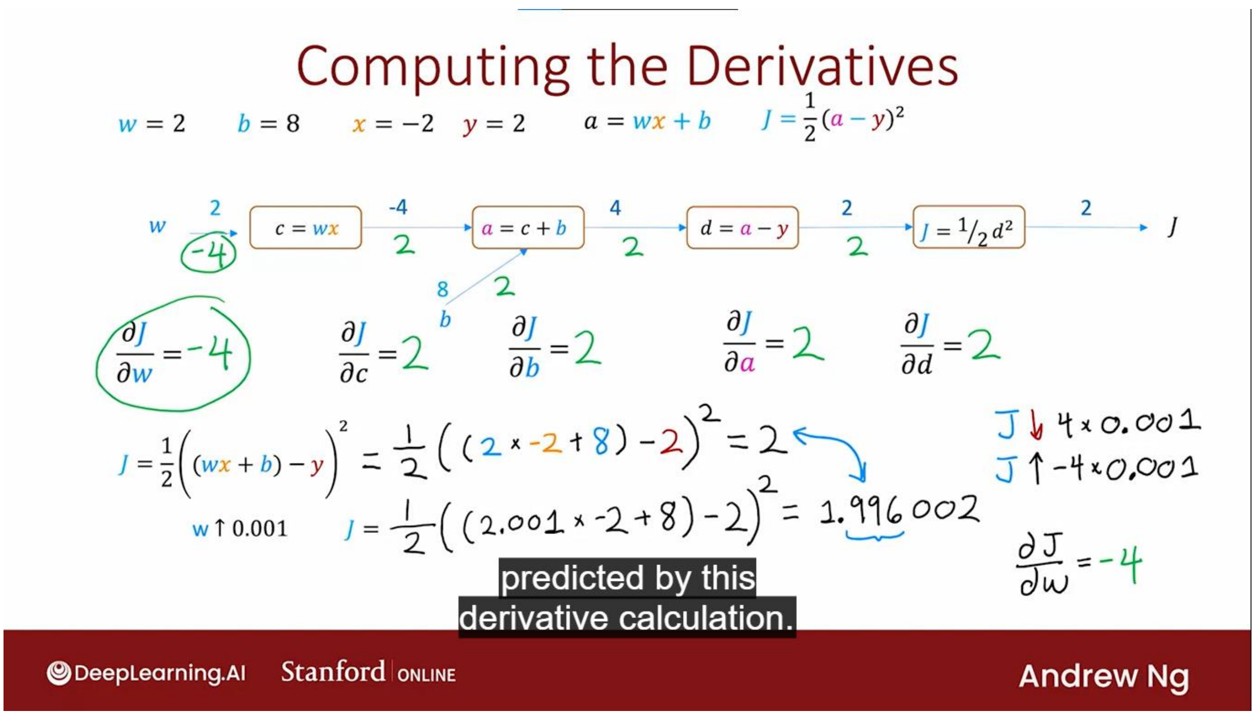

Convolutional Neural Network
EKG
height
90°
2D
1D
x_1 x_2 x_3 .... x_{100}
x_1
x_2
x_3
x_{100}
$x_1 - x_{20}$
$x_{11} - x_{30}$
$x_{21} - x_{40}$
$x_{81} - x_{100}$
$\vec{x}$
convolutional layer
$\vec{a}^{[1]}$
$a_1^{[1]} - a_5^{[1]}$
$a_3^{[1]} - a_7^{[1]}$
$a_5^{[1]} - a_9^{[1]}$
9 units
3 units
$\vec{a}^{[2]}$
$\vec{a}^{[3]}$
presence or absence of heart disease
sigmoid
how many neurons should layer have.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
7:24 / 8:55

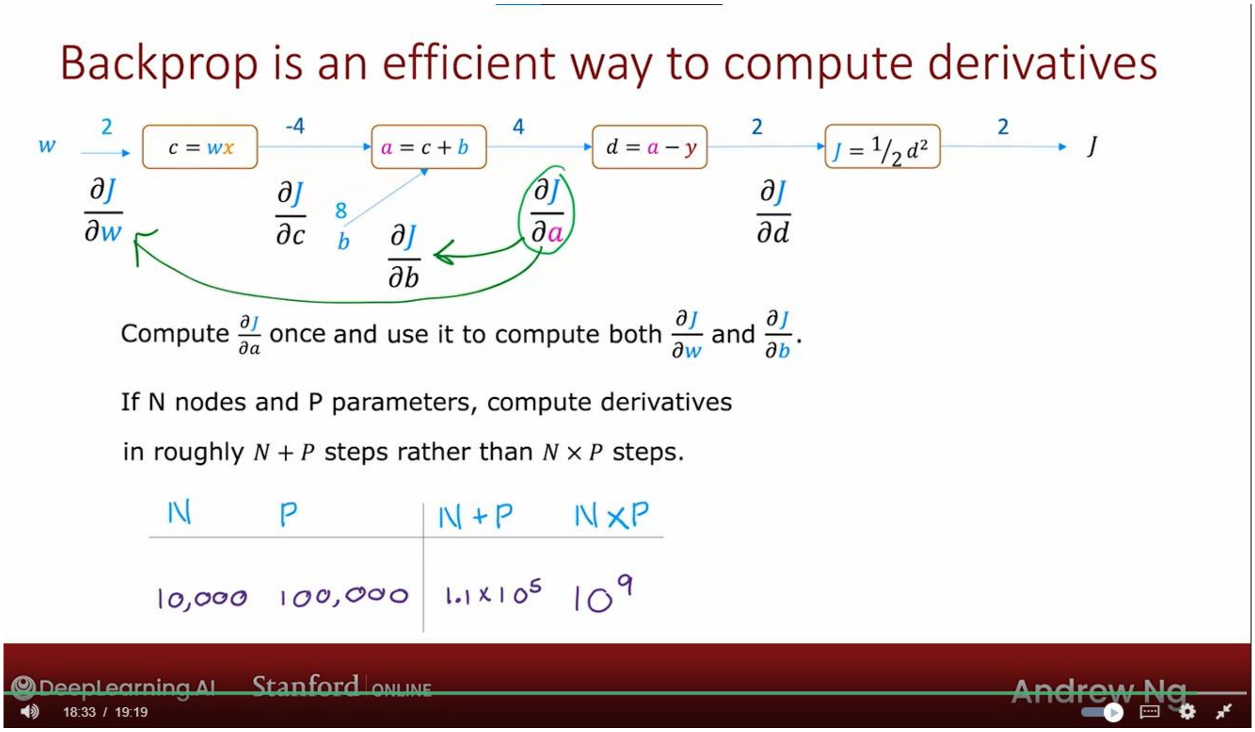

More Derivative Examples
$w = 3$
$J(w) = w^2 = 9$
$w \uparrow 0.001$
$J(w) = J(3.001) = 9.006001$
$J(w) \uparrow 6 \times 0.001$
$\frac{\partial}{\partial w} J(w) = 6$
$w = 2$
$J(w) = w^2 = 4$
$J(w) = w^2$
$w \uparrow 0.001$
$J(w) = J(2.001) = 4.004001$
$J(w) \uparrow 4 \times 0.001$
$\frac{\partial}{\partial w} J(w) = 4$
$\downarrow 0.006$
$w = -3$
$J(w) = w^2 = 9$
$w \uparrow 0.001$
$J(w) = J(-2.999) = 8.994001$
$J(w) \downarrow 6 \times 0.001$
$J(w) \uparrow -6 \times 0.001$
$\frac{\partial}{\partial w} J(w) = -6$
$-3 + 0.001$
J(w)
9
4
-6
6
height
width
w
-3
2
3
Calculus
$\frac{\partial}{\partial w} J(w) = 2w$
w
$\partial J(w)/\partial w$
3 2 x 3 = 6
2 2 x 2 = 4
-3 2 x -3 = -6
2 turns out to give you the derivative.
DeepLearning.AI
Stanford ONLINE
Andrew Ng
12:48 / 22:56

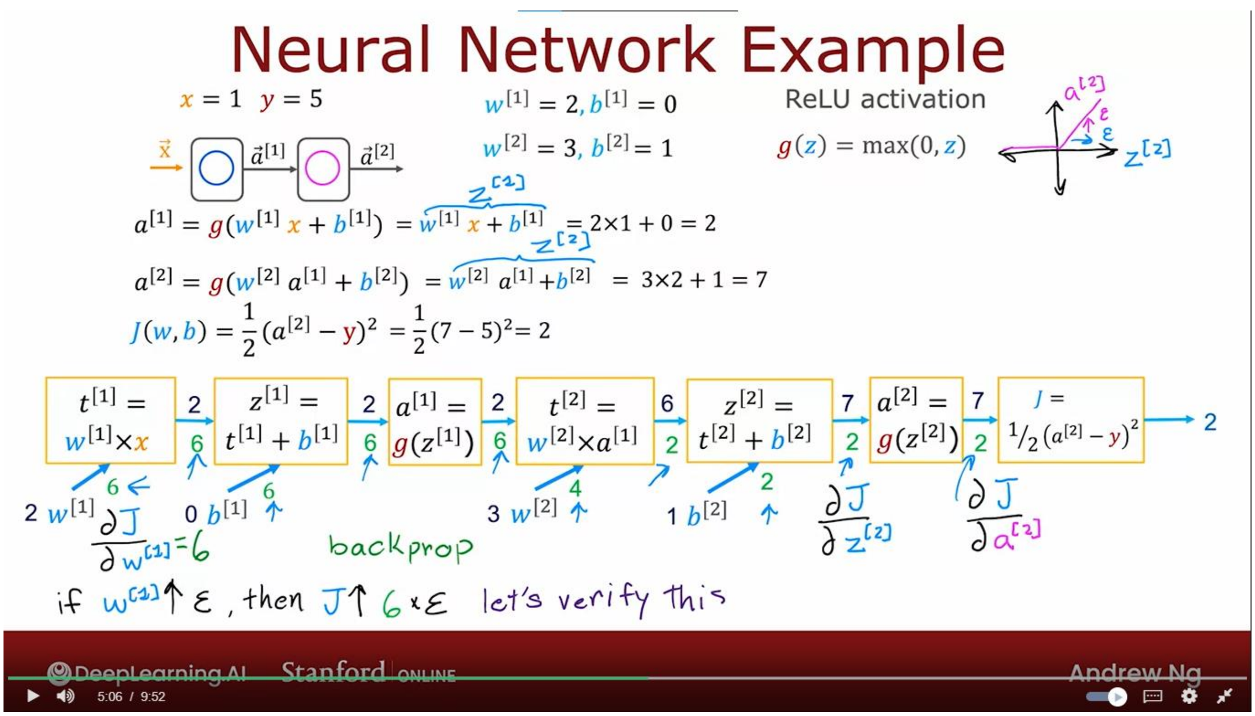

jupyter Using code to get derivatives (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
Validate
In [2]: J, w = sympy.symbols('J,w')
In [3]: J = w**2
J
Out[3]: w^2
In [5]: dJ_dw = sympy.diff(J,w)
dJ_dw
Out[5]: 2w
In [6]: dJ_dw.subs([(w,2)])
Out[6]: 4
In [ ]:
Let's look at some
other examples.
15:27 / 22:56


Even More Derivative Examples
w = 2
J(w) = w² = 4
∂/∂w J(w) = 2w = 4
w ↑ 0.001
ε
J(w) = 4.004001
J(w) ↑ 4 × ε
J(w) = w³ = 8
∂/∂w J(w) = 3w² = 12
w ↑ ε
J(w) = 8.012006
J(w) ↑ 12 × ε
J(w) = w = 2
∂/∂w J(w) = 1
w ↑ ε
J(w) = 2.001
J(w) ↑ 1 × ε
-0.25 x 0.001
0.5 - 0.00025
J(w) = 1/w = 1/2 = 0.5
∂/∂w J(w) = -1/w² = -1/4
w ↑ ε
w = 1/2.001
J(w) = 0.49975
J(w) ↑ -1/4 × ε
∂/∂w J(w)   w ↑ ε   J(w) ↑ k × ε
DeepLearning.AI
Stanford ONLINE
Andrew Ng
19:53 / 22:56


Computing the Derivatives
Forward prop
Back prop
w = 2   b = 8   x = −2   y = 2   a = wx + b   J = 1/2 (a − y)²
c = wx
a = c + b
d = a − y
J = ½ d²
w 2 −4 ; −4 2 ; 8 b 2 ; 4 2 ; 2 2 ; 2 J
∂J/∂w = −4   ∂J/∂c = 2   ∂J/∂b = 2   ∂J/∂a = 2   ∂J/∂d = 2
w=2.001 c=−4.002
w ↑ 0.001   c ↓ 2×0.001
c ↑ −2×0.001
J ↑ −4×0.001
∂J/∂w = ∂c/∂w × ∂J/∂c
−2 2
c ↑ 0.001 a ↑ 0.001 J ↑ 0.002
∂J/∂c = ∂a/∂c × ∂J/∂a
1 2
b ↑ 0.001 a ↑ 0.001 J ↑ 0.002
∂J/∂b = ∂a/∂b × ∂J/∂a
1 2
a ↑ 0.001 d ↑ 0.001
a=4.001 d=2.001
J ↑ 0.002
∂J/∂a = ∂d/∂a × ∂J/∂d
1 2
d↑ 0.001 J↑ 0.002
ε 2ε
chain rule
(calculus)
DeepLearning.AI   Stanford ONLINE   Andrew Ng
13:26 / 19:19

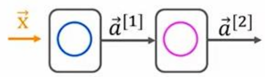

Computing the Derivatives
w = 2   b = 8   x = −2   y = 2   a = wx + b   J = 1/2 (a − y)²
c = wx
a = c + b
d = a − y
J = ½ d²
w 2 −4 ; −4 2 ; 8 b 2 ; 4 2 ; 2 2 ; 2 J
∂J/∂w = −4   ∂J/∂c = 2   ∂J/∂b = 2   ∂J/∂a = 2   ∂J/∂d = 2
J = 1/2 ((wx + b) − y)² = 1/2 ((2 × −2 + 8) − 2)² = 2
w ↑ 0.001   J = 1/2 ((2.001 × −2 + 8) − 2)² = 1.996002
J ↓ 4 × 0.001
J ↑ −4 × 0.001
∂J/∂w = −4
predicted by this
derivative calculation.
DeepLearning.AI   Stanford ONLINE   Andrew Ng


Backprop is an efficient way to compute derivatives
w 2 c = wx -4 a = c + b 4 d = a − y 2 J = 1/2 d² 2 J
8 b
∂J/∂w ∂J/∂c ∂J/∂b ∂J/∂a ∂J/∂d
Compute ∂J/∂a once and use it to compute both ∂J/∂w and ∂J/∂b.
If N nodes and P parameters, compute derivatives
in roughly N + P steps rather than N × P steps.
N P N+P N×P
10,000 100,000 1.1×10⁵ 10⁹
DeepLearning.AI Stanford ONLINE
Andrew Ng
18:33 / 19:19


Neural Network Example
x = 1 y = 5
w[1] = 2, b[1] = 0
w[2] = 3, b[2] = 1
ReLU activation
g(z) = max(0, z)
a[2] ε z[2]
x a[1] a[2]
a[1] = g(w[1] x + b[1]) = w[1] x + b[1] = 2×1 + 0 = 2
z[1]
a[2] = g(w[2] a[1] + b[2]) = w[2] a[1] + b[2] = 3×2 + 1 = 7
z[2]
J(w, b) = 1/2 (a[2] − y)² = 1/2 (7 − 5)² = 2
t[1] = w[1]×x 2 6
z[1] = t[1] + b[1] 2 6
a[1] = g(z[1]) 2 6
t[2] = w[2]×a[1] 6 2
z[2] = t[2] + b[2] 7 2
a[2] = g(z[2]) 7 2
J = 1/2 (a[2] − y)² 2
2 w[1] 6 ∂J/∂w[1] = 6
0 b[1] 6
3 w[2] 4
1 b[2] 2
∂J/∂z[2] ∂J/∂a[2]
backprop
if w[1] ↑ ε, then J ↑ 6×ε let's verify this
DeepLearning.AI Stanford ONLINE
Andrew Ng
5:06 / 9:52

Neural Network Example

$x = 1$ $y = 5$ $w^{[1]} = 2, b^{[1]} = 0$ ReLU activation



$w^{[2]} = 3, b^{[2]} = 1$ $g(z) = \max(0, z)$

$$a^{[1]} = g(w^{[1]}x + b^{[1]}) = w^{[1]}x + b^{[1]} = \cancel{2}^{\,2.001}\times 1 + 0 = \cancel{2}\ 2.001$$

$$a^{[2]} = g(w^{[2]}a^{[1]} + b^{[2]}) = w^{[2]}a^{[1]} + b^{[2]} = 3\times\cancel{2}^{\,2.001} + 1 = \cancel{7}\ 7.003$$

$$J(w,b) = \frac{1}{2}(a^{[2]} - y)^2 = \frac{1}{2}(\cancel{7}^{\,7.003} - 5)^2 = \cancel{2}\quad 2\boxed{.006}005$$

$$\frac{(2.003)^2}{2}$$

$w^{[1]} \uparrow 0.001$

$J \uparrow 6 \times 0.001$

$$\frac{\partial J}{\partial w^{[1]}} = 6$$

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

$$\frac{\partial J}{\partial w^{[2]}} \quad \frac{\partial J}{\partial b^{[2]}}$$

N nodes □→□→□

P parameters

$w_1, b_1, w_2, b_2 \ldots$

inefficient way

$N \times P$

efficient way (backprop)

$N + P$