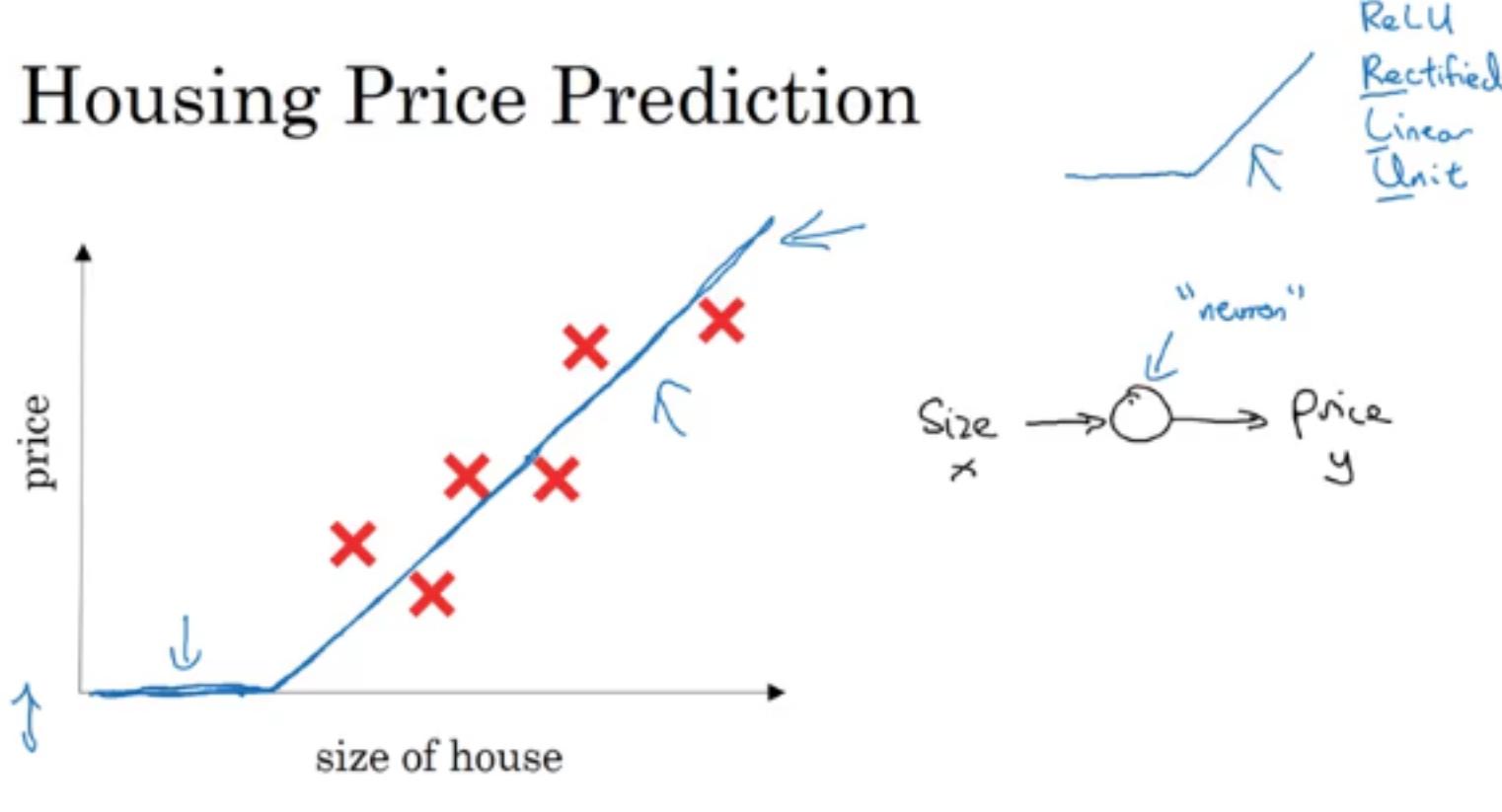




What is a neural network?

You're using Coursera offline

Housing Price Prediction



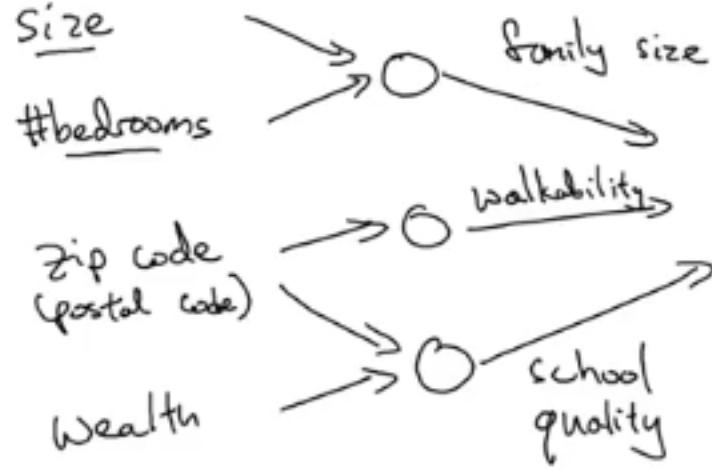
This function which goes to zero sometimes and then it'll take off as a straight line. This function is called a ReLU function which stands for rectified linear units. So R-E-L-U. And rectify just means taking a max of 0 which is why you get a function shape like this.

You don't need to worry about ReLU units for now but it's just something you see again later in this course. So if this is a single neuron, neural network, really a tiny little neural network, a larger neural network is then formed by taking many of the single neurons and stacking them together.

What is a neural network?

You're using Coursera offline

Housing Price Prediction



you how good is the school quality. So each of these little circles I'm drawing, can be one of those ReLU, rectified linear units or some other slightly non linear function.

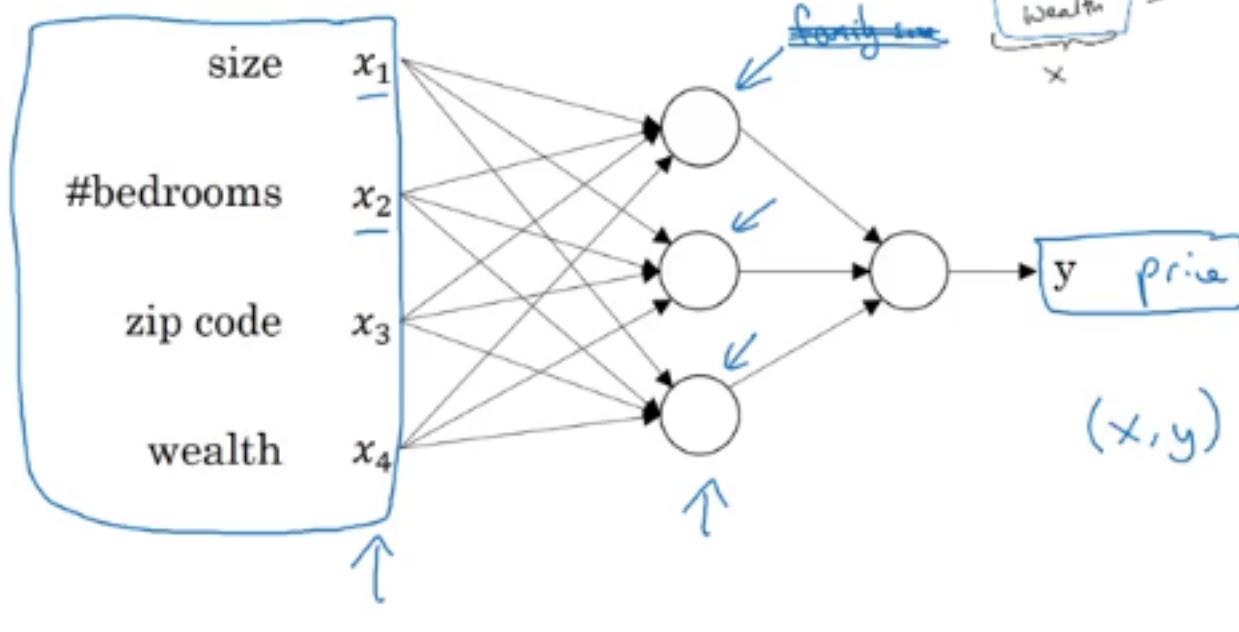
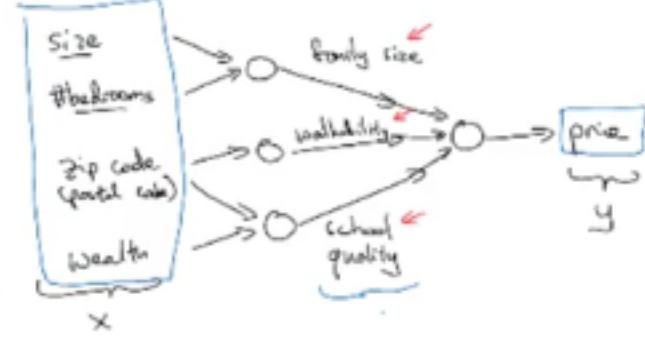
So that based on the size and number of bedrooms, you can estimate the family size, their zip code, based on walkability, **based on zip code and wealth can estimate the school quality.**

And then finally you might think that well the way people decide how much they're will to pay for a house, is they look at the things that really matter to them. In this case family size, walkability, and school quality and that

What is a neural network?

You're using Coursera offline

Housing Price Prediction



And the remarkable thing about neural networks is that, given enough data about x and y , given enough training examples with both x and y , neural networks are remarkably good at figuring out functions that accurately map from x to y . So, that's a basic neural network.

In turns out that as you build out your own neural networks, you probably find them to be most useful, most powerful in supervised learning incentives, meaning that you're trying to take an input x and map it to some output y , like we just saw in the housing price prediction example.

← Supervised Learning Networks

You're using Coursera offline

Supervised Learning

Input(x)	Output (y)	Application
Home features	Price	Real Estate
Ad, user info	Click on ad? (0/1)	Online Advertising
Image	Object (1,...,1000)	Photo tagging
Audio	Text transcript	Speech recognition
English	Chinese	Machine translation
Image, Radar info ↑	Position of other cars ↑	Autonomous driving

Chinese, the alphabets or the words come one at a time.

So language is also most naturally represented as sequence data. And so more complex versions of RNNs are often used for these applications.

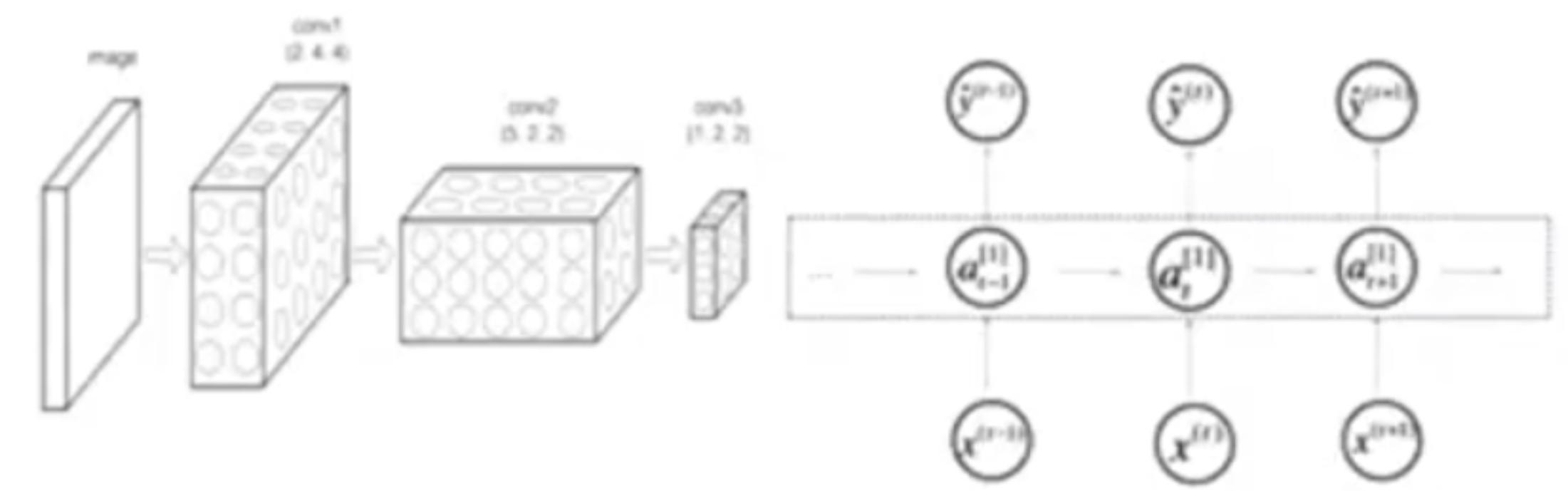
And then, for more complex applications, like autonomous driving, where you have an image, that might suggest more of a CNN convolution neural network structure and **radar info which is something quite different.**

You might end up with a more custom, or some more complex, hybrid neural network architecture. So, just to be a bit more concrete about what are the

Neural Network examples



Standard NN



Convolutional NN



Recurrent NN

← Supervised Learning Networks

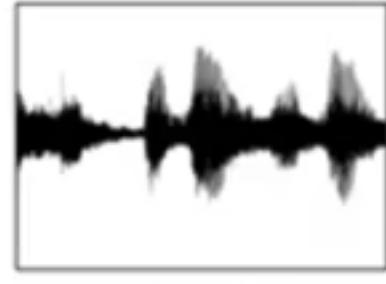
You're using Coursera offline

Supervised Learning

Structured Data

Size	#bedrooms	...	Price (1000\$)
2104	3		400
1600	3		330
2400	3		369
:	:		:
3000	4		540

Unstructured Data



Audio

Image

User Age	Ad Id	...	Click
41	93242		1
80	93287		0
18	87312		1
:	:		:
27	71244		1

Four scores and seven years ago...

Text

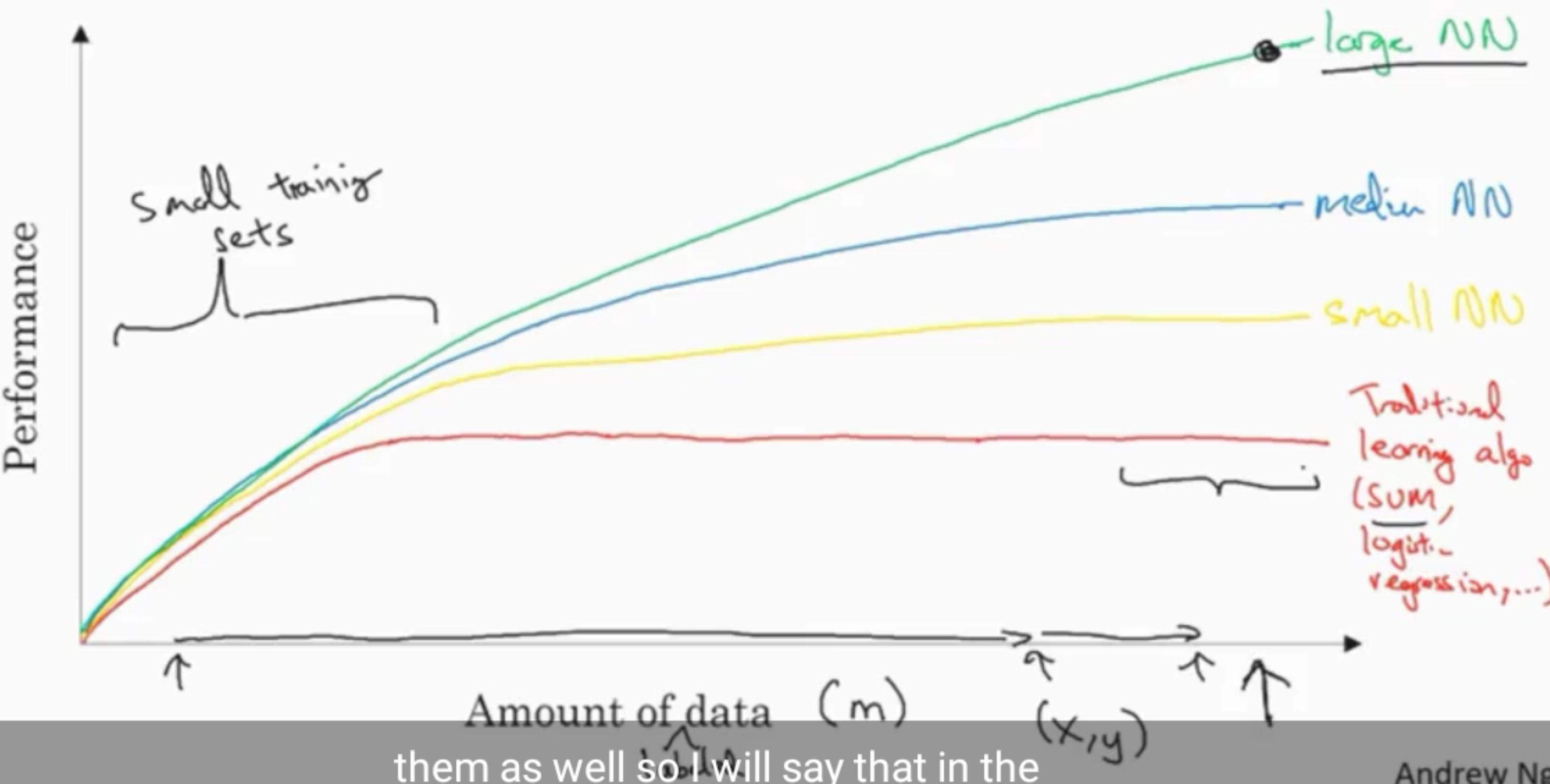
Historically, it has been much harder for computers to make sense of unstructured data compared to structured data. And the fact the human race has evolved to be very good at understanding audio cues as well as images.

And then text was a more recent invention, but people are just really good at interpreting unstructured data.

And so one of the most exciting things about the rise of neural networks is that, thanks to deep learning, thanks to neural networks, computers are now much better at interpreting unstructured data as well

compared to just a few years ago.

Scale drives deep learning progress





Course Resources

Course Resources

Discussion forum

- Questions, technical discussions, bug reports, etc.

Contact us: feedback@deeplearning.ai

Companies: enterprise@deeplearning.ai

Universities: academic@deeplearning.ai

Andrew Ng

I hope you enjoyed this course and to help you complete it I want to make sure that there are few course resources that you know about first if you have any questions or you want to discuss anything with the classmates or the teaching staff including me or if you want to file a bug report the best place to do that is the discussion forum the teaching staff and I will be monitoring that regularly and this is also a good place for you to get answers to your questions from your classmates or if you wish try to answer your classmates questions to get to the discussion forum from



Logistic regression on m examples

$$J=0; \frac{\partial w_1}{\partial w_1}=0; \frac{\partial w_2}{\partial w_2}=0; \frac{\partial b}{\partial b}=0$$

For $i = 1$ to m

$$z^{(i)} = \omega^T x^{(i)} + b$$

$$\alpha^{(i)} = \sigma(z^{(i)})$$

$$J_t = -[y^{(i)} \log \alpha^{(i)} + (1-y^{(i)}) \log (1-\alpha^{(i)})]$$

$$\frac{\partial z^{(i)}}{\partial w_1} = \alpha^{(i)} - y^{(i)}$$

$$\frac{\partial J_t}{\partial w_1} = x_1^{(i)} \frac{\partial z^{(i)}}{\partial w_1}$$

$$\frac{\partial J_t}{\partial w_2} = x_2^{(i)} \frac{\partial z^{(i)}}{\partial w_2}$$

$$\frac{\partial J_t}{\partial b} = \frac{\partial z^{(i)}}{\partial b}$$

$$J_t/m \leftarrow$$

$$\frac{\partial J_t}{\partial w_1}/m; \quad \frac{\partial J_t}{\partial w_2}/m; \quad \frac{\partial J_t}{\partial b}/m \leftarrow$$

$$\frac{\partial J}{\partial w_1} = \frac{\partial J}{\partial w_1}$$

$$w_1 := w_1 - \alpha \frac{\partial J}{\partial w_1}$$

$$w_2 := w_2 - \alpha \frac{\partial J}{\partial w_2}$$

$$b := b - \alpha \frac{\partial J}{\partial b}$$

Vectorization demo

localhost:8888/notebooks/Dropbox/Deep%20Learning%20MOOC/C1W2/Vectorization%20demo.ipynb

jupyter Vectorization demo Last Checkpoint: 5 minutes ago (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Code

```
In [1]: import time

a = np.random.rand(1000000)
b = np.random.rand(1000000)

tic = time.time()
c = np.dot(a,b)
toc = time.time()

print(c)
print("Vectorized version:" + str(1000*(toc-tic)) +"ms")

c = 0
tic = time.time()
for i in range(1000000):
    c += a[i]*b[i]
toc = time.time()

print(c)
print("For loop:" + str(1000*(toc-tic)) + "ms")
```

249946.964024
Vectorized version:1.505136489868164ms

In []:

Just run it again.

← More Vectorization Examples

You're using Coursera offline

Vectors and matrix valued functions

Say you need to apply the exponential operation on every element of a matrix/vector.

$$v = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \rightarrow u = \begin{bmatrix} e^{v_1} \\ e^{v_2} \\ \vdots \\ e^{v_n} \end{bmatrix}$$

$\rightarrow u = np.zeros((n, 1))$

$\rightarrow \boxed{\text{for } i \text{ in range}(n):} \leftarrow$

$\rightarrow u[i] = \text{math.exp}(v[i])$

import numpy as np
 u = np.exp(v) ←
 ↗
 np.log(u)
 np.abs(u)
 np.maximum(v, 0)
 v**2 u/v

Andrew Ng

the vector value functions.

So `np.log(v)` will compute the element-wise log, `np.abs` computes the absolute value, `np.maximum` computes the element-wise maximum to take the max of every element of `v` with 0. `v**2` just takes the element-wise square of each element of `v`. One over `v` takes the element-wise inverse, and so on.

So, whenever you are tempted to write a for-loop take a look, and see if there's **a way to call a NumPy built-in function to do it without that for-loop**.

So, let's take all of these learnings and apply it to our logistic regression



Broadcasting in Python

General Principle

$$\begin{array}{c} (m, n) \\ \text{matrix} \end{array} \quad \begin{array}{c} + \\ - \\ * \\ / \end{array} \quad \begin{array}{c} (1, n) \\ (n, 1) \end{array} \rightsquigarrow (m, n)$$

$$\begin{array}{c} (m, 1) \\ [?] \end{array} \quad + \quad \mathbb{R}$$

If you have an (m,n) matrix and you add or subtract or multiply or divide with a $(1,n)$ matrix, then this will copy it n times into an (m,n) matrix. And then apply the addition, subtraction, and multiplication or division element wise.

If conversely, you were to take the (m,n) matrix and add, subtract, multiply, divide by an $(m,1)$ matrix, then also this would copy it now n times. And turn that into an (m,n) matrix and then apply the operation element wise.

Just one of the broadcasting, which is if you have an $(m,1)$ matrix, so that's really a column vector like $[1,2,3]$, and

A note on pyth...umpy vectors

You're using Coursera offline

The screenshot shows a Jupyter Notebook interface with the title "Python-Numpy vectors". The code cell In [1] contains:

```
import numpy as np  
a = np.random.randn(5)
```

The output of In [2] is:

```
[ 0.50290632 -0.29691149  0.95429604 -0.82126861 -1.46269164]
```

The output of In [3] is:

```
(5,)
```

The output of In [4] is:

```
[ 0.50290632 -0.29691149  0.95429604 -0.82126861 -1.46269164]
```

The output of In [5] is:

```
4.06570109321
```

In the current input cell In [6], the user has typed "pr" and is awaiting further input.

So what I would recommend is that when you're coding new networks, that you just not use data structures where the shape is 5, or n, rank 1 array. Instead, if you set a to be this, (5,1), then this commits a to be (5,1) column vector.

And whereas previously, a and a transpose looked the same, it becomes now a transpose, now a transpose is a row vector. Notice one subtle difference. In this data structure, there are two square brackets when we print a transpose. Whereas previously, there was one square bracket.



Python/numpy vectors

```
a = np.random.randn(5)  
a.shape = (5,)  
"rank 1 array"
```

{ Don't use

```
a = np.random.randn(5, 1) → a.shape = (5, 1)
```

column
vector ✓

```
a = np.random.randn(1, 5) → a.shape = (1, 5)
```

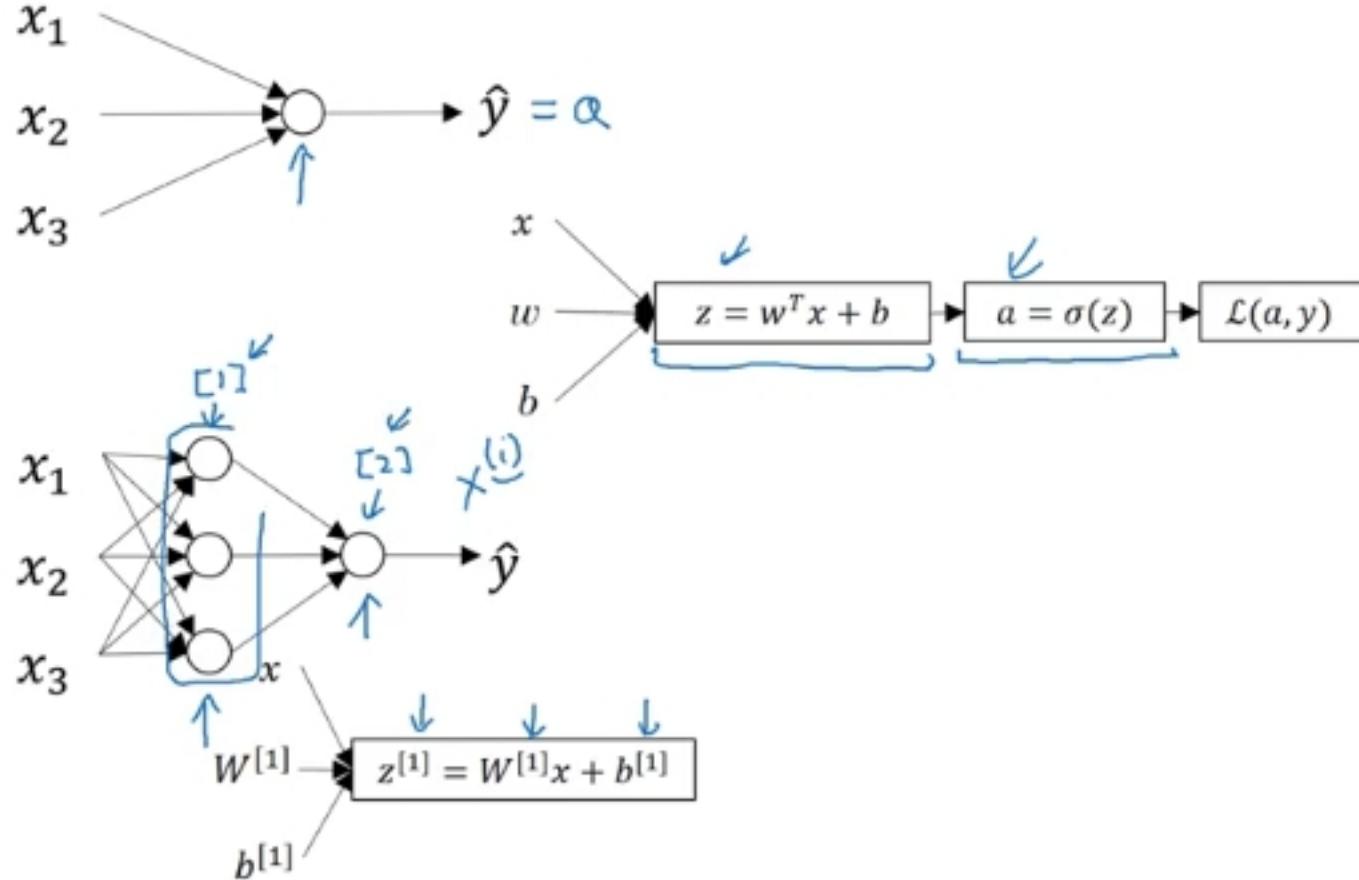
row
vector. ✓

```
assert(a.shape == (5, 1)) ←  
a = a.reshape((5, 1))
```

← Neural Networks Overview

You're using Coursera offline

What is a Neural Network?



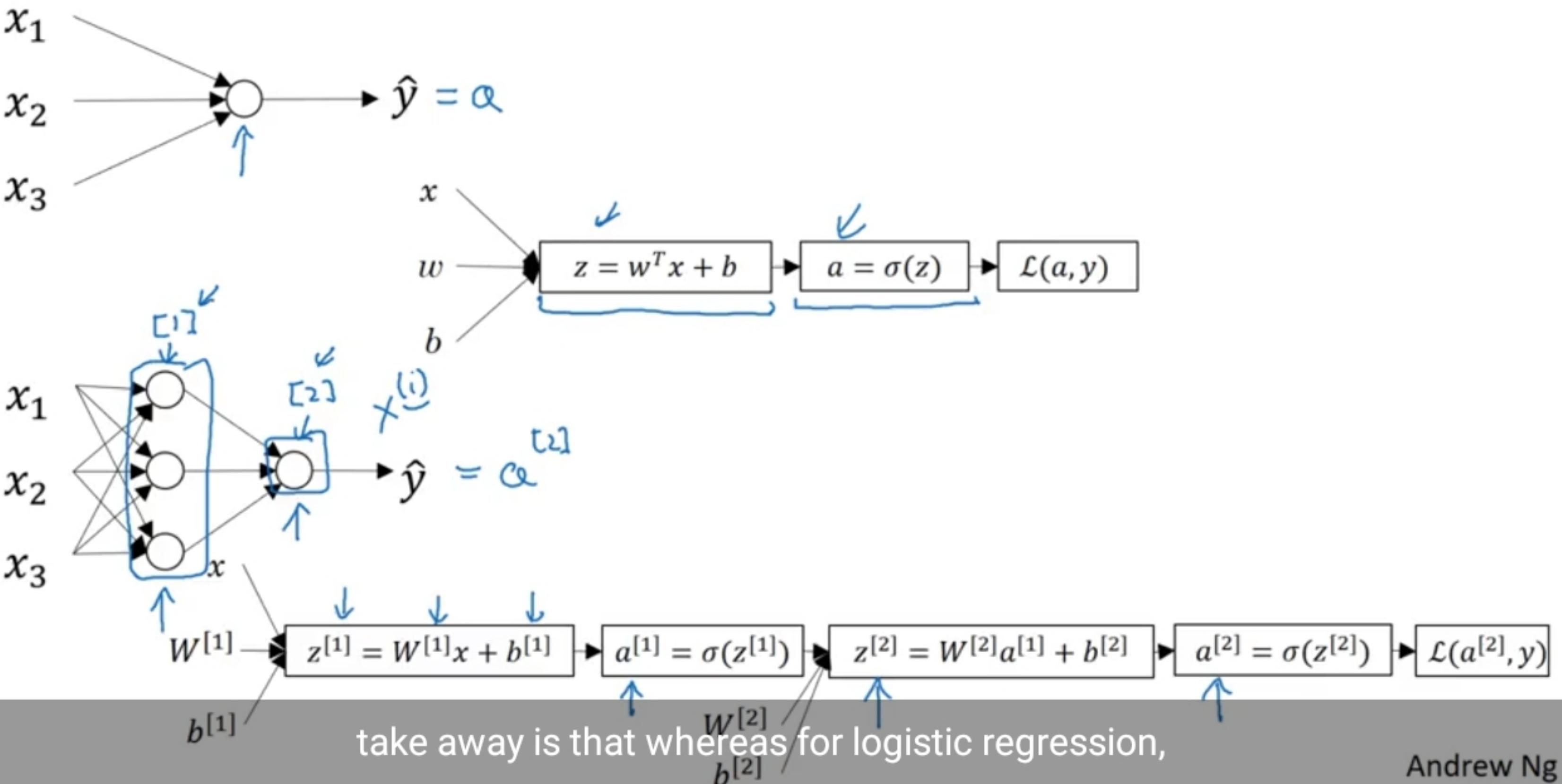
Andrew Ng

The superscript square brackets, like we have here, are not to be confused with the superscript round brackets which we use to refer to individual training examples.

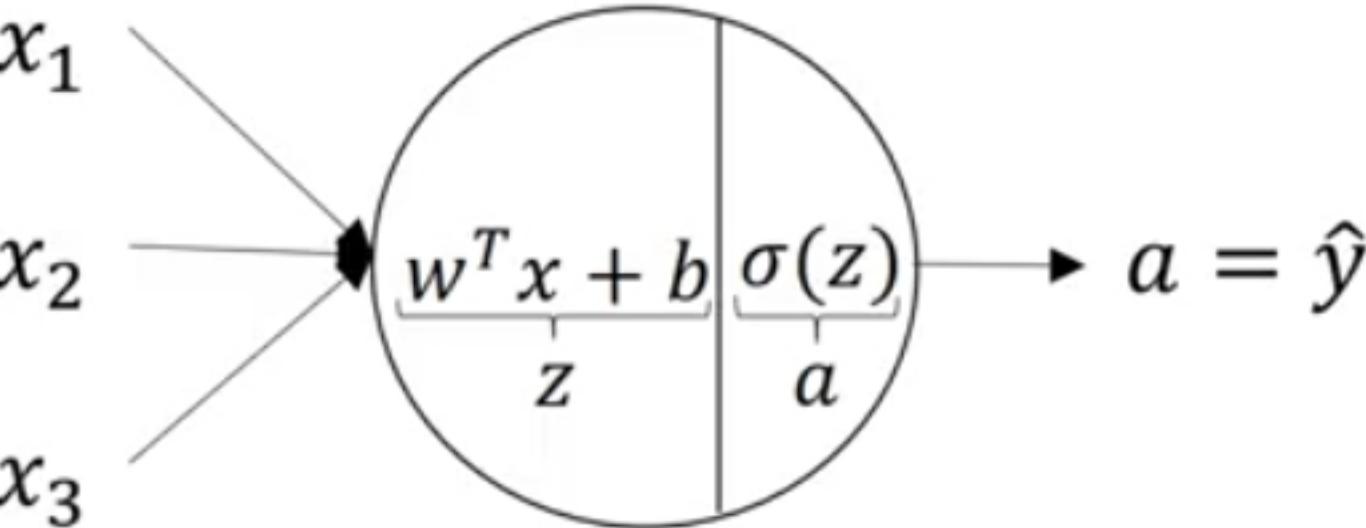
So, whereas x superscript round bracket I refer to the i th training example, superscript square bracket one and two refer to these different layers; **layer one and layer two in this neural network.**

But so going on, after computing z_1 similar to logistic regression, there'll be a computation to compute a_1 , and that's just sigmoid of z_1 , and then you compute z_2 using another linear

What is a Neural Network?

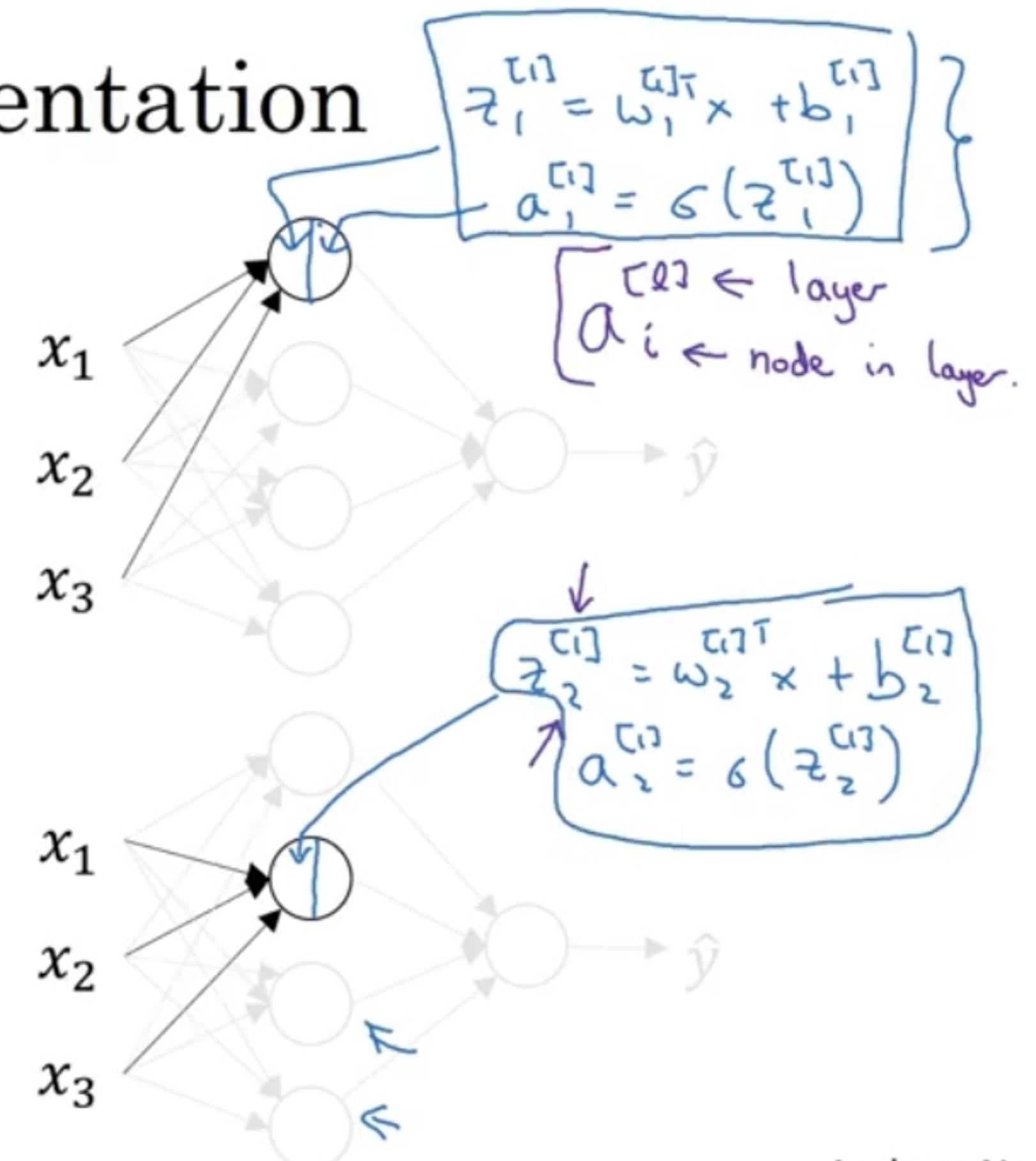


Neural Network Representation

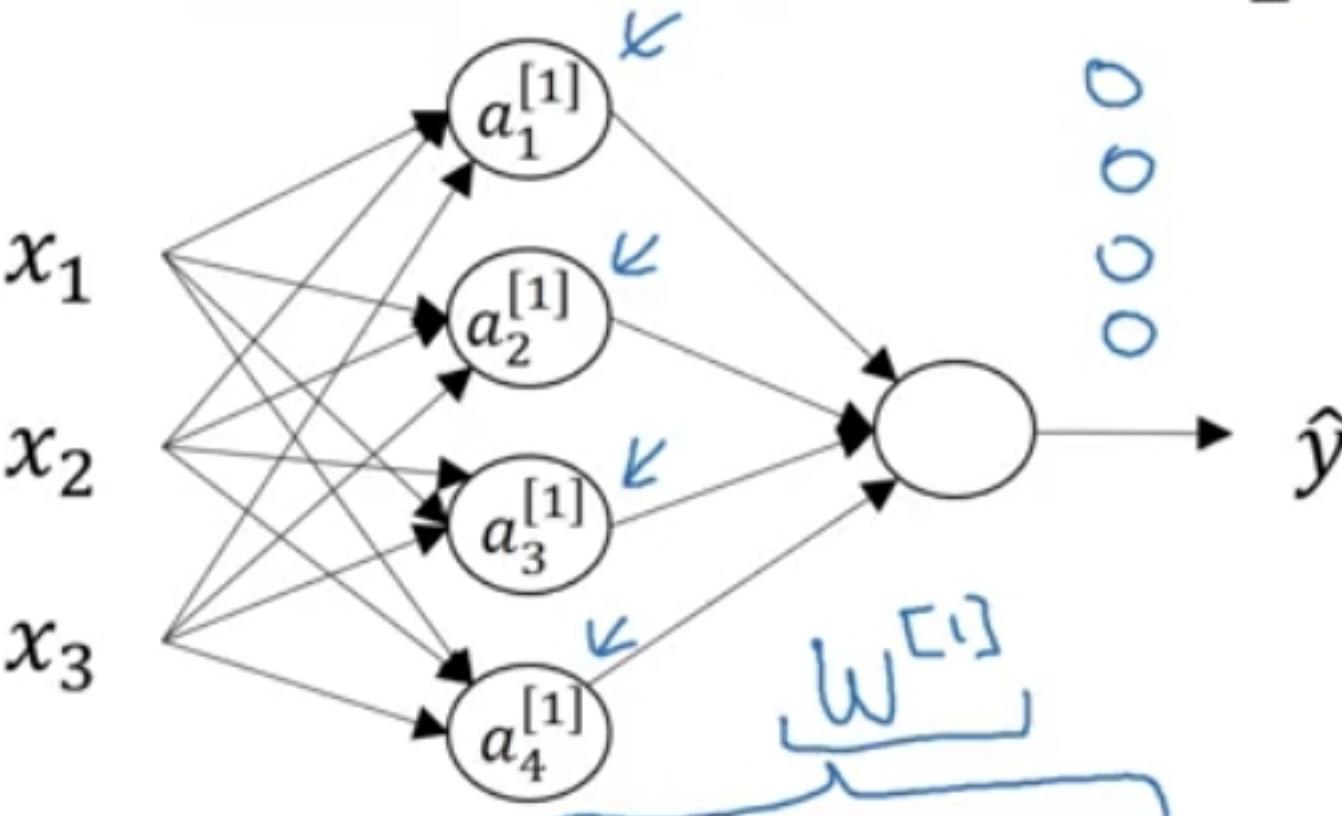


$$z = w^T x + b$$

$$a = \sigma(z)$$



Neural Network Representation



$$\rightarrow z^{[1]} = \begin{bmatrix} w_1^{T,1} \\ w_2^{T,1} \\ w_3^{T,1} \\ -w_4^{T,1} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

$$\rightarrow \alpha^{(1)} = \begin{bmatrix} \alpha_1^{(1)} \\ \vdots \\ \alpha_{43}^{(1)} \end{bmatrix} = G(z^{(1)})$$

representation

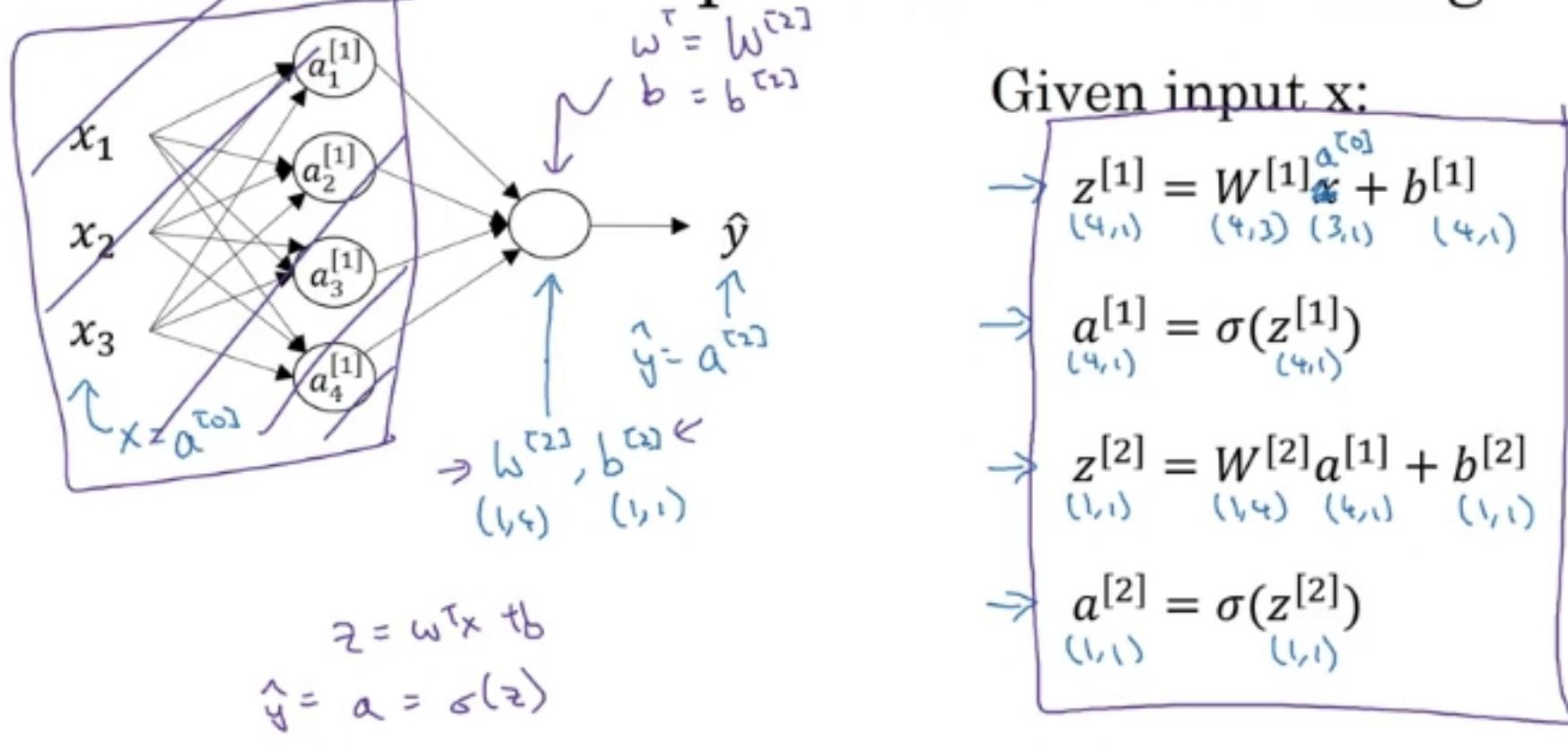
$$\begin{aligned}
 z_1^{[1]} &= w_1^{[1]T} x + b_1^{[1]}, & a_1^{[1]} &= \sigma(z_1^{[1]}), \\
 z_2^{[1]} &= w_2^{[1]T} x + b_2^{[1]}, & a_2^{[1]} &= \sigma(z_2^{[1]}), \\
 z_3^{[1]} &= w_3^{[1]T} x + b_3^{[1]}, & a_3^{[1]} &= \sigma(z_3^{[1]}), \\
 z_4^{[1]} &= w_4^{[1]T} x + b_4^{[1]}, & a_4^{[1]} &= \sigma(z_4^{[1]}) \\
 \end{aligned}$$

$$+ \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} \rightarrow w_1^{[1]T} x + b_1^{[1]} \\ \rightarrow w_2^{[1]T} x + b_2^{[1]} \\ \rightarrow w_3^{[1]T} x + b_3^{[1]} \\ \rightarrow w_4^{[1]T} x + b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

← Computing a Network's Output

You're using Coursera offline

Neural Network Representation learning



Andrew Ng

When you have a neural network with one hidden layer, what you need to implement, is to compute this output is just these four equations.

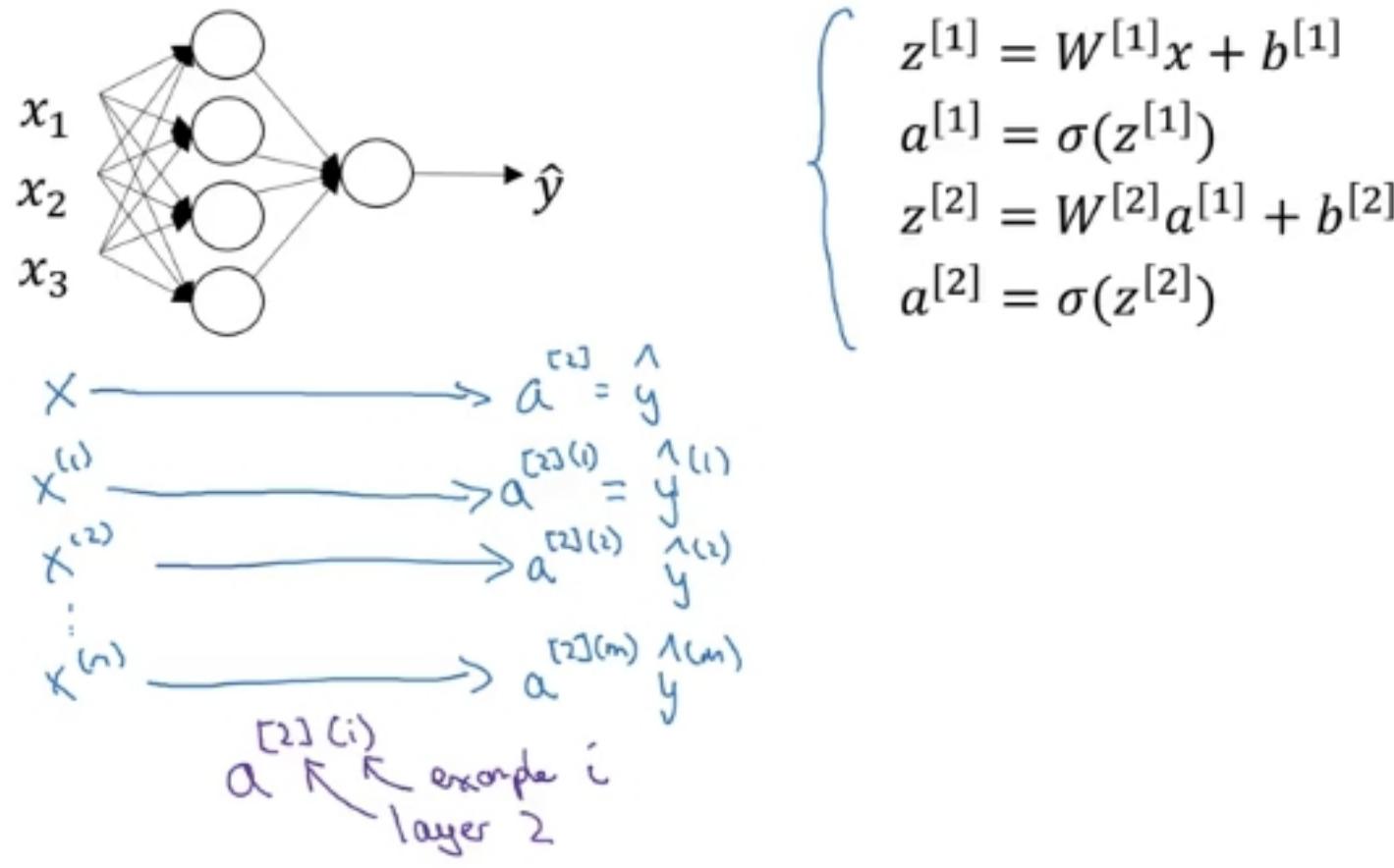
You can think of this as a vectorized implementation of computing **the output of first these for logistic regression units in the hidden layer**, that's what this does, and then this logistic regression in the output layer which is what this does.

I hope this description made sense, but the takeaway is to compute the output of this neural network, all you need is those four lines of code. So now, you've seen how given a single **input feature vector a** you can with

← Vectorizing across multiple examples

You're using Coursera offline

Vectorizing across multiple examples



Andrew Ng

The round bracket i refers to training example i , and the square bracket 2 refers to layer 2, okay. So that's how the square bracket and the round bracket indices work.

And so to suggest that if you have an unvectorized implementation and want to compute the predictions of all your training examples, you need to do for $i = 1$ to m . Then basically implement these four equations, right?

You need to make a $z[1](i) = W[1]x(i) + b[1]$, $a[1](i) = \sigma(z[1](i))$, $z[2](i) = w[2]a[1](i) + b[2]$ and $a[2](i) = \sigma(z[2](i))$.

← Vectorizing across multiple examples

You're using Coursera offline

Vectorizing across multiple examples

for $i = 1$ to m :

$$z^{[1](i)} = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]}a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$X = \begin{bmatrix} & & & \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ & & & \end{bmatrix}$$

\uparrow \uparrow
 (n_x, m)

$$z^{[1]} = W^{[1]}X + b^{[1]}$$

$$\rightarrow A^{[1]} = \sigma(z^{[1]})$$

$$\rightarrow z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$\rightarrow A^{[2]} = \sigma(z^{[2]})$$

$$z^{[1]} = \begin{bmatrix} z^{1} & z^{[1](2)} & \dots & z^{[1](m)} \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} a^{1} & a^{[1](2)} & \dots & a^{[1](m)} \end{bmatrix}$$

Andrew Ng

example, when you sweep from left to right you're scanning through the training cells. And vertically this vertical index corresponds to different nodes in the neural network.

So for example, this node, this value at the top most, top left most corner of the mean corresponds to the activation of the first heading unit on the first training example.

One value down corresponds to the activation in the second hidden unit on the first training example, then the third heading unit on the first training sample and so on. So as you scan down this is your indexing to the hidden units number.

Vectorizing across multiple examples

for $i = 1$ to m :

$$z^{[1]}(i) = W^{[1]}x^{(i)} + b^{[1]}$$

$$a^{[1]}(i) = \sigma(z^{[1]}(i))$$

$$z^{[2]}(i) = W^{[2]}a^{[1]}(i) + b^{[2]}$$

$$a^{[2]}(i) = \sigma(z^{[2]}(i))$$

$$X = \begin{bmatrix} & & & \\ x^{(1)} & x^{(2)} & \dots & x^{(m)} \\ & & & \end{bmatrix}$$

↑
 (n_x, m)

↑
hidden units.
→ many samples

$$z^{[1]} = W^{[1]}X + b^{[1]}$$

$$\rightarrow A^{[1]} = \sigma(z^{[1]})$$

$$\rightarrow z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$

$$\rightarrow A^{[2]} = \sigma(z^{[2]})$$

$$z^{[1]} = \begin{bmatrix} & & & \\ z^{[1]}(1) & z^{[1]}(2) & \dots & z^{[1]}(m) \\ & & & \end{bmatrix}$$

$$A^{[1]} = \begin{bmatrix} & & & \\ a^{[1]}(1) & a^{[1]}(2) & \dots & a^{[1]}(m) \\ & & & \end{bmatrix}$$

↑
hidden units
Andrew Ng

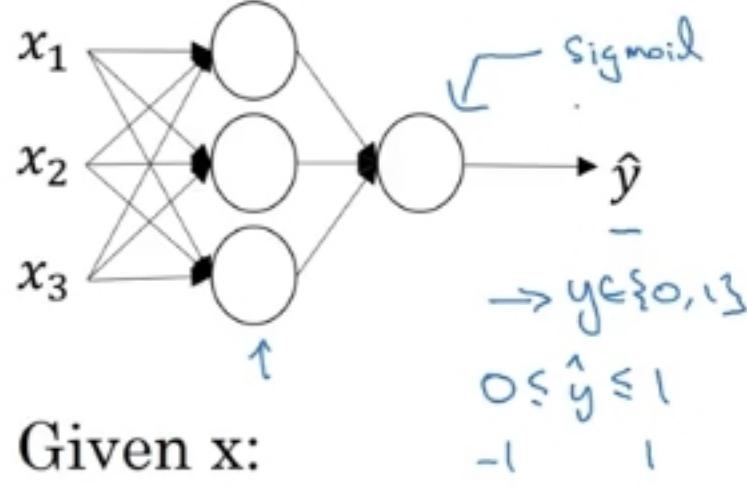


Activation functions

You're using Coursera offline

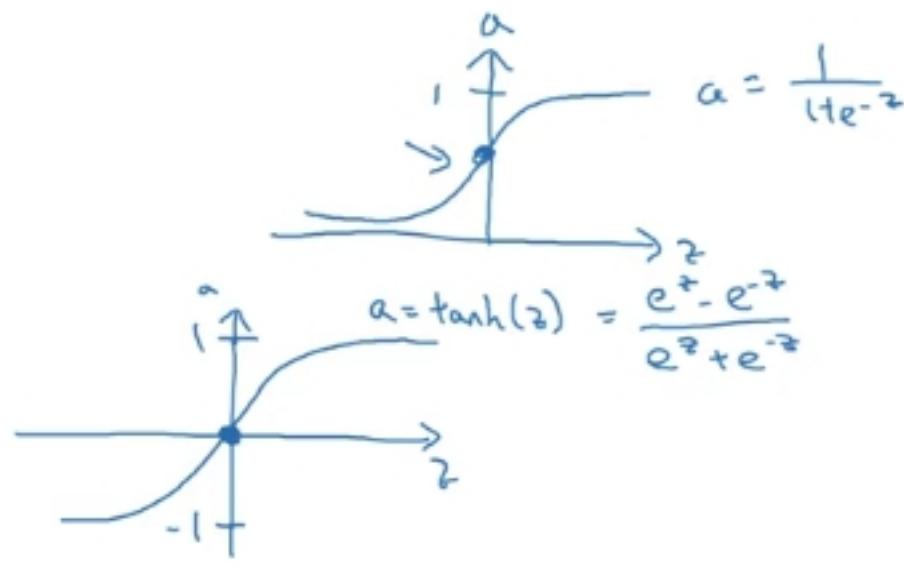
Activation functions

$$\downarrow g(z^{[1]}) = \tanh(z^{[1]})$$



Given x:

$$-1 \quad 1$$



$$\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]} \\ \rightarrow a^{[1]} &= \sigma(z^{[1]}) \quad g(z^{[1]}) \\ z^{[2]} &= W^{[2]}a^{[1]} + b^{[2]} \\ \rightarrow a^{[2]} &= \sigma(z^{[2]}) \quad g(z^{[2]}) \end{aligned}$$

Andrew Ng

So the one exception where I would use the sigmoid activation function is when you are using binary classification, in which case you might use the sigmoid activation function for the output layer. So g of z 2 here is equal to sigma of z 2.

And so what you see in this example is where you might have a tanh activation function for the hidden layer, and sigmoid for the output layer. So deactivation functions can be different for different layers.

And sometimes to note that activation functions are different for different layers, we might use these square brackets as well to