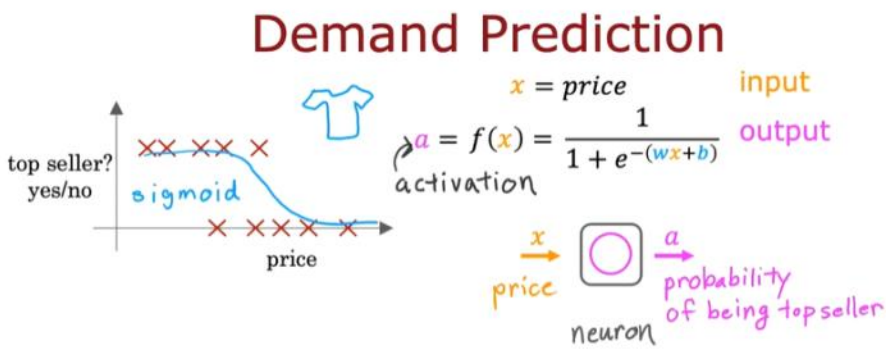
Module 1

01 September 2025 22:42

More data to train as everything digital and better types of models



In this we will use f(x) as a i.e activation which is a nerual network terminology which Refers to how much a neuron is sending high output to other nerons donwstream from it.

Here the circle are nerons and basically think it as a computer which took all the inputs such as price ans shipping cost and give output of lets say affordibility . Layer ia a groups of neutron it can hame many neutorns or a single neutron . Last layer is called output layer which outputs the final thing . The factors required to predict the final thing is called activatiions

The way a neural network is implemented in
practice each neuron in a certain layer;
say this layer in the middle,
will have access to every feature,
to every value from the previous layer,
from the input layer which is why I'm now drawing arrows
from every input feature to every one
of these neurons shown here in the middle.
You can imagine that if you're trying to predict
affordability and it knows

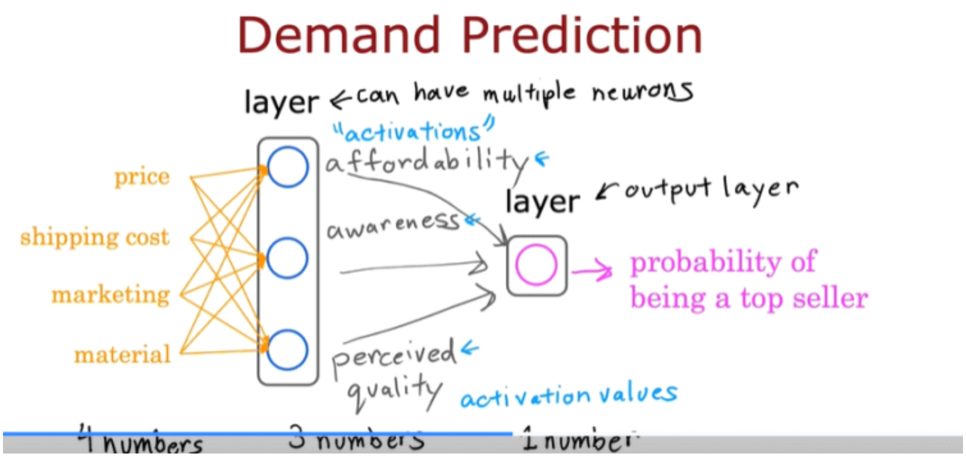
what's the price shipping cost marketing and material,
may be you'll learn to ignore
marketing and material and just figure
out through setting the parameters appropriately to only
focus on the subset of features that are
most relevant to affordability.
To further simplify the notation and
the description of this neural network I'm going
to take these four input features
and write them as a vector x,
and we're going to view the neural network as having
four features that comprise this feature vector x.
This feature vector is fed to this layer in
the middle which then computes three activation values.
That is these numbers and
these three activation values
in turn becomes another vector which
is fed to this final output layer that
finally outputs the probability
of this t-shirt to being a top seller.
That's all a neural network is.
It has a few layers where each layer inputs
a vector and outputs another vector of numbers
In a training set,
you get to observe both x and y.
Your data set tells you what is x and what is y,
and so you get data that tells
you what are the correct inputs and the correct outputs.
But your dataset doesn't tell you what
are the correct values for affordability,
awareness, and perceived quality.
The correct values for those are hidden.
You don't see them in the training set,
which is why this layer in
the middle is called a hidden layer.

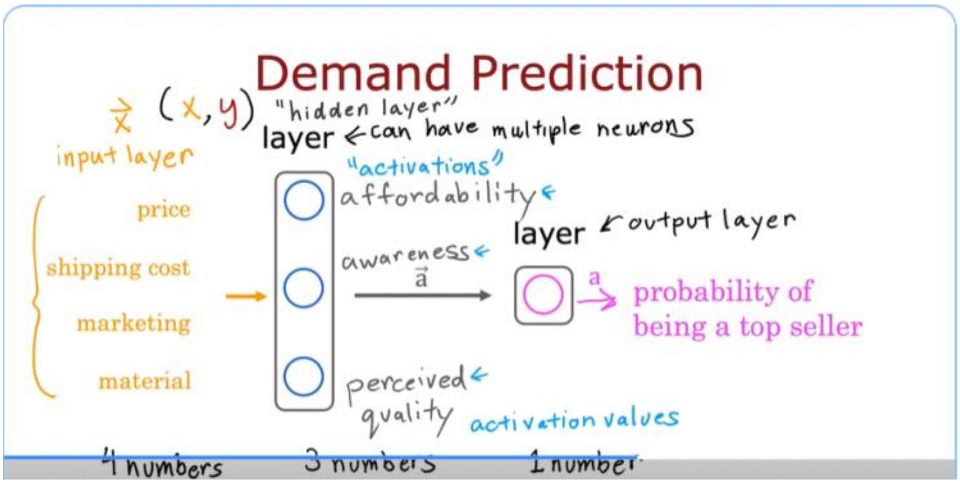
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/MsbrF/demand-prediction>

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/MsbrF/demand-prediction>

.
Play

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/MsbrF/demand-prediction>



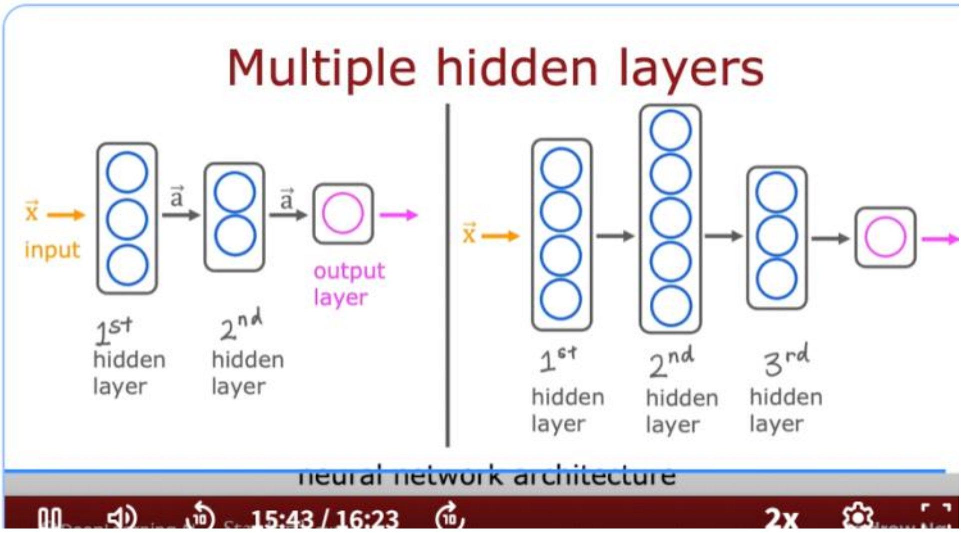


One way to think of this neural network
is, just logistic regression.
But as a version of logistic regression,
they can learn its own features that
makes it easier to make accurate predictions.
In fact, you might remember from the previous course,
this housing example where we said
that if you want to predict the price of the house,
you might take the frontage or
the width of lots and multiply that
by the depth of a lot to construct
a more complex feature,
x_1 times x_2,
which was the size of the lawn.
There we were doing manual feature engineering
where we had to look at the features x_1
and x_2 and decide by hand how to combine
them together to come up with better features.
What the neural network does is instead of you
needing to manually engineer the features,
it can learn, as you'll see later,
its own features to make
the learning problem easier for itself.
This is what makes
neural networks one of
the most powerful learning algorithms in the world today.
To summarize, a neural network,
does this, the input layer has a vector of features,
four numbers in this example,

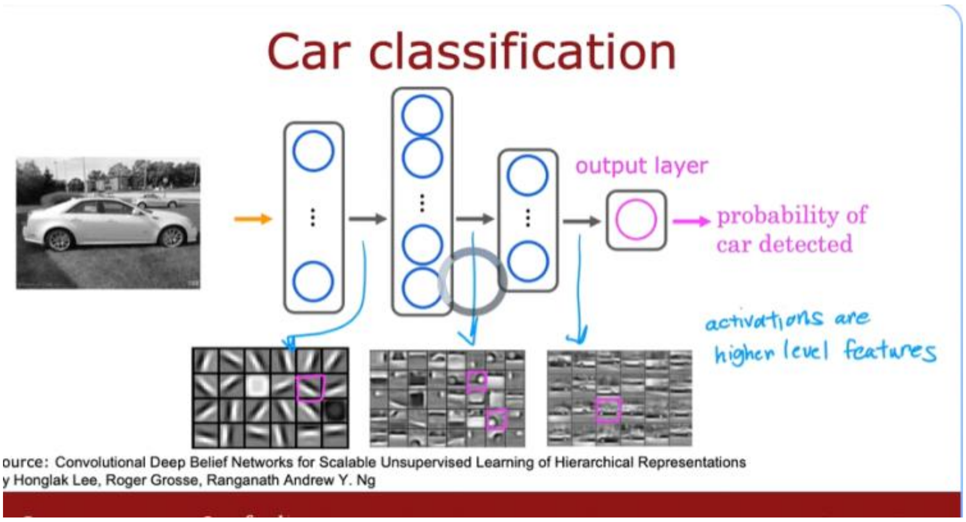
it is input to the hidden layer,
which outputs three numbers.
I'm going to use a vector to denote
this vector of activations
that this hidden layer outputs.
Then the output layer takes its input to
three numbers and outputs one number,
which would be the final activation,
or the final prediction of the neural network.

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/MsbrF/demand-prediction>

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/MsbrF/demand-prediction>



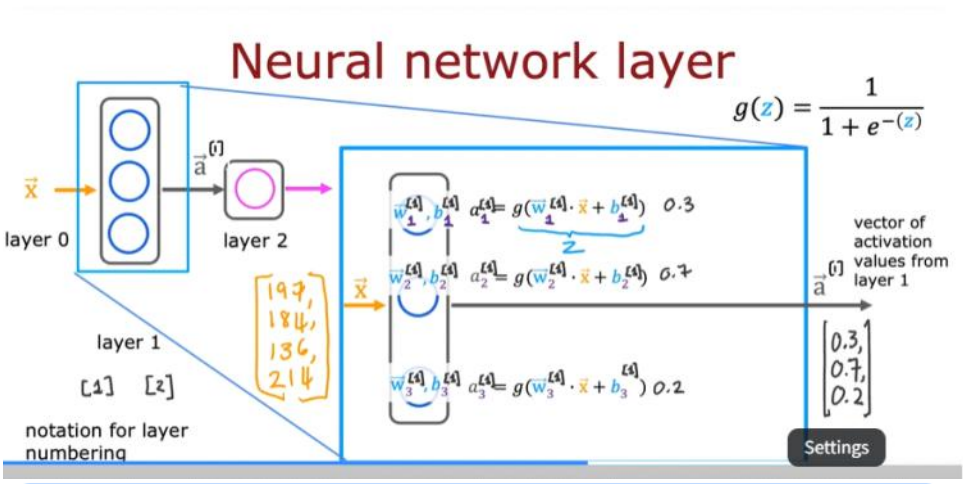
This is called multilayer preceptron.

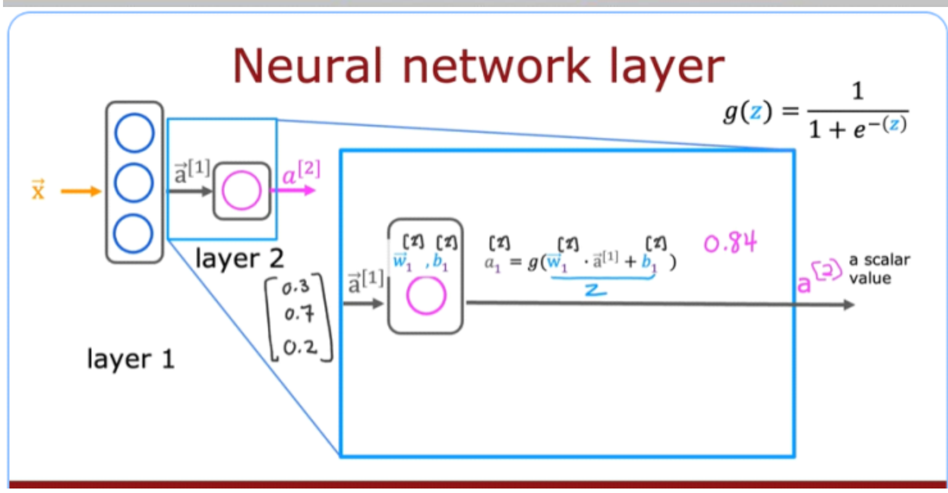


This hidden layer inputs four numbers and
these four numbers are inputs to each of three neurons.
Each of these three neurons is just implementing
a little logistic regression unit
or a little bit logistic regression function
Take this first neuron.
It has two parameters, w and b.
In fact, to denote that,
this is the first hidden unit,
I'm going to subscript this as w_1, b_1.
What it does is I'll output some activation value a,
which is g of w_1 in a product with x plus b_1,
where this is the familiar z value that you
have learned about in
logistic regression in the previous course,
and g of z is the familiar logistic function,
1 over 1 plus e to the negative z.
Maybe this ends up being a number 0.3 and
that's the activation value a of the first neuron.
To denote that this is the first neuron,
I'm also going to add a subscript a_1 over here,
and so a_1 may be a number like 0.3.
There's a 0.3 chance of
this being highly affordable based on the input features
Similarly, it computes
an activation value a_3 equals g of
w_3 dot product x plus b_3 and that may be say, 0.2.
In this example, these three neurons output 0.3,
0.7, and 0.2,
and this vector of three numbers
becomes the vector of activation values a,
that is then passed to
the final output layer of this neural network.
I'm going to use superscript square bracket
1 to index into different layers.
In particular, a superscript
in square brackets 1, I'm going to use,
that's a notation to denote the output of
layer 1 of this hidden layer of this neural network,
and similarly, w_1,
b_1 here are the parameters of

the first unit in layer 1 of the neural network,
so I'm also going to add
a superscript in square brackets 1
I know maybe this notation
is getting a little bit cluttered.
But the thing to remember is whenever
you see this superscript square bracket 1,
that just refers to a quantity
that is associated with layer 1 of the neural network.
If you see superscript square bracket 2,
that refers to a quantity associated with layer
2 of the neural network
and similarly for other layers as well,
including layer 3,
layer 4 and so on for neural networks with more layers.
That's the computation of layer 1 of this neural network.
Its output is this activation vector,
a^[1] and I'm going to copy this over here
because this output a_1 becomes the input to layer 2.
Now let's zoom into
the computation of layer 2 of this neural network,
which is also the output layer.
The input to layer 2 is the output of layer 1,
so a_1 is this vector 0.3, 0.7,
0.2 that we just computed
on the previous part of this slide.

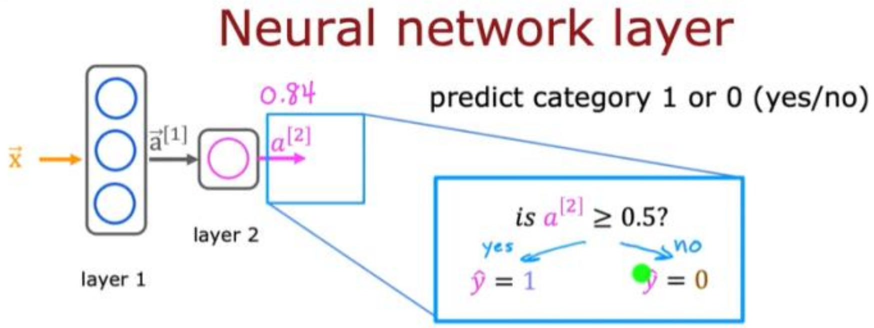
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/z5sks/neural-network-layer>





Similar in layer 2

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/z5sks/neural-network-layer>



If we want to predict the yes and no then we can do this

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/z5sks/neural-network-layer>

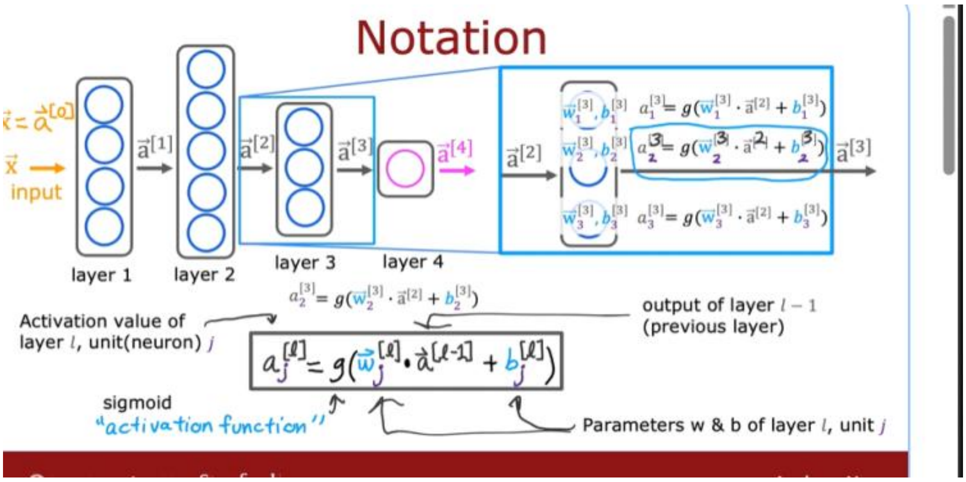
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/z5sks/neural-network-layer>

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/z5sks/neural-network-layer>

This gives you the activation of layer l unit j,
where the superscript in square brackets l denotes
layer l and a subscript j denotes unit j.
When building neural networks,
unit j refers to the jth neuron,
so we use those terms a little bit
interchangeably where each unit
is a single neuron in the layer.
G here is the sigmoid function.
In the context of a neural network,
g has another name,
which is also called the activation function,
because g outputs this activation value.
When I say activation function,
I mean this function g here.
So far, the only activation function you've seen,
this is a sigmoid function but next week,
we'll look at when other functions,
then the sigmoid function can be
plugged in place of g as well..
The activation function is just that function
that outputs these activation values.
Just one last piece of notation.

In order to make all this notation consistent,
I'm also going to give
the input vector X and another name which is a_0,
so this way, the same equation
also works for the first layer,
where when l is equal to 1,
the activations of the first layer,
that is a_1,
would be the sigmoid times
the weights dot-product with a_0,
which is just this input feature vector X.
With this notation, you now know how to compute
the activation values of any layer in
a neural network as a function of
the parameters as well as
the activations of the previous layer.
You now know how to compute the activations of
any layer given the activations of the previous layer.

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/a5AfY/more-complex-neural-networks>



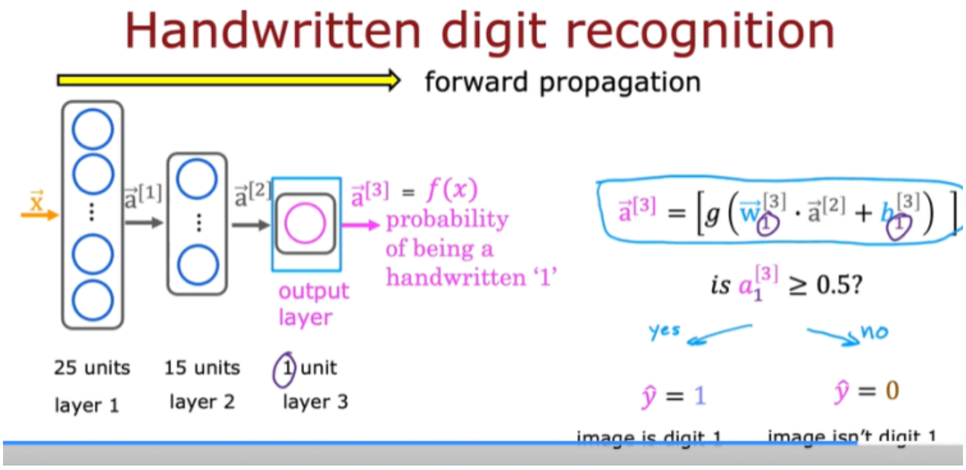
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/a5AfY/more-complex-neural-networks>

Play video

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/a5AfY/more-complex-neural-networks>

Because this computation goes from left to right, you start from x and compute a1, then a2, then a3.
This album is also called forward propagation because you're
propagating the activations of the neurons.
So you're making these computations in the forward direction from left to right.
And this is in contrast to a different algorithm called backward propagation or
back propagation, which is used for learning.
And that's something you learn about next week.
And by the way, this type of neural network architecture where you have more
hidden units initially and
then the number of hidden units decreases as you get closer to the output layer.
There's also a pretty typical choice when choosing neural network architectures.

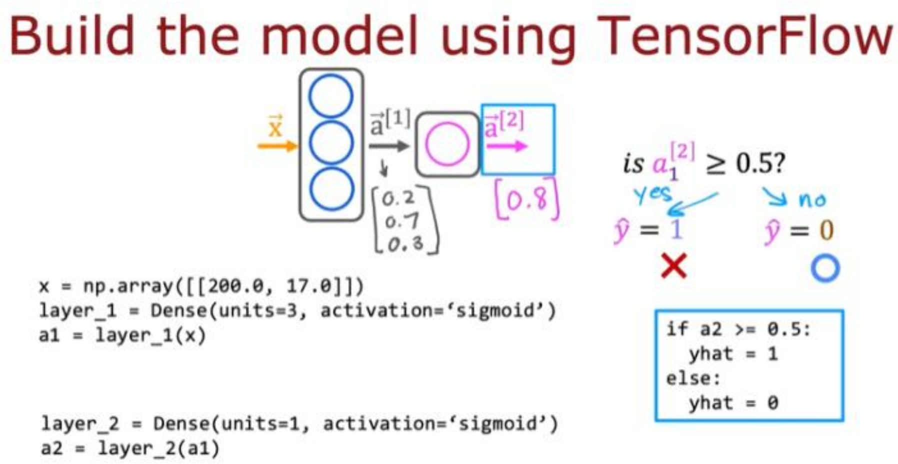
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/vYsrR/inference-making-predictions-forward-propagation>



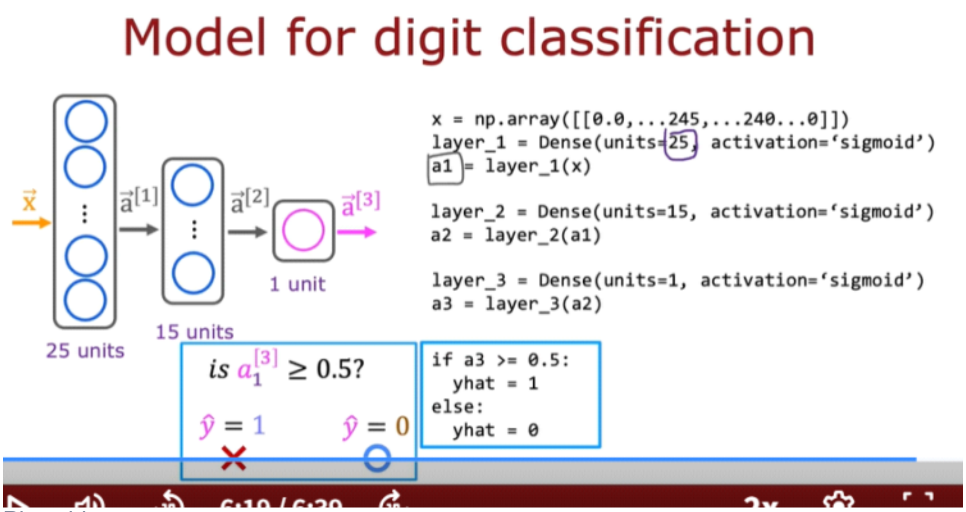
We're going to set x to be an array of two numbers.
The input features 200 degrees celsius and 17 minutes.
Then you create Layer 1
as this first hidden layer, the neural network,
as dense open parenthesis units 3,
that means three units or three hidden units in
this layer using as
the activation function, the sigmoid function.
Dense is another name for
the layers of a neural network
that we've learned about so far.
As you learn more about neural networks,
you learn about other types of layers as well.
But for now, we'll just use the dense layer,
which is the layer type you've learned about in
the last few videos for all of our examples.
Next, you compute a1 by taking Layer 1,
which is actually a function,
and applying this function Layer 1 to the values of x.
That's how you get a1,
which is going to be a list of three numbers
because Layer 1 had three units.
So a1 here may,
just for the sake of illustration,
be 0.2, 0.7, 0.3.
Next, for the second hidden layer,
Layer 2, would be dense.
Now this time it has one unit and
again to sigmoid activation function,
and you can then compute a2 by applying
this Layer 2 function to
the activation values from Layer 1 to a1.
That will give you the value of a2,
which for the sake of illustration is maybe 0.8.
Finally, if you wish to threshold it at 0.5,
then you can just test if a2
is greater and equal to 0.5 and

set y-hat equals to
one or zero positive or negative cross accordingly.
That's how you do inference in
the neural network using TensorFlow.
There are some additional details
that I didn't go over here,
such as how to load
the TensorFlow library and how to also
load the parameters w and b of the neural network.
But we'll go over that in the lab.
Please be sure to take a look at the lab.
But these are the key steps for forward propagation in
how you compute a1 and a2 and optionally threshold a2.

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/rJMKC/inference-in-code>



From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/rJMKC/inference-in-code>



Play video

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/rJMKC/inference-in-code>

So whereas in course one when we're working with linear regression and
logistic regression, we use these 1D vectors to represent the input features x.
With TensorFlow the convention is to use matrices to represent the data.
And why is there this switching conventions?
Well it turns out that TensorFlow was designed to handle very large datasets and
by representing the data in matrices instead of 1D arrays,
it lets TensorFlow be a bit more computationally efficient internally.
So going back to our original example for the first training, example in
this dataset with features 200°C in 17 minutes, we were represented like this.
And so this is actually a 1 x 2 matrix that happens to have one row and
two columns to store the numbers 217.
And in case this seems like a lot of details and really complicated
conventions, don't worry about it all of this will become clearer.
And you get to see the concrete implementations of the code yourself in
the optional labs and in the practice labs.
Going back to the code for carrying out for propagation or
influence in the neural network.
When you compute a1 equals layer 1 applied to x, what is a1?
Well, a1 is actually going to be because the three numbers,
is actually going to be a 1 x 3 matrix.
And if you print out a1 you will get something like this
is tf.tensor 0.2, 0.7, 0.3 as a shape of 1 x 3,
1, 3 refers to that this is a 1 x 3 matrix.
And this is TensorFlow's way of saying that this is a floating point number
meaning that it's a number that can have a decimal point represented using
32 bits of memory in your computer, that's where the float 32 is.
And what is the tensor?
A tensor here is a data type that the TensorFlow team had created in order to
store and carry out computations on matrices efficiently.
So whenever you see tensor just think of that matrix on these few slides.
Technically a tensor is a little bit more general than the matrix but for
the purposes of this course,
think of tensor as just a way of representing matrices.

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/eTL7B/data-in-tensorflow>

Well, a1 is actually going to be because the three numbers,
is actually going to be a 1 x 3 matrix.
And if you print out a1 you will get something like this
is tf.tensor 0.2, 0.7, 0.3 as a shape of 1 x 3,
1, 3 refers to that this is a 1 x 3 matrix.
And this is TensorFlow's way of saying that this is a floating point number
meaning that it's a number that can have a decimal point represented using
32 bits of memory in your computer, that's where the float 32 is.
And what is the tensor?
A tensor here is a data type that the TensorFlow team had created in order to
store and carry out computations on matrices efficiently.
So whenever you see tensor just think of that matrix on these few slides.
Technically a tensor is a little bit more general than the matrix but for
the purposes of this course,
think of tensor as just a way of representing matrices.

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/eTL7B/data-in-tensorflow>

Feature vectors

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

```
x = np.array([[200.0, 17.0]])
[[200.0, 17.0]]
```

[200.0 17.0] 1 x 2

Note about numpy arrays

```
x = np.array([[200, 17]])
```

$\begin{bmatrix} 200 & 17 \end{bmatrix}$ 1 x 2

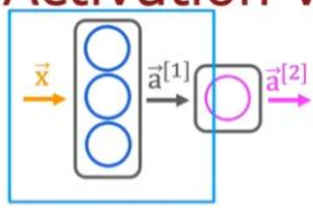
```
x = np.array([[200],
              [17]])
```

$\begin{bmatrix} 200 \\ 17 \end{bmatrix}$ 2 x 1

```
x = np.array([200,17])
```

1D "Vector"

Activation vector




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

[[0.2, 0.7, 0.3]] 1 x 3 matrix

```
tf.Tensor([[0.2 0.7 0.3]], shape=(1, 3), dtype=float32)
```

```
a1.numpy()
```

```
array([[0.2, 0.7, 0.3]], dtype=float32)
```

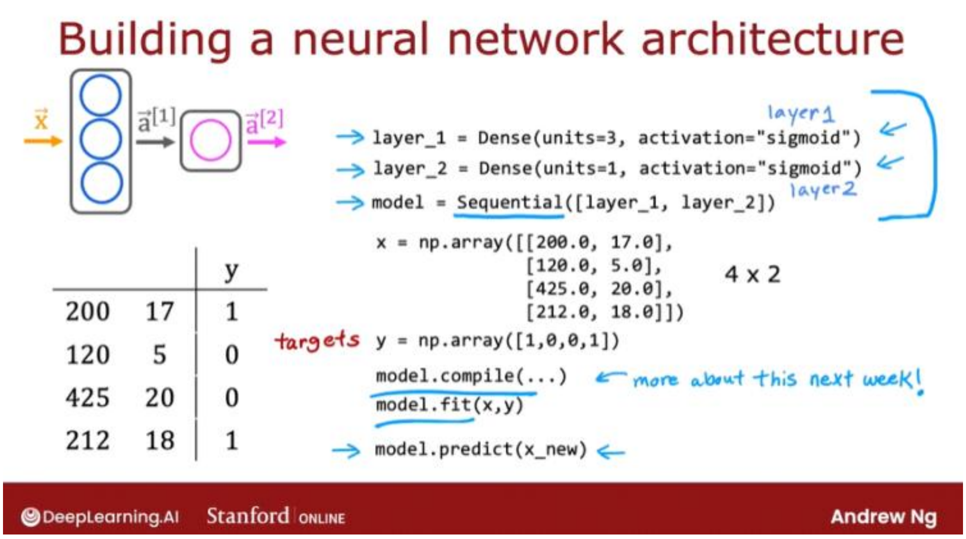
Pl

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/eTL7B/data-in-tensorflow>

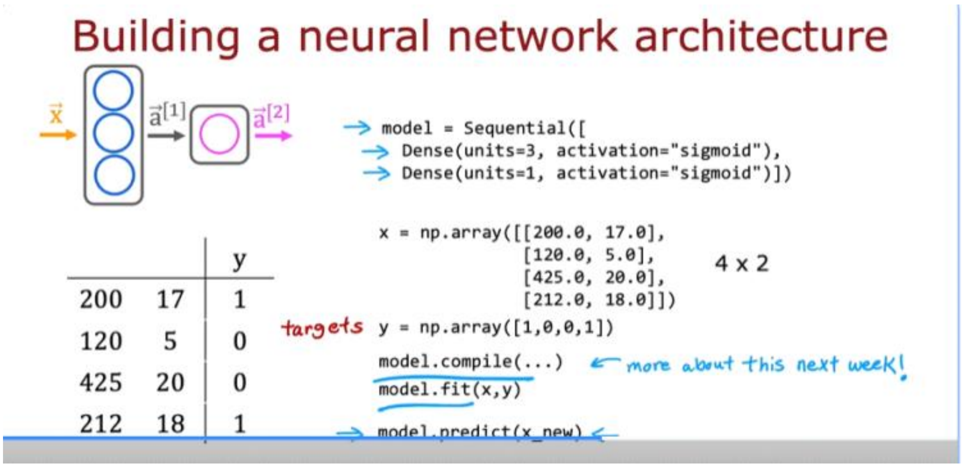
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/eTL7B/data-in-tensorflow>

If you want to do forward prop, you initialize the data X create layer one then compute a one, then create layer two and compute a two.
So this was an explicit way of carrying out forward prop one layer of computation at the time.
It turns out that tensor flow has a different way of implementing forward prop as well as learning.
Let me show you a different way of building a neural network in TensorFlow, which is that same as before you're going to create layer one and create layer two.
But now instead of you manually taking the data and passing it to layer one and then taking the activations from layer one and pass it to layer two.
We can instead tell tensor flow that we would like it to take layer one and layer two and string them together to form a neural network.
That's what the sequential function in TensorFlow does
Let's say you have a training set like this on the left.
This is for the coffee example.
You can then take the training data as inputs X and put them into a numpy array.
This here is a four by two matrix and the target labels.
Y can then be written as follows.
And this is just a one dimensional array of length four
Y this set of targets can then be stored as a 1-D array like this 1001 corresponding to four train examples.
And it turns out that given the data, X and Y stored in this matrix X and this array, Y.
If you want to train this neural network, all you need to do is call to functions you need to call model dot compile with some parameters.
We'll talk more about this next week, so don't worry about it for now.
And then you need to call model dot fit X Y, which tells tensor flow to take this neural network that are created by sequentially string together layers one and two, and to train it on the data, X and Y.
ow do you do forward prop if you have a new example, say X new,

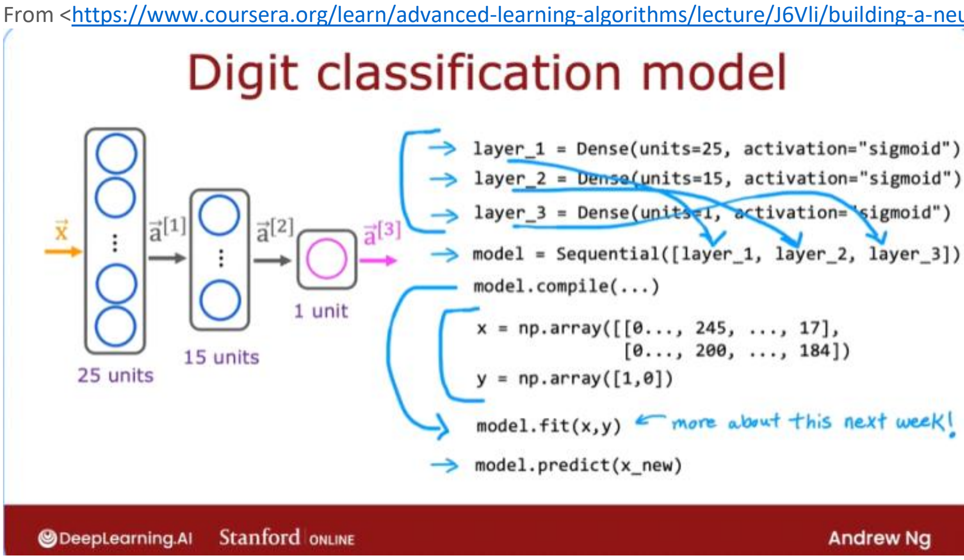
which is NP array with these two features than to carry out forward prop instead of having to do it one layer at a time yourself, you just have to call model predict on X new and this will output the corresponding value of a two for you given this input value of X.
So model predicts carries out forward propagation and carries an inference for you, using this neural network that you compiled using the sequential function.
Now I want to take these three lines of code on top and just simplify it a little bit further, which is when coding in Tensorflow.



From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/J6Vli/building-a-neural-network>



From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/J6Vli/building-a-neural-network>



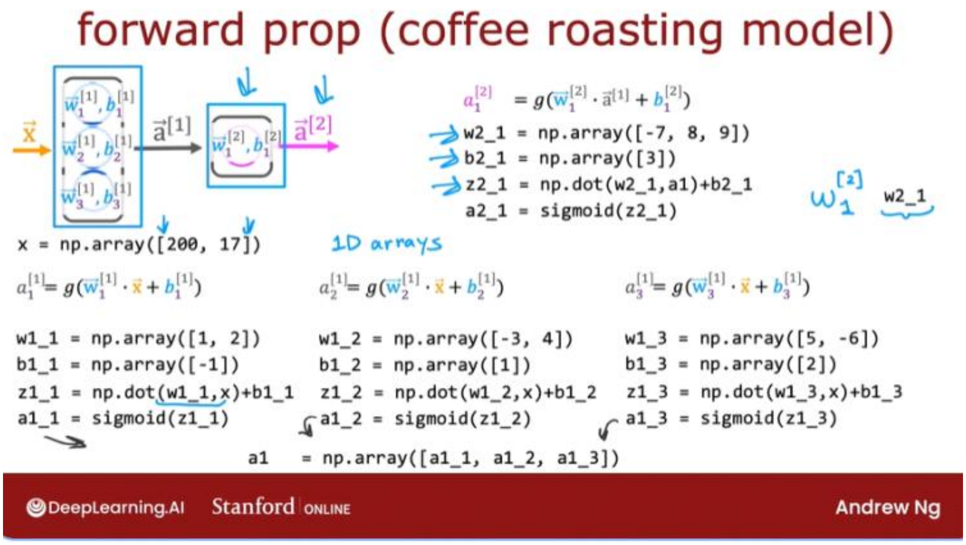
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/J6Vli/building-a-neural-network>

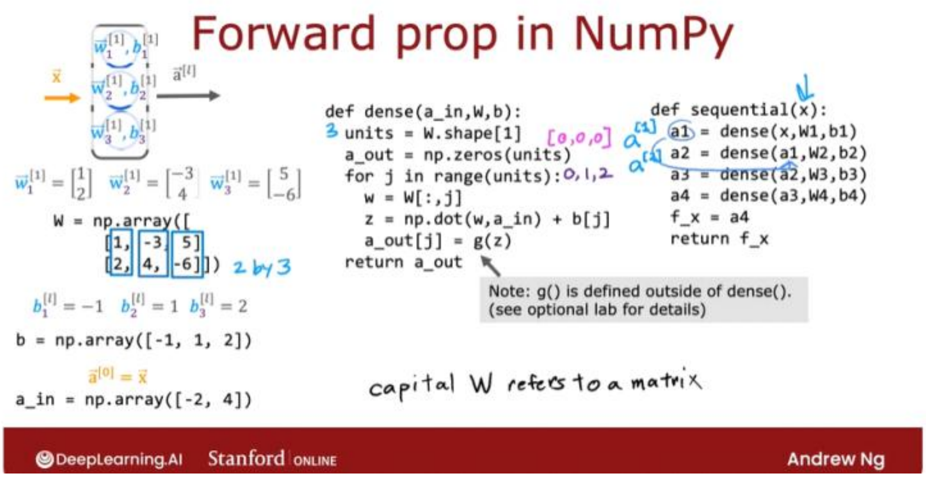
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/J6Vli/building-a-neural-network>

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/J6Vli/building-a-neural-network>

that's how you implement forward prop using just python and np. Now, there are a lot of expressions in this page of code that you just saw, let's in the next video look at how you can simplify this to implement forward prop for a more general neural network, rather than hard coding it for every single neuron like we just did.

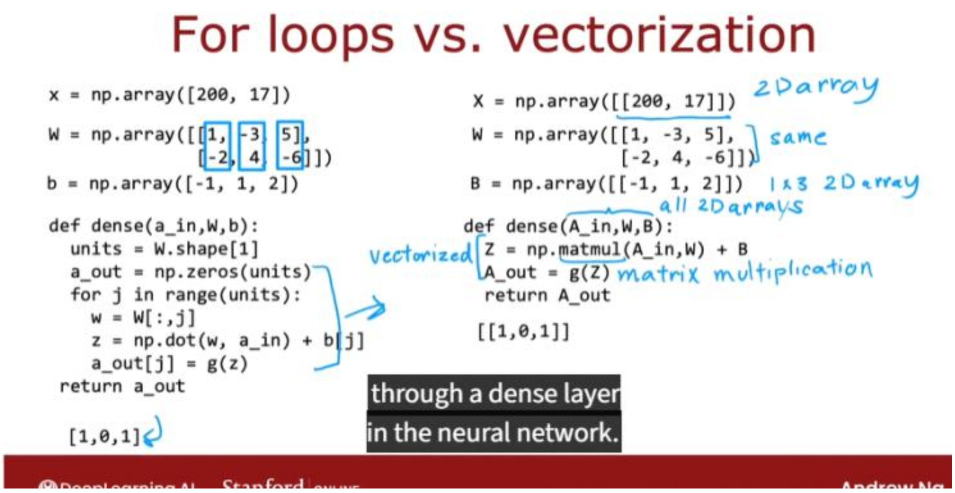
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/AJc5g/forward-prop-in-a-single-layer>





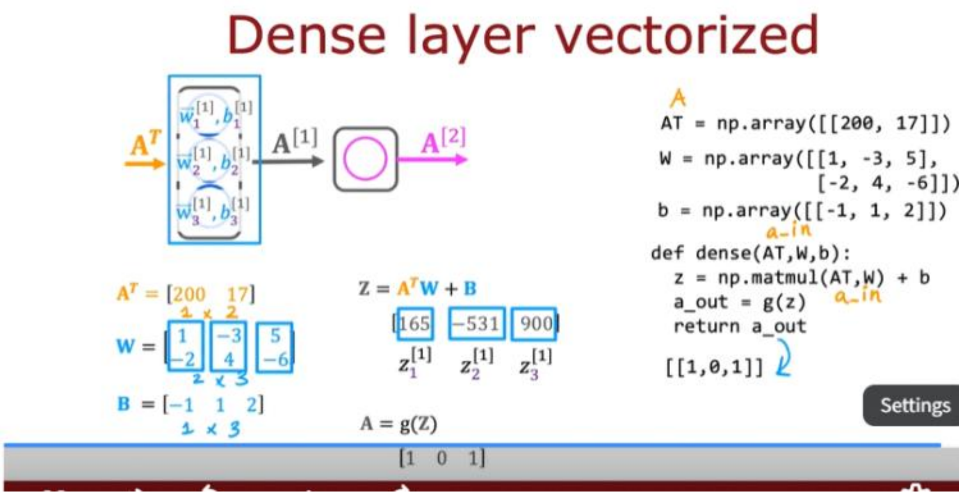
This is what the code looks like. Notice that in the vectorized implementation, all of these quantities, x, which is fed into the value of A in as well as W, B, as well as Z and A out, all of these are now 2D arrays. All of these are matrices. This turns out to be a very efficient implementation of one step of forward propagation through a dense layer in the neural network. This is code for a vectorized implementation of forward prop in a neural network.

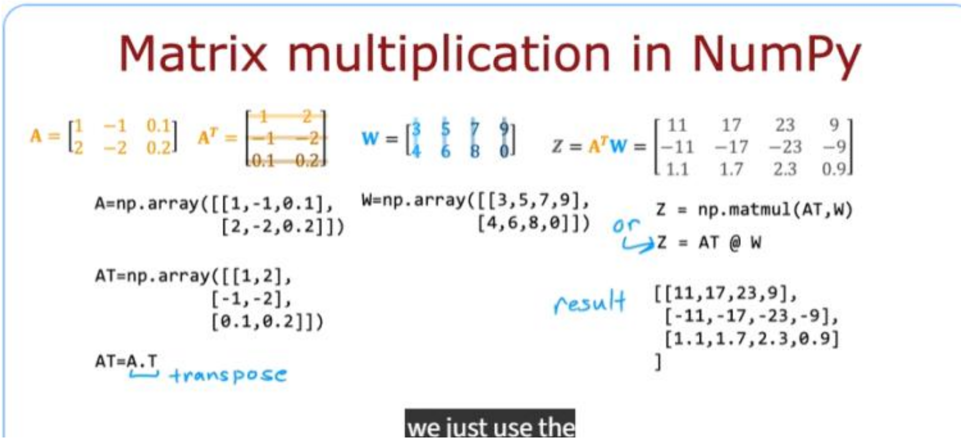
From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/qkJy8/how-neural-networks-are-implemented-efficiently>



'm going to set A transpose to be equal to the input feature values 217. These are just the usual input feature values, 200 degrees roasting coffee for 17 minutes. This is a one by two matrix. I'm going to take the parameters w_1, w_2, and w_3, and stack them in columns like this to form this matrix W. The values b_1, b_2, b_3, I'm going to put it into a one by three matrix, that is this matrix B as follows. Then it turns out that if you were to compute Z equals A transpose W plus B, that will result in these three numbers and that's computed by taking the input feature values and multiplying that by the first column and then adding B to get 165. Taking these feature values, dot-producting with the second column, that is a weight w_2 and adding b_2 to get negative 531.
These feature values dot product with the weights w_3 plus b_3 to get 900. Feel free to pause the video if you wish to double-check these calculations. But this gives you is the values of z^1_1, Z^1_2, and Z^1_3. Then finally, if the function g applies the sigmoid function to these three numbers element-wise, that is, applies the sigmoid function to 165, to negative 531, and to 900, then you end up with A equals g of this matrix Z ends up being 1,0,1. It's 1,0,1 because sigmoid of 165 is so close to one that up to numerical round off is based to one and these are bases 0 and 1. Let's look at how you implement this in code. A transpose is equal to this, is this one by two array of 217.
In case you're comparing this slide with the slide a few videos back, there was just one little difference, which was by convention, the way this is implemented in TensorFlow, rather than calling this variable A,T, we were calling it A_in, which is why this too is the correct implementation of the code. There is a convention in TensorFlow that individual examples are actually laid out in rows in the matrix X rather than in the matrix X transpose which is why the code implementation actually looks like this in TensorFlow.

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/ysRAb/matrix-multiplication-code>





From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/ysRAb/matrix-multiplication-code>

From <https://www.coursera.org/learn/advanced-learning-algorithms/lecture/ysRAb/matrix-multiplication-code>