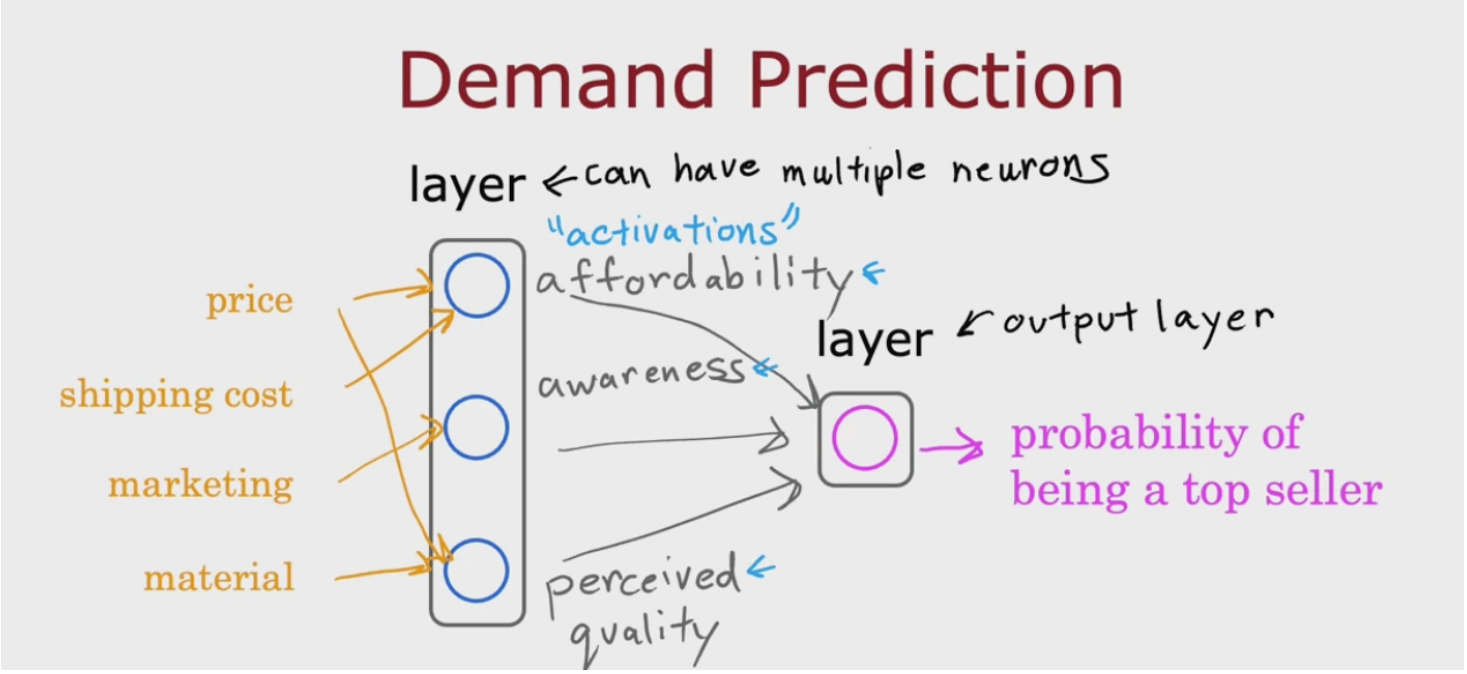
Advanced Learning Algorithms - Week 1&2

Neural networks: Mimic brain. Speech recognition - images - NLP

Neuron
some # of inputs, a number as output

Traditional AI, not capable of handling this much amount of data - Everything is digital now.
so neural networks.



"activation values"
"input layer"
"neural vector" : can consider all the inputs as one vector, and all the inputs for the output layer as another vector.

it just looks like logistic regression, but it does not use ready features - price, shipping cost... - but it creates other features, maybe better ones.
feature engineering

How many hidden layers? How many neurons in which?

Face recognition - 1000x1000 pixels - can there be a model of neural network that takes a vector with 1m parameters.

what each neuron outputs:
a[j] = g(w.a[j-1] + b)

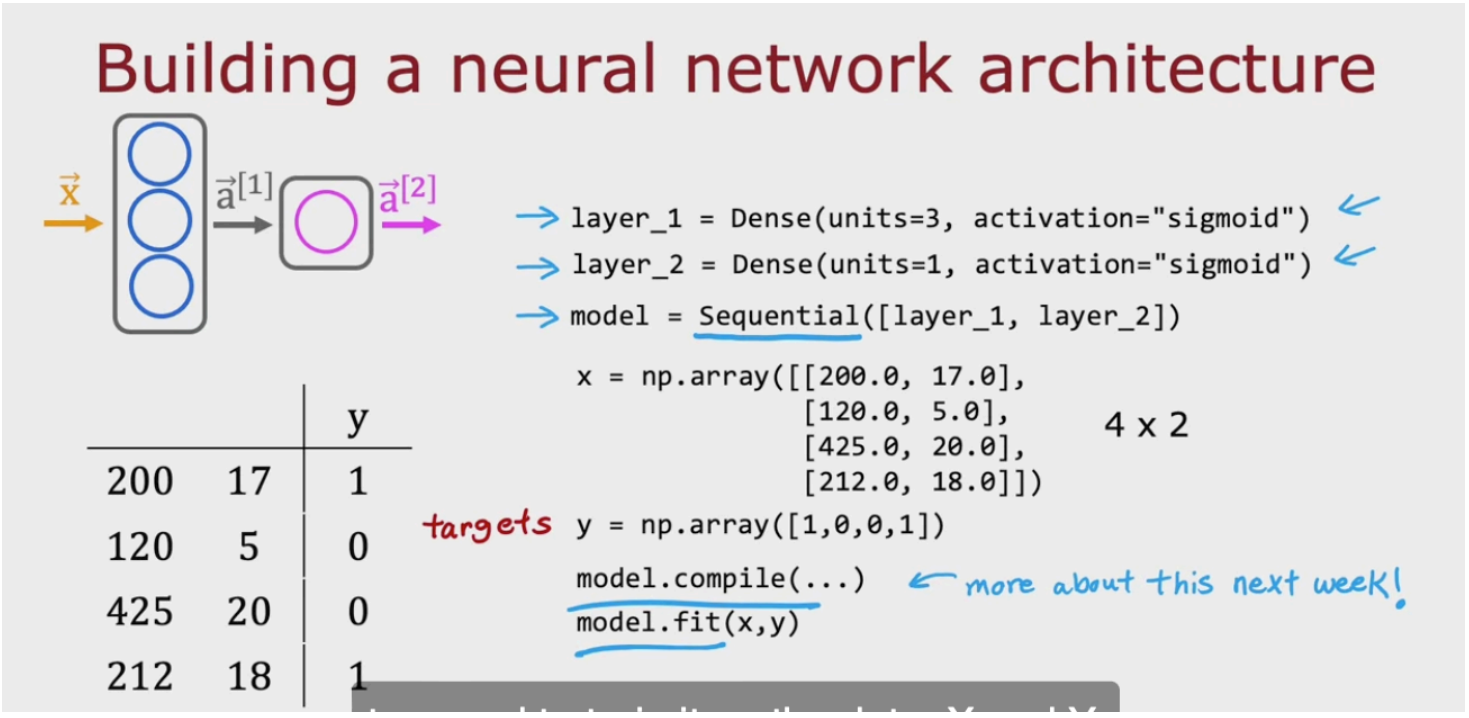
Handwritten digit recognition:

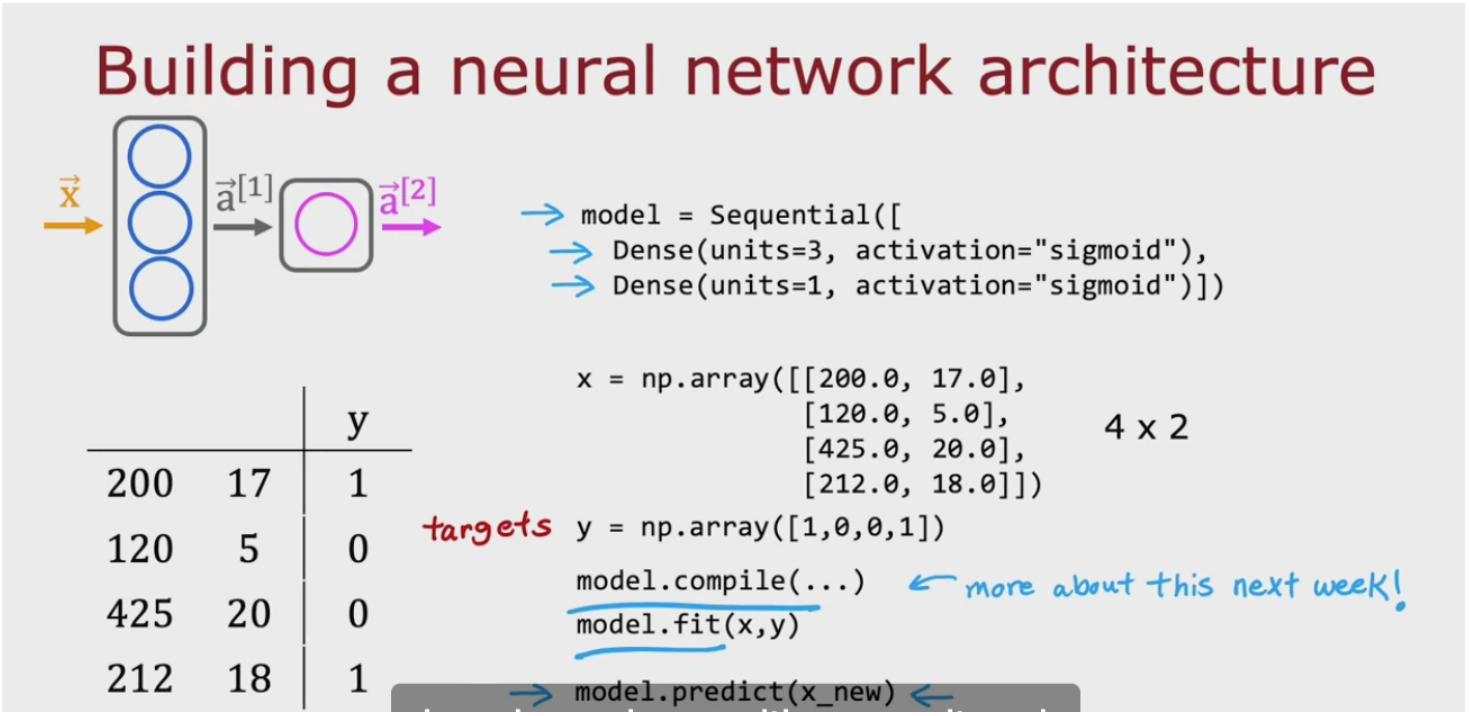
forward computation: run something over neural network, running it.

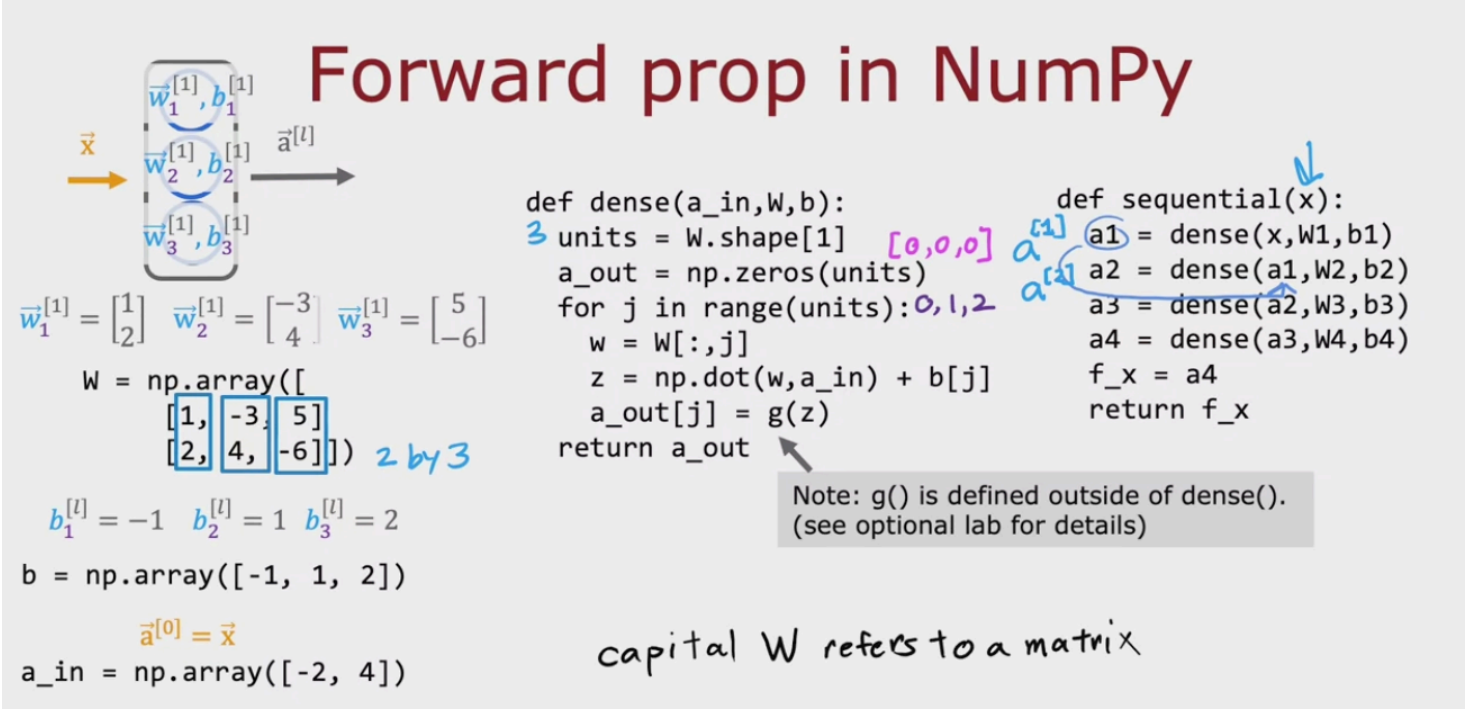
How TensorFlow represents data?:

numpy array 2x3 (2 rows, 3 columns):
x = np.array([[1,2,3] ,
[4,5,6]])
tf.Tensor([[ ]]) -> we can say this is an matrix. (More general than matrix)

How to deploy on tf:



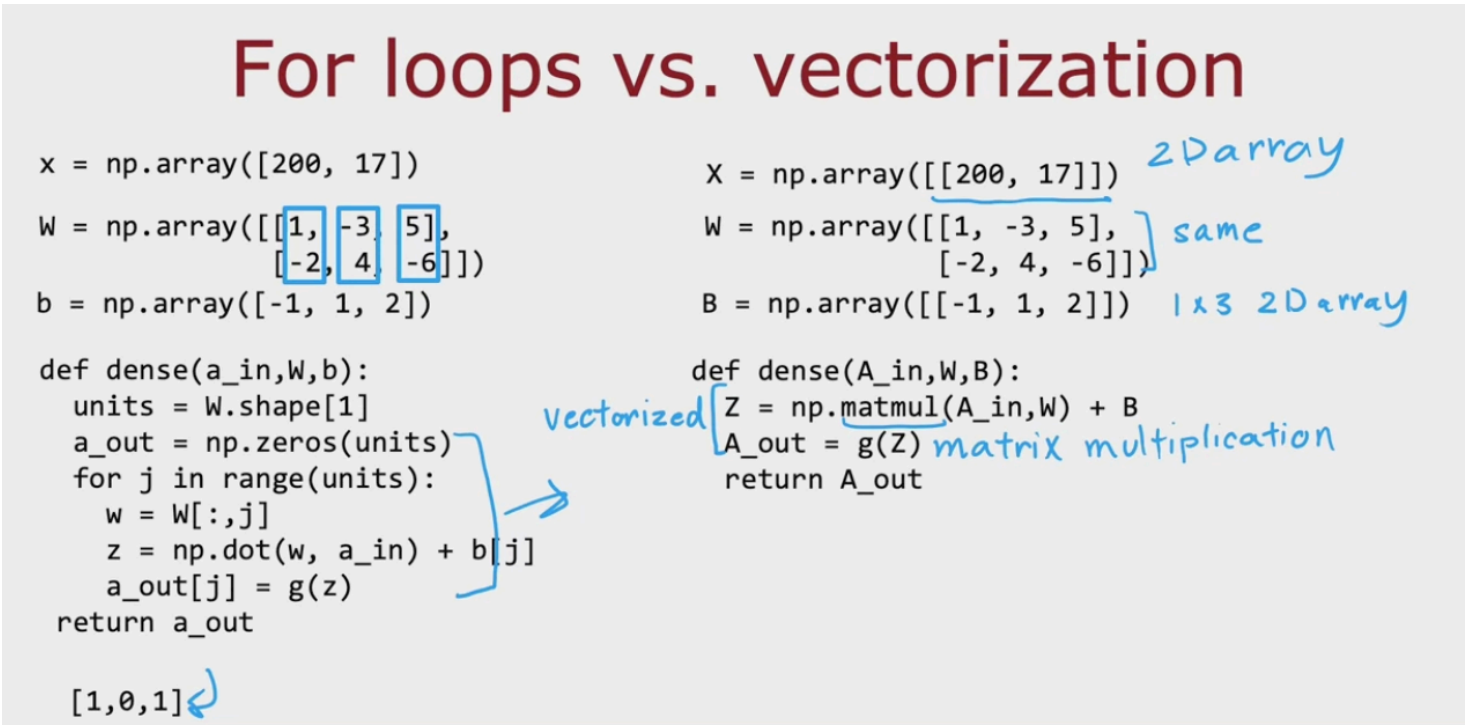




units : w.shape[1] = # of columns

a_out : [0,0,0]

Vectorization:

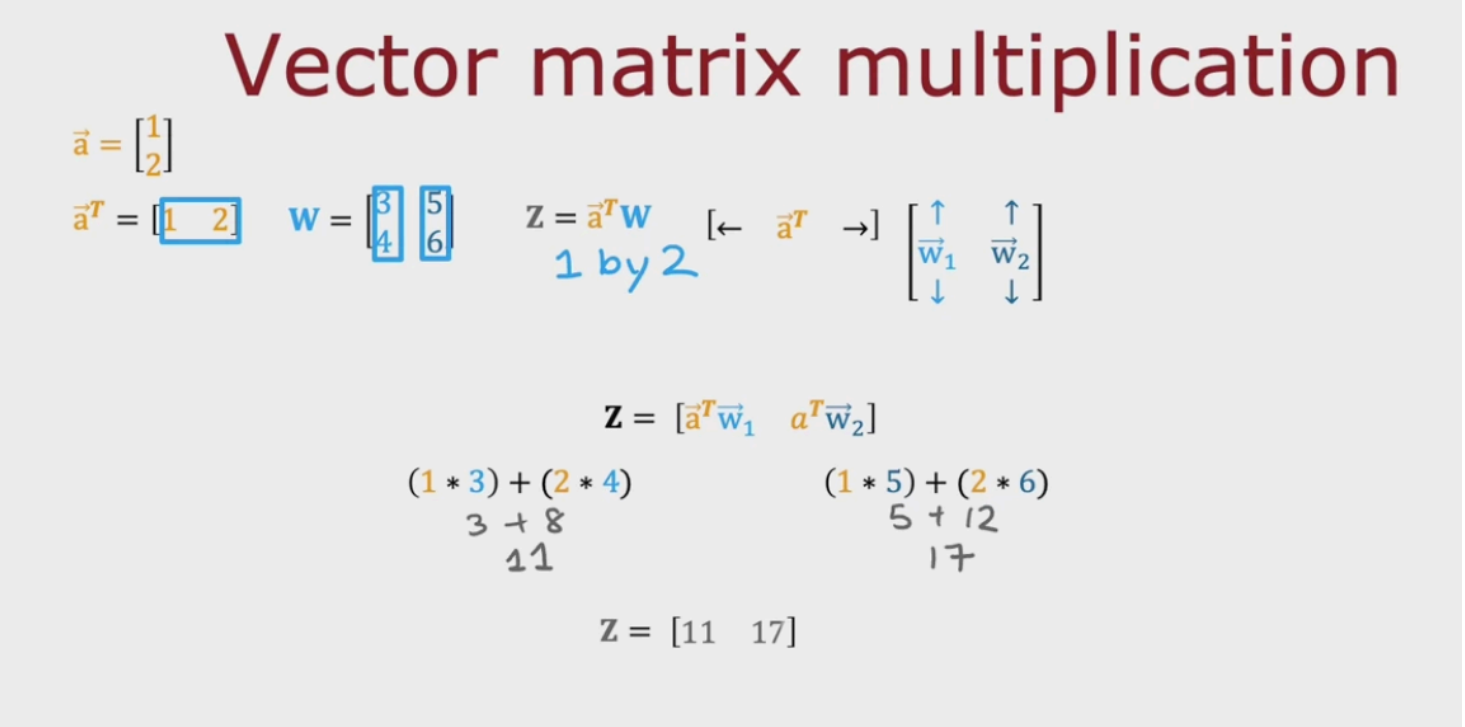


Dot product:

1x2 . 2x1 = 1x1

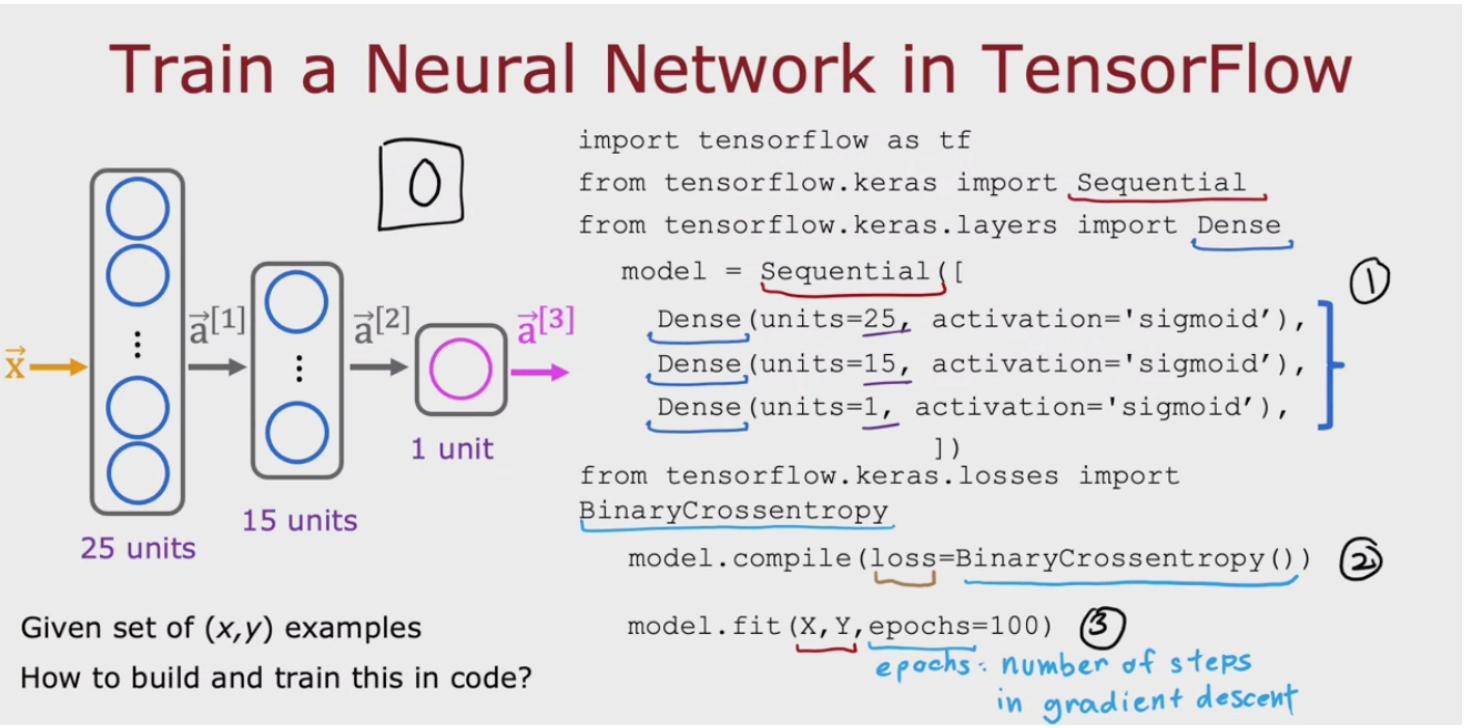
Transpose: 2x1 -> 1x2

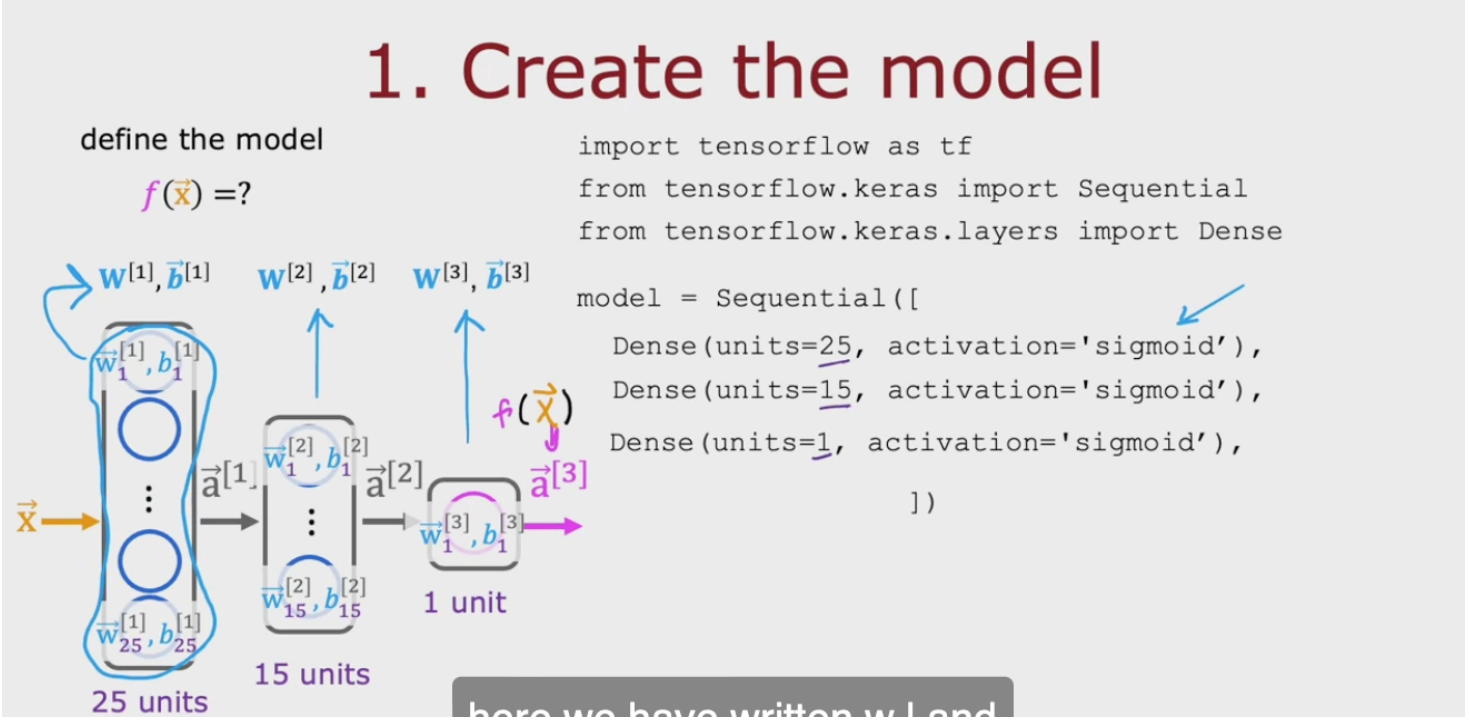
Matrix Multiplication:



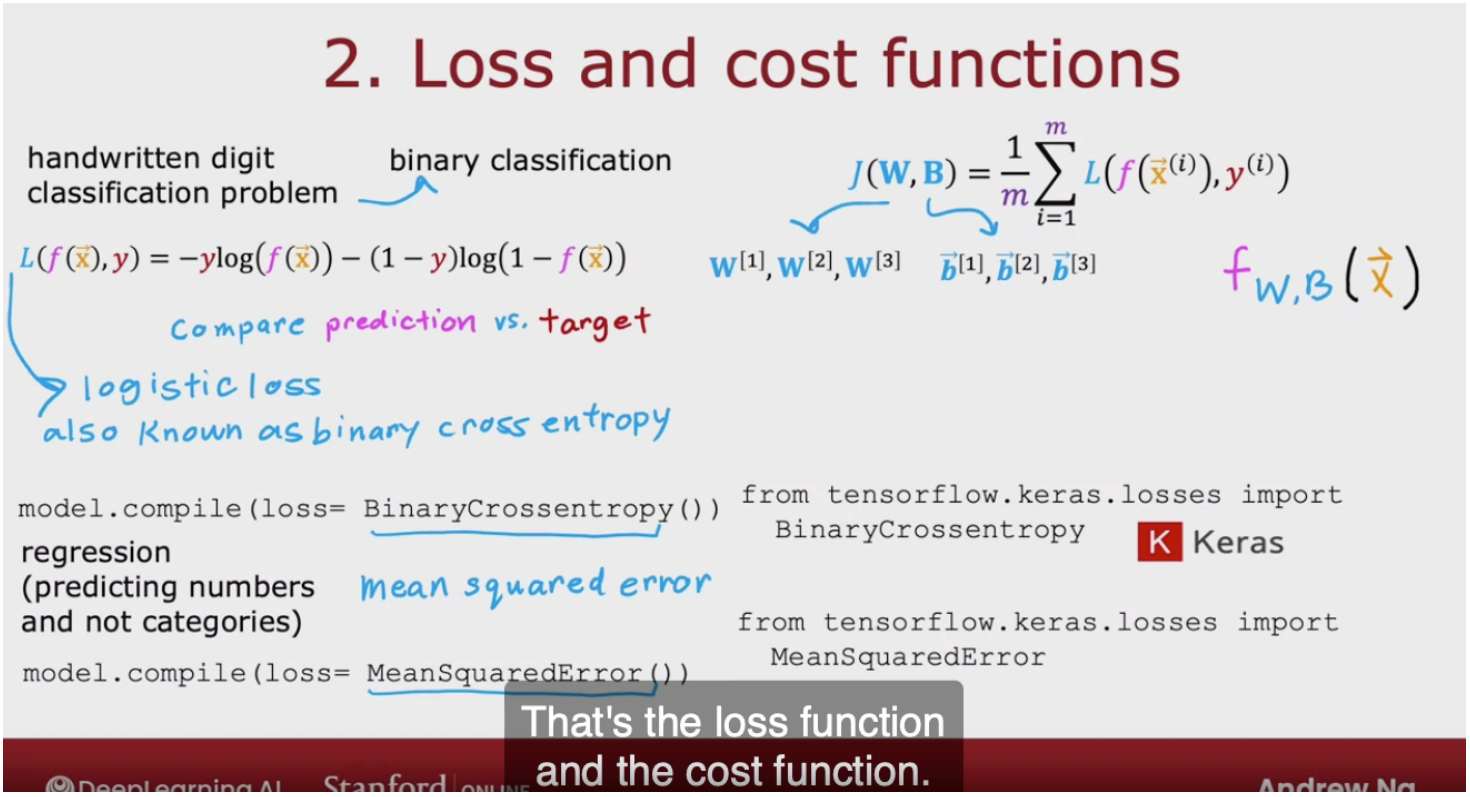
BinaryCrossEntropy() loss function

How to train a model on tf:





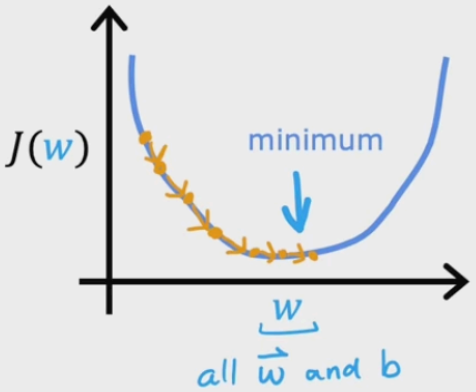
when we initialize the model, we are not giving exact values to w and b, but we are also initialize them so that we can train the model and the model will decide itself on those values.



Loss func: how distant is the model's prediction from the actual values.
Cost function = average error over all the training examples.

• **Binary Cross-Entropy** (or **Logistic Loss**) is used in binary classification problems (i.e., problems where there are only two possible outcomes like 0 or 1, yes or no). It works well for classification problems because it compares the probability that the model assigns to the correct class.
• **Mean Squared Error (MSE)** is commonly used for **regression** tasks, where the goal is to predict a continuous number (e.g., predicting house prices). It calculates the squared difference between the predicted value and the actual value.

3. Gradient descent



repeat {

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

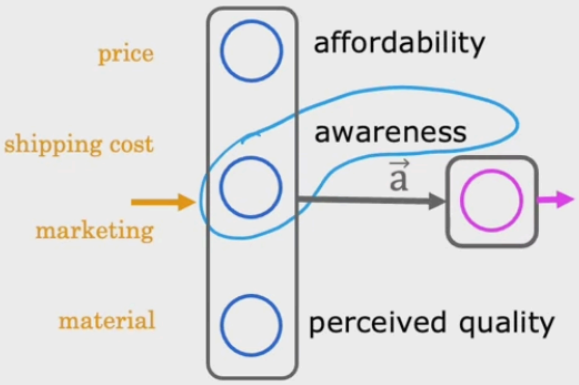
$$b_j^{[l]} = b_j^{[l]} - \alpha \frac{\partial}{\partial bj} J(\vec{w}, b)$$

} Compute derivatives for gradient descent using "back propagation"

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

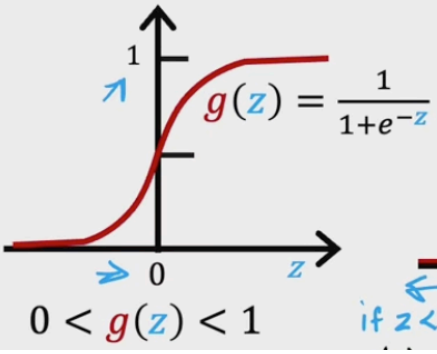
Activation Functions:

Demand Prediction Example



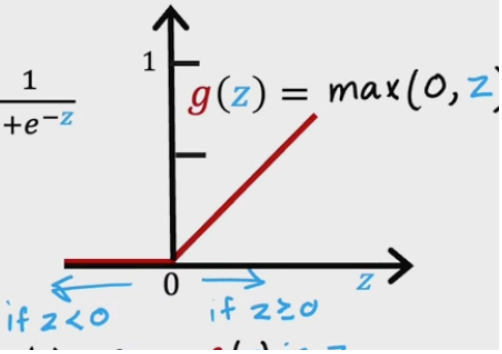
$$a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}) \quad (z)$$

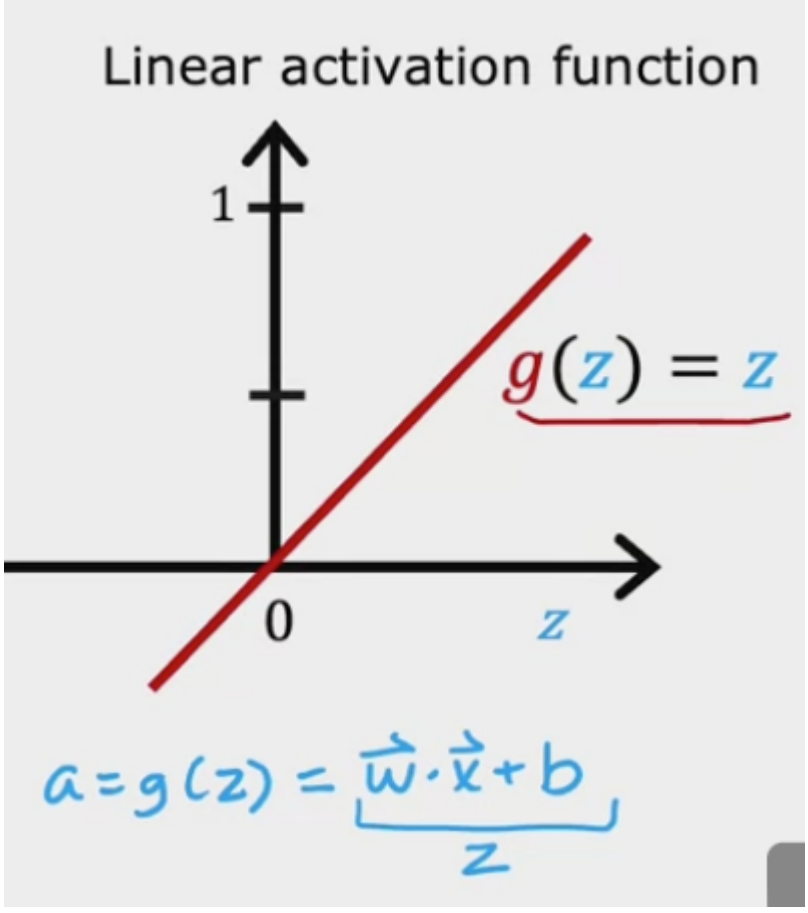
Sigmoid



$0 < g(z) < 1$

ReLU Rectified Linear Unit



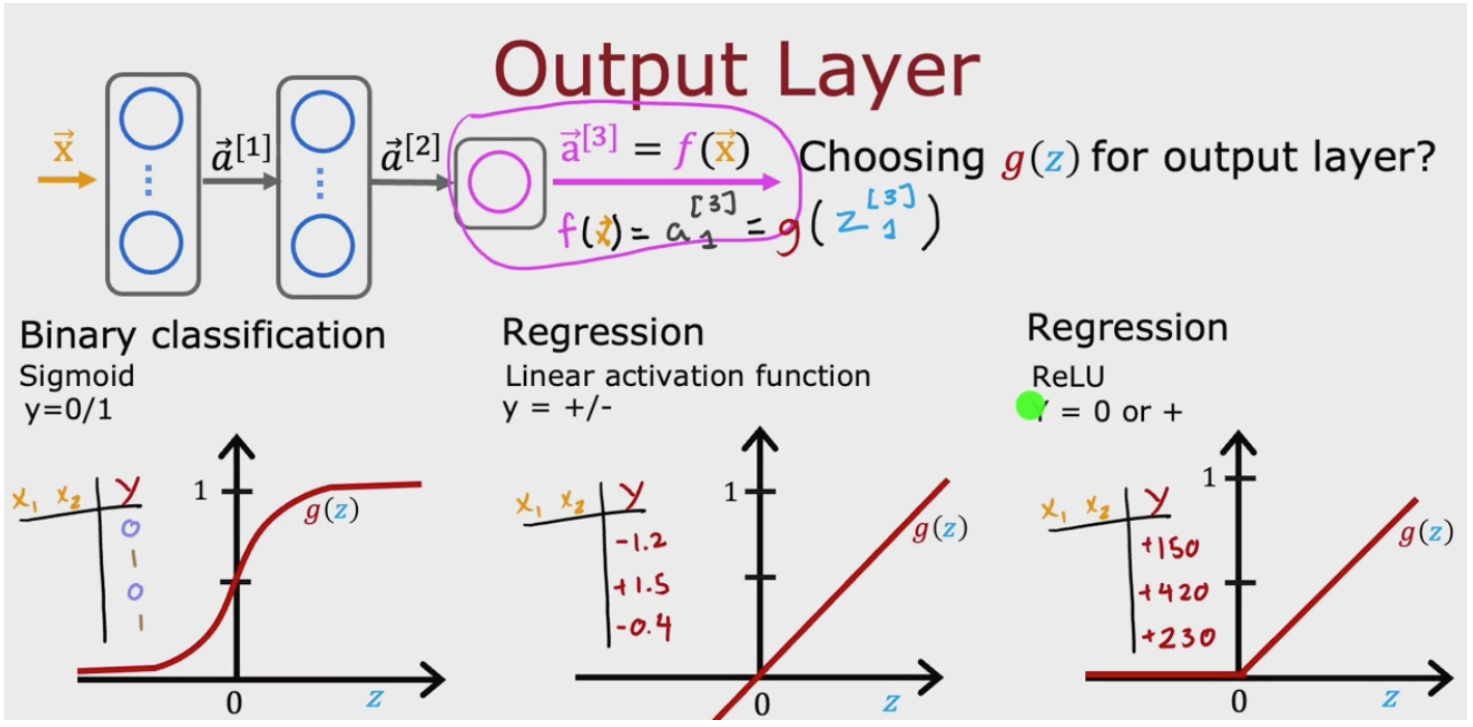


(-same as- not using any activation function)

Choosing the activation function:

Output Layer

Binary Classification: Sigmoid - nearly always a natural choice
Regression: Linear activation function
Cases where only non negative values can be outputs: ReLU



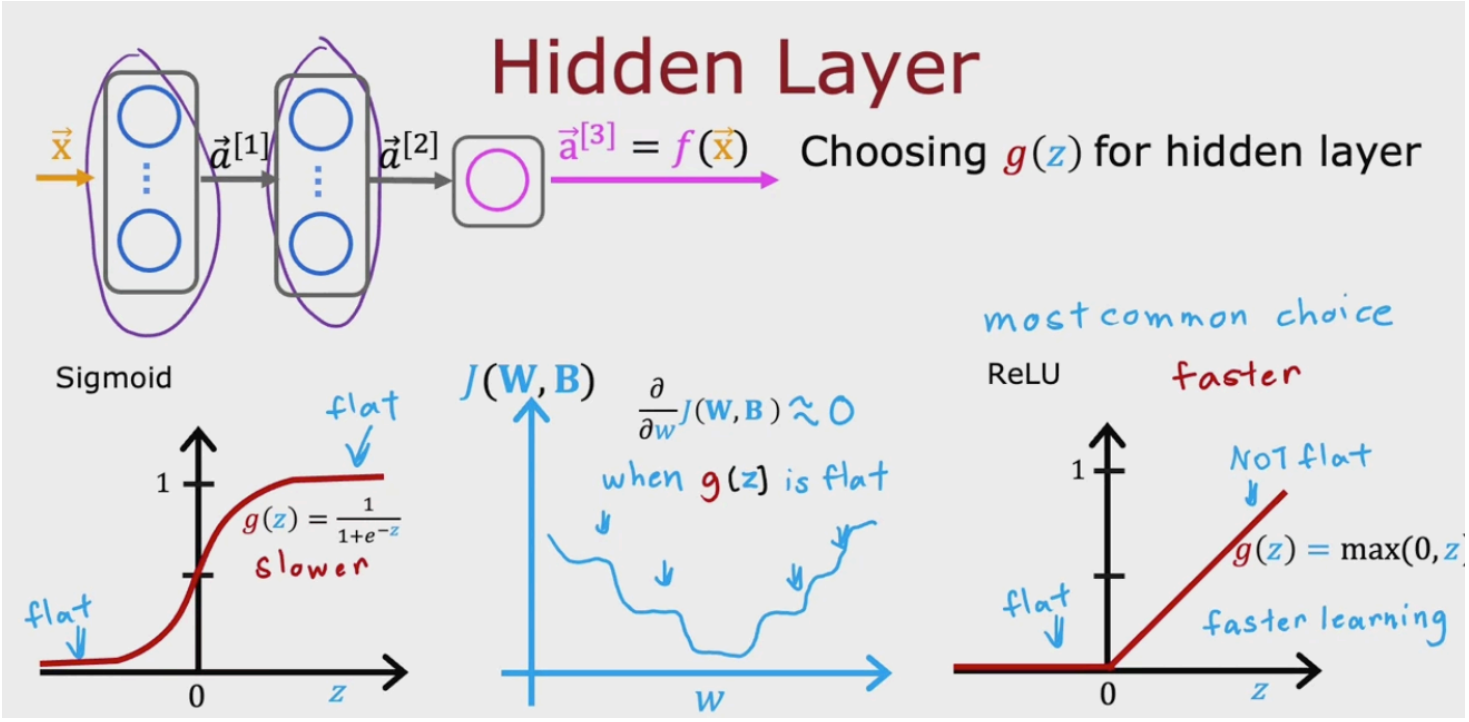
Hidden Layers

ReLU most common for hidden layers.
Early history: sigmoid. but now hardly ever.

Why?

A bit faster : g(z) = max(0, z)

at sigmoid, flat on two sides. but in ReLU, flat in only one side. Effects on Gradient Descent. So neural networks may learn faster in ReLU.

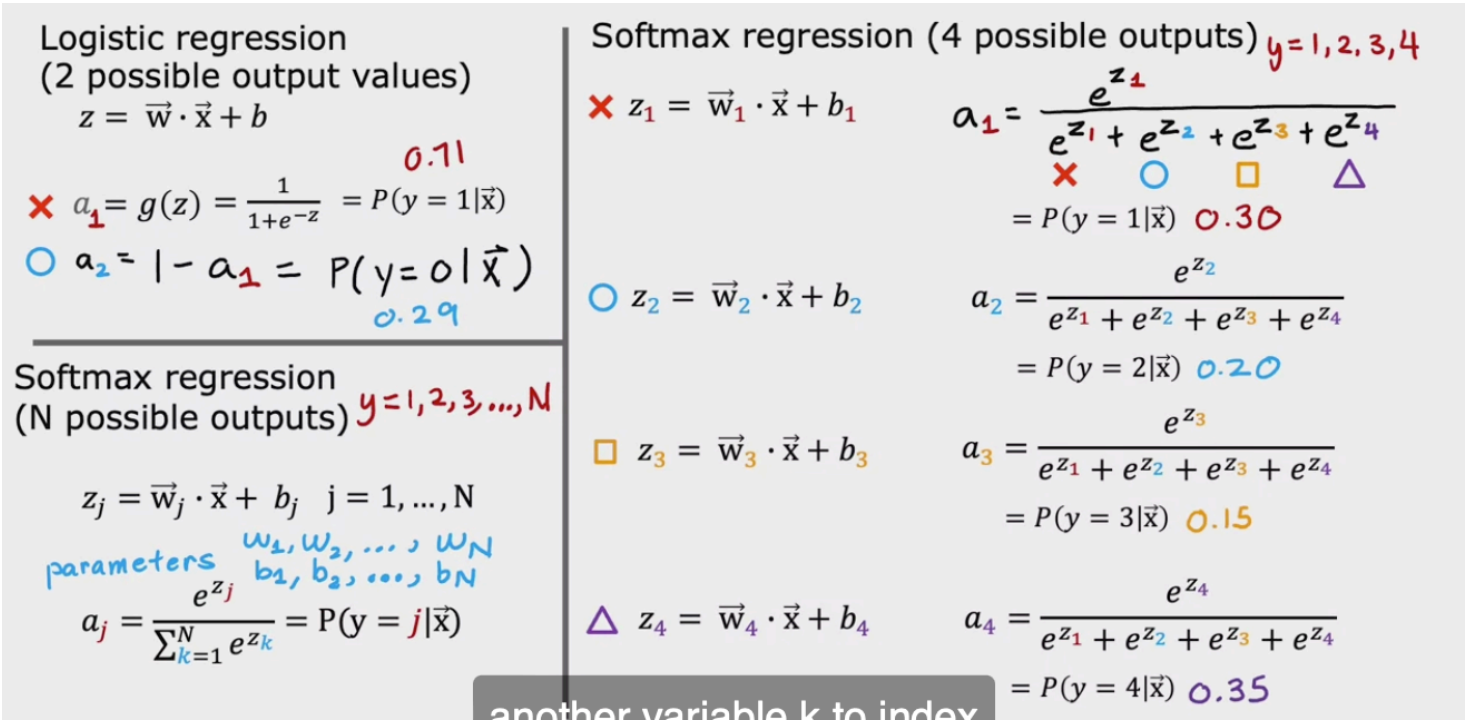


there are other activation functions where some cases they work better, note.

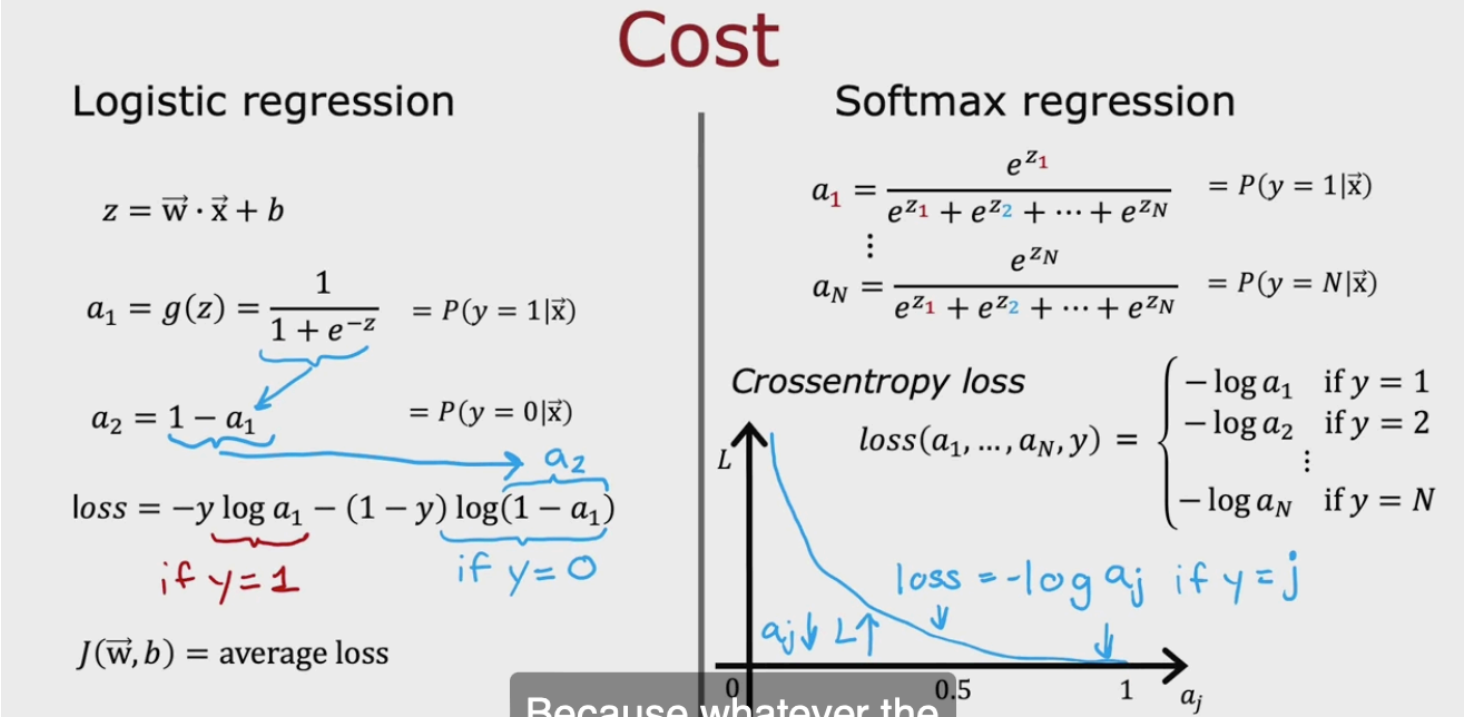
Multiclass Classification

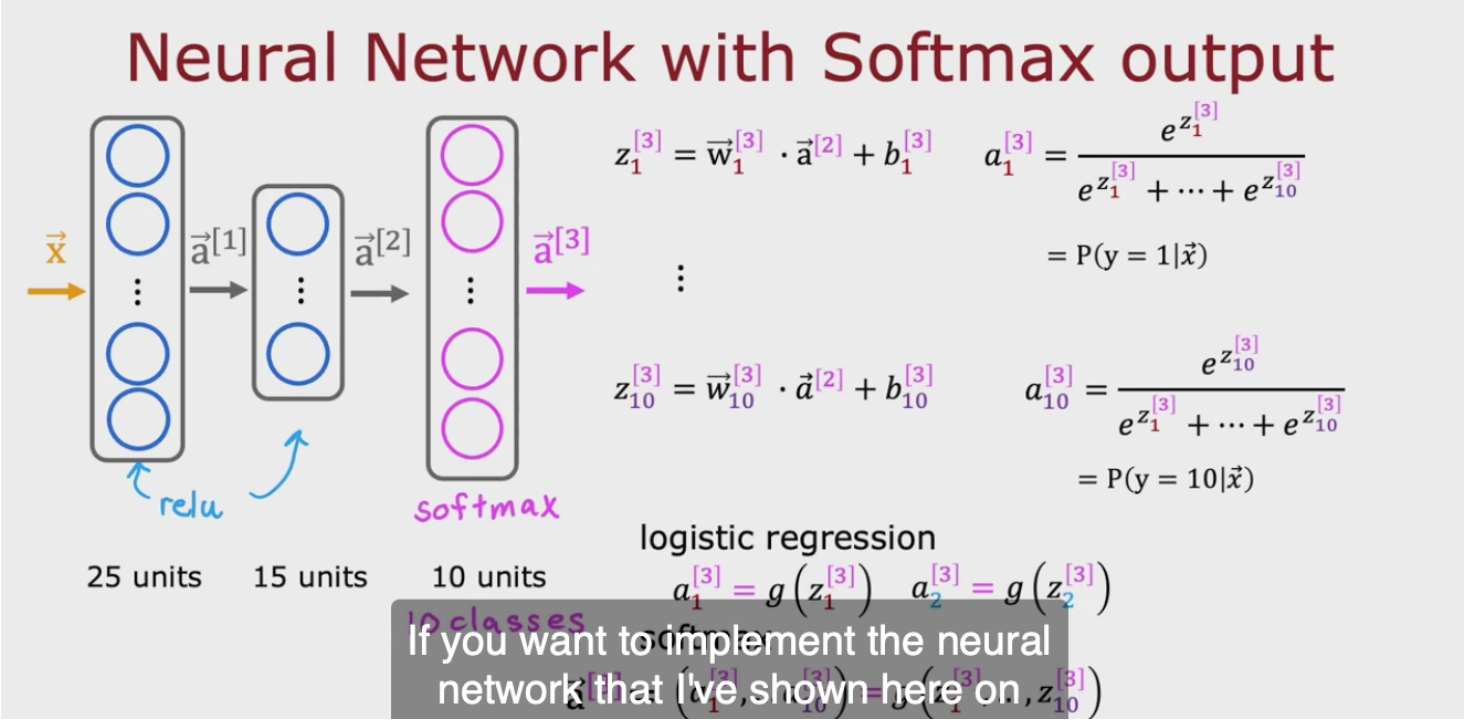
There is no 2 options - good coffee, bad coffee - there are not 2 digits to draw - 0 and 1 - but all numbers.

Softmax:



if n=2 in softmax -> then this becomes logistic regression, so softmax is a generalization for logistic regression.





How is the neural network with multiple outs? Ex:10 outs, 10 units in the final layer - output layer. now with softmax activation function.

roundoff errors, minimizing this error
x = 2/10000
(after making some substraction and addition, it becomes something different)
{ x = .18f }

Numerical Roundoff Errors

More numerically accurate implementation of logistic loss:

Logistic regression:

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

Original loss

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

More accurate loss (in code)

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

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

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

logit: z

to have a little bit less

this is worse in softmax

More numerically accurate implementation of softmax

Softmax regression

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

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

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

'linear'

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

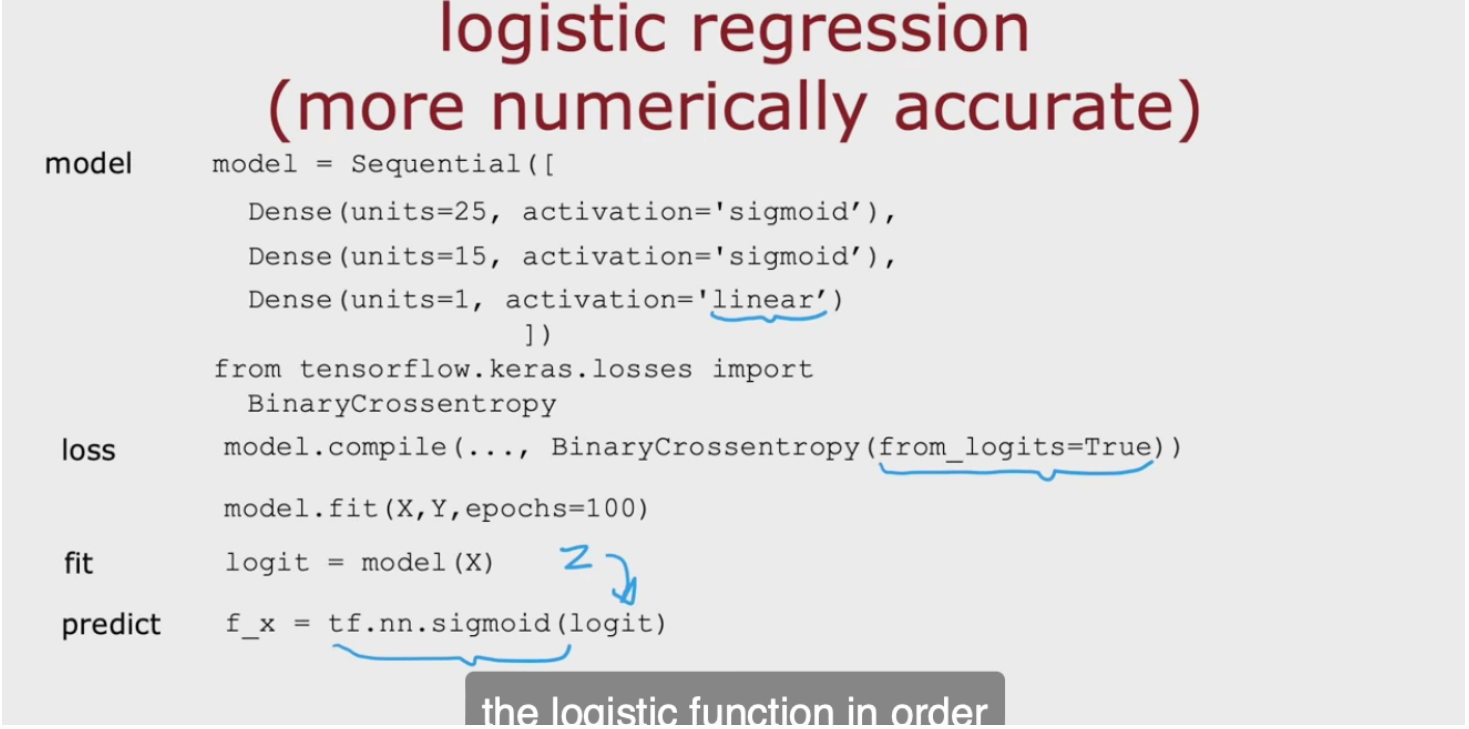
More Accurate

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

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

except that it is a little bit

but here, you do not actually get the probability. to do so, use the following:



1. **Softmax Activation Function**: When using the softmax activation function directly in the final layer of a neural network (especially for classification problems), you convert the raw logits (pre-activation values) into probabilities. This works fine most of the time, but softmax involves calculating exponentials of the logits, which can lead to numerical issues (like overflow or underflow) when the logits are very large or very small. This is particularly problematic for high-dimensional input data or deep networks where extreme values in logits can destabilize training.
2. **Alternative with Linear Activation**: Instead of using the softmax activation directly, you can use a **linear activation** (i.e., no activation function) for the final layer, which outputs the raw logits. Then, during the loss calculation, you apply softmax **inside the loss function** (such as SparseCategoricalCrossEntropy with from_logits=True). By doing this, TensorFlow (or the framework you're using) handles the softmax computation internally in a more numerically stable way, often leveraging techniques like log-sum-exp to prevent overflow and underflow.

Why Linear Activation Helps:

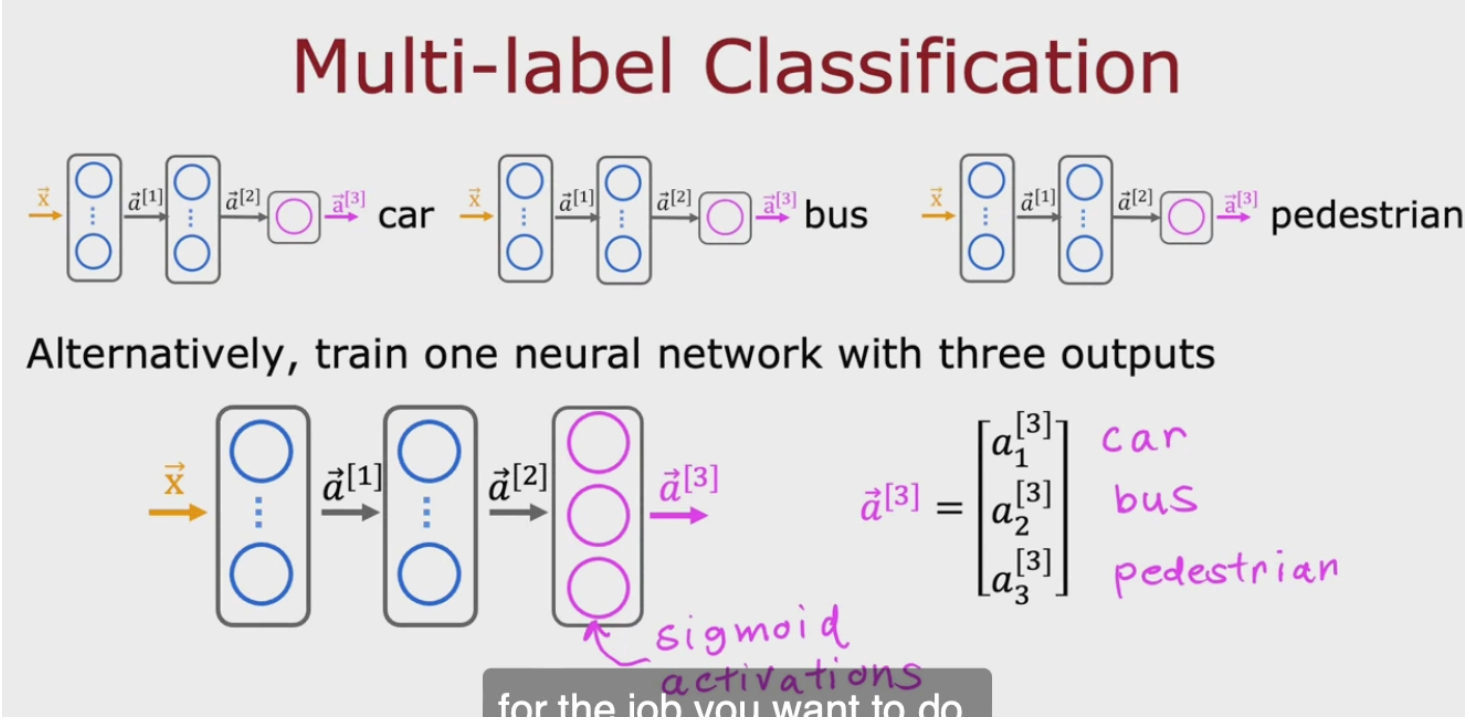
• **From Logits=True**: When you use a linear activation followed by a cross-entropy loss with from_logits=True, the softmax and the logarithm are combined into one numerically stable operation. This prevents large exponents from being computed explicitly, which can avoid rounding errors.

• **Log-sum-exp Trick**: The underlying loss function uses the "log-sum-exp" trick, which is mathematically equivalent to softmax but avoids extreme values that cause numerical problems.

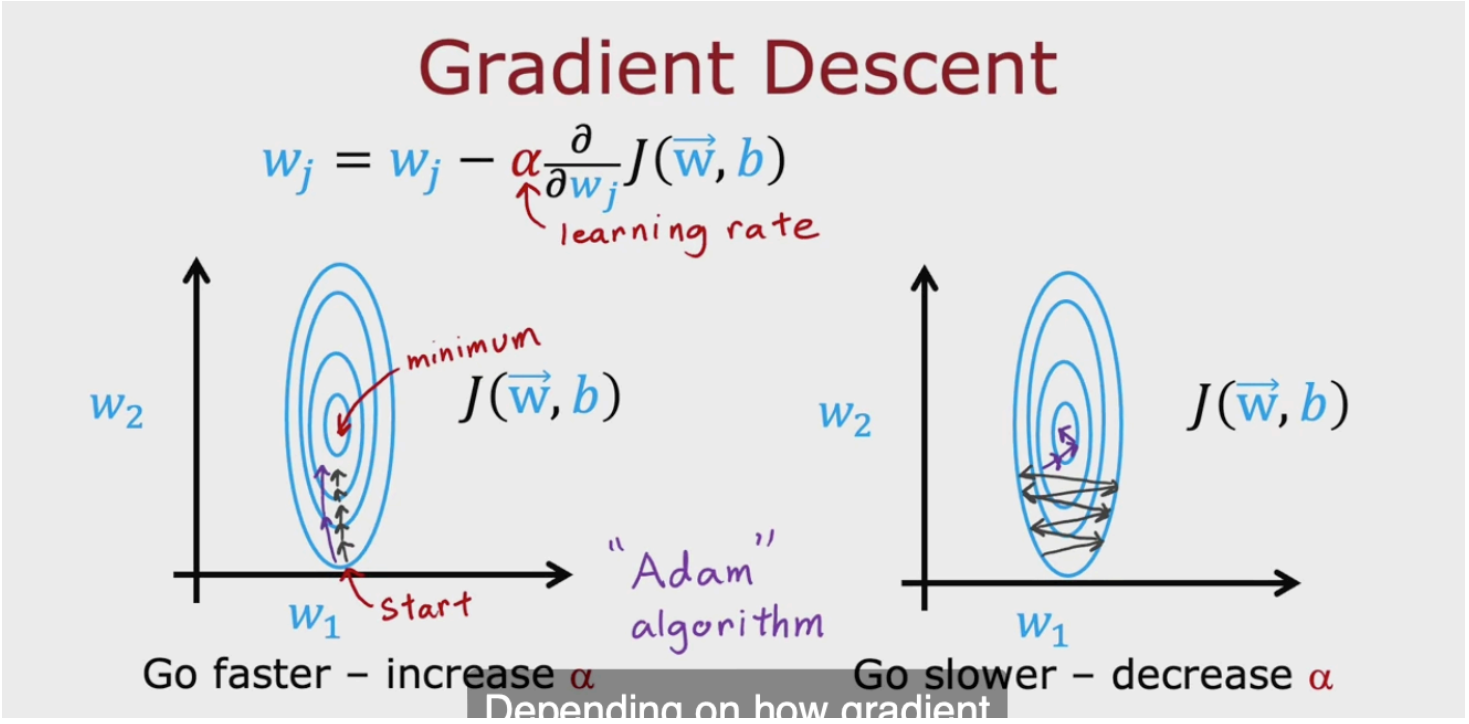
Multi-Label Classification Problem

Is there a car in the image? A person? Ambulance? - More than one questions neural network should answer.

Can use more than one neural network. (One for each)



Training your neural network much faster than gradient descent



"Adam" algorithm decides the alpha step for the gradient descent.

Uses different learning rates for each of the parameters. (lets say 10 units, 11 learning rate alpha's - *one for the b constant*)

`MNIST Adam`

```
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 the model

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

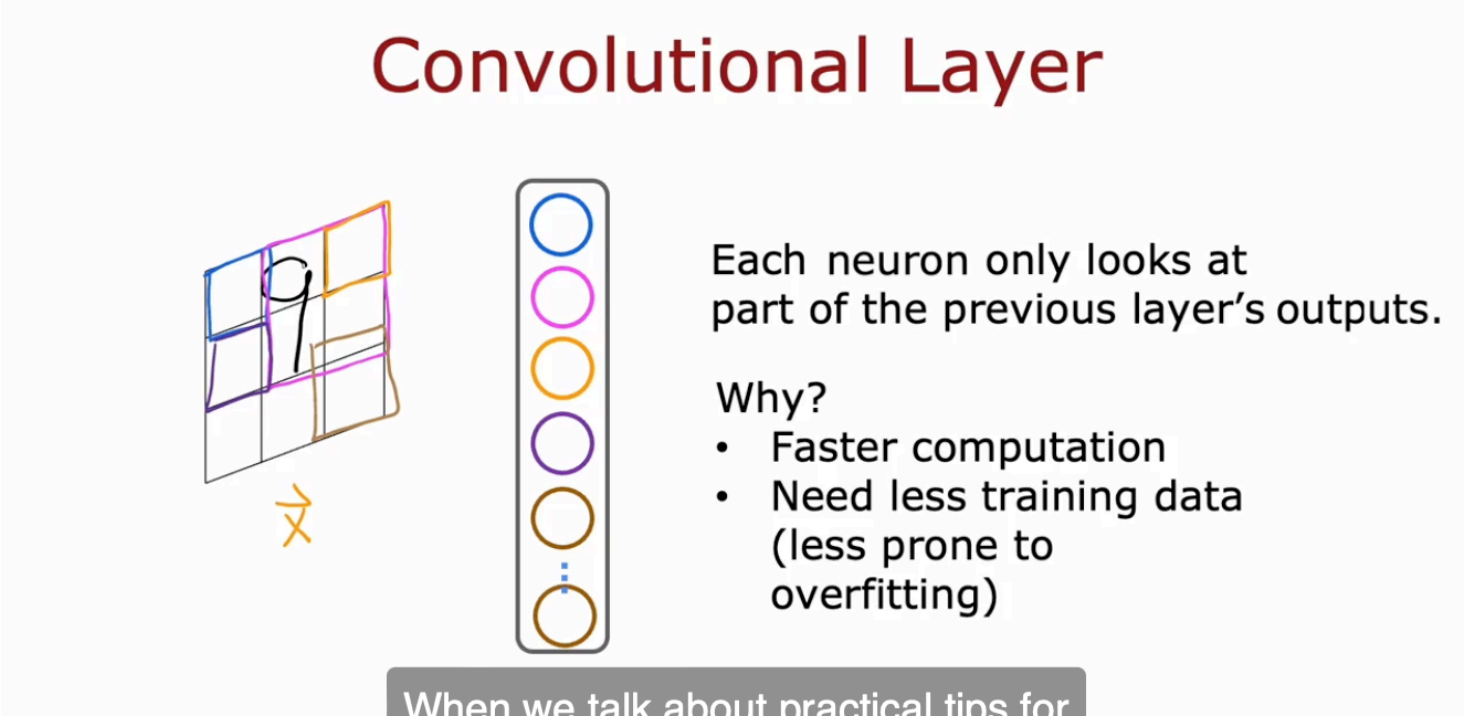
Fit the model

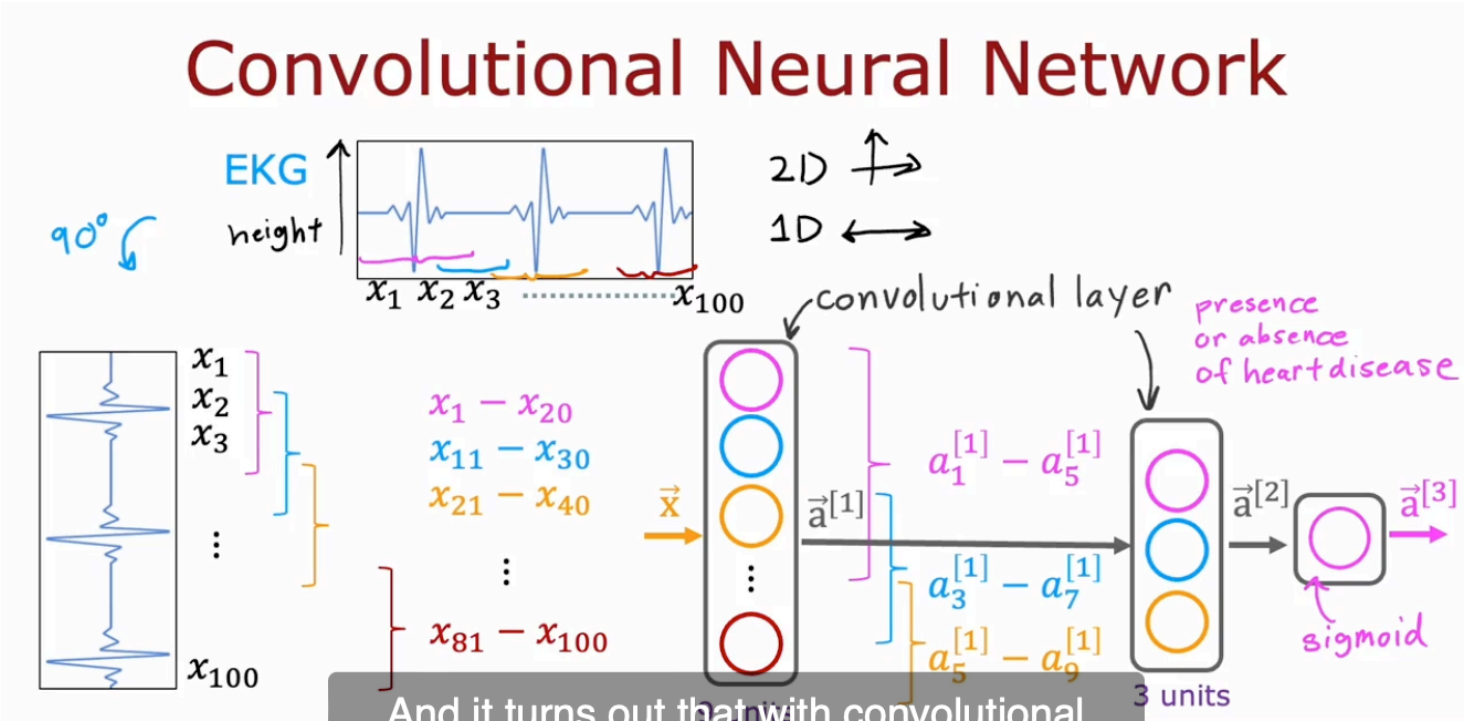
```
model.fit(X, Y, epochs=100)
```

become a standart instead of gradient descent - much faster.

Additional Layer Types

Dense Layer: Each neuron output is a function of all the activation outputs of the previous layer.
Convolutional Layer:





- May look into here more later.