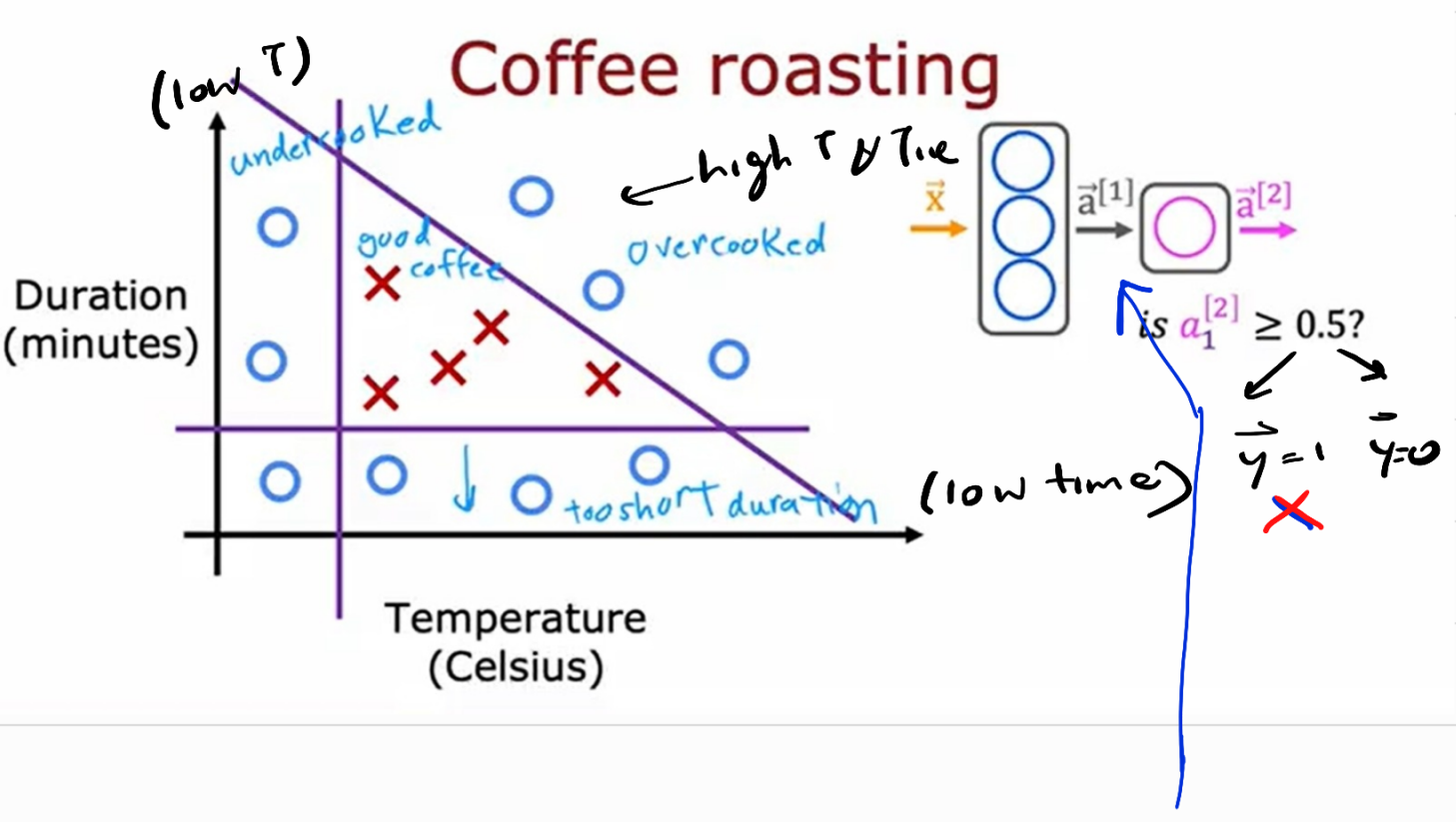
Tensorflow;

eg to understand cofee roasting




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

```
layer_1 = Dense(units = 3, activation = 'sigmoid')
```

```
a1 = layer_1(x) ⟶ tensor(list of 3 numbers
```

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

≥ 0.5 → $\hat{y} = 1$

< 0.5 → $\hat{y} = 0$

Digit classification;

hexadecimal values

```
x = np.array([[0.0, ..... .245, ...]])
layer_1 = Dense(units=25, activ='sigm')
a1 = layer_1(x)
layer_2 = Dense(units=2, act. 'sigm')
a2 = layer_2(a1)
layer_3 = Dense(u..=1, ....)
a3 = layer_3(a2)
```

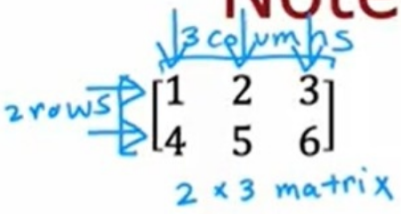
≥ 0.5

Data in Tensorflow;

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

? [[

Note about numpy arrays




```
x = np.array([[1, 2, 3],
              [4, 5, 6]])

[[1, 2, 3],
 [4, 5, 6]]
```

matrix ← 2D array

$$\begin{bmatrix} 0.1 & 0.2 \\ -3 & -4 \\ -.5 & -.6 \\ 7 & 8 \end{bmatrix}$$

4 x 2 matrix

```
x = np.array([[0.1, 0.2],
              [-3.0, -4.0,],
              [-0.5, -0.6,],
              [7.0, 8.0,]])

[[0.1, 0.2],
 [-3.0, -4.0,],
 [-0.5, -0.6,],
 [7.0, 8.0,]]
```

2D array

matrix

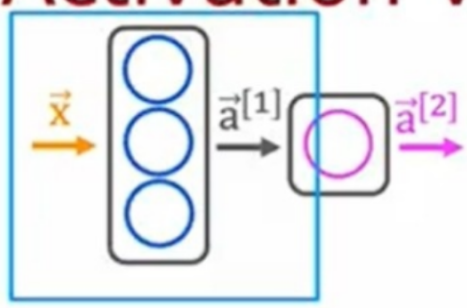
if written

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

1D array
linear vector

* tenserflow we use 2D (matrix) arrays

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

if print shows a1 this

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

```
a1.numpy()
```

← numpy way

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

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

```
tf.Tensor([[0.8]], shape=(1, 1), dtype=float32)
```

← 1x1 matrix

```
a2.numpy()
```

```
array([[0.8]], dtype=float32)
```

* w, b are automatically created in dense layer

```
w, b = layer.get_weights()
```

parameters ;

$\vec{x}$ eg:- 400

x x neurons

W_1 parameters + B parameters

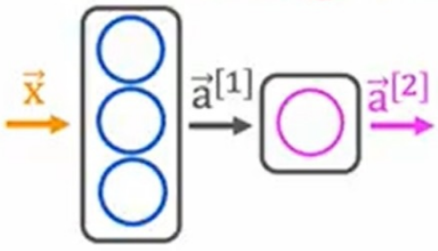
Building a neural network architecture;

before $a_2 = layer\_1(a_1)$ . . . .

Now;

```
model = Sequential(
    [layer_1, layer_2]
    )
```

Building a neural network architecture




```
layer_1 = Dense(units=3, activation="sigmoid")
layer_2 = Dense(units=1, activation="sigmoid")
model = Sequential([layer_1, layer_2])

x = np.array([[200.0, 17.0],
              [120.0, 5.0],
              [425.0, 20.0],
              [212.0, 18.0]])
y = np.array([1,0,0,1])
model.compile(...)
model.fit(x,y)
model.predict(x_new)
```

directly

4 x 2

targets

by convention don't explicitly write

more about this next week!

		y
200	17	1
120	5	0
425	20	0
212	18	1

Why normalize data in tensorflow

* norm = tf.keras.layers.Normalization(axis = -1)

norm.adapt(x)
xn = norm(x)
↳ because data was uneven

Now g.d works faster &
no dominating features

* np.tile(Xn, (1000, 1))
np.tile(Y, (1000, 1))

↳ Repeat data so the data set gets much bigger

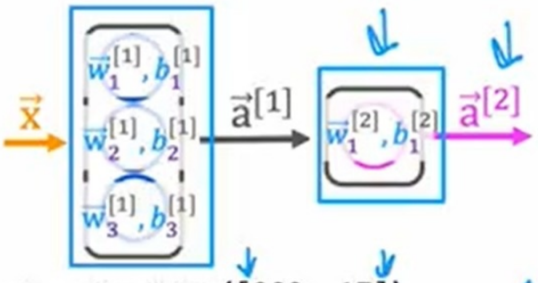
- teaching practice not Real w.

After training weights
↳ large num.
— sharp decisions

Build own dense layer using Numpy

Forward Propogation in a Single Layer:

forward prop (coffee roasting model)



$$a_1^{[2]} = g(\vec{w}_1^{[2]} \cdot \vec{a}^{[1]} + b_1^{[2]})$$

```
w2_1 = np.array([-7, 8, 9])
b2_1 = np.array([3])
z2_1 = np.dot(w2_1,a1)+b2_1
```

$w_1^{[2]}$ w2_1

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

1D arrays

$$a_1^{[1]} = g(\vec{w}_1^{[1]} \cdot \vec{x} + b_1^{[1]}) \qquad a_2^{[1]} = g(\vec{w}_2^{[1]} \cdot \vec{x} + b_2^{[1]}) \qquad a_3^{[1]} = g(\vec{w}_3^{[1]} \cdot \vec{x} + b_3^{[1]})$$

```
w1_1 = np.array([1, 2])          w1_2 = np.array([-3, 4])          w1_3 = np.array([5, -6])
b1_1 = np.array([-1])            b1_2 = np.array([1])              b1_3 = np.array([2])
z1_1 = np.dot(w1_1,x)+b1_1       z1_2 = np.dot(w1_2,x)+b1_2        z1_3 = np.dot(w1_3,x)+b1_3
a1_1 = sigmoid(z1_1)             a1_2 = sigmoid(z1_2)              a1_3 = sigmoid(z1_3)

a1 = np.array([a1_1, a1_2, a1_3])
```

in Numpy:

```
def dense(a_in, w, b):
```

```
w = np.array([
    [1, -3, 5],
    [2, 4, -6]
])
```

w1 w2 w3

```
b = np.array([-1, 1, 2])
```

$\vec{a}^{[0]} = \vec{x}$ a_in = np.array([ ])

```
def dense(a_in, W, b):
    units = W.shape[1]        ---> 3
                              (2 x 3 matrix)
    a_out = np.zeros(units)
    for j in range(units):
        w = W[:, j]
        z = np.dot(w, a_in) + b[j]
        a_out[j] = g(z)
    return a_out
```

now

```
def seq(x):
    a1 = dense(x, W1, b1)
    a2 = dense(a1, W2, b2)
    f_x = a4
    return f_x
```

For loops vs. vectorization

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

W = np.array([[1, -3, 5],
              [-2, 4, -6]])
b = np.array([-1, 1, 2])

def dense(a_in,W,b):
  units = W.shape[1]
  a_out = np.zeros(units)
  for j in range(units):
    w = W[:,j]
    z = np.dot(w, a_in) + b[j]
    a_out[j] = g(z)
 return a_out

[1,0,1]
```

vectorized

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

W = np.array([[1, -3, 5],
              [-2, 4, -6]])
B = np.array([[-1, 1, 2]])

def dense(A_in,W,B):
  Z = np.matmul(A_in,W) + B
  A_out = g(Z)
  return A_out

[[1,0,1]]
```

2D array

same

1x3 2D array

matrix multiplication

$$z = \vec{a} \cdot \vec{w} = \vec{a}^T \cdot \vec{w}$$

Numpy broadcasting

Vector b → resized in a way that can be added to the A_inxW