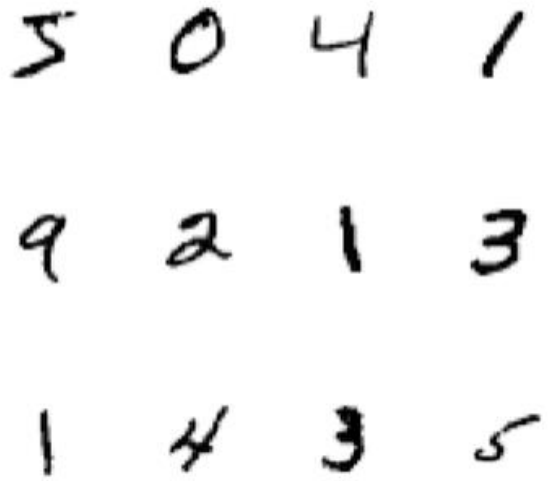
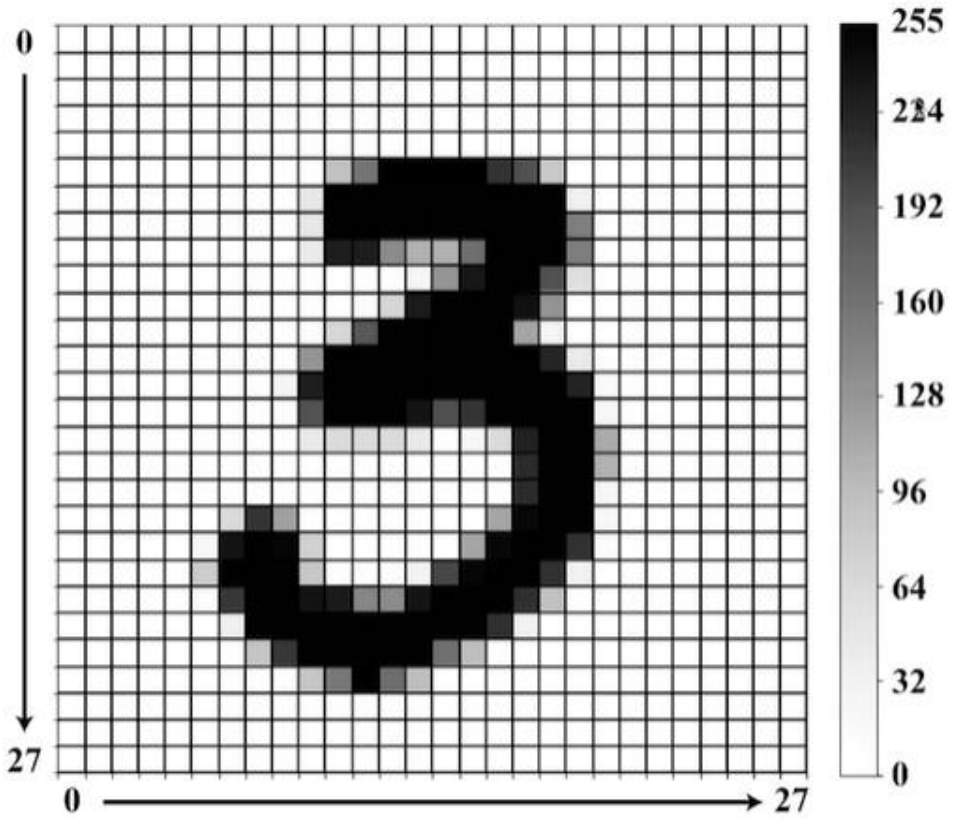
Neural Networks

Sample MNIST Digits

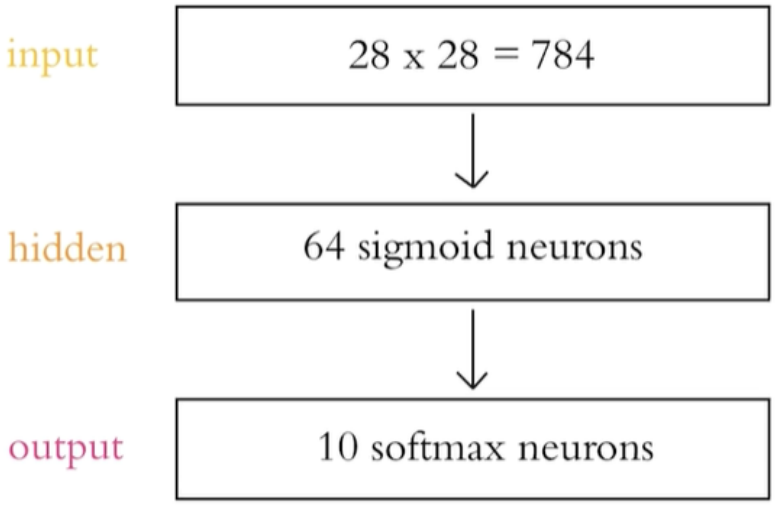


MNIST is Modified National Institute of Standards and Technology database

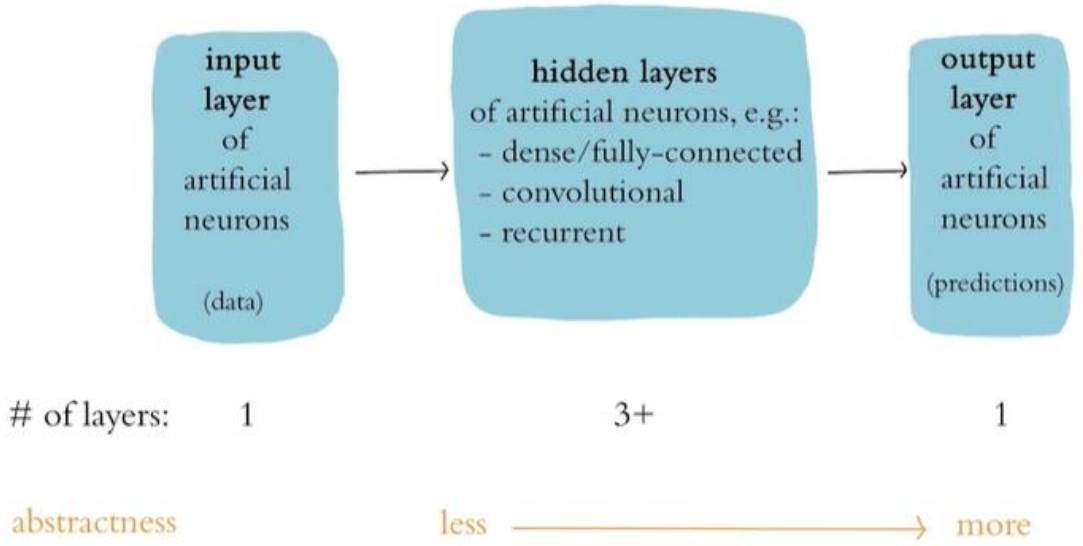


MNIST Digit

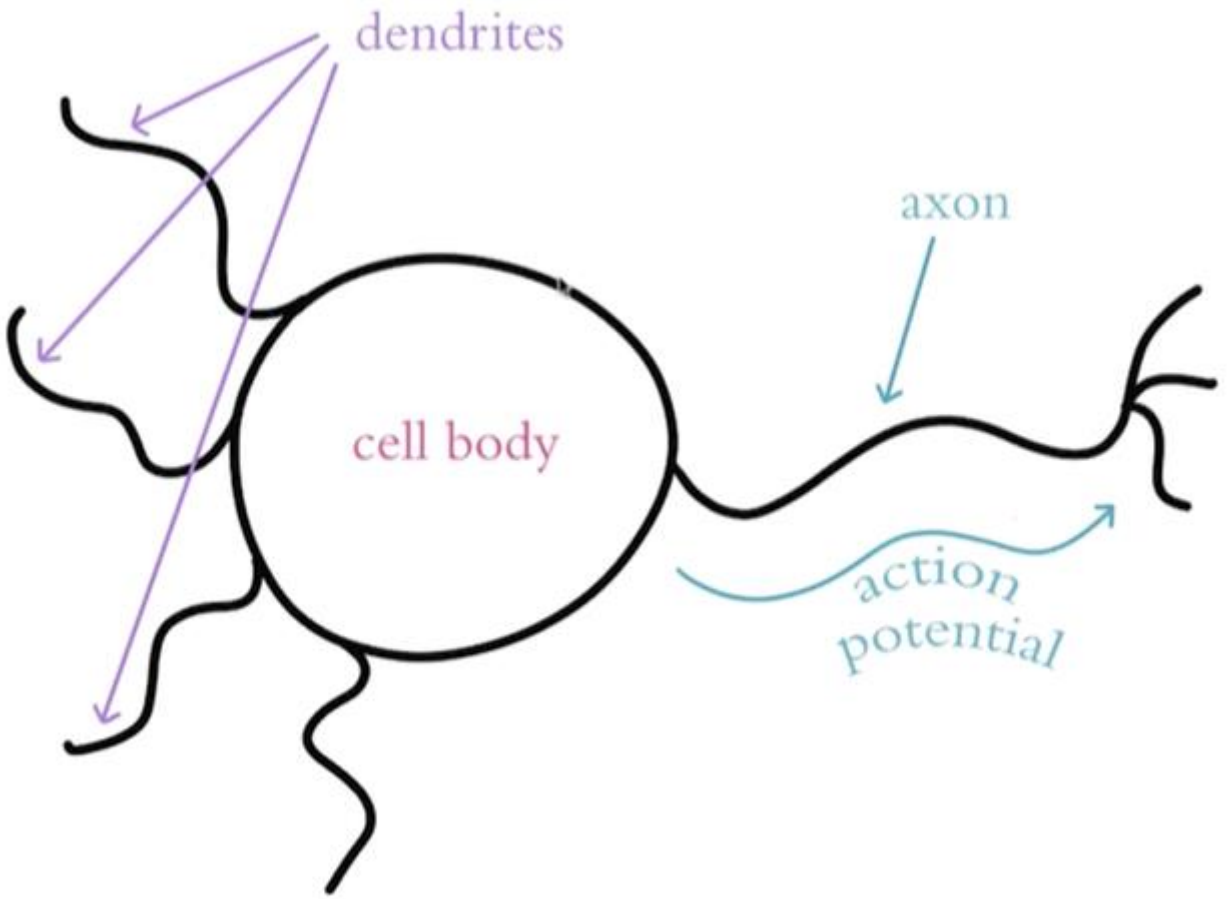
Shallow Neural Net that we will be building soon



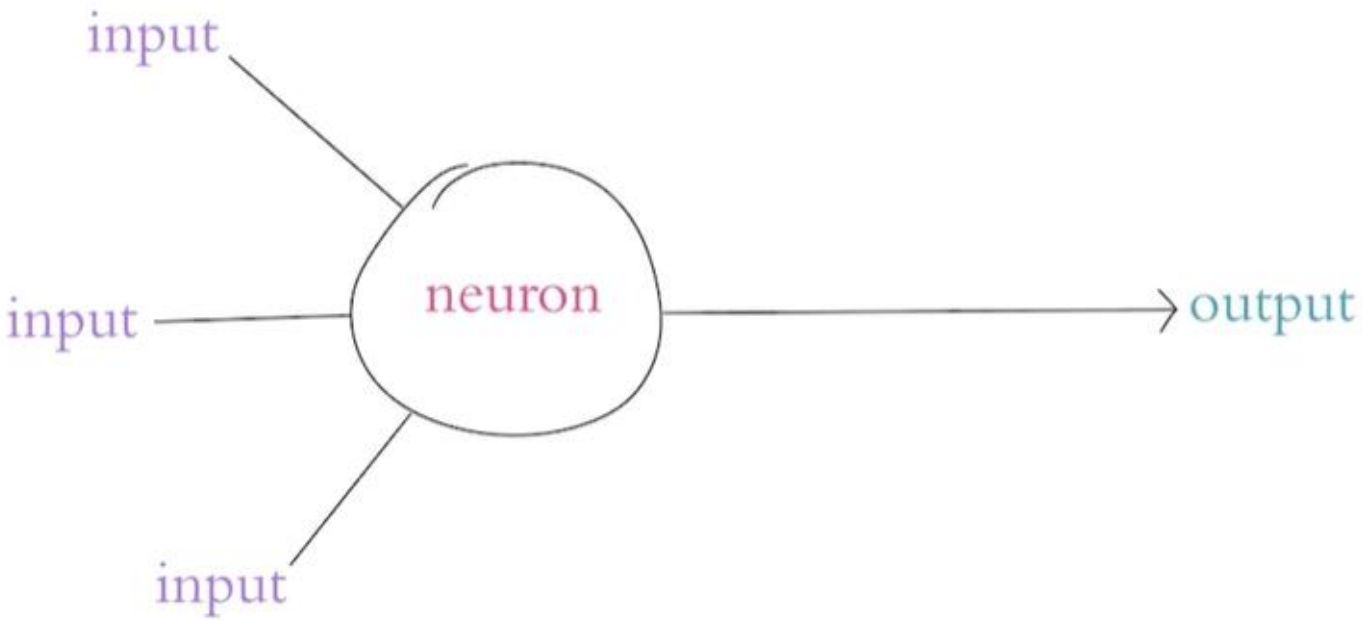
A Generalized Deep Learning Model

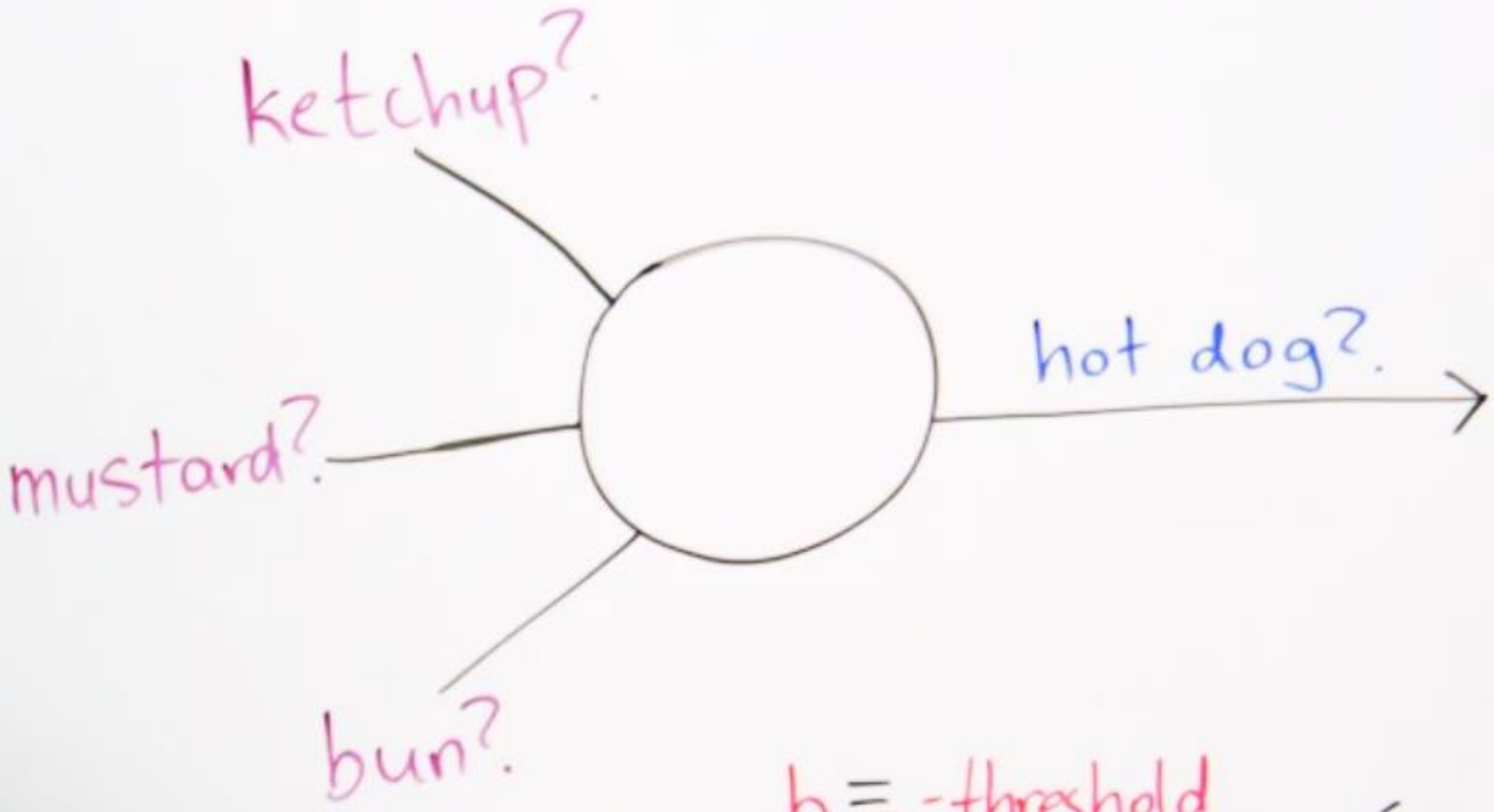


A Biological Neuron



Frank Rosenblatt – The Perceptron

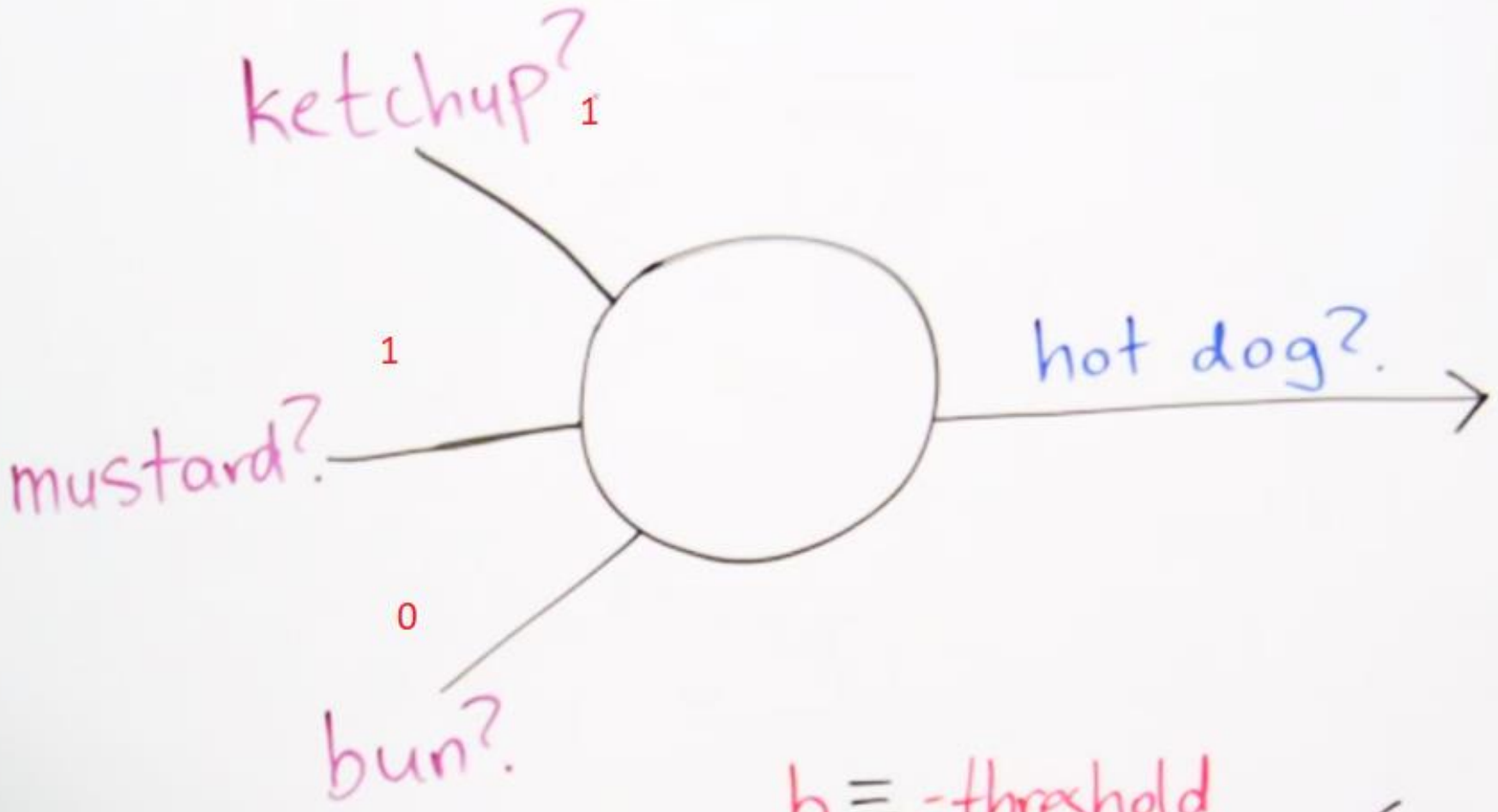




$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$b \equiv -\text{threshold}$$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

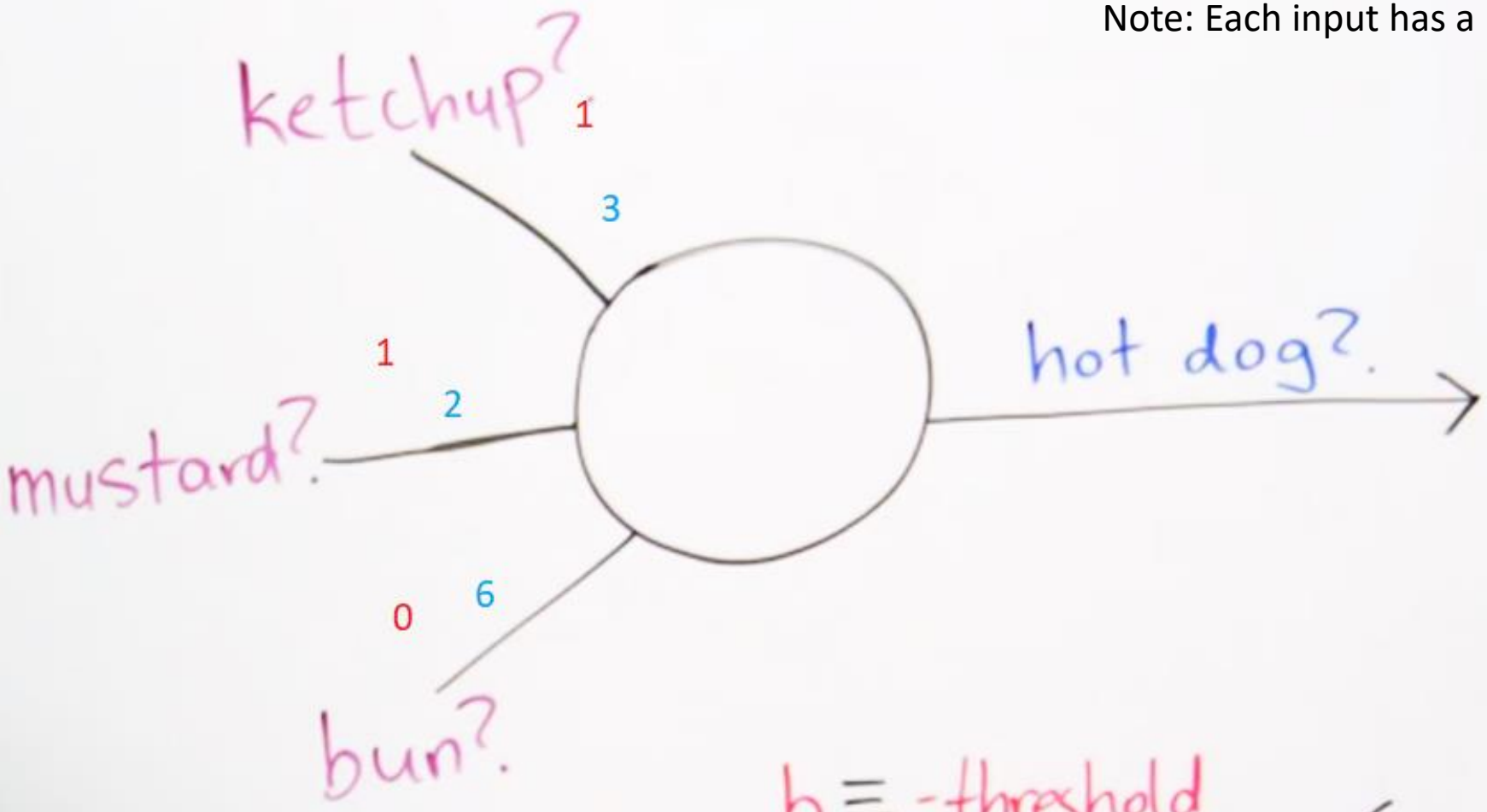


$b \equiv -\text{threshold}$

$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

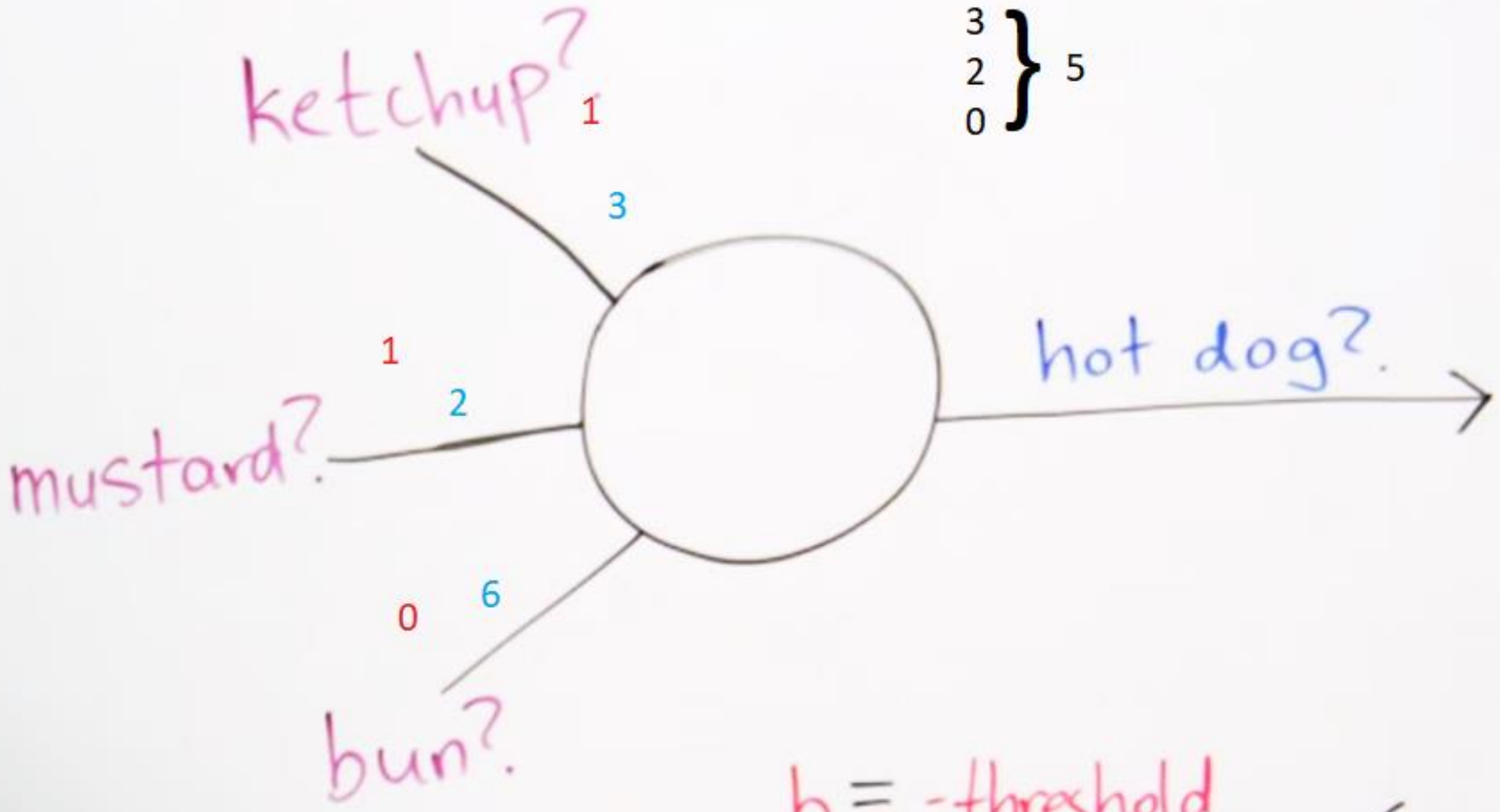
Note: Each input has a particular weight



$b \equiv -\text{threshold}$

$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

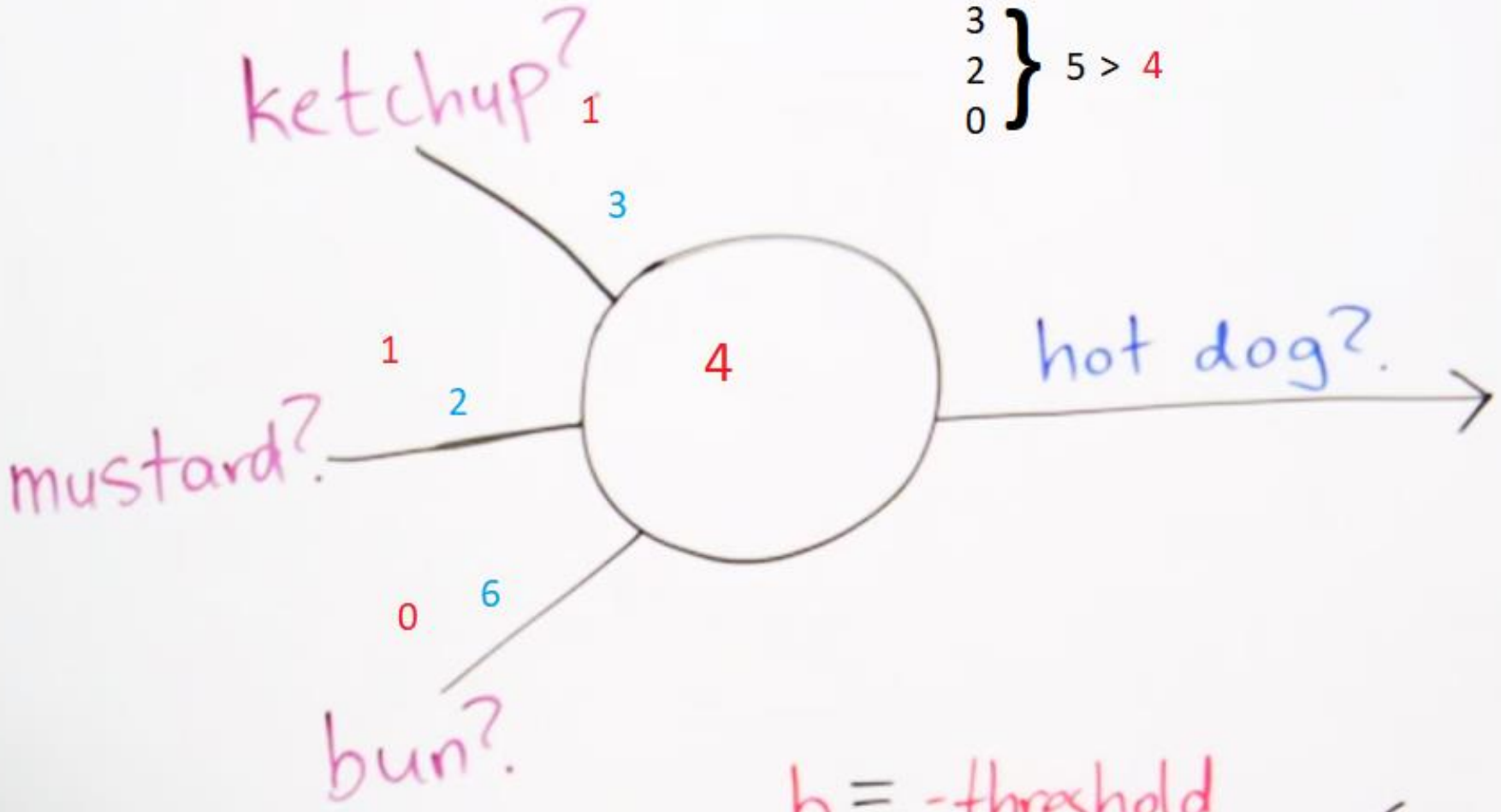


3
2
0
} 5

$b \equiv -\text{threshold}$

$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

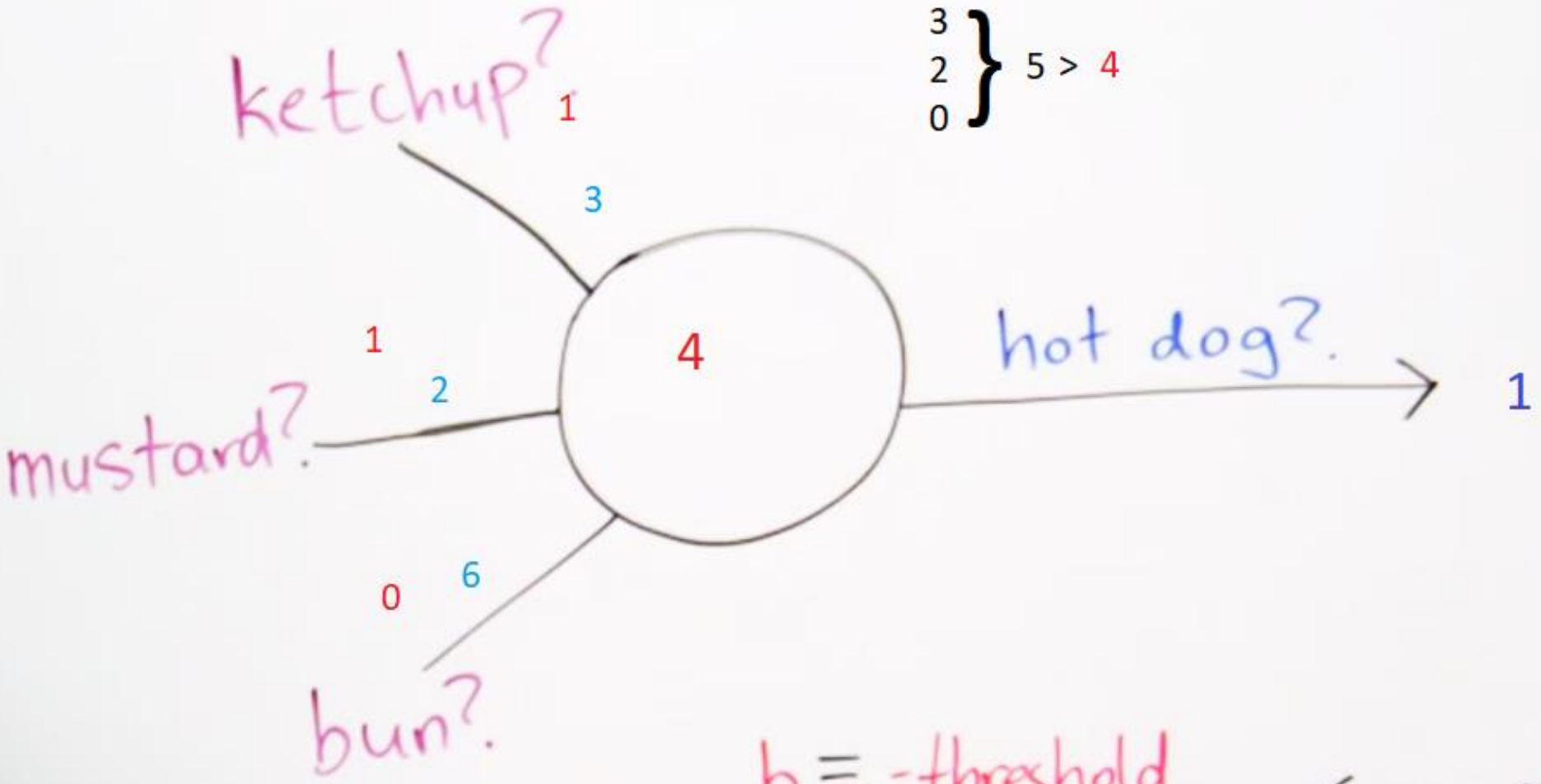


3
2
0
} 5 > 4

$b \equiv -\text{threshold}$

$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

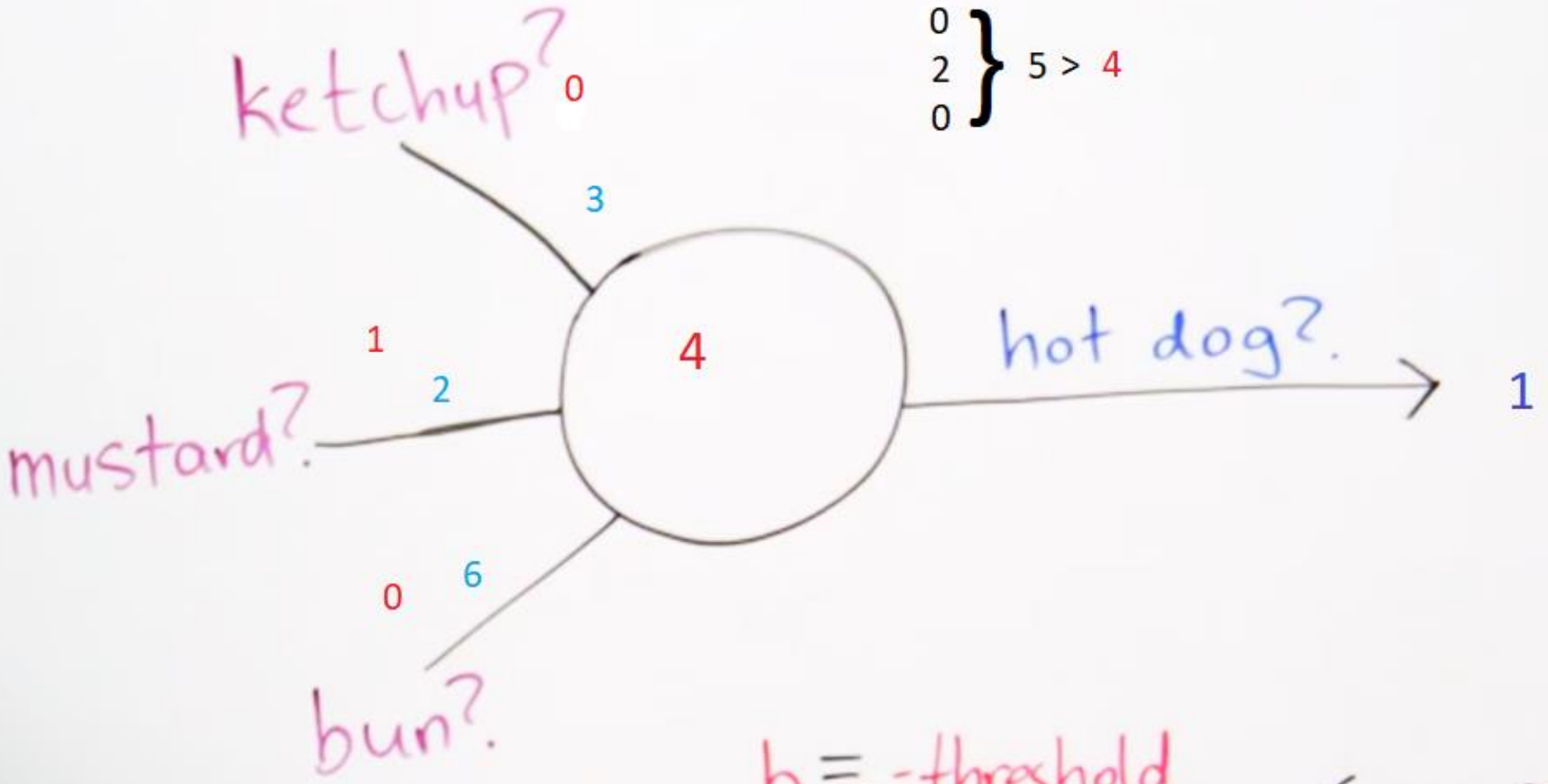


3
2
0
} 5 > 4

$b \equiv -\text{threshold}$

$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

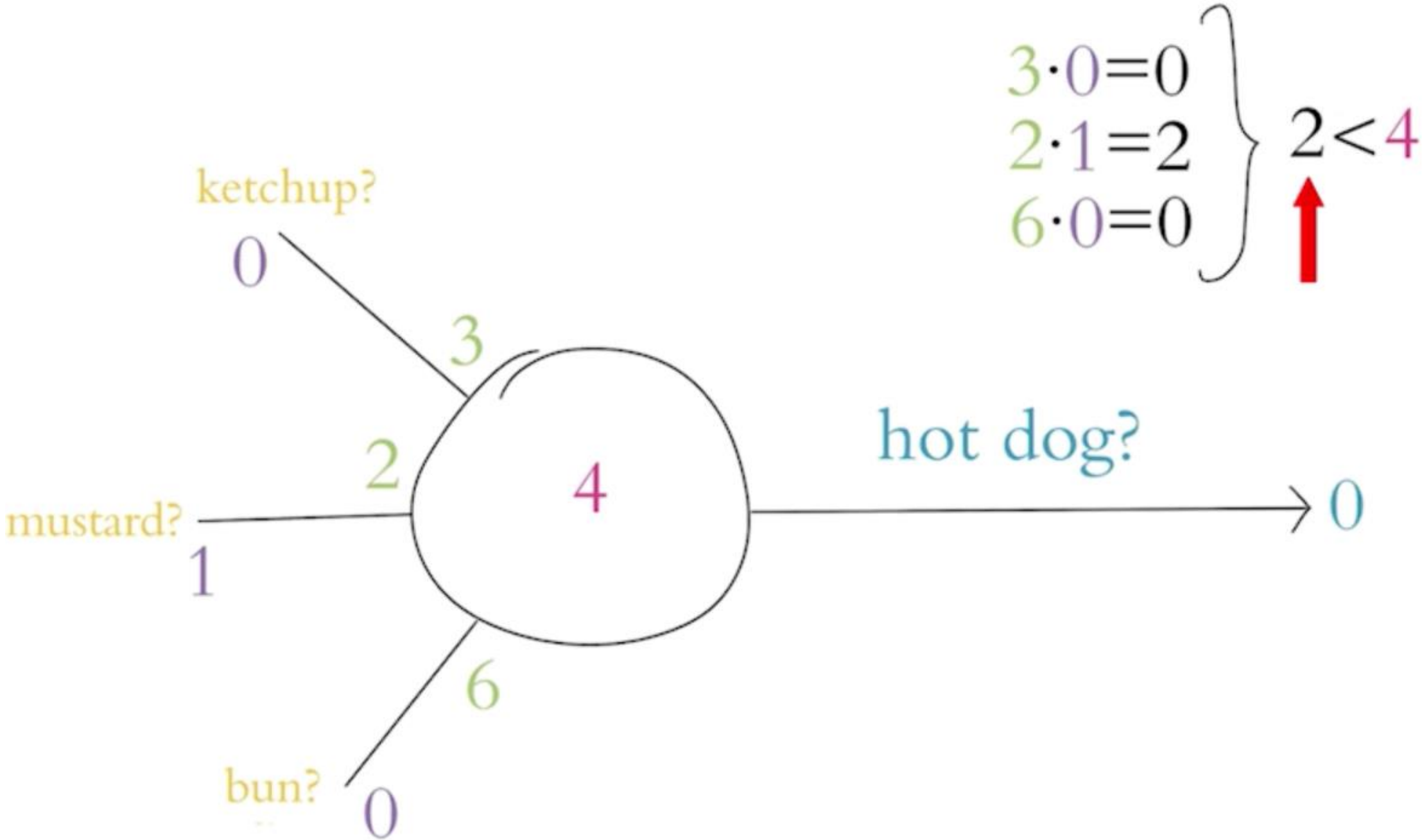


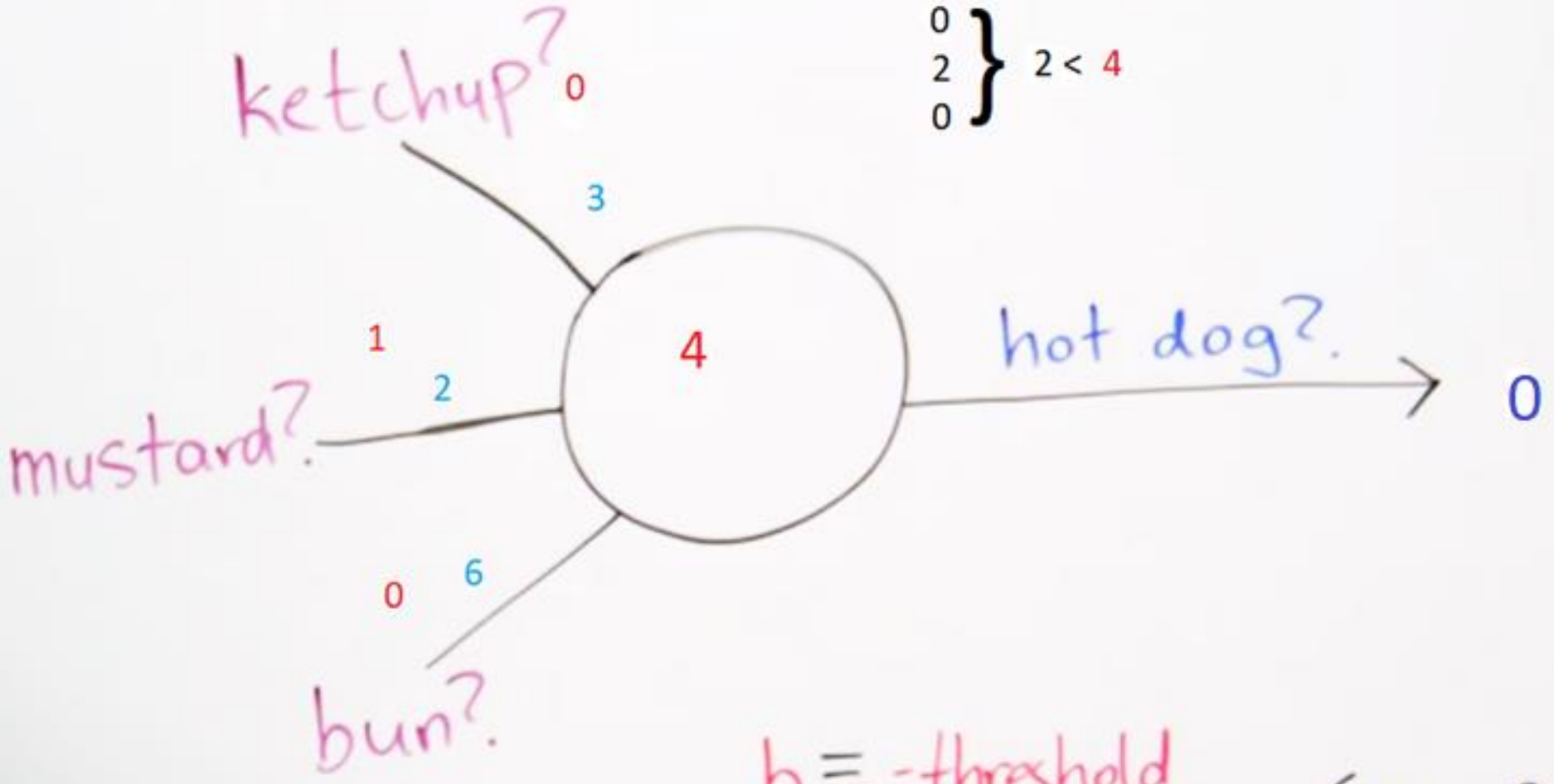
0
2
0
} 5 > 4

$b \equiv -\text{threshold}$

$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$


3·0=0
2·1=2
6·0=0
2<4
ketchup?
0
3
2
mustard?
1
4
hot dog?
0
6
bun?
0

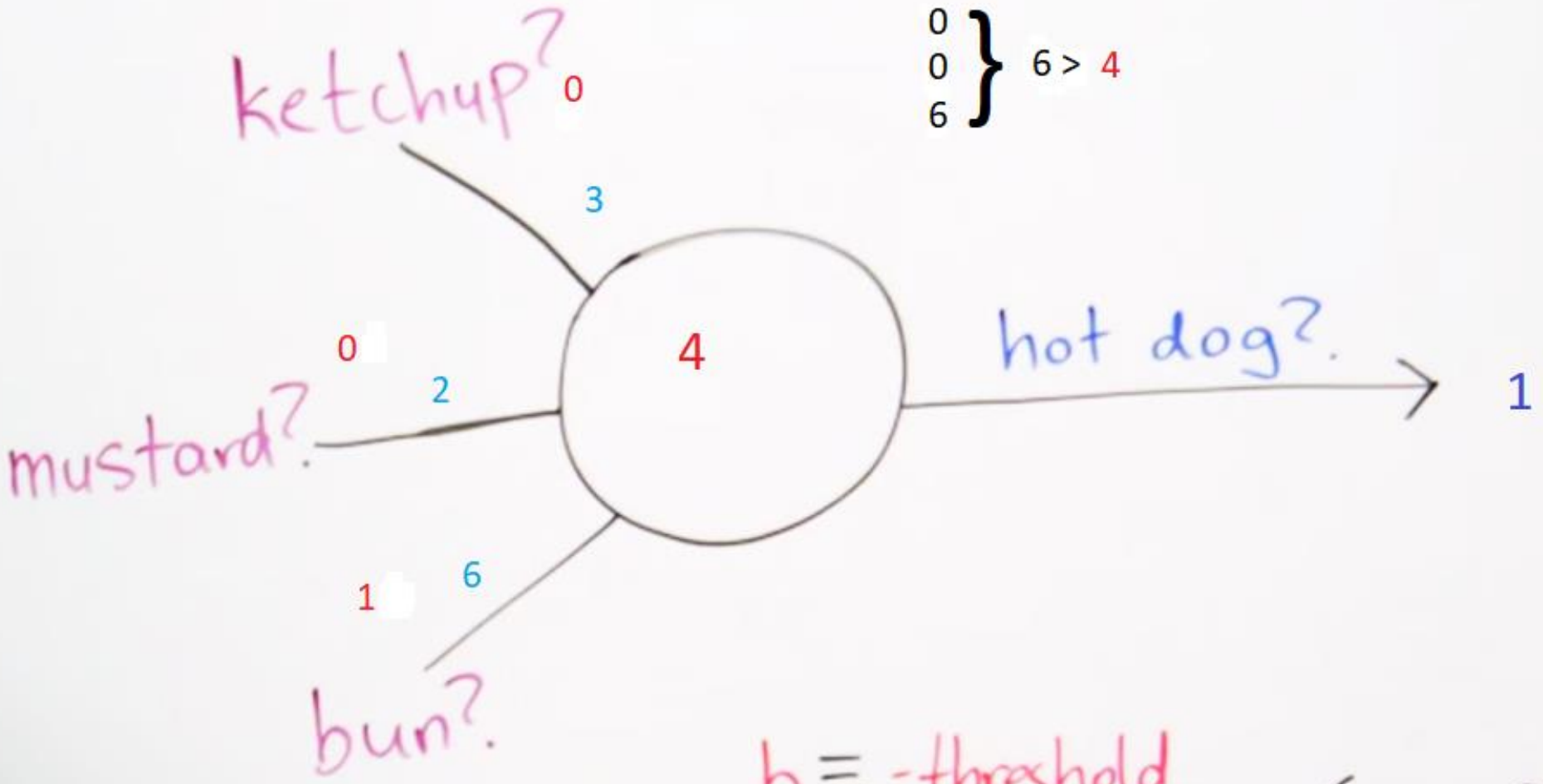


0
2
0
} 2 < 4

$b \equiv -\text{threshold}$

$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

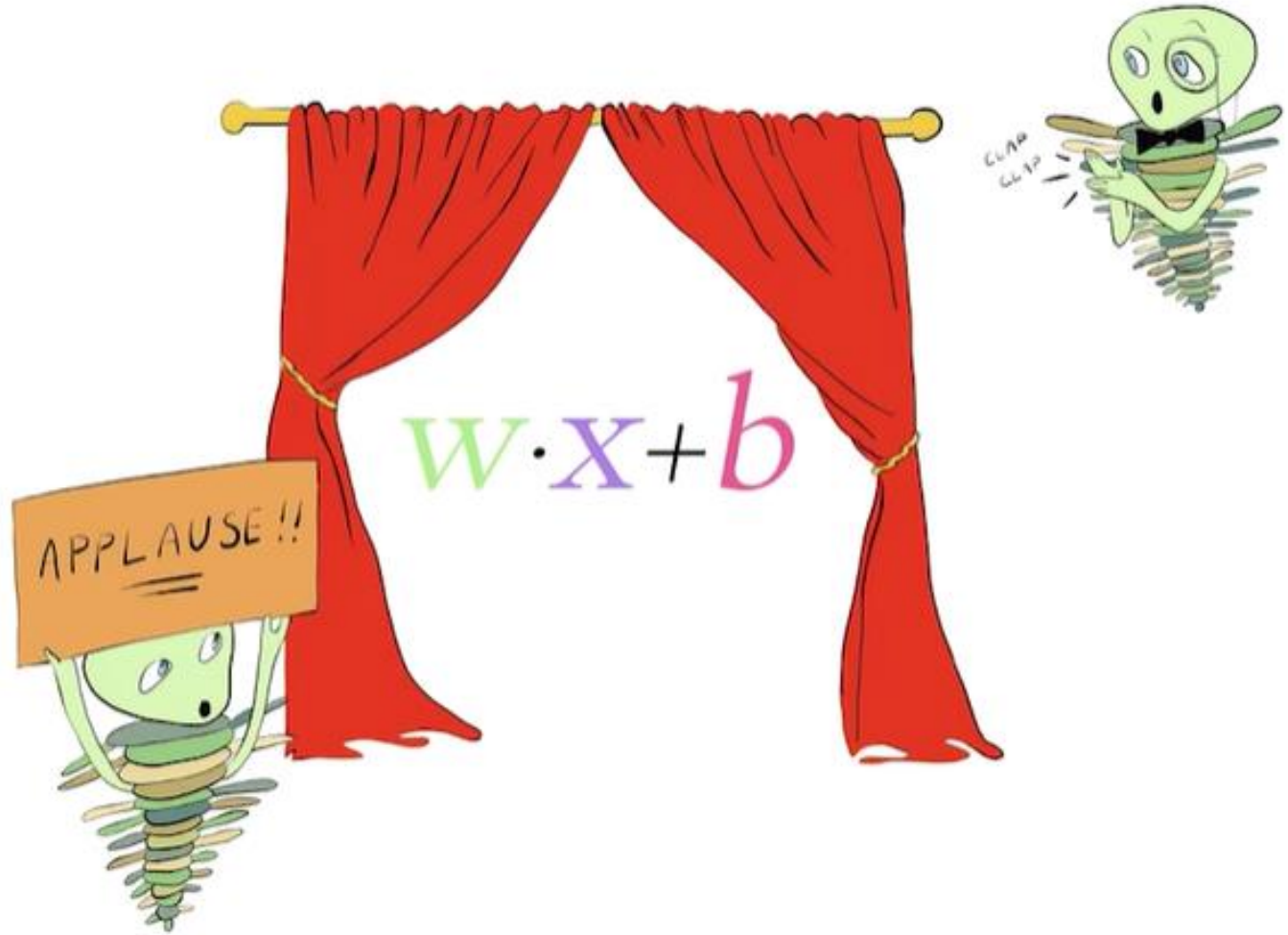


0
0
6
} 6 > 4

$b \equiv -\text{threshold}$

$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

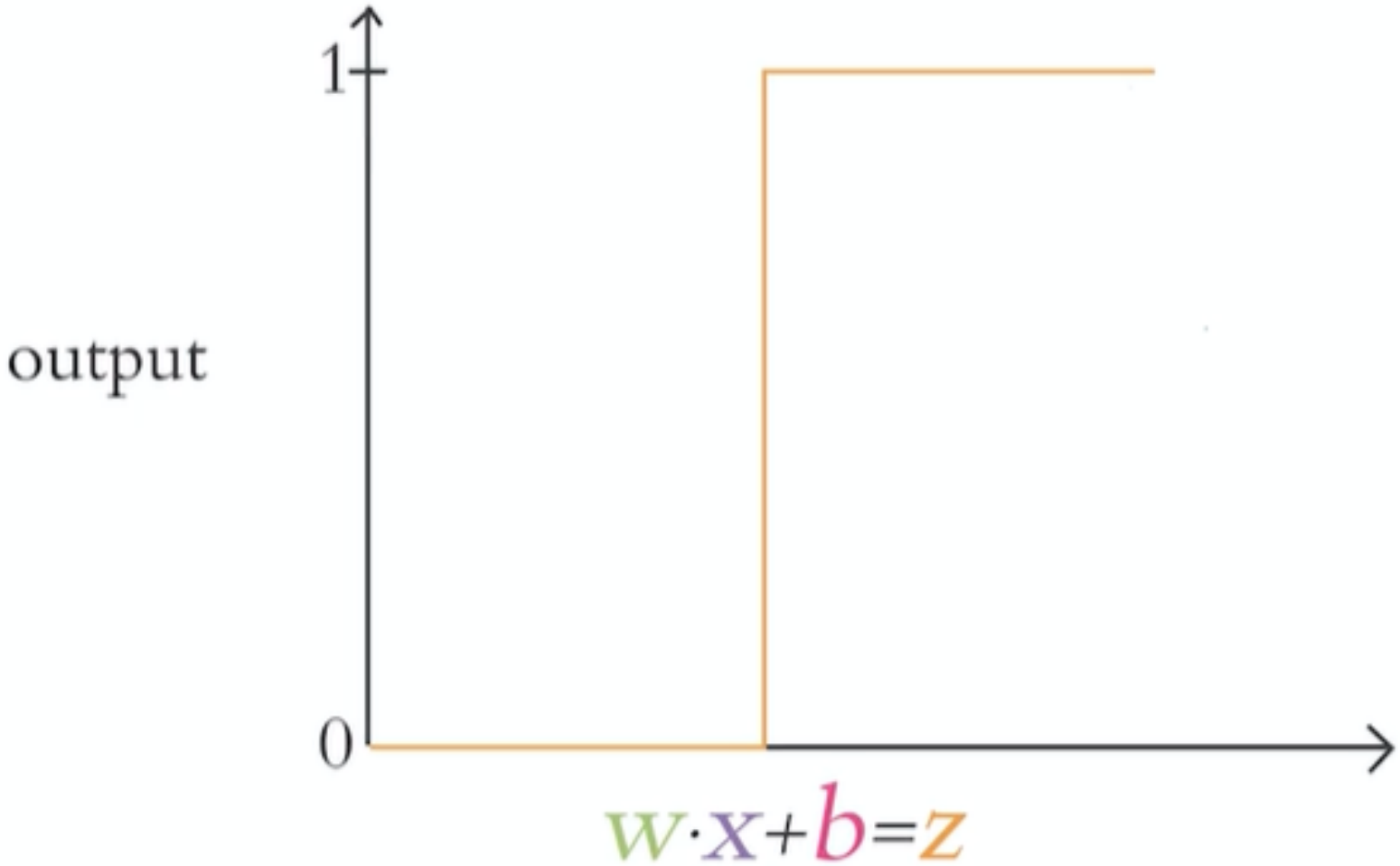

APPLAUSE !!
$w \cdot x + b$

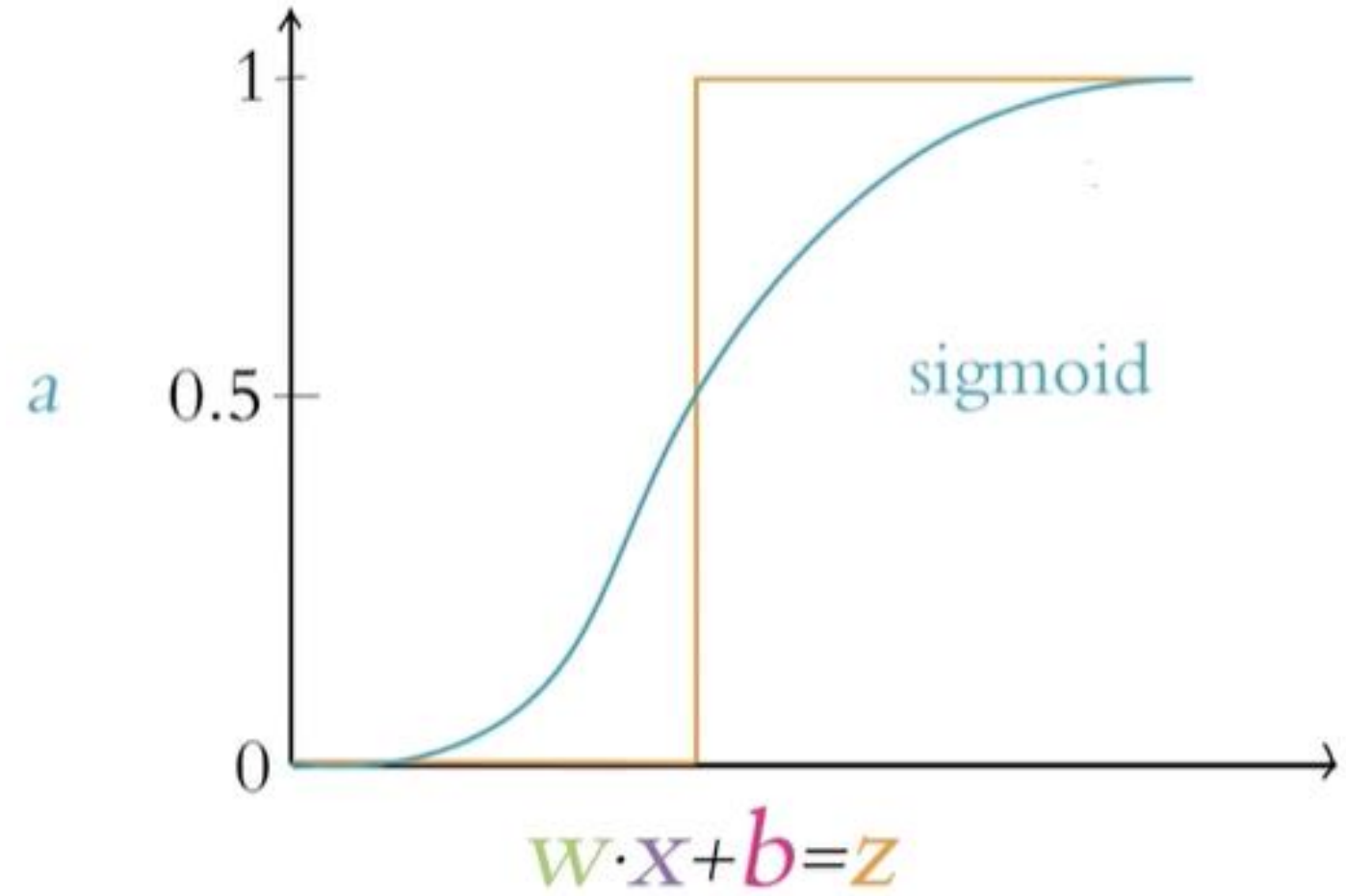
Design neural network architecture

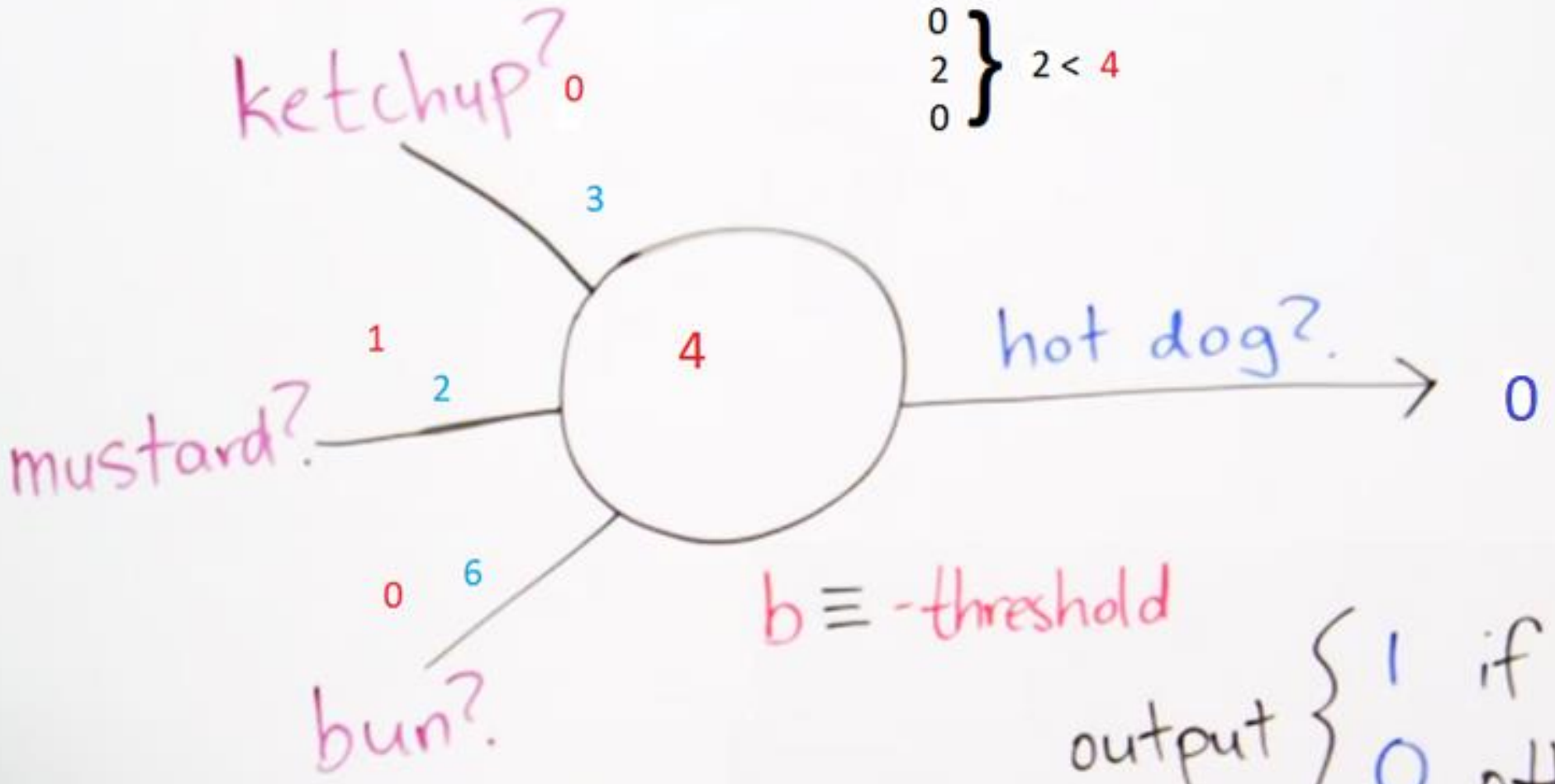
In [17]:
```
model = Sequential()
model.add(Dense(64, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
```

In [18]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```


output
1
0
w·x+b=z


1
0.5
0
a
sigmoid
w·x+b=z



$$\begin{matrix}0\\2\\0\end{matrix}\Big\} \; 2 < 4$$

$b \equiv -\text{threshold}$

$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

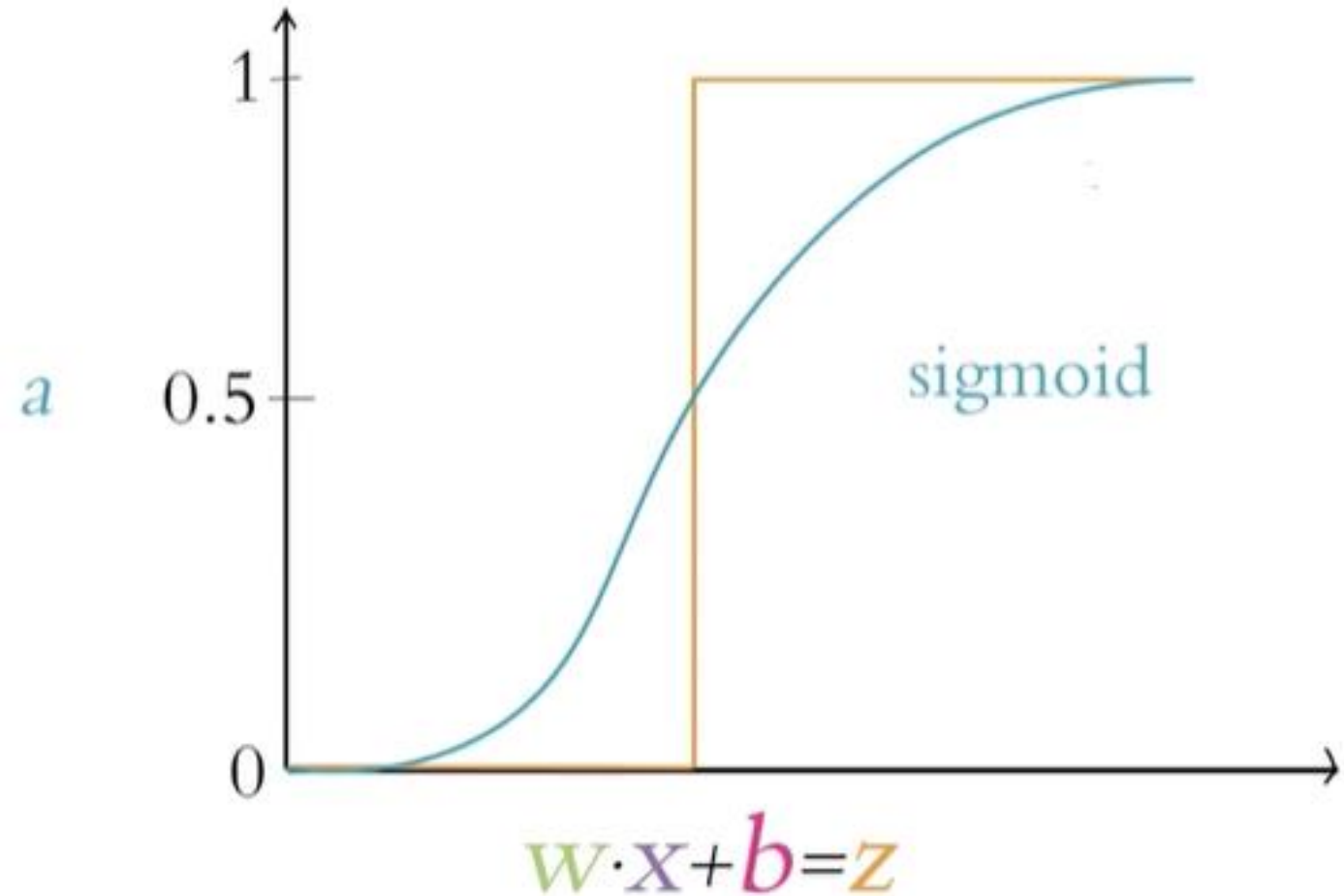
$$\text{sigmoid } \sigma(z) = \frac{1}{1+e^{-z}}$$

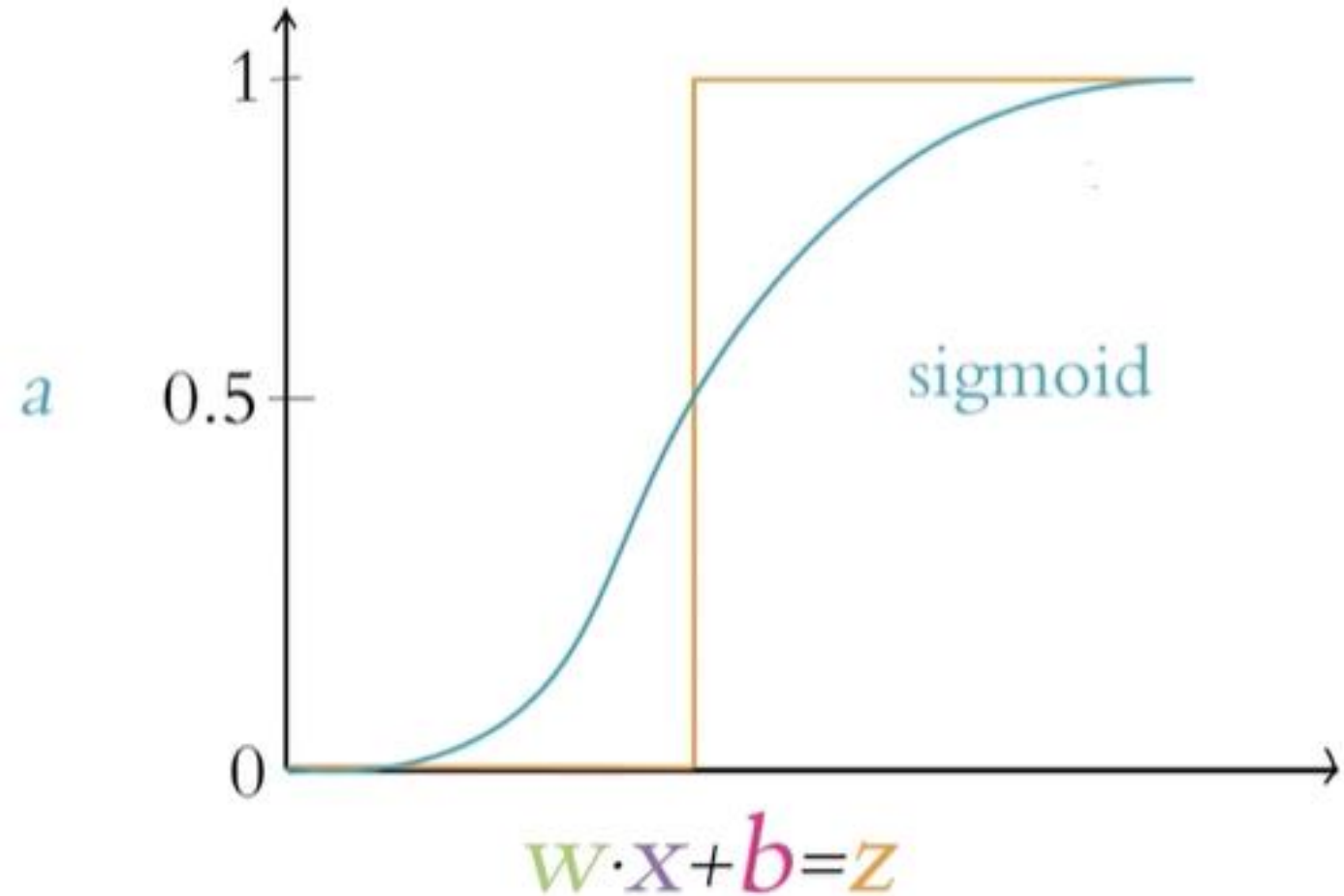
Design neural network architecture

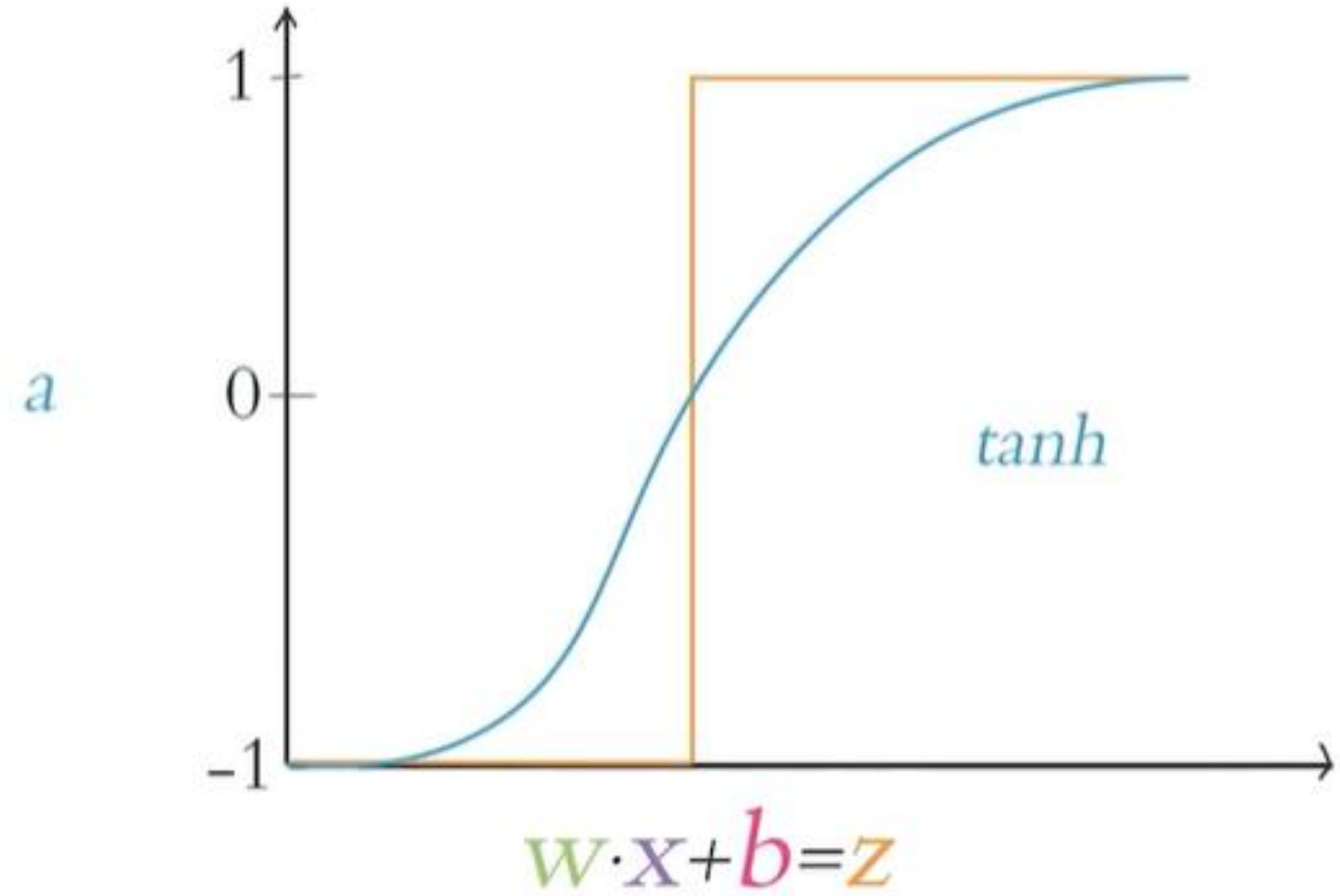
In [17]:
```
model = Sequential()
model.add(Dense(64, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
```

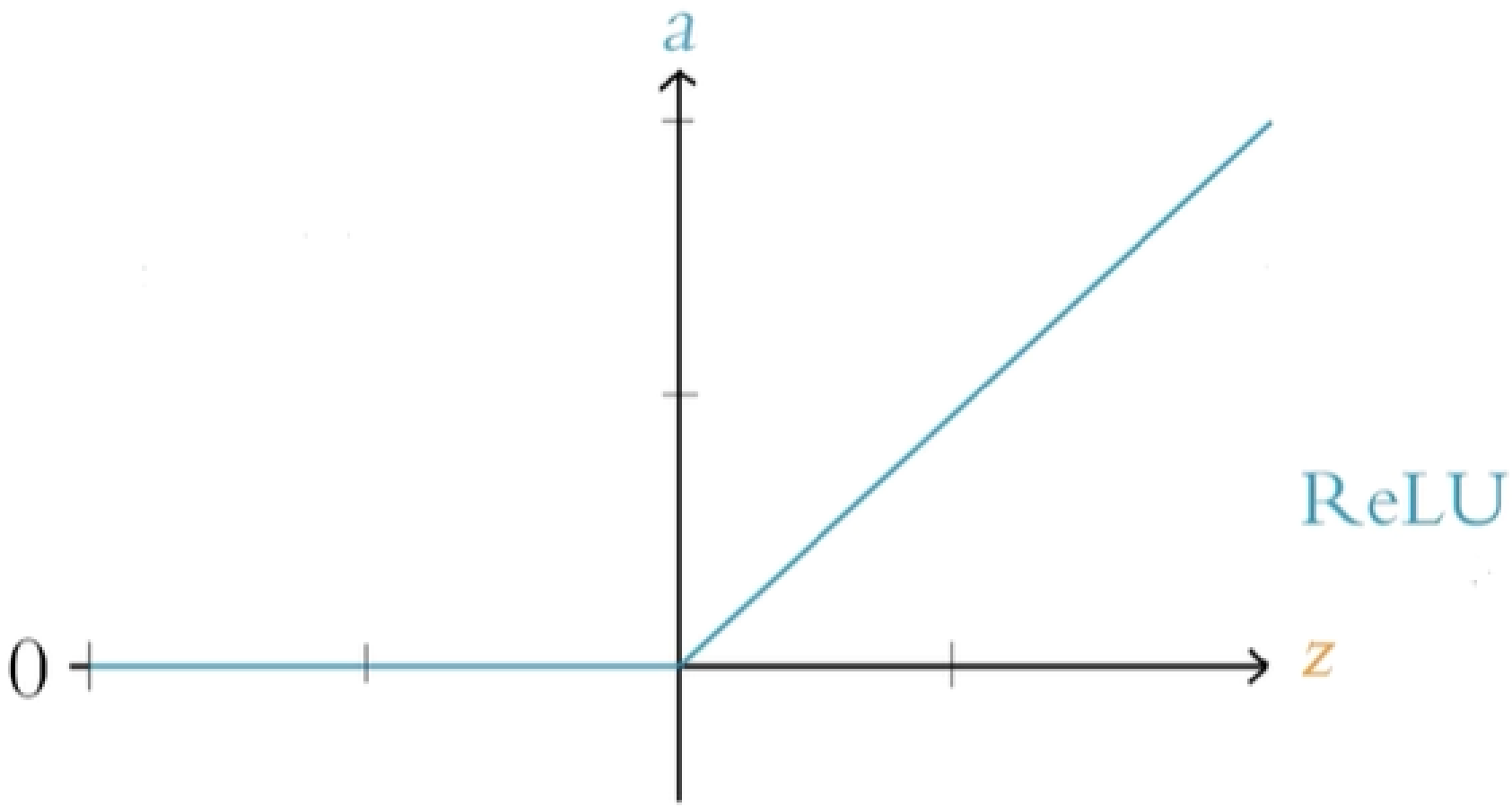
In [18]:
```
model.summary()
```

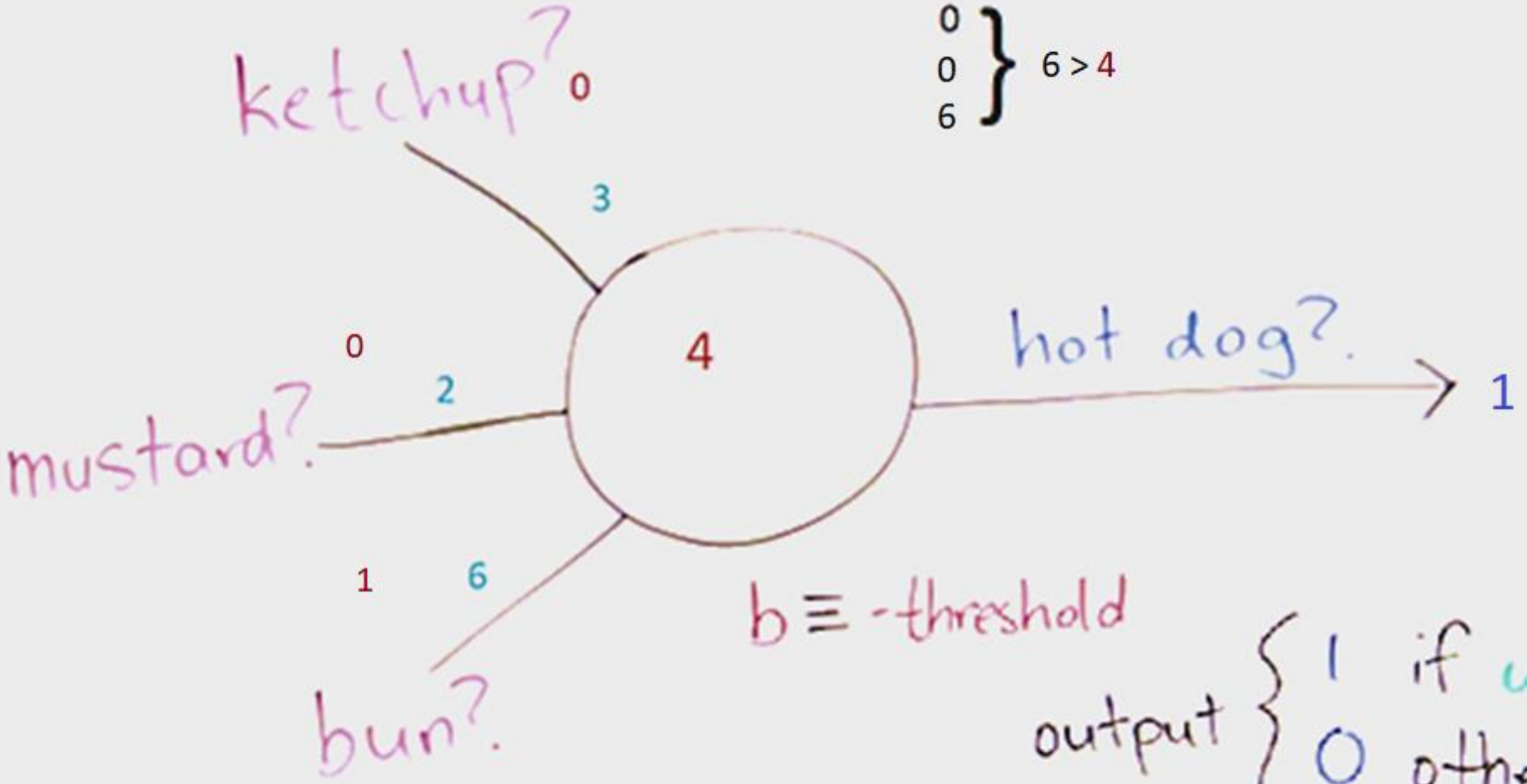
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```


1
0.5
0
a
sigmoid
w·x+b=z


1
0.5
0
a
sigmoid
w·x+b=z


1
0
-1
a
tanh
w·x+b=z


a
ReLU
0
z



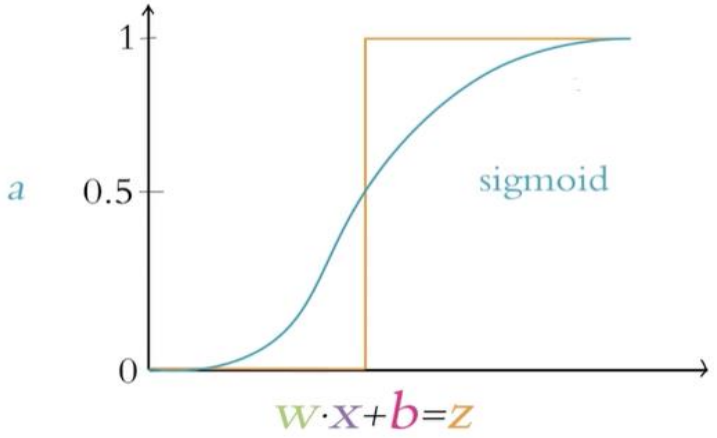
0
0
6
} 6 > 4

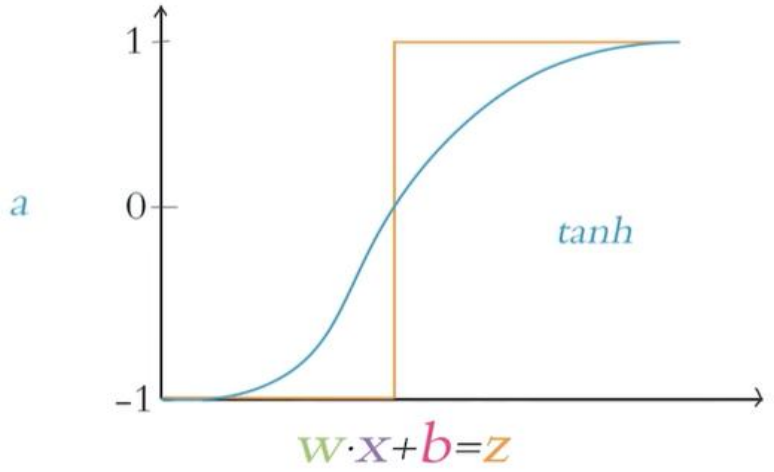
$$\text{output} \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases}$$

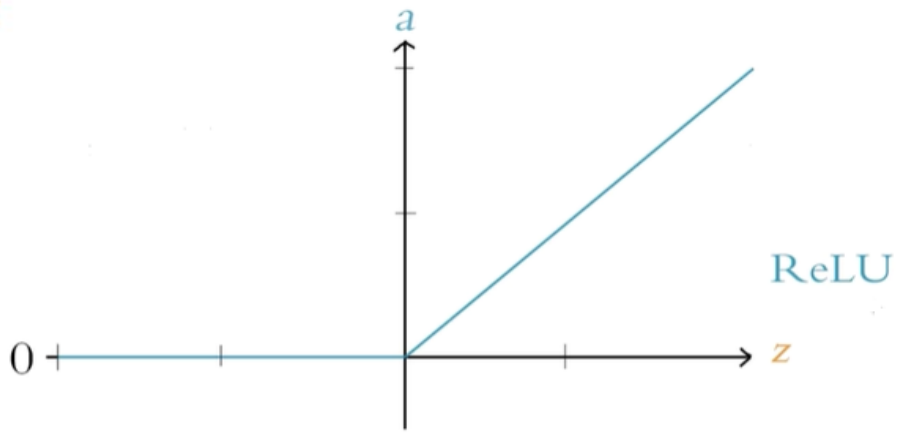
$$\sum_{i=1}^{n} w_i x_i = w \cdot x$$

$$\text{sigmoid } \sigma(z) = \frac{1}{1+e^{-z}}$$

$$\text{ReLU } \sigma(z) = \max(0, z)$$

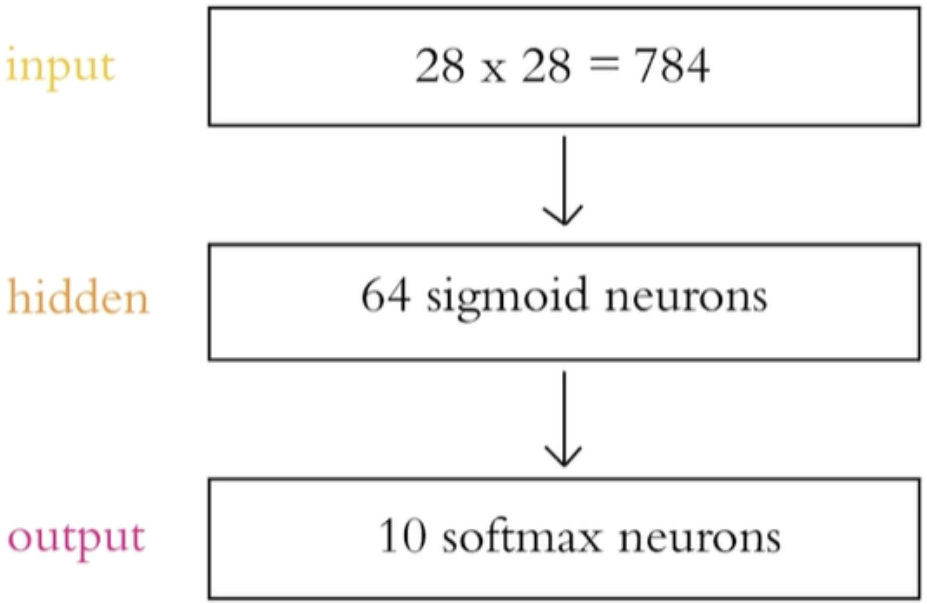

1
0.5
0
a
sigmoid
w·x+b=z


1
0
-1
a
tanh
w·x+b=z


a
ReLU
0
z

Key Concepts

- parameters:
  - weight w
  - bias b
- activation a
- neurons:
  - sigmoid
  - tanh
  - ReLU

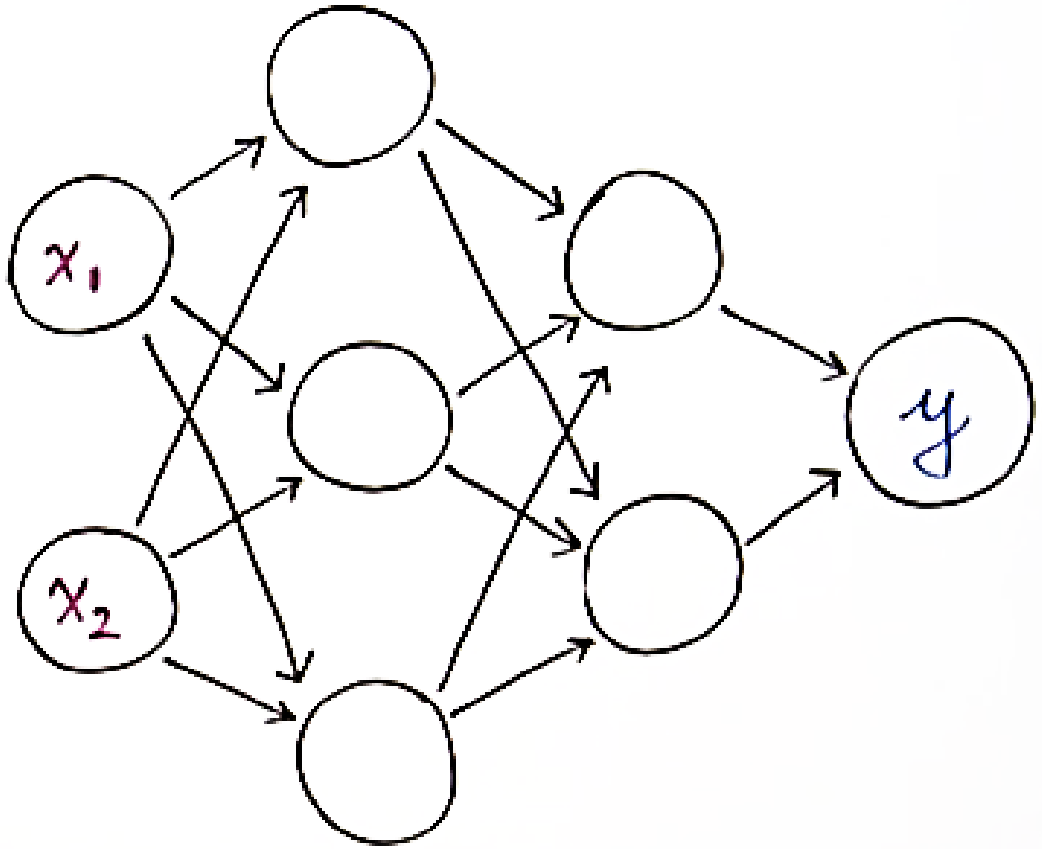


Design neural network architecture

In [17]:
```
model = Sequential()
model.add(Dense(64, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
```

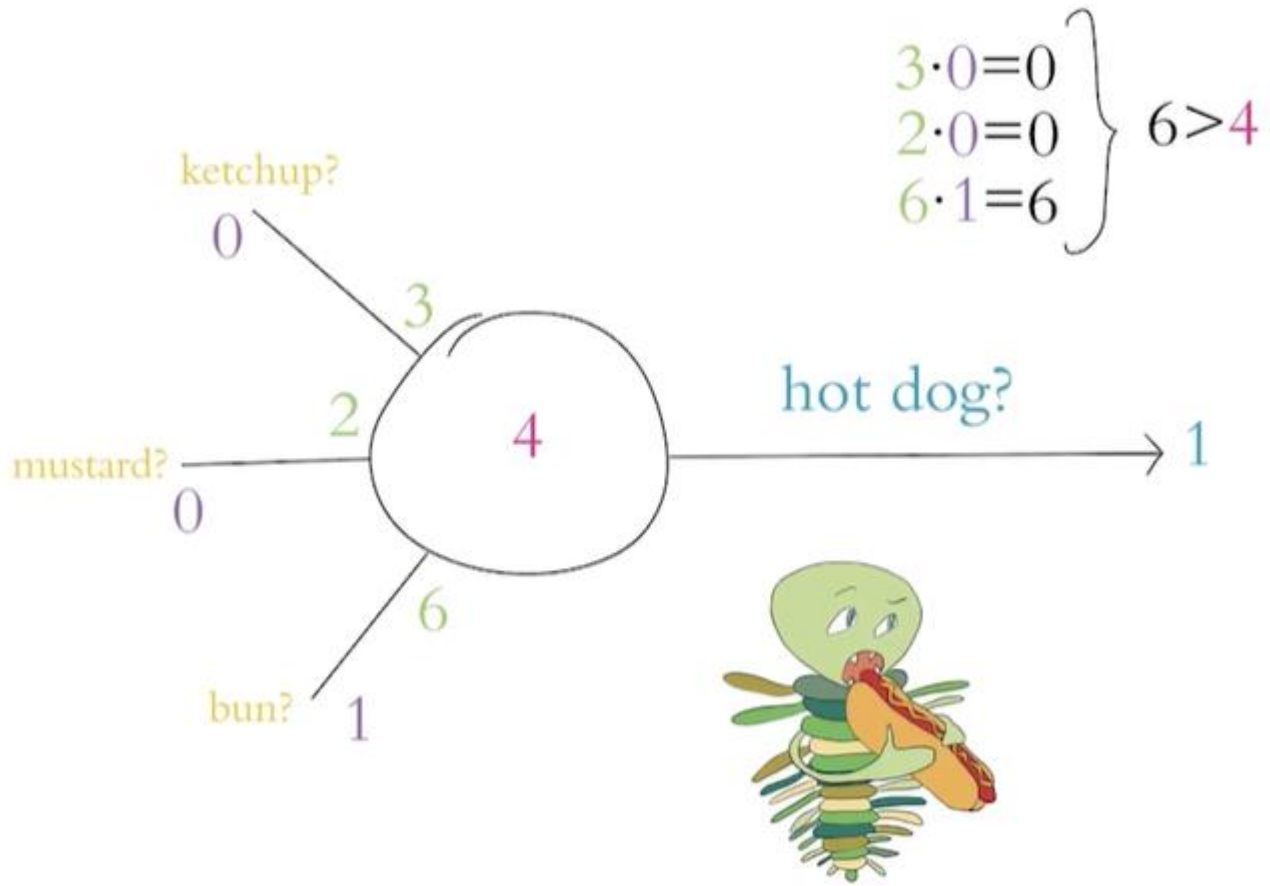
In [18]:
```
model.summary()
```

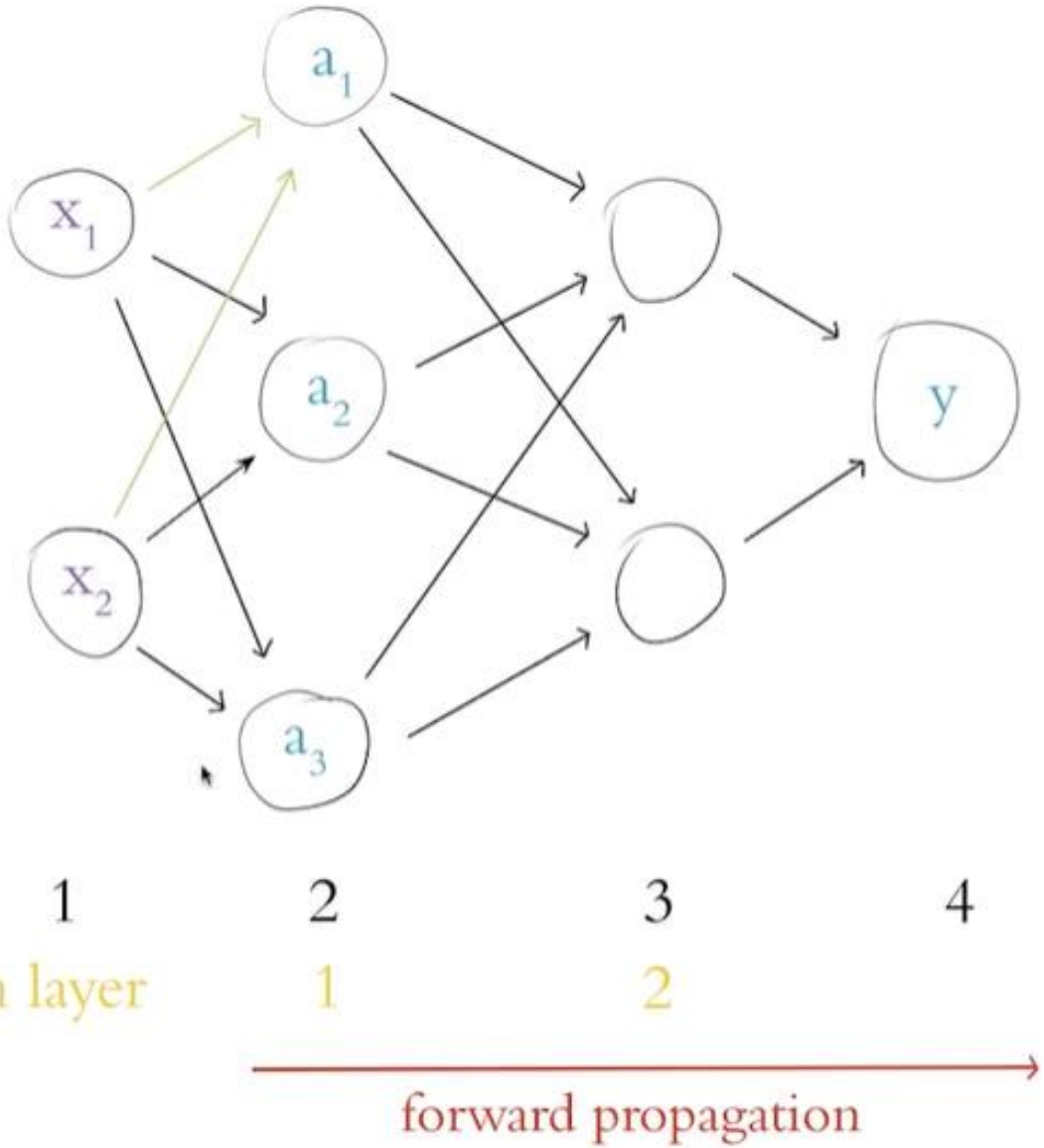
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```

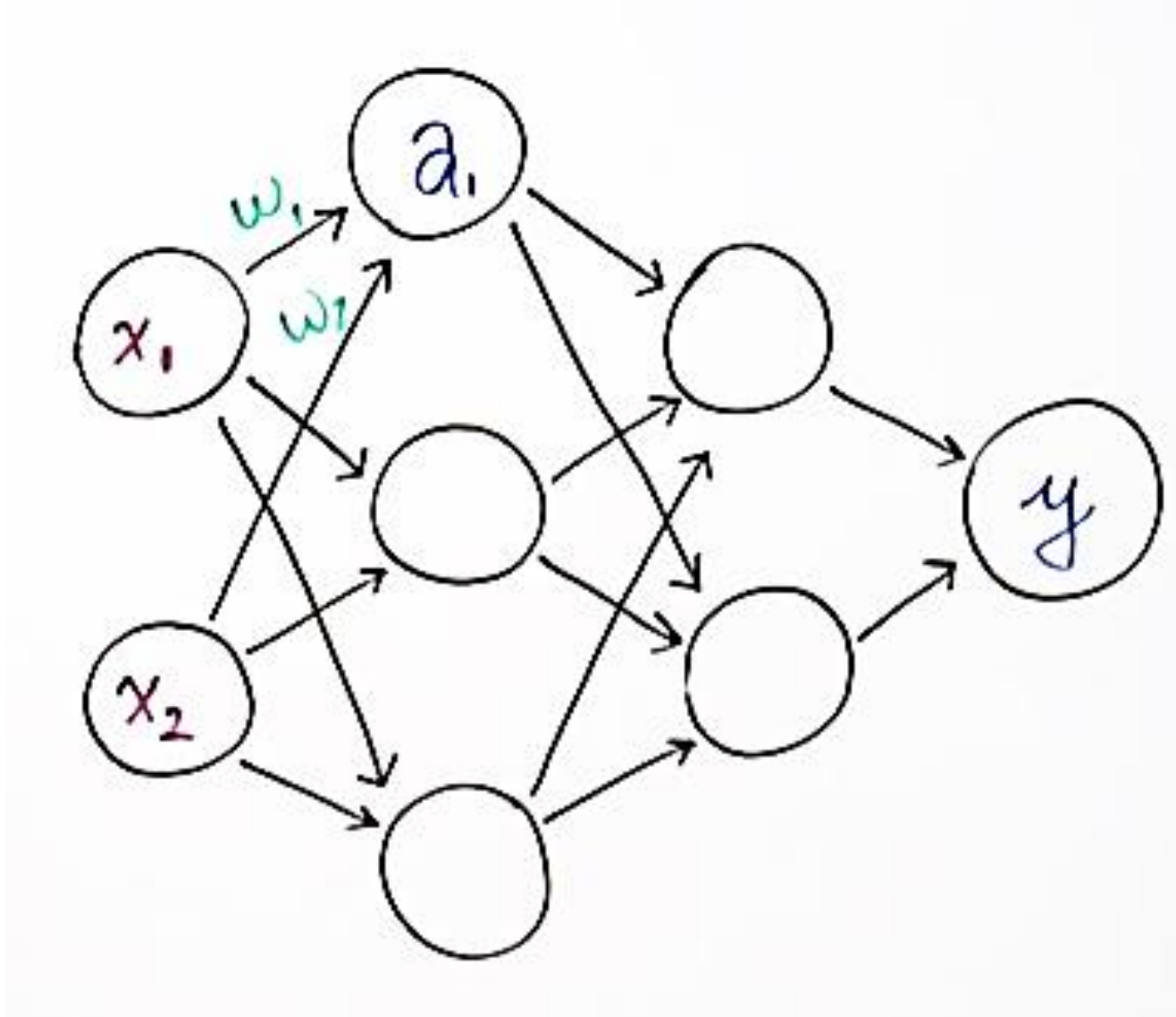


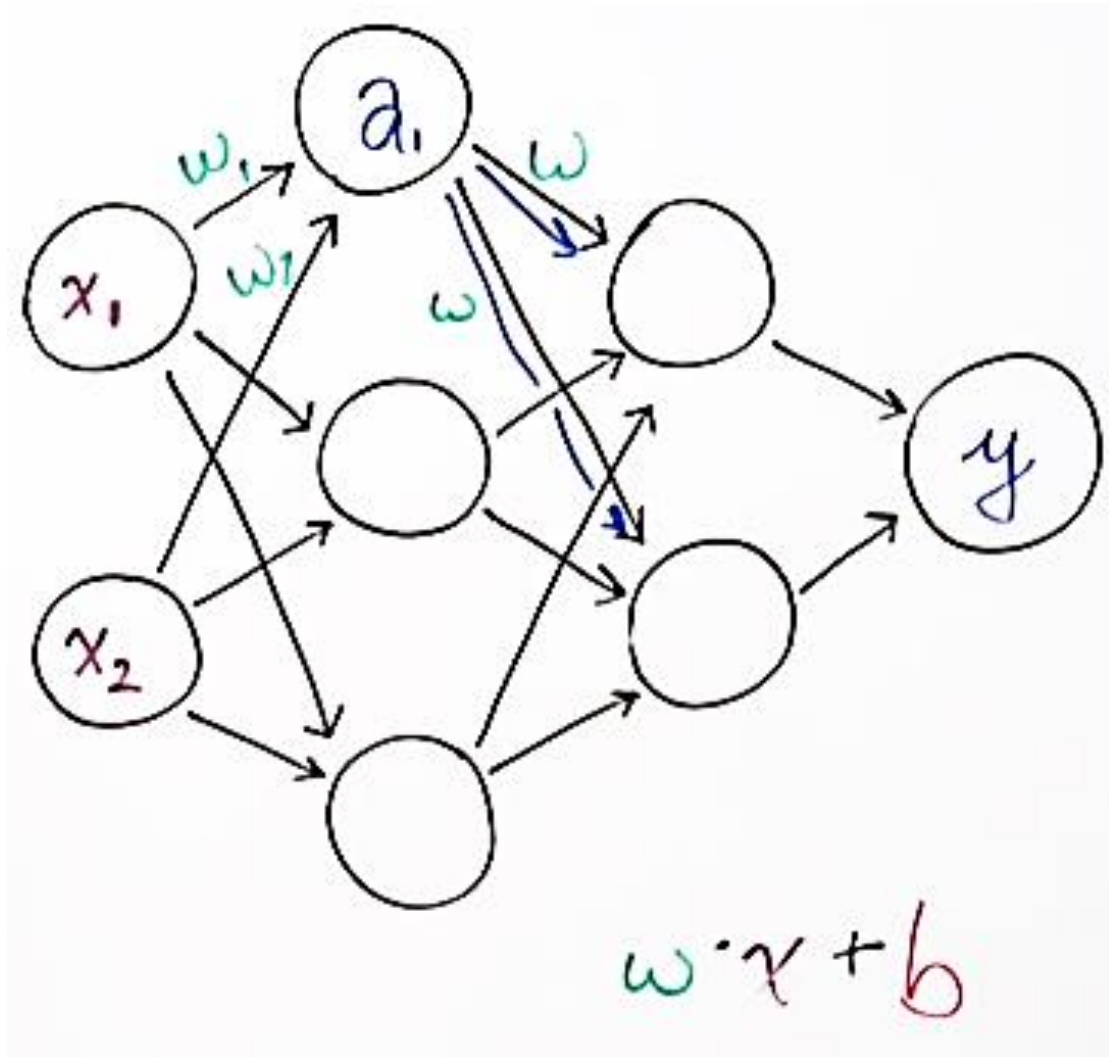
Hidden Layers

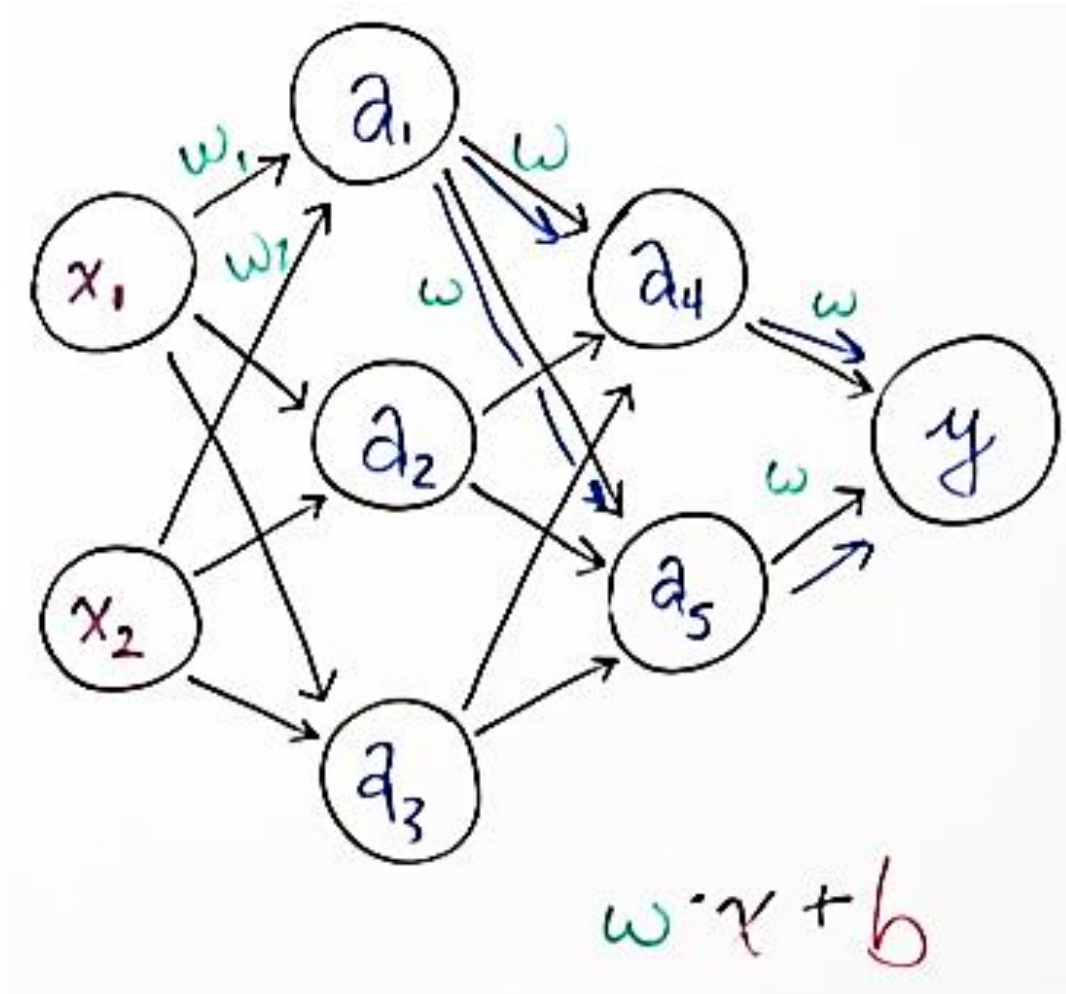
- Dense Layer/ Fully Connected Layer

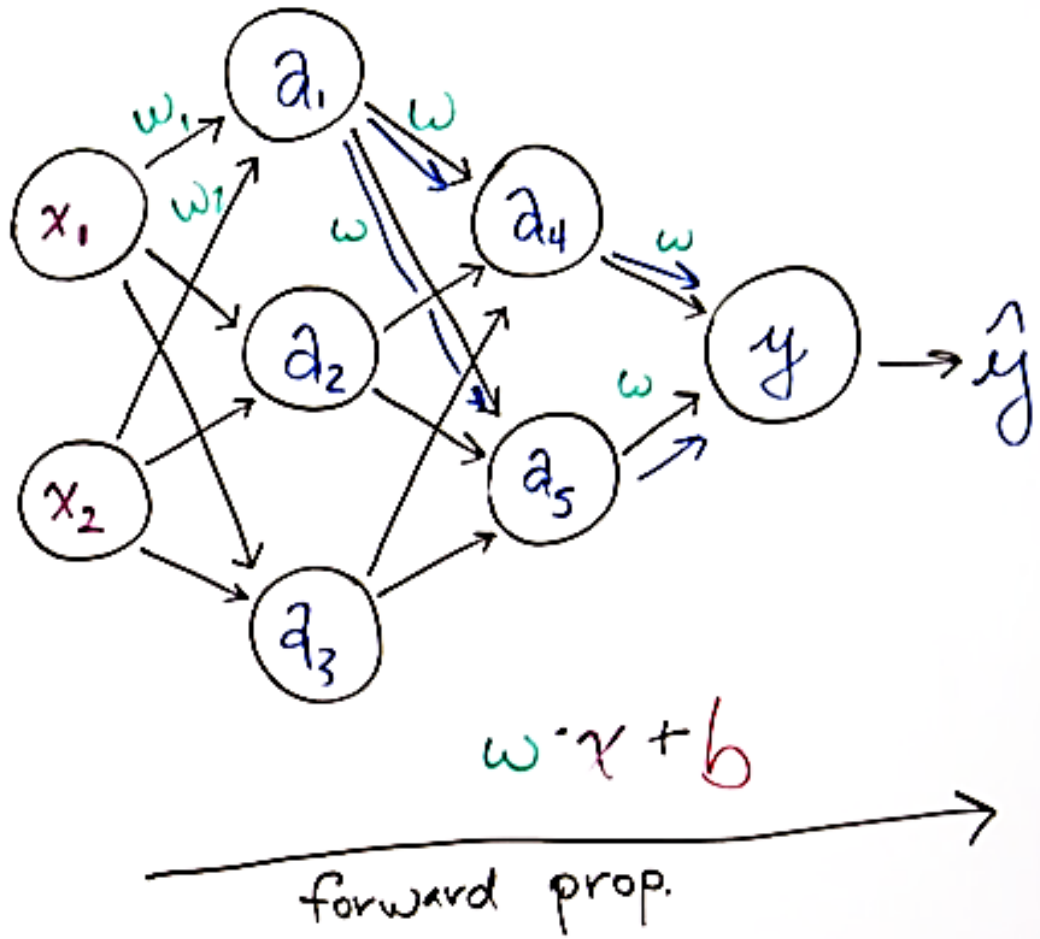

3·0=0
2·0=0
6·1=6
6>4
ketchup?
0
3
2
mustard?
0
4
hot dog?
1
6
bun?
1

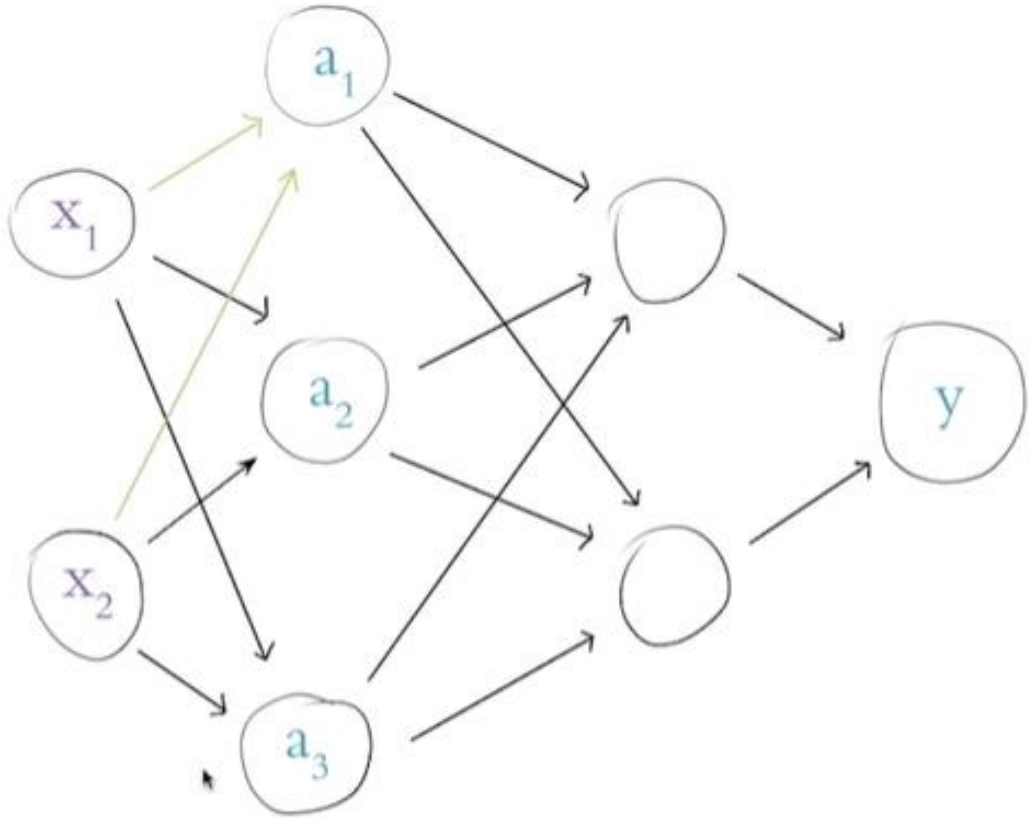

a_1
x_1
a_2
y
x_2
a_3
layer 1 2 3 4
hidden layer 1 2
forward propagation


a_1
w_1
w_2
x_1
x_2
y


a_1
w_1
w_2
w
w
x_1
x_2
y
$w \cdot x + b$


a_1
w_1
w
x_1
w_2
w
a_4
w
a_2
y
w
x_2
a_5
a_3
$w \cdot x + b$


a_1
w_1
w
x_1
w_2
w
a_4
w
a_2
y
$\hat{y}$
w
x_2
a_5
a_3
$w \cdot x + b$
forward prop.

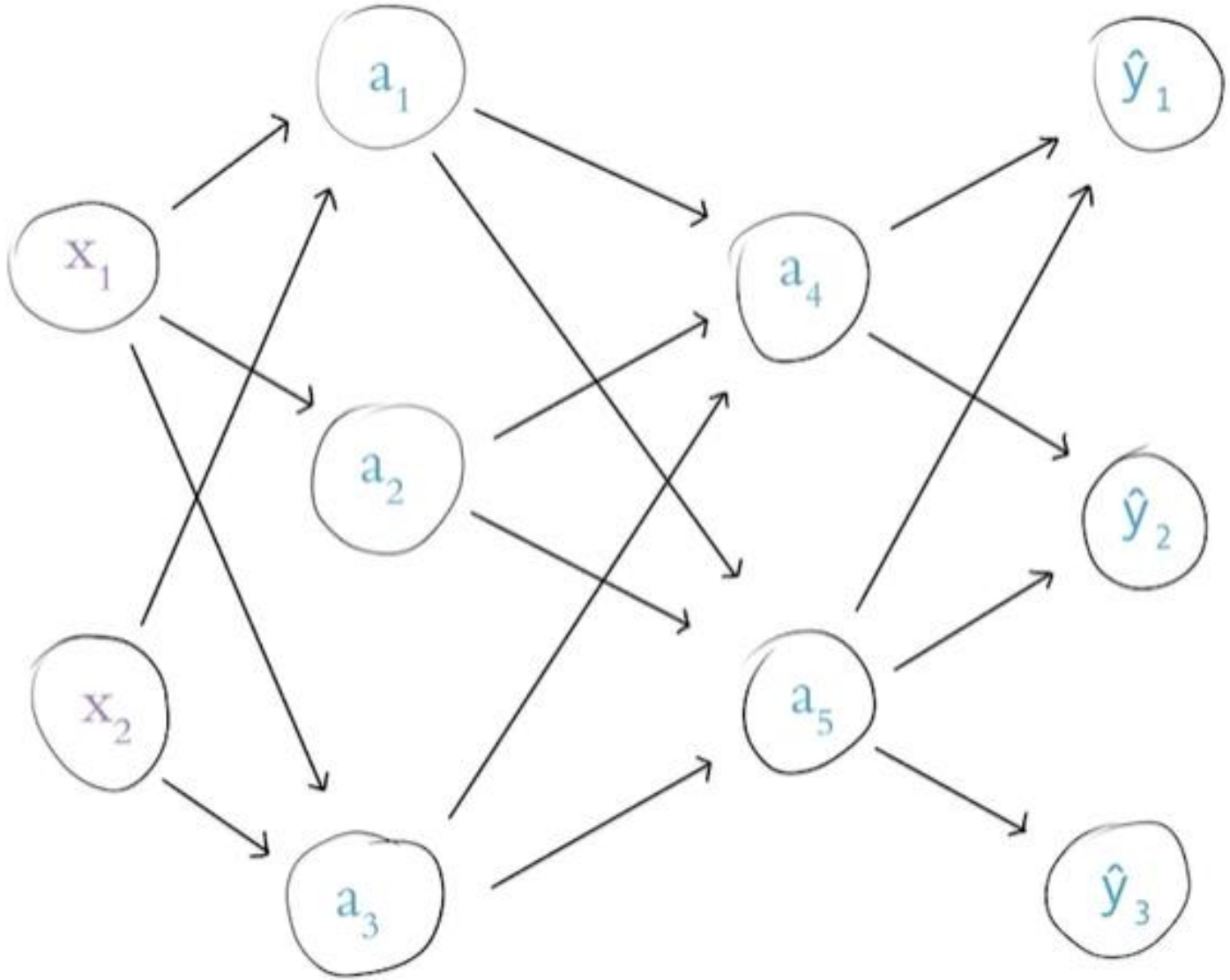

a_1
x_1
a_2
y
x_2
a_3
layer 1 2 3 4
hidden layer 1 2
forward propagation

Design neural network architecture

In [17]:
```
model = Sequential()
model.add(Dense(64, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
```

In [18]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```

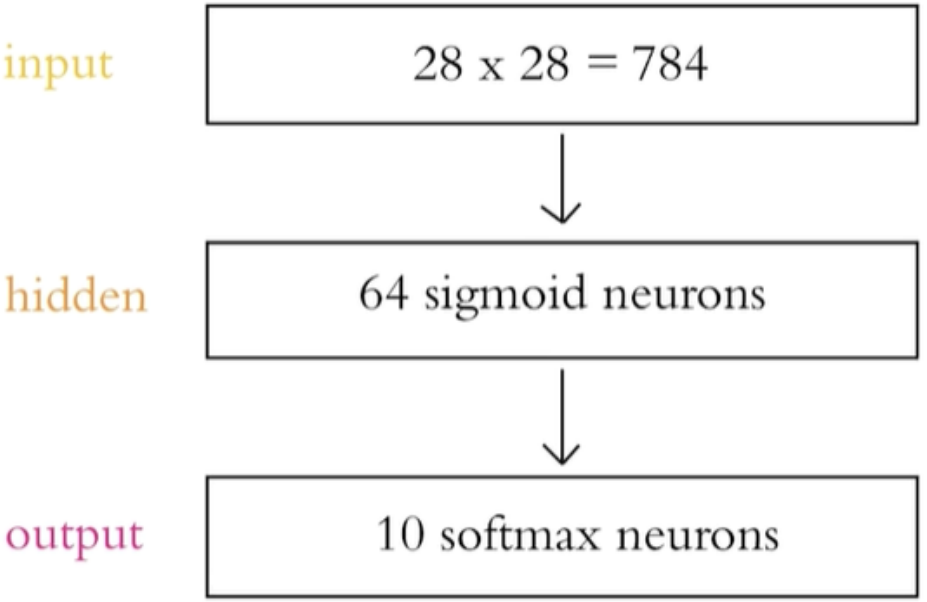

a_1
$\hat{y}_1$
x_1
a_4
a_2
$\hat{y}_2$
x_2
a_5
a_3
$\hat{y}_3$

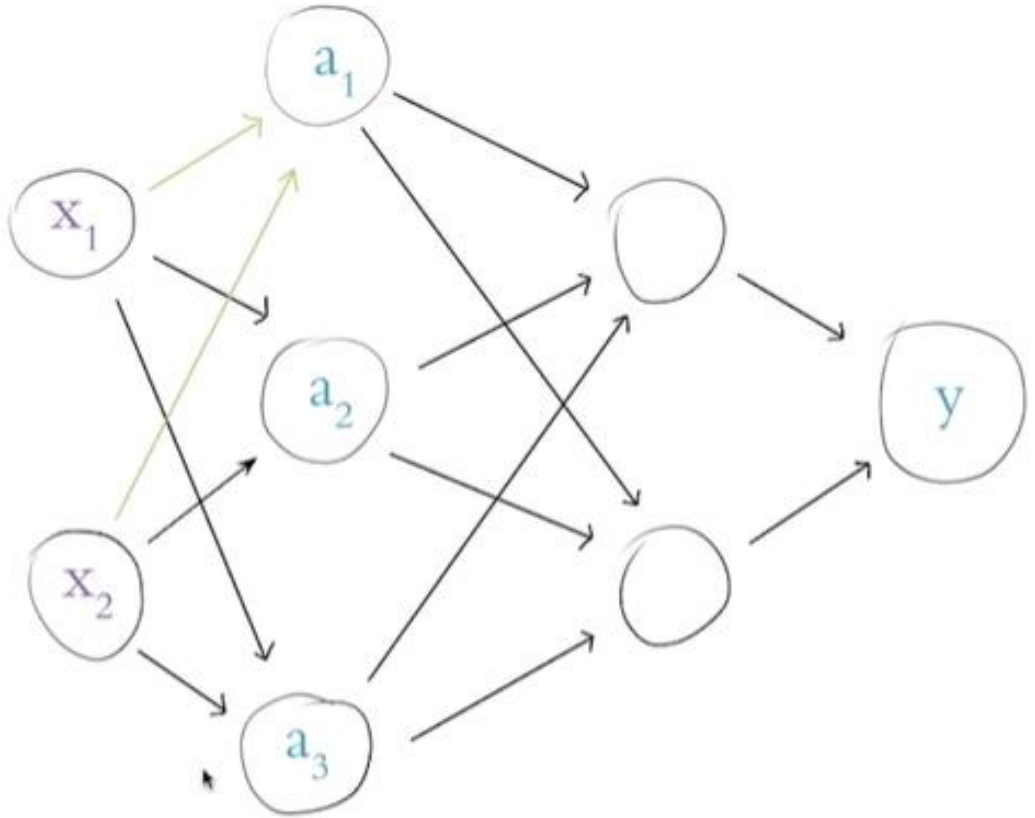
Design neural network architecture

In [17]:
```
model = Sequential()
model.add(Dense(64, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
```

In [18]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```



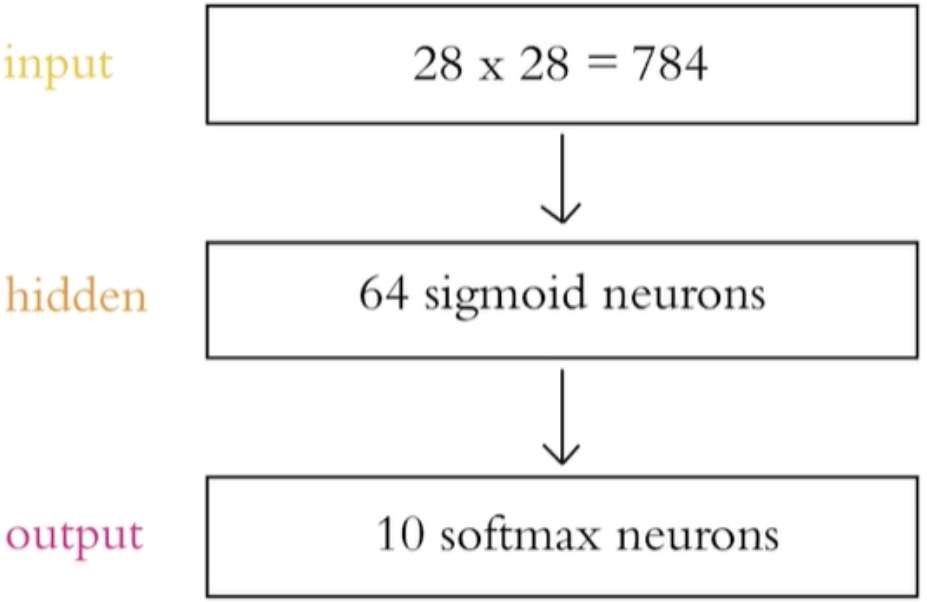

a_1
x_1
a_2
y
x_2
a_3
layer 1 2 3 4
hidden layer 1 2
forward propagation

Design neural network architecture

In [17]:
```
model = Sequential()
model.add(Dense(64, activation='sigmoid', input_shape=(784,)))
model.add(Dense(10, activation='softmax'))
```

In [18]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                50240
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 50,890
Trainable params: 50,890
Non-trainable params: 0
_________________________________________________________________
```

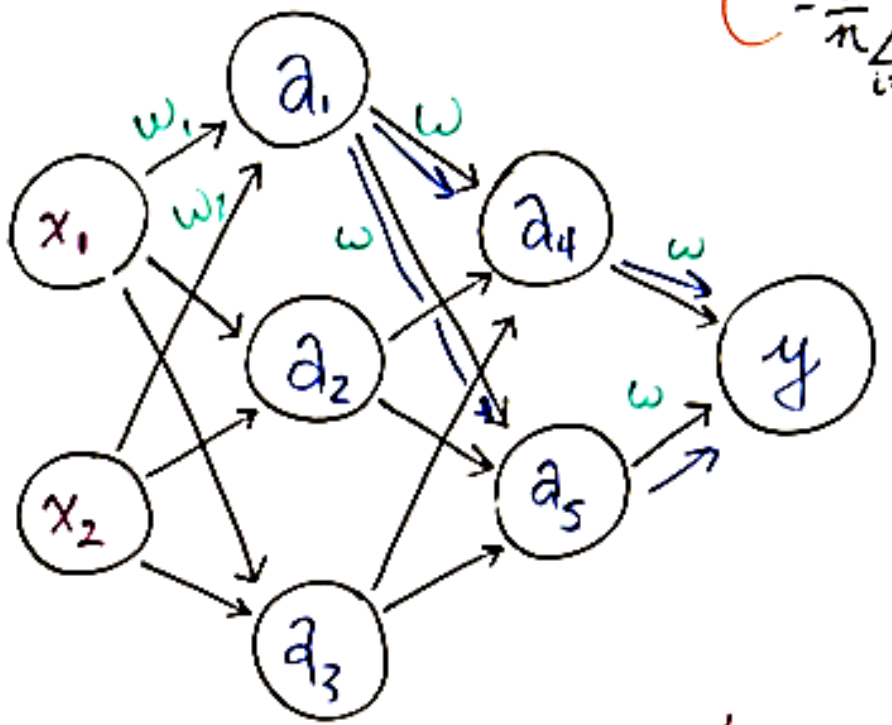


Key Concepts

- parameters:
  - weight w
  - bias b
- activation a
- neurons:
  - sigmoid
  - tanh
  - ReLU
- input layer
- hidden layer
- output layer
- forward prop.

- layer types:
  - dense/FC
  - softmax

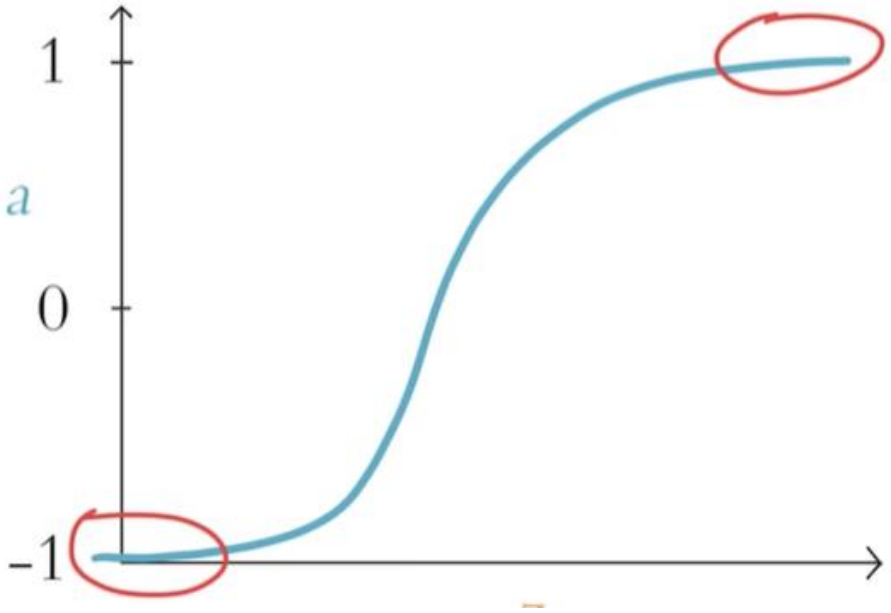
$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

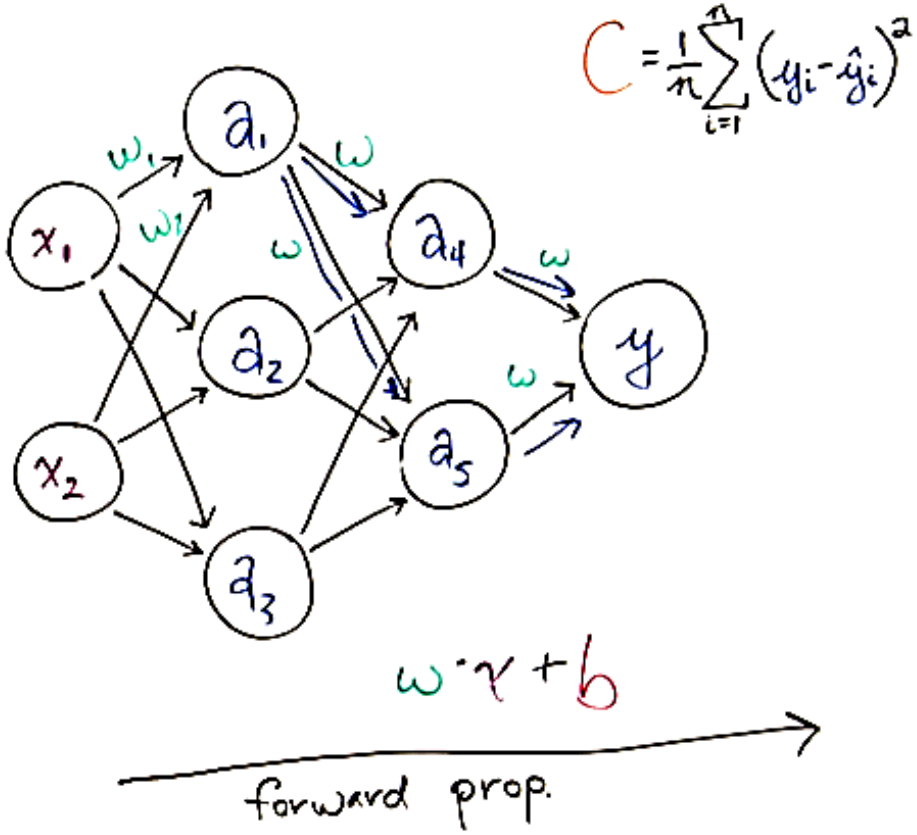


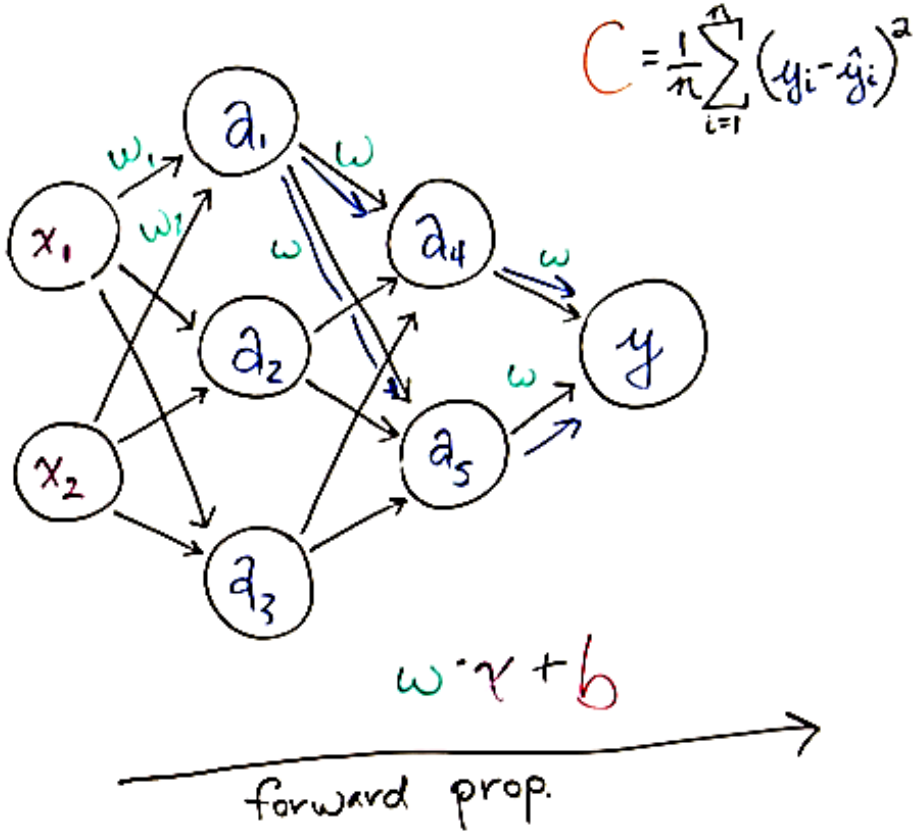
$$w \cdot x + b$$

forward prop.

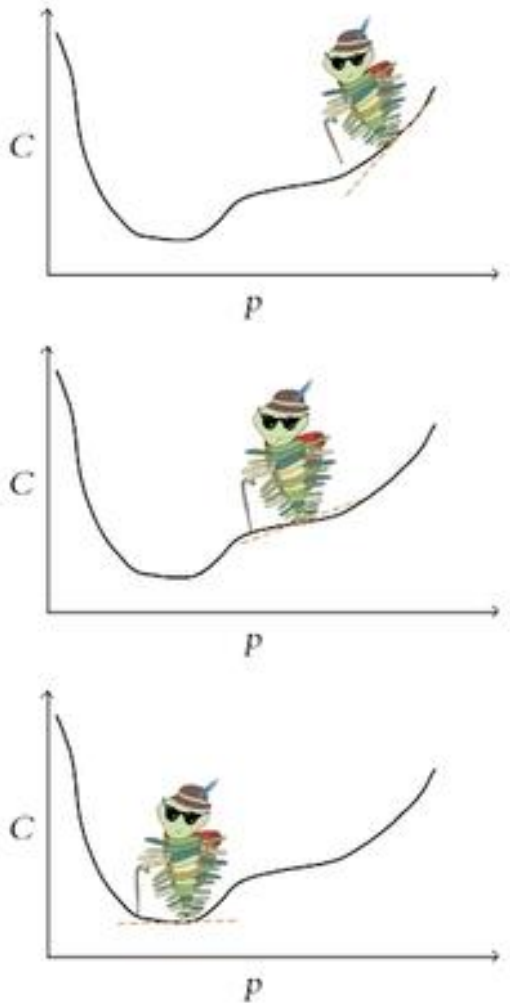
Neuron Saturation

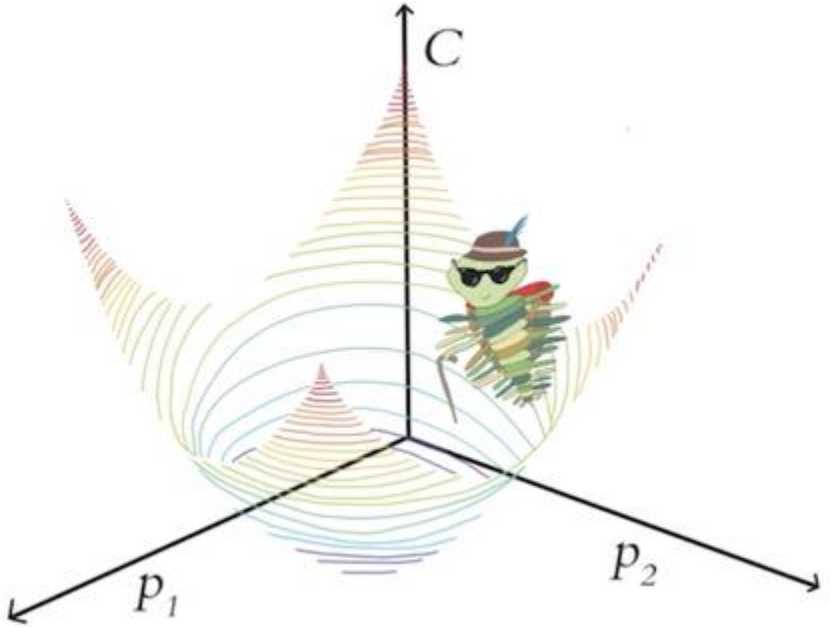




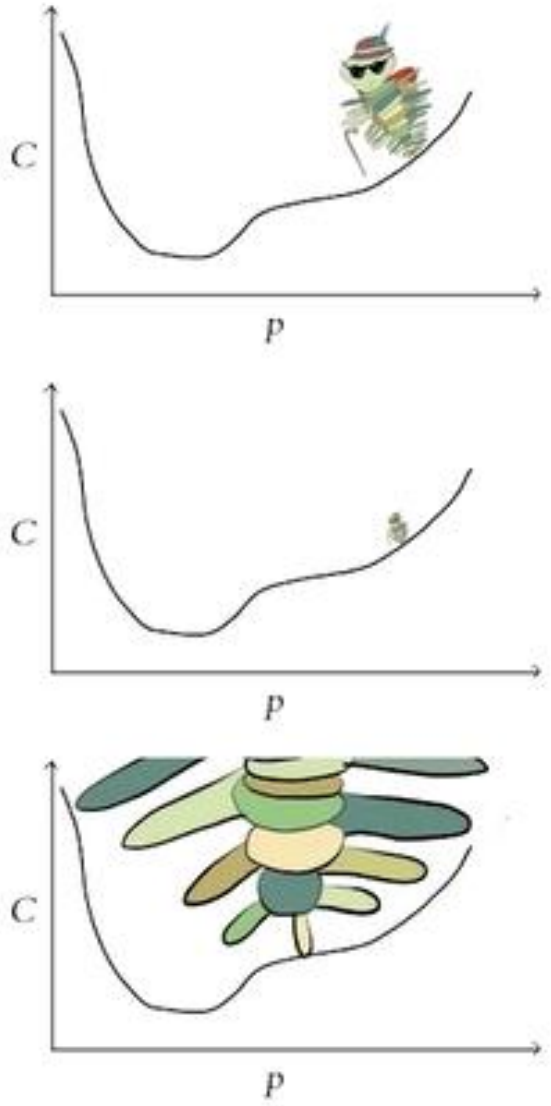

$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
x_1
x_2
a_1
a_2
a_3
a_4
a_5
y
w_1
w_2
w
$w \cdot x + b$
forward prop.

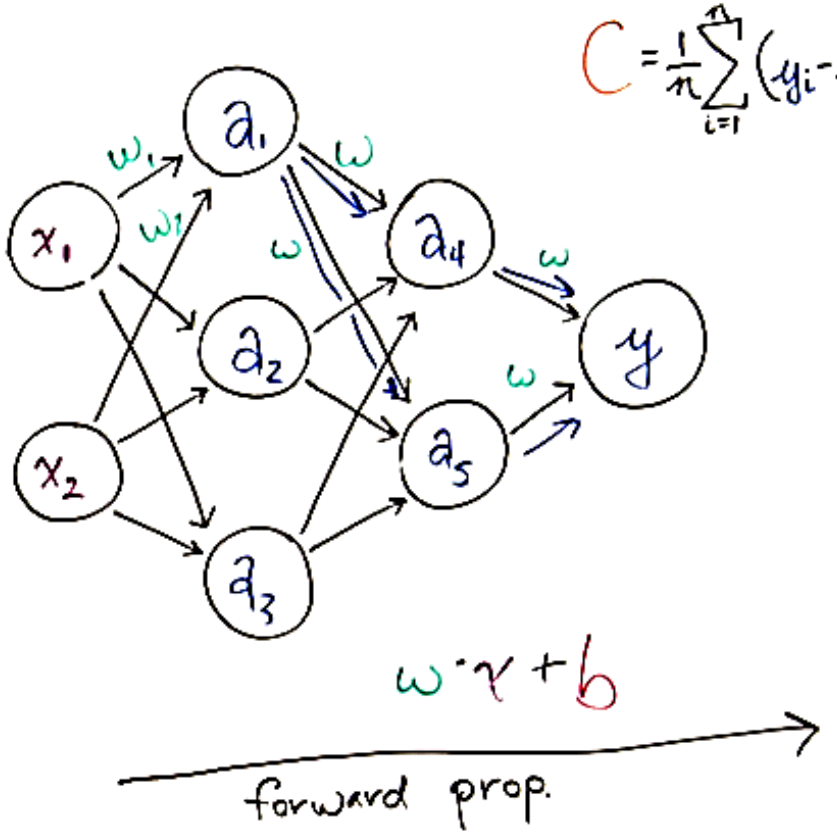
Gradient Descent



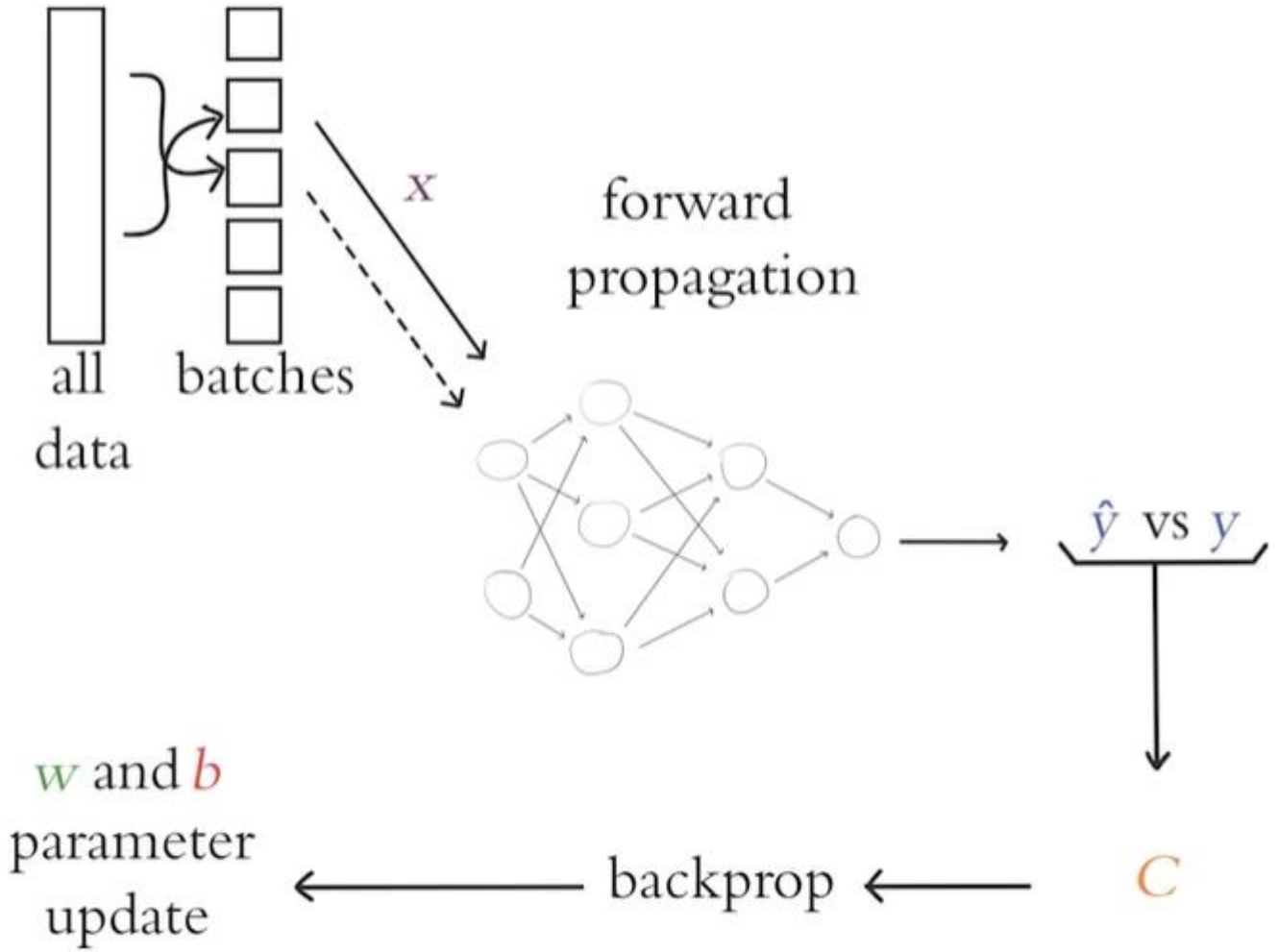


Learning Rate





Stochastic Gradient Descent



Configure model

```
In [22]: model.compile(loss='mean_squared_error', optimizer=SGD(lr=0.01), metrics=['accuracy'])
```

Train!

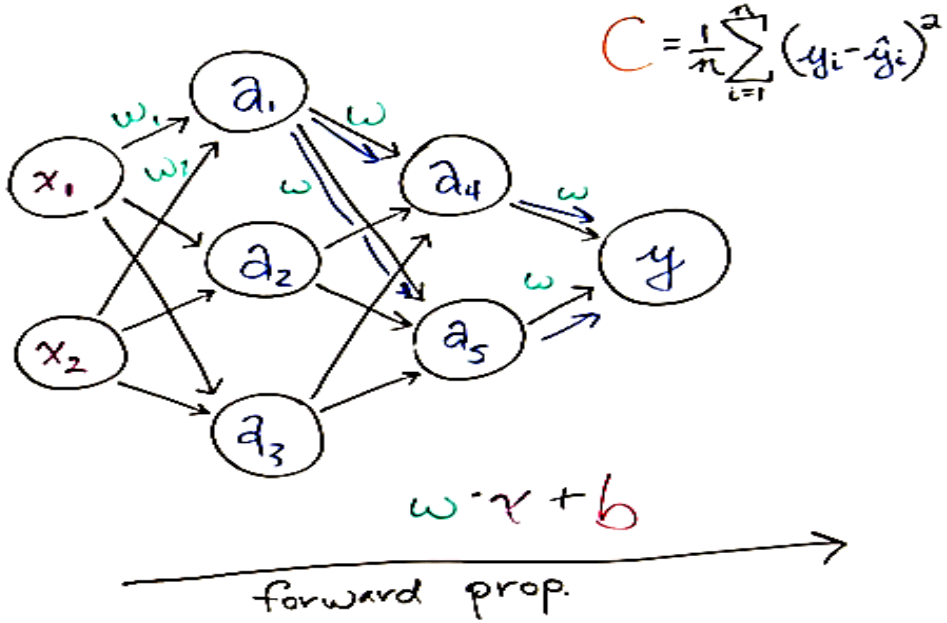
```
In [23]: model.fit(X_train, y_train, batch_size=128, epochs=200, verbose=1, validation_data=(X_valid, y_valid))
```

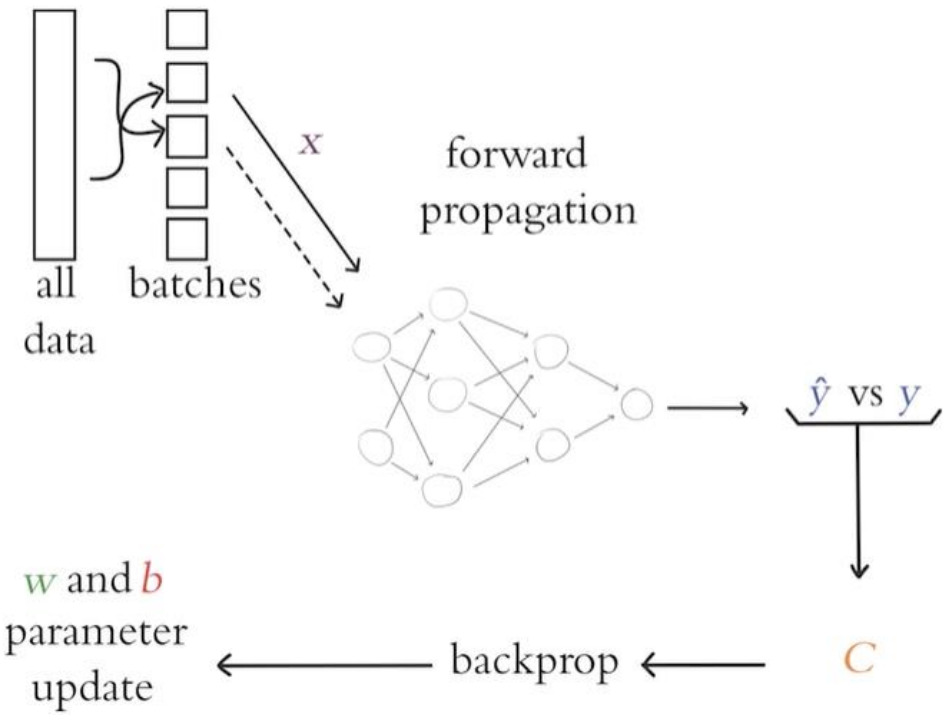
```
Epoch 1/200
469/469 [==============================] - 14s 15ms/step - loss: 0.0921 - accuracy: 0.1042 - val_loss: 0.0913 - val_accur
acy: 0.0996
Epoch 2/200
469/469 [==============================] - 5s 11ms/step - loss: 0.0911 - accuracy: 0.1008 - val_loss: 0.0906 - val_accura
cy: 0.0990
Epoch 3/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0905 - accuracy: 0.1018 - val_loss: 0.0901 - val_accura
cy: 0.1043
Epoch 4/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0900 - accuracy: 0.1094 - val_loss: 0.0897 - val_accura
cy: 0.1204
Epoch 5/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0896 - accuracy: 0.1248 - val_loss: 0.0893 - val_accura
cy: 0.1433
Epoch 6/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0892 - accuracy: 0.1473 - val_loss: 0.0890 - val_accura
cy: 0.1728
Epoch 7/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0889 - accuracy: 0.1682 - val_loss: 0.0887 - val_accura
cy: 0.1933
```

60000/128 = 468.75

Round of Training:

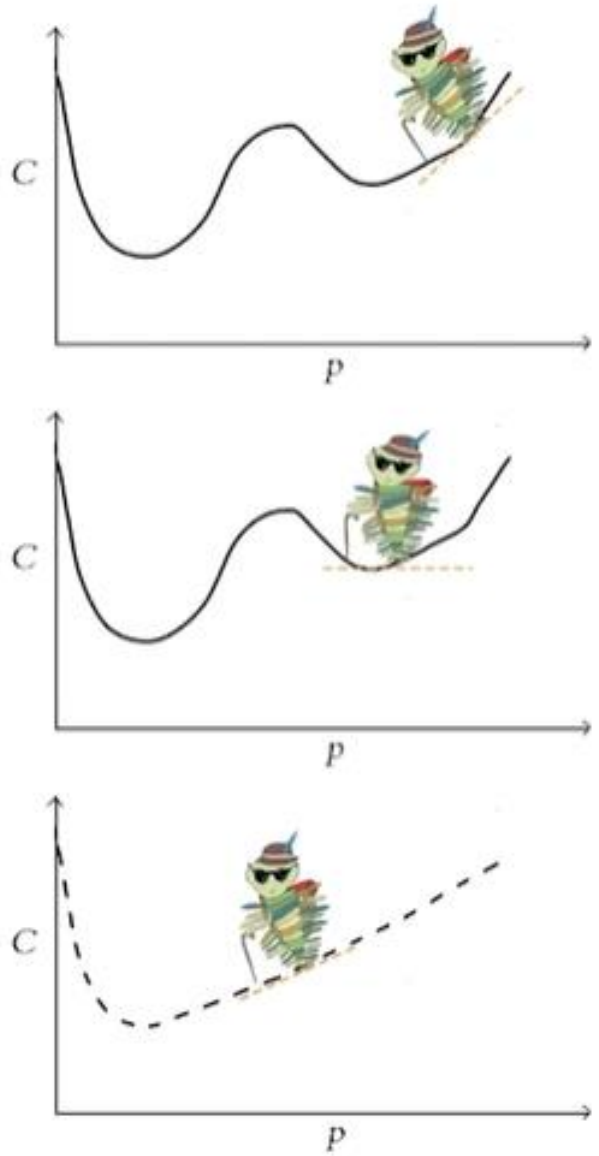
1. Sample a mini-batch of x values
2. Forward propagate x through network to estimate y with $\hat{y}$
3. Calculate cost C by comparing y and $\hat{y}$
4. Descend gradient of C to adjust w and b, enabling x to better predict y

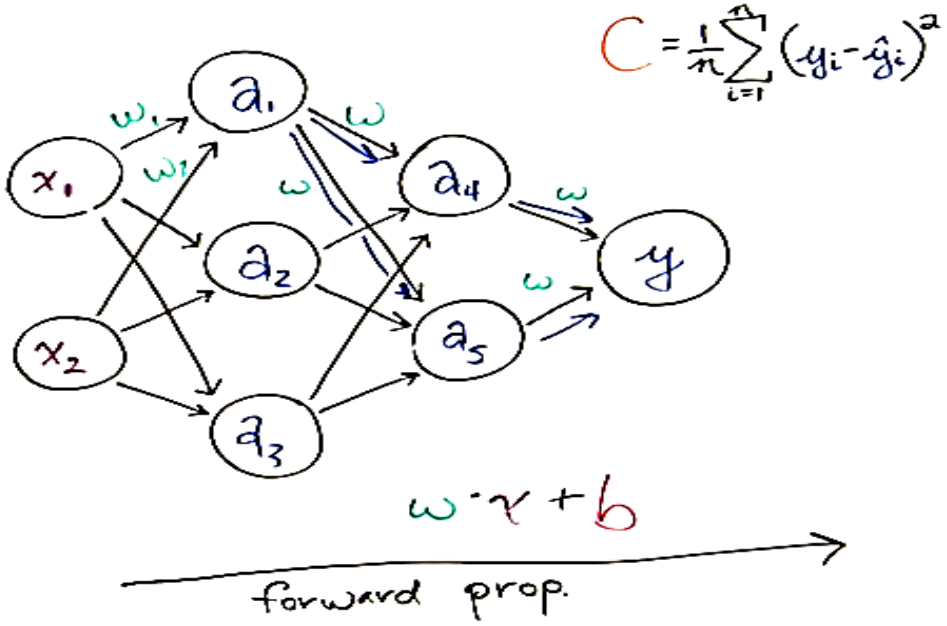




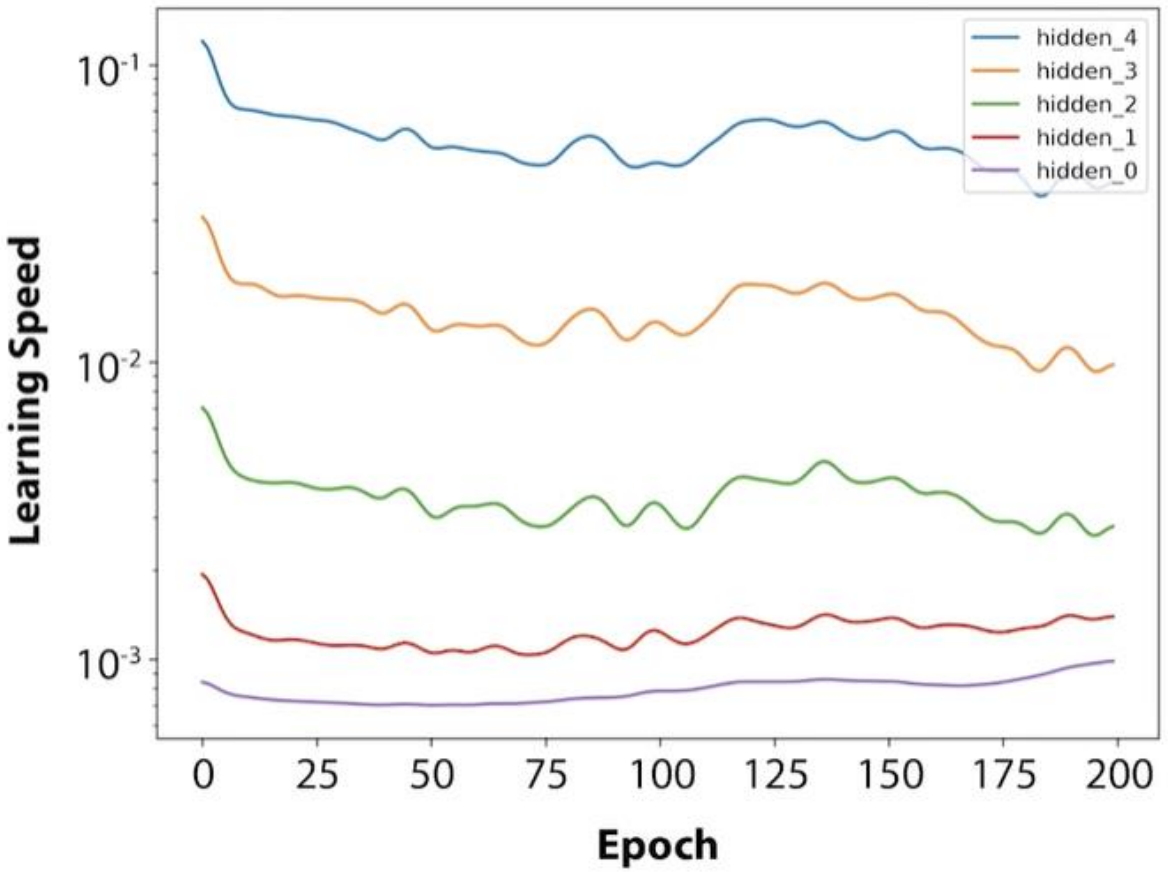
Escaping the local minimum of cost

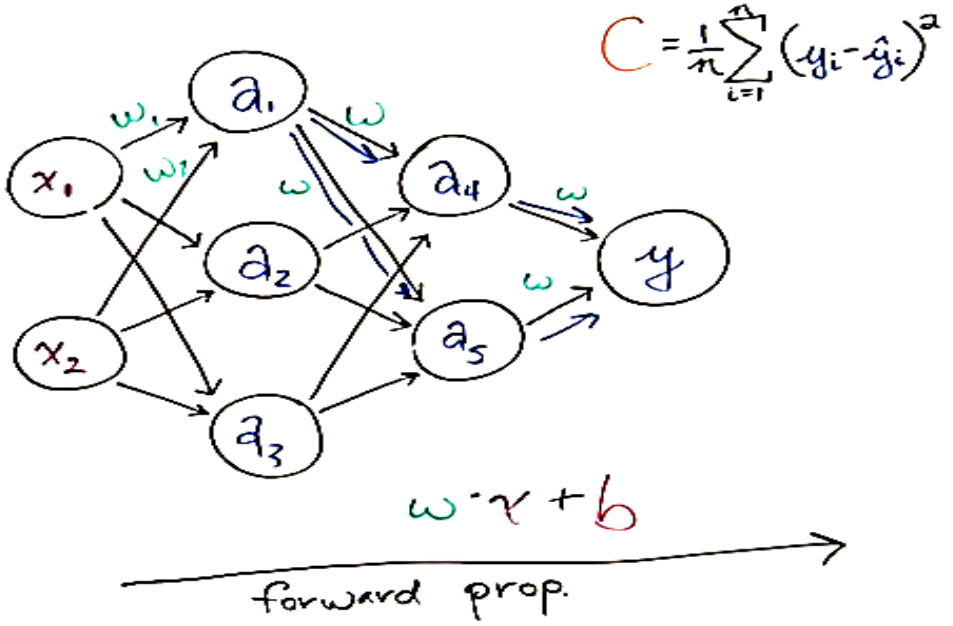
To avoid getting trapped in local minima, a batch size of larger than 128 is not recommended.

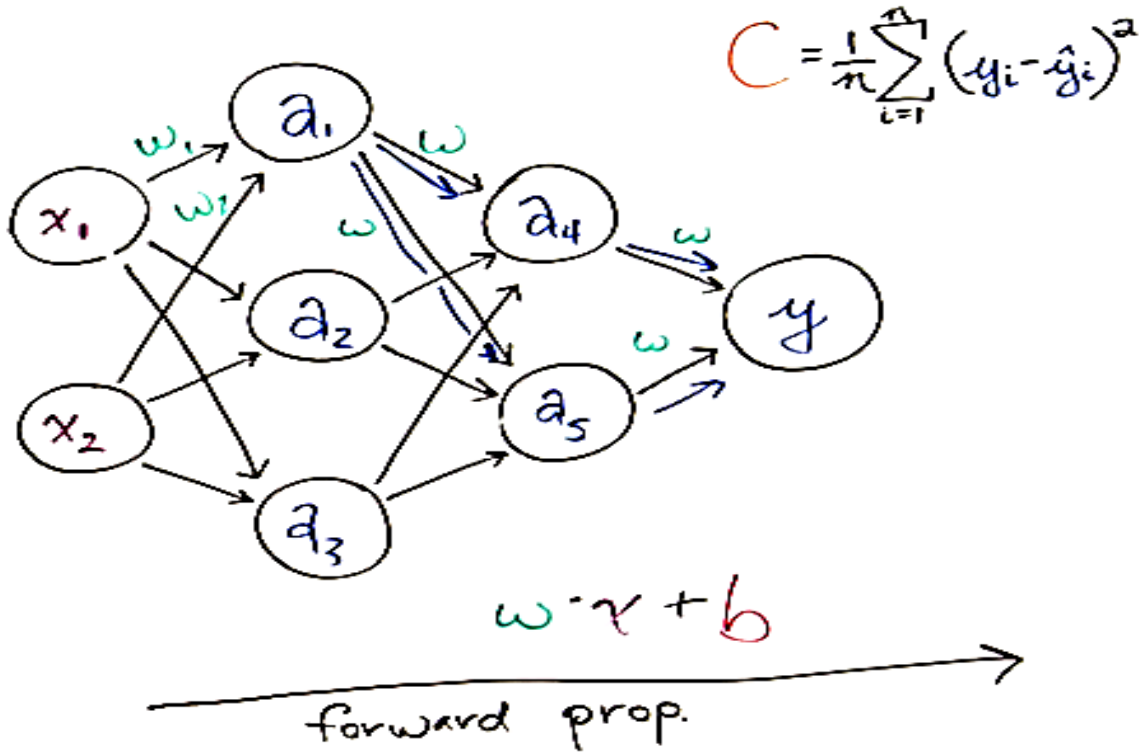




Learning Speed by layer





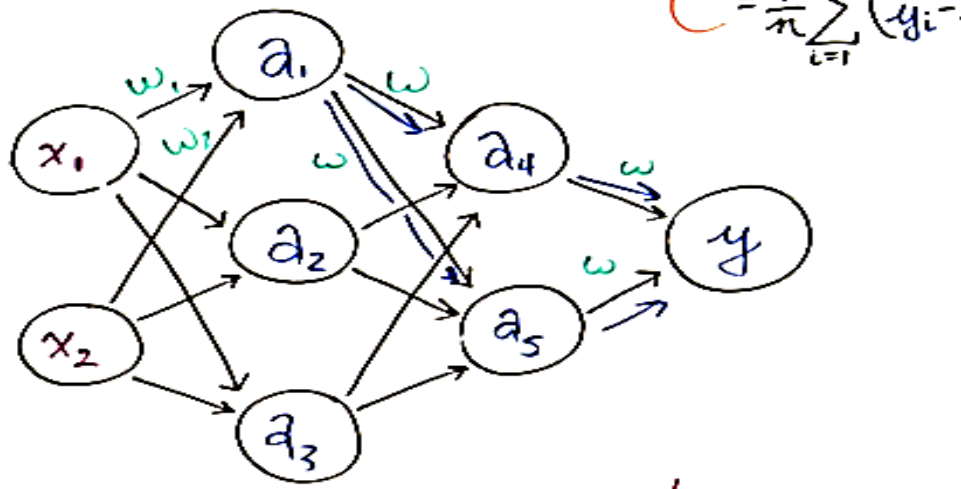

$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
a_1
a_2
a_3
a_4
a_5
x_1
x_2
y
w_1
w_2
w
$w \cdot x + b$
forward prop.

Key Concepts

- parameters:
  - weight w
  - bias b
- activation a
- neurons:
  - sigmoid
  - tanh
  - ReLU
- input layer
- hidden layer
- output layer
- forward/backprop.

- layer types:
  - dense/FC
  - softmax
- cost function C
  - quadratic
  - cross-entropy
- optimizers:
  - SGD
- optimizer hyperparams:
  - learning rate η
  - batch size

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$



Train!

```
In [23]: model.fit(X_train, y_train, batch_size=128, epochs=200, verbose=1, validation_data=(X_valid, y_valid))
```

```
Epoch 1/200
469/469 [==============================] - 14s 15ms/step - loss: 0.0921 - accuracy: 0.1042 - val_loss: 0.0913 - val_accur
acy: 0.0996
Epoch 2/200
469/469 [==============================] - 5s 11ms/step - loss: 0.0911 - accuracy: 0.1008 - val_loss: 0.0906 - val_accura
cy: 0.0990
Epoch 3/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0905 - accuracy: 0.1018 - val_loss: 0.0901 - val_accura
cy: 0.1043
Epoch 4/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0900 - accuracy: 0.1094 - val_loss: 0.0897 - val_accura
cy: 0.1204
Epoch 5/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0896 - accuracy: 0.1248 - val_loss: 0.0893 - val_accura
cy: 0.1433
Epoch 6/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0892 - accuracy: 0.1473 - val_loss: 0.0890 - val_accura
cy: 0.1728
Epoch 7/200
469/469 [==============================] - 6s 12ms/step - loss: 0.0889 - accuracy: 0.1682 - val_loss: 0.0887 - val_accura
```

Key Concepts

- parameters:
  - weight w
  - bias b
- activation a
- neurons:
  - sigmoid
  - tanh
  - ReLU
- input layer
- hidden layer
- output layer
- forward/backprop.

- layer types:
  - dense/FC
  - softmax
- cost function C
  - quadratic
  - cross-entropy
- optimizers:
  - SGD
- optimizer hyperparams:
  - learning rate η
  - batch size