NEURAL NETWORKS

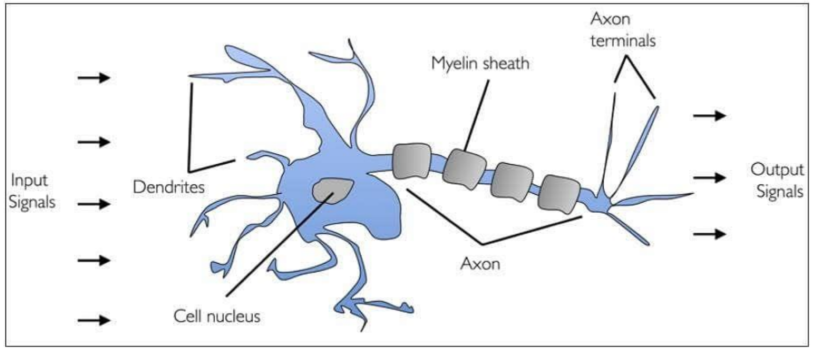
DR. FARHAD RAZAVI

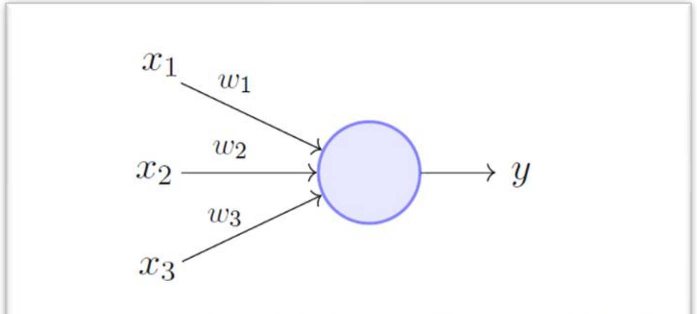
OUTLINE

- Artificial Neural Networks
  - History
  - Biological Neural Networks
  - Perceptron
  - Multi-layered Perceptron
  - Feedforward Propagation
- TensorFlow implementation

HISTORY OF ARTIFICIAL NEURAL NETWORKS

- **1958:** Frank Rosenblatt introduced the idea of perceptron (a form of neural network).
- **1969:** Minsky's book "Perceptron", proved a one-layer perceptron cannot mimic the XOR logic.
  - Many contributes first AI winter to this book!

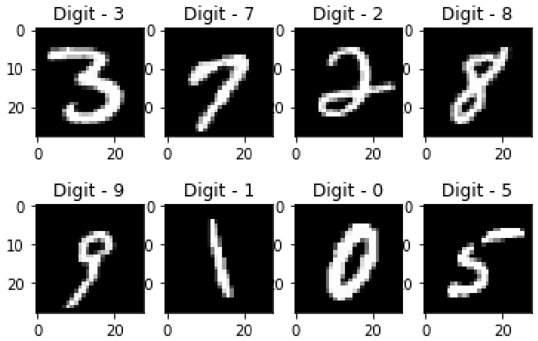




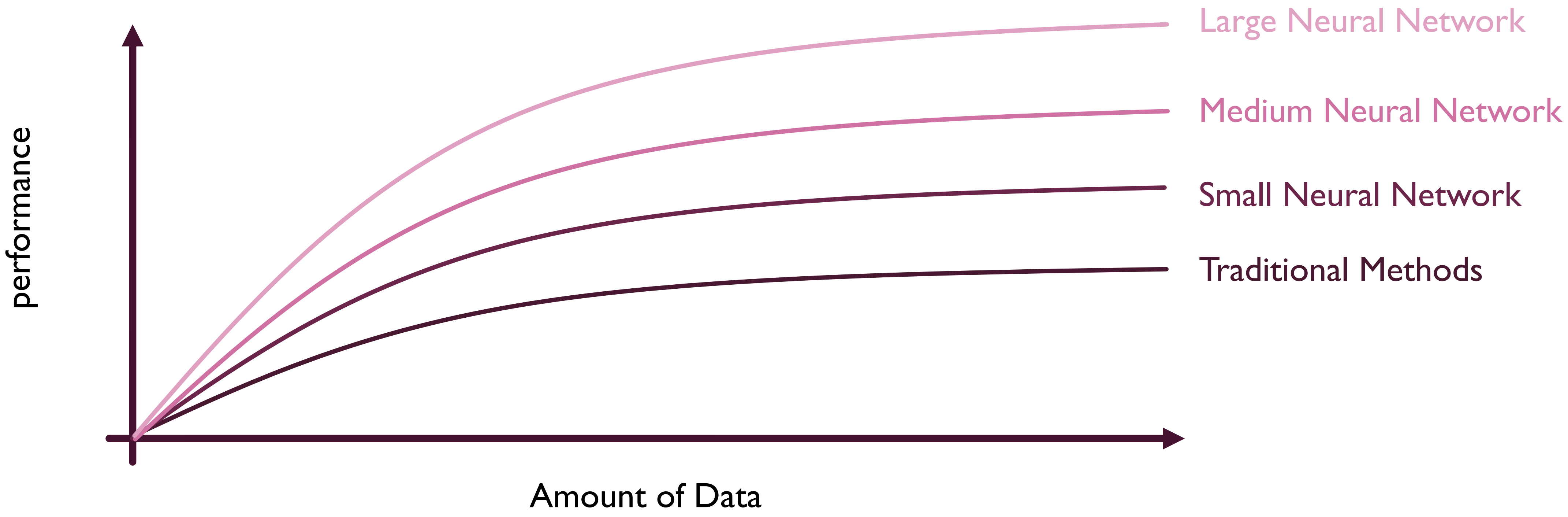
Perceptron Model (Minsky-Papert in 1969)

HISTORY OF ARTIFICIAL NEURAL NETWORKS

- **1975:** Werbos effectively solve the XOR problem by implementing the backpropagation algorithm.
- **1980-1990:** Neural networks got traction again after showing promising results in handwritten digits recognition.
- **2000:** Significant change toward data collection rather than algorithm development.
- **2005:** Neural networks successfully was implemented in speech recognition.
- **2006:** AI researcher Fei-Fei Li (Stanford) began working on the idea for ImageNet in 2006 (14 millions images).
- **2012:** AlexNet achieved 15.3% error.
- Speech → Images → text (NLP) → Climate change, medical imaging, online advertisement, …

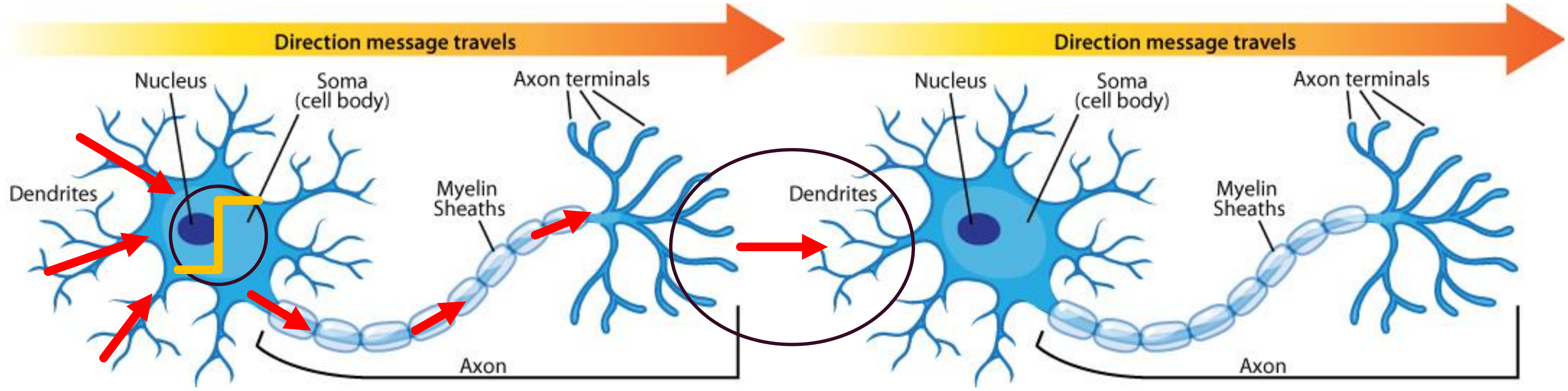


WHY THE HYPE ABOUT NEURON NETWORK



- Rise of big data.
- Rise of computational powers (GPUs).

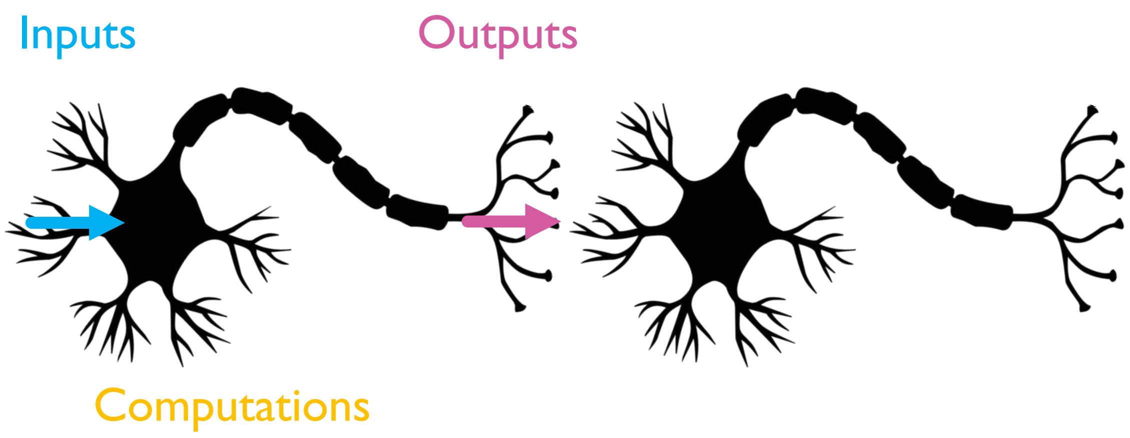
ANATOMY OF A BIOLOGICAL NEURON



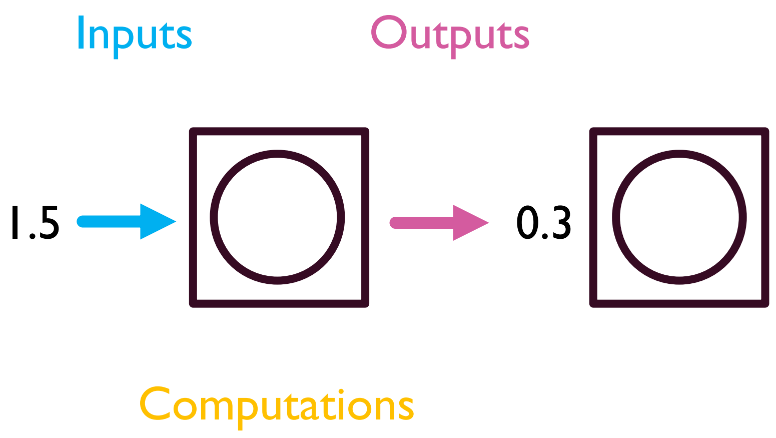
- Dendrites acts as inputs to neurons. Each neuron can have multiple inputs.
- Inside the cell some computation will happen (mostly chemically).
- Neuron will then send an electrical impulse depending passing a threshold or not to another neorons.

ARTIFICIAL NEURON INSPIRATION

Biological Model of Neuron



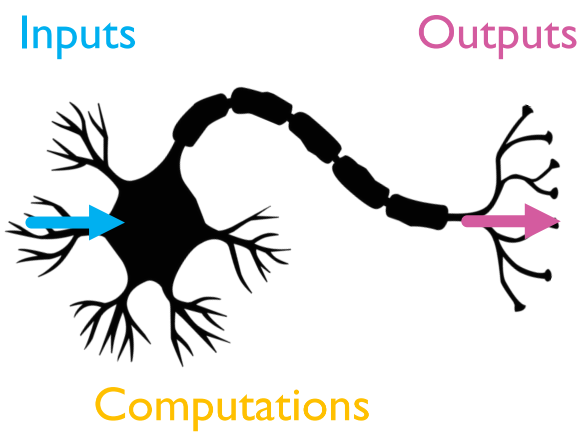
Simplified Mathematical Model



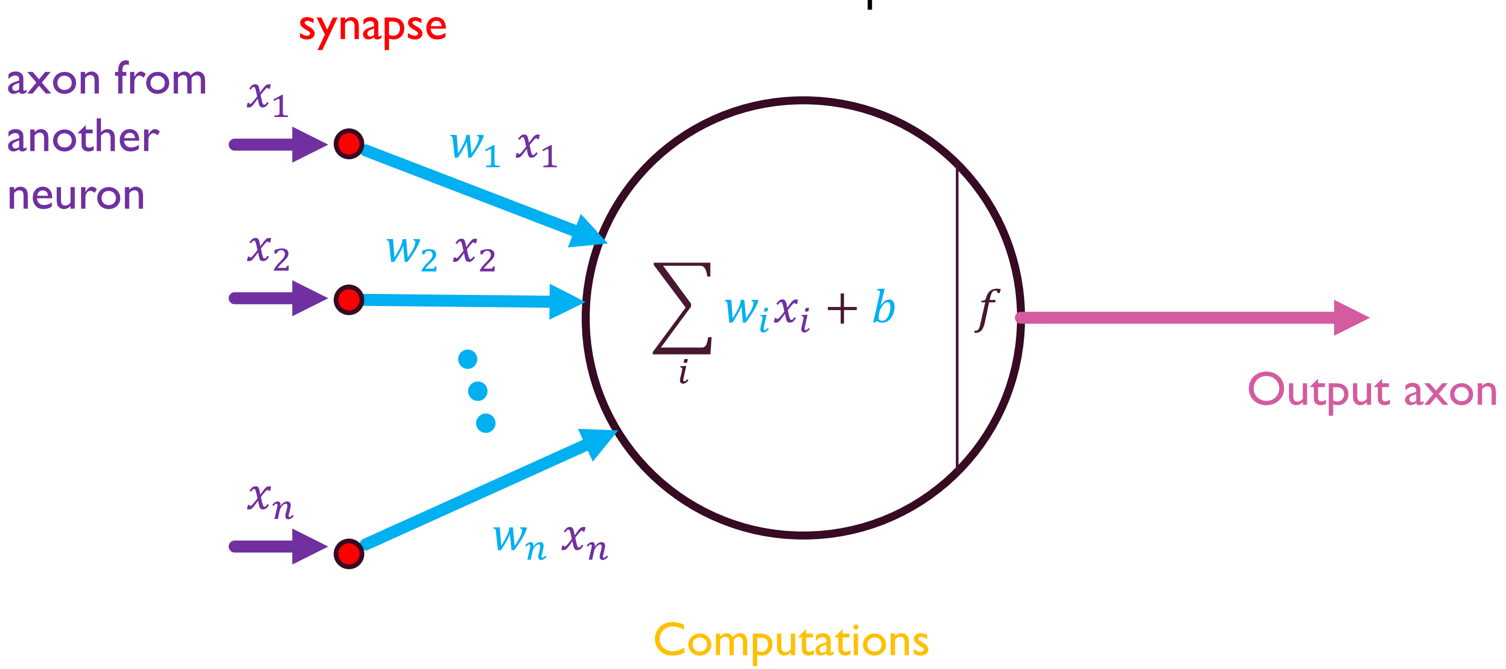
- The research is no longer focused to mimic the biological neurons.
- In the simplified mathematical model gets some input numbers and carry outs some computation and then outputs some number.
- Even with this simplified model of neuron we can still do power computation.

ARTIFICIAL NEURON INSPIRATION-SIMPLE PERCEPTRON

Biological Model of Neuron

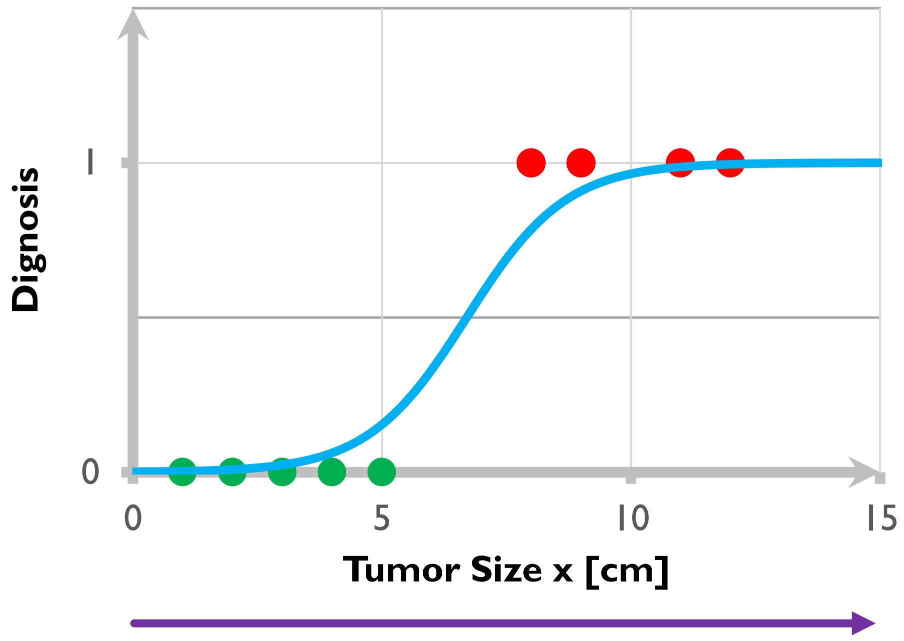


Simplified Mathematical Model



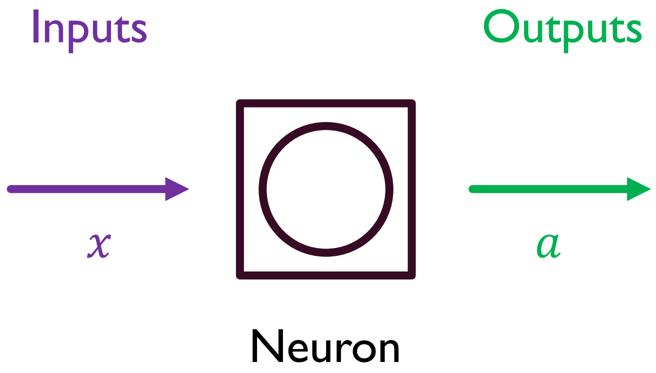
- We are going to adapt the simplified model of neuron to our previously studied models.

TUMOR ANALYSIS (LOGISTIC REGRESSION)



$$a = f_{w,b}(x) = \frac{1}{1 + e^{-(wx+b)}}$$

- Remember the sigmoid function for logistic regression.
- We will switch the terminology to apply the logistic function as an activation function a.



- We can define a very simple model of the neuron in which a unit neuron accepts an input x and generates an output a which is the probability of a tumor being malignant.

SIMPLE YET POWERFUL

- Except for KNN, all the different models that we have studied so far can be reduced to this "simple" model!
- Linear Regression

$$\hat{y} = f(\vec{x}) = \vec{w} \, . \, \vec{x} + b$$

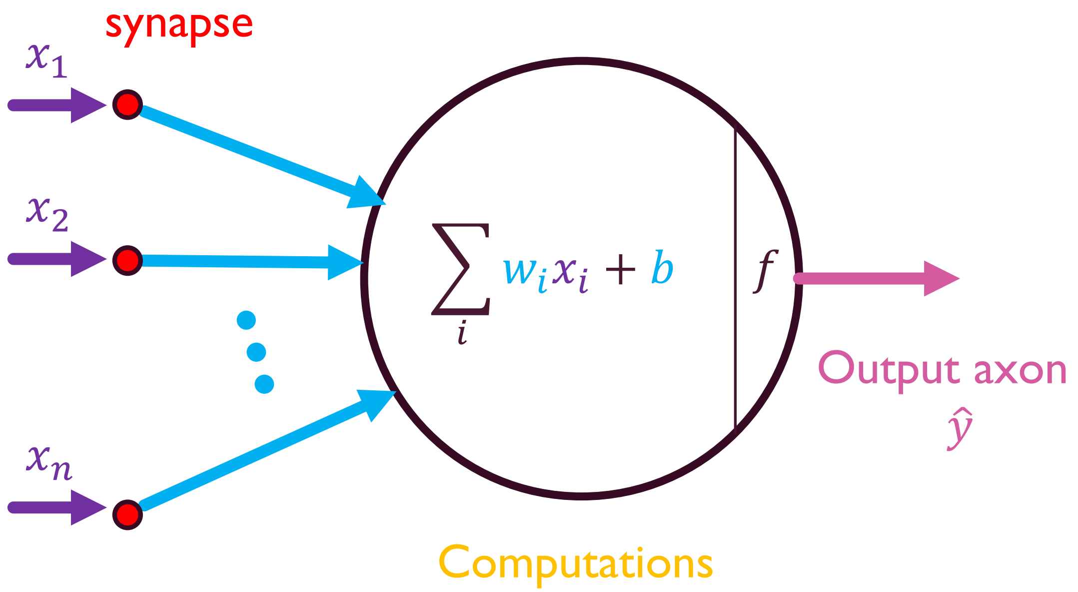
- Logistic Regression

$$\hat{y} = f(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \, . \, \vec{x} + b)}}$$

- Bayesian Classifier
  - Gaussian model

$$\hat{y} = f(\vec{x}) = \frac{1}{\sqrt{(2\pi)^2 |\boldsymbol{\Sigma}_K|}} e^{-\frac{1}{2}(\vec{x}-\vec{\mu}_K)^T \boldsymbol{\Sigma}_K{}^{-1}(\vec{x}-\vec{\mu}_K)}$$

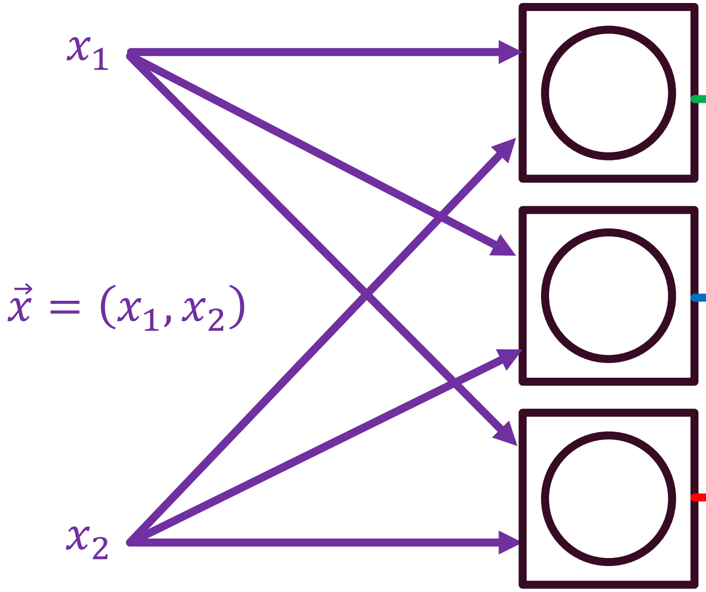
Simplified Mathematical Model



$$a(\vec{x}) = f(\vec{x}) = \text{activation function}$$

BUILDING MORE SOPHISTICATED MODELS

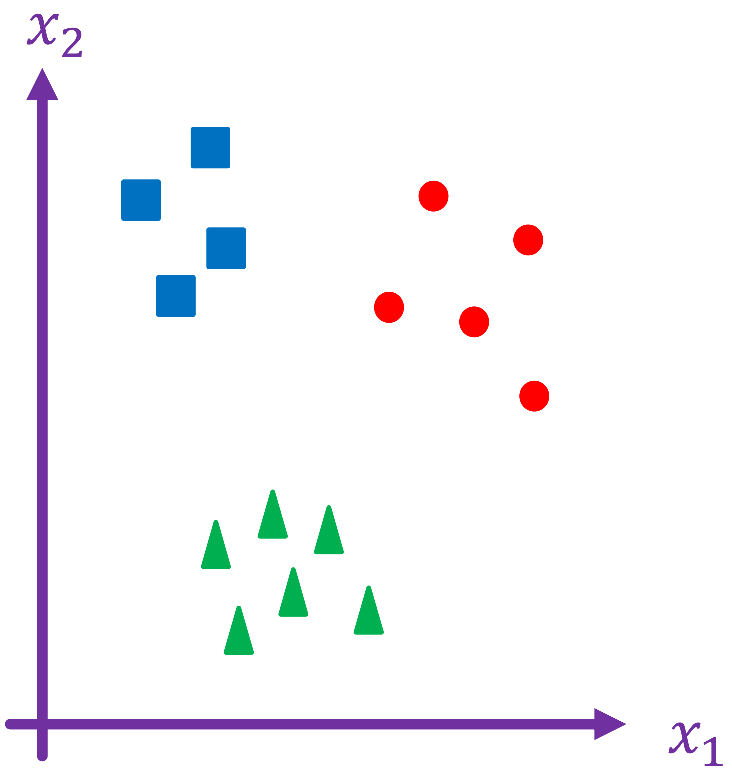
- Remember the multi-class logistic regression.
- One of the methods we used was one-vs-all



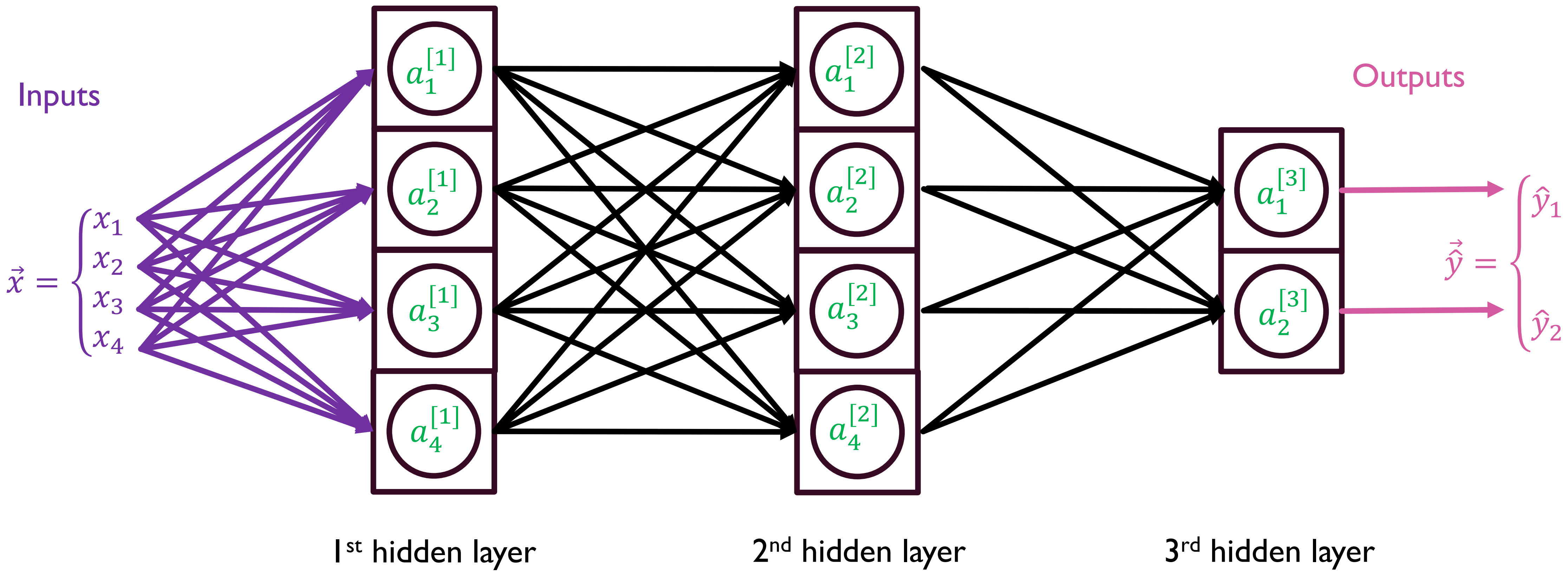
$$a_1 = \frac{1}{1 + e^{-(\vec{w}_1 \vec{x} + b)}}$$

$$a_2 = \frac{1}{1 + e^{-(\vec{w}_2 \vec{x} + b)}}$$

$$a_3 = \frac{1}{1 + e^{-(\vec{w}_3 \vec{x} + b)}}$$

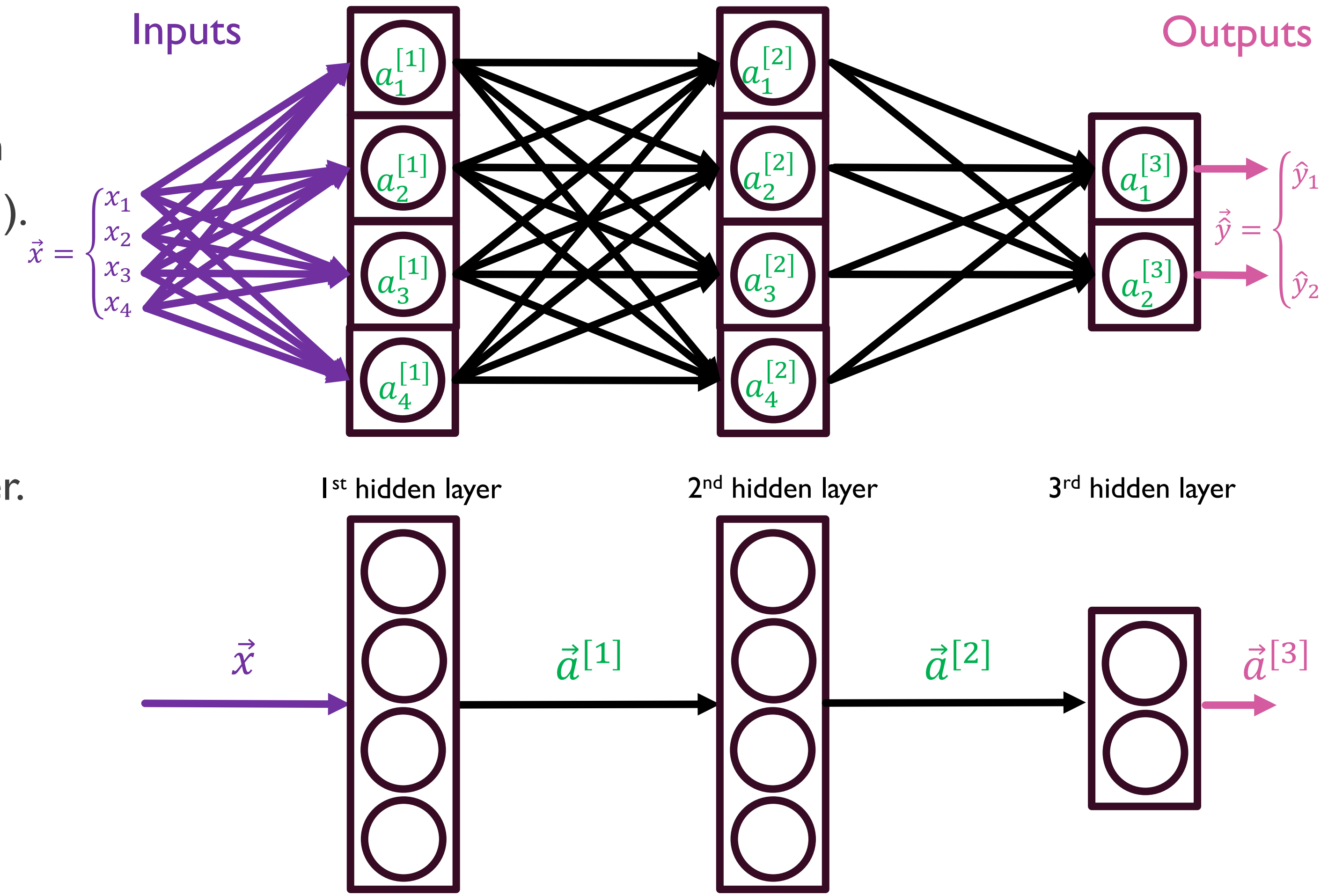


MULTI LAYERED PERCEPTRON (MLP)

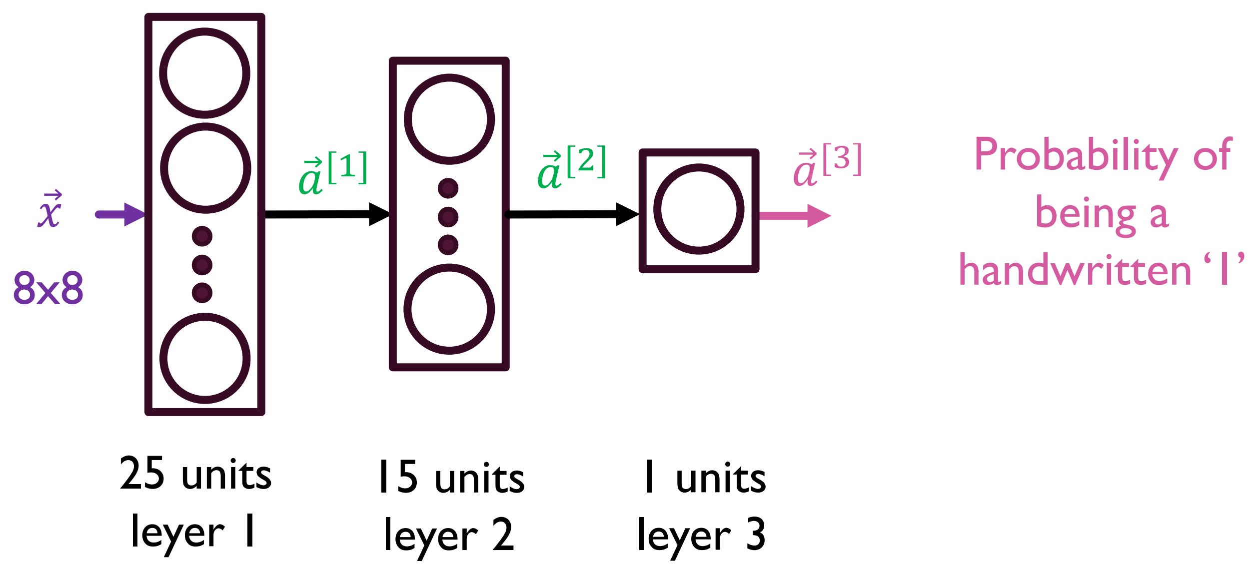


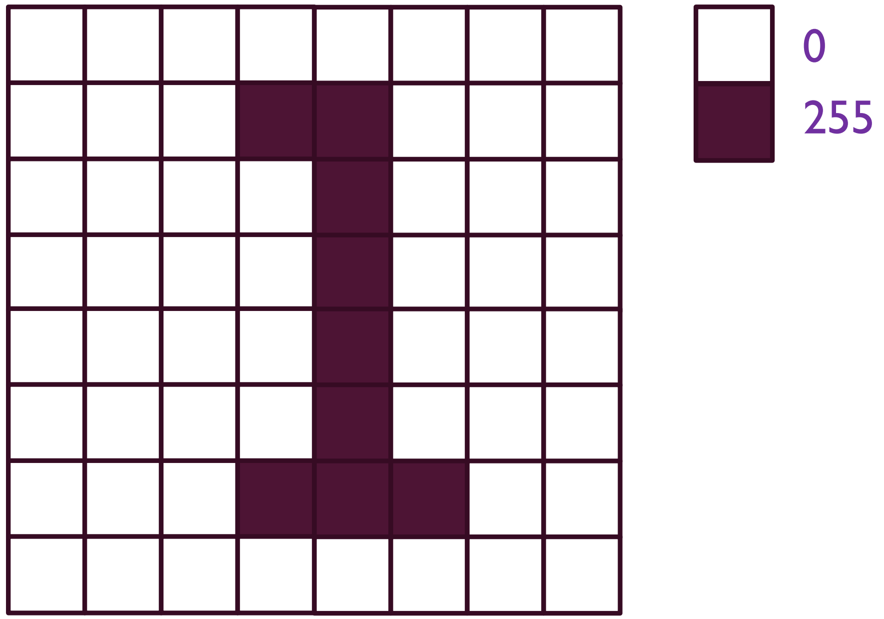
NOTATION

- $a_i^{[j]}$ is the activation function for the *ith* neuron of the *jth* hidden layer.
- Remember that each activation function has its own set of parameters $(\vec{w}_i^{[j]}, b_i^{[j]})$.
- Layer jth activation functions can be represented as a vector $\vec{a}^{[j]} = \left(a_1^{[j]}, a_2^{[j]}, \dots, a_m^{[j]}\right)$ with m representing the total number of neurons in that layer.
- $a_i^{[j]} = f\left(\vec{w}_i^{[j]} . \vec{a}^{[j-1]} + b_i^{[j]}\right)$, with f being the sigmoid function.
- The process of calculating $a_i^{[j]}$ at each layer from the previous layer is called Forward Propagation.



HANDWRITTEN DIGIT RECOGNITION



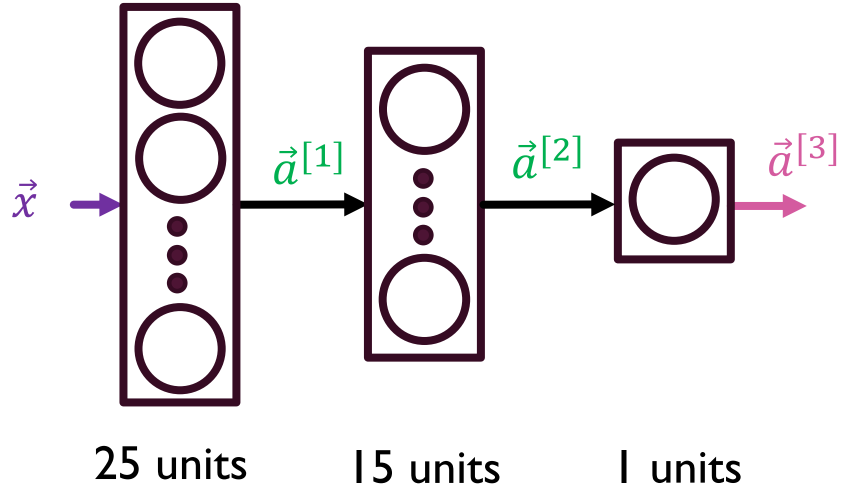


$$\vec{a}^{[1]} = \begin{cases} f\left(\vec{w}_1^{[1]}.\vec{x} + b_1^{[1]}\right) \\ \vdots \\ f\left(\vec{w}_{25}^{[1]}.\vec{x} + b_{25}^{[1]}\right) \end{cases}$$

$$\vec{a}^{[2]} = \begin{cases} f\left(\vec{w}_1^{[2]}.\vec{a}^{[1]} + b_1^{[2]}\right) \\ \vdots \\ f\left(\vec{w}_{15}^{[2]}.\vec{a}^{[1]} + b_{15}^{[2]}\right) \end{cases}$$

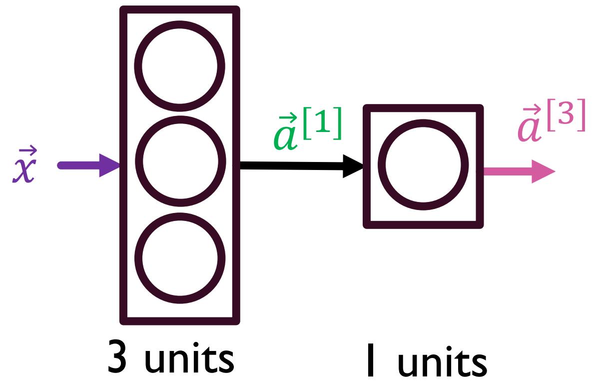
$$\vec{a}^{[3]} = f\left(\vec{w}_1^{[1]}.\vec{a}^{[2]} + b_1^{[1]}\right)$$

TENSOR FLOW NEURAL NETWORK ARCHITECTURE




```
x = np.array([[0, ..., 255, 0, 255, ..., 0]])

layer_1 = Dense(units = 25, activation='sigmoid')
a1 = layer_1(x)

layer_2 = Dense(units = 15, activation='sigmoid')
a2 = layer_2(a1)

layer_3 = Dense(units = 1,  activation='sigmoid')
a3 = layer_3(a2)

if a3 >= 0.5:
    yhat = 1
else:
    yhat = 0
```

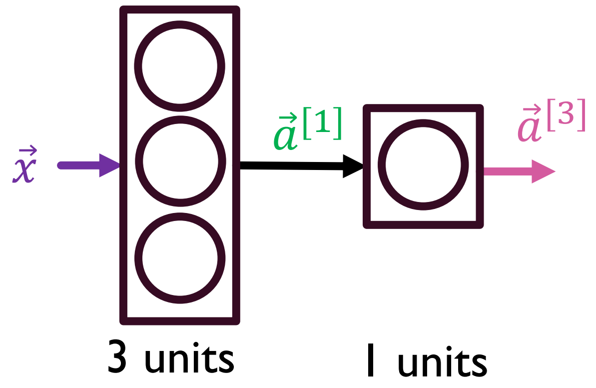
BUILDING A NEURAL NETWORK ARCHITECTURE



x1	x2	Y
119	200	1
112	12	0
34	323	0
100	43	1

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

layer_1 = Dense(units = 1, activation='sigmoid')

model = Sequential([layer_1, layer_2])

x = np.array([[119, 200],
              [112, 12 ],
              [34 , 323],
              [100, 43 ]])

y = np.array([1, 0, 0, 1])

model.compile(...)

model.fit(x, y)

x_new = np.array([[120, 40]])

model.predict(x_new)
```

BUILDING A NEURAL NETWORK ARCHITECTURE



x1	x2	Y
119	200	1
112	12	0
34	323	0
100	43	1

```python
model = Sequential([
    Dense(units = 3, activation='sigmoid'),
    Dense(units = 1, activation='sigmoid')])

x = np.array([[119, 200],
              [112, 12 ],
              [34 , 323],
              [100, 43 ]])

y = np.array([1, 0, 0, 1])

model.compile(...)

model.fit(x, y)

x_new = np.array([[120, 40]])

model.predict(x_new)
```