11210IPT553000 Deep Learning in Biomedical Optical Imaging

Week3
Neural Network Basics (Part II)

Instructor: Hung-Wen Chen @NTHU, Fall 2023
2022/09/25

SPONSORED BY

DEEP LEARNING INSTITUTE

DEEP LEARNING INSTITUTE
CERTIFIED INSTRUCTOR

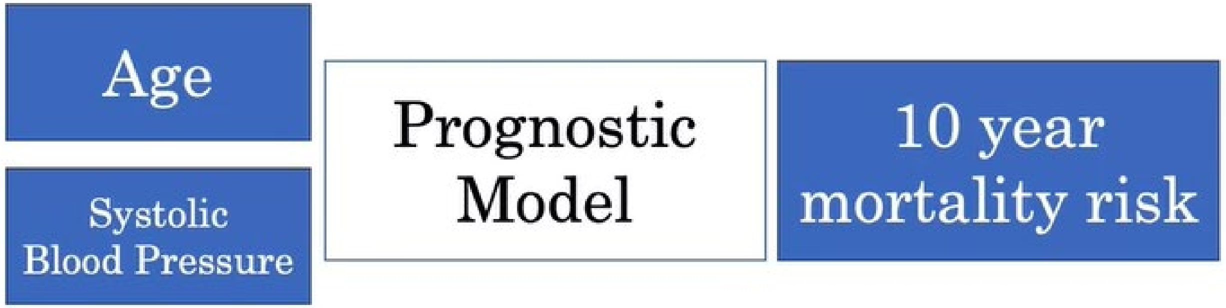
Outline of Today’s Lecture

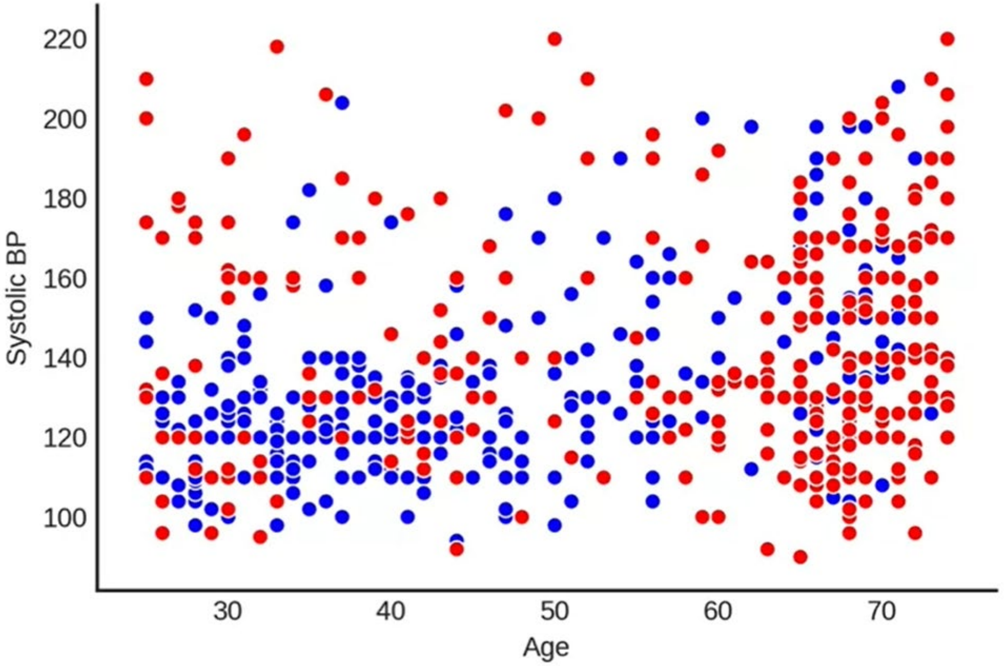
- HW1 Due today
- Shallow Neural Networks (Course 1 Week 3)
- Deep Neural Networks (Course 1 Week 4)
- Lab Practice: Build an ANN

Interpreting Model with Decision Trees

Prognostic Model

Interpreting models

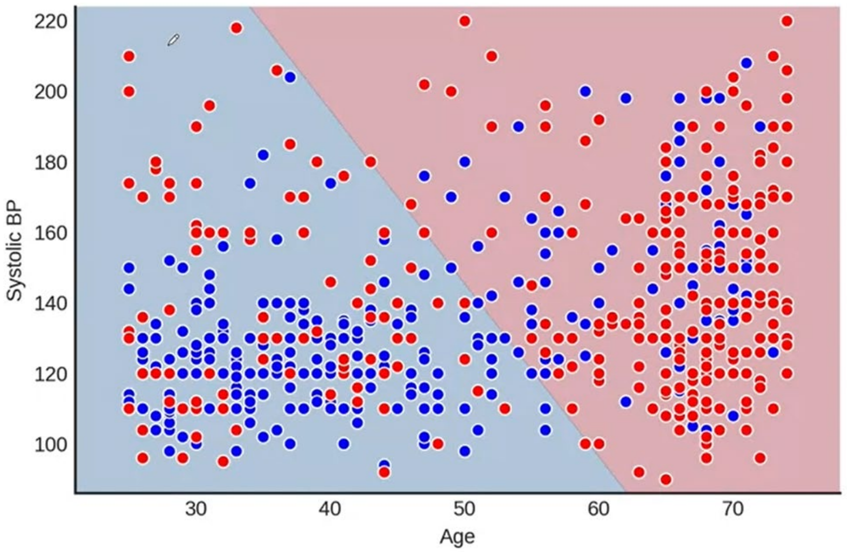


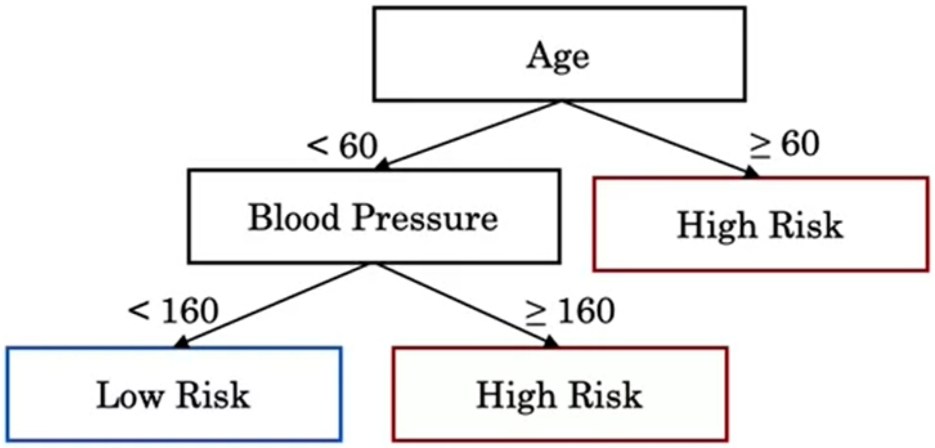


Interpreting Models

Decision tree

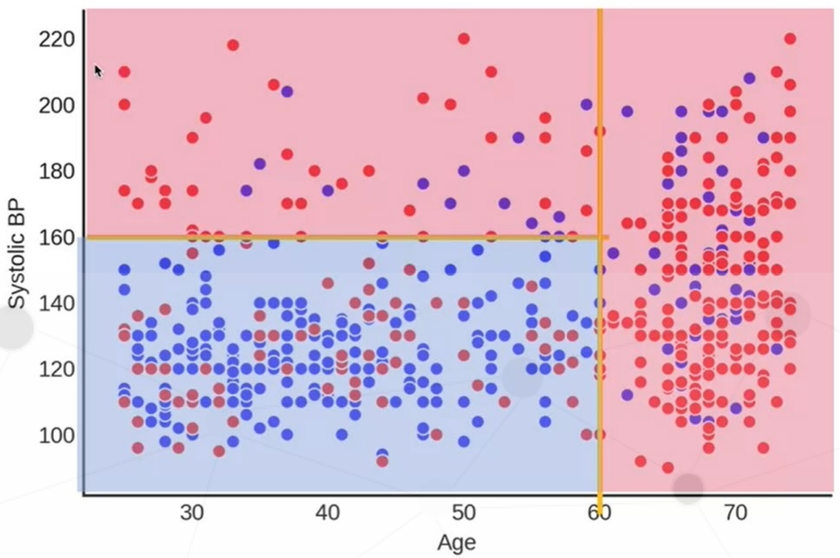
Linear Model

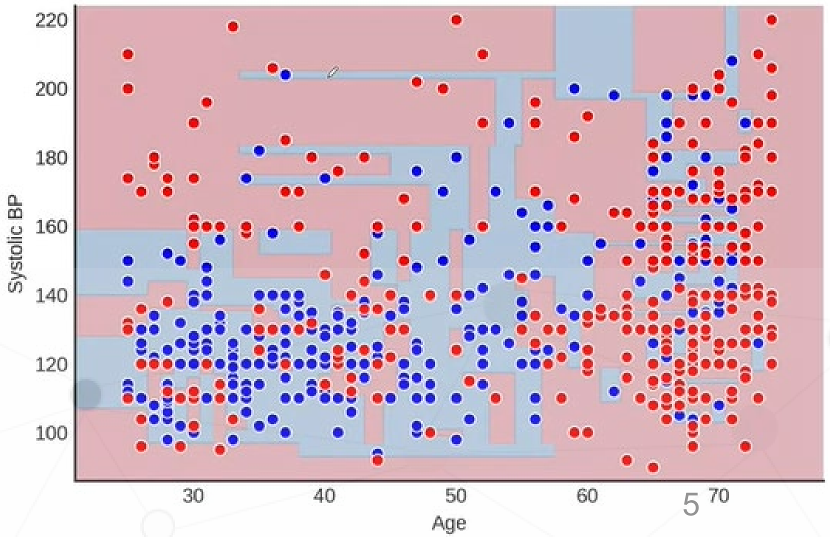




- Decision Tree
- Ensembling decision tree:
  - Random forest
  - Gradient Boosting
  - XGBoost
  - LightGBM

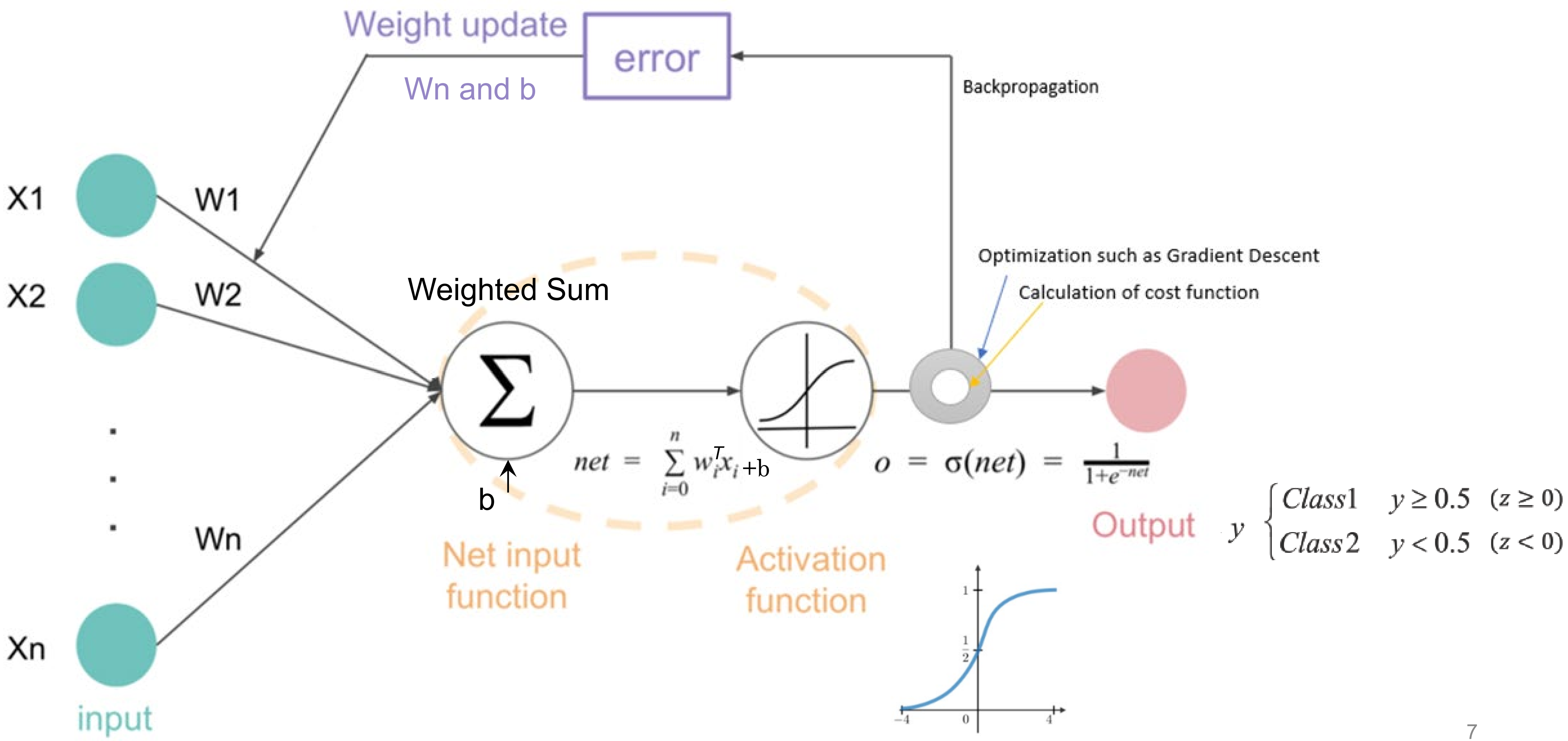
Decision Tree



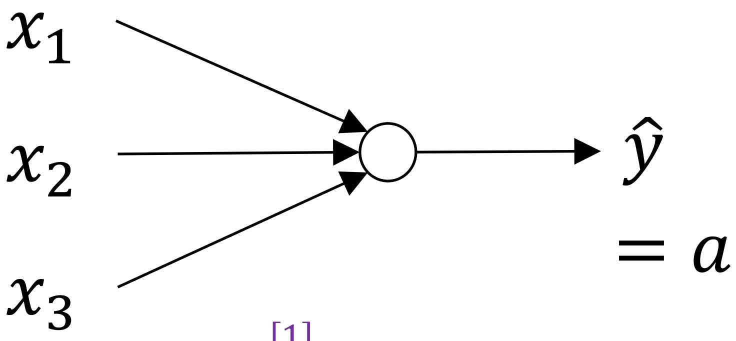


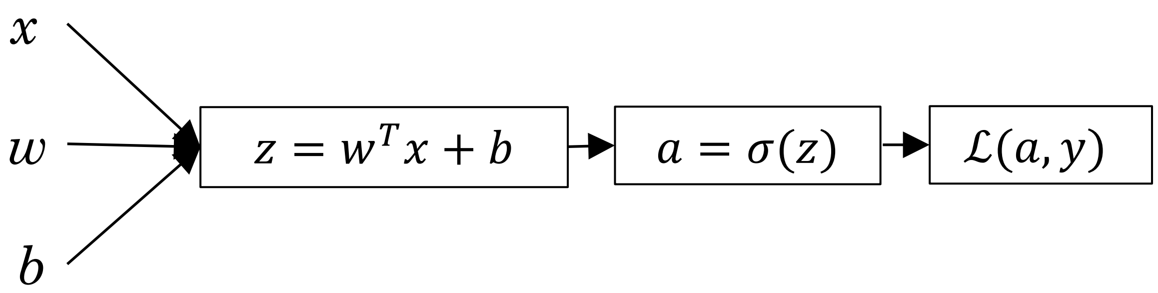
Shallow Neural Networks

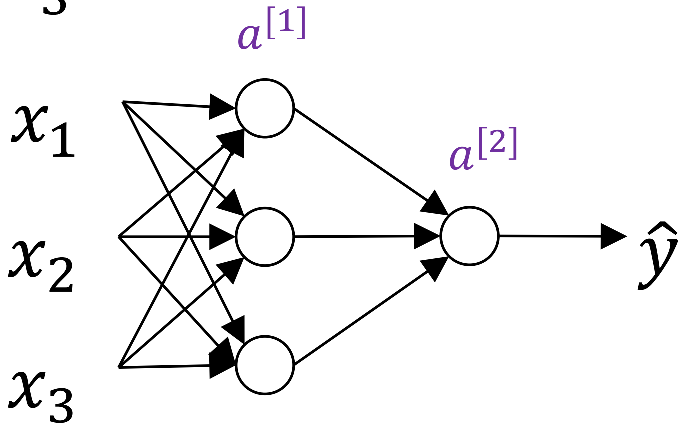
Computations of a neural network

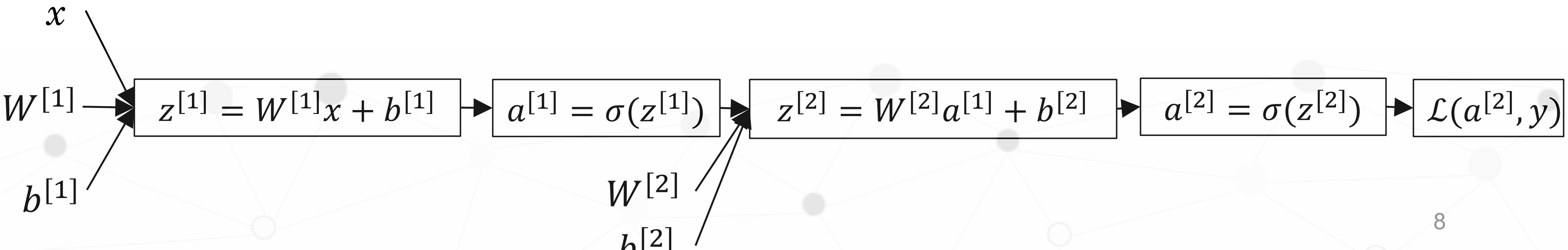


What is a neural network?



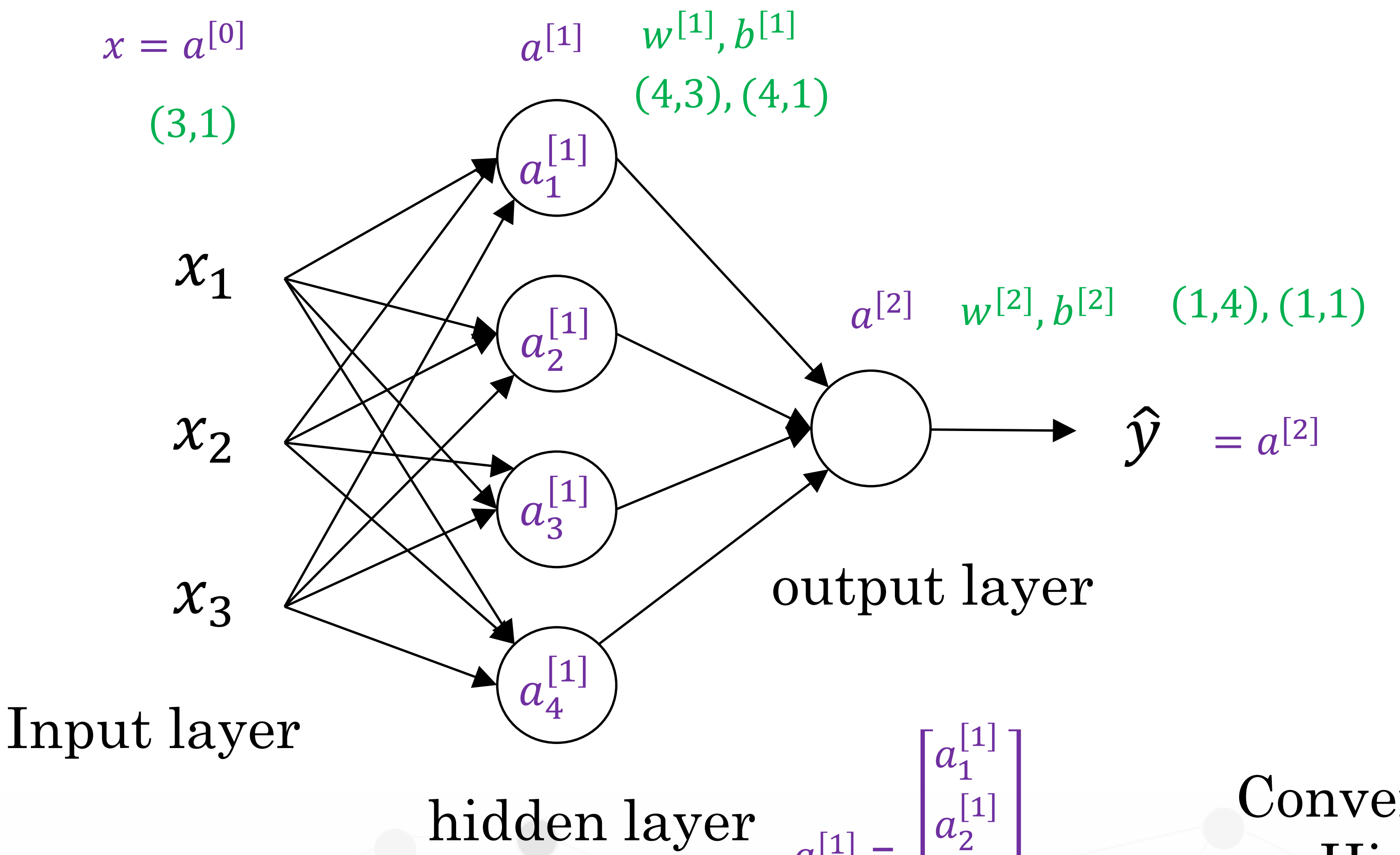






One Hidden Layer Neural Network

Neural network representation



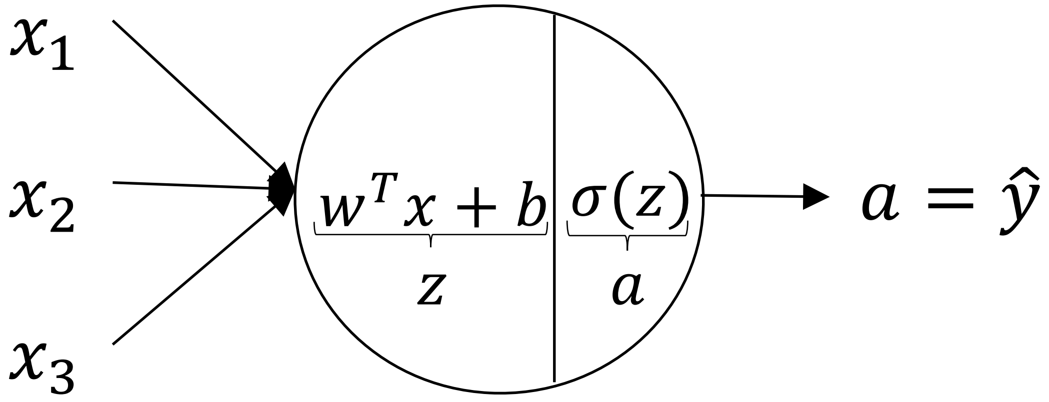
$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix}$$

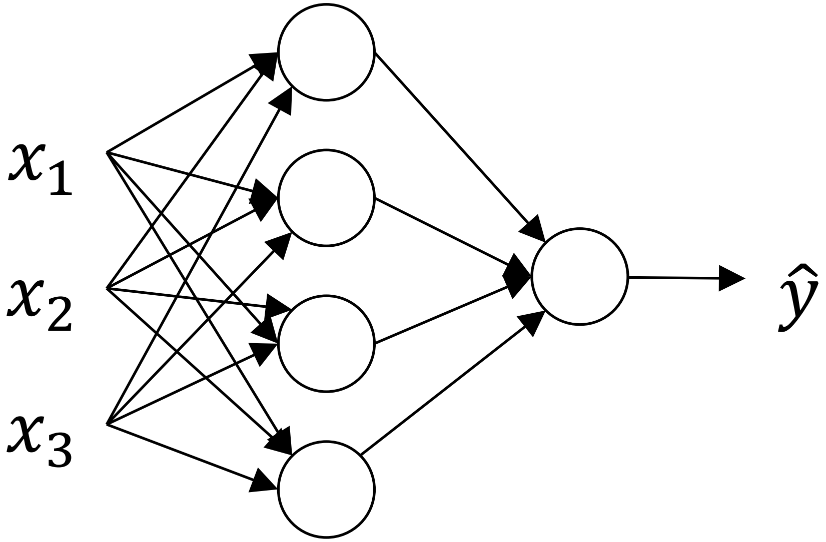
(4,1)

Conventionally, 2-layer NN
Hidden layer: layer 1
Output layer: layer 2

Computing a Neural Network's Output

Neural network representation



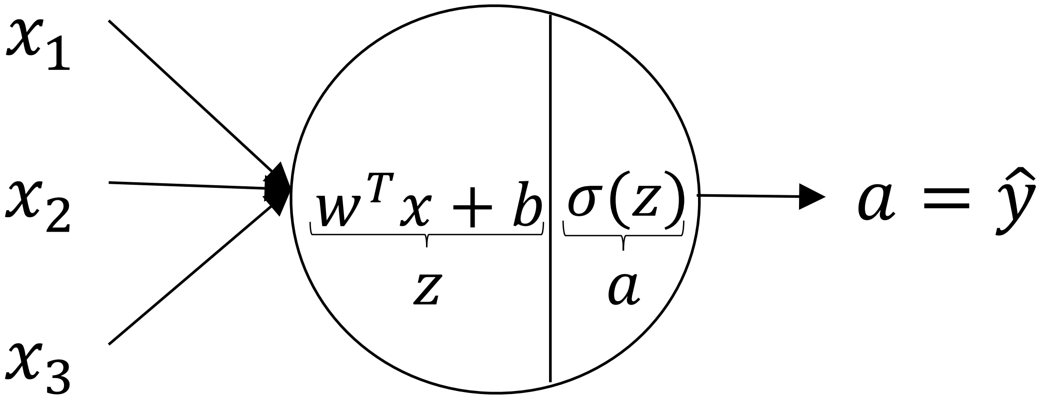


$$z = w^T x + b$$

$$a = \sigma(z)$$

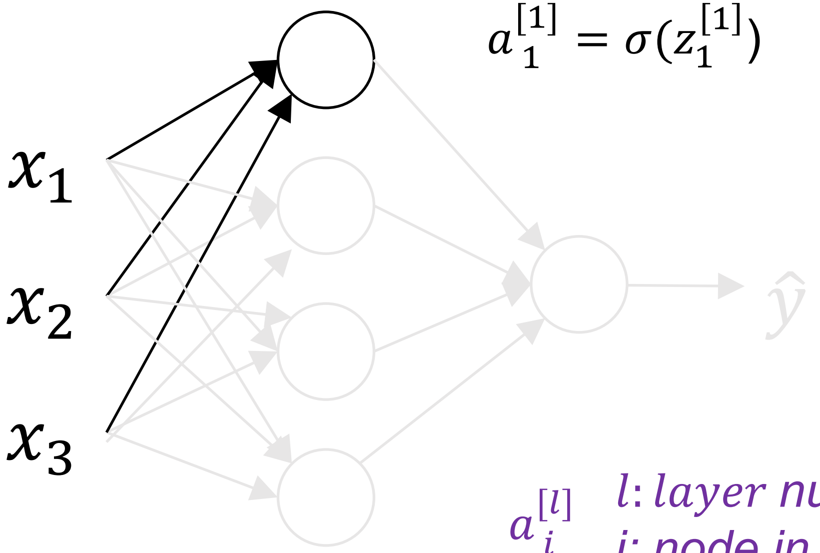
Computing a Neural Network's Output

Neural network representation



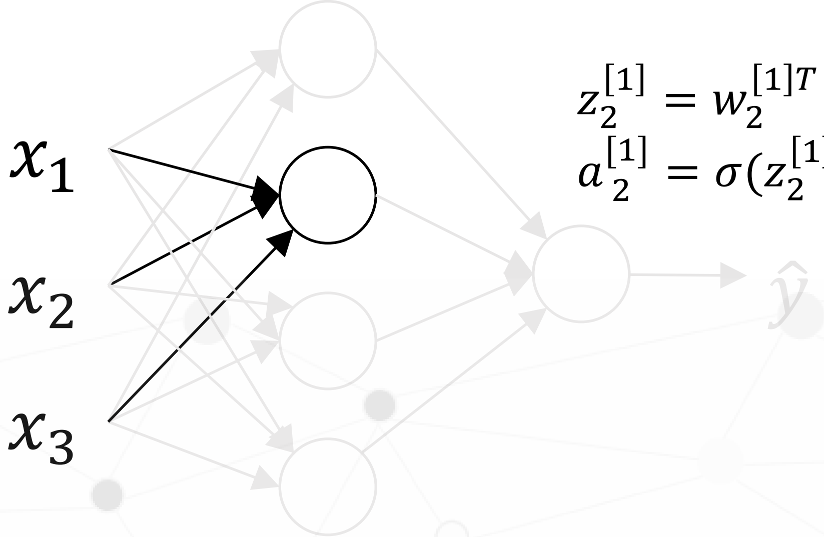
$$z = w^T x + b$$

$$a = \sigma(z)$$



$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]}$,
$a_1^{[1]} = \sigma(z_1^{[1]})$

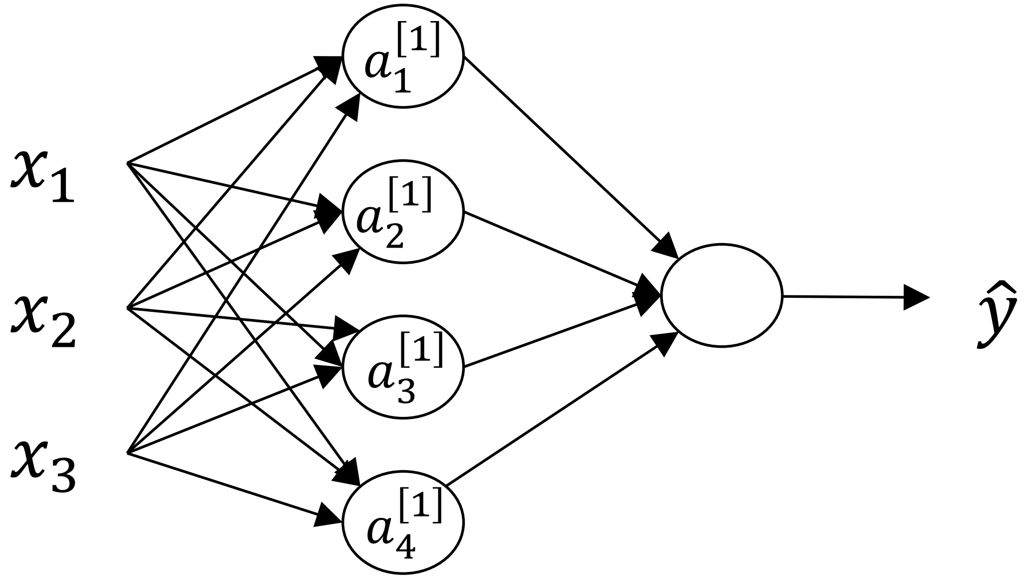
$a_i^{[l]}$ *l: layer number*
i: node in that layer



$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]}$,
$a_2^{[1]} = \sigma(z_2^{[1]})$

Computing a Neural Network's Output

Neural network representation



$$z_1^{[1]} = w_1^{[1]T} x + b_1^{[1]},\ a_1^{[1]} = \sigma(z_1^{[1]})$$

$$z_2^{[1]} = w_2^{[1]T} x + b_2^{[1]},\ a_2^{[1]} = \sigma(z_2^{[1]})$$

$$z_3^{[1]} = w_3^{[1]T} x + b_3^{[1]},\ a_3^{[1]} = \sigma(z_3^{[1]})$$

$$z_4^{[1]} = w_4^{[1]T} x + b_4^{[1]},\ a_4^{[1]} = \sigma(z_4^{[1]})$$

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}$$

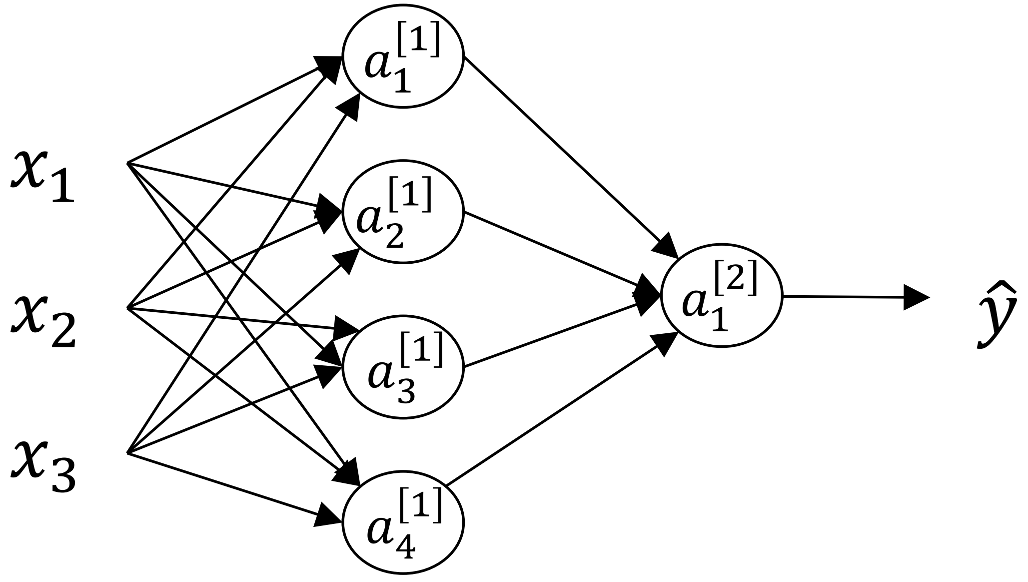
$$z^{[1]} = \underbrace{\begin{bmatrix} w_1^{[1]T} \\ w_2^{[1]T} \\ w_3^{[1]T} \\ w_4^{[1]T} \end{bmatrix}}_{W^{[1]}} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_1^{[1]} \\ b_2^{[1]} \\ b_3^{[1]} \\ b_4^{[1]} \end{bmatrix} = \begin{bmatrix} w_1^{[1]T} x & & b_1^{[1]} \\ w_1^{[1]T} x & + & b_2^{[1]} \\ w_3^{[1]T} x & & b_3^{[1]} \\ w_4^{[1]T} x & & b_4^{[1]} \end{bmatrix} = \begin{bmatrix} z_1^{[1]} \\ z_2^{[1]} \\ z_3^{[1]} \\ z_4^{[1]} \end{bmatrix}$$

$$a^{[1]} = \begin{bmatrix} a_1^{[1]} \\ a_2^{[1]} \\ a_3^{[1]} \\ a_4^{[1]} \end{bmatrix} = \sigma(z_1^{[1]})$$

$$\boxed{\begin{aligned} z^{[1]} &= W^{[1]}x + b^{[1]} \\ a^{[1]} &= \sigma(z^{[1]}) \end{aligned}}$$

Computing a Neural Network's Output

Neural network representation learning



$a^{[0]} = x$

Given input x: $= a^{[0]}$

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

(4,1) (4,3) (3,1) (4,1)

$$a^{[1]} = \sigma(z^{[1]})$$

(4,1) (4,1)

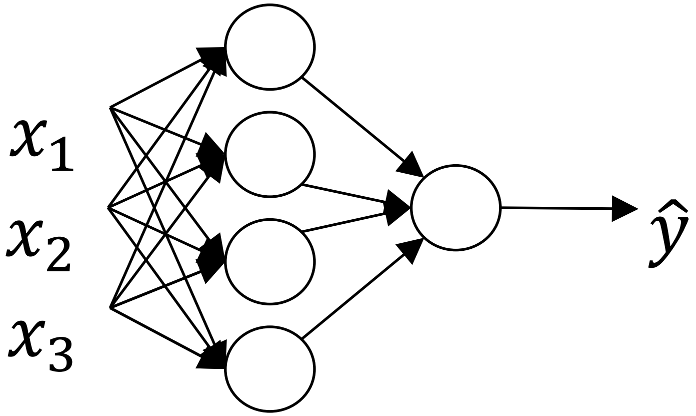
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

(1,1) (1,4) (4,1) (1,1)

$$a^{[2]} = \sigma(z^{[2]})$$

(1,1) (1,1)

Explanation for Vectorized Implementation



Vectorizing across multiple examples (m)

m training examples: $\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), \ldots\ldots, (x^{(m)}, y^{(m)})\}$

$$X = \begin{bmatrix} | & | & & | \\ x^{(1)} & x^{(2)} & \cdots & x^{(m)} \\ | & | & & | \end{bmatrix}$$

i-th hidden unit

$$\mathrm{A}^{[1]} = \begin{bmatrix} | & | & & | \\ a^{[1](1)} & a^{[1](2)} & \cdots & a^{[1](m)} \\ | & | & & | \end{bmatrix}$$

i-th training example

for i = 1 to m

$$z^{[1](i)} = W^{[1]} x^{(i)} + b^{[1]}$$

$$a^{[1](i)} = \sigma(z^{[1](i)})$$

$$z^{[2](i)} = W^{[2]} a^{[1](i)} + b^{[2]}$$

$$a^{[2](i)} = \sigma(z^{[2](i)})$$

$$Z^{[1]} = W^{[1]} \overset{= A^{[0]}}{X} + b^{[1]}$$

$$A^{[1]} = \sigma(Z^{[1]})$$

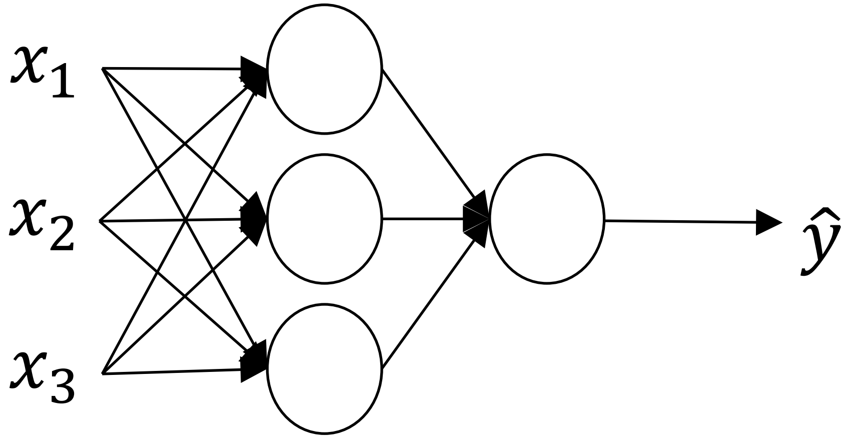
$$Z^{[2]} = W^{[2]} A^{[1]} + b^{[2]}$$

$$A^{[2]} = \sigma(Z^{[2]})$$

Activation functions

One Hidden Layer Neural Network

Activation functions



Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

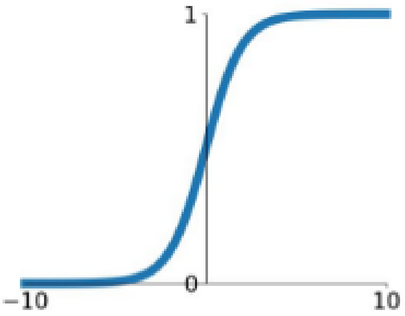
$$a^{[1]} = \sigma(z^{[1]}) \leftarrow g(z^{[1]})$$

$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = \sigma(z^{[2]}) \leftarrow g(z^{[2]})$$

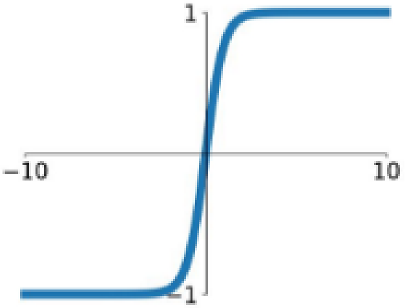
Sigmoid

$$\sigma(x) = \frac{1}{1+e^{-x}}$$



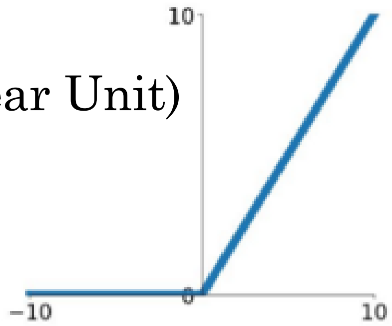
tanh

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$



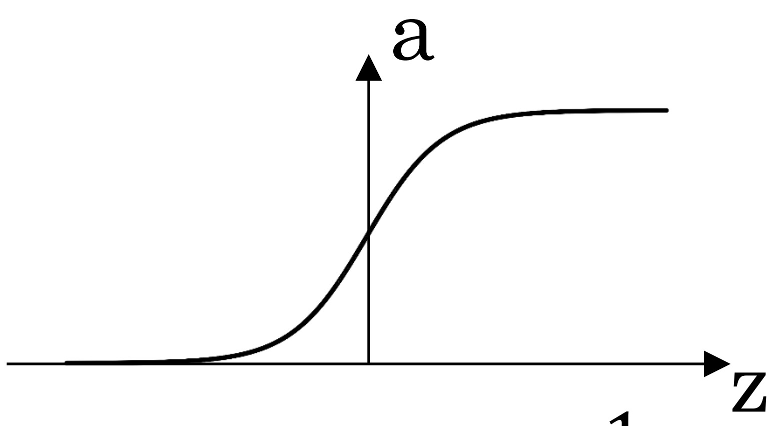
ReLU (Rectified Linear Unit)

$$\max(0, x)$$

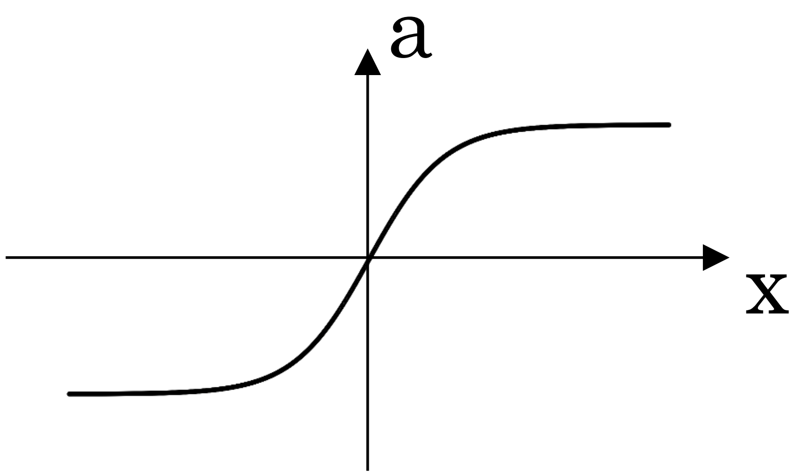


Activation Functions

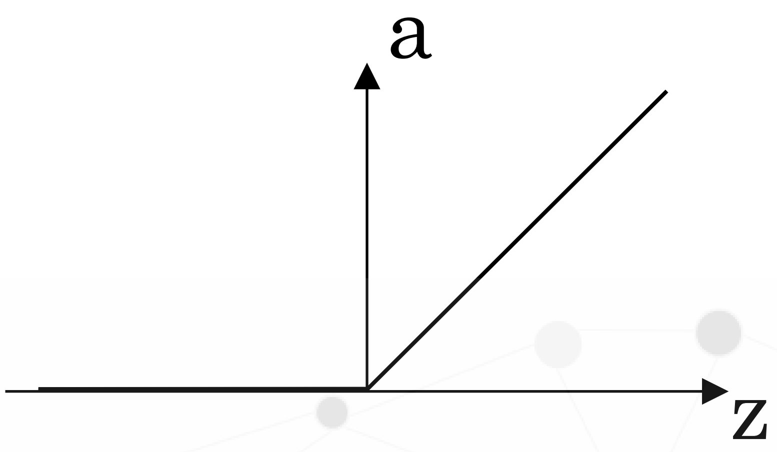
Pros and cons of activation functions



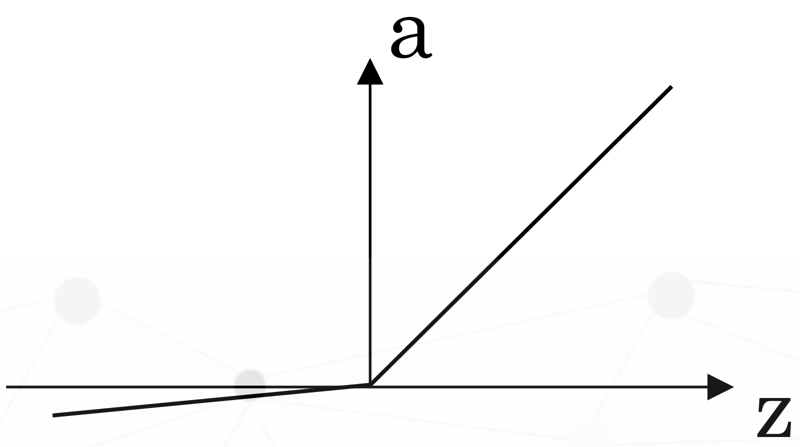
sigmoid: $a = \dfrac{1}{1+e^{-z}}$



$tanh: a = \dfrac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$



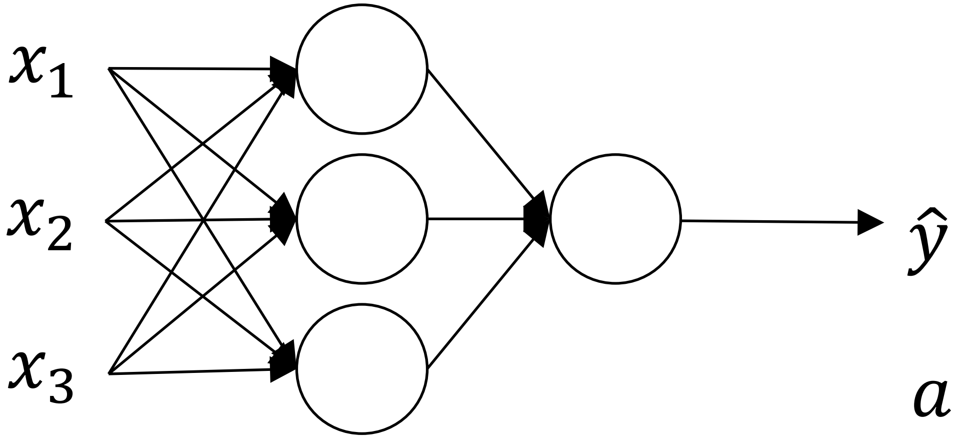
$Relu: a = \max(0, z)$



$Leaky\ Relu: a = \max(0.01z, z)$

Why Do You Need Non-Linear Activation Functions?

Activation function



$$a^{[1]} = z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[2]} = z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$\begin{aligned} a^{[2]} &= z^{[2]} = W^{[2]}\left(W^{[1]}x + b^{[1]}\right) + b^{[2]} \\ &= (W^{[2]}W^{[1]})x + (W^{[2]}b^{[1]} + b^{[2]}) \\ &= (W')x + (b') \end{aligned}$$

Given x:

$$z^{[1]} = W^{[1]}x + b^{[1]}$$

$$a^{[1]} = g^{[1]}(z^{[1]}) \leftarrow z^{[1]}$$

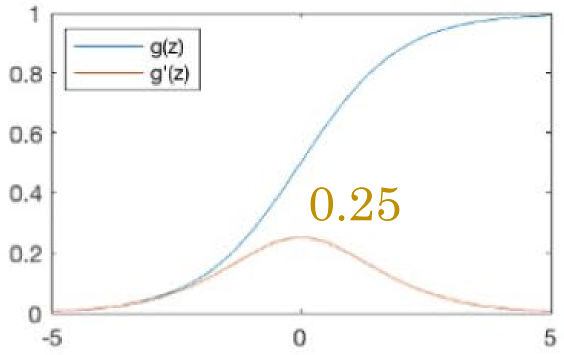
$$z^{[2]} = W^{[2]}a^{[1]} + b^{[2]}$$

$$a^{[2]} = g^{[2]}(z^{[2]}) \leftarrow z^{[2]}$$

Activation Functions

Derivative of common activation functions

Sigmoid Function

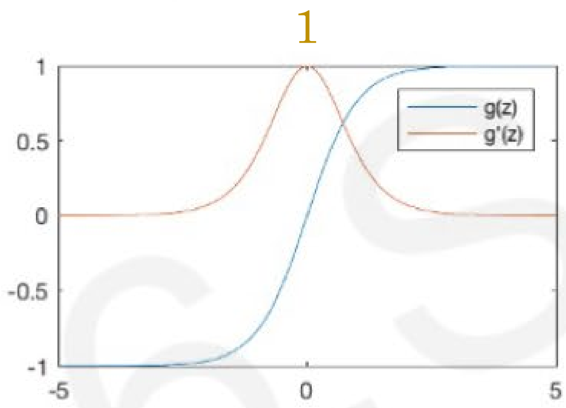


$$g(z) = \frac{1}{1+e^{-z}}$$

$$g'(z) = g(z)(1 - g(z))$$

```
tf.math.sigmoid(z)
```

Hyperbolic Tangent

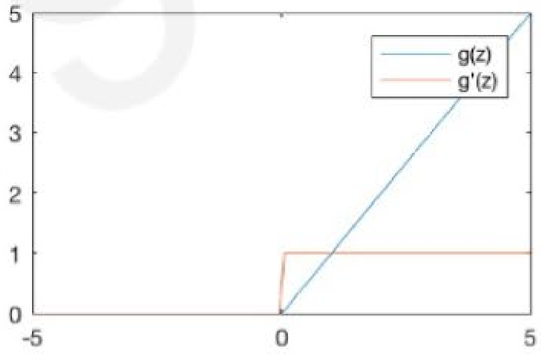


$$g(z) = \frac{e^{z} - e^{-z}}{e^{z} + e^{-z}}$$

$$g'(z) = 1 - g(z)^2$$

```
tf.math.tanh(z)
```

Rectified Linear Unit (ReLU)



$$g(z) = \max(0, z)$$

$$g'(z) = \begin{cases} 1, & z > 0 \\ 0, & \text{otherwise} \end{cases}$$

```
tf.nn.relu(z)
```

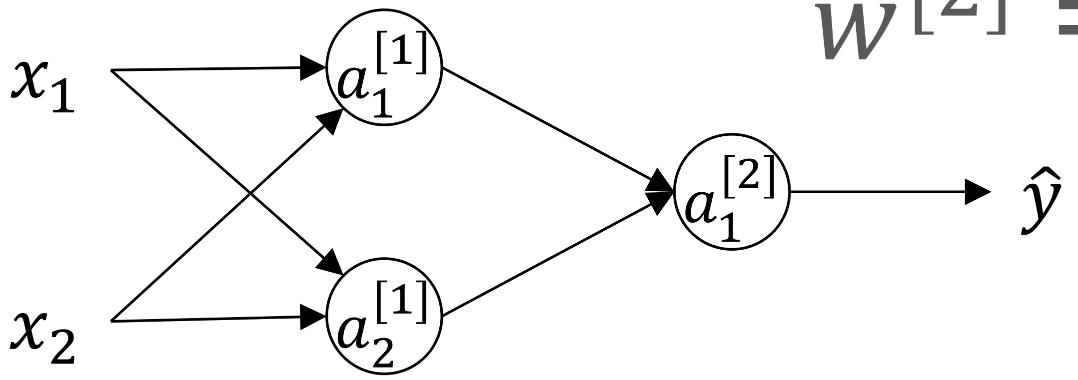
TensorFlow code blocks

NOTE: All activation functions are non-linear

Initialization

One Hidden Layer Neural Network

What happens if you initialize weights to zero?



$$w^{[2]} = \begin{bmatrix} 0 & 0 \end{bmatrix}$$

$$w^{[1]} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$$

The initial weights to the network should be unequal to break the symmetry.

Random initialization! (Small numbers)

One Hidden Layer Neural Network

What happens if you initialize weights to zero?

4.2 - Initialize the model's parameters

Exercise: Implement the function `initialize_parameters()`.

Instructions:

- Make sure your parameters' sizes are right. Refer to the neural network figure ab
- You will initialize the weights matrices with random values.
  - Use: `np.random.randn(a,b) * 0.01` to randomly initialize a matrix of shap
- You will initialize the bias vectors as zeros.
  - Use: `np.zeros((a,b))` to initialize a matrix of shape (a,b) with zeros.

In [9]:
```
# GRADED FUNCTION: initialize_parameters

def initialize_parameters(n_x, n_h, n_y):
    """
    Argument:
    n_x -- size of the input layer
    n_h -- size of the hidden layer
    n_y -- size of the output layer

    Returns:
    params -- python dictionary containing your parameters:
                    W1 -- weight matrix of shape (n_h, n_x)
                    b1 -- bias vector of shape (n_h, 1)
                    W2 -- weight matrix of shape (n_y, n_h)
                    b2 -- bias vector of shape (n_y, 1)
    """

    np.random.seed(2) # we set up a seed so that your output matches

    ### START CODE HERE ### (≈ 4 lines of code)
    W1 = np.zeros((n_h,n_x))*0.01
    b1 = np.zeros((n_h,1))
    W2 = np.zeros((n_y,n_h))*0.01
    b2 = np.zeros((n_y,1))
    ### END CODE HERE ###
```

In [27]:
```
X_assess, Y_assess = nn_model_test_case()
parameters = nn_model(X_assess, Y_assess, 4, num_iterations=10, print_cost=True)
print("W1 = " + str(parameters["W1"]))
print("b1 = " + str(parameters["b1"]))
print("W2 = " + str(parameters["W2"]))
print("b2 = " + str(parameters["b2"]))
```

```
Cost after iteration 0: 0.693147
W1 = [[ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]]
b1 = [[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
W2 = [[ 0.  0.  0.  0.]]
b2 = [[ 0.2]]
```

```
Cost after iteration 1: 0.664806
W1 = [[ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]
 [ 0.  0.]]
b1 = [[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
W2 = [[ 0.  0.  0.  0.]]
b2 = [[ 0.3401992]]
```

One Hidden Layer Neural Network

What happens if you initialize weights to one?

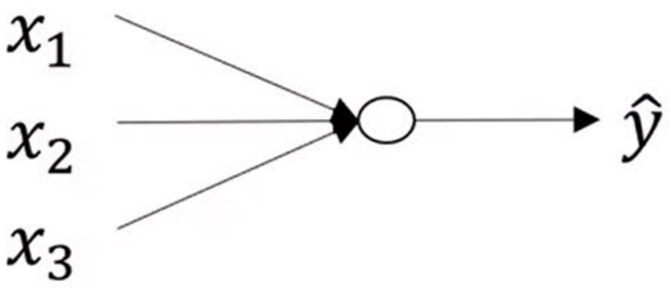
```
### START CODE HERE ### (≈ 4 lines of code)
W1 = np.zeros((n_h,n_x)) + 1
b1 = np.zeros((n_h,1))
W2 = np.zeros((n_y,n_h)) +1
b2 = np.zeros((n_y,1))
### END CODE HERE ###
```

```
Cost after iteration 0: 1.808501
W1 = [[ 1.22234102  0.71237626]
 [ 1.22234102  0.71237626]
 [ 1.22234102  0.71237626]
 [ 1.22234102  0.71237626]]
b1 = [[-0.23297283]
 [-0.23297283]
 [-0.23297283]
 [-0.23297283]]
W2 = [[ 0.56141072  0.56141072  0.56141072  0.56141072]]
b2 = [[ 0.14812489]]
```

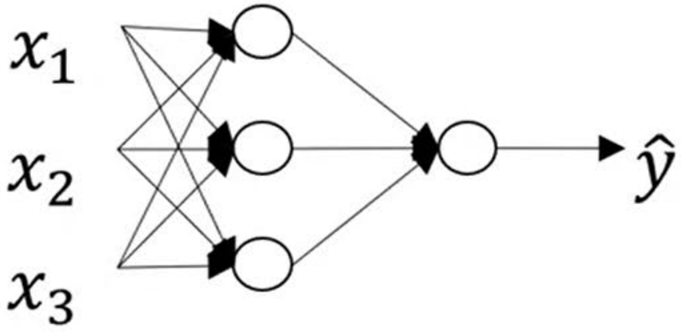
```
Cost after iteration 1: 0.919667
W1 = [[ 1.28266063  0.62747155]
 [ 1.28266063  0.62747155]
 [ 1.28266063  0.62747155]
 [ 1.28266063  0.62747155]]
b1 = [[-0.28306887]
 [-0.28306887]
 [-0.28306887]
 [-0.28306887]]
W2 = [[ 0.30042713  0.30042713  0.30042713  0.30042713]]
b2 = [[ 0.41922517]]
```

Deep L-layer Neural Networks

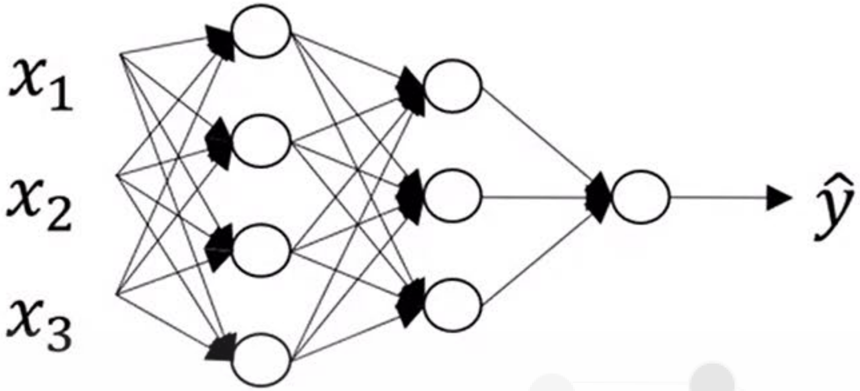
Notation



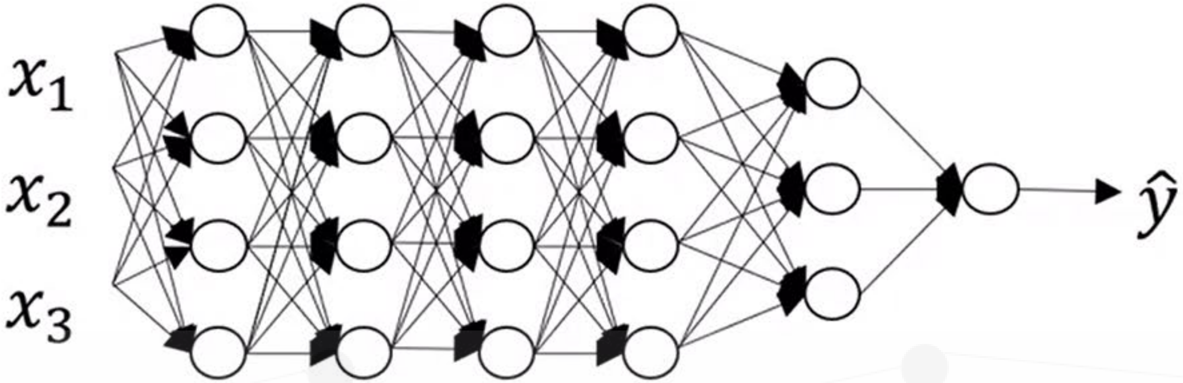
logistic regression



1 hidden layer



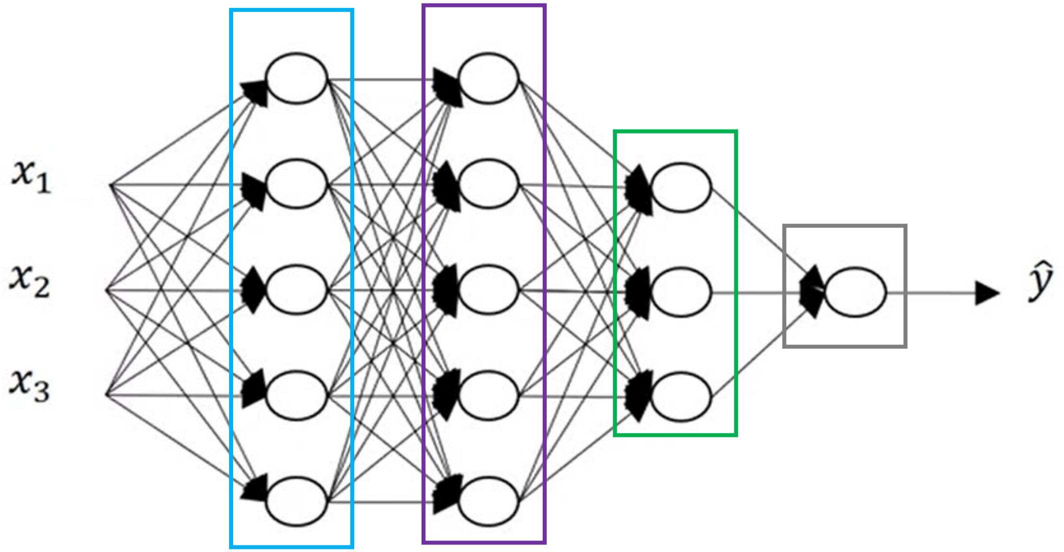
2 hidden layers



5 hidden layers

Deep L-layer Neural Network

Forward propagation in a deep network



$$Z^{[1]} = W^{[1]}X + b^{[1]}$$
$$A^{[1]} = g^{[1]}(Z^{[1]})$$
$$Z^{[2]} = W^{[2]}A^{[1]} + b^{[2]}$$
$$A^{[2]} = g^{[2]}(Z^{[2]})$$
$$Z^{[3]} = W^{[3]}A^{[2]} + b^{[3]}$$
$$A^{[3]} = g^{[3]}(Z^{[3]})$$
$$A^{[4]} = g^{[4]}(Z^{[4]}) = \hat{Y}$$

L = 4

$n^{[l]}$*: number of units in layer l*

$a^{[l]}$*: activations in layer l*

$a^{[l]} = g^{[l]}(Z^{[l]})$

$Layer\ l:\ W^{[l]}, b^{[l]}$

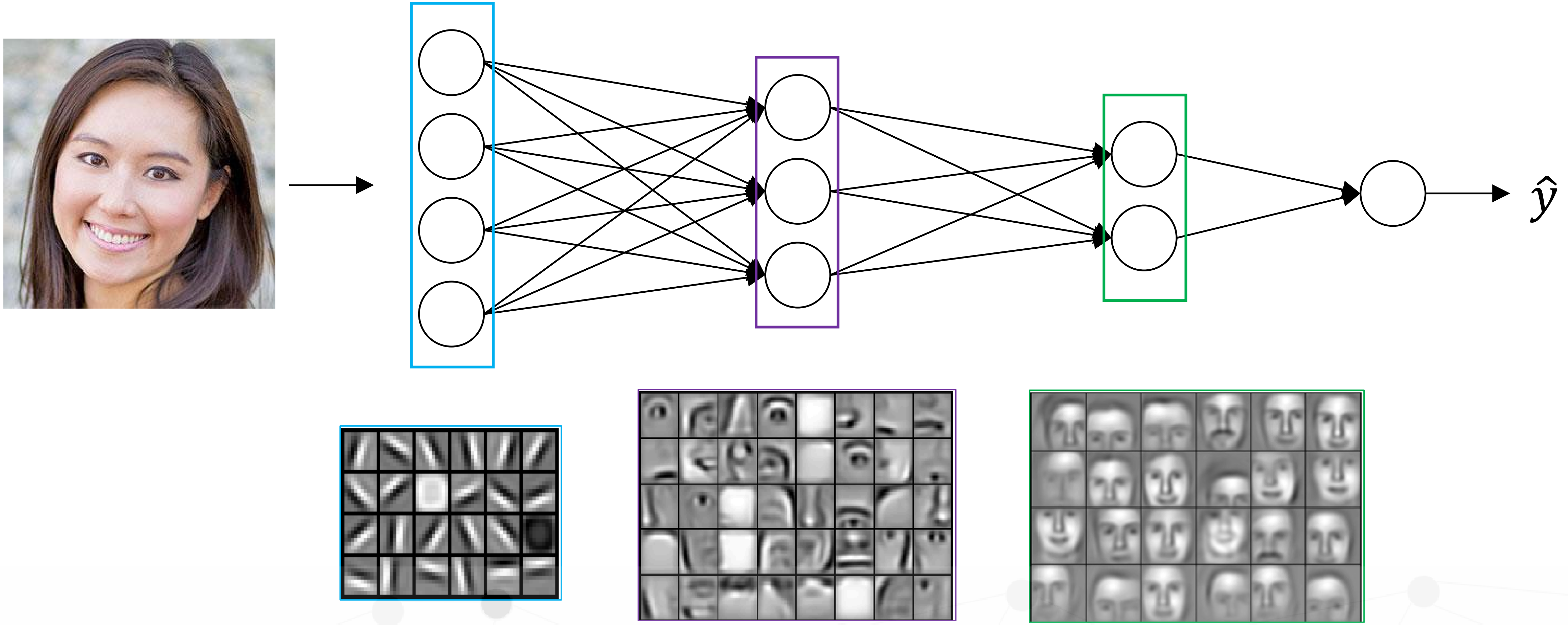
$Forward: Input\ A^{[l-1]}, output\ A^{[l]}$

$$Z^{[l]} = W^{[l]}A^{[l-1]} + b^{[l]}$$
$$A^{[l]} = g^{[l]}(Z^{[l]})$$

Why Deep Representations?

Intuition about deep representation

Intuition about deep representation

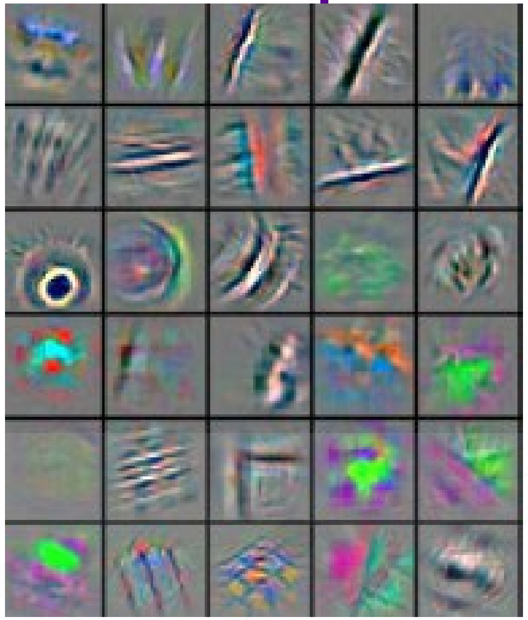


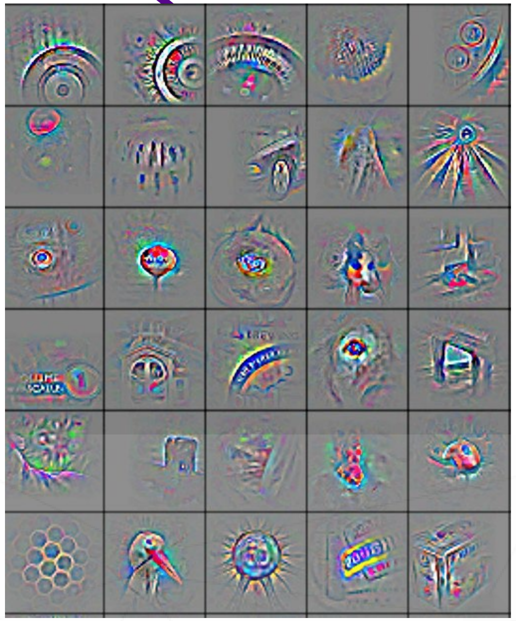
Why Deep Representations?

Deep learning = Learning hierarchical representations

It's deep if it has more than one stage of non-linear feature transformation

Low-level feature → Mid-level feature → High-level feature → Trainable classifier →



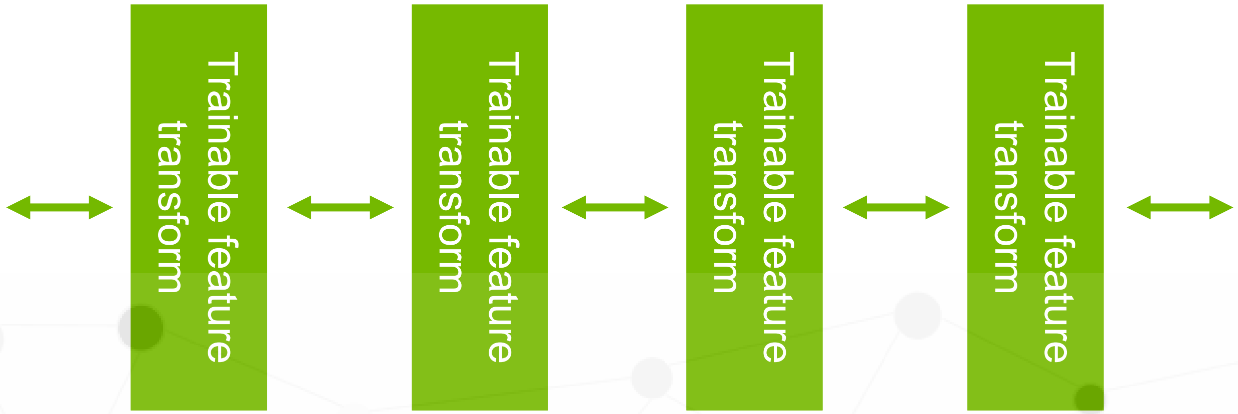


Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Why Deep Representations?

Trainable feature hierarchy

- Hierarchy of representations with increasing level of abstraction. Each stage is a kind of trainable feature transform
- Image recognition
  - Pixel → edge → texton → motif → part → object
- Text
  - Character → word → word group → clause → sentence → story
- Speech
  - Sample → spectral band → sound → … → phone → phoneme → word



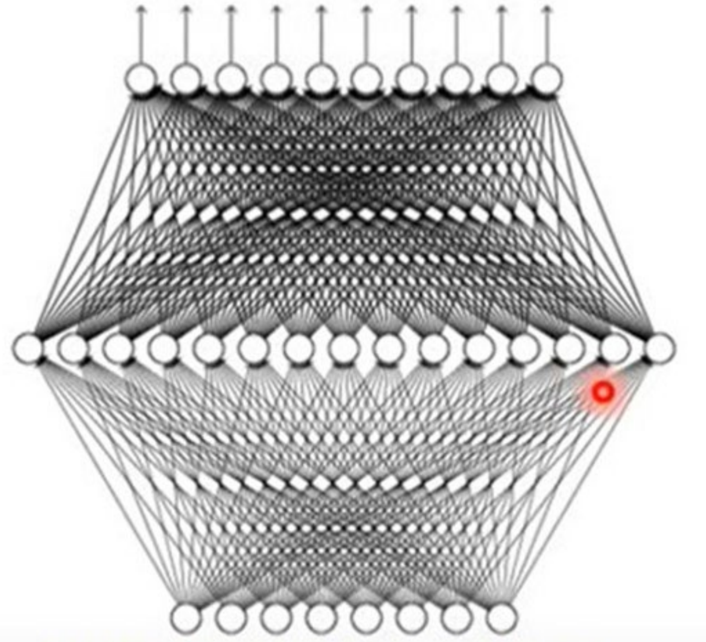
Why Deep Representations?

Universal approximation theorem

Any continuous function *f*

$$f : R^N \rightarrow R^M$$

Can be realized by a network with one hidden layer (given **enough** hidden neurons)



Yes, shallow network can represent any function.

However, using deep structure is more effective.

Reference for the reason:

http://neuralnetworksanddeeplearning.com/chap4.html

A feedforward network with a single layer is sufficient to represent any function, but the layer may be infeasibly large and may fail to learn and generalize correctly.
— Ian Goodfellow, DLB

Why Deep Representations?

Universal approximation theorem

- In worse case, exponential number of hidden units (possibly one for each input config that needs to be distinguished) is required.

- In binary case, easy to see: number of possible binary vectors on v in {0,1}n is 2^2^n, selecting one such function requires 2^n bits.

- While single hidden layer is sufficient to represent any function, the layer may be unfeasibly large and may fail to learn and generalize correctly.

Parameters vs Hyperparameters

What are hyperparameters?

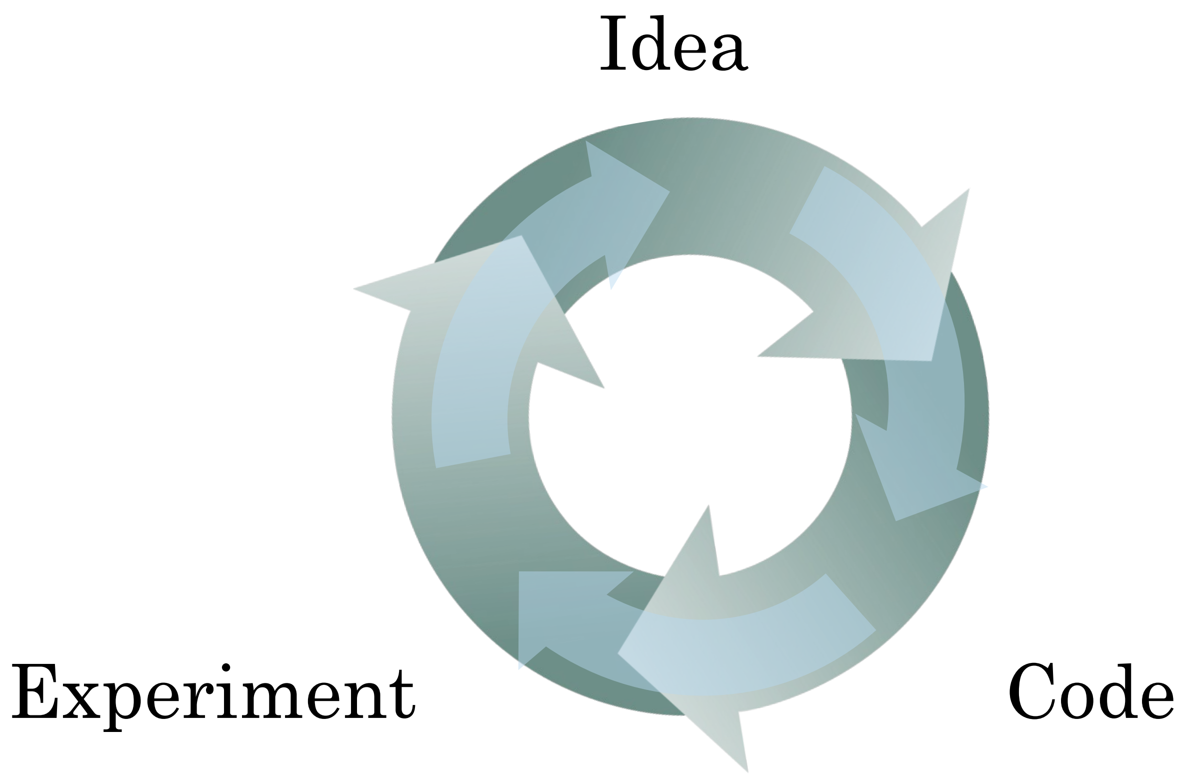
Parameters: $W^{[1]}, b^{[1]}, W^{[2]}, b^{[2]}, W^{[3]}, b^{[3]}$ ...

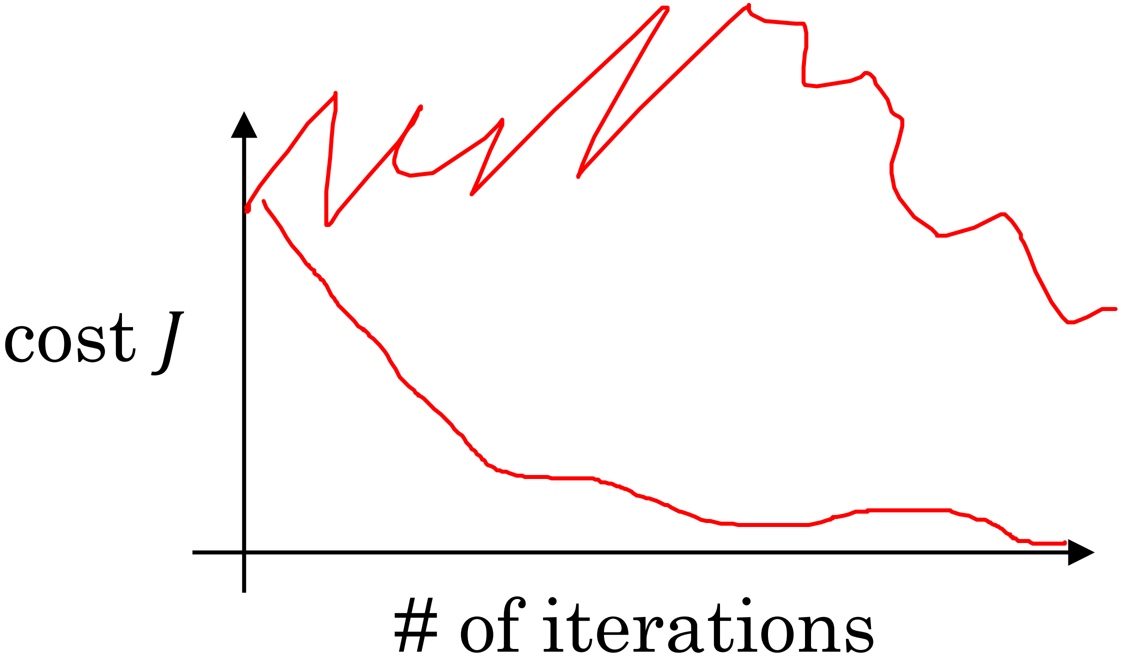
Hyperparameters: *learning rate* α
# of iterations
# of hidden layers L
# of hidden units $n^{[1]}, n^{[2]}, \ldots\ldots$
choice of activation function

Later: *momentum, mini-batch size, regularization*

Parameters vs Hyperparameters

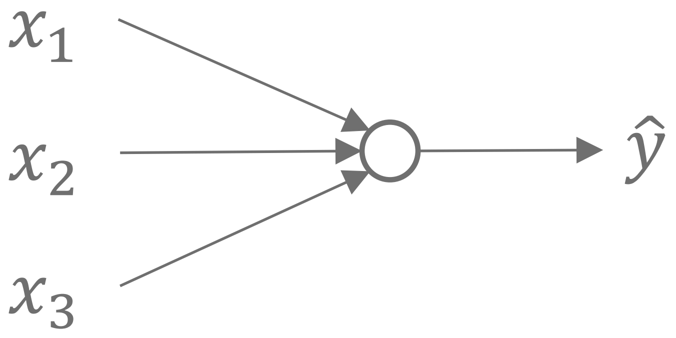
Applied deep learning is a very empirical process

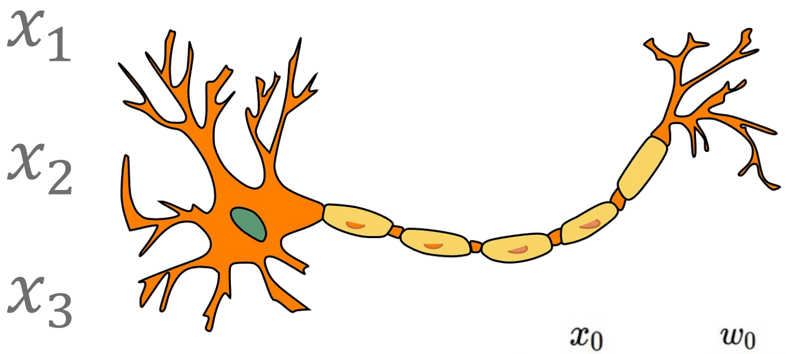


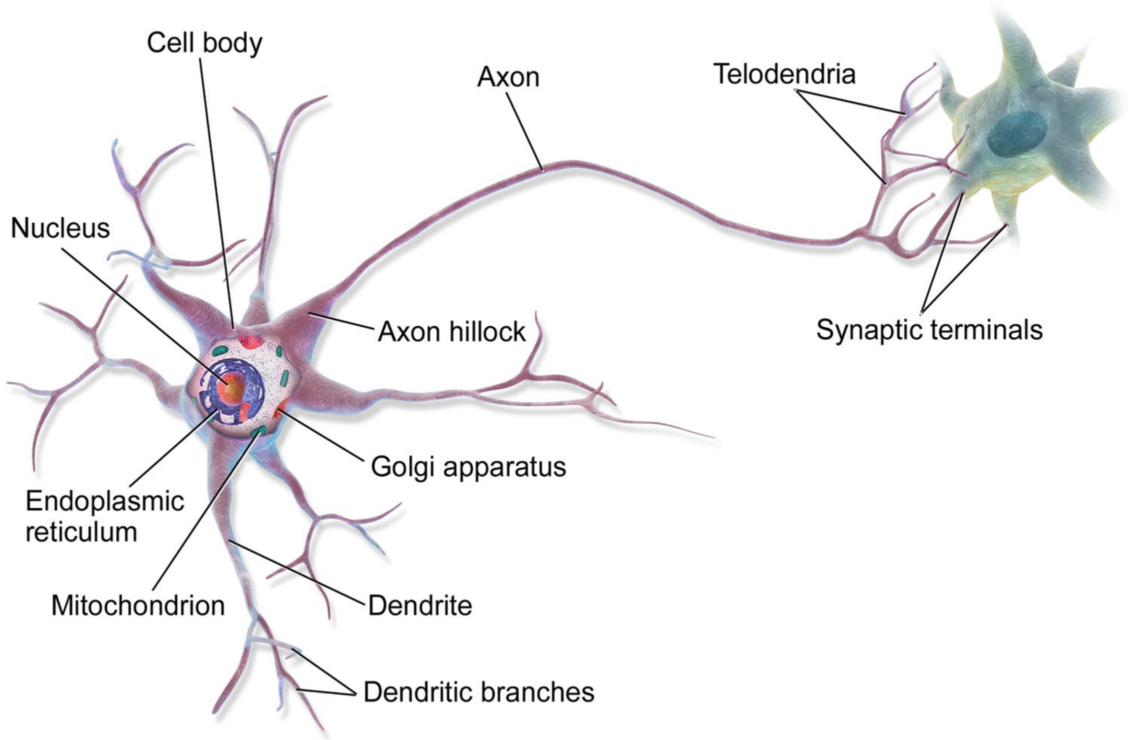


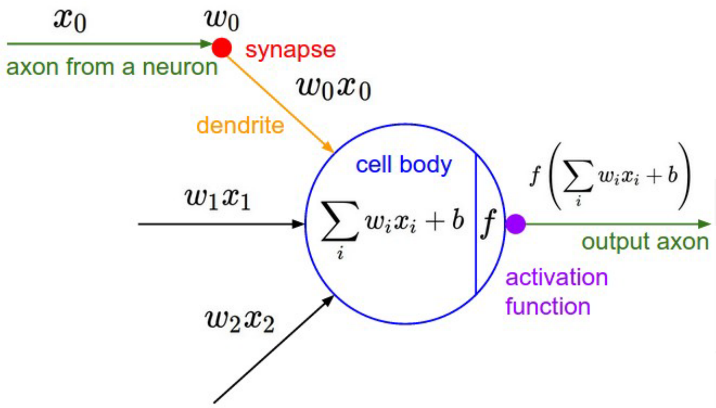
What Does This Have to Do With the Brain?

Biological inspiration









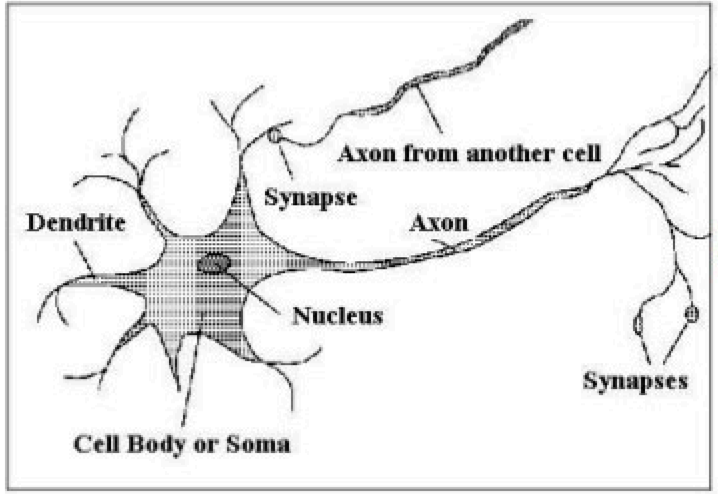
Biological inspiration

The Perceptron (Neuron)

The Perceptron is seen as an **analogy** to a biological neuron.

Biological neurons fire an impulse once the sum of all inputs is over a threshold.

The sigmoid emulates the thresholding behavior → act like a switch.



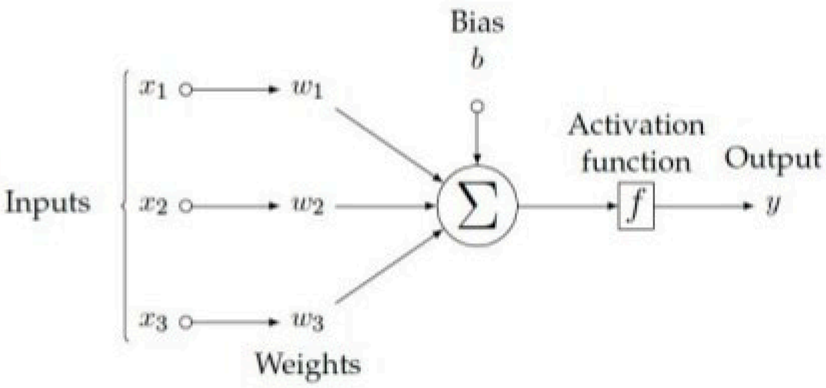
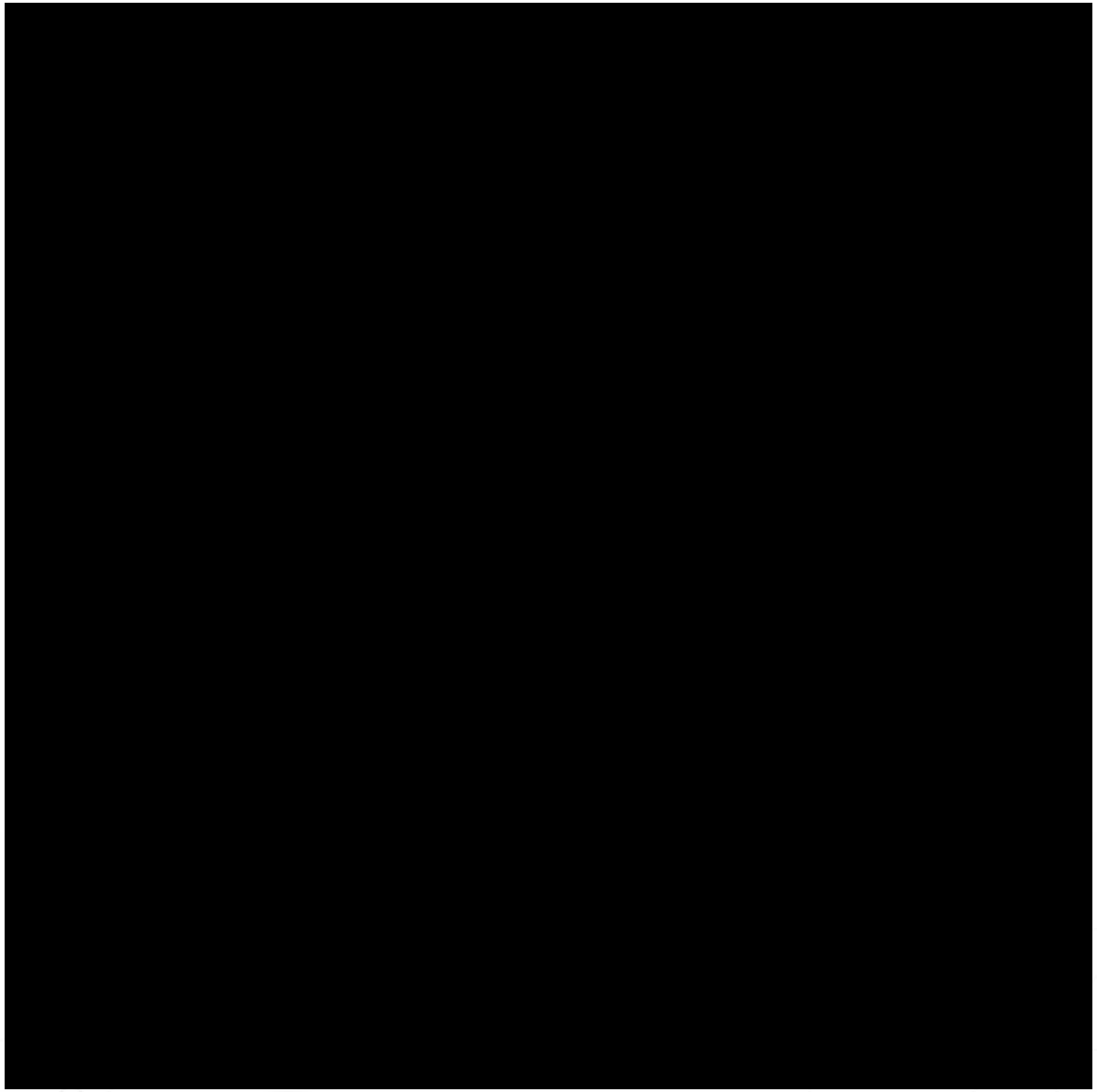
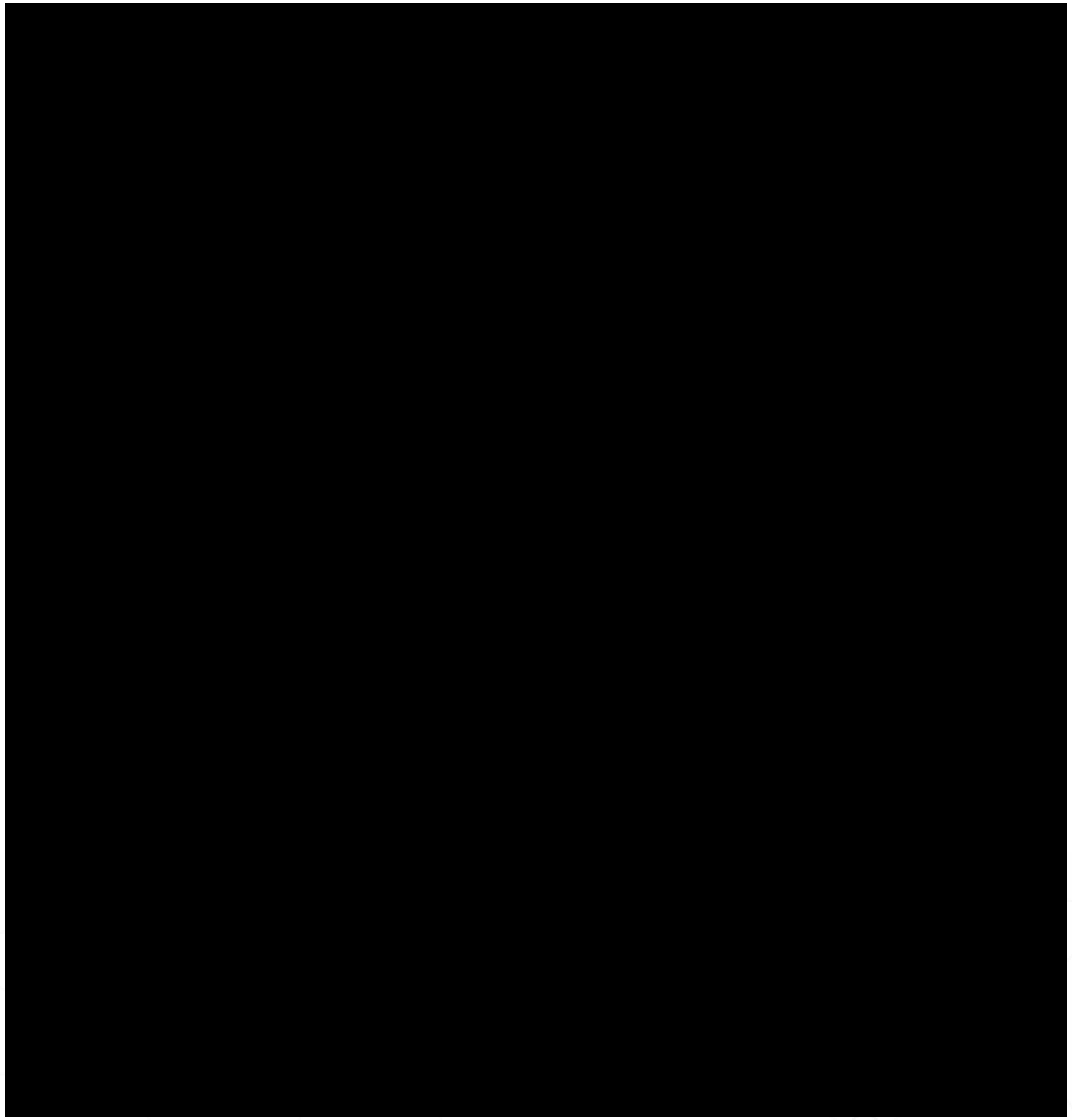


Figure credit: [Introduction to AI]

What Does This Have to Do With the Brain?

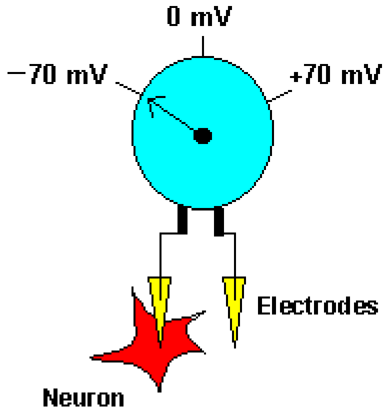
Finding connections

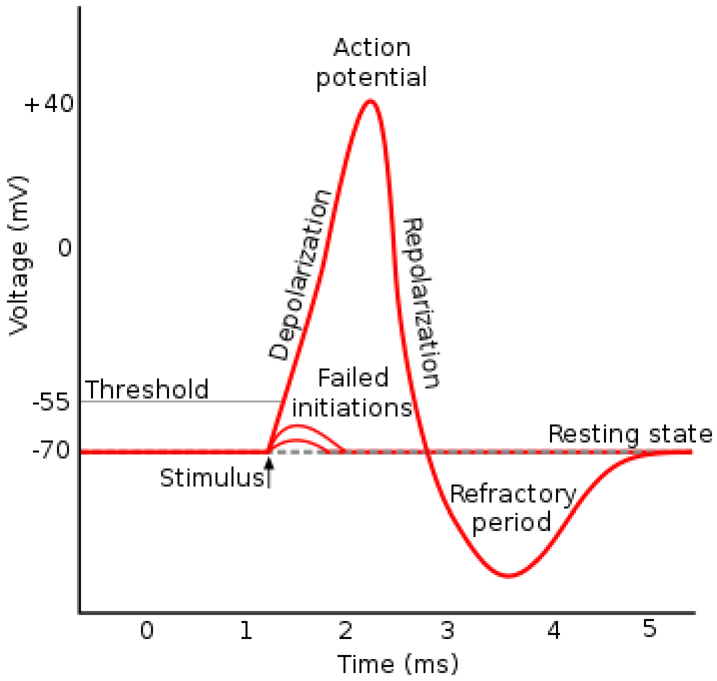


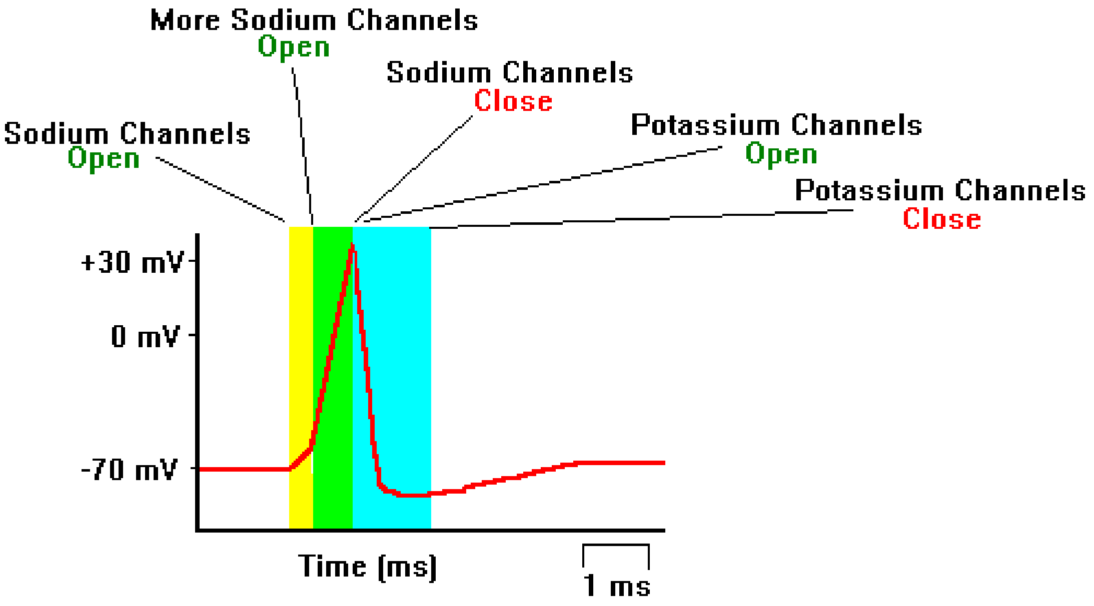


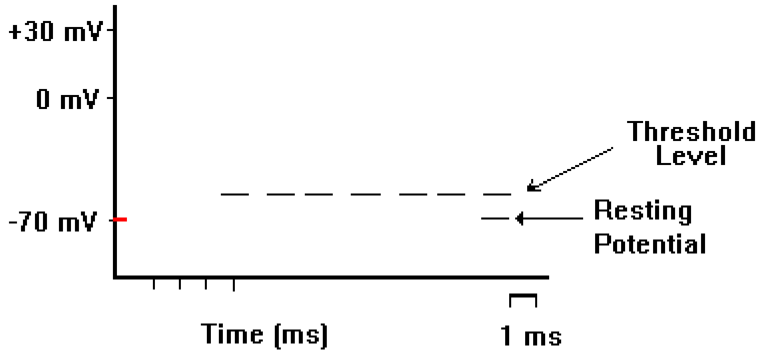
What Does This Have to Do With the Brain?

Biological inspiration





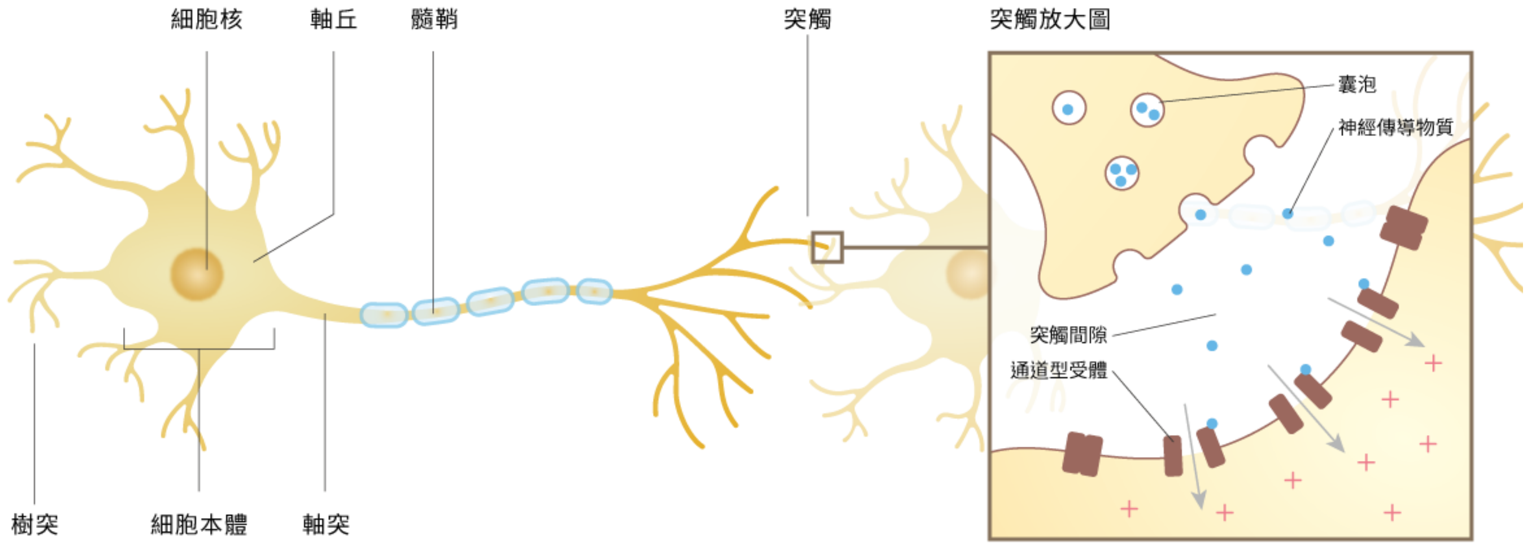


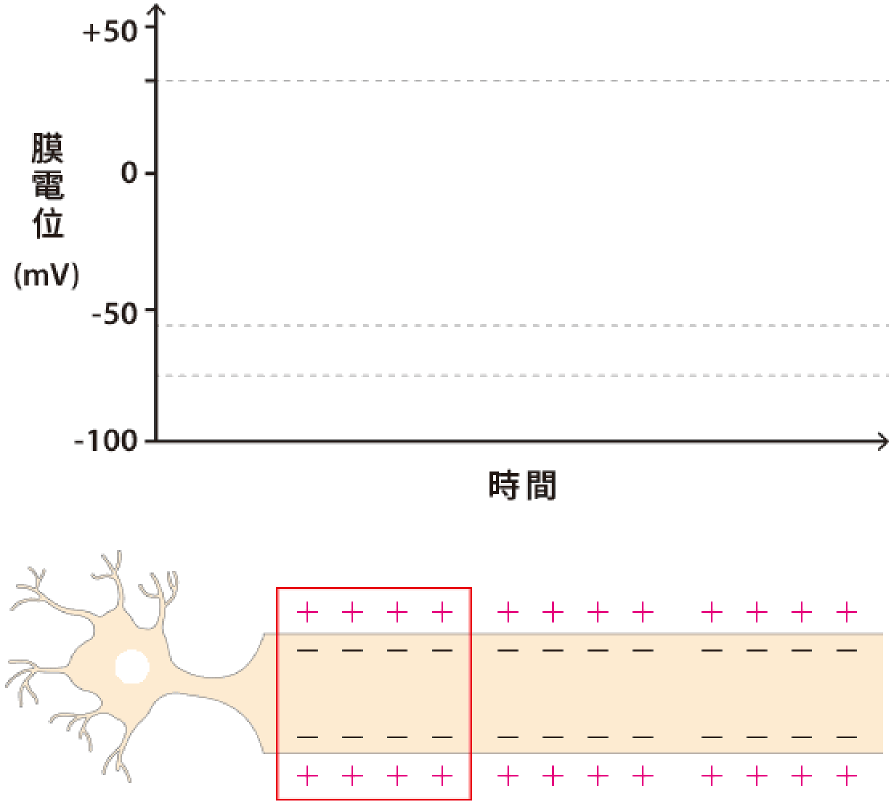


Resting Membrane Potential
Action Potential

What Does This Have to Do With the Brain?

Biological inspiration



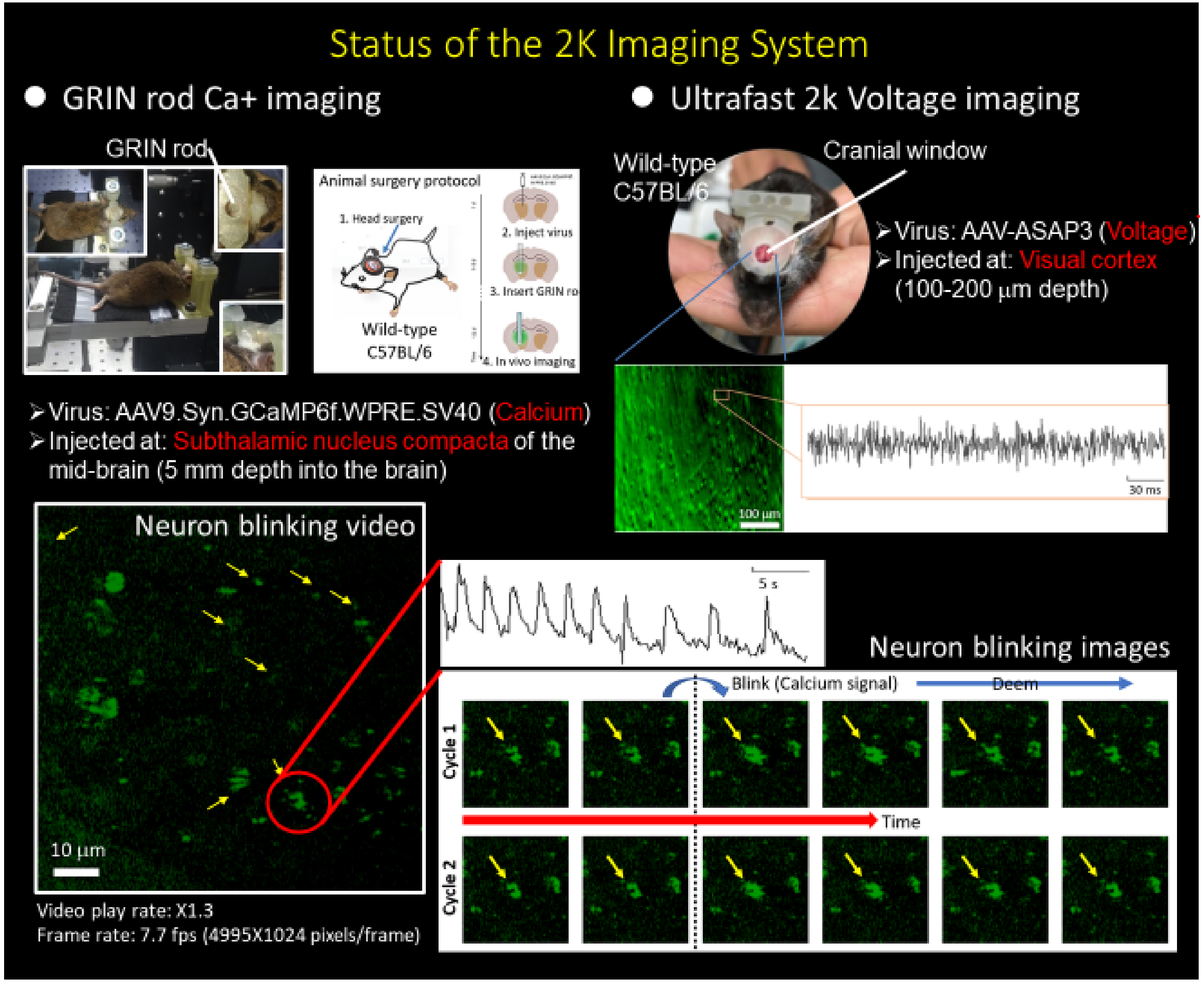


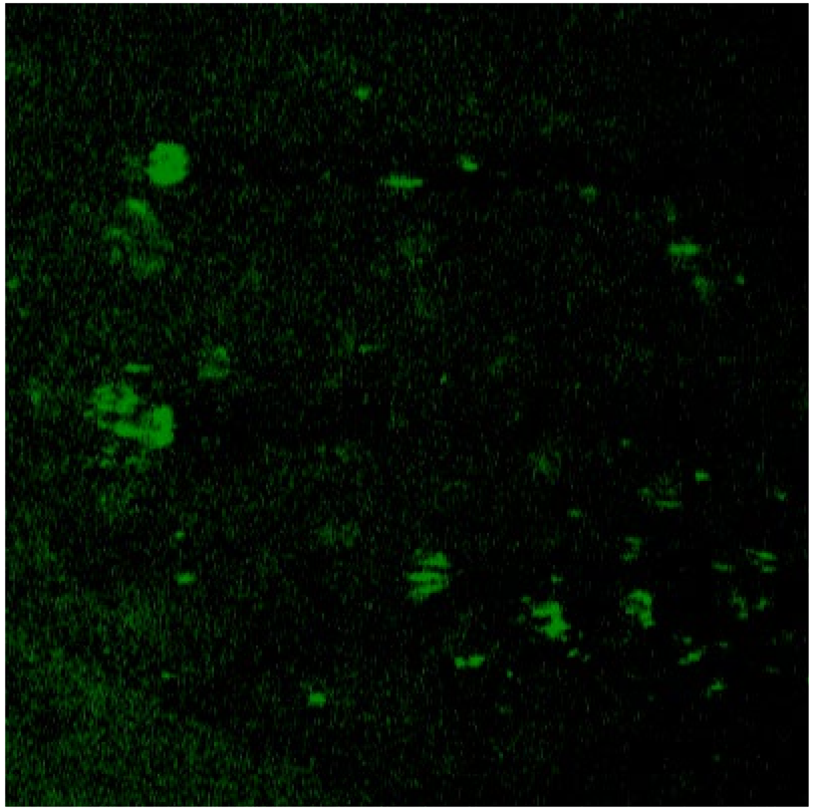
動作電位產生四階段：(1) 極化 - 神經細胞在休息狀態時膜內帶負電、膜外帶正電，靜止膜電位約 -70 mV。 (2) 去極化 - 細胞接受刺激或動作電位傳遞，細胞膜上的電位依賴性鈉離子通道部份開啟，鈉離子進入細胞內，膜電位上升。 (3) 膜電位上升超過閾值，引發大量鈉離子通道開啟，膜電位更正，達動作電位高峰。 (4) 再極化 - 鈉離子通道迅速關閉，鉀離子通道開啟、鉀離子離開細胞。 (5) 過極化 - 鉀離子通道延遲關閉，膜電位降至靜止膜電位以下。最後，鉀離子通道關閉，膜電位回復到極化狀態。（動畫來源：國研院動物中心）

https://www.narlabs.org.tw/xcscience/cont?xsmsid=0I148638629329404252&sid=0J193509885517004464

What Does This Have to Do With the Brain?

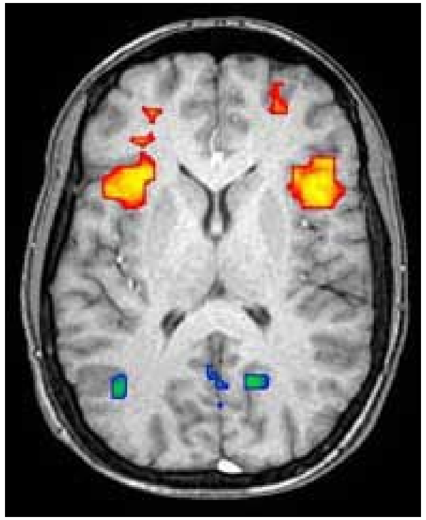
Two-Photon fluorescence brain imaging

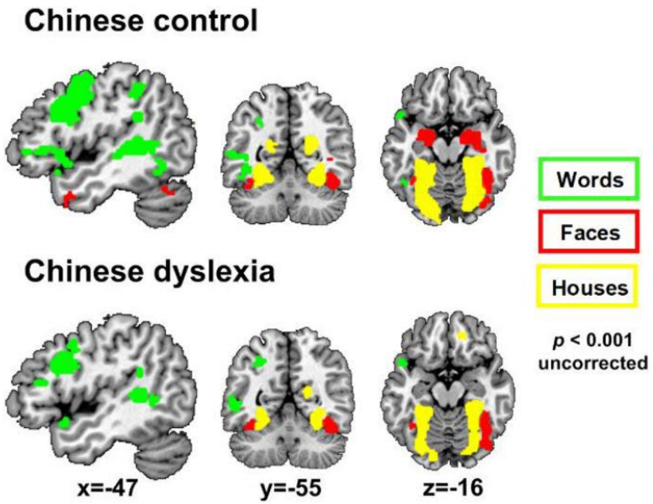


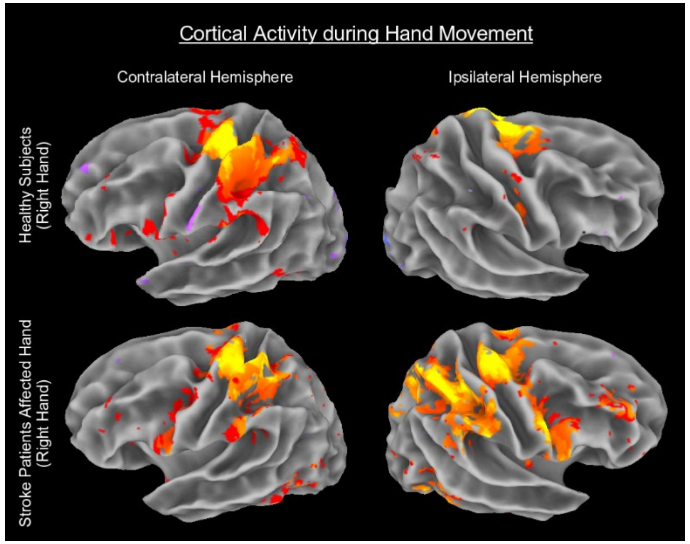


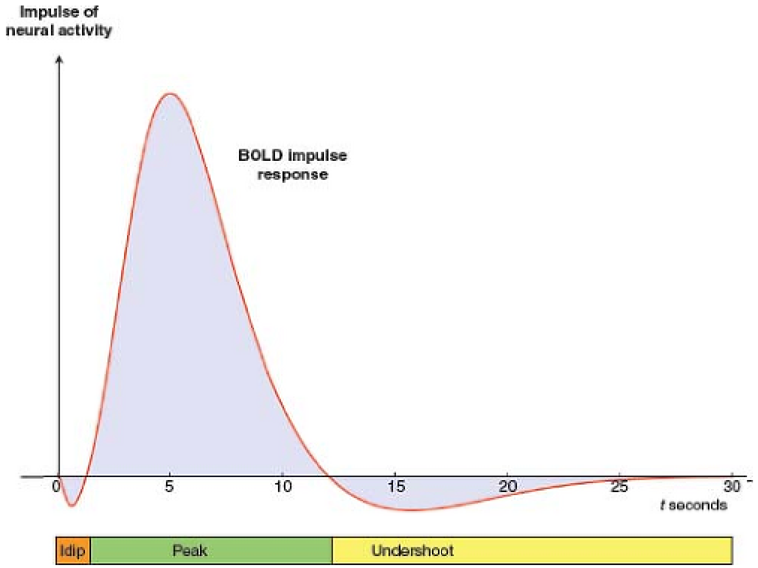
What Does This Have to Do With the Brain?

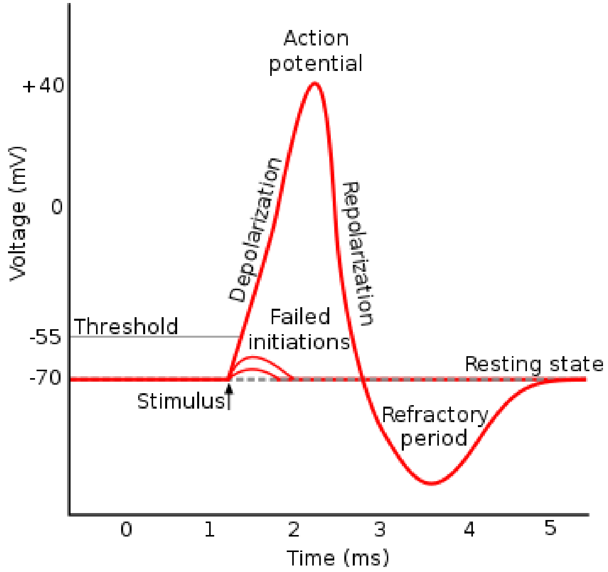
Functional Magnetic Resonance Imaging (Functional MRI)









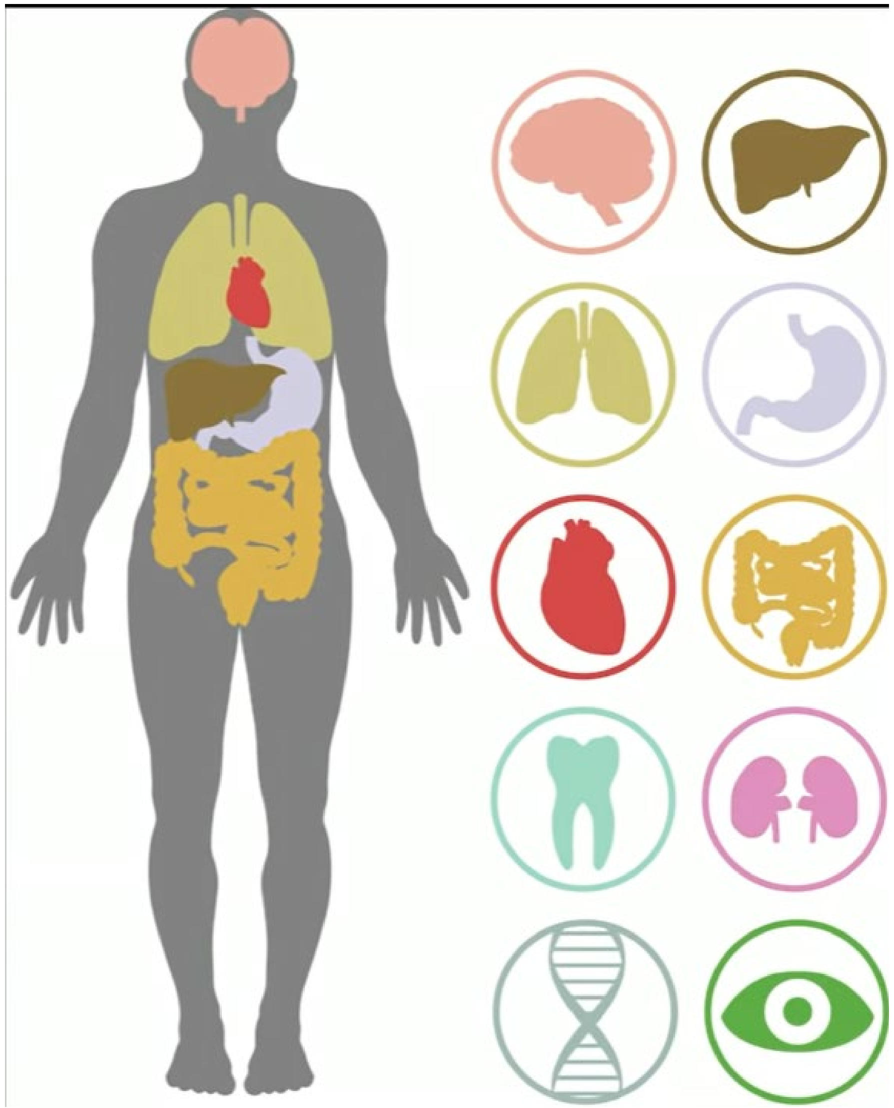


What Does This Have to Do With the Brain?

Energy consumption

- Brain mass: 2% of body mass
- Energy Consumption per day: 2000 kCal
- Energy Consumption of the brain per day?

25%!



https://youtu.be/_7_XH1CBzGw

Next Lecture

- Setting Up Your Machine Learning Application (Course 2 Week 1)

- Optimizing Algorithms (Course 2 Week 2)

Next:

Lab Practice

Build an ANN