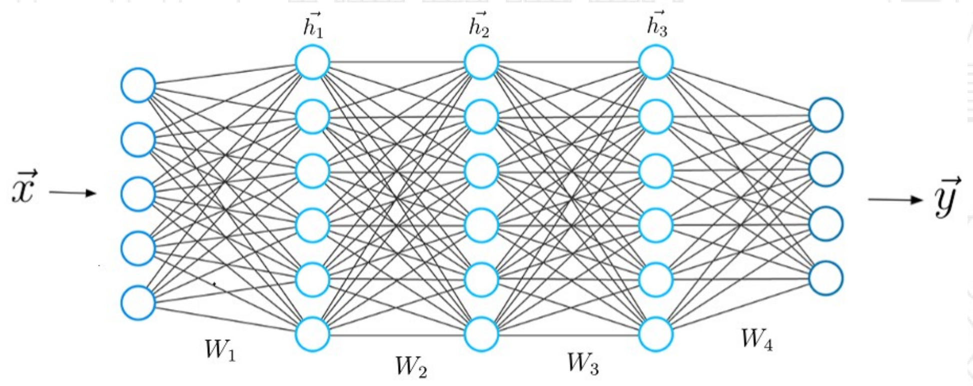
Module 3

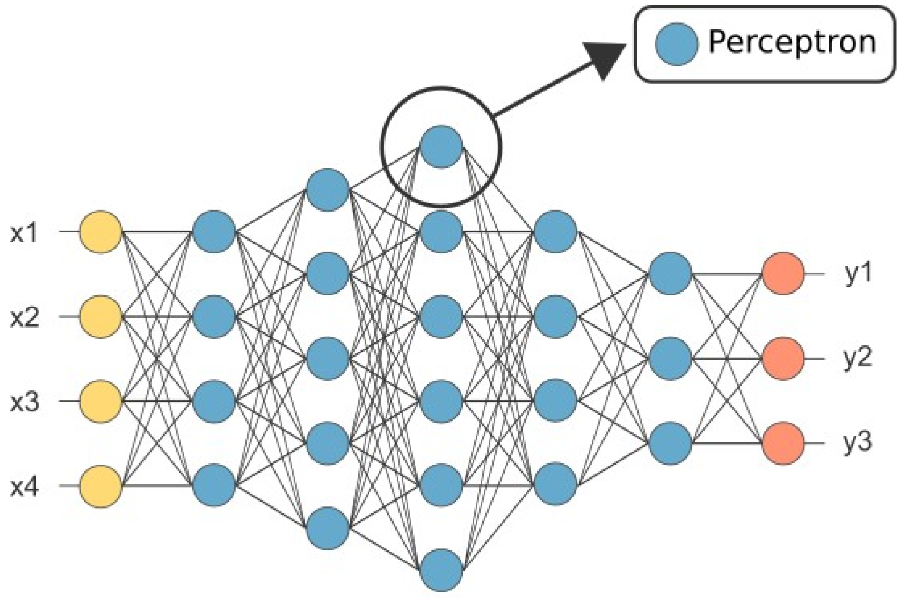
Deep Learning: Feed-Forward Architectures

DL: Feed-Forward Architectures



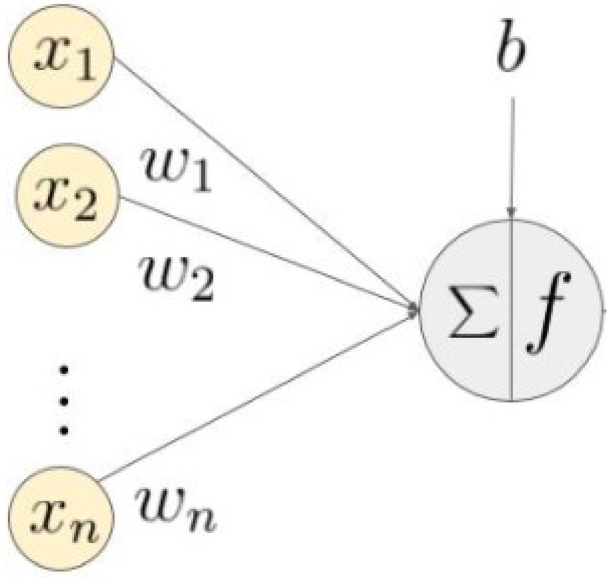
Feed-Forward (MLP) Architecture

- Flow is unidirectional
- Information flows forward
- Input nodes ▻ hidden nodes ▻ output nodes
- Composed of neurons (e.g., perceptrons)



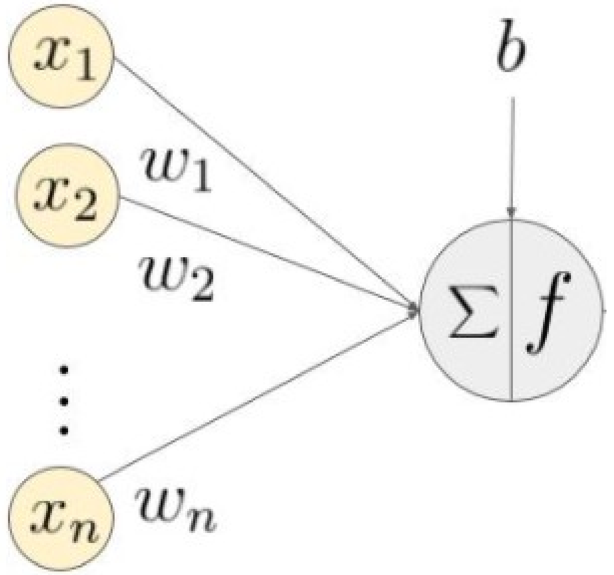
Neurons

- Composed of a sum function and an activation function
- SUM: performs linear function of inputs plus bias: $z = x^\top w + b$
- ACTIVATION: modifies the sum (usually non-linear): $f(z)$



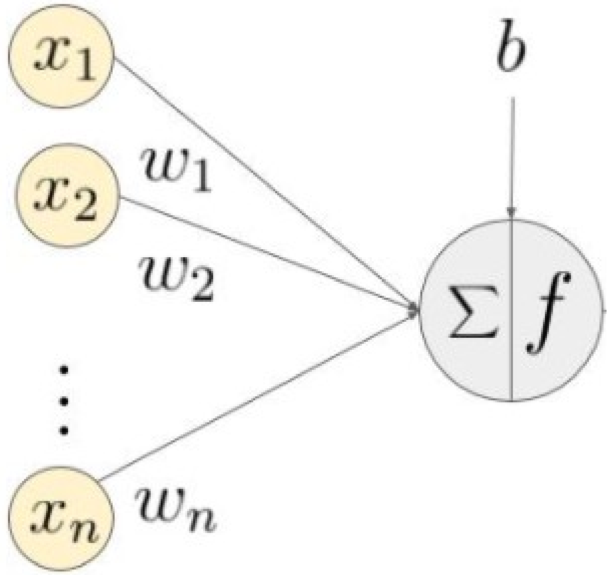
Neurons

- SUM: performs linear function of inputs plus bias: $z = x^{\top}w + b$
- ACTIVATION: modifies the sum (usually non-linear): $f(z)$
- A single neuron can approximate a linear regression model



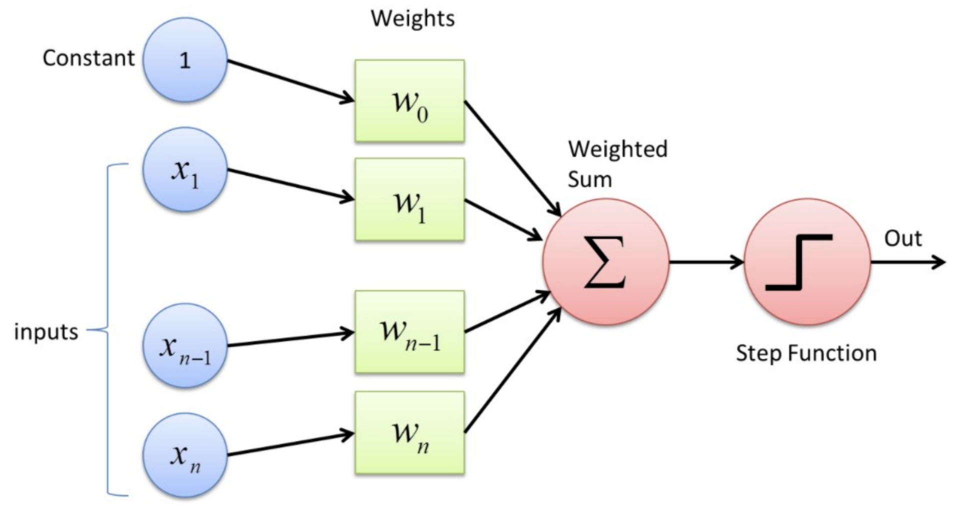
Neurons

- SUM: performs linear function of inputs plus bias: $z = x^{\top} w + b$
- ACTIVATION: modifies the sum (usually non-linear): $f(z)$
- A single neuron can approximate a logistic regression model



The Perceptron

- The Perceptron is the most basic NN unit.
- It contains a single unit (neuron)
- Activation is a threshold logical unit (TLU).

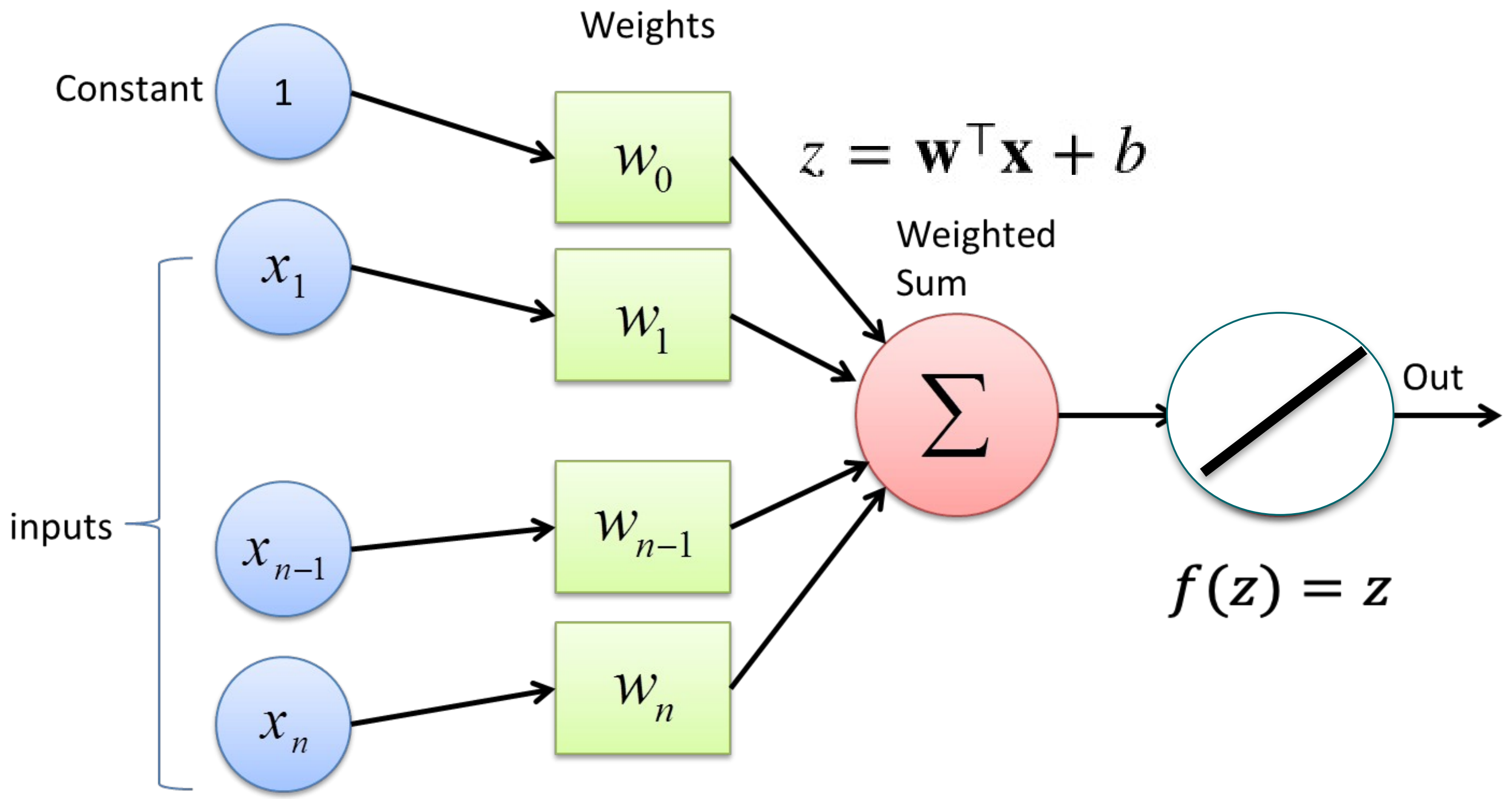


$$\text{heaviside}(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

$$\text{sgn}(z) = \begin{cases} -1 & \text{if } z < 0 \\ 0 & \text{if } z = 0 \\ +1 & \text{if } z > 0 \end{cases}$$

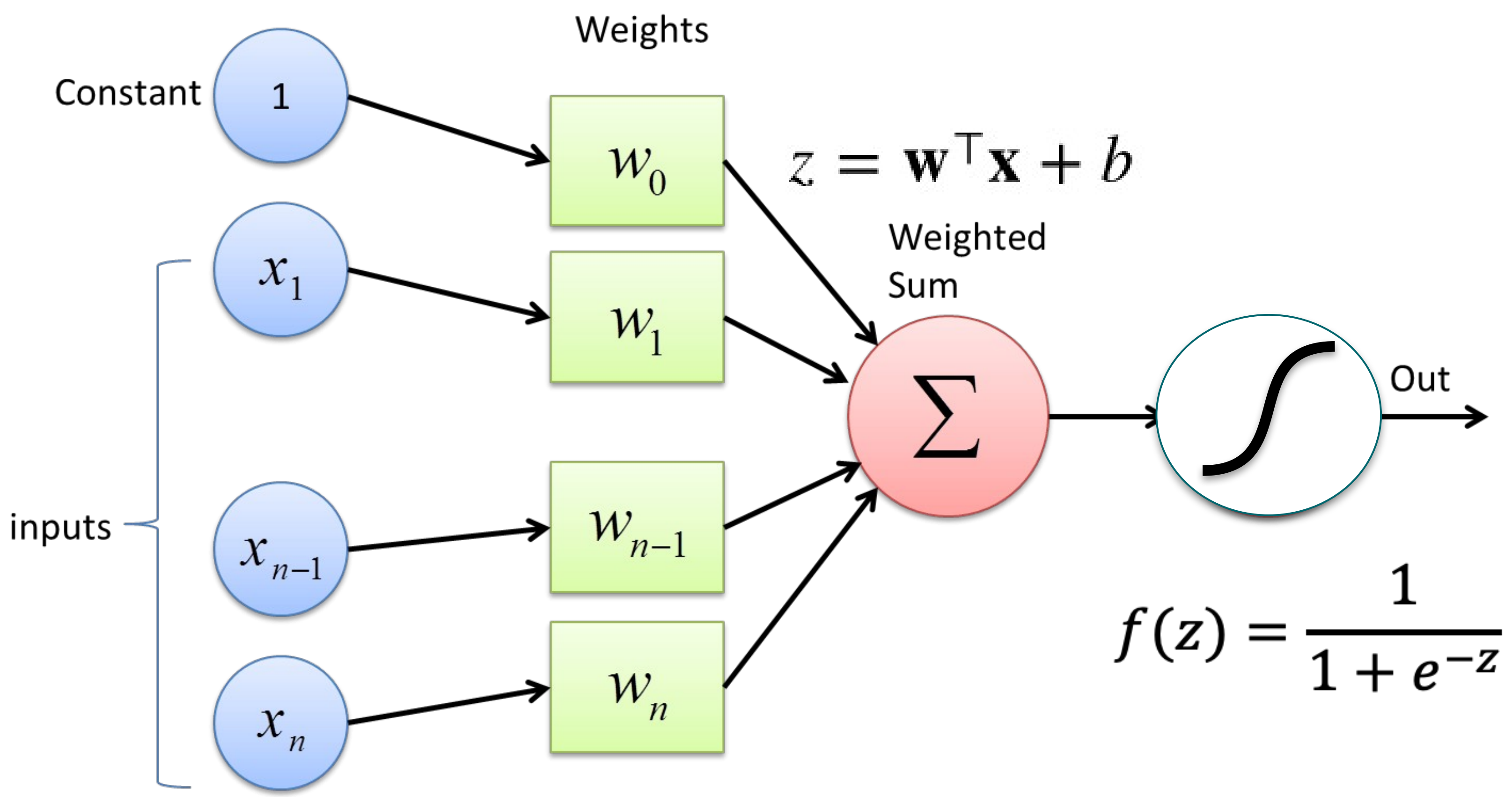
Linear Regression as a Perceptron

- The chosen activation is would be “linear”



Logistic Regression as a Perceptron

- The chosen activation is would be "sigmoid"

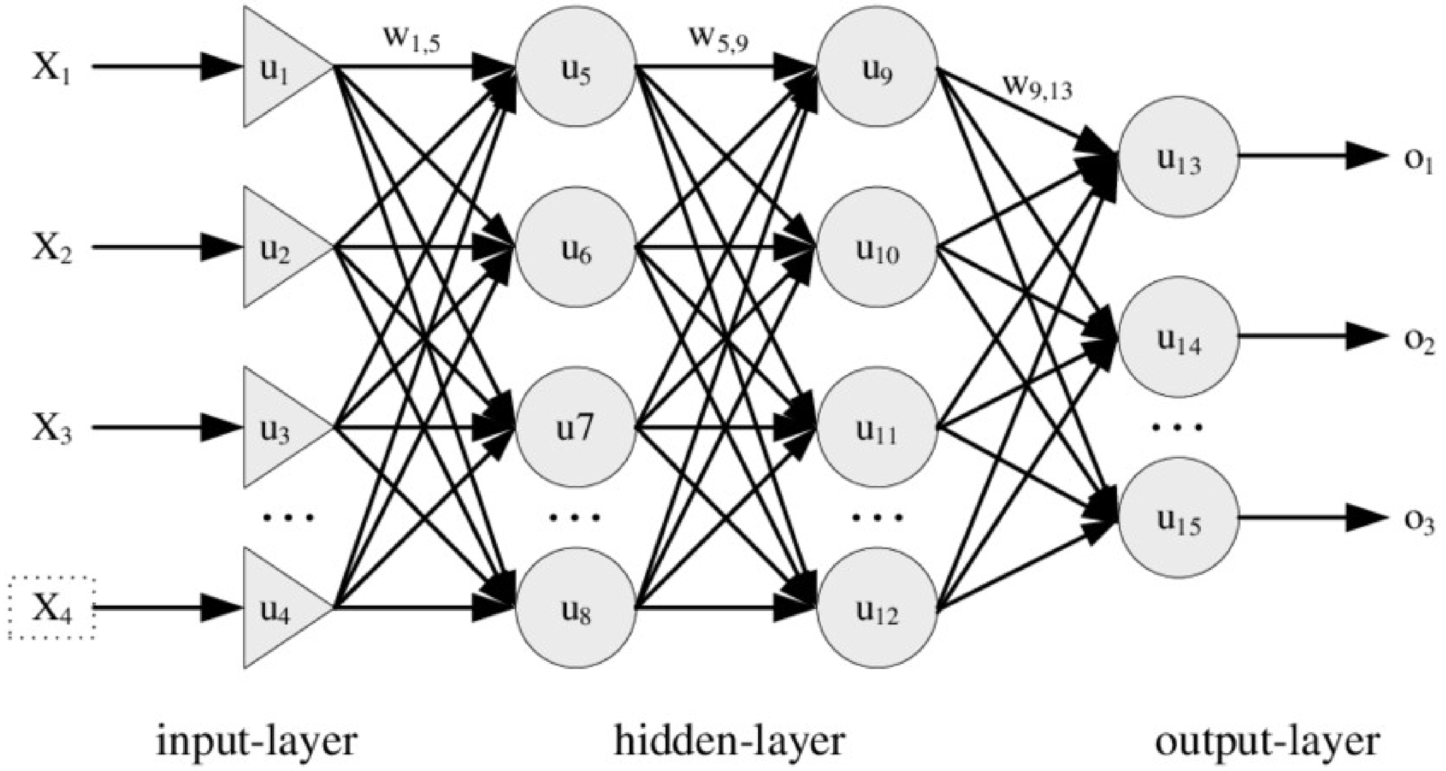


$$z = \mathbf{w}^\top \mathbf{x} + b$$

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

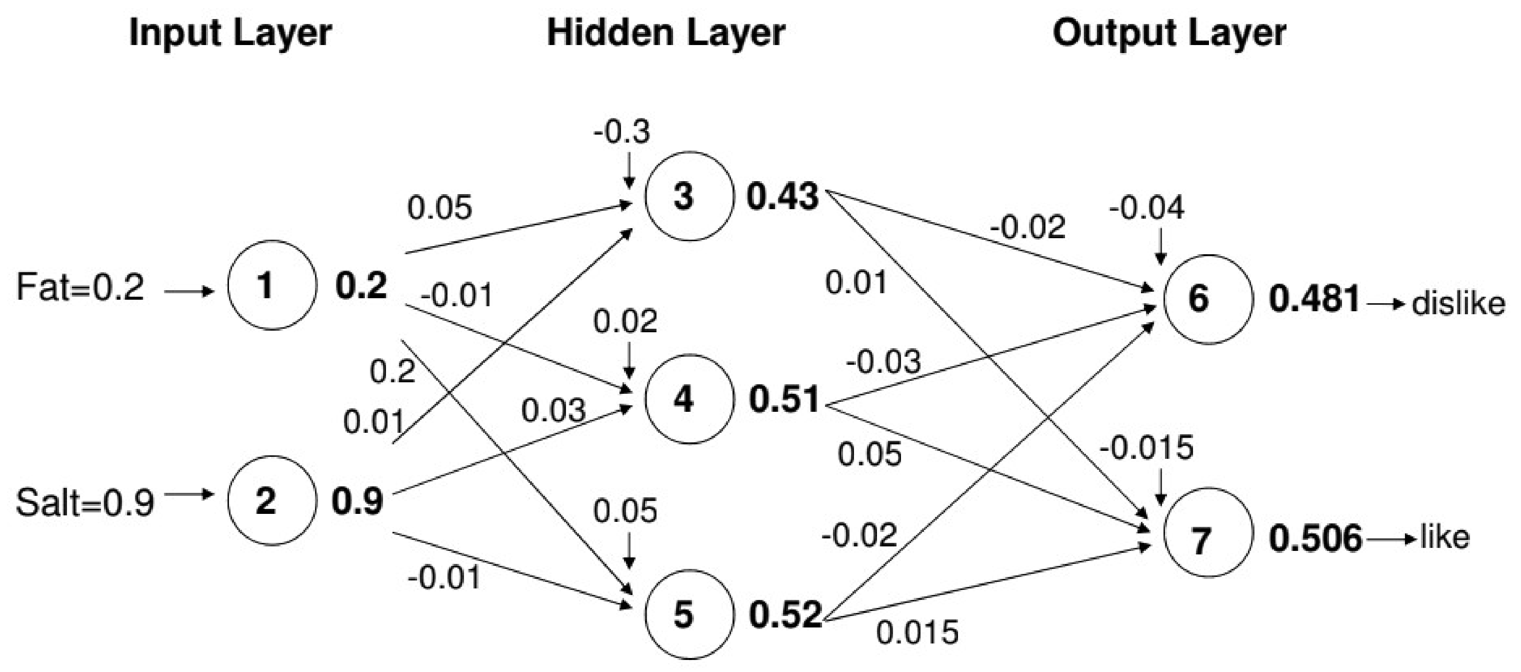
Multi-Layer Perceptron

- Connect neurons in a forward approach



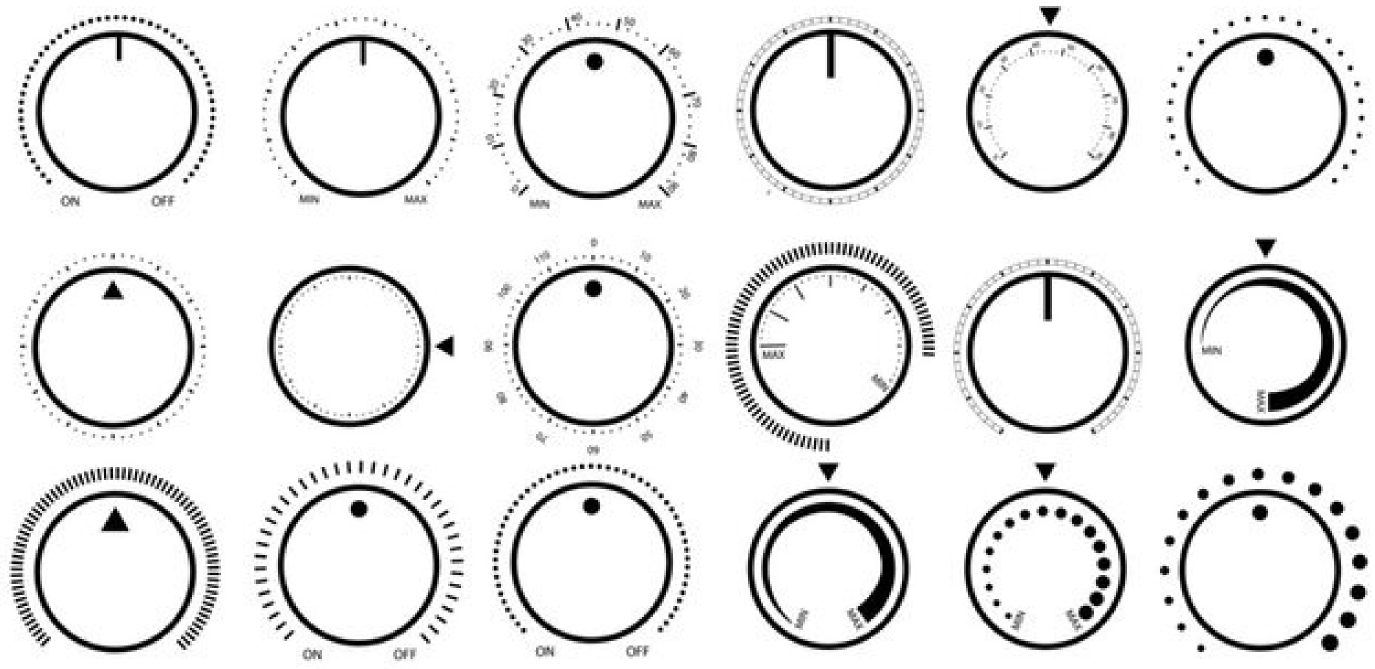
Multi-Layer Perceptron

- Illustration (sigmoid): Input layer ⊳ hidden layer ⊳ output layer



Multi-Layer Perceptron

- Imagine each weight and bias is a knob we need to tune

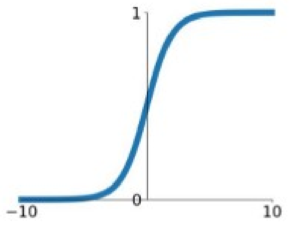


Common Activation Functions

- We need activation functions with easily computable derivatives
- Goal: amplify, lessen or cut the signal

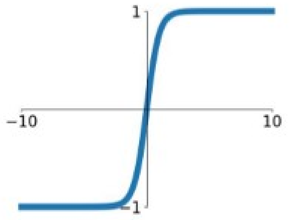
Sigmoid

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



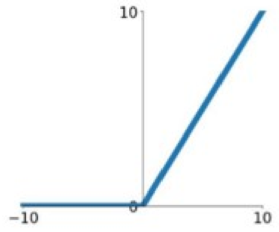
tanh

$\tanh(x)$



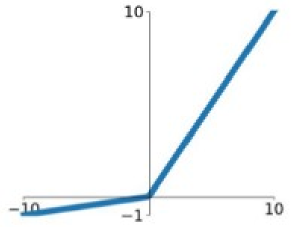
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

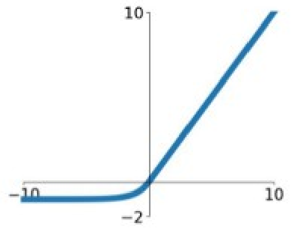


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

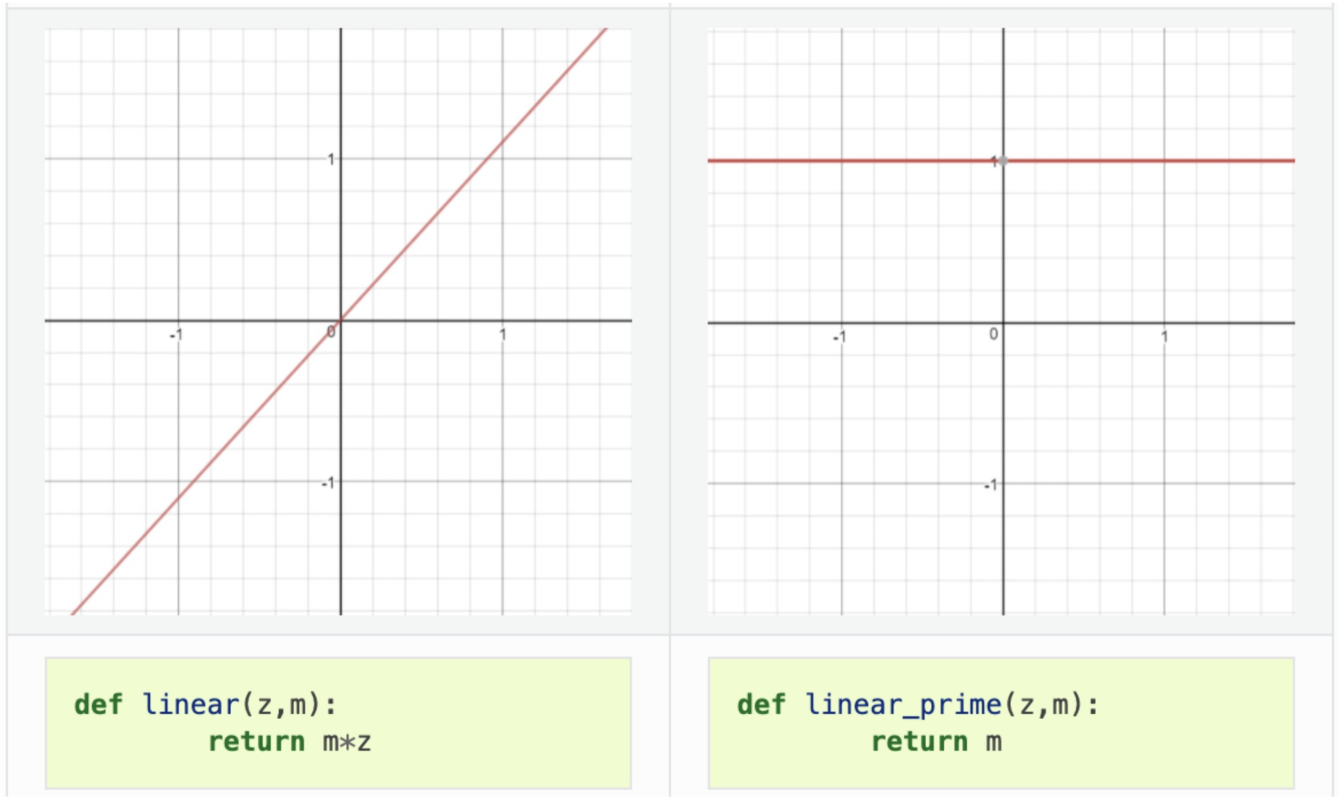
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Activation: “linear”

- Given an input signal z, it returns $f(z) = z$
- Derivative $f'(z) = 1$

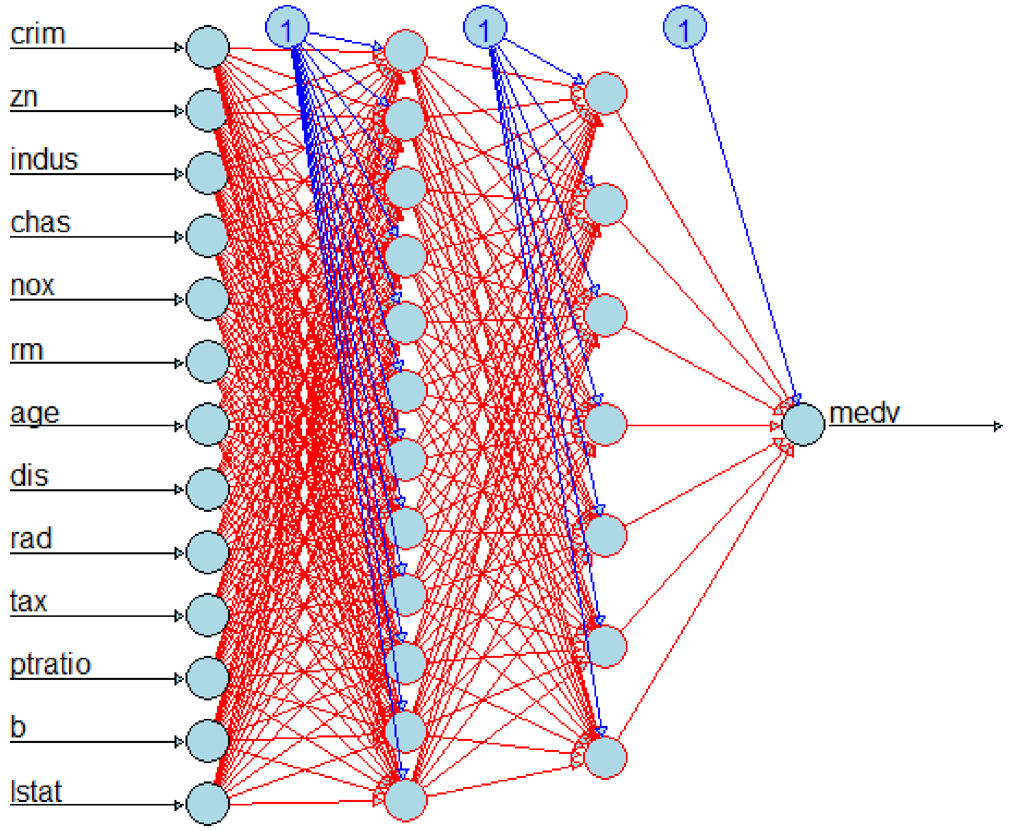


```
def linear(z,m):
        return m*z
```

```
def linear_prime(z,m):
        return m
```

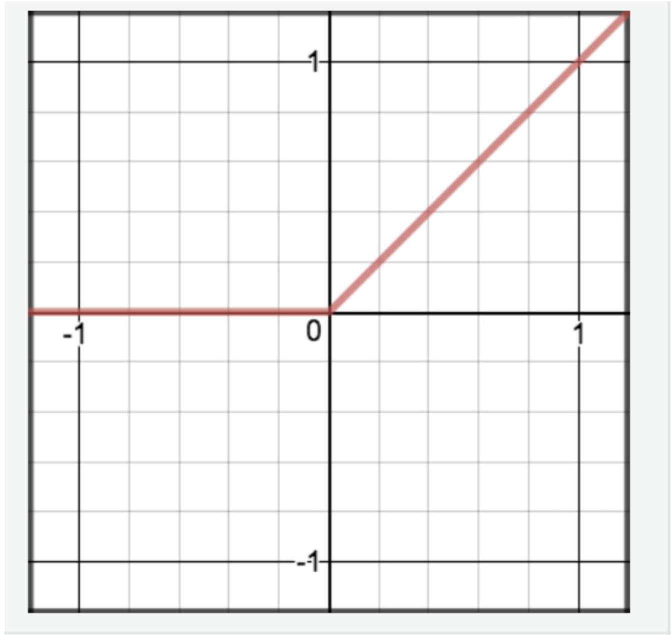
Activation: “linear”

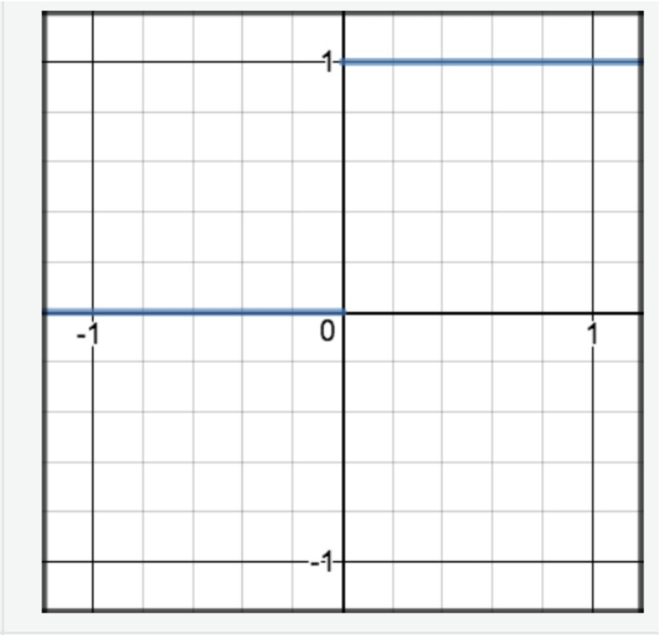
- Useful for the output layer (numeric response)



Activation: Rectified Linear Unit "ReLu"

- Given an input signal z, it returns $f(z) = \max(0, z)$
- Derivative $f'(z) = 0$ for $z \leq 0$ and $f'(z) = 1$ for $z > 0$



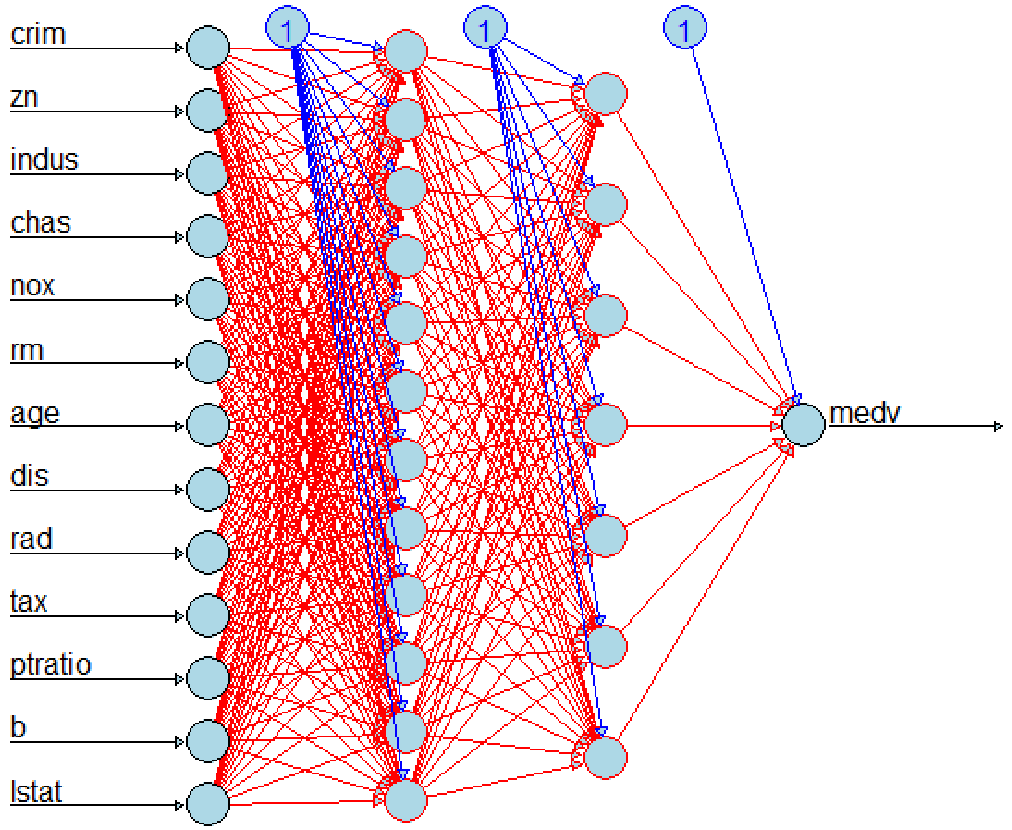


```
def relu(z):
  return max(0, z)
```

```
def relu_prime(z):
  return 1 if z > 0 else 0
```

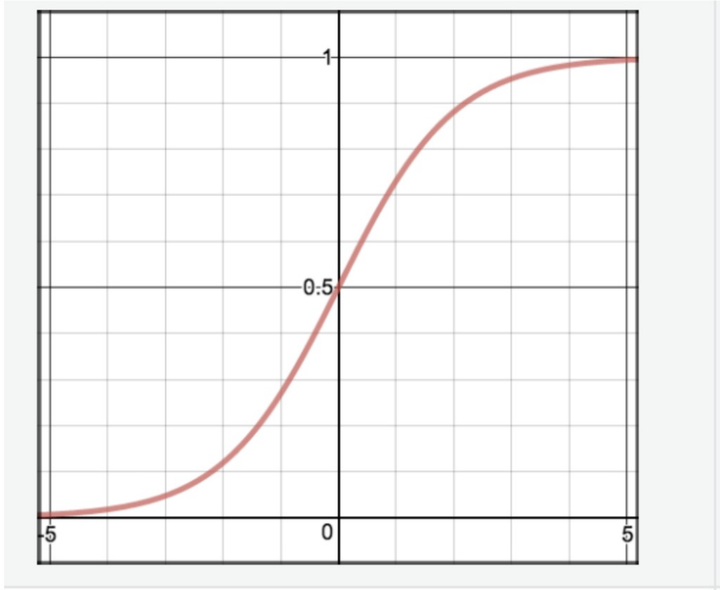
Activation: “ReLu”

- Useful for hidden layers and output layer (positive numeric)

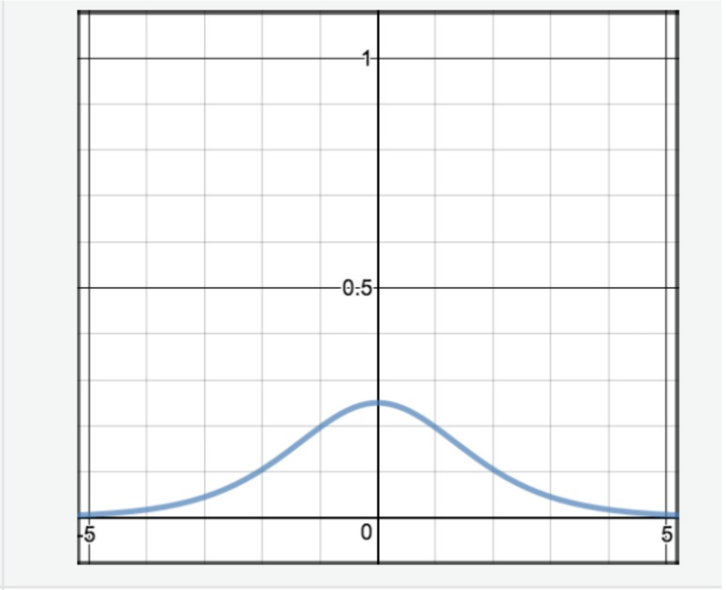


Activation: Sigmoid / Logistic

- Given an input signal z, it returns $f(z) = \frac{1}{1+e^{-z}}$
- Derivative $f'(z) = f(z)(1 - f(z))$



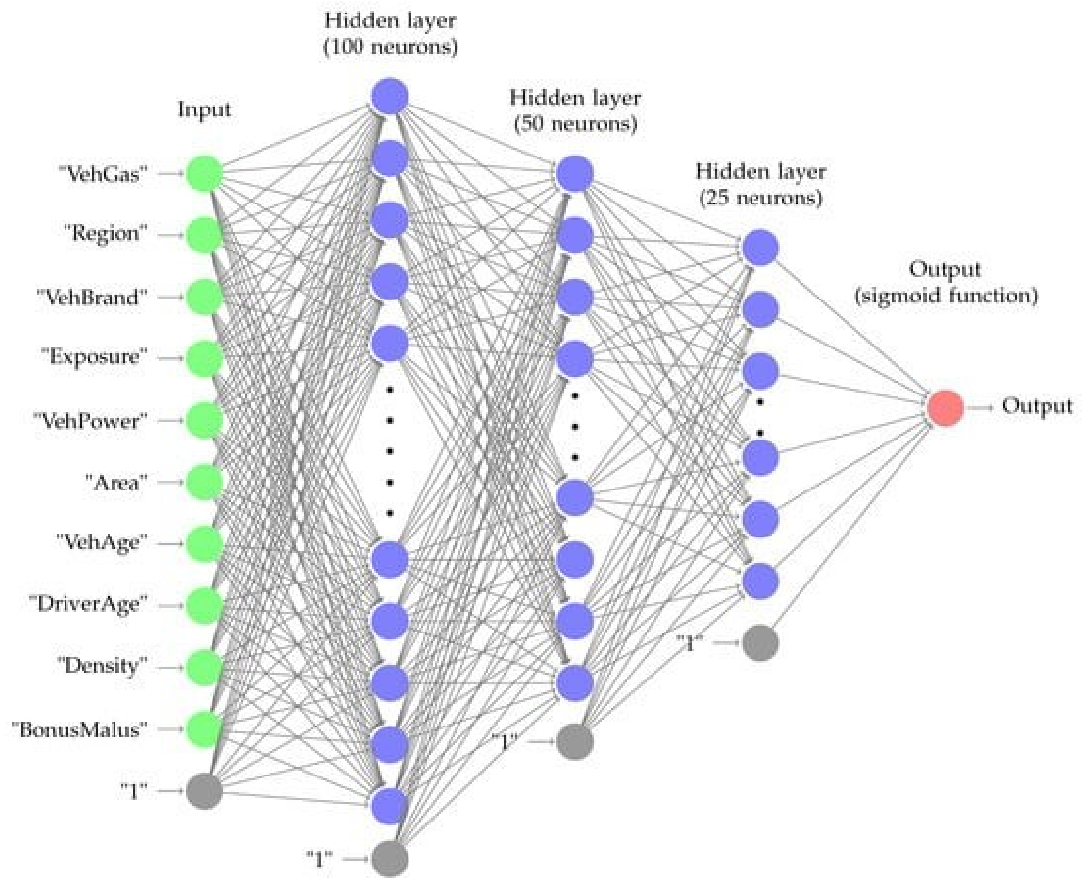
```
def sigmoid(z):
  return 1.0 / (1 + np.exp(-z))
```



```
def sigmoid_prime(z):
  return sigmoid(z) * (1-sigmoid(z))
```

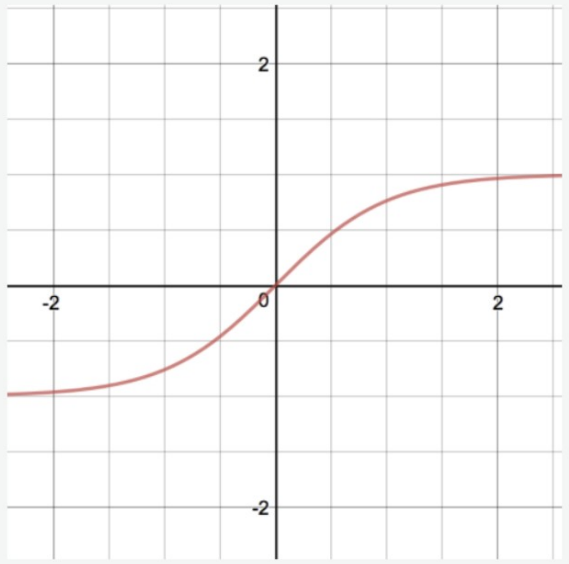
Activation: “Sigmoid”

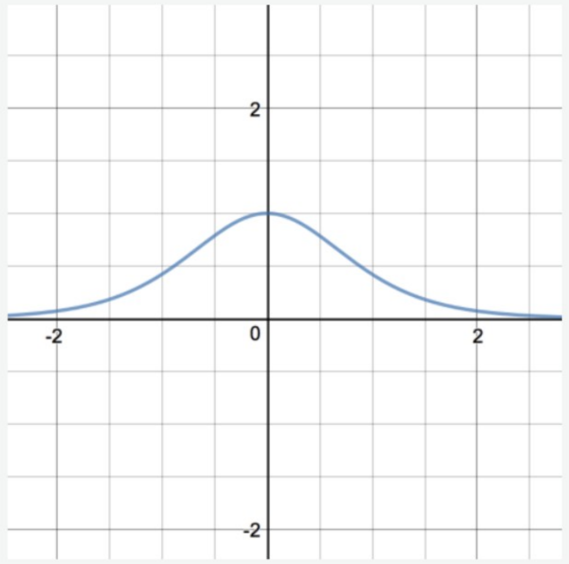
- Useful for hidden layers and output layer (binary response)



Activation: hyperbolic tangent ”tanh”

- Given an input signal z, it returns $f(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$
- Derivative $f'(z) = 1 - f(z)^2$



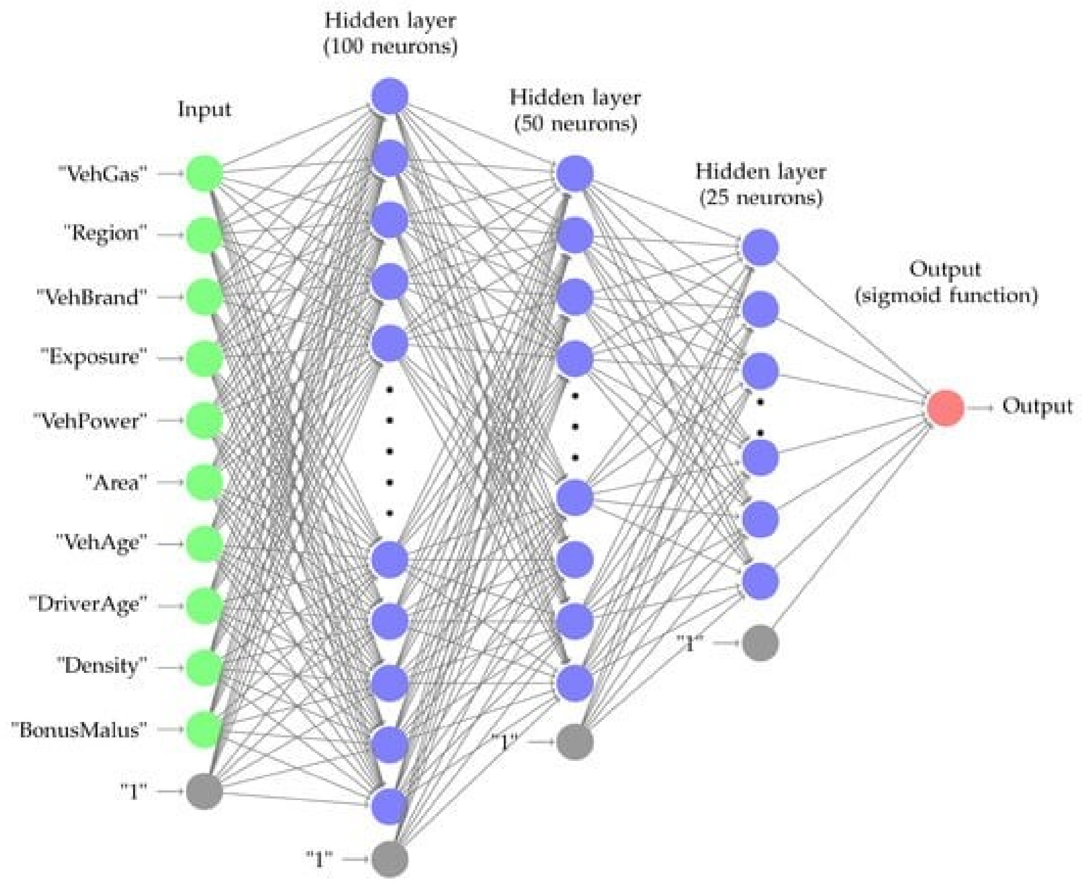


```
def tanh(z):
        return (np.exp(z) - np.exp(-z)) / (np.exp(z) + np.exp(-z))
```

```
def tanh_prime(z):
        return 1 - np.power(tanh(z), 2)
```

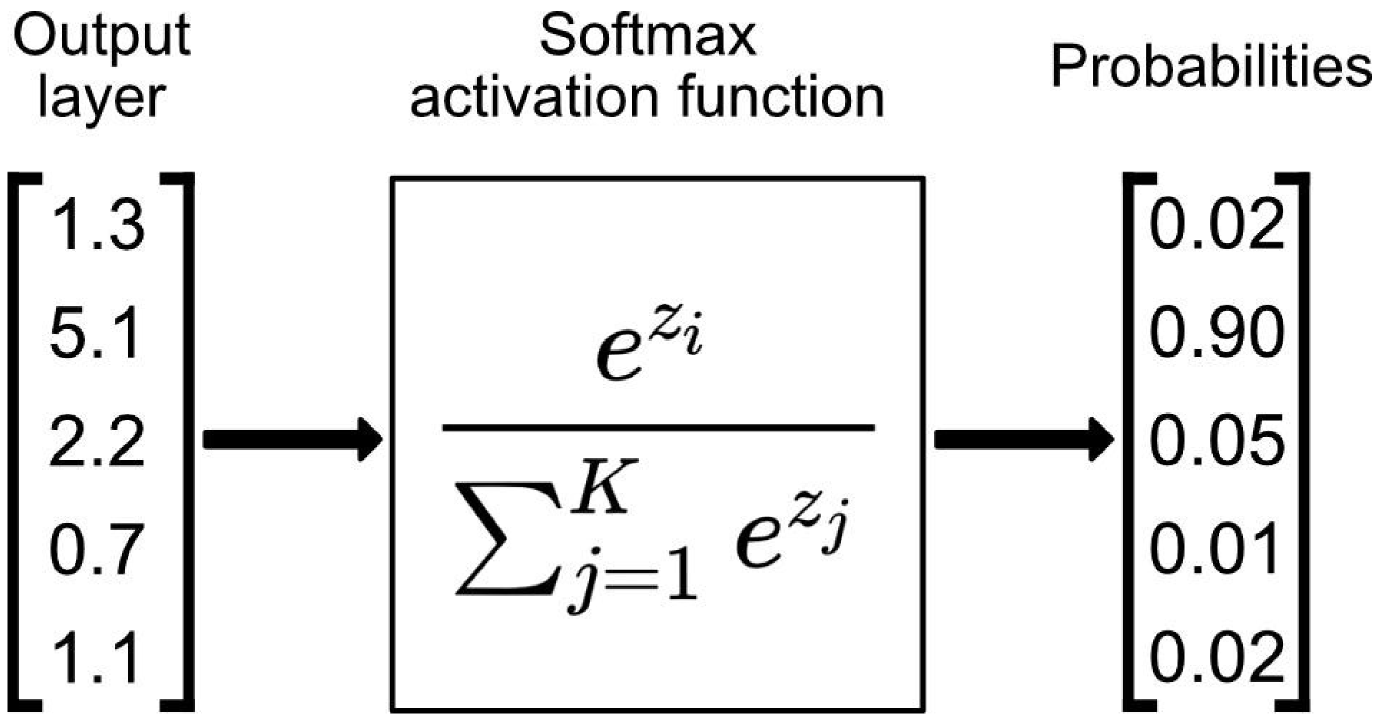
Activation: “tanh”

- Useful for hidden layers and output layer (binary response)



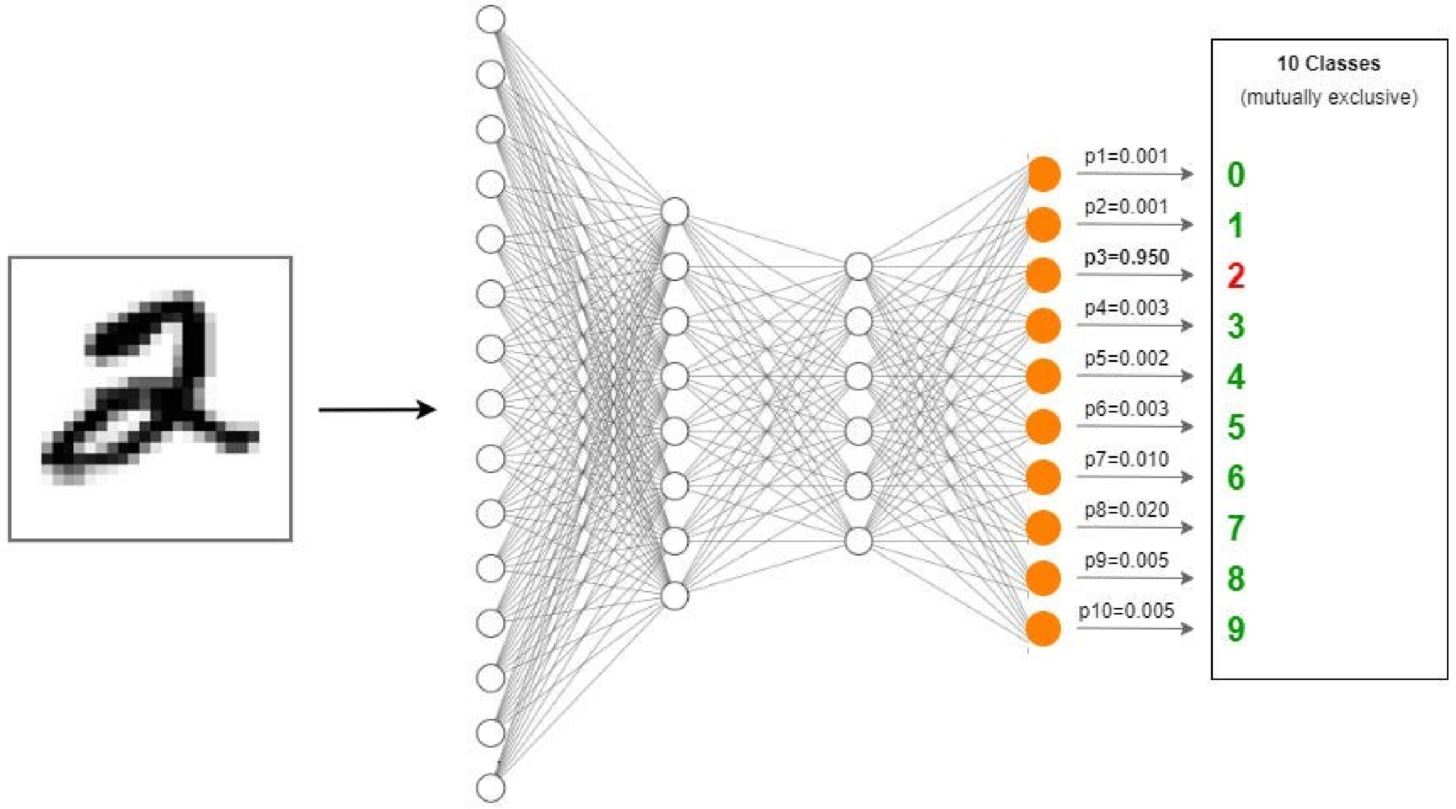
Activation: ”softmax”

- Given an input signal $\boldsymbol{z}$, it returns $f(\boldsymbol{z}) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$


Output layer
Softmax activation function
Probabilities
1.3
5.1
2.2
0.7
1.1
$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$
0.02
0.90
0.05
0.01
0.02

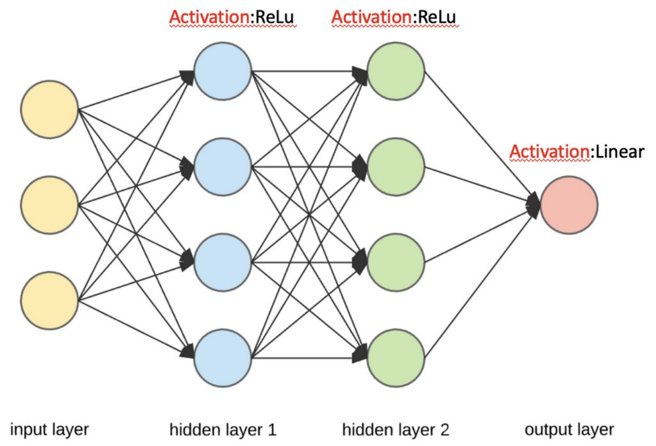
Activation: “softmax”

- Useful for output layer (multi-class response)



Number of Estimated Parameters

❑ Useful to know when we might overfit



- Input to Hidden Layer 1: 3 variables $\times$ 4 units + 4 bias terms = 16 weights
- Hidden Layer 1 to Hidden Layer 2: 4 units $\times$ 4 units + 4 bias terms = 20 weights
- Hidden Layer 2 to Output Layer: 4 units $\times$ 2 units + 2 bias terms = 10 weights

Total: 16+20+10 = 46 parameters

Python