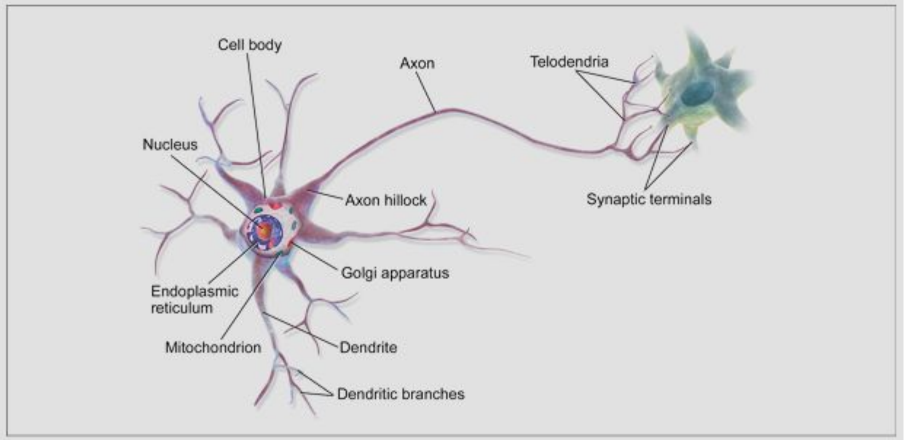
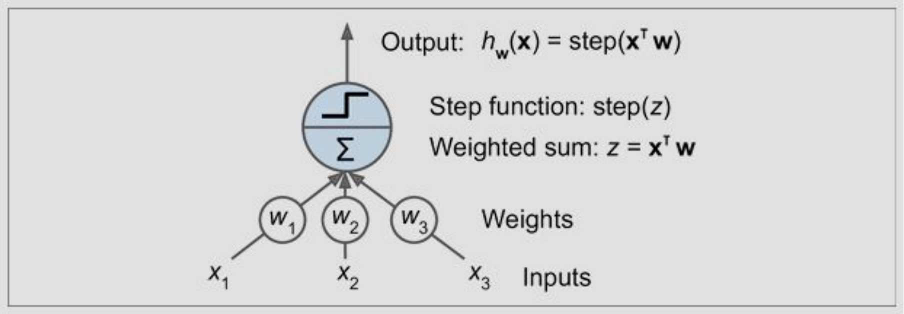
Introduction to Artificial Neural Networks with Keras

From Biological Neurons to Deep Learning with TensorFlow

From Biological to Artificial Neurons

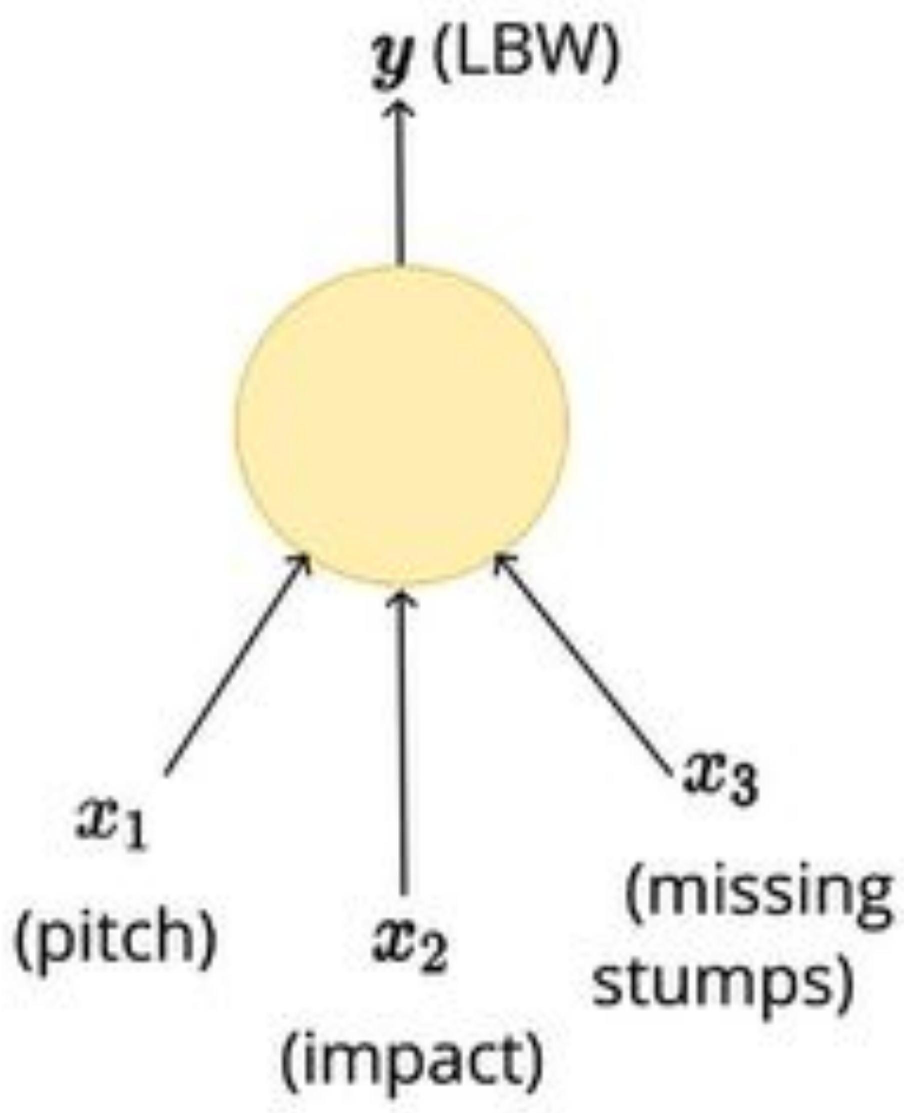
- **Biological Neuron Inspiration**
  - Dendrites (inputs)
  - Cell body (processing)
  - Axon (output)
- **Artificial Neuron (Perceptron)**
  - Inputs: $x_1, x_2, \ldots, x_n$
  - Weights: $w_1, w_2, \ldots, w_n$
  - Bias: b
  - Activation Function: ϕ
  - Output: $y = \phi(w \cdot x + b)$





McCulloch Pitts Neuron

-
- $\hat{y} = \begin{cases} 1, if \ \sum_{i=1}^{n} x_i \geq b \\ 0, \quad\quad otherwise \end{cases}$
- $loss = \sum_i (y_i - \hat{y_i})^2$
- Boolean input
- Boolean output
- Fixed slope
- Few possible intercepts (b



The Perceptron

- **Mathematical Model**

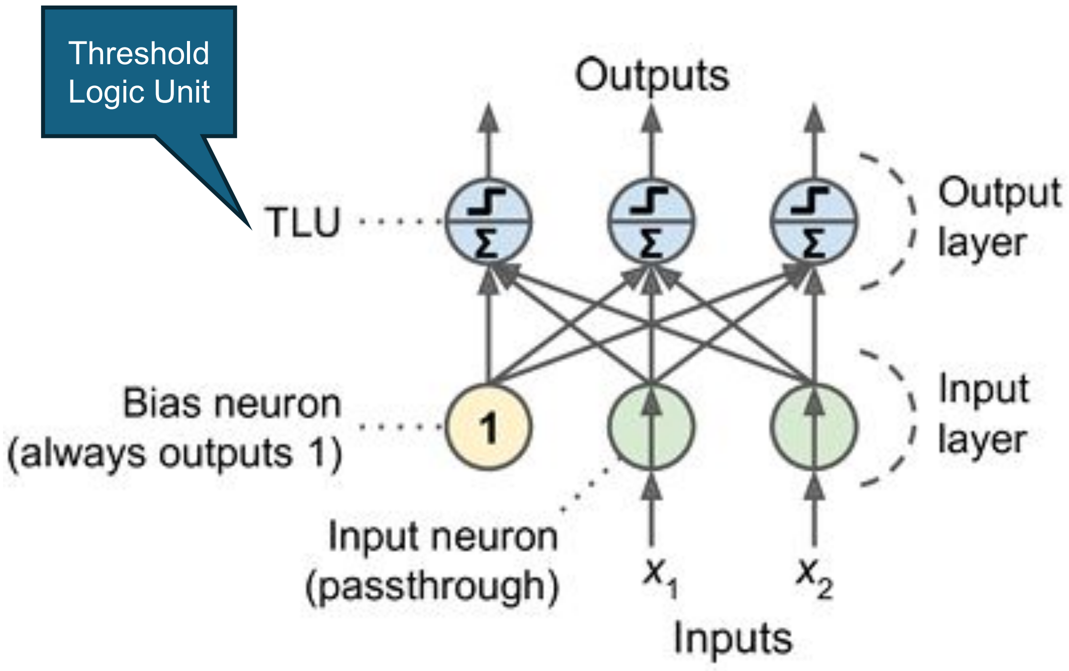
$$z = w \cdot x + b$$

$$y = \phi(z)$$

- **Step Function (Heaviside)**

$$\phi(z) = \begin{cases} 1, & if\ z \geq 0 \\ 0, & otherwise \end{cases}$$

- **Learning algorithm**
  - If $\boldsymbol{x} \in \boldsymbol{P}$ and $\sum_{i=1}^{n} w_i x_i < 0$
    - $\boldsymbol{w} = \boldsymbol{w} + \boldsymbol{x}$
  - If $\boldsymbol{x} \in \boldsymbol{N}$ and $\sum_{i=1}^{n} w_i x_i \geq 0$
    - $\boldsymbol{w} = \boldsymbol{w} - \boldsymbol{x}$

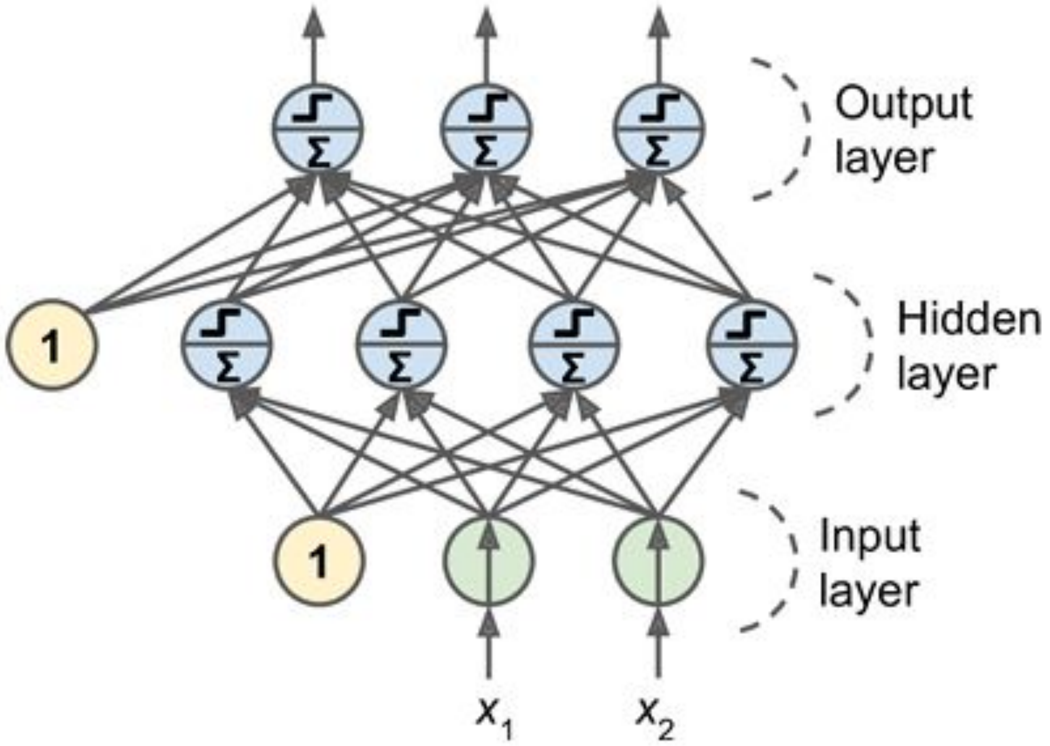


Multilayer Perceptron (MLP)

- **Architecture:**
  - Input Layer
  - One or More Hidden Layers
  - Output Layer
- **Feedforward Process**

$$a^{(l)} = \phi\left(W^{(l)} a^{(l-1)} + b^{(l)}\right)$$

- **Universal Approximation Theorem:** An MLP can approximate any continuous function



Backpropagation

- **Goal:** Minimize cost function $J(W, b)$
- **Process:**
  - Forward pass: compute output
  - Backward pass: compute gradients using chain rule
  - Update weights and biases using gradient descent
- **Gradient Update Rule:**

$$W \leftarrow W - \eta \frac{\partial J}{\partial W}$$

$$b \leftarrow b - \eta \frac{\partial J}{\partial b}$$

Chain Rule

- Let's take a simple **2-layer feedforward network**:

$$z_1 = W_1 x + b_1$$
$$a_1 = f(z_1)$$
$$z_2 = W_2 a_1 + b_2$$
$$\hat{y} = g(z_2)$$

- $Loss\ (L) = Loss(\hat{y}, y)$
- We need

$$\frac{\partial L}{\partial W_1}, \frac{\partial L}{\partial b_1}, \frac{\partial L}{\partial W_2}, \frac{\partial L}{\partial b_2}$$

Backpropagation Using Chain Rule

- **Step 1: Output layer gradient**
  - $\delta_2 = \frac{\partial L}{\partial z_2} = \frac{\partial L}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial z_2}$
  - $\frac{\partial L}{\partial W_2} = \delta_2 a_1^T$
  - $\frac{\partial L}{\partial b_2} = \delta_2$

- **Step 2: Hidden layer gradient**
  - $\delta_1 = \frac{\partial L}{\partial z_1} = \frac{\partial L}{\partial z_2} \cdot \frac{\partial z_2}{\partial a_1} \cdot \frac{\partial a_1}{\partial z_1}$
  - $\frac{\partial L}{\partial W_1} = \delta_1 x^T$
  - $\frac{\partial L}{\partial b_1} = \delta_1$

Activation Functions

-
- **Sigmoid**

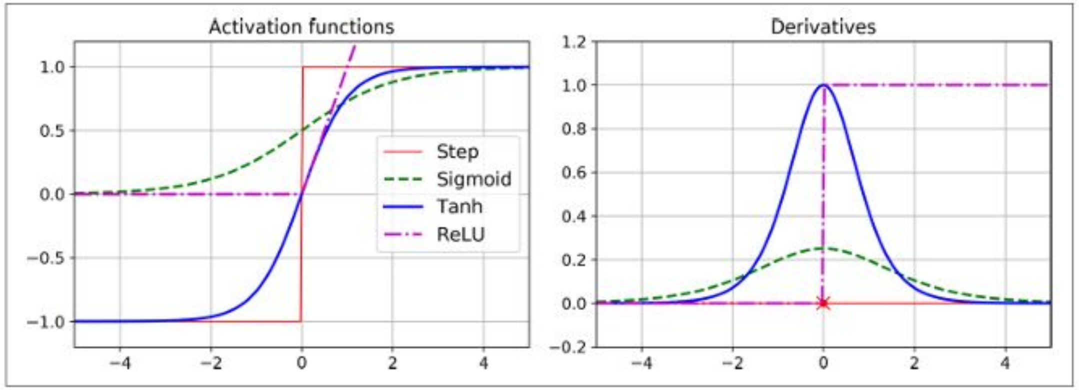
$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

- **Hyperbolic tangent function (Tanh)**

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

- **Rectified Linear Unit function (ReLU)**

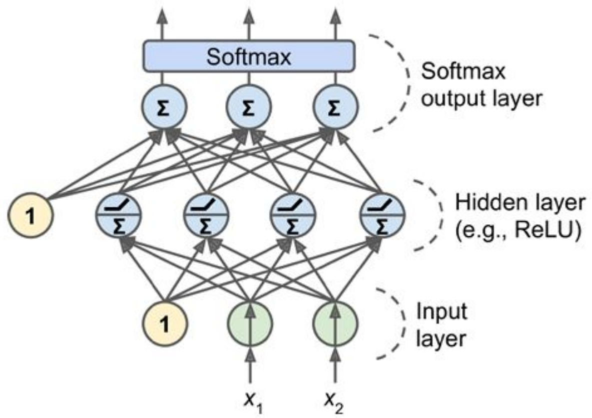
$$ReLU(z) = \max(0, z)$$



Regression and Classification MLPs

- **Softmax (for output layer in classification):**

$$e(z_i) = \frac{e^{z_i}}{\sum_j e^{z_i}}$$



Hyperparameter	Binary classification	Multilabel binary classification	Multiclass classification
Input and hidden layers	Same as regression	Same as regression	Same as regression
# output neurons	1	1 per label	1 per class
Output layer activation	Logistic	Logistic	Softmax
Loss function	Cross entropy	Cross entropy	Cross entropy

Hyperparameter	Typical value
# input neurons	One per input feature (e.g., 28 x 28 = 784 for MNIST)
# hidden layers	Depends on the problem, but typically 1 to 5
# neurons per hidden layer	Depends on the problem, but typically 10 to 100
# output neurons	1 per prediction dimension
Hidden activation	ReLU (or SELU, see Chapter 11)
Output activation	None, or ReLU/softplus (if positive outputs) or logistic/tanh (if bounded outputs)
Loss function	MSE or MAE/Huber (if outliers)

Introduction to Keras

- https://drive.google.com/file/d/17DTPHndXAczQS6cDLOffz0IcVJVwSmd3/view?usp=sharing

Fine-Tuning Hyperparameters

- **Number of Hidden Layers:**
  - Start with 1–2, increase for complex problems
  - Better outcome by increasing the number of layers instead of the number of neurons per layer
- **Number of Neurons per Layer:**
  - Often set as a decreasing pyramid
- **Learning Rate:**
  - Use learning rate scheduling or adaptive optimizers (e.g., Adam)
- **Batch Size:**
  - Smaller batches → noisier updates, better generalization
- **Activation Functions:**
  - ReLU for hidden layers, Softmax for output in classification