ASTR8150/PHYS8150
Neural Networks

Fabien Baron

Georgia State University

fbaron@gsu.edu

Fall 2025

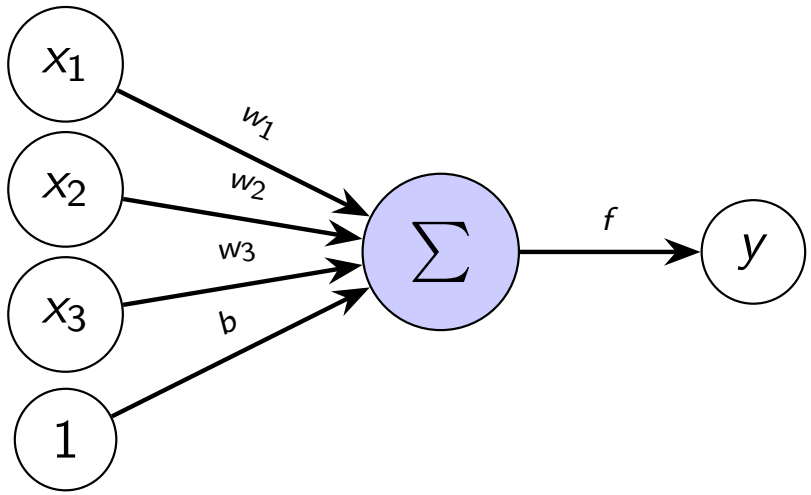
What is a Neural Network?

- A neural network is a computational model inspired by the way biological neural networks in the brain process information.
- Neural network "models" consist of layers of interconnected nodes (neurons) that can learn complex patterns from data
- A well-designed network can approximate any continuous function on compact (finite) domains. Better approximations will require larger networks.
- Models are optimized by adjusting weights based on errors between predicted and actual values, using gradient descent variants adapted to large number of parameters.
- These models are the foundation of deep learning models
- Typical tasks: classification, regression, pattern recognition, regularization, generation of patterns.

What is a neuron?

- A neuron, is a weighted sum of the inputs, followed by an activation function.
- A neuron first computes a linear transformation known as preactivation:

$$a = \sum_i w_i x_i + b$$

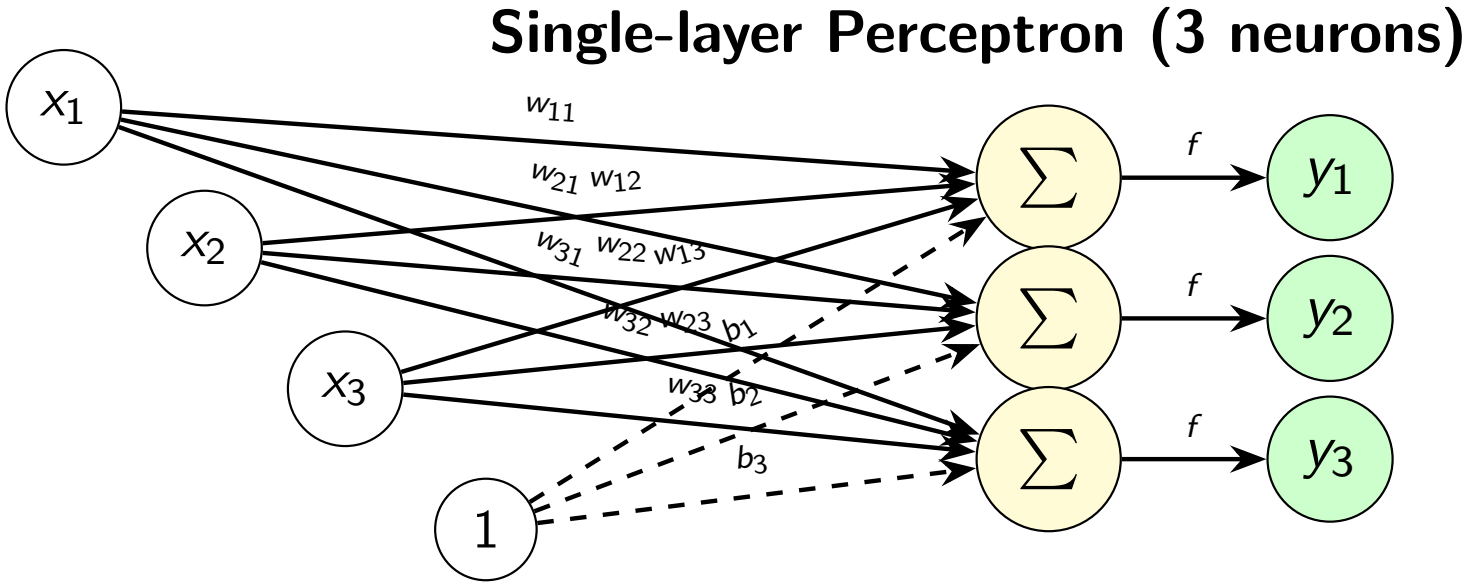


- Nonlinearity is introduced by using an activation function f:

$$y = f(a)$$

- The single-perceptron was the first neuron (developed for classification by Rosenblatt in 1958).

The Single-layer Perceptron



- Single layer of neurons that compute a weighted sum of inputs and apply an activation
- Suitable for binary classification tasks (output: 0 or 1).

Weights and Biases

- **Weights:** Parameters that determine the strength of the connection between neurons.
- **Biases:** the bias $b = w_0$ allows shifting of activation thresholds and this added flexibility helps the model make better predictions.
- The output of a neuron is calculated as:

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

where:

- w_i are the weights
- x_i are the inputs
- b is the bias
- f is the activation function

Activation Functions

- Activation functions introduce nonlinearity and enable networks to model complex mappings. Which ones to use depend on what the layer is designed to do (comes with experience).
- **Sigmoid:**

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

  Maps input values to a range between 0 and 1.
- **Tanh:**

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

  Maps input values to a range between -1 and 1.
- **ReLU (Rectified Linear Unit):**

$$f(x) = \max(0, x)$$

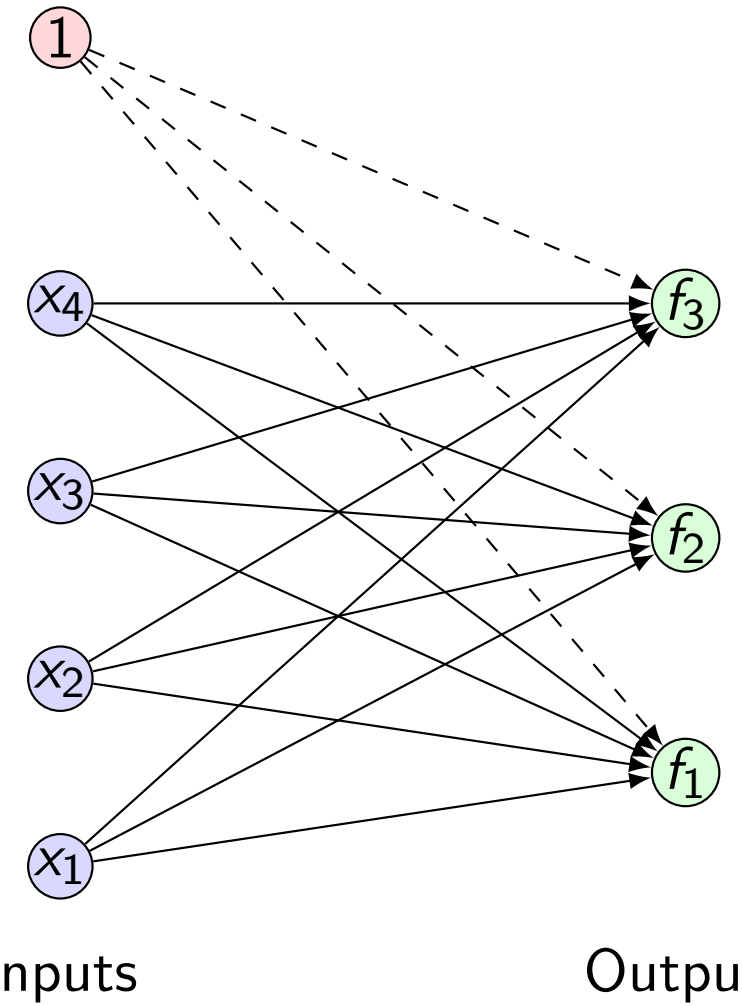
  Allows only positive values to pass through.
- **Softmax:**

$$f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}}$$

Dense Layers

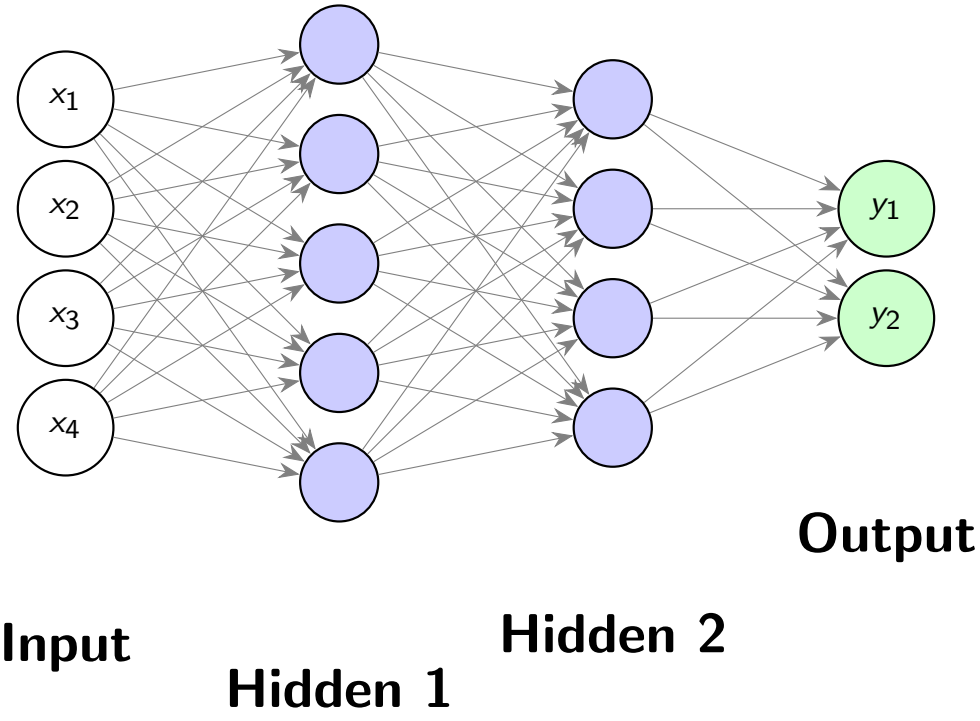
- A **dense layer** (fully connected layer) connects every neuron in the current layer to every neuron in the next layer.
- Each neuron in a layer is connected to neurons in adjacent layers through weighted connections.
- Each connection has an associated weight and each neuron has an associated bias.

Dense Layer



Neural network architecture: the Multilayer Perceptron

- A neural network consists of three types of layers
- **Input layer:** Receives the input data (contain features).
- **Hidden layers:** Intermediate layers where features are extracted but outputs are not directly observed.
- **Output layer:** Produces the final output (prediction).



- MLPs can model complex relationships and solve problems that are not linearly separable.
- MLPs are widely used in tasks like image recognition, speech processing, and time series prediction.

Training Neural Networks

- The goal of training a neural network is to minimize the error between the predicted and actual outputs.
- Training involves:
  - Feeding input data into the network.
  - Computing the output through forward propagation.
  - Comparing the output with the true label (using a loss function).
  - Adjusting weights to minimize the error using an optimization algorithm.
- Loss functions depend on the task
  - Regression: squared error.
  - Classification: cross-entropy.
- **Epochs:** The number of times the network is trained on the entire dataset.

Gradient Descent for neural networks

- Variants of gradient descent are used to minimize the loss function.
- The weight update rule for gradient descent is:

$$w_i = w_i - \eta \frac{\partial L}{\partial w_i}$$

where:

  - η is the learning rate.
  - $\frac{\partial L}{\partial w_i}$ is the gradient of the loss function with respect to weight w_i.
- There are different variants of gradient descent:
  - **Stochastic Gradient Descent (SGD):** Uses a single data point for each update.
  - **Batch Gradient Descent:** Uses the entire dataset for each update.
  - **Mini-batch Gradient Descent:** Uses a small batch of data for each update.

Backpropagation

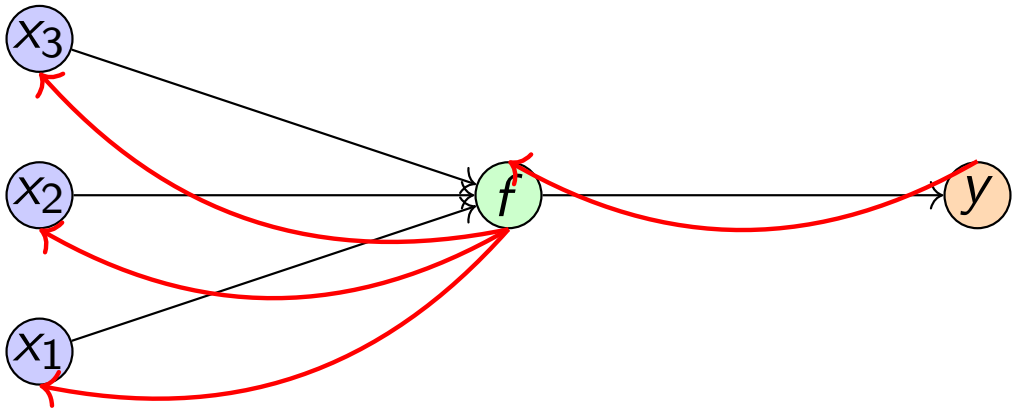
- How do we calculate the gradient of the network with respect to weights? Via backpropagation.
- This led to the development of automatic differentiation
- The algorithm involves two main steps:
  - **Forward Propagation:** Input data is passed through the network to get the output.
  - **Backward Propagation:** The error is calculated, and the gradients of the loss function with respect to the weights are computed.

Backpropagation: Intuition

- Goal: Compute $\frac{\partial L}{\partial w_{ij}}$ for all weights w_{ij} in the network.
- Uses the **chain rule** to propagate gradients from the output layer back to hidden layers.
- For a single hidden neuron f_j:

$$\frac{\partial L}{\partial w_{ji}} = \frac{\partial L}{\partial f_j} \frac{\partial f_j}{\partial a_j} \frac{\partial a_j}{\partial w_{ji}}$$

where $a_j = \sum_i w_{ji} x_i + b_j$ is the pre-activation.



Backpropagation: Algorithm

1. **Forward pass:** Compute all activations a_j and outputs f_j, y_k.
2. **Compute output layer gradient:**

$$\delta_k = \frac{\partial L}{\partial y_k} \odot g'(a_k)$$

where g is the **output layer activation function** (e.g., softmax for classification).

3. **Propagate to hidden layers:**

$$\delta_j = f'(a_j) \sum_k w_{kj} \delta_k$$

4. **Compute weight gradients:**

$$\frac{\partial L}{\partial w_{ji}} = \delta_j x_i$$

5. **Update weights:**

$$w_{ji} \leftarrow w_{ji} - \eta \frac{\partial L}{\partial w_{ji}}$$