INTRO TO NEURAL NETWORKS

Elizabeth Heon

5/19/2025

INTRO

What, where, why?

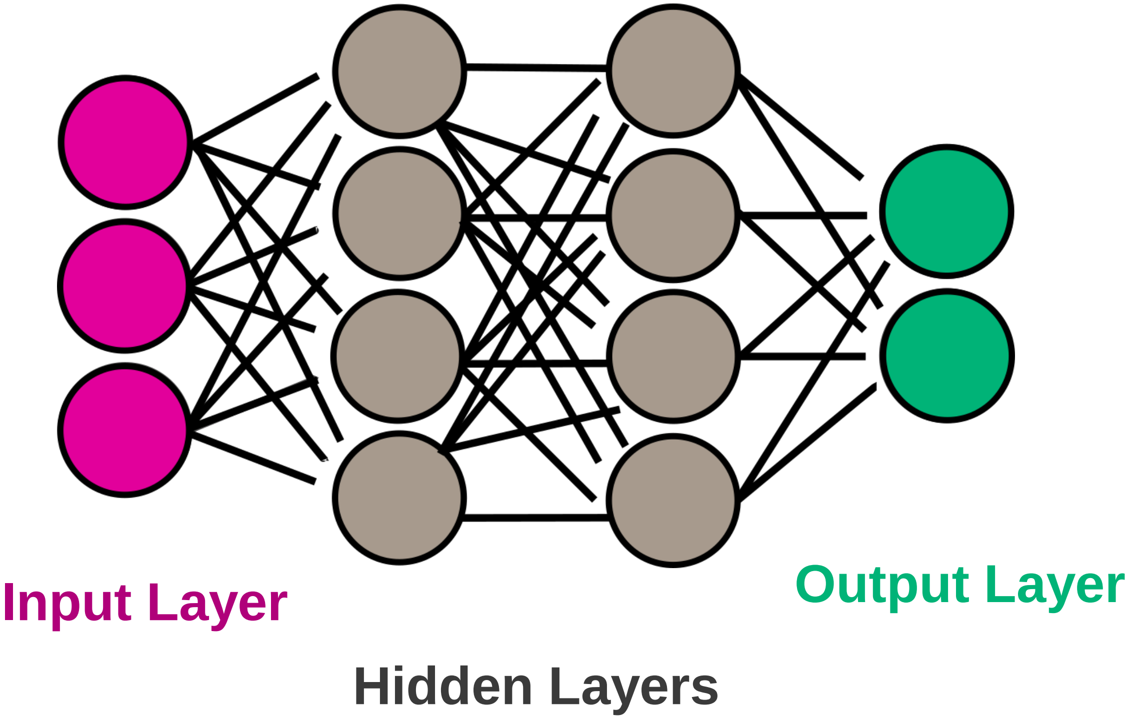
What is a Neural Network?

A neural network is a machine learning model formed of interconnected layers of nodes, called neurons, which are designed to mimic the function of biological neurons

- Neurons are connected together in layers, forming networks. The output of one neuron becomes input to another.

- The connections in networks are associated with **weights**, which determine how much the output of one neuron influences

- Networks are **trained** by adjusting the weights to improve performance



Where are Neural Networks Used?

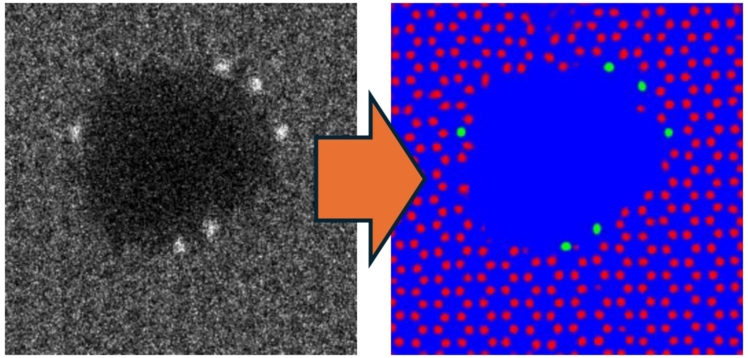
- Image recognition
- Speech recognition
- Recommendation Algorithm

NETFLIX

AlphaFold

Accelerating breakthroughs in biology with AI

- **Many applications in the sciences:**
  - Predict protein structure from amino acids
  - Predict chemical compounds with useful properties
  - Find atoms in EM datasets
  - Etc.



M. Ziatdinov, ACS Nano 2017

Neural networks are used in many applications in the physical sciences, technology, etc.

Types of Learning:

- Supervised Learning: Labelled training data is available – network learns to predict the correct answer for new data
  - Regression
  - Classification
  - Semantic and Instance Segmentation

- Unsupervised Learning: No labelled training data provided – networks learns intrinsic patterns in the data
  - Clustering

- Semi-supervised Learning: Some labelled training data is provided

This talk will focus on supervised learning, where a network is supplied labelled training data

Regression and Classification

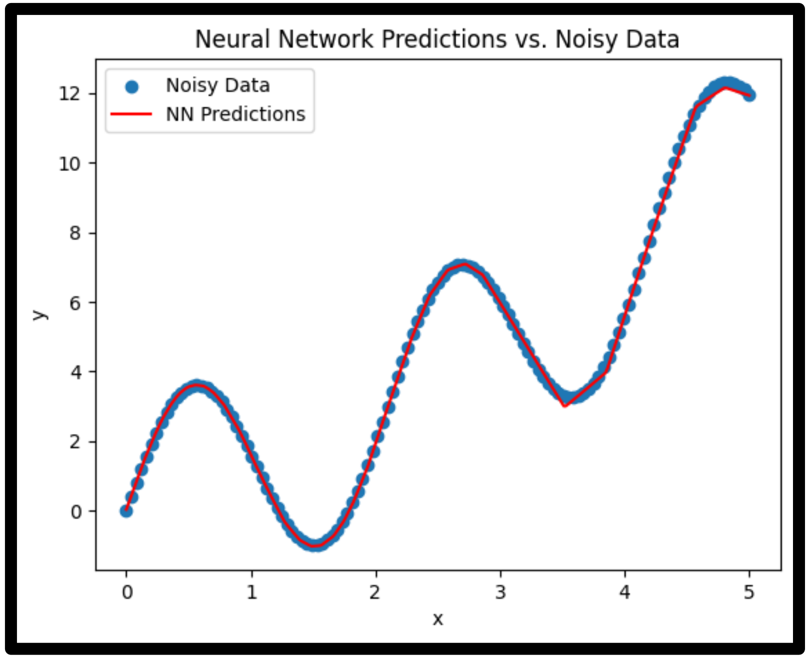
Classification	Regression

Classification

- Classification tasks -> Output of network is categorical
- Examples:
  - Medical Diagnosis: Input -> medical imaging, Output -> Diagnosis
  - Character Recognition: Input -> Image of handwritten character, Output -> Letter A-Z or digit 0-9

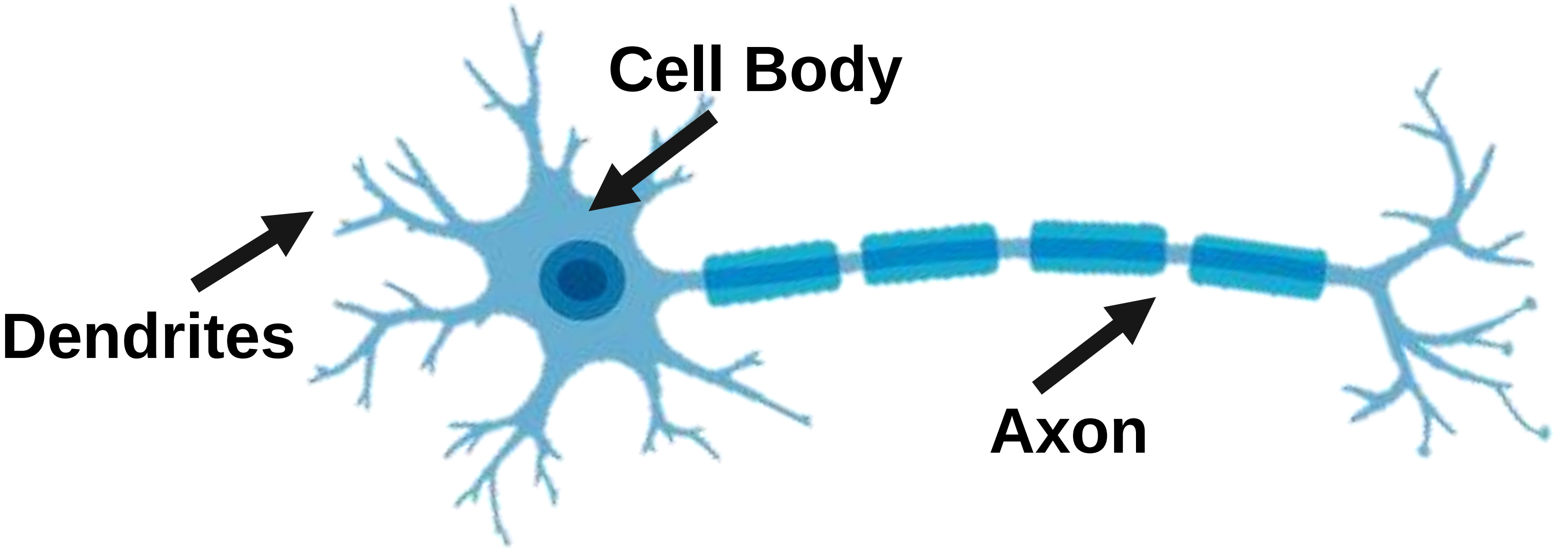
Regression

- Regression tasks -> Output of network is numerical
- Examples:
  - Linear regression / function interpolation



I. NEURONS

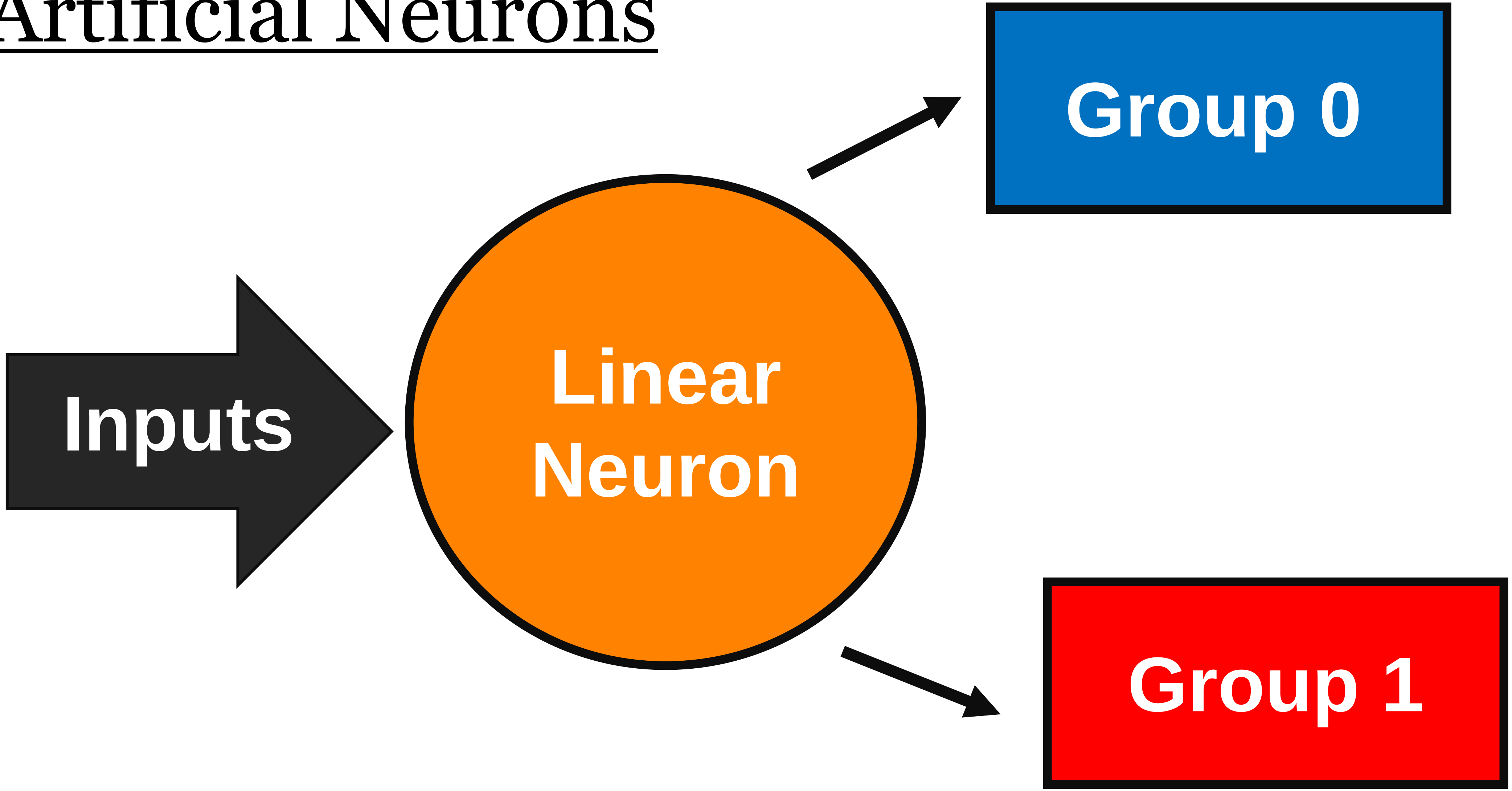
Biological Neurons:



- Dendrites receive messages from other neurons – these messages can be excitatory (encourage the neuron to fire) or inhibitory (prevent firing)
- Depending on the combination of inputs, the neuron may fire an "action potential" which travels down the axon to other neurons

A (biological) neuron takes a combination of inputs and decides which output to produce – potential or no potential

Artificial Neurons



Consider a binary linear neuron – given a set of inputs, the neuron can output either 0 or 1 -> essentially, the neuron can classify a set of inputs into one of two groups

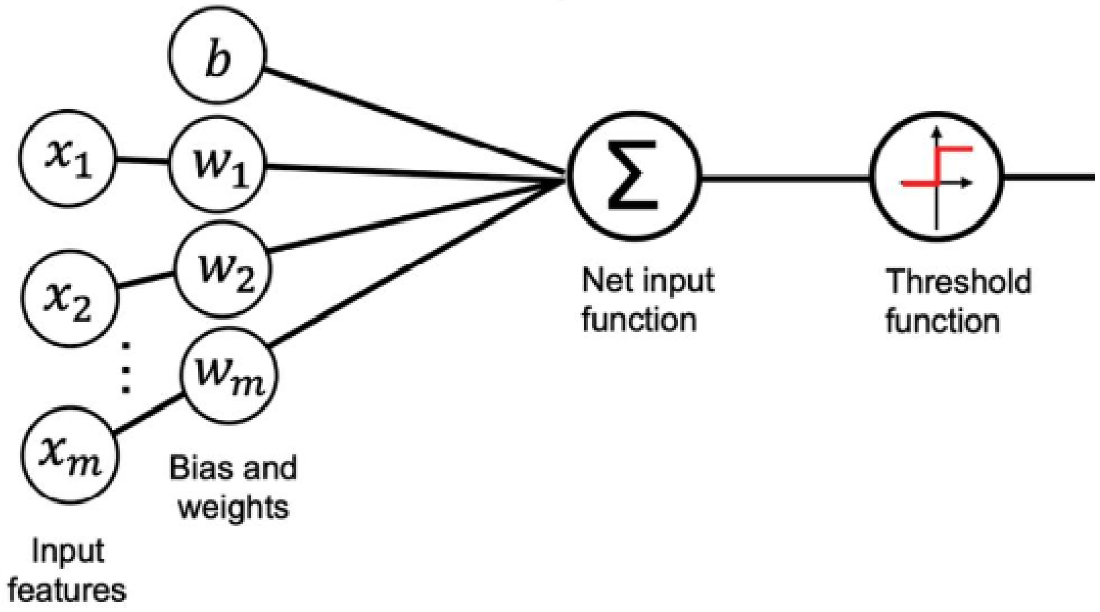
Artificial Neurons

- Neuron receives a vector of **inputs**, x_i
- Inputs are multiplied by vector of **weights** to form net input function:

$$z = \sum_i x_i w_i$$

- The net input is compared to a value called the **threshold** (θ):

- **Output:** $\sigma(z) = \begin{cases} 1 \ if \ z > \ \theta \\ 0 \ otherwise \end{cases}$



Artificial neurons take in a set of inputs and determine what output to produce using a set of weights and a threshold

Bias

- The critical point between inputs that result in output 0 and inputs that result in output 1 is given by:

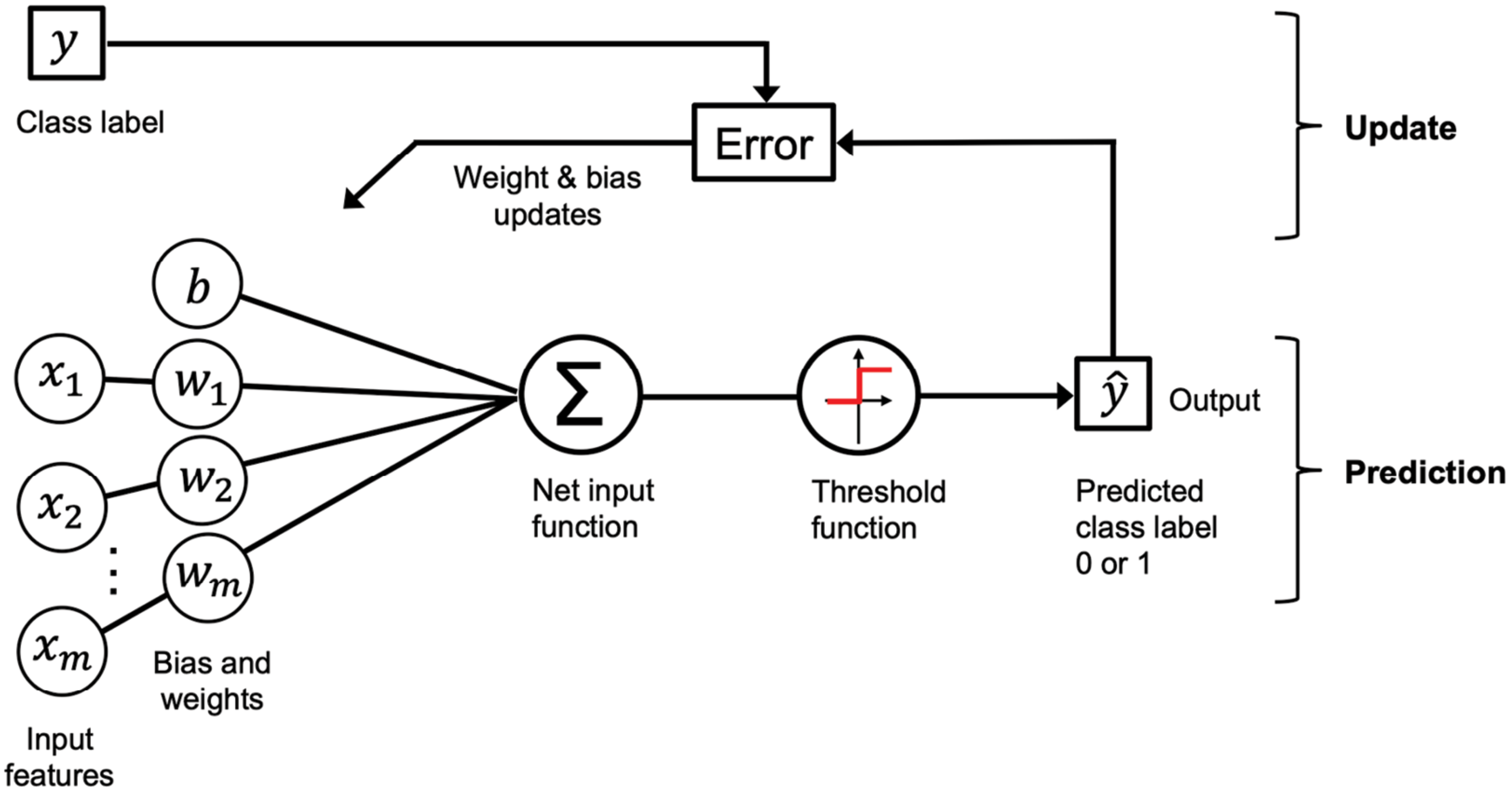
$$z = \sum_i x_i w_i = \theta$$

- Subtract the value of θ from both sides:

$$x_i w_i + \ \ldots + x_n w_n \ - \ \theta = 0$$
$$x_i w_i + \ \ldots + x_n w_n + b = 0$$

- The term b is known as the **bias** – using the bias makes the threshold value trainable along with the weights

Training a Linear Neuron - Overview



Training is the process of updating the weights and bias to improve the performance of the model. Training is performed iteratively

Training Algorithm - Verbal

- To train the network we require a set of training data. Training data consists of:
  - Vectors of inputs -> x_i
  - The **"ground truth"** (i.e. correct) output for each set of inputs, $\mathbf{y^{(i)}}$

- For each vector in the training, g set, calculate the **model prediction**, $\widehat{y^{(i)}}$, then:
  - If the prediction is correct, do nothing
  - If the prediction is wrong, update the weights and bias to move the decision boundary in a way that reduces error
  - Repeat until the bias and weights stop changing

Training Algorithm - Math

1) Initialize the bias and weights either to zero or to small random numbers

2) For each example in the training set:

- **Compute the output using the current bias and weights:** $\hat{y}^{(i)} = w^T x^{(i)} + b$
- **Update the weights and the bias:**
  - $w_j = w_j + \Delta w$ where $\Delta w = \eta(y^{(i)} - \hat{y}^{(i)})x_j^{(i)}$
  - $b = b + \Delta b$ where $\Delta b = \eta(y^{(i)} - \hat{y}^{(i)})$

3) Repeat until prediction matches output for all examples in training set

The Problem of Separability

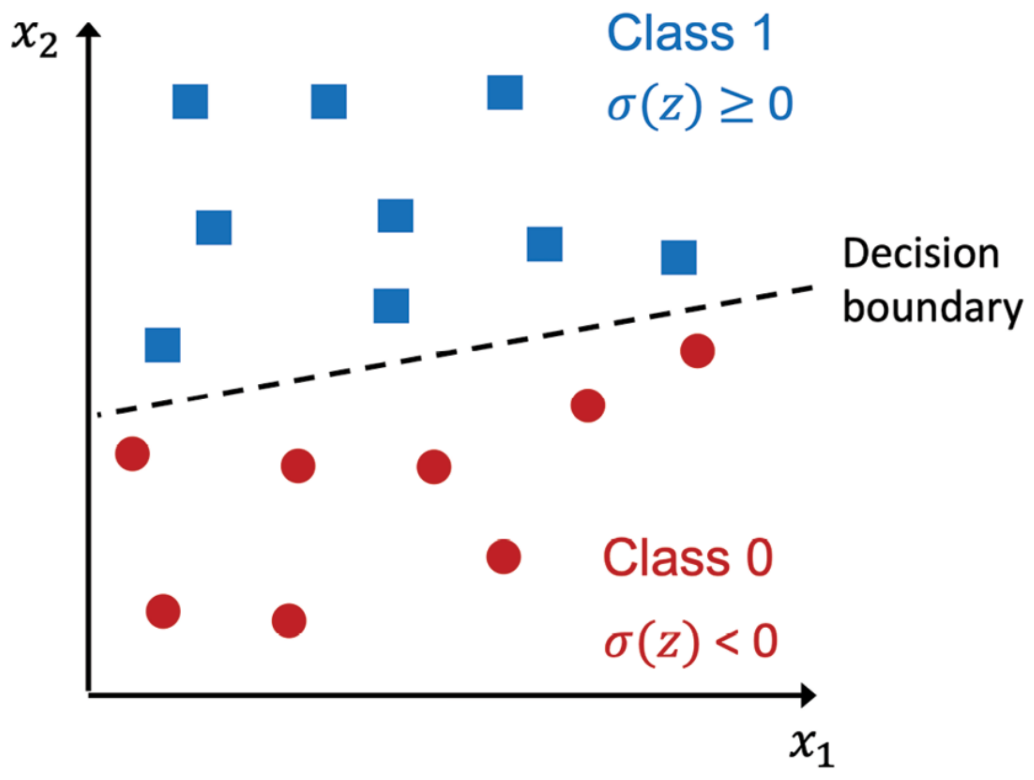
- Consider a linear neuron that takes two inputs, x1 and x2:
- We thus have: $x_1 w_1 + x_2 w_2 = \theta$
- Rearrange:

$$x_2 w_2 = \theta - x_1 w_1$$

$$x_2 = \frac{\theta}{w_2} - \frac{w_1}{w_2} x_1$$

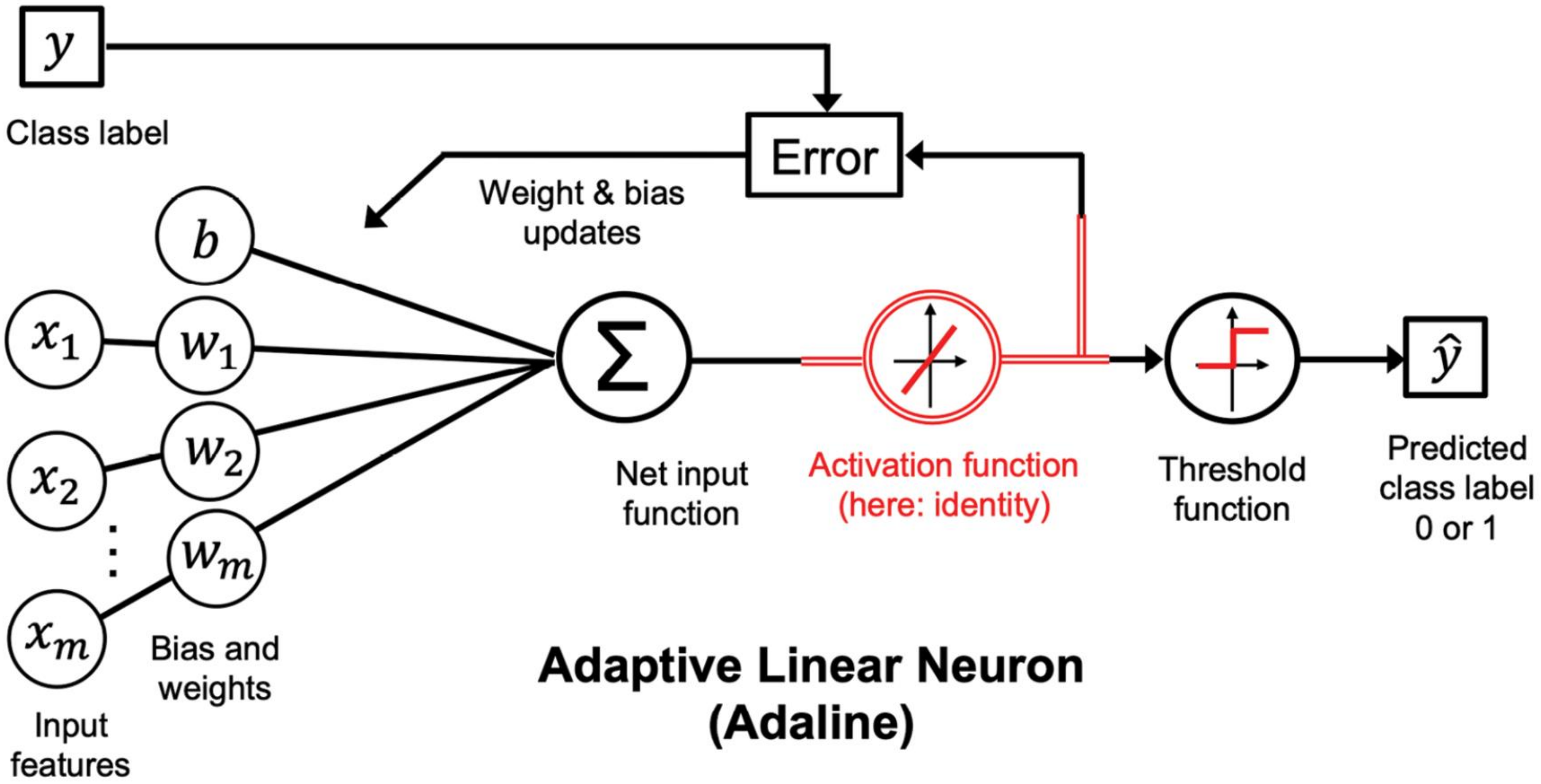
$$\boldsymbol{x_2 = a x_1 + b}$$

- Results in linear equation – **decision line/plane**



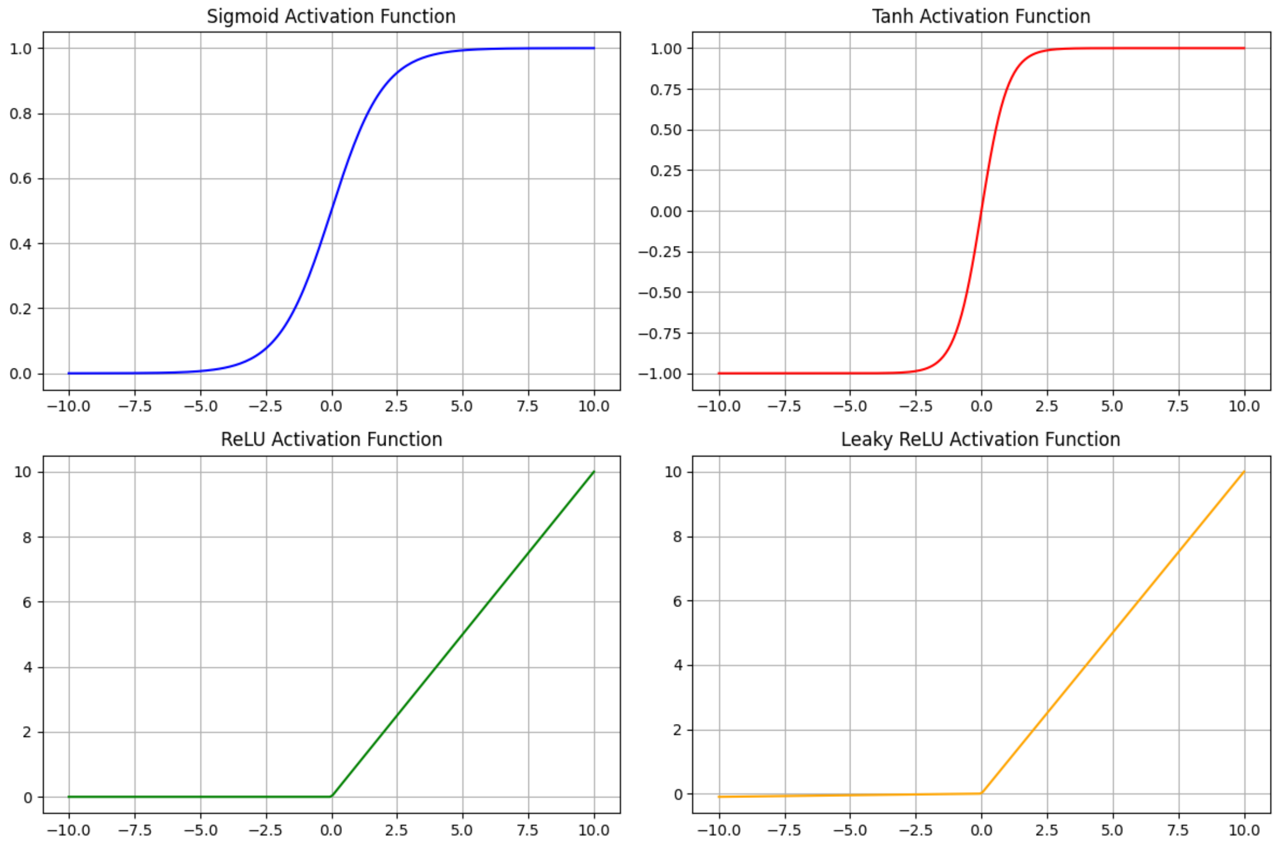
Single neurons can only solve problems that are linearly separable

Adaptive Linear Neuron (Adaline)



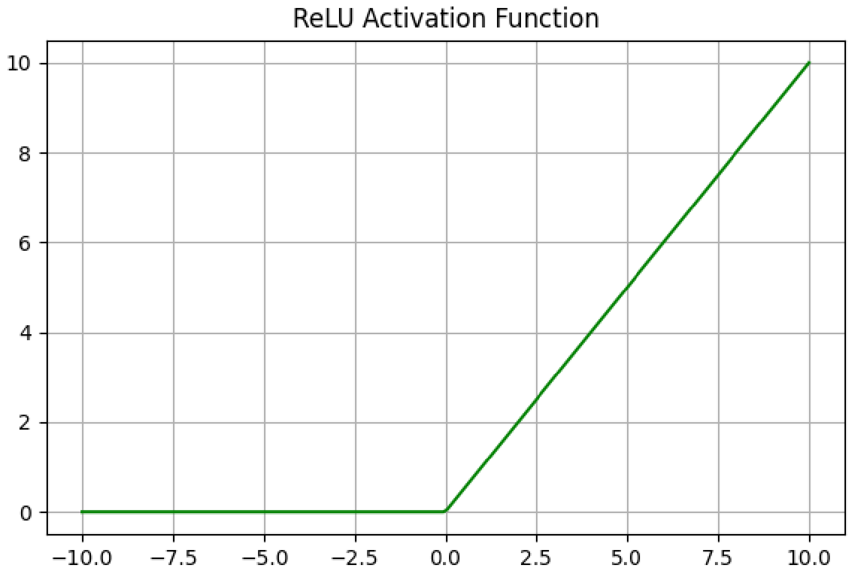
Activation Functions:

- Activation functions introduce **non-linearity** to neural networks – otherwise even deep networks can only solve linear problems
- Non-linearity ->output has a non-linear relationship to input – decision surface can be curved

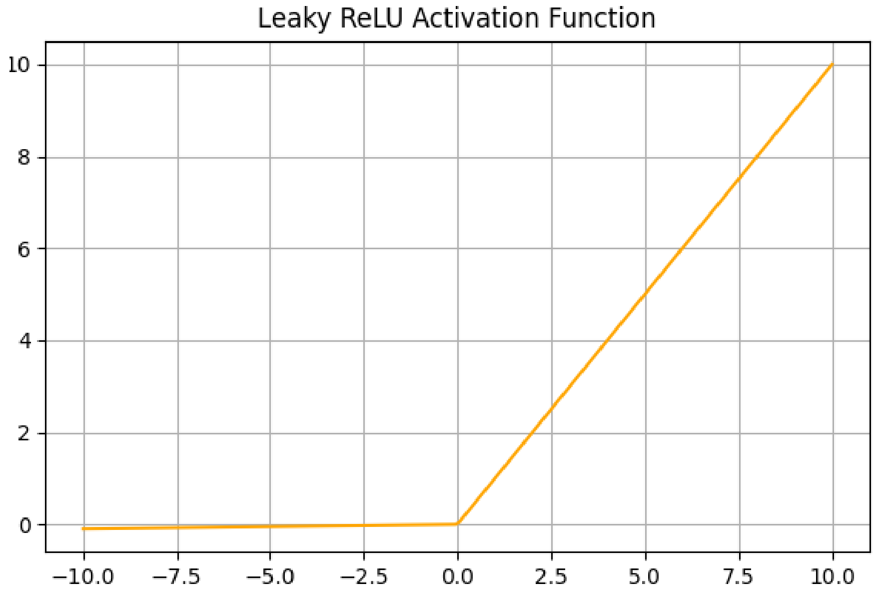


Activation functions introduce non-linearity to neural networks, allowing networks to learn complex patterns in data

Choosing Activation Functions

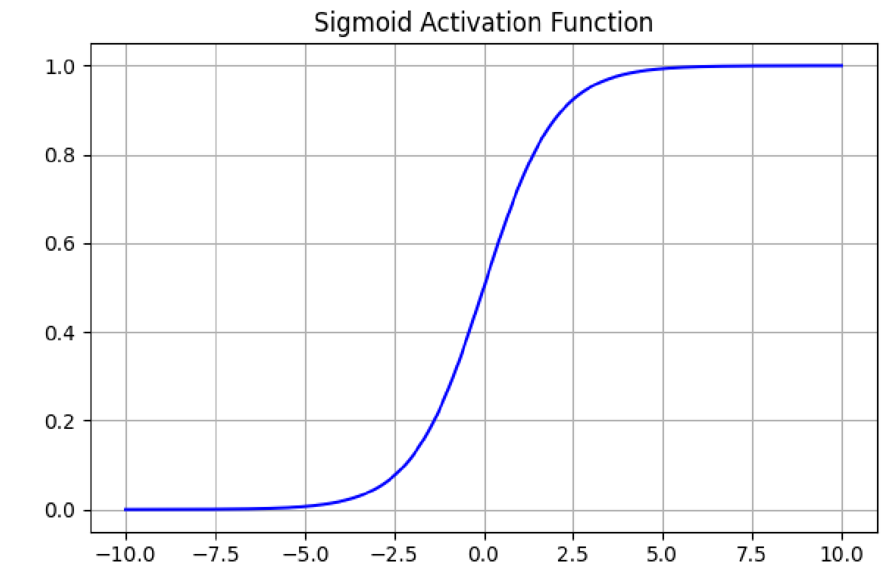


- **Re**ctified **L**inear **U**nit: If input is negative, output is zero. If input is positive, output = input
- Often used for image data or in other cases where negative inputs are not physically meaningful

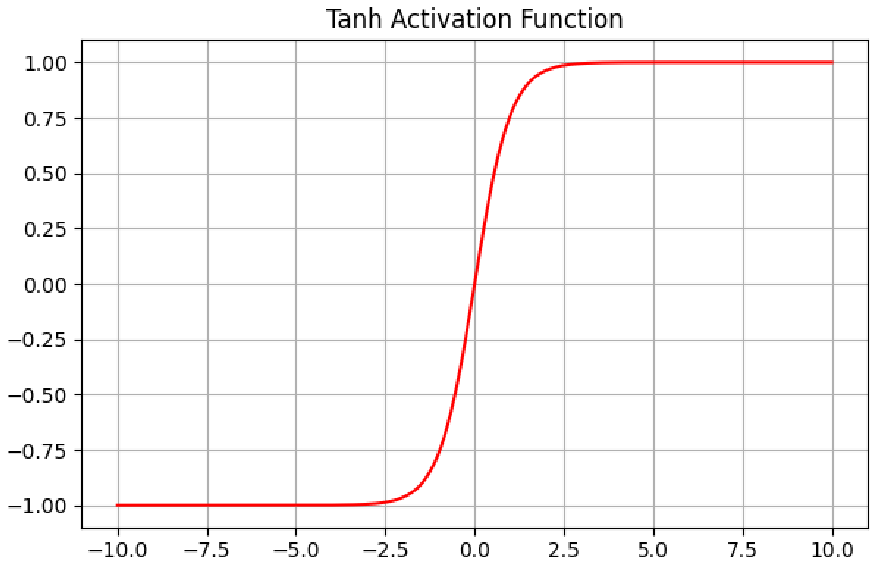


- **Leaky ReLU:** Introduce a small non-zero slope for negative inputs
- Helps avoid “dying ReLU” where negative inputs cause a node is a network to become inactive

Choosing Activation Functions



- **Sigmoid:** Maps output to values between 0 and 1
- Ideal for situations where output should be interpreted as a probability



- **Tanh:** Maps output to values between -1 and 1
- Often used in hidden layers

Training an Adaline – Loss Function

- We train an Adaline by minimizing a **loss function**:

Model Prediction

$$L(w,b) = \frac{1}{n}\sum_{i=1}^{n}\left(y^{(i)} - \sigma\left(z^{(i)}\right)\right)^2$$

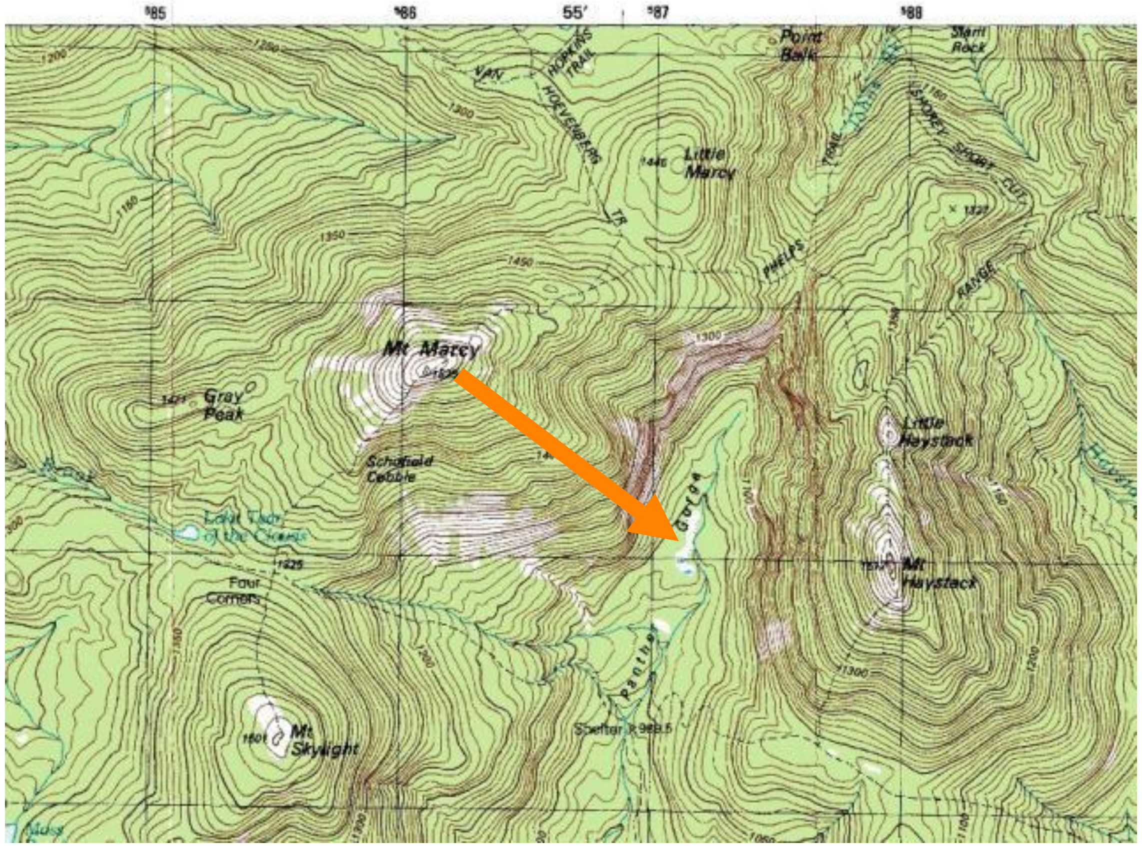
Ground Truth

- A loss function quantifies the difference between the network prediction and the ground truth
- The above is called the Mean Squared Error (MSE) loss and is common for regression problems

Loss functions quantify error for Adaline; training occurs by minimizing the loss

Gradient Descent - Analogy

- For every combination of weights and bias there is a corresponding value of the loss. This forms a loss surface/hypersurface. To train the network we must move to the minimum of the surface



- You are on a hill, and want to reach the valley (minimum) nearby
- To reach the minimum, make a series of steps, always making sure to **move down hill**

Gradient Descent - Math

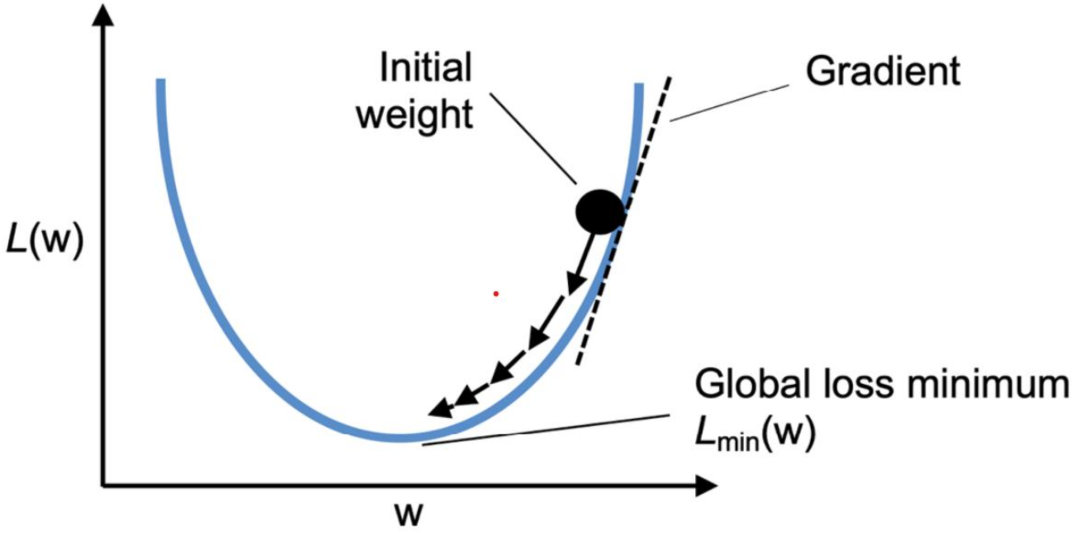
- To move "downhill" we need the slope of the loss function
  - To find the slope of a function, take the derivative (in this case **partial derivatives** since loss is a multivariate function)

1) Start at an arbitrary position on the loss surface.

2) Use the learning rule:

$$\Delta w = -\eta \nabla_w L(w,b)$$
$$\Delta b = -\eta \nabla_b L(w,b)$$

$$w := w + \Delta w$$
$$b := w + \Delta b$$

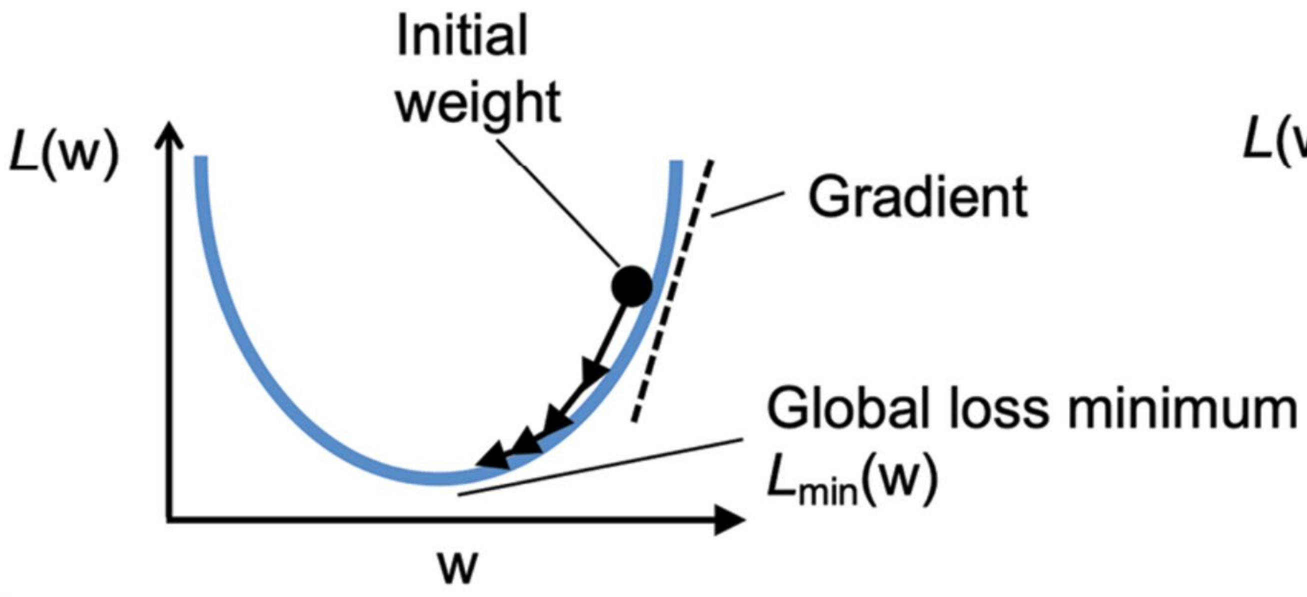


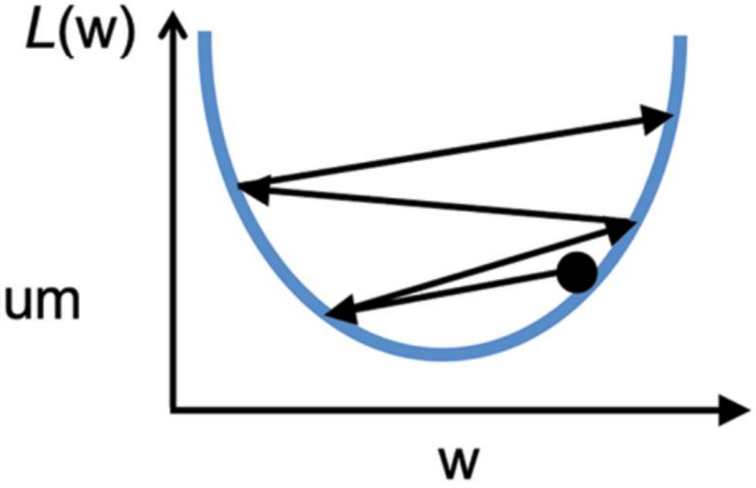
η = Learning Rate

Determines how large each "step" is

Role of Learning Rate

- Learning rate, **η**, determines how quickly weights and bias change – how large each "step" is. Too small and the network will take a very long time to reach the minimum. Too large and it may overshoot the minimum and fail to converge.

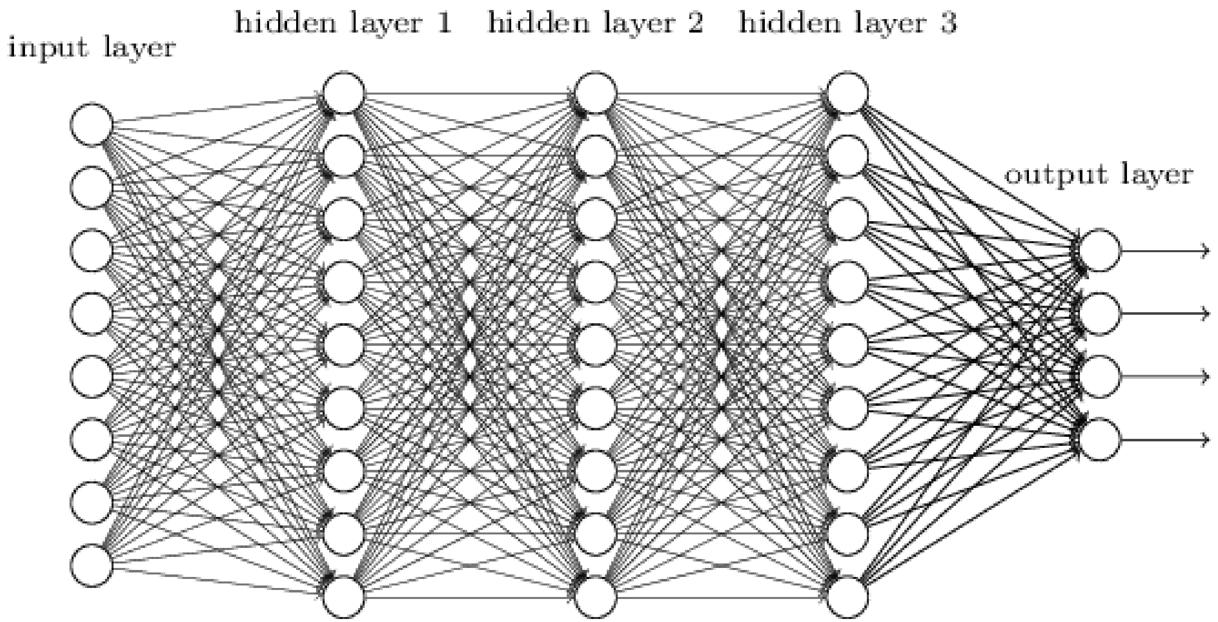




II. FROM NEURONS TO NETWORK

Putting Neurons Together

- Deep architectures allow networks to learn complex patterns in data (so long as they have non-linear activation functions) – how does a multi-layer network work?



- Composed of multiple layers of artificial neurons – the inner layers are known as **"hidden layers"**
  - The number of nodes per layer and the connectivity between nodes varies between networks
- Each node receives inputs, applies weights and biases, and passes outputs onto the next layer

Training a Multilayer Network

- To train a multi-layer network, each neurons weights and bias must be updated. To train multi-layer networks an approach known as backpropagation is used.

- Backpropagation consists of a forward and backward pass:
  - **Forward pass:** put training data into network at input layer, and propagate forward through the network, producing an output.
  - Calculate the loss based on the network prediction and the ground truth
  - **Backward pass:** Compute the gradient of the loss function with respect to each neuron's weights by chain rule
  - Update weights

Multi-layer networks are trained by backpropagation, which moves layer by layer to determine appropriate updates to the network weights

Metrics

Once a network is trained, the performance can be evaluated according to several possible metrics, including:

- **Accuracy:** Defined as the ratio of correctly predicted observations to the total observations.
- **Precision:** The ratio of correctly predicted positive observations to the total predicted positive observations. Important in scenarios where the cost of a false positive is high.
- **Recall** (Sensitivity): The ratio of correctly predicted positive observations to all observations in the actual class. Crucial in situations where missing a positive instance is costly.

III. USING NETWORKS

There are Many Types of Neural Networks

- Multilayer perceptron
- Convolutional Neural Networks
- Autoencoders
- Physics informed neural networks
- Etc.

Given the many types of networks that are possible, how do you know which type to use?

Where to Start:

Problem-Centric Approach:

- Always start with the problem at hand: Analyze the nature of input data and desired output
- Choose a network architecture that aligns with the type and structure of your data
- Select a loss function that reflects the objective of the problem
- Metrics should be chosen based on what measures success for your specific task

Hyperparameter Tuning:

- Once the architecture and loss function are set, proceed to tune hyperparameters including network structure, optimizers, regularization, etc.
- Hyperparameter tuning should be guided by the chosen metrics and the nature of the problem

Coding Task

Thank you