THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 12: Neural Networks

Sergei V. Kalinin

Types of Machine Learning

Supervised (inductive) learning
– Given: training data + desired outputs (labels)

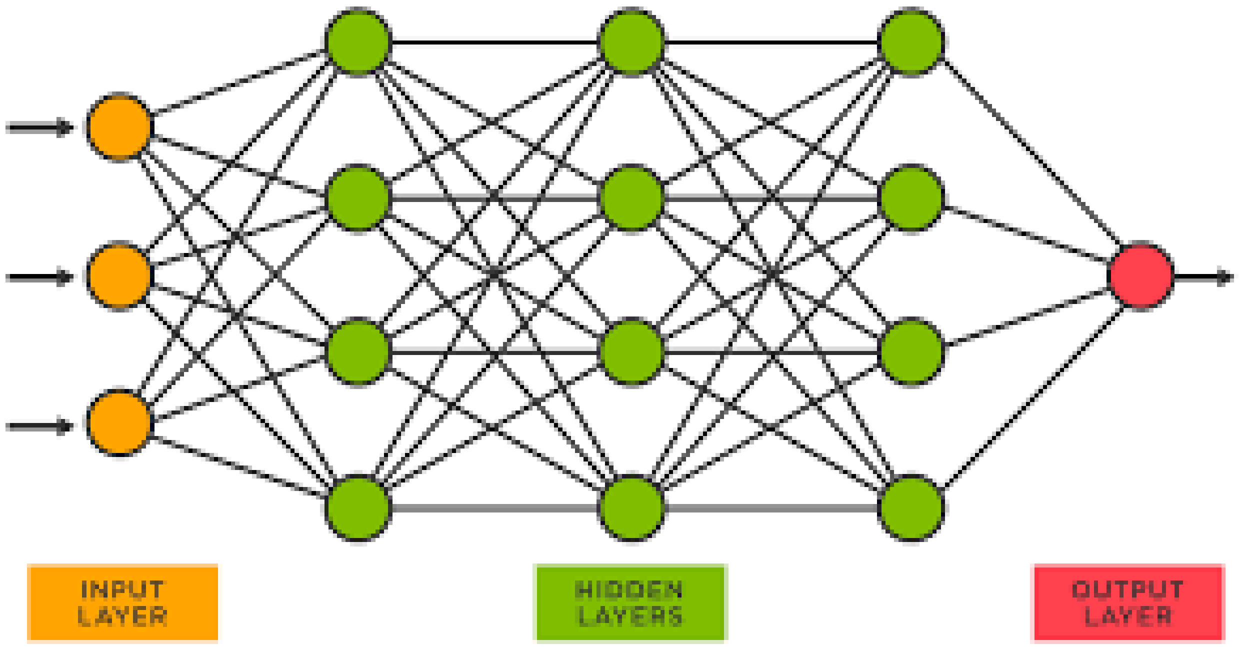
- **Unsupervised learning**
– Given: training data (without desired outputs)

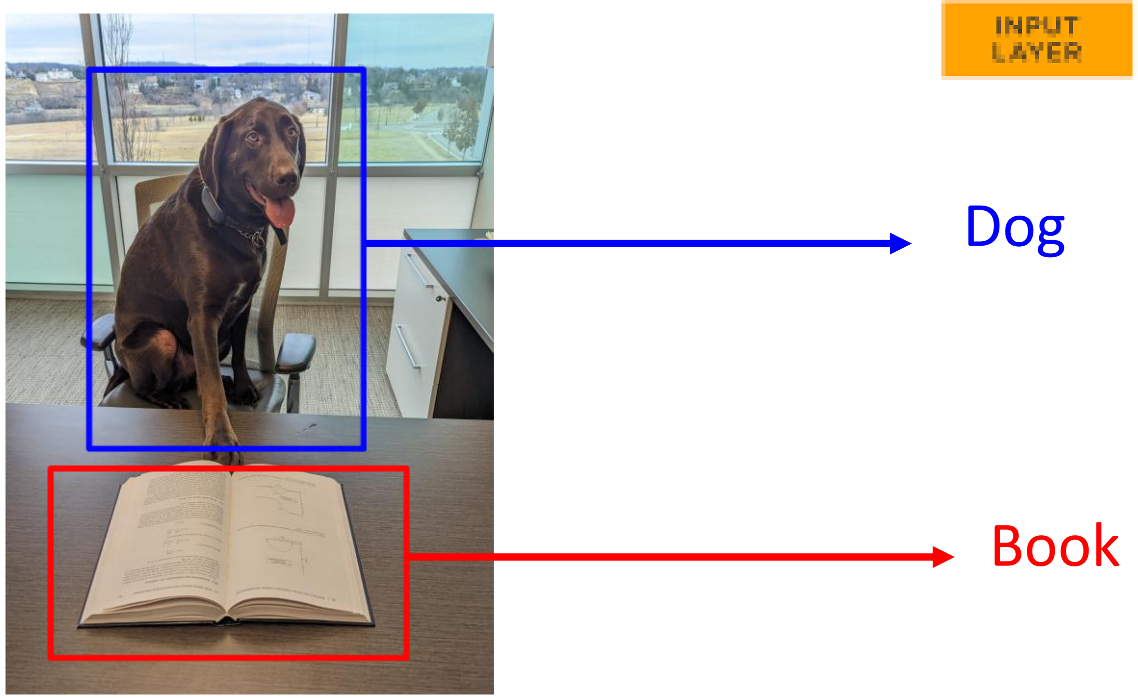
- **Semi-supervised learning**
– Given: training data + a few desired outputs

- **Reinforcement learning**
– Rewards from sequence of actions

Supervised Machine Learning

- Regression
- Classification
- Semantic segmentation
- Instance segmentation
- …





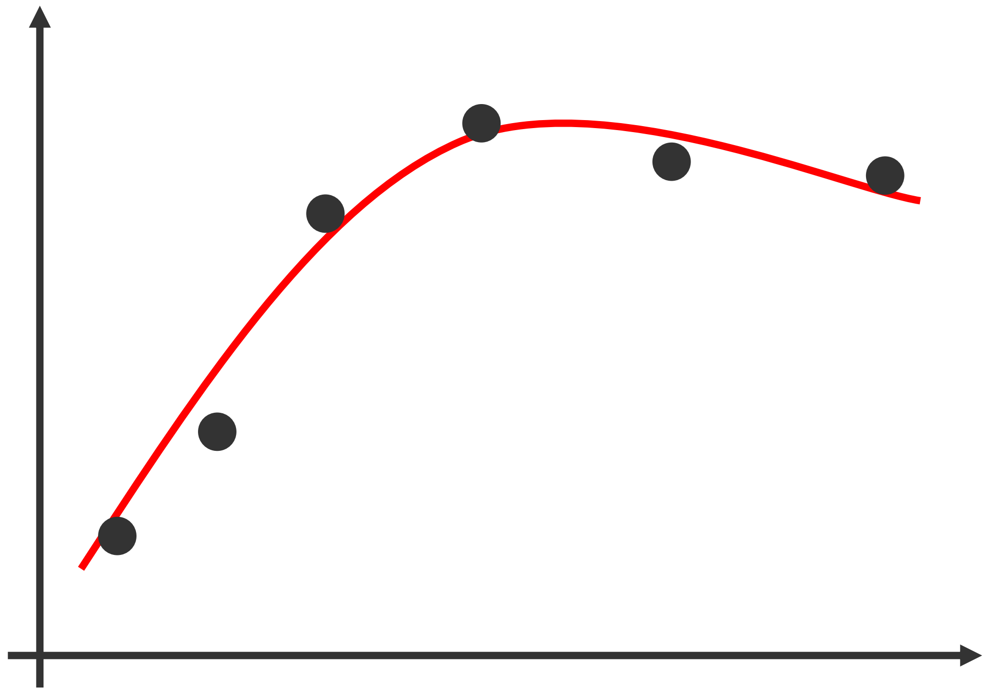
Classification

- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Learn a function f(x) to predict y given x
- If y is categorical == classification

Application	Input Data	Classification
Medical Diagnosis	Noninvasive tests	Results from invasive measurements
Optical Character Recognition	Scanned bitmaps	Letter A-Z and digits 0-9
Protein Folding	Amino acid sequence	Protein shape (helices, loops, sheets)
Materials Discovery	Composition	Metal/Semiconducotr
Research Paper Acceptance	Words in paper title	Paper accepted or rejected

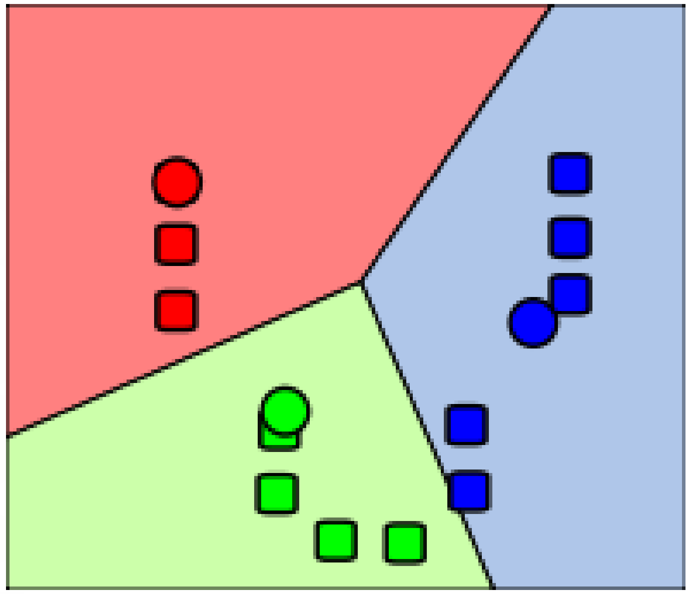
Regression

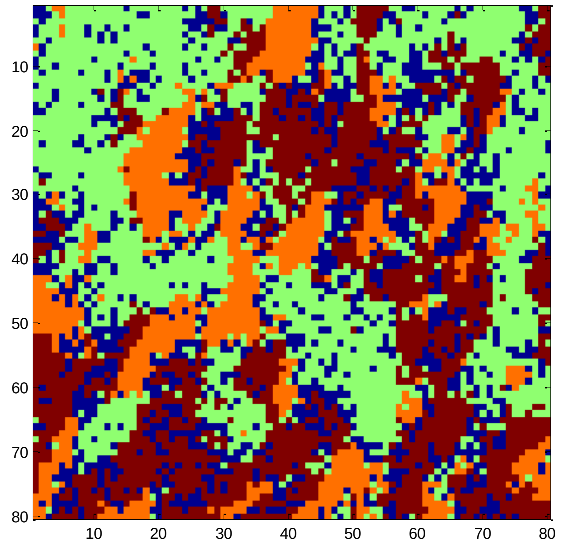
- Given $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- Learn a function $f(x)$ to predict y given x
- y is real-valued == regression

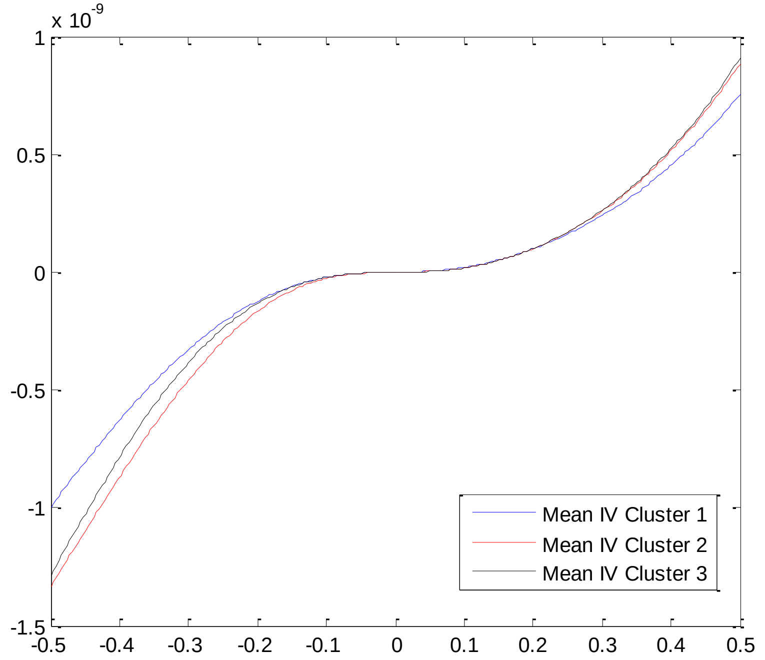


Unsupervised Learning

- Given $x_1, x_2, \ldots, x_n$ (without labels)
- Output hidden structure behind the x's
- E.g., clustering



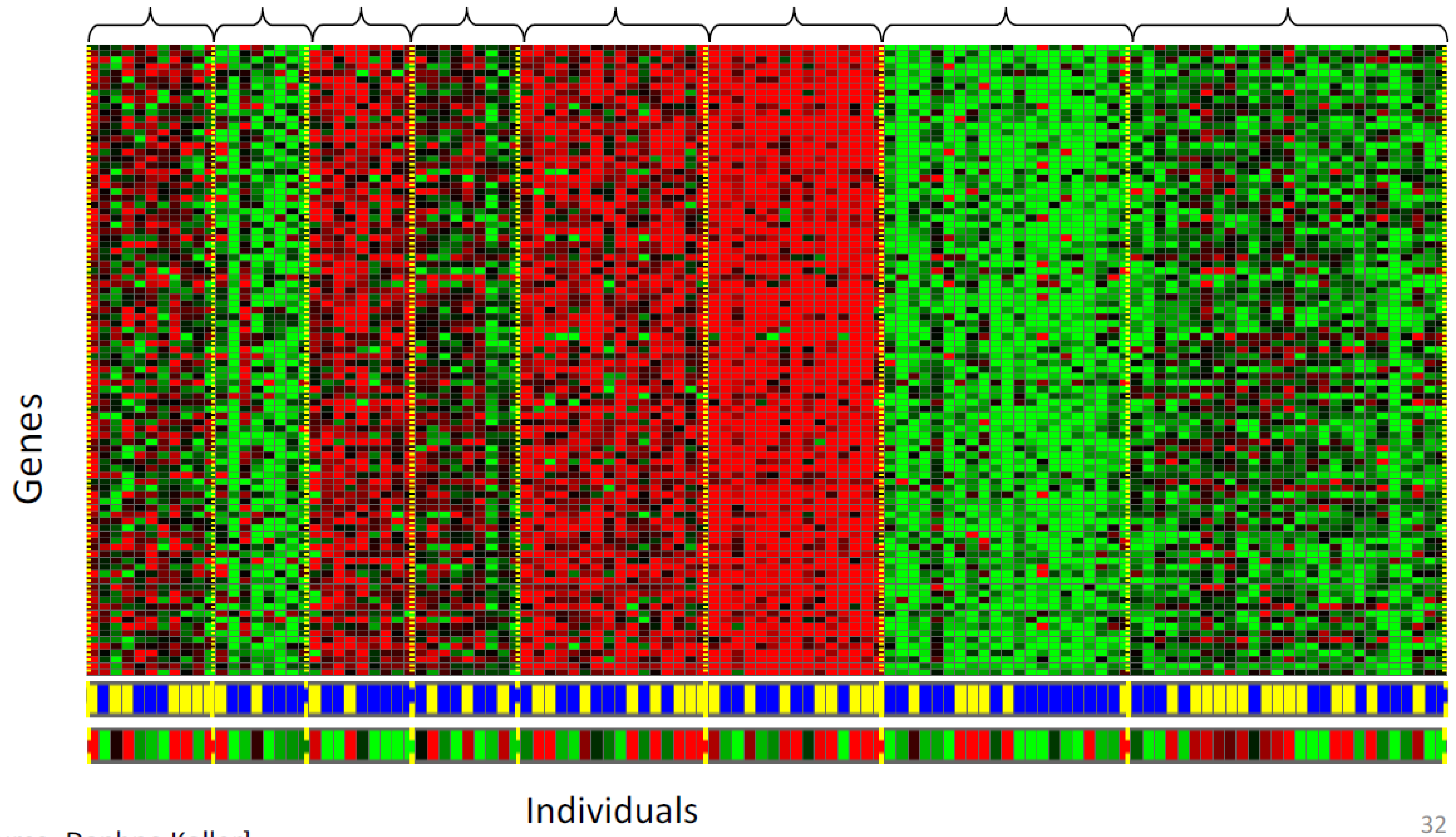




M. Ziatdinov, A. Maksov, L. Li, A. Sefat, P. Maksymovych, and S.V. Kalinin, *Deep data mining in a real space: Separation of intertwined electronic responses in a lightly-doped* $BaFe_2As_2$, Nanotechnology **27**, 475706 (2016).

Unsupervised Learning

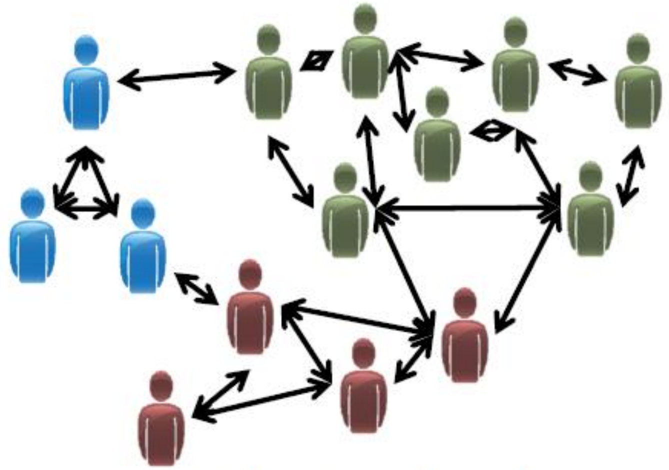
Genomics application: group individuals by genetic similarity



[Source: Daphne Koller]

Unsupervised Learning

Organize computing clusters



Social network analysis

Market segmentation

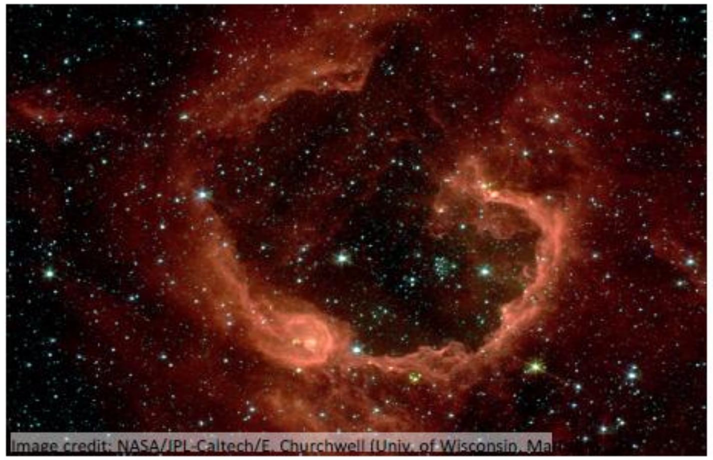


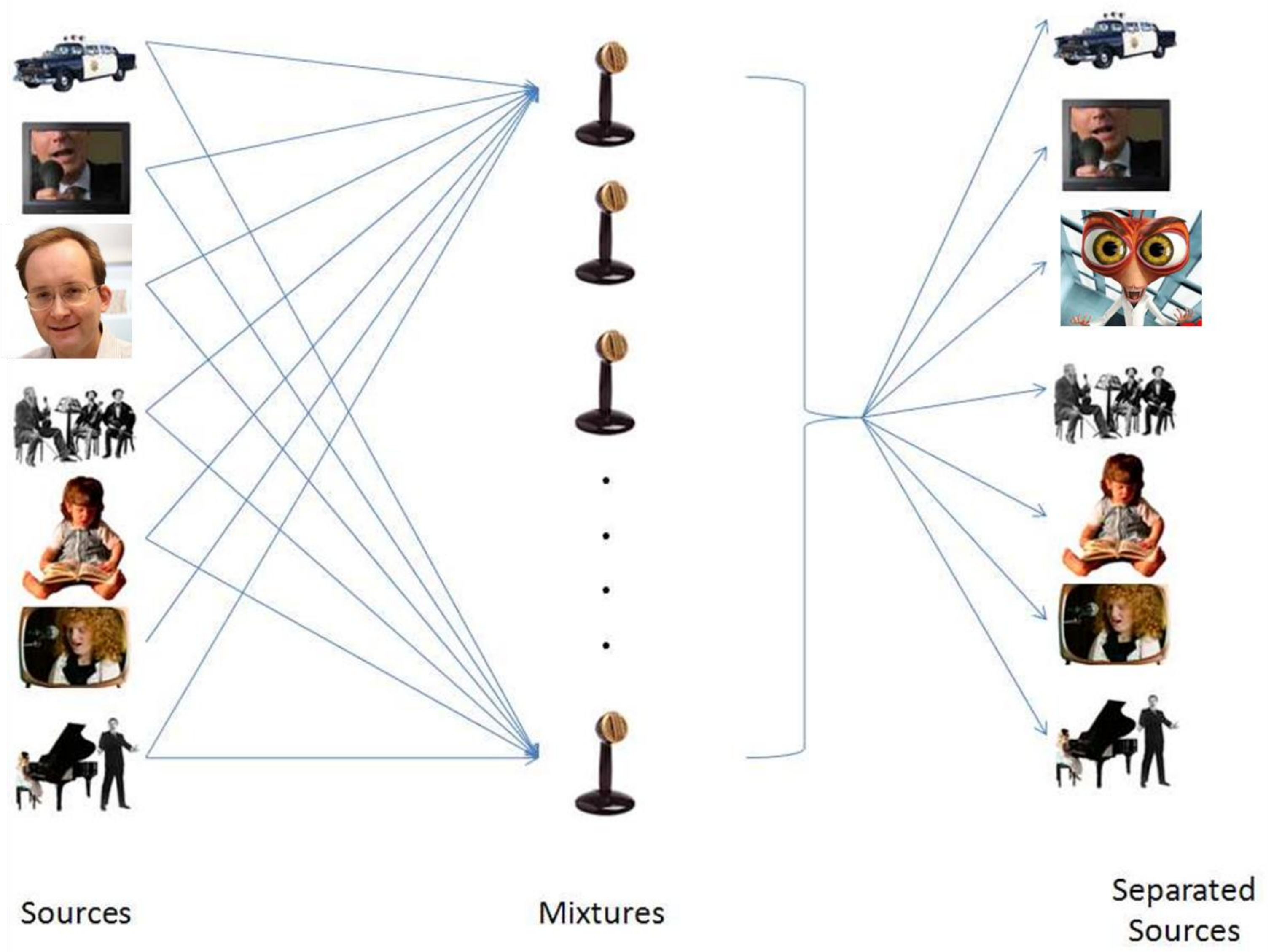
Image credit: NASA/JPL-Caltech/E. Churchwell (Univ. of Wisconsin, Ma

Astronomical data analysis

Slide credit: Andrew Ng

Unsupervised Learning

Number of signals are being produced simultaneously; with the objective of separating and following each source separately



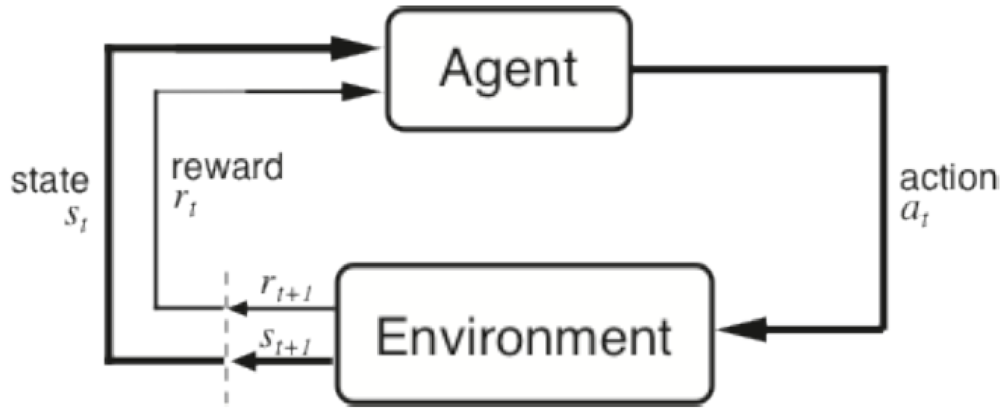
Reinforcement Learning

Given a sequence of states and actions with (delayed) rewards, output a policy
– Policy is a mapping from states to actions that tells you what to do in a given state

- Examples:

– Credit assignment problem
– Game playing
– Robot in a maze
– Balance a pole on your hand

RL: Agent and Environment



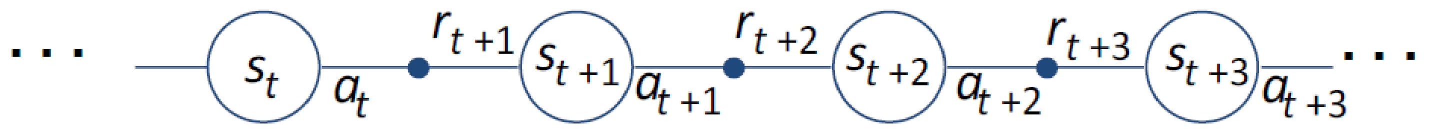
Agent and environment interact at discrete time steps : $t = 0, 1, 2, K$

Agent observes state at step t: $s_t \in S$

produces action at step t: $a_t \in A(s_t)$

gets resulting reward : $r_{t+1} \in \Re$

and resulting next state : s_{t+1}



Slide credit: Sutton & Barto

Reinforcement Learning in Action

https://www.youtube.com/watch?v=GtYlVxv0py8

Reinforcement Learning Applicatioons

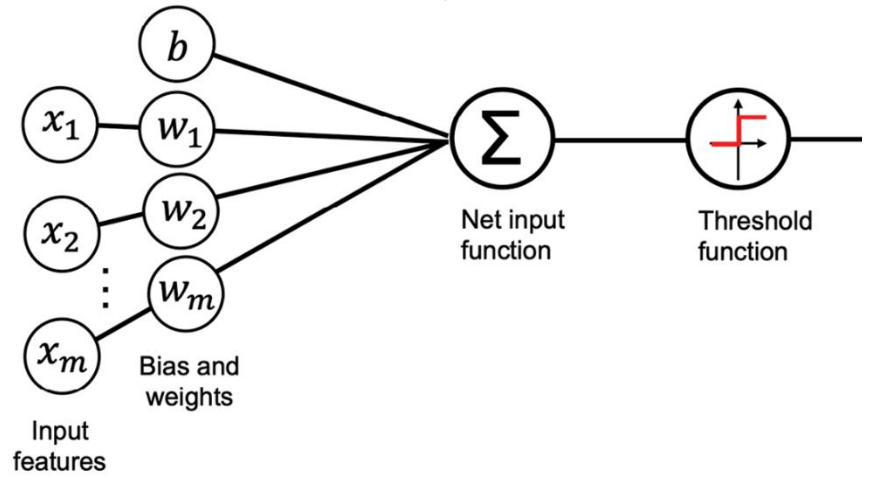
Chemical Synthesis and Drug Discovery

M. Ahmadi lab
UTK MSE

Cloud Laboratories

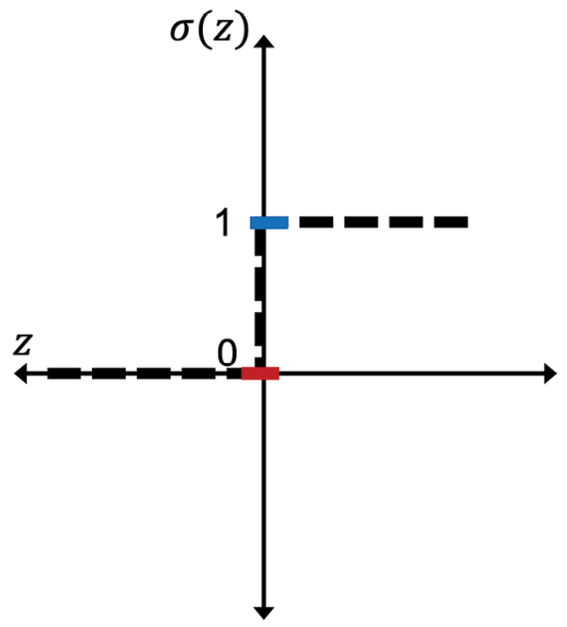
Emerald Cloud Lab,
SF and CMU

Building Linear Neuron



Input: $\boldsymbol{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix}$

Weights: $\boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$



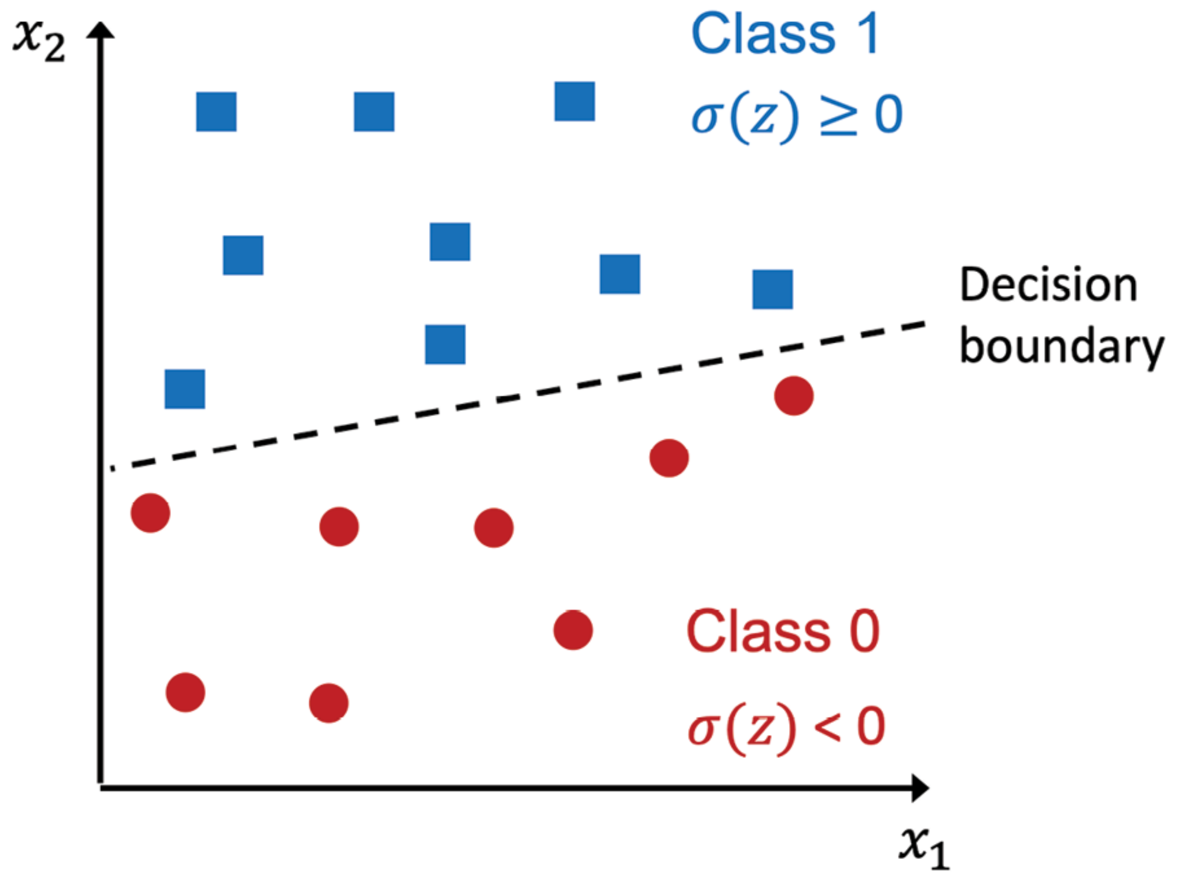
Linear transform: $\boldsymbol{z = w_1 x_1 + \dots + w_m x_m + b =}$
$\boldsymbol{= w^T x + b}$

Output: $\sigma(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$

where $z = \boldsymbol{w}^T\boldsymbol{x} + b$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Linear Neuron in 2D



Linear transform: $z = w_1 x_1 + w_2 x_2 + b$

Line: $x_2 = - w_1 / w_2 \, x_1 - b / w_2$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Training Linear Neuron

- Initialize the weights and bias unit to 0 or small random numbers
- For each training example, ***x(i)***:
- Compute the output value, ***y(i) = w^Tx(i) + b***
- Update the weights and bias unit: $w_j := w_j + \Delta w_j$ and $b := b + \Delta b$
- Where $\Delta w_j = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)x_j^{(i)}$ and $\Delta b = \eta\left(y^{(i)} - \hat{y}^{(i)}\right)$

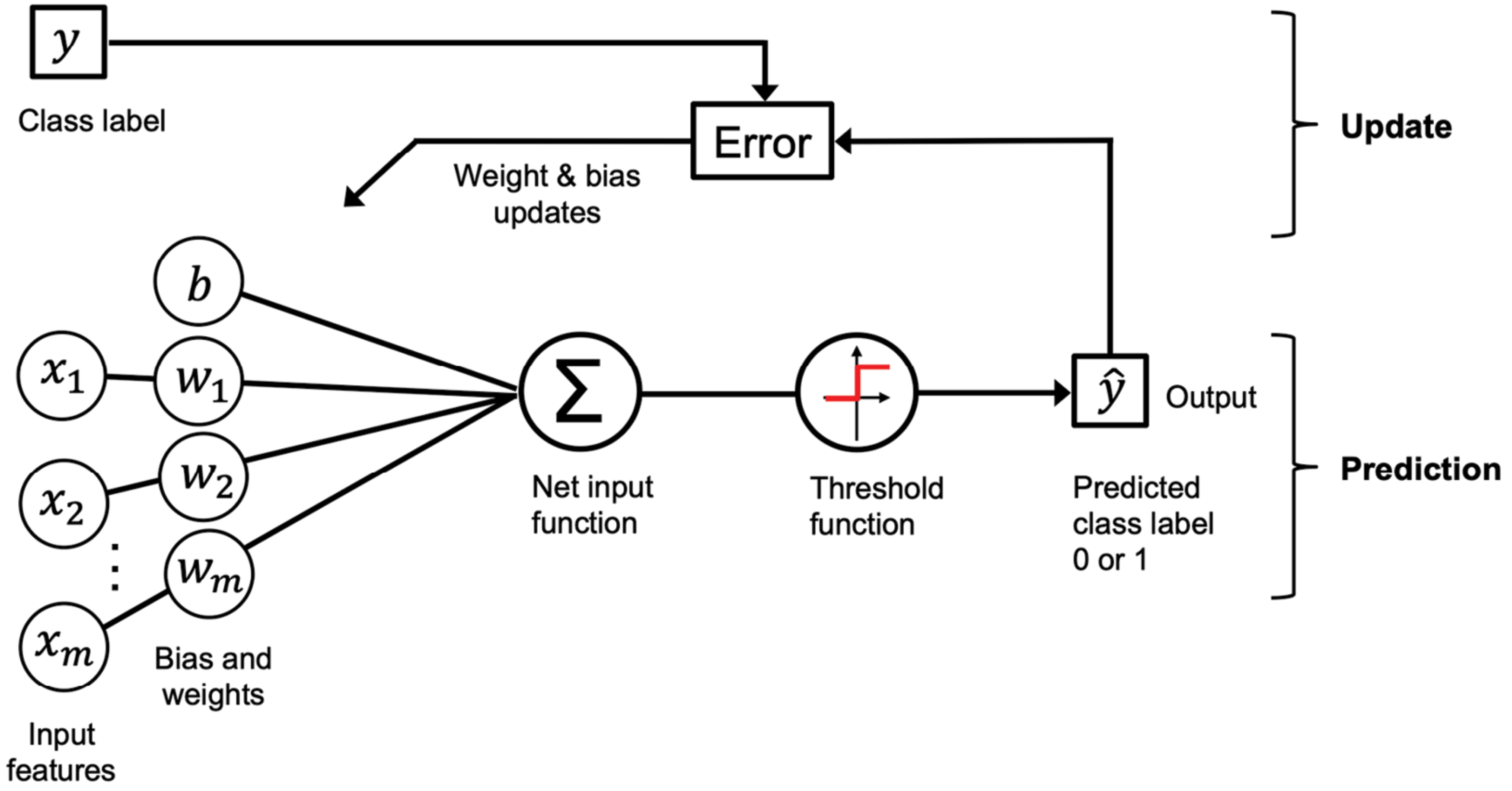
Each weight, w_j, corresponds to a feature, x_j, in the dataset,

η is the **learning rate** (typically a constant between 0.0 and 1.0),

$y^{(i)}$ is the **true class label** of the *i*th training example,

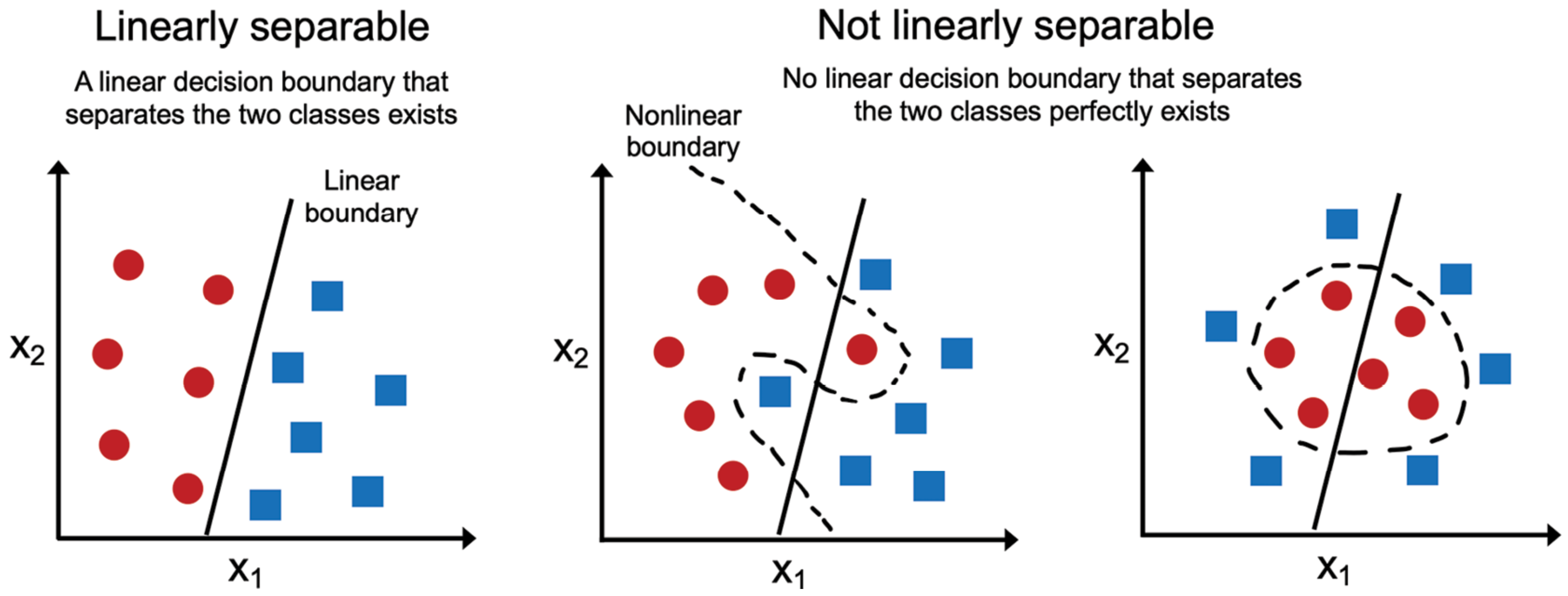
$\hat{y}^{(i)}$ is the **predicted class label**

Training Linear Neuron



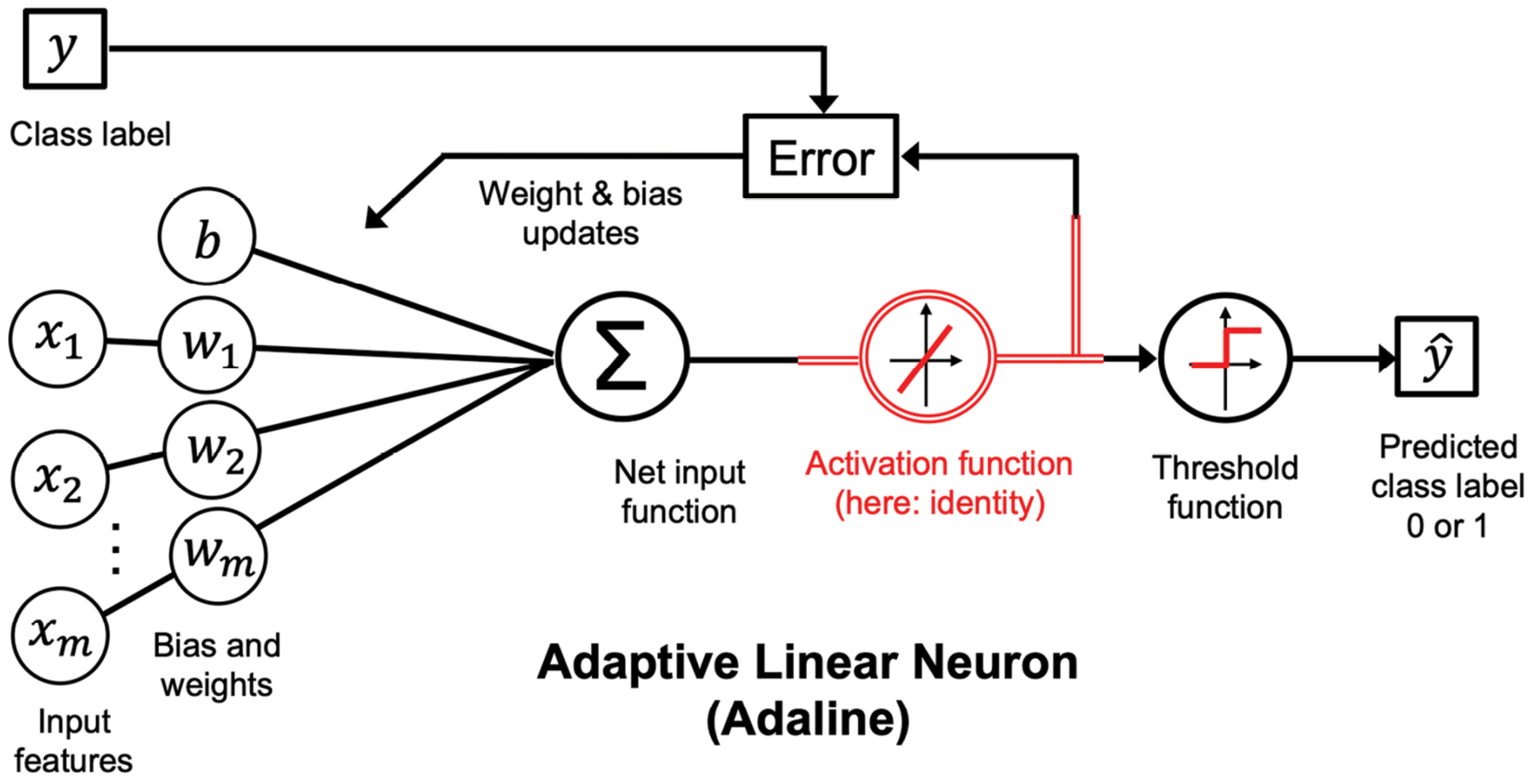
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

What problems can perceptron solve?


Linearly separable
A linear decision boundary that separates the two classes exists
Linear boundary
x_2
x_1
Not linearly separable
No linear decision boundary that separates the two classes perfectly exists
Nonlinear boundary
x_2
x_1
x_2
x_1

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Adaline

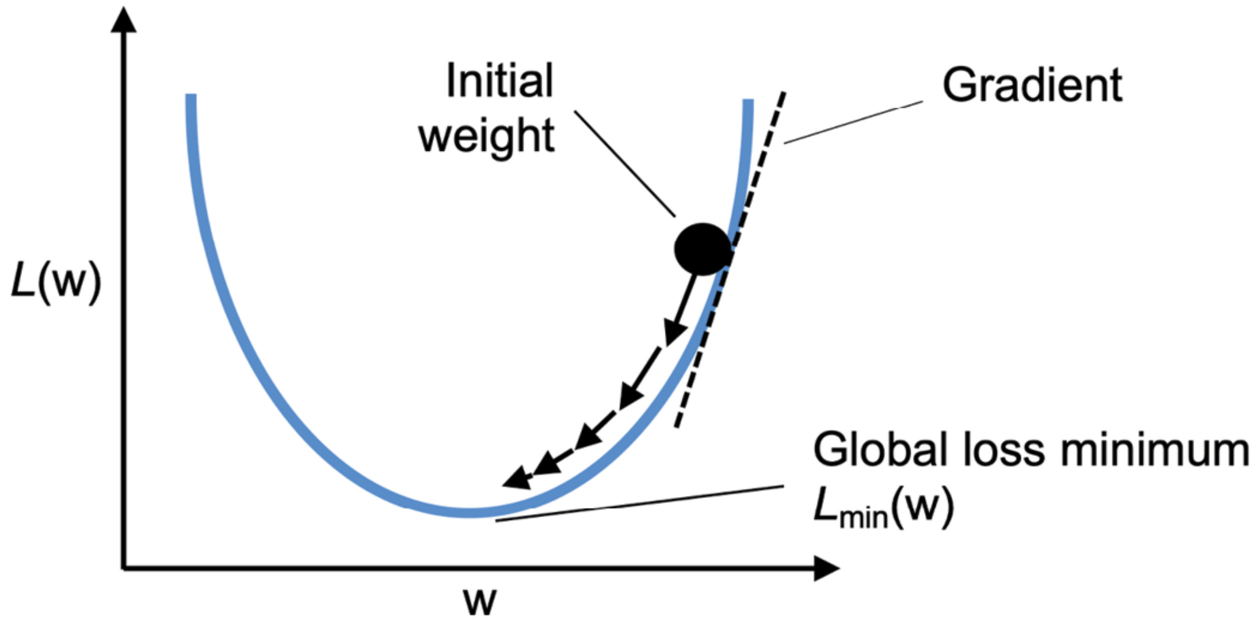


From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Adaline training

Loss function:

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



Weights update:

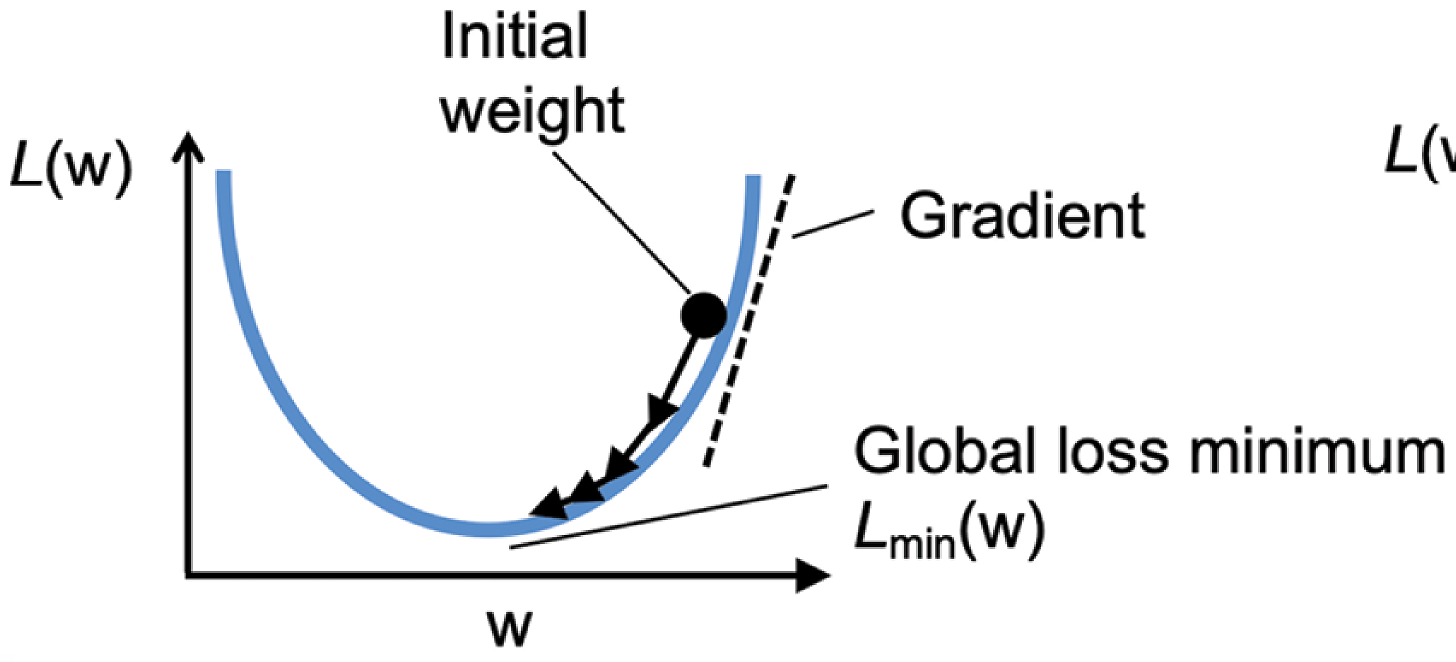
$$\boldsymbol{w} := \boldsymbol{w} + \Delta\boldsymbol{w}, \quad b := b + \Delta b$$

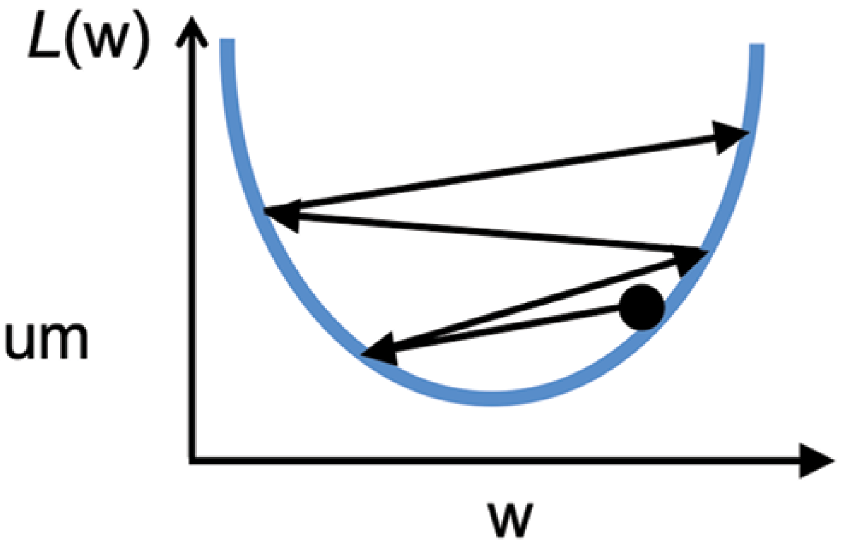
Learning rule:

$$\Delta\boldsymbol{w} = -\eta\nabla_w L(\boldsymbol{w}, b), \quad \Delta b = -\eta\nabla_b L(\boldsymbol{w}, b)$$

From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

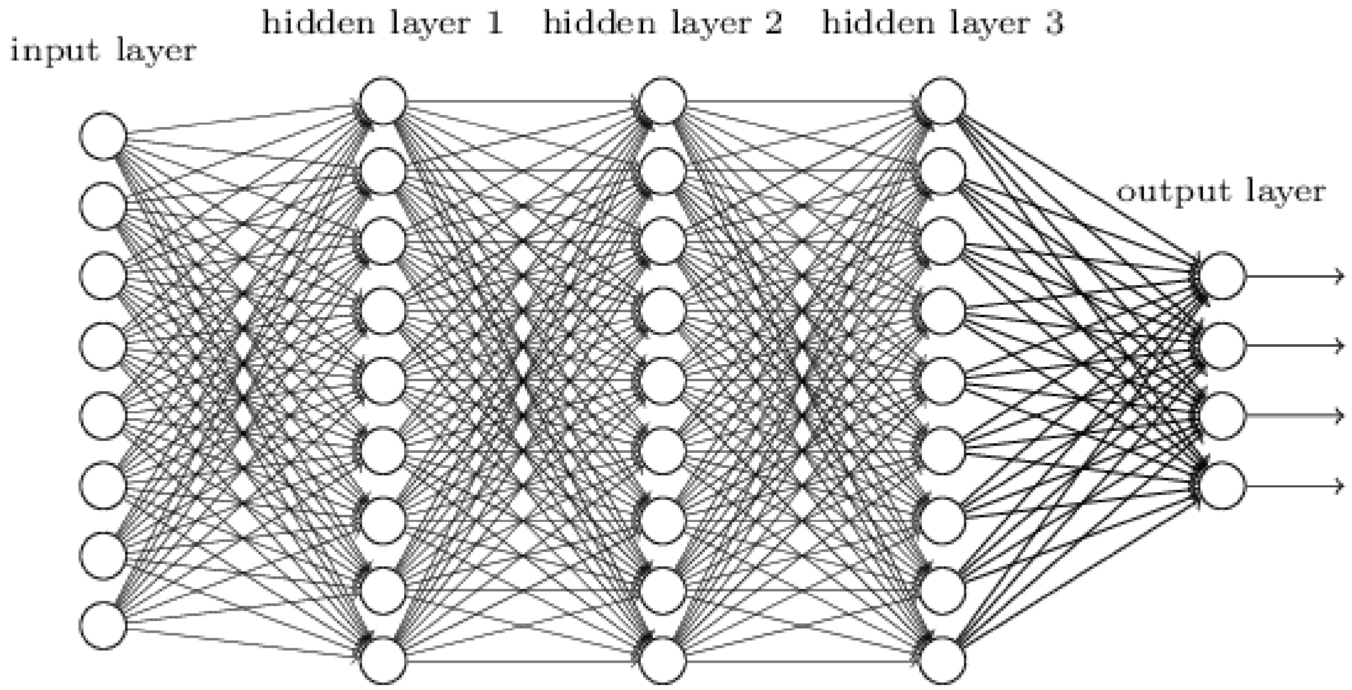
Role of learning rate





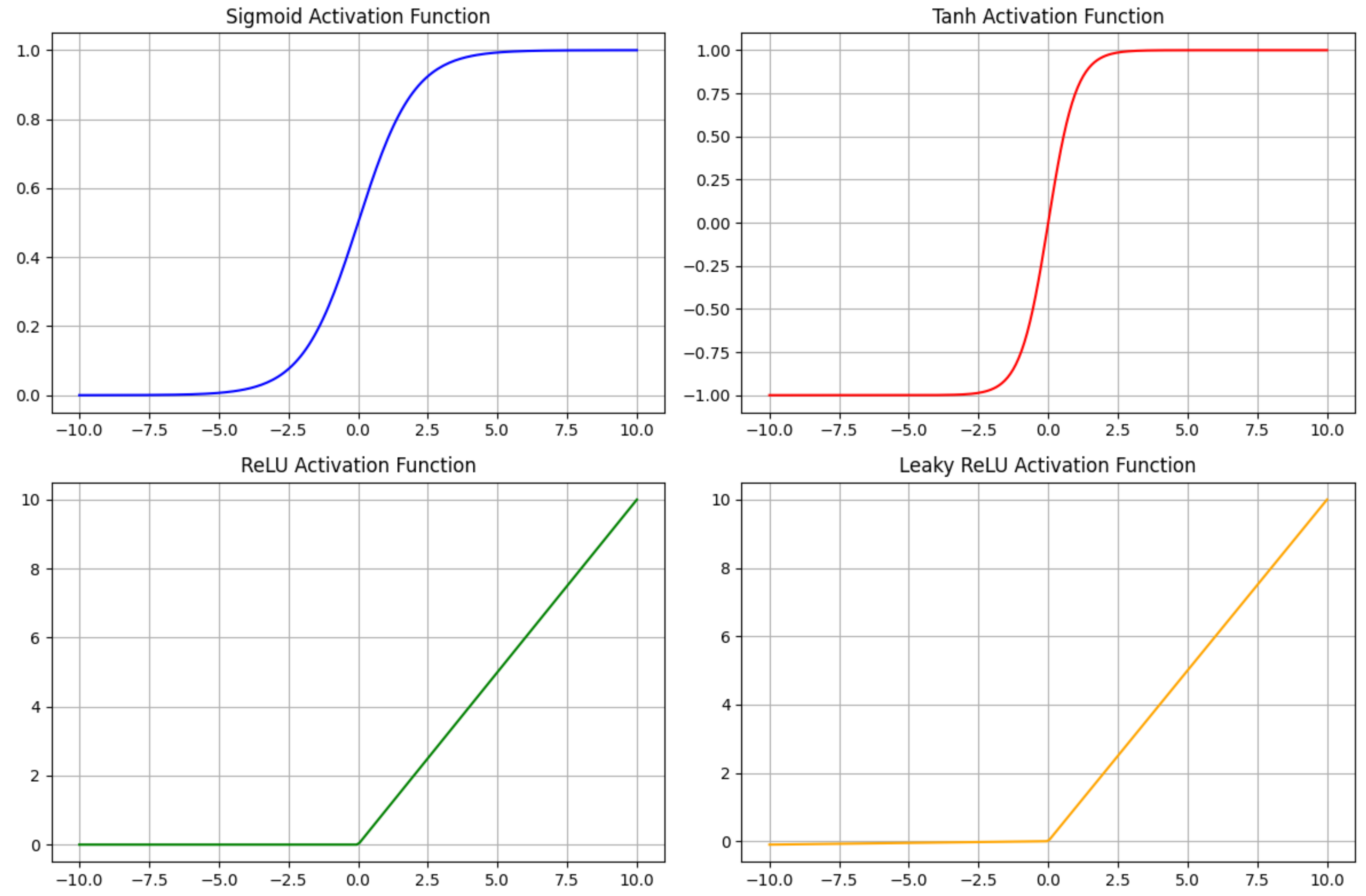
From S. Raschka, Machine Learning with PyTorch and Scikit-Learn

Putting Neurons Together



- Composed of multiple layers of artificial neurons.
- Each layer processes inputs received, applies a transformation (weights, biases, activation function), and passes the output to the next layer.
- Training a DNN involves adjusting weights and biases using backpropagation and a chosen optimization algorithm.
- The deep architecture enable the network to learn complex and abstract patterns in data.

Activation functions



Loss functions for supervised ML

A loss function, also known as a cost function, quantifies the difference between the predicted values and the actual target values. It guides the training of neural networks by providing a measure to minimize during optimization

Mean Squared Error (MSE): Used for regression problems.

$$MSE = \frac{1}{N} \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Measures the average squared difference between actual and predicted values

Cross-Entropy Loss: Used for classification problems.

$$CE = - \sum_{i=1}^{N} y_i \log(\hat{y}_i)$$

Measures the performance of a classification model whose output is a probability value between 0 and 1

- Loss functions provide the primary feedback signal for learning.
- The choice of loss function can significantly affect the model's performance and convergence

Backpropagation

Backpropagation is a mechanism used to update the weights in a neural network efficiently, based on the error rate obtained in the previous epoch (i.e., iteration). It effectively distributes the error back through the network layers

- **Forward Pass:** Calculating the predicted output, moving the input data through the network layers
- **Loss Function:** Determining the error by comparing the predicted output to the actual output
- **Backward Pass:** Computing the gradient of the loss function with respect to each weight by the chain rule
- **Weight Update:** Adjusting the weights of the network in a direction that minimally reduces the loss (gradient descent)

Input Data → Forward Pass → Calculate Loss → Backward Pass → Update Weights

https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c

How to not get lost?

- A vast array of network architectures ranging from Multilayer Perceptrons (MLPs) to Graph Neural Networks and Transformers
- Each architecture has unique characteristics suited for different types of data and tasks, ways to define the architecture, and so on
- Numerous methods to engineer loss functions
- Numerous ways to implement regularization

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

Some useful resources:

https://udlbook.github.io/udlbook/

https://dmol.pub/ml/classification.html

https://keras.io/examples/