Argonne Intro to HPC Bootcamp

An Introduction to Linear Regression with Applications to AI

Wilbur Ouma

HPC Partnerships and Outreach Specialist, Argonne Leadership Computing Facility
Argonne National Laboratory

August 13, 2025

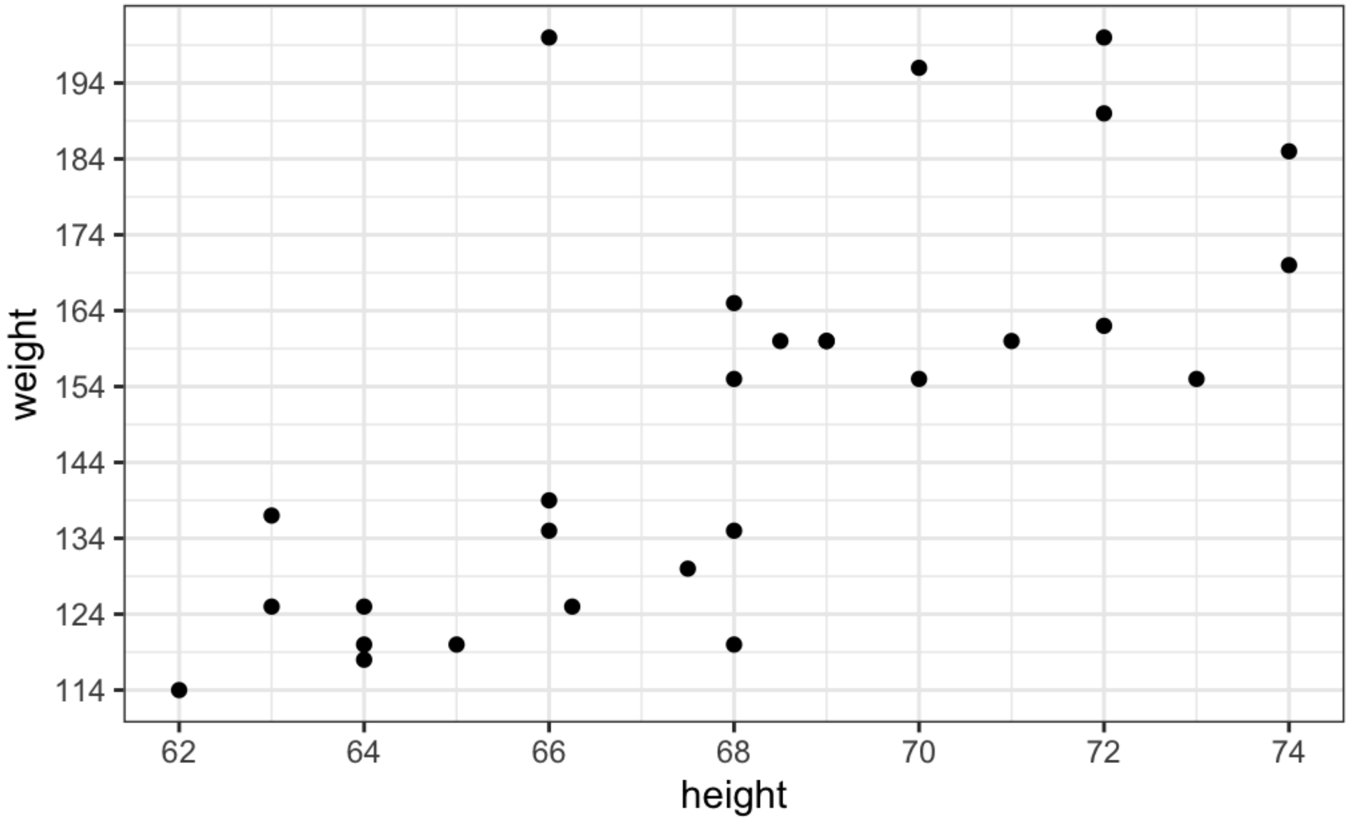
Motivation for Linear Regression

- Allows us to investigate the relationship between two or more variables statistically
- Basic introduction to supervised learning
- Can be thought of as a building block of artificial neural networks (many perceptrons)
- Learning Objectives

    1. Fit a SLR model to data and interpret model coefficients (parameters)
    2. Build a Perceptron to solve a regression problem
    3. Build and train a deep neural network using Keras

Simple Linear Regression: Model Definition

- The goal for SLR is to investigate the relationship between the ***response*** (***Y***) and the ***predictor*** (***X***) variables
- Recall from high school algebra: $\boldsymbol{y = b + mx}$
  - —Where ***m*** is the slope and ***b*** is the y-intercept



Simple Linear Regression: Model Definition

- The general form of the SLR model closely resembles the algebraic equation ($\boldsymbol{y = b + mx}$):

$$Y = \beta_0 + \beta_1 X + \epsilon$$

- For an individual observation (x_i, y_i), the regression equation becomes:

$$y_i = \beta_0 + \beta_1 x_i + \epsilon_i$$

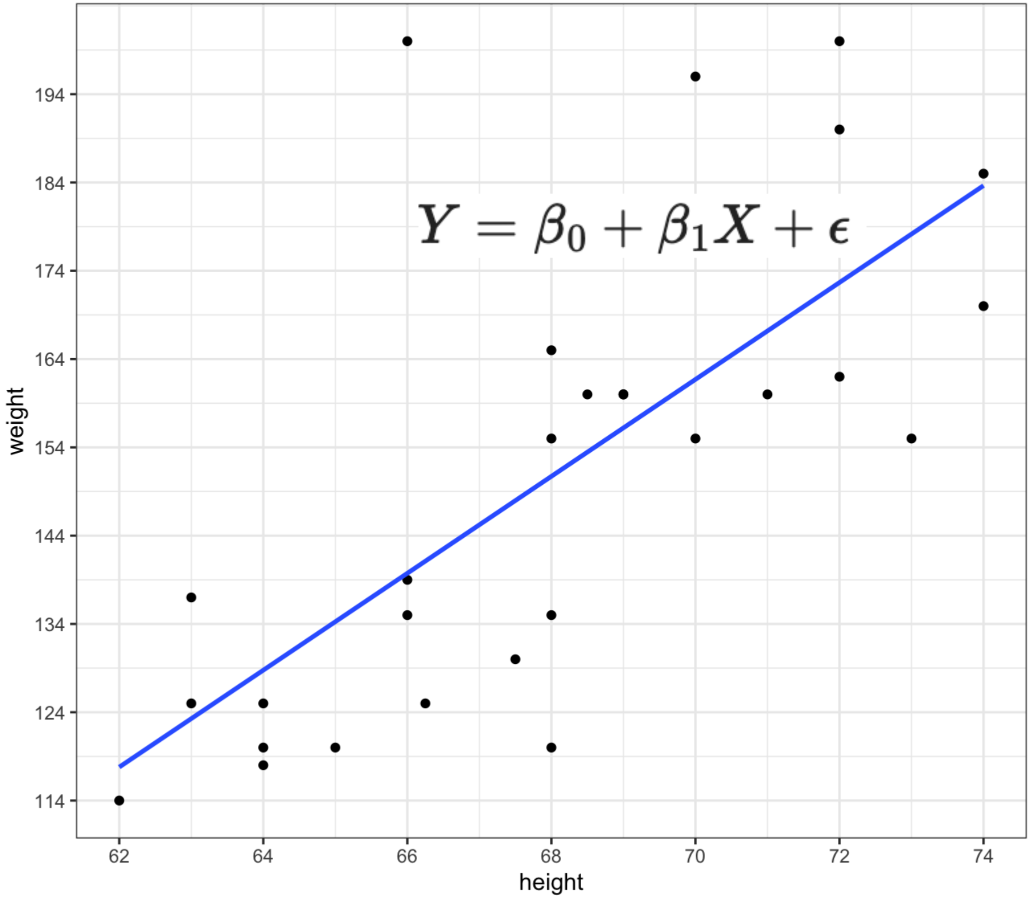
- Where
  - — β_0 is the is the population y-intercept,
  - — β_1 is the population slope
  - — ϵ_i is the error or deviation of an observation from regression line

- Together, β_0 and β_1 are the (unknown) population model ***coefficients*** or ***parameters***

- The goal of regression is to estimate parameters ($\hat{\beta}$) to describe the relationship between ***Y*** and ***X***

- We estimate $\hat{\beta}$ using the method of Ordinary Least Squares (OLS)

Simple Linear Regression: Errors (Loss)



- Predicted value ($\hat{y}_i$) based on the $\hat{x}_i$ is obtained by

$$fit_i = \hat{y}_i = \hat{\beta}_0 + \hat{\beta}_1 x_i$$

- An error is defined by

$$res_i = \epsilon_i = y_i - \hat{y}_i$$

- We quantify the total error (loss)

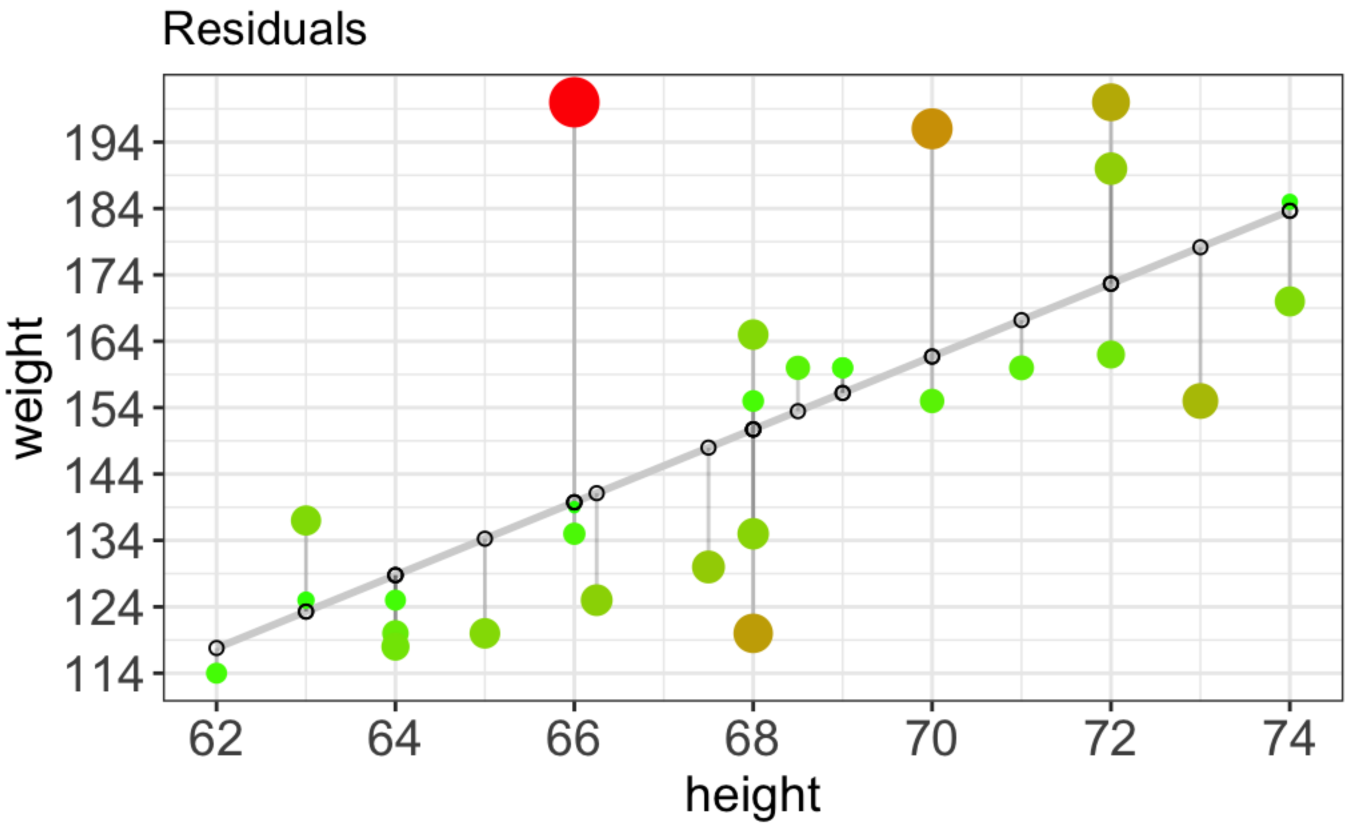
$$RSS = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$MSE = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{y}^{(i)} - y^{(i)}\right)^2$$

- Goal: obtain least squares estimates of β to minimize MSE

Simple Linear Regression: Errors (Loss)

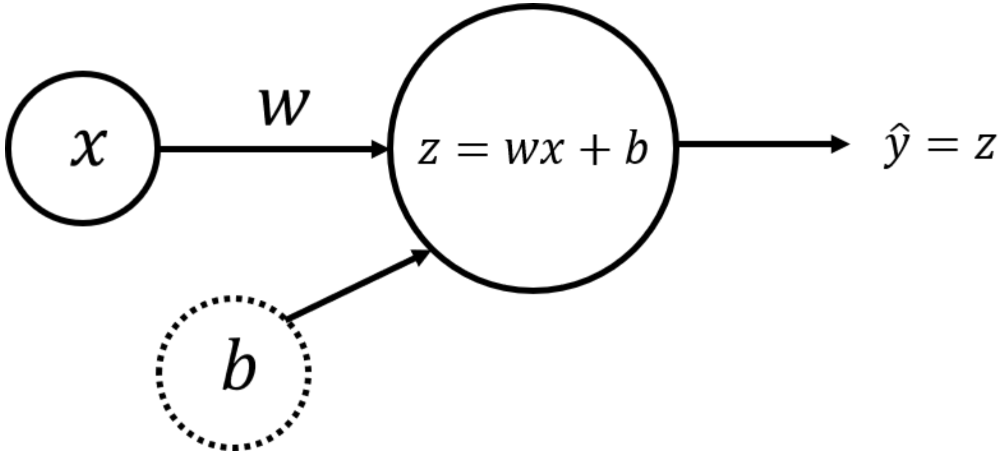
Errors, visualized



Hands-On!

SLR with Perceptron

- We will construct a ‘neuron’ corresponding to a SLR model, and train with **gradient descent**

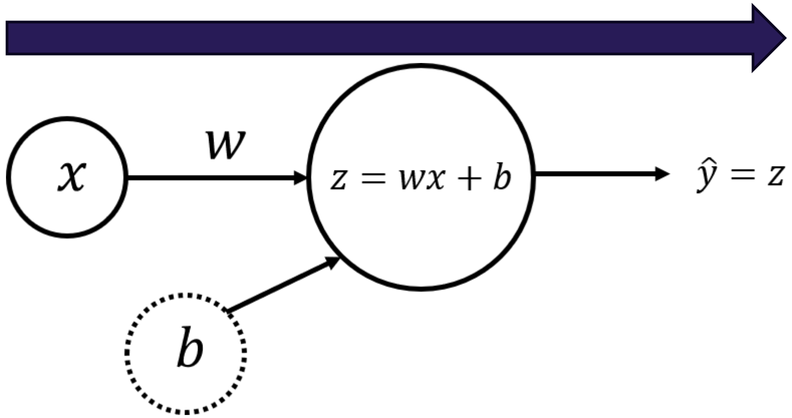


$$\hat{y} = wx + b$$

- **Weight** (*w*) and **bias** (*b*) are the parameters that will get updated when you **train** the model
- As in the previous SLR model, the goal is to identify parameters that minimize the average loss function: **cost function**

$$\mathcal{L}(w, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)^2$$

Training the model



- After defining the model structure,
  - — initialize parameters to some random values (or set to 0) and
  - — **update** as the training progresses
- For each training example (x_i, yi), predict $\hat{y}_i$:

$$z_i = wx_i + b,$$

$$\hat{y}_i = z_i,$$

where $i = 1, \ldots, m$

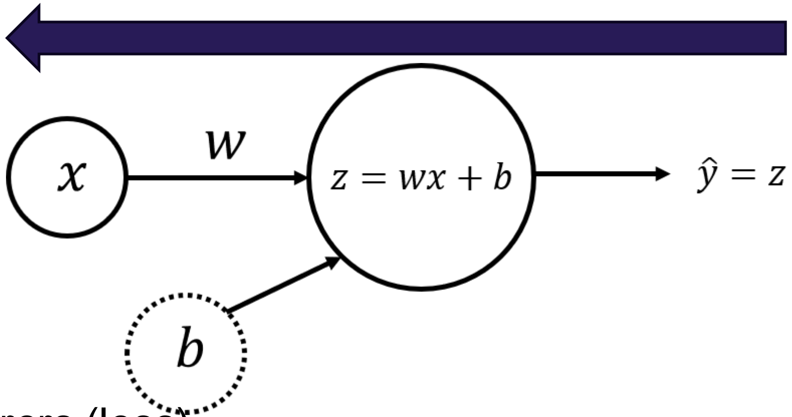
- Training examples are a vector $\boldsymbol{X}$ of size $(1 \times m)$
- Perform scalar multiplication of $\boldsymbol{X}$ by a scalar $\boldsymbol{w}$, adding $\boldsymbol{b}$.

$$Z = wX + b,$$

$$\hat{Y} = Z,$$

- This set of calculations is called **forward propagation**

Training the model

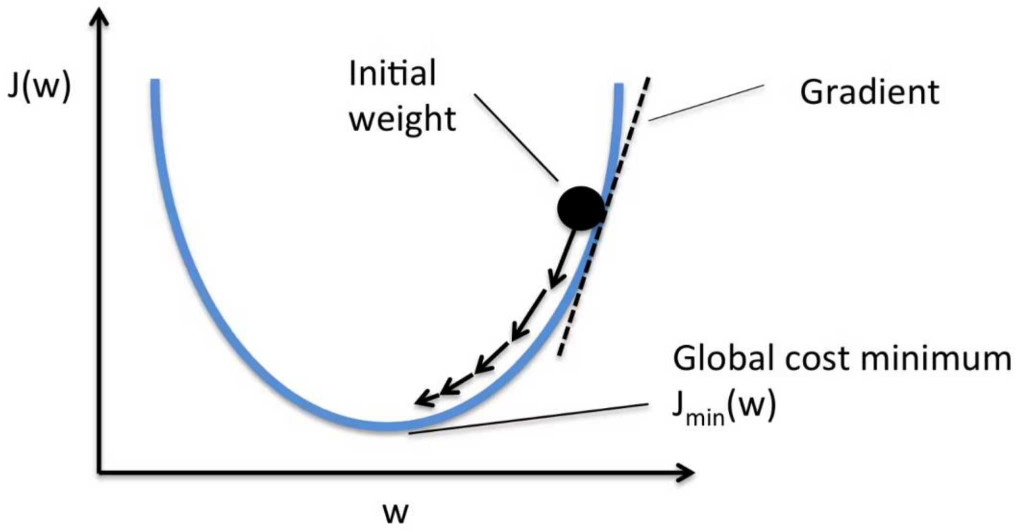


- Given $\hat{Y}$, calculate the cost function

$$\mathcal{L}(w, b) = \frac{1}{2m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)^2$$

  — Aim is to optimize the cost function during training, hence minimize the errors (loss)

- We minimize the cost function by use of an optimization algorithm, like **gradient descent**
  — Attempt to identify parameter values that minimize the cost function by taking partial derivatives of the cost function



We calculate partial derivatives as:

$$\frac{\partial \mathcal{L}}{\partial w} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right) x_i,$$

$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} \left(\hat{y}_i - y_i\right)$$

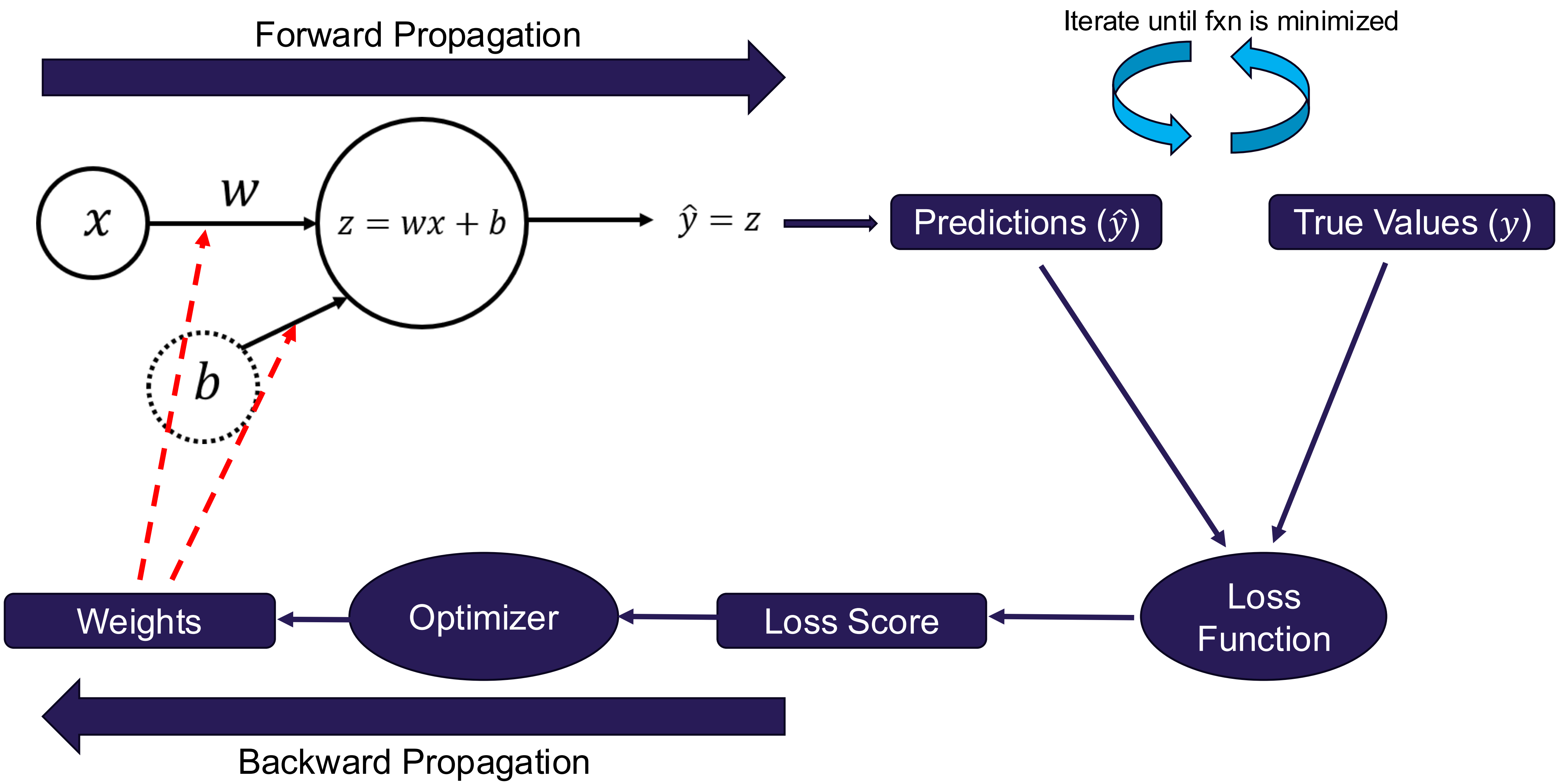
We then update the parameters iteratively using the expressions:

$$w = w - \alpha \frac{\partial \mathcal{L}}{\partial w},$$

$$b = b - \alpha \frac{\partial \mathcal{L}}{\partial b},$$ where α is the learning rate

- This set of calculations is called **backward propagation**

Training the model



Methodology to build a neural network

- Define the neural network structure (# of input units, # of hidden units, etc)

- Initialize the model's parameters

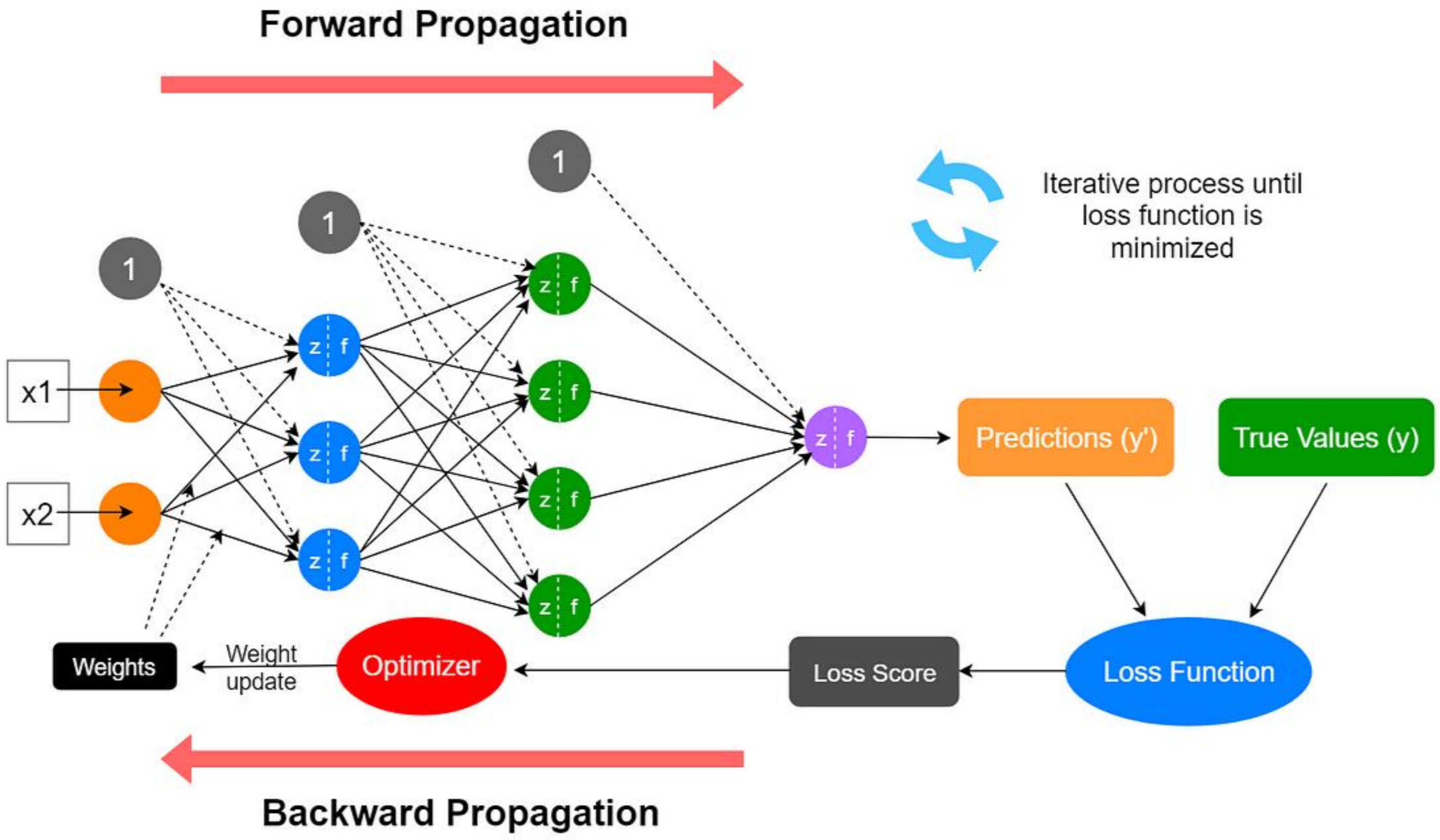
- Loop (iterate)
  - — Implement forward propagation (calculate the perceptron output)
  - — Implement backward propagation (to get the required corrections for the parameters)
  - — Update parameters

- Make predictions with “best” parameters

Hands-On!

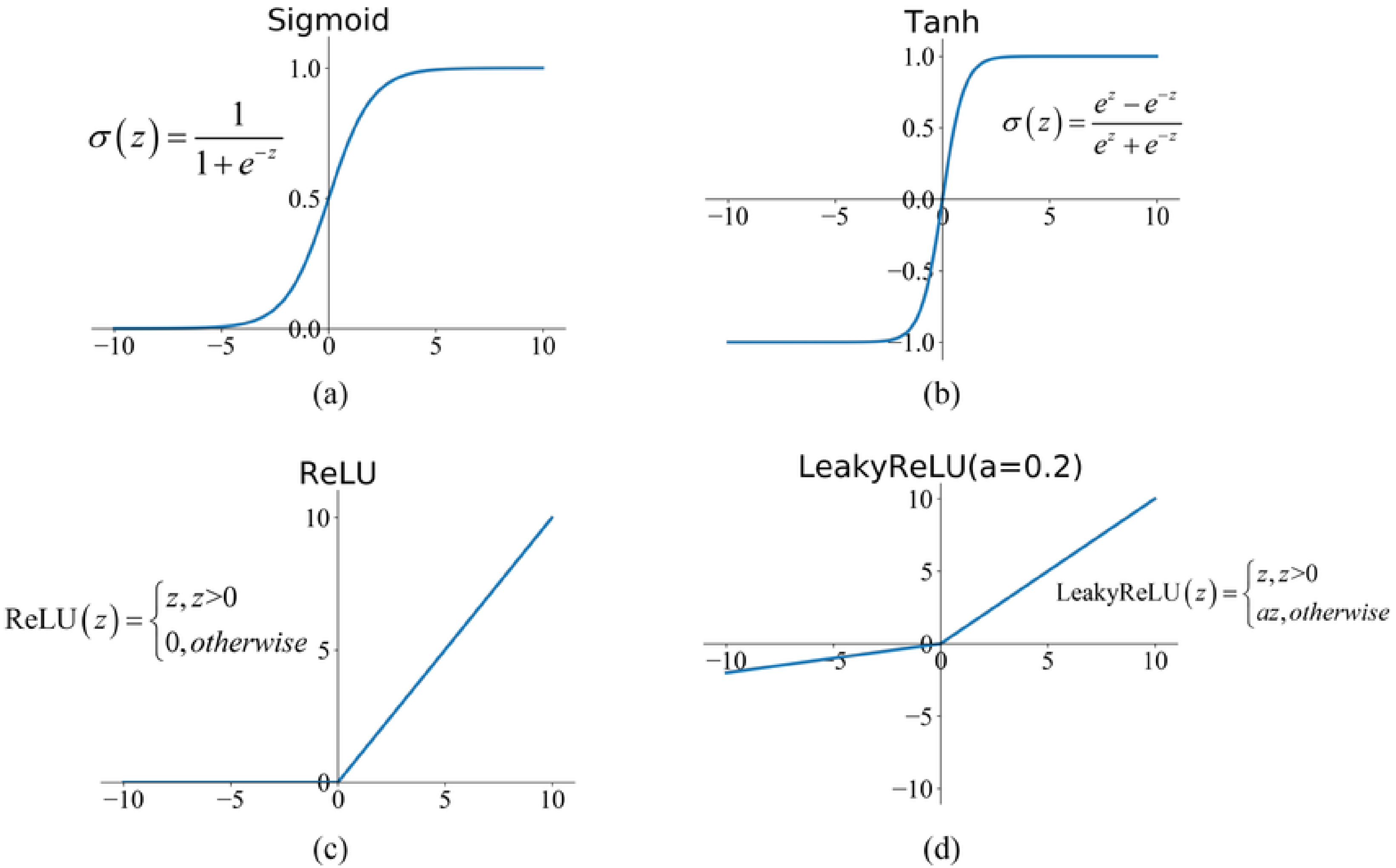
Let's Dive Deep! An Introduction to Deep Learning

The ***deep*** in deep learning: successive layers of data representation



Let's Dive Deep! An Introduction to Deep Learning

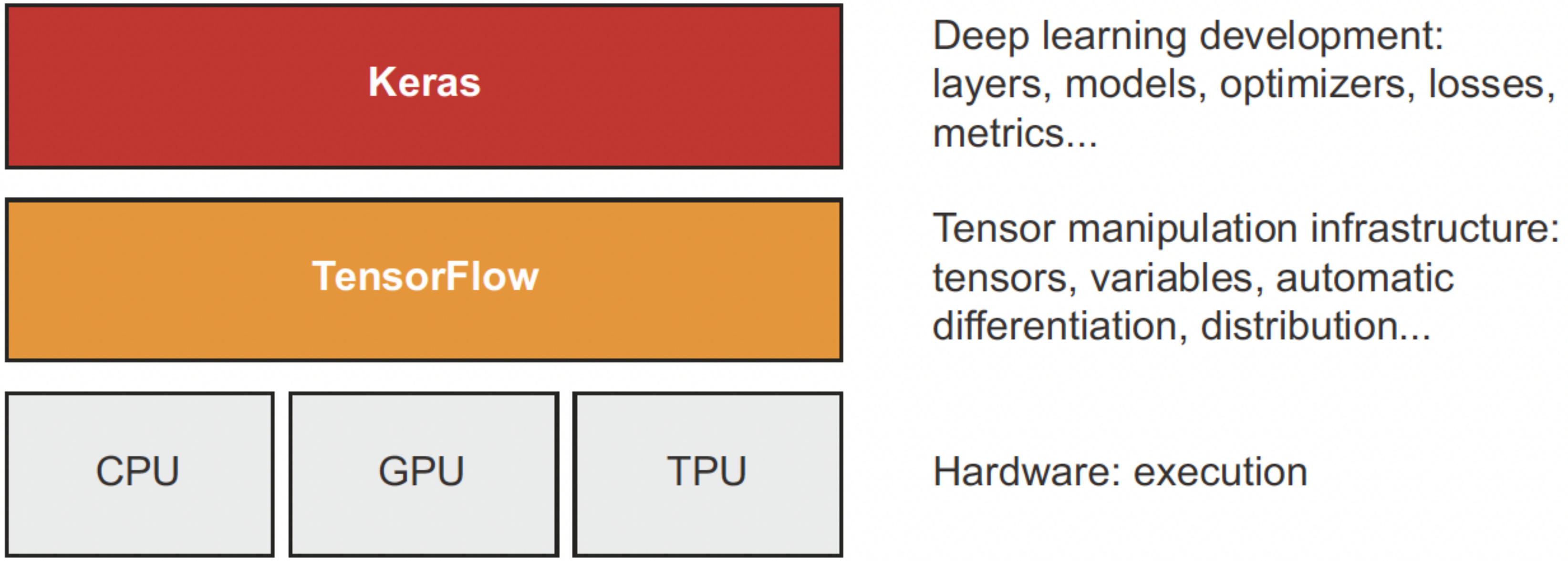
Commonly used activation functions



Argonne NATIONAL LABORATORY

Let's Dive Deep! An Introduction to Deep Learning

You don’t have to build neural nets from scratch — somebody already did this for you!



Hands-On!

Questions?