# 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

Argonne NATIONAL LABORATORY

# 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: $y = b + mx$
  - Where **m** is the slope and **b** is the y-intercept
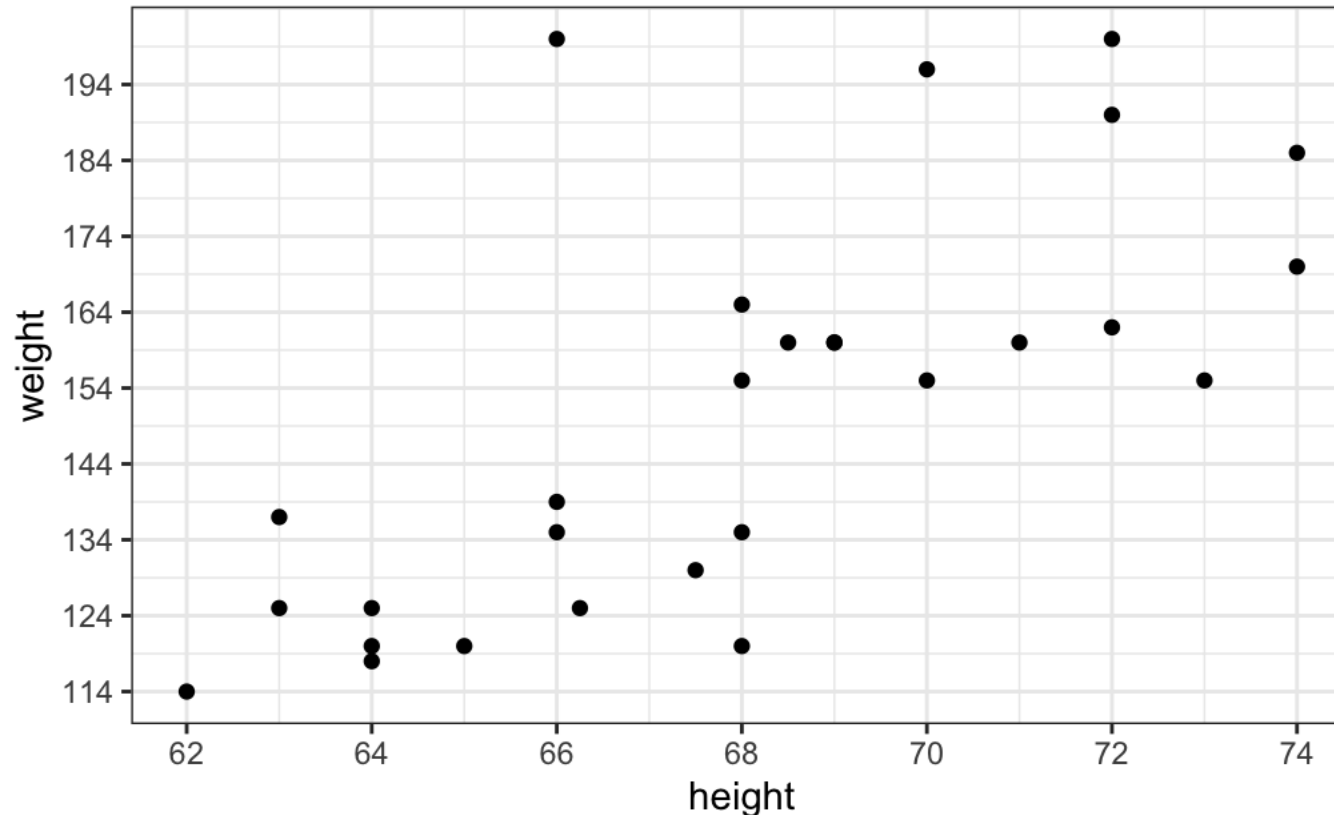
# Simple Linear Regression: Model Definition

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

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

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

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

- Where
  - $\beta_0$ is the is the population y-intercept,
  - $\beta_1$ is the population slope
  - $\epsilon_i$ is the error or deviation of an observation from regression line

- Together, $\beta_0$ and $\beta_1$ 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)

Argonne
NATIONAL LABORATORY

# Simple Linear Regression: Errors (Loss)



$$Y = \beta_0 + \beta_1 X + \epsilon$$
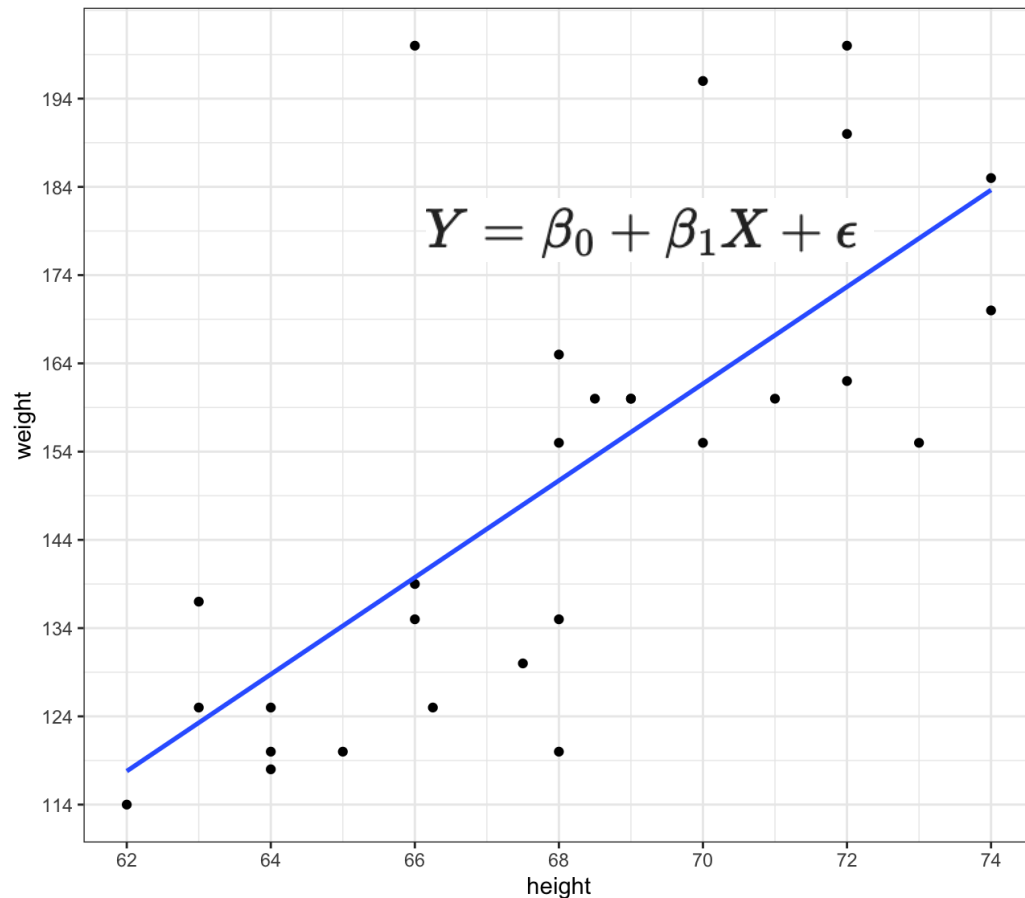
- Predicted value ($\hat{y}_i$) based on the $\hat{x}_i$ is obtained by

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

- An error is defined by

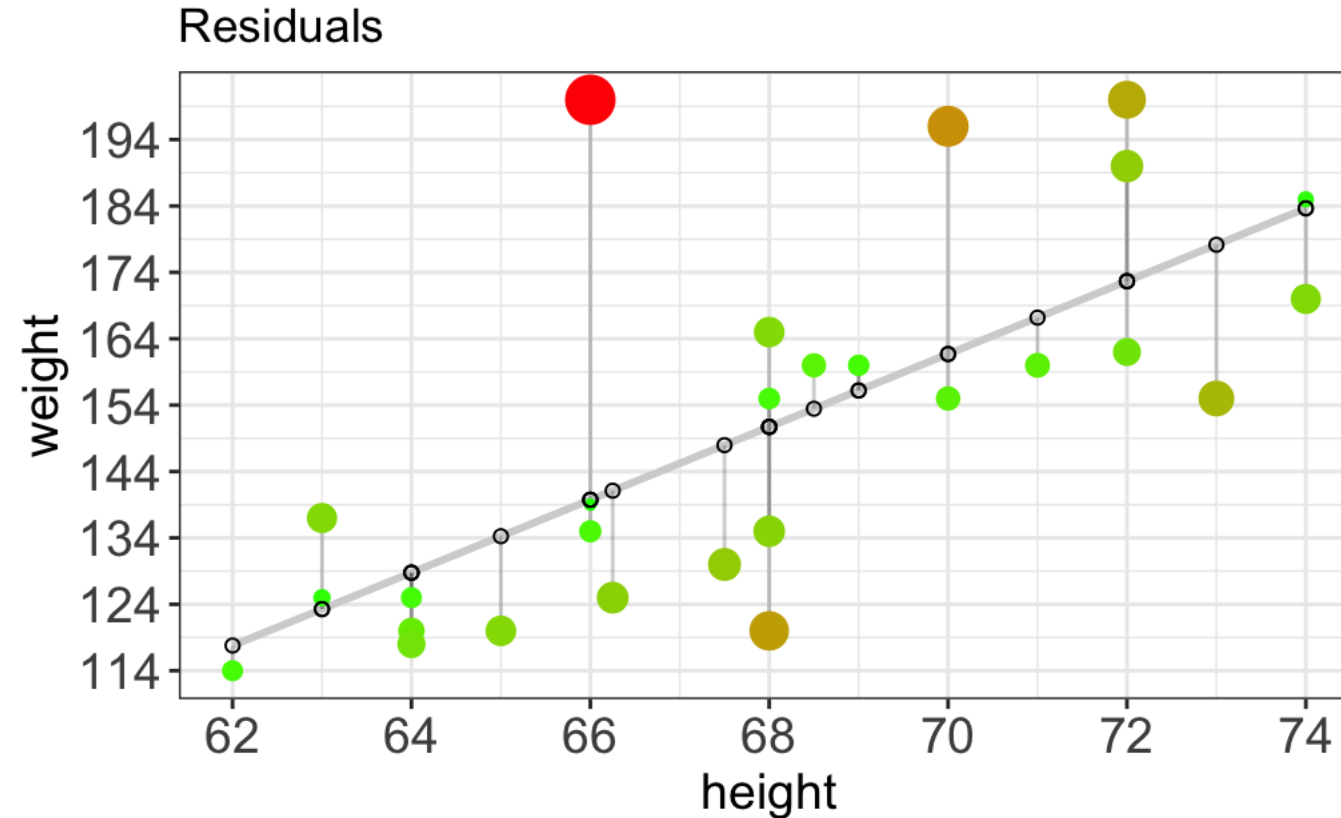$$res_i = \epsilon_i = y_i - \hat{y}_i$$

- 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 $\beta$ to minimize MSE

# Simple Linear Regression: Errors (Loss)

**Errors, visualized**

# Hands-On!

Argonne Leadership Computing Facility
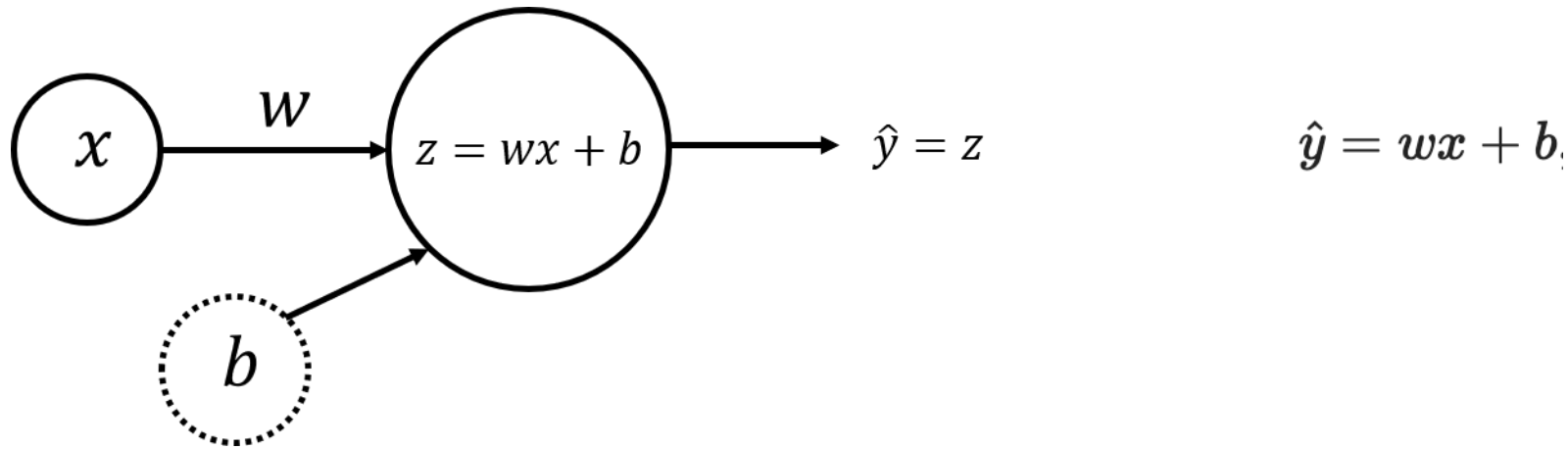
# SLR with Perceptron

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



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

- **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**

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

# 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 $\hat{y}_i$ :
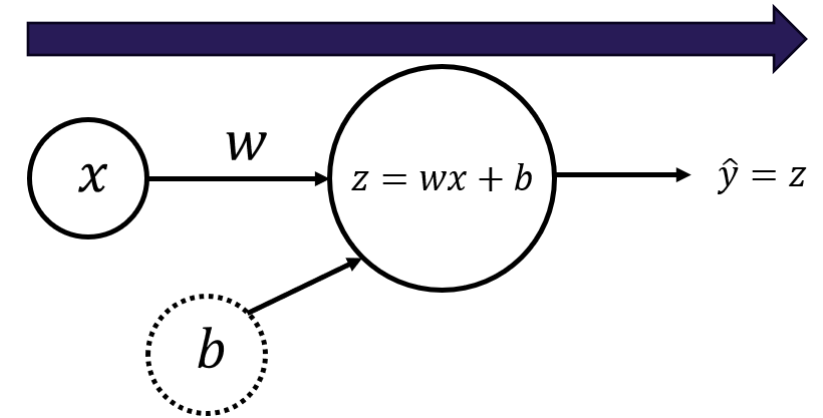
$$z_i = wx_i + b,$$

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

where $i = 1, \ldots, m$

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

$$Z = wX + b,$$

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

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

Argonne NATIONAL LABORATORY

# Training the model

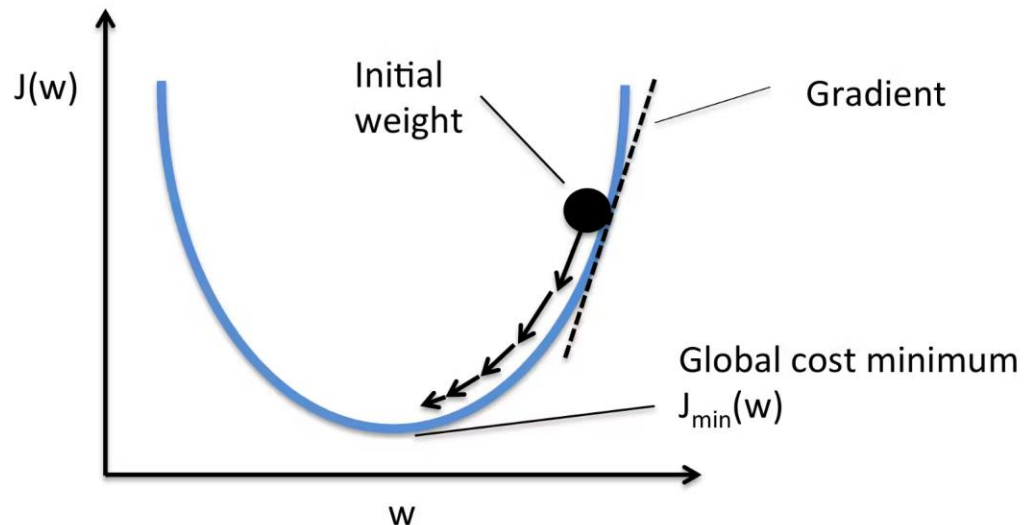$$x \xrightarrow{w} z = wx + b \longrightarrow \hat{y} = z$$

$$b$$

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

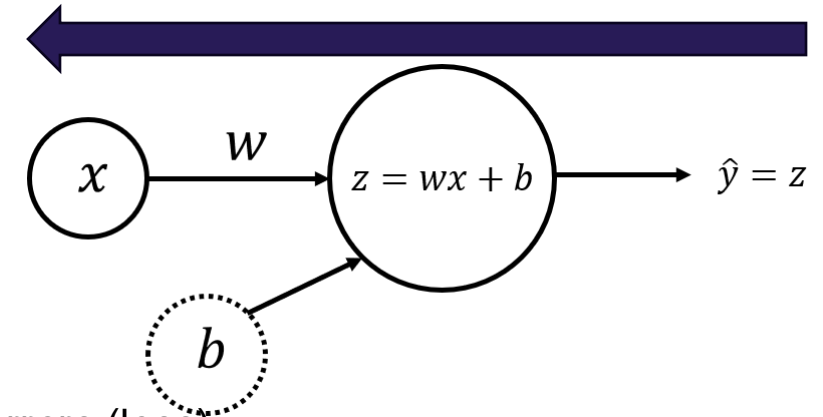$$\mathcal{L}(w, b) = \frac{1}{2m} \sum_{i=1}^{m} (\hat{y}_i - y_i)^2$$

  —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:

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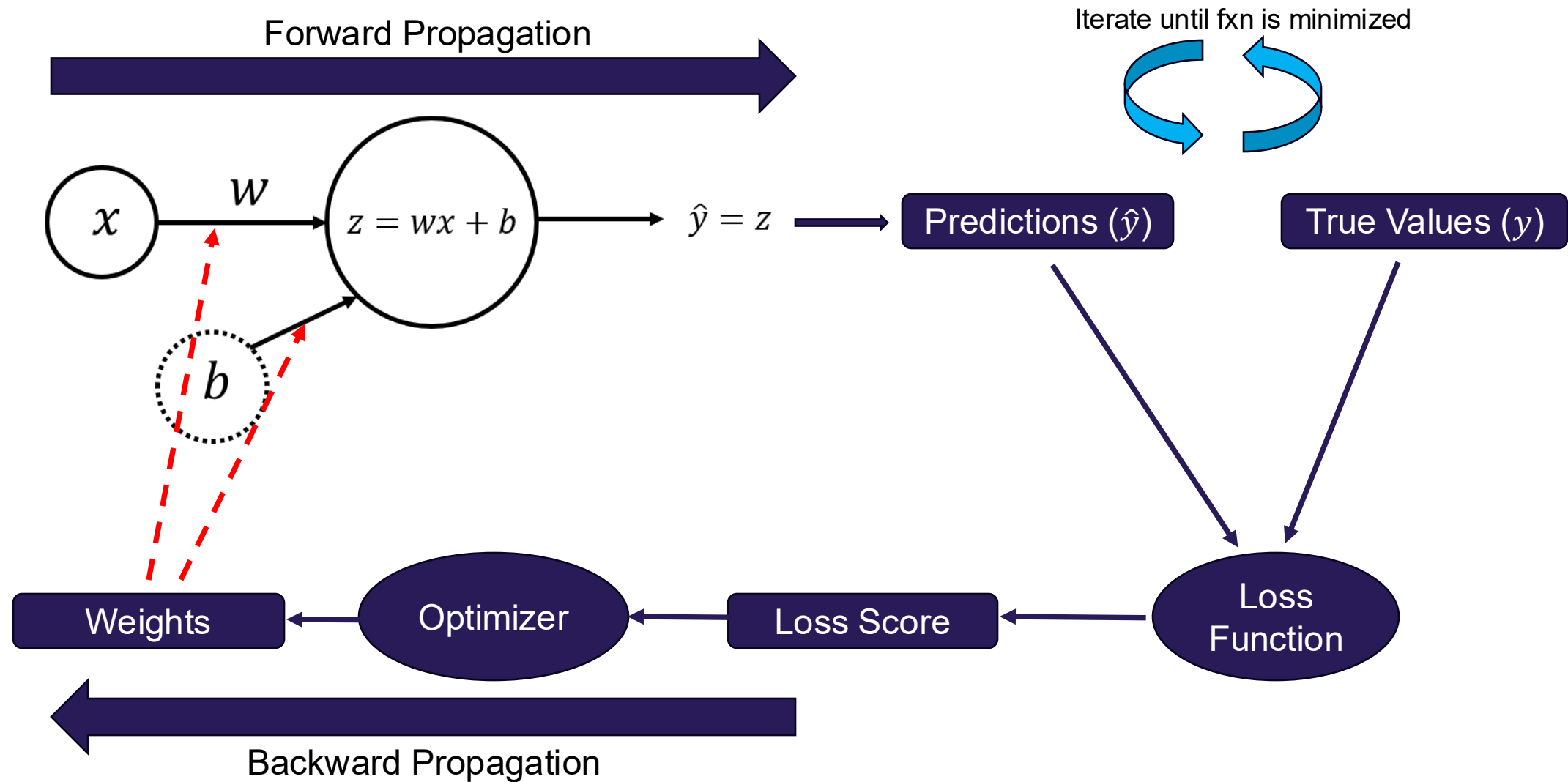
$$\frac{\partial \mathcal{L}}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} (\hat{y}_i - y_i)$$

We then update the parameters iteratively using the expressions:

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

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

J(w)

Initial weight

Gradient

Global cost minimum
$J_{min}(w)$

w

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

Argonne
NATIONAL LABORATORY

# Training the model



Forward Propagation

Iterate until fxn is minimized

$z = wx + b$

$\hat{y} = z$

Predictions ($\hat{y}$)

True Values ($y$)

Weights

Optimizer

Loss Score

Loss Function

Backward Propagation

Argonne
NATIONAL LABORATORY

# 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

Argonne
NATIONAL LABORATORY

# Hands-On!
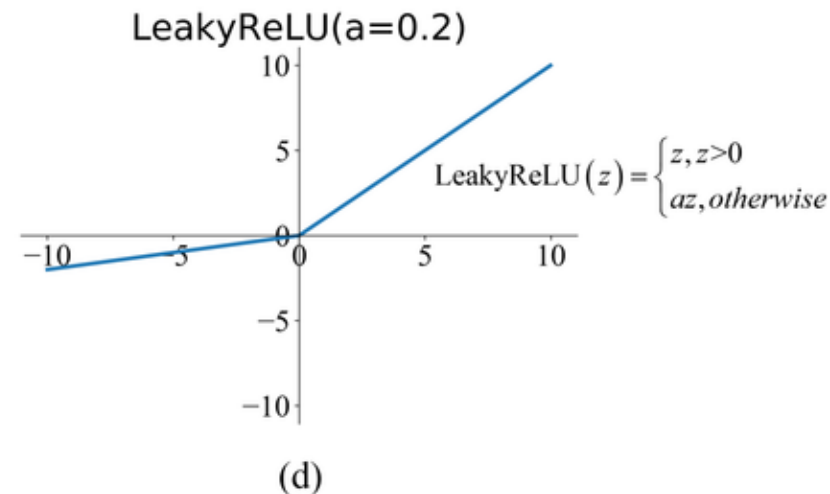
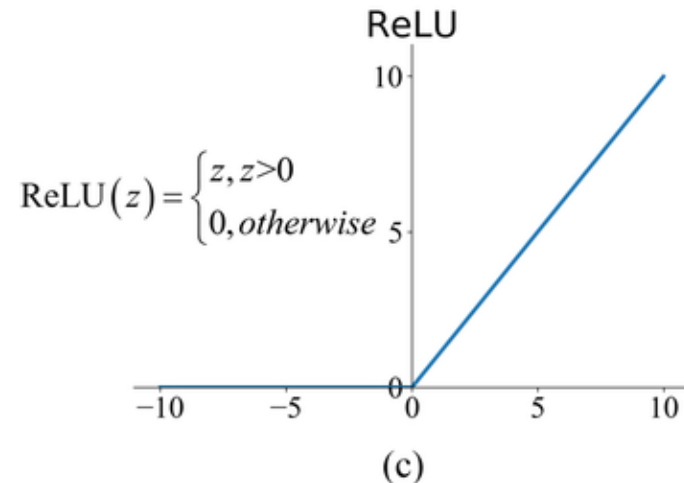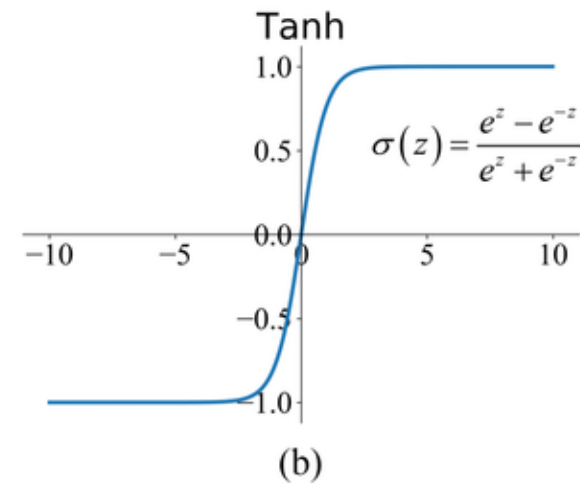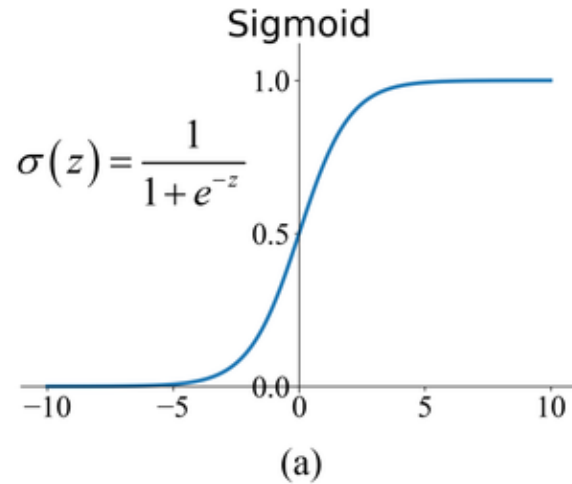Argonne Leadership Computing Facility

# Let's Dive Deep! An Introduction to Deep Learning

The **deep** in deep learning: successive layers of data representation
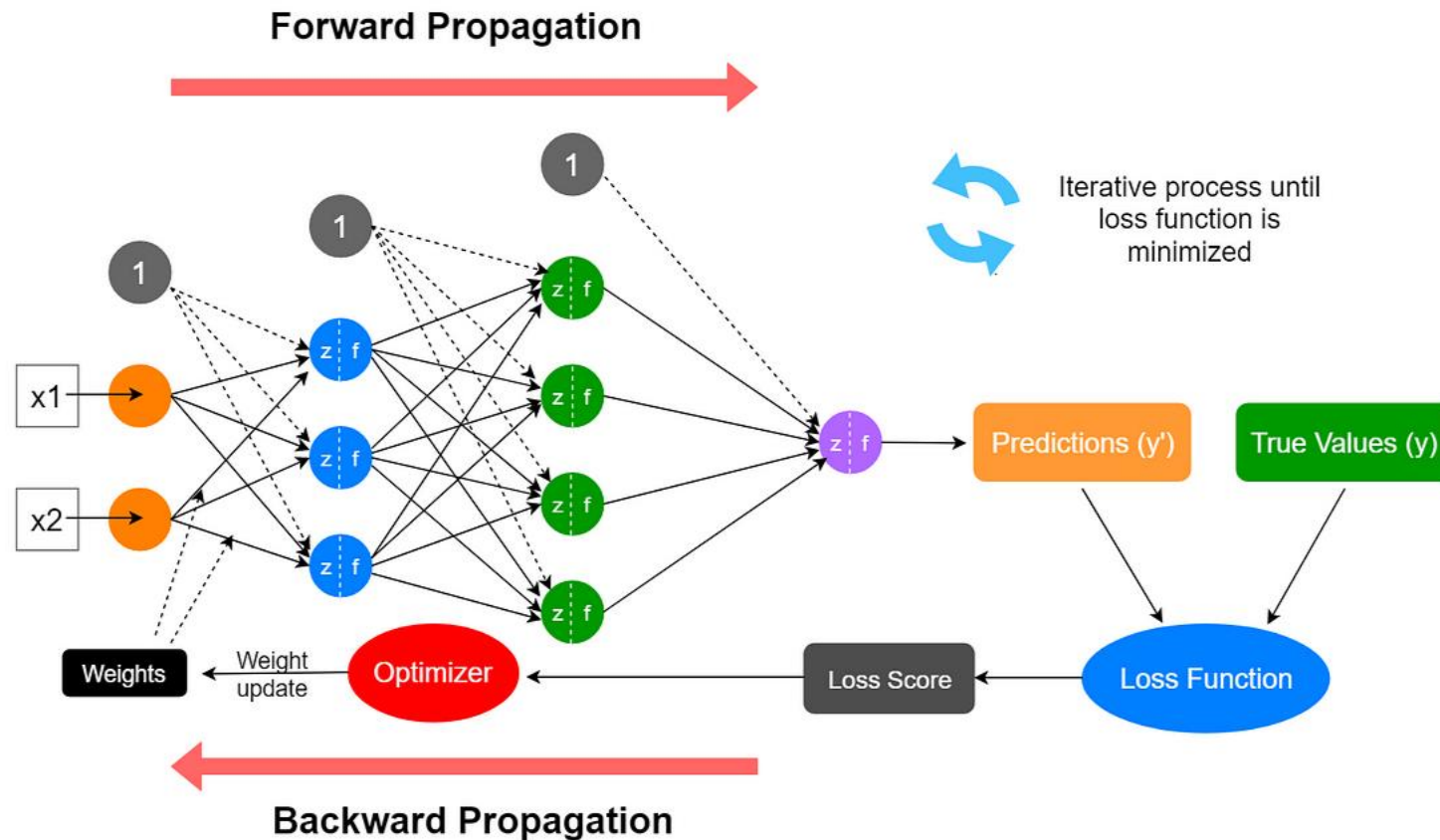
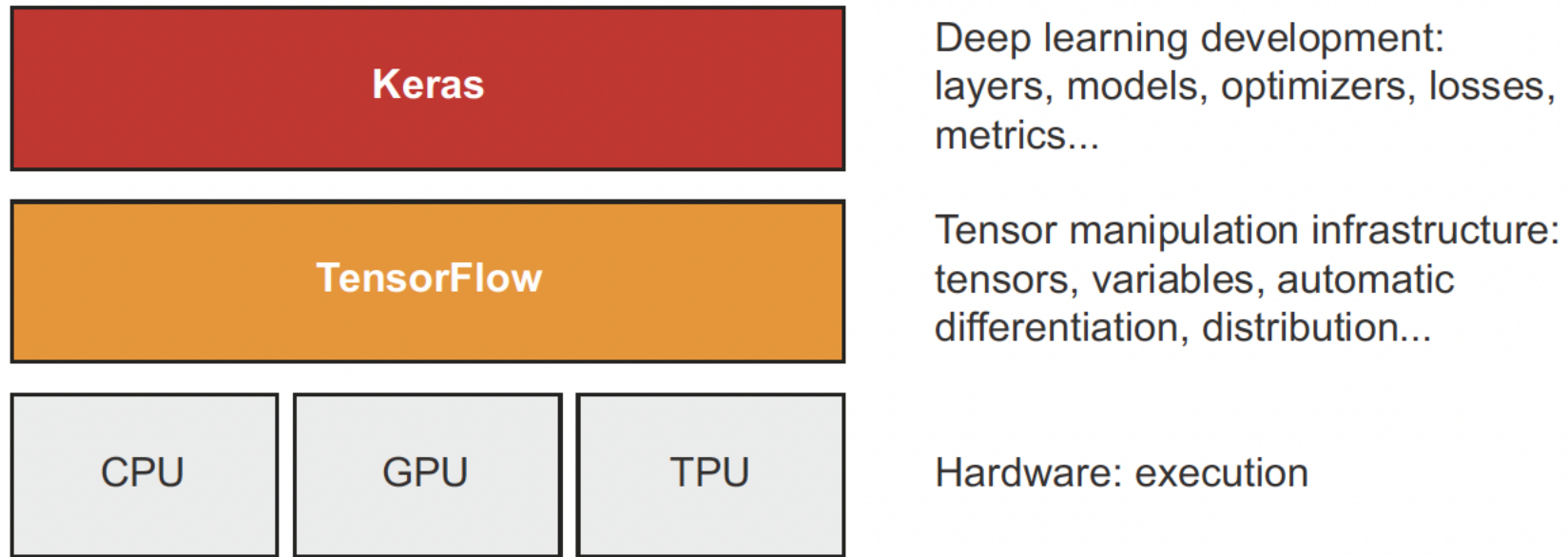# Let's Dive Deep! An Introduction to Deep Learning

Commonly used activation functions

### Sigmoid

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

(a)

### Tanh

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

(b)

### ReLU

$$\text{ReLU}(z) = \begin{cases} z, z > 0 \\ 0, otherwise \end{cases}$$

(c)

### LeakyReLU(a=0.2)

$$\text{LeakyReLU}(z) = \begin{cases} z, z > 0 \\ az, otherwise \end{cases}$$
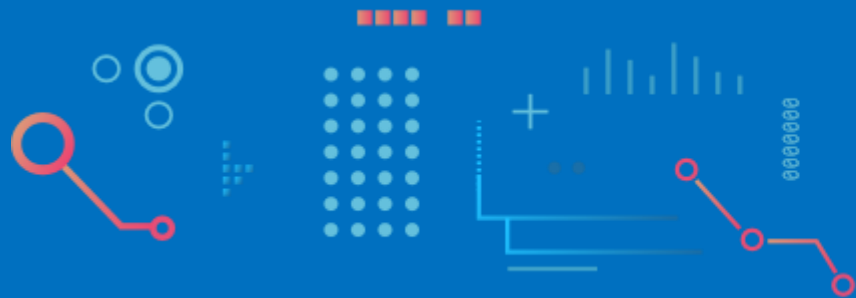
(d)

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!



| | |
|---|---|
| **Keras** | Deep learning development: layers, models, optimizers, losses, metrics... |
| **TensorFlow** | Tensor manipulation infrastructure: tensors, variables, automatic differentiation, distribution... |
| CPU    GPU    TPU | Hardware: execution |

# Hands-On!

Argonne Leadership Computing Facility

# Questions?