Lecture 05
Deep Learning: Basics

February 20, 2025

Data Science Using Python

Jaeung Sim

Assistant Professor

School of Business, University of Connecticut

Agenda

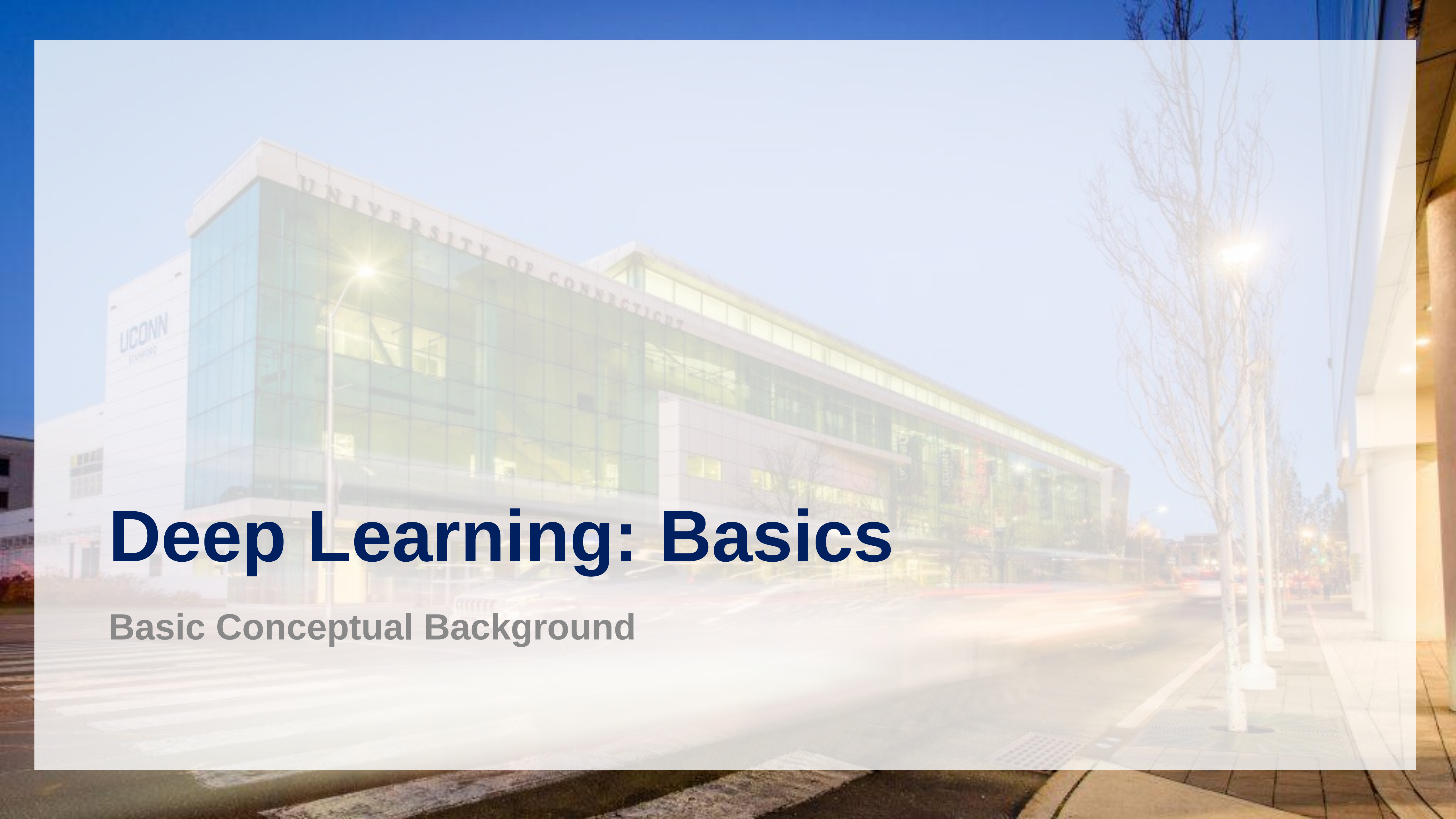
- **Python Hands-on for Concepts in Lecture 04**
- **Q&A for Data Collection Methods**
- **Deep Learning: Basics**
- **Announcements**

Python Hands-on for Concepts in Lecture 04

Regression and Predictive Analysis

Q&A for Data Collection Methods

Beautiful Soup and Selenium

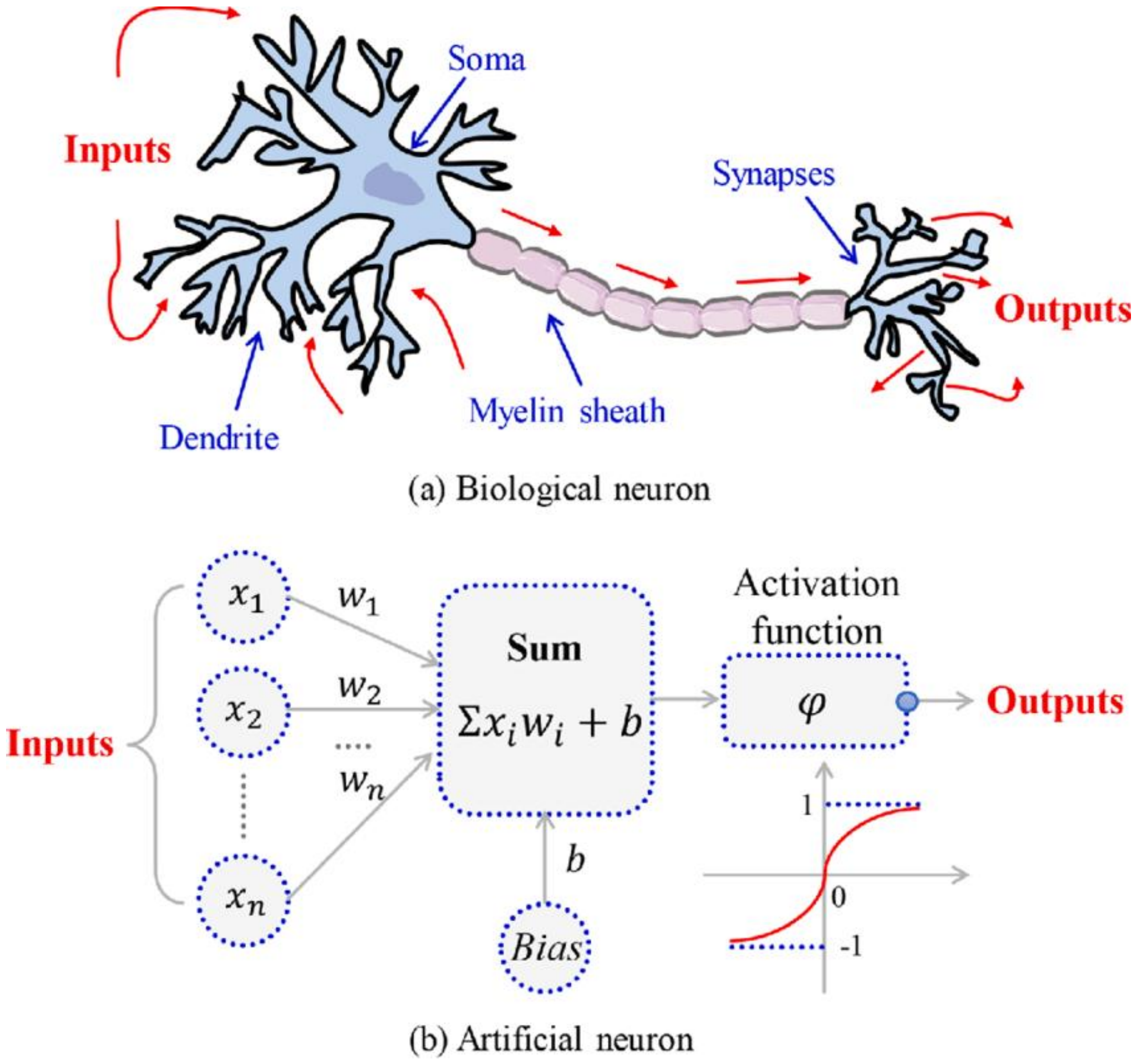


Deep Learning: Basics

Basic Conceptual Background

Neural Network

- What is a **Neural Network**?
  - A neural network is a type of machine learning algorithm that is modeled after the structure and function of the human brain.
  - It is composed of interconnected **nodes** or artificial **neurons** that are organized in layers.



Neural Network

- What is **Deep Learning**?
  - Deep learning uses artificial neural networks with **many layers** (thus, “**deep**”) to model and solve complex problems.
  - It is based on the idea of learning hierarchical representations of data, where each layer of the network learns to represent increasingly abstract features of the input data.
  - Deep learning algorithms require large amounts of data and computational power.

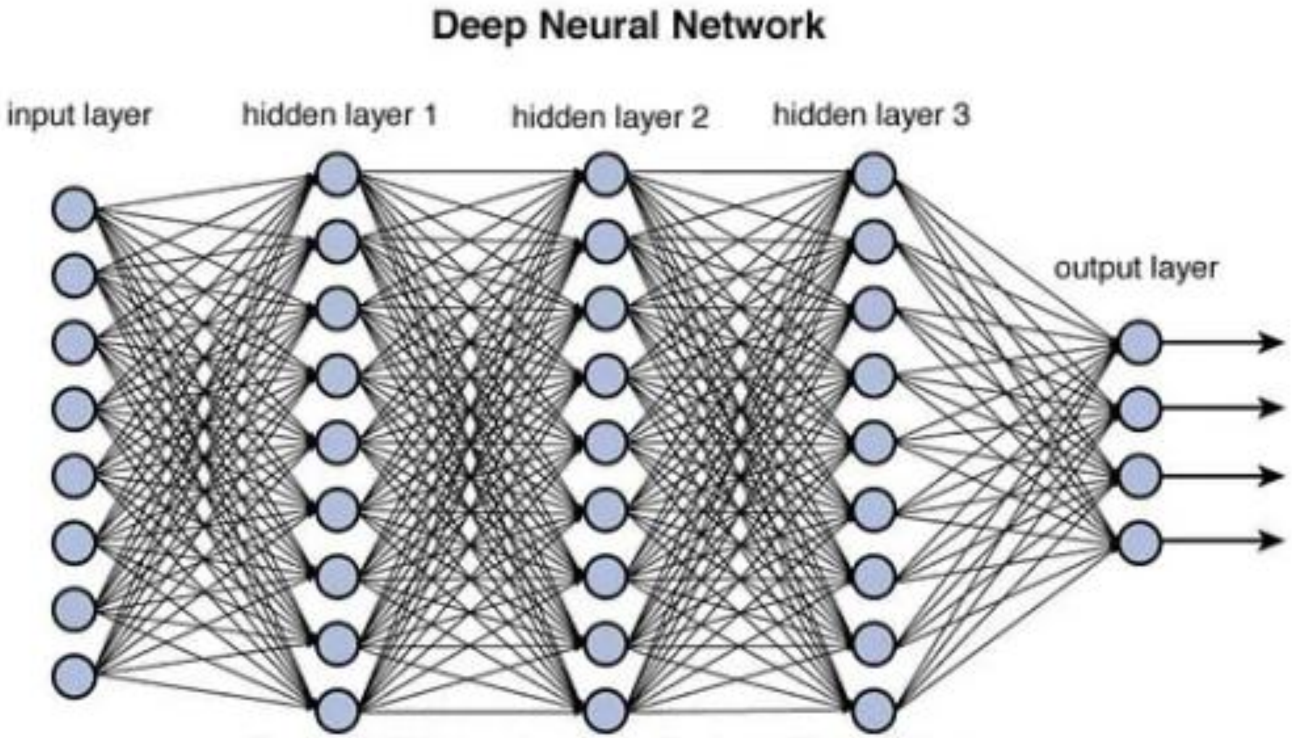
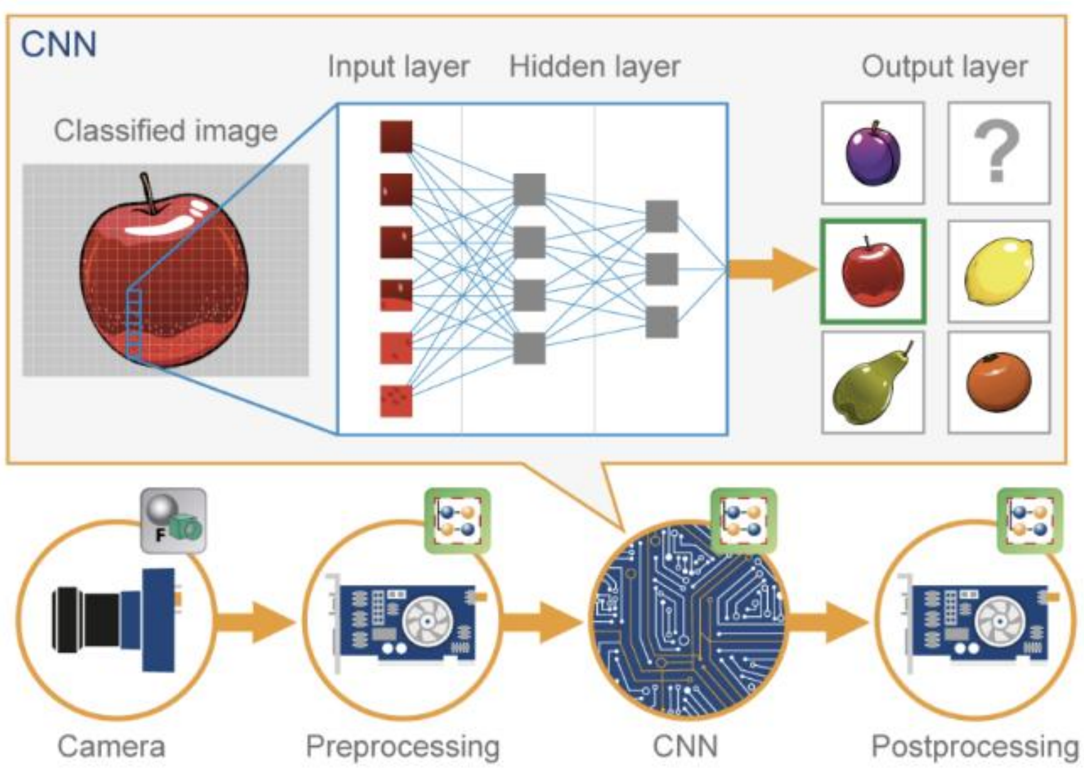
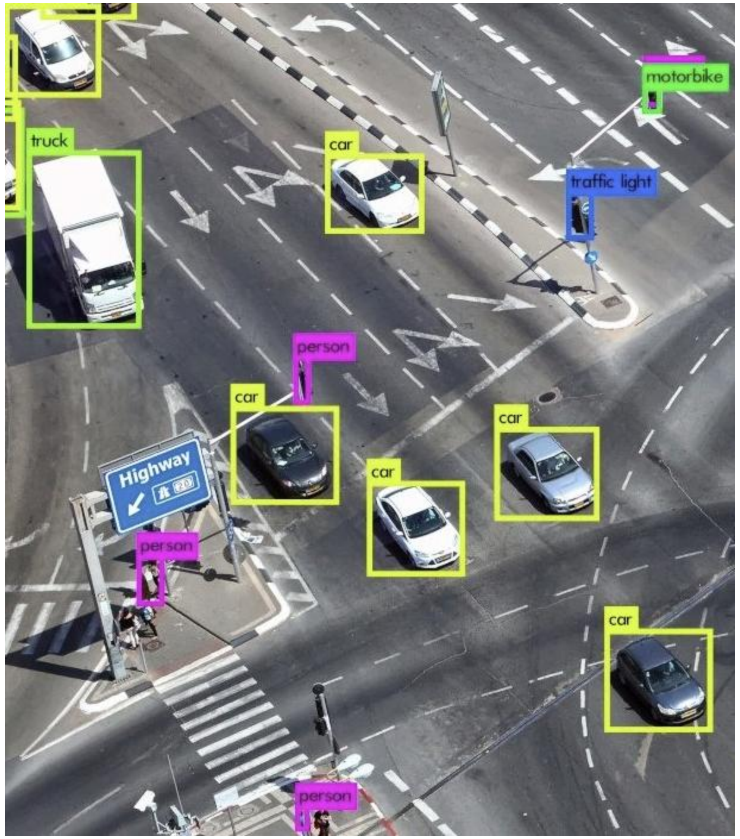


Figure 12.2 Deep network architecture with multiple layers.

Neural Network

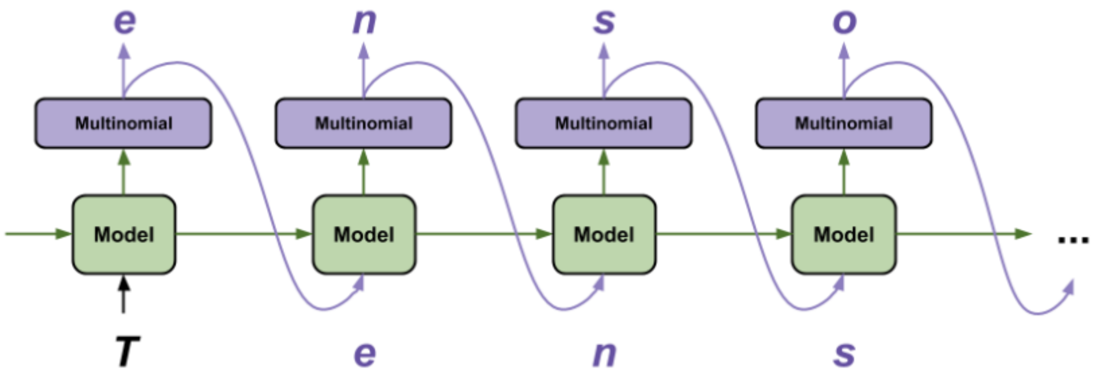
- Examples of Deep Learning
  - **Convolutional Neural Network (CNN)**
    - Image classification, object detection, video action recognition

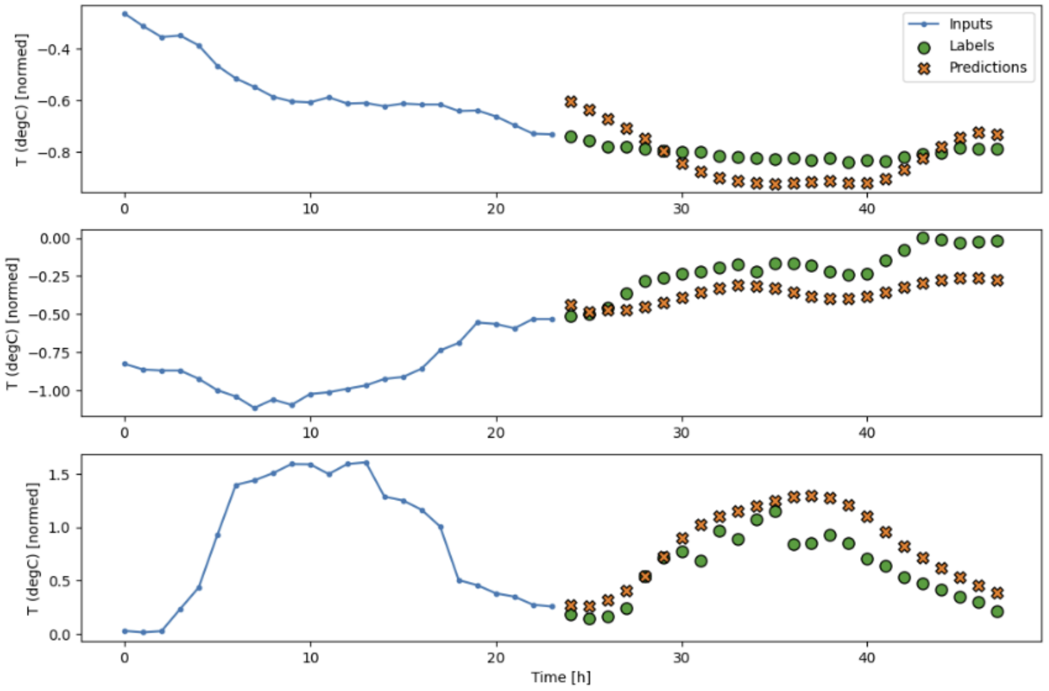




Neural Network

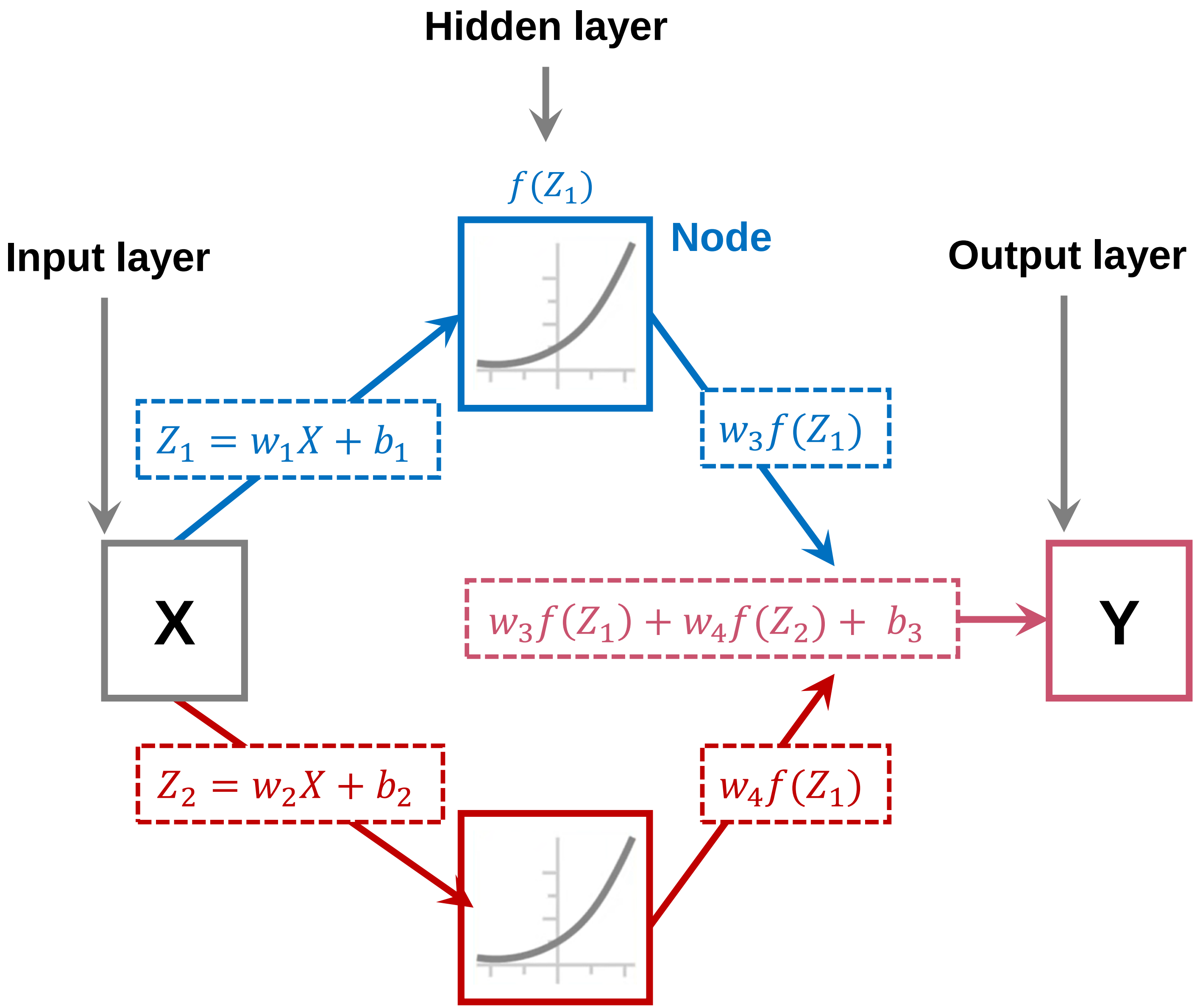
- Examples of Deep Learning
  - **Recurrent Neural Network (RNN)**
    - Text classification and generation
    - Time series forecasting





Neural Network

- **Structure**
  - **Nodes**
  - **Layers**
  - **Inputs (*X*)**
  - **Weights (*w*)**
  - **Bias (*b*)**
  - **Summation Function (*Z*)**
  - **Activation Function (*f(Z)*)**



Neural Network

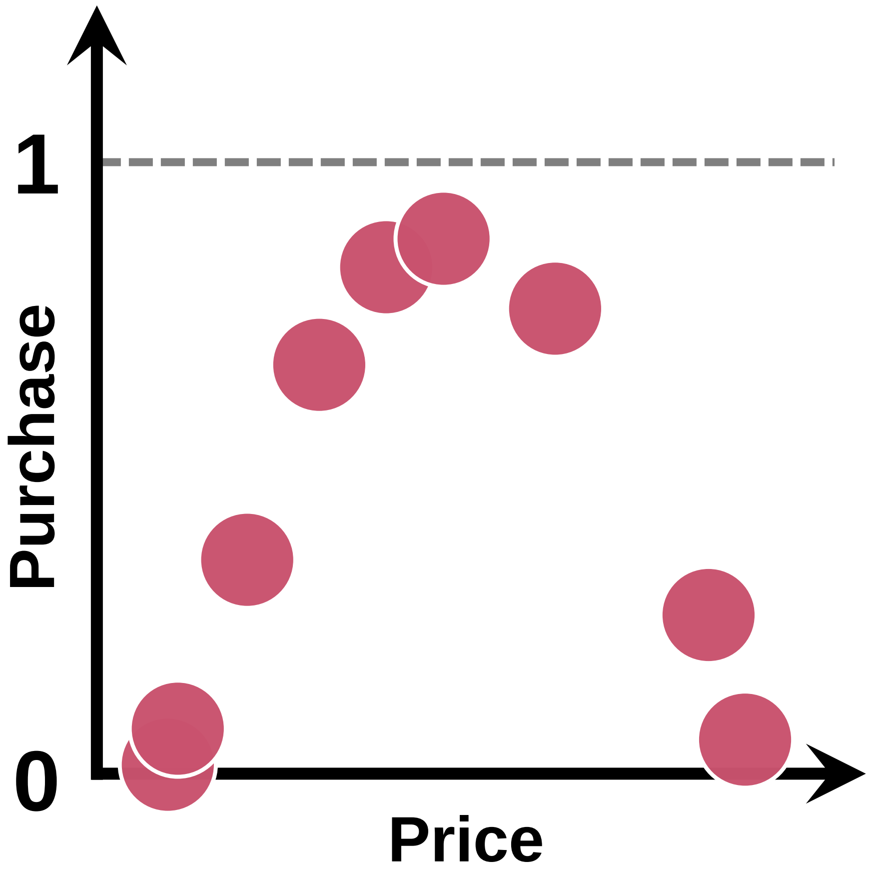
- **Structure**
  - **Inputs:** Inputs are the set of values for which we need to predict an output value. They can be viewed as features or attributes in a dataset.
  - **Weights:** Weights are the real values that are attached with each input/feature and they convey the importance of that corresponding feature in predicting the final output.
  - **Bias:** Bias is used for shifting the activation function towards left or right, you can compare this to y-intercept in the line equation.
  - **Summation Function:** The work of the summation function is to bind the weights and inputs together and calculate their sum.
  - **Activation Function:** It is used to introduce non-linearity in the model.

Neural Network

- **Estimation Steps**
  - Initialize the weights and biases with some random values.
  - Calculate the output which is to predict the values and estimate the loss.
    - Refer to: ***Activation function***
  - Adjust the weights and biases such that the loss will be minimized.
    - Refer to: ***Gradient descent***, ***backpropagation***

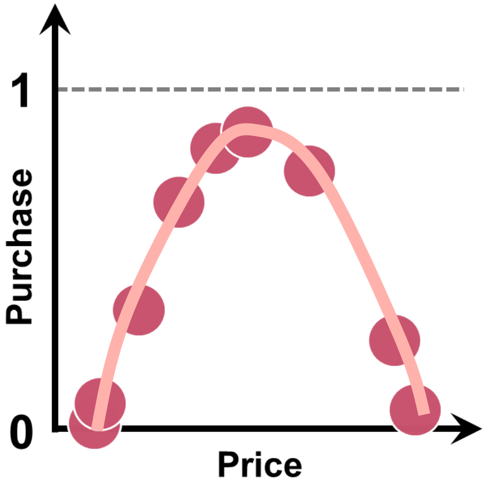
Basic Example

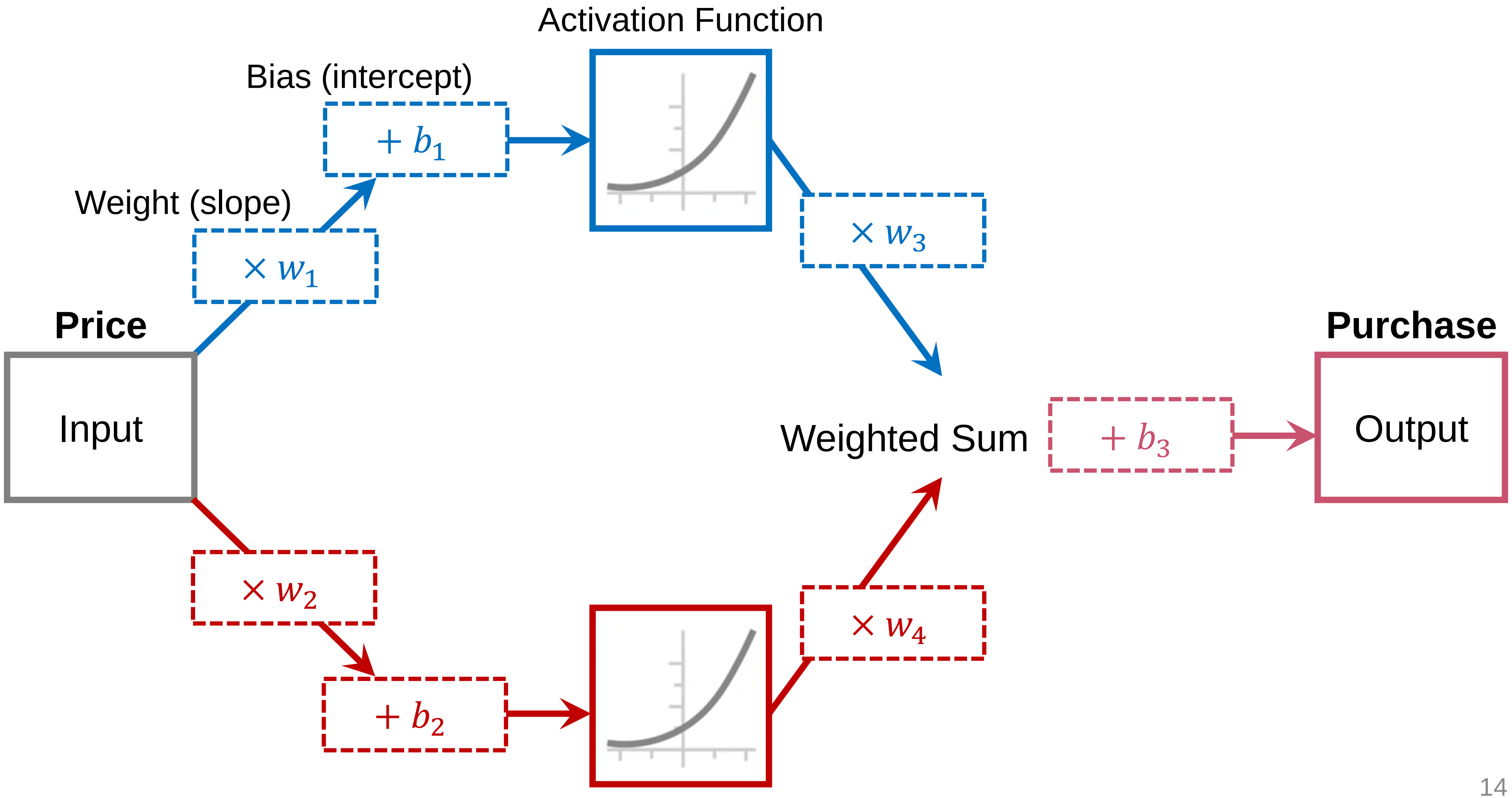
- **Bag Price and Purchase Decision**



Basic Example

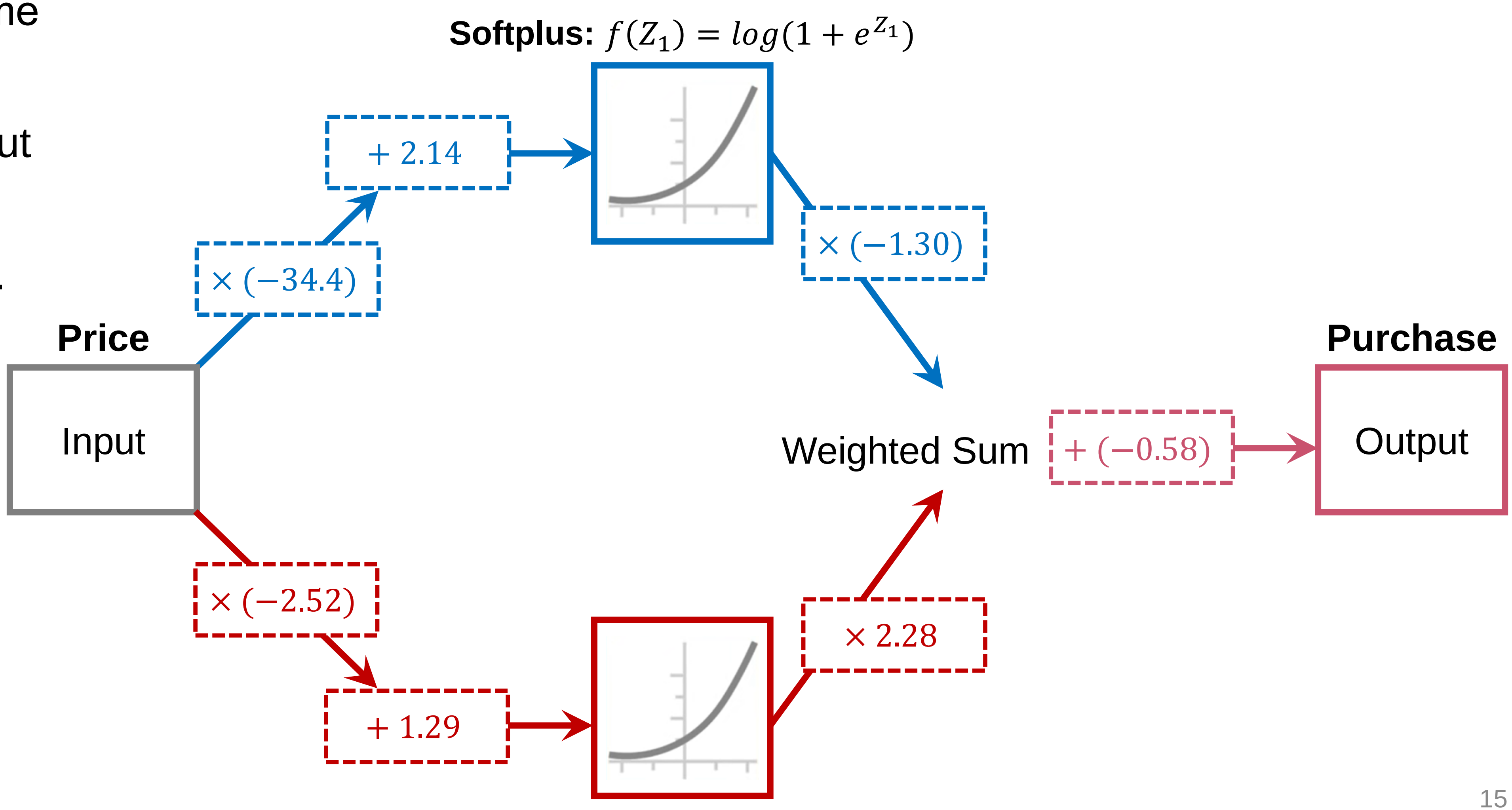
- **Bag Price and Purchase Decision**





Basic Example

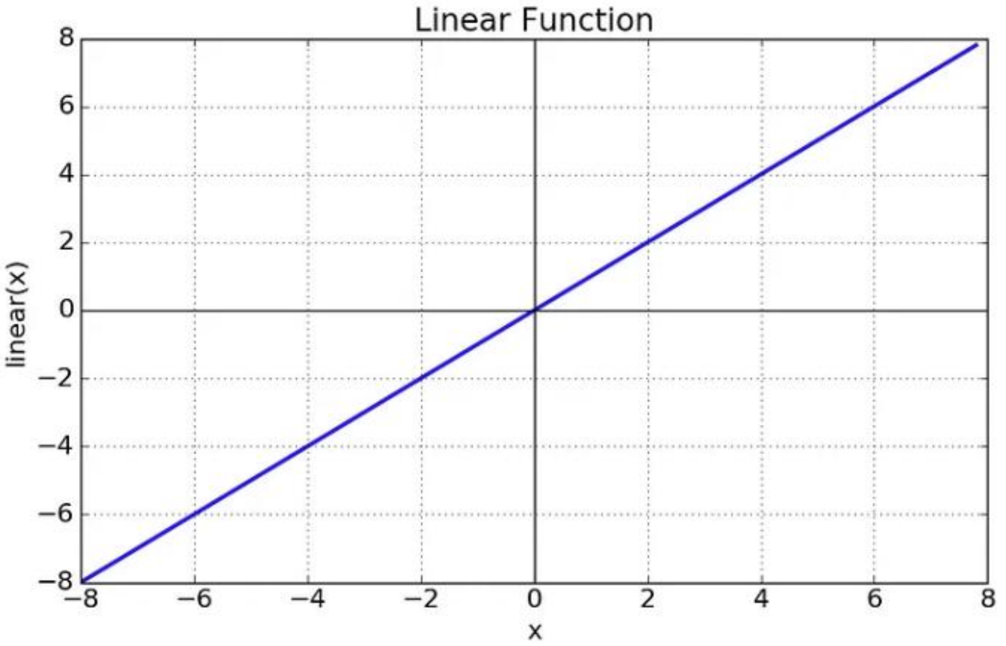
- **Estimation Steps**
  - Let's start with some given parameters.
  - Calculate the output and loss.
  - Adjust weight/loss.



Activation Function

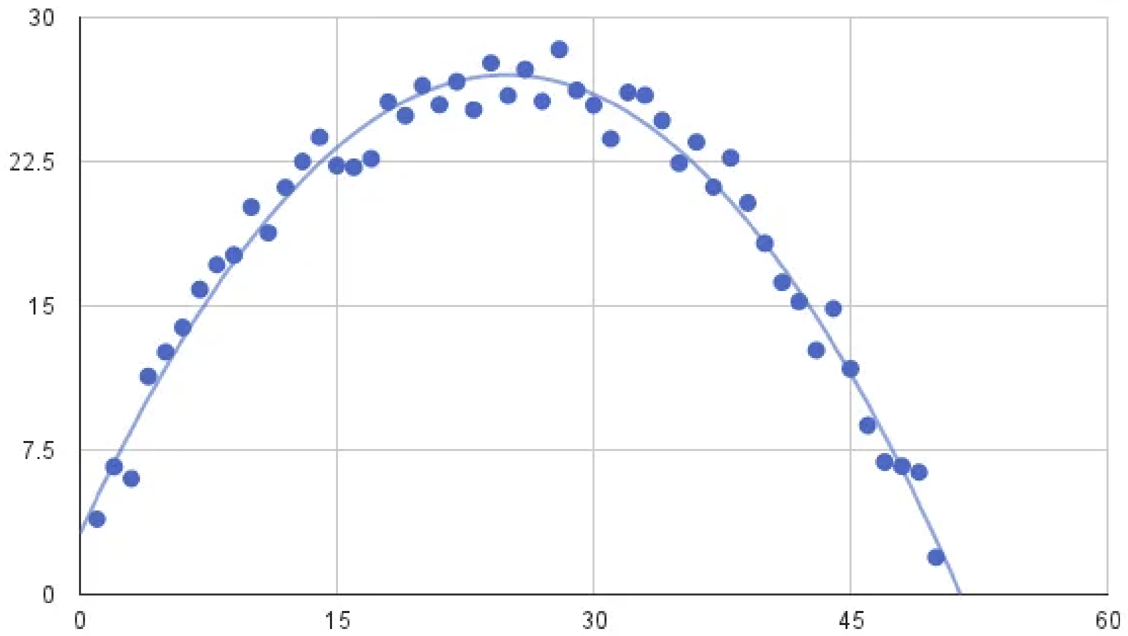
- Also known as a **transfer function**
- Mapping a linear function to another function

Linear Activation Function



Doesn't help with the complexity in neural networks

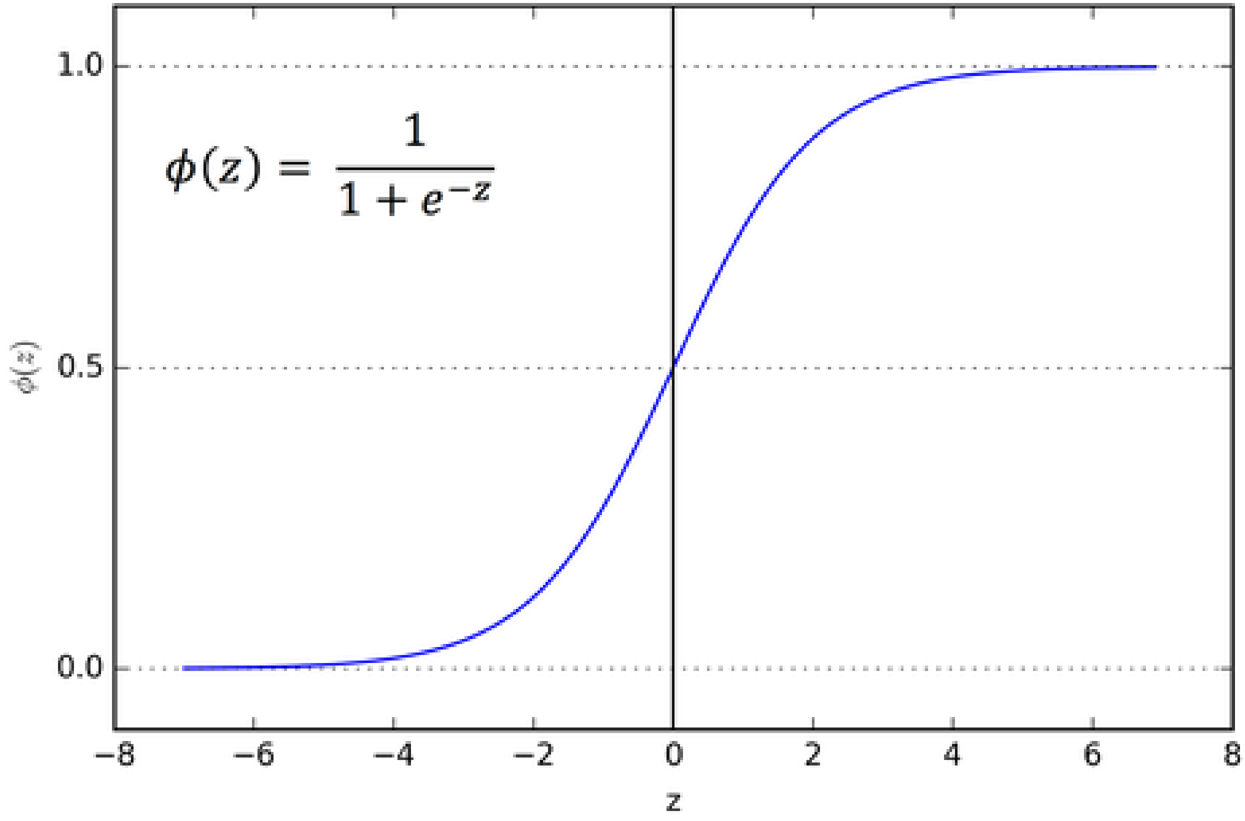
Nonlinear Activation Functions



Makes it easy for the model to adapt with variety of data

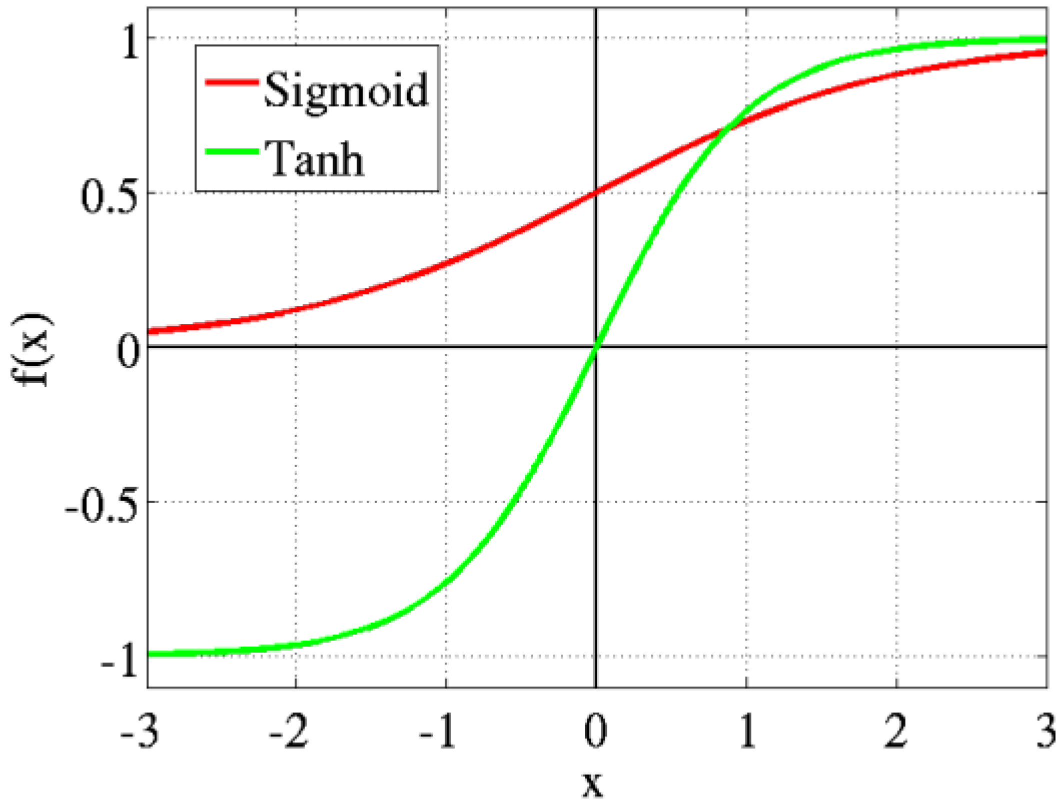
Activation Function

- **Sigmoid function**
  - Exists **between 0 and 1**.
  - Especially used for models to predict the probability as an output.
  - The function is **differentiable**.
  - The function is **monotonic**, but the function's derivative is not.
  - The logistic sigmoid function can cause a neural network to get stuck at the training time.
  - The softmax function is a more generalized logistic activation function which is used for multiclass classification.



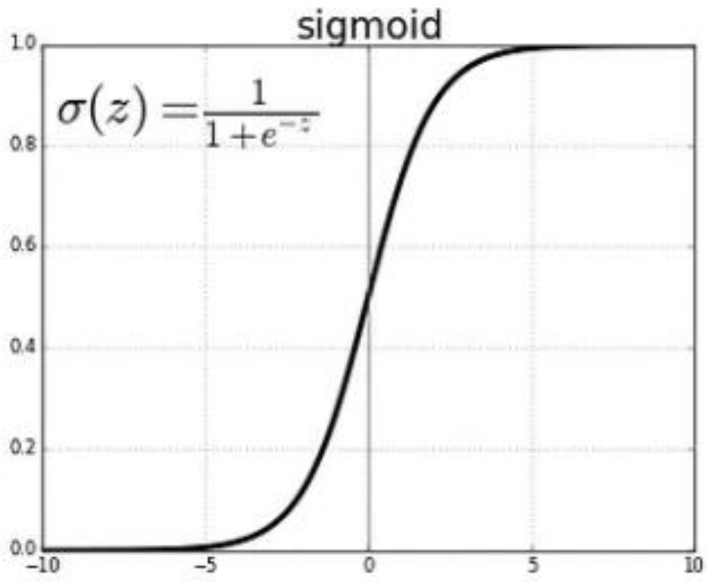
Activation Function

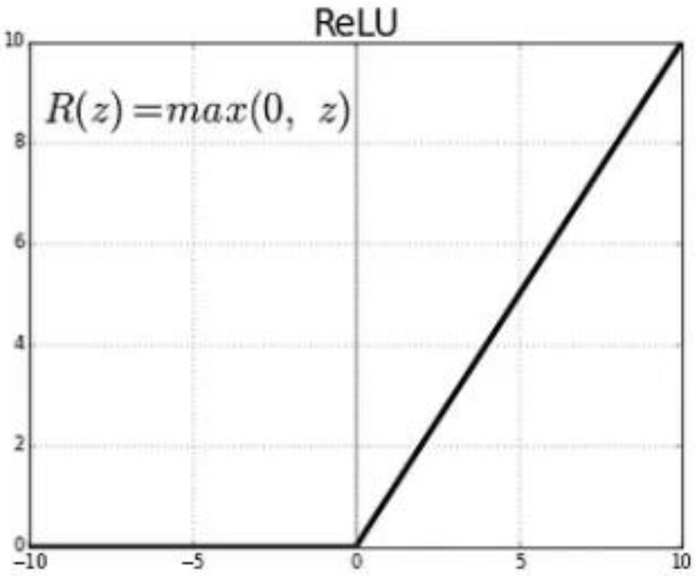
- **Hyperbolic Tangent (tanh)**
  - Exists **between -1 and 1**.
  - The tanh function is mainly used classification between two classes.
  - The negative inputs are mapped strongly negative, and the zero inputs are mapped near zero in the tanh graph.
  - The function is **differentiable**.
  - The function is **monotonic** while its derivative is not monotonic.



Activation Function

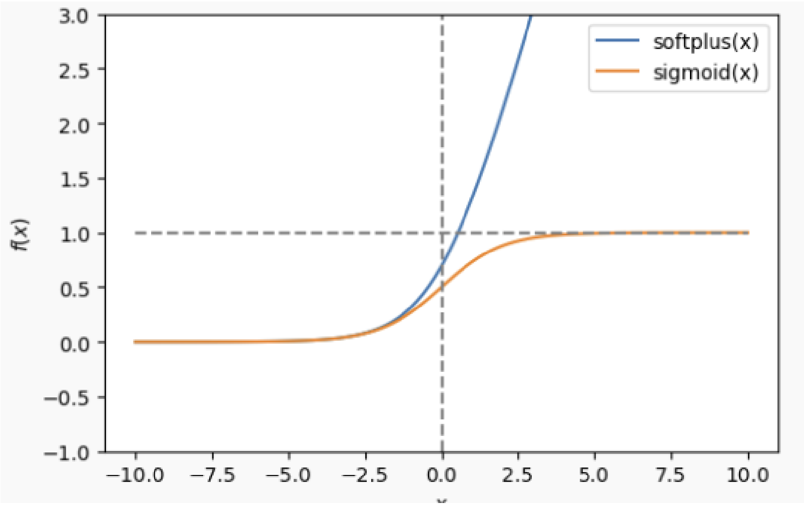
- **Rectified Linear Unit (ReLU)**
  - $R(x) = \max(0, x)$
  - Exists from 0 to infinity
  - **The most used activation function**
  - Half rectified (from bottom)
  - The function and its derivative both are monotonic.
  - All the negative values become zero immediately which decreases the ability of the model to fit or train from the data properly.

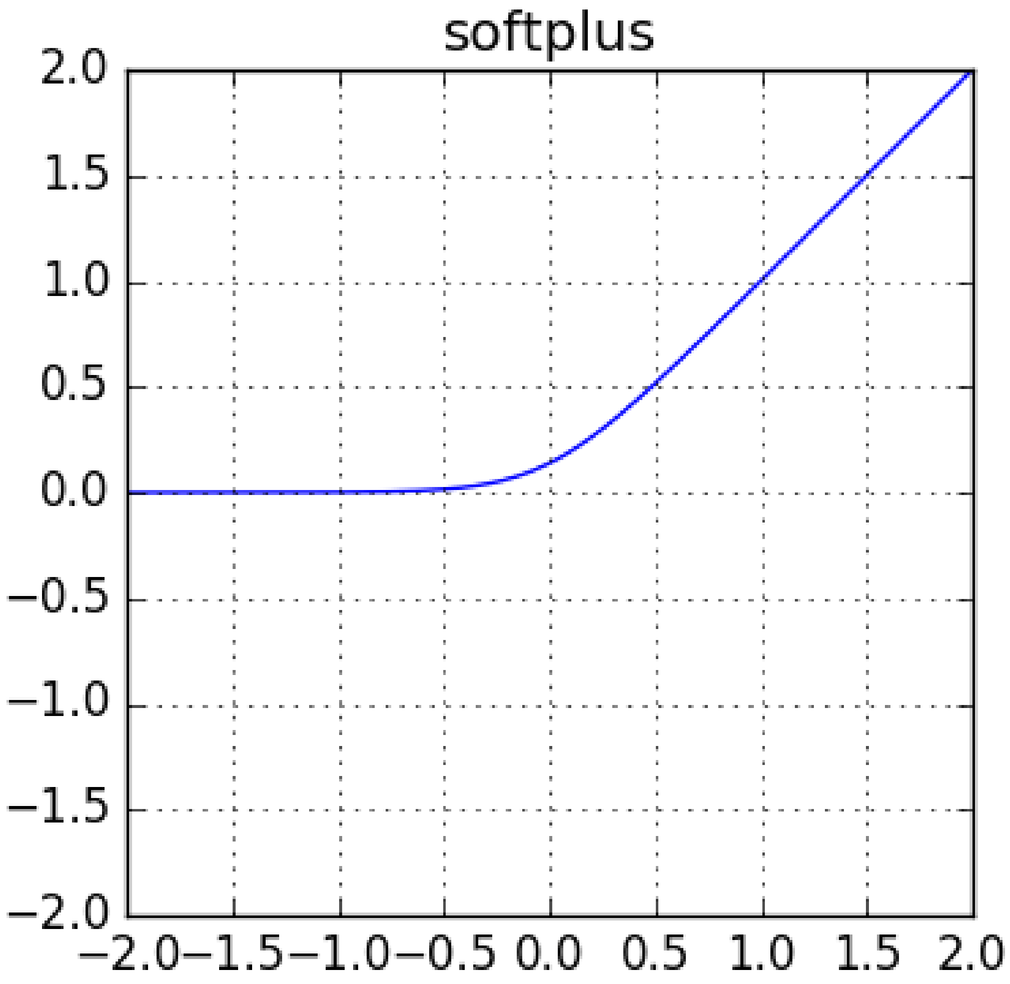




Activation Function

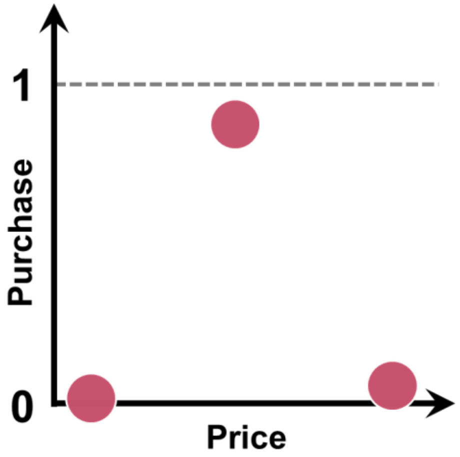
- **Softplus**
  - $f(x) = \log(1 + \exp(x))$
  - A smooth approximation to the ReLU activation function.
  - As $x \to -\infty$, it becomes identical to $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$

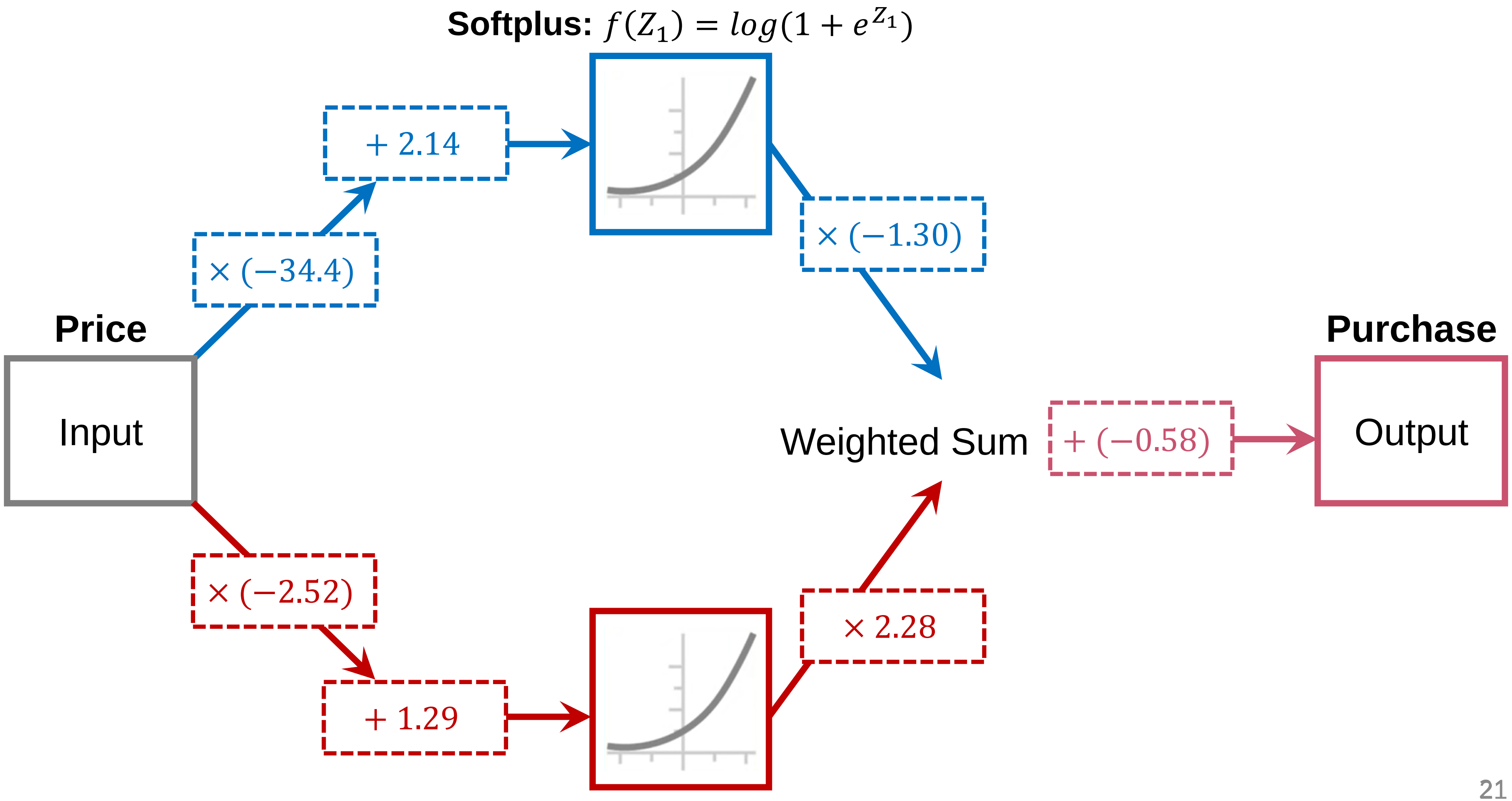




Activation Function

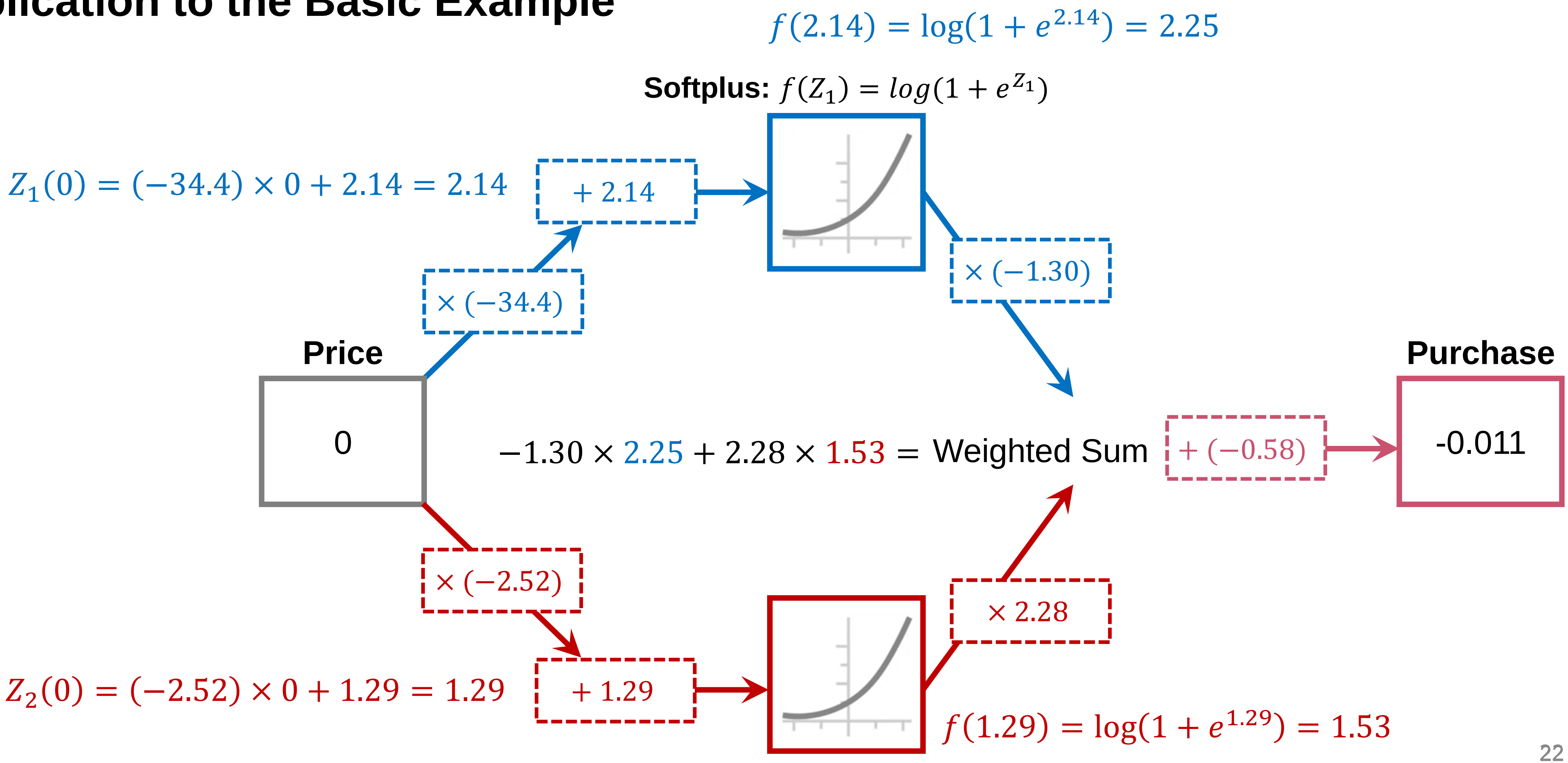
- **Application to the Basic Example**





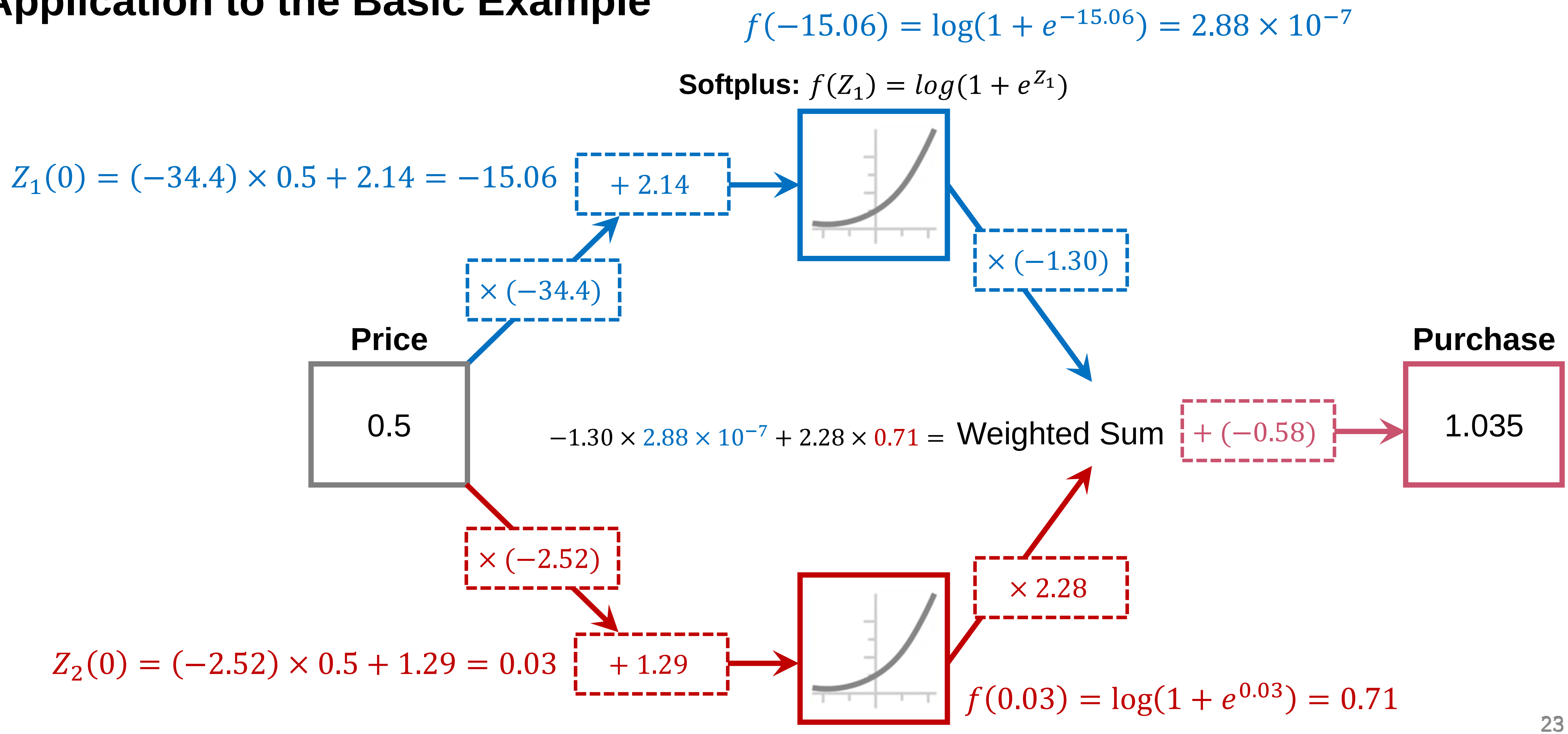
Activation Function

- **Application to the Basic Example**



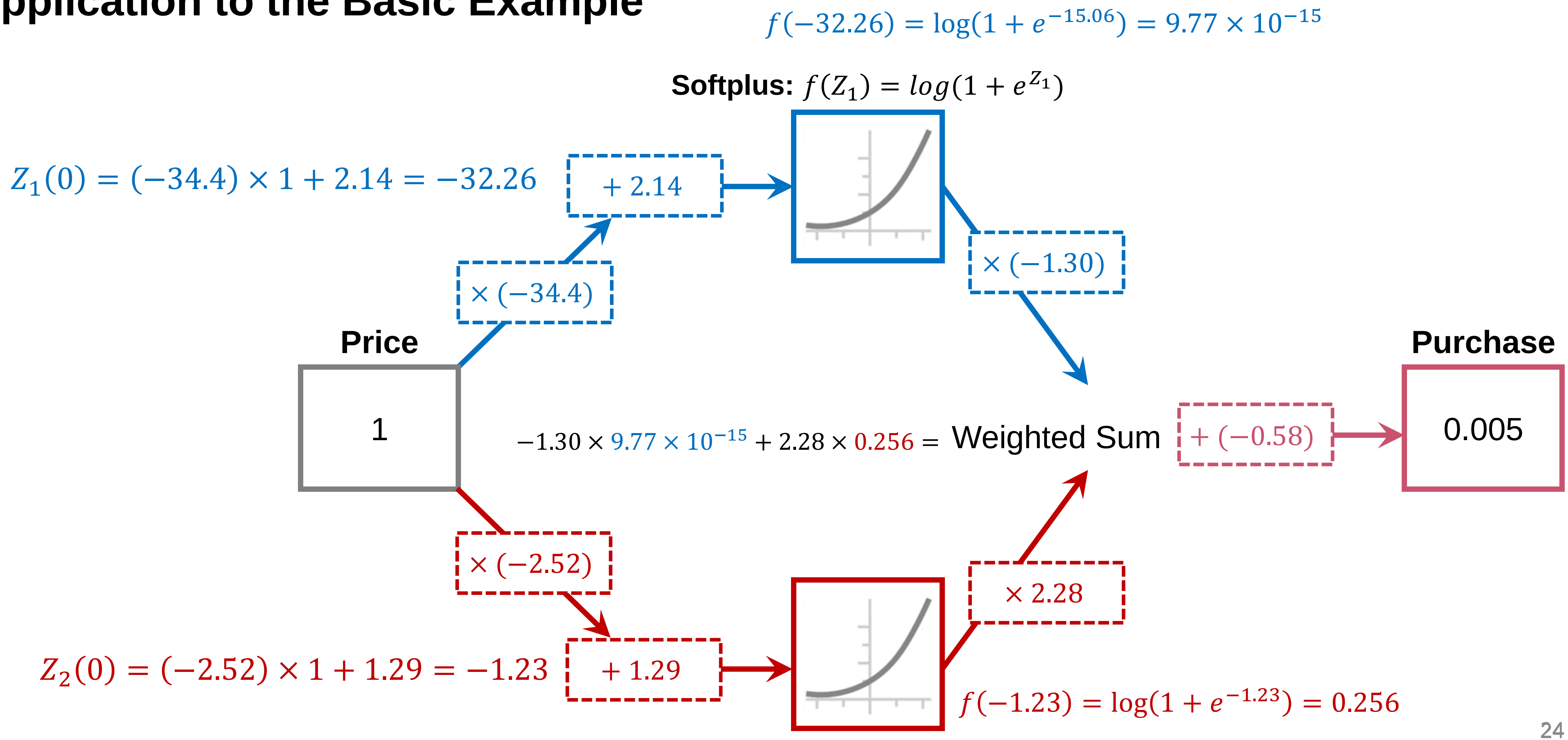
Activation Function

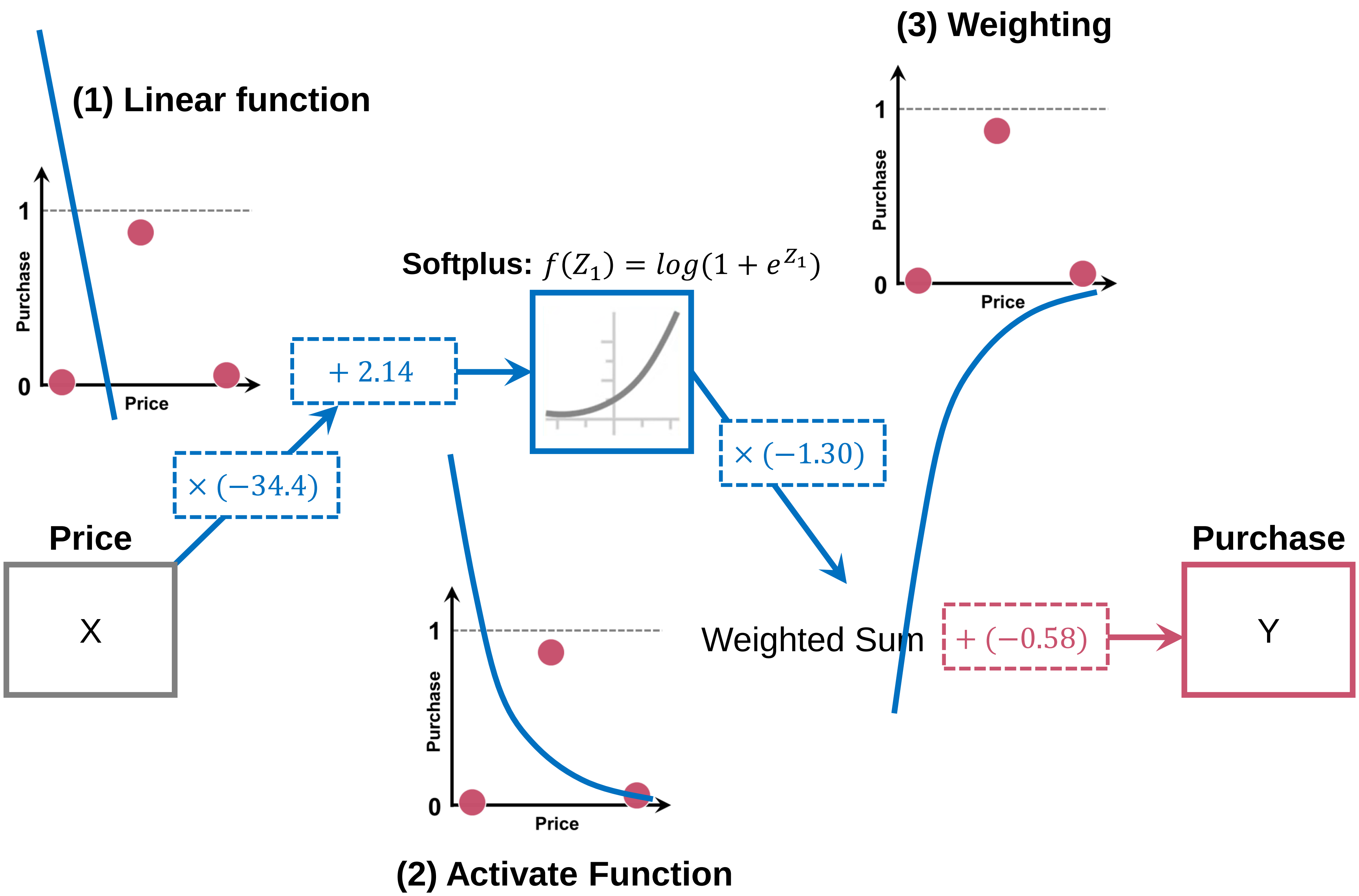
- **Application to the Basic Example**

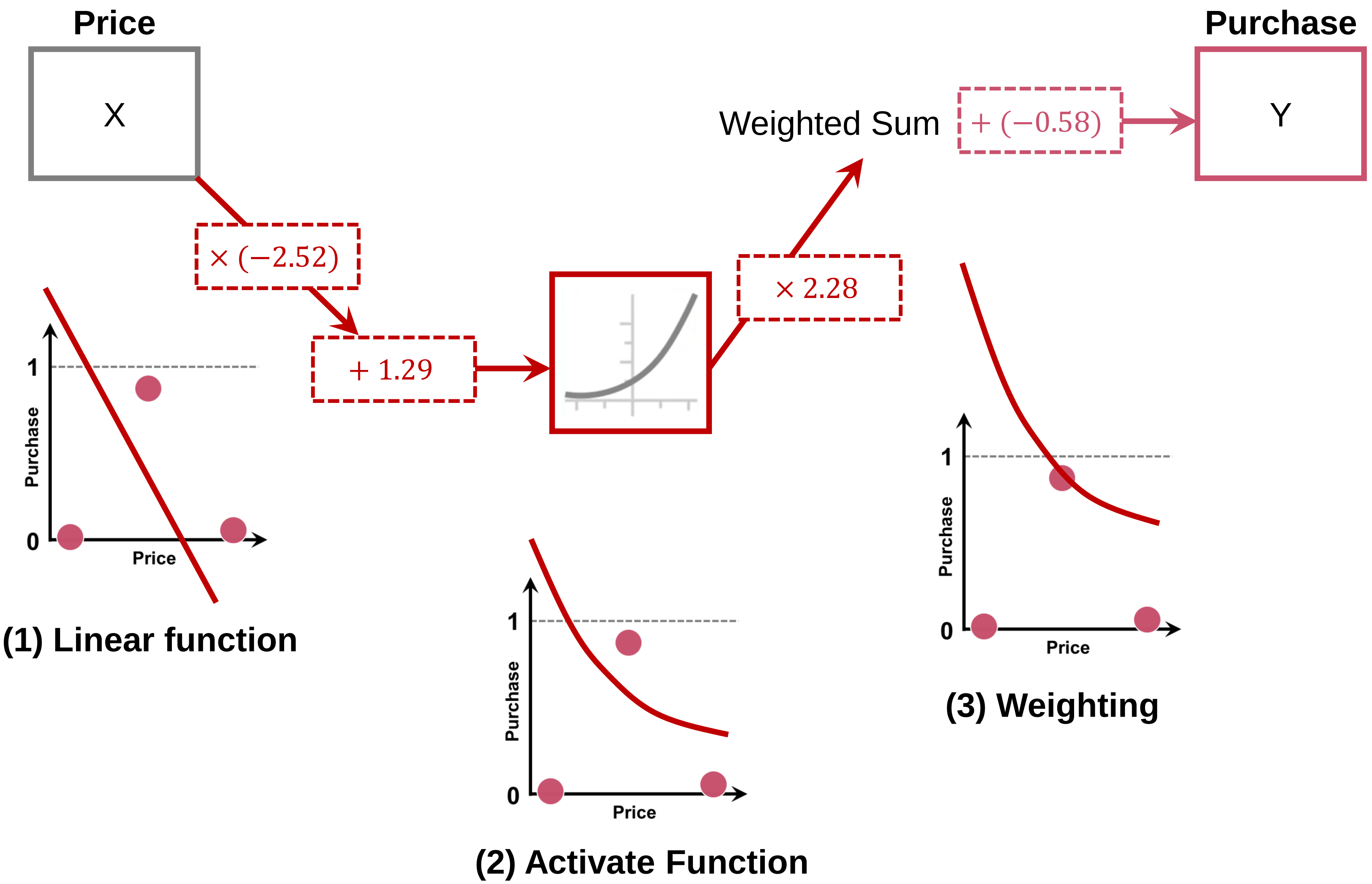


Activation Function

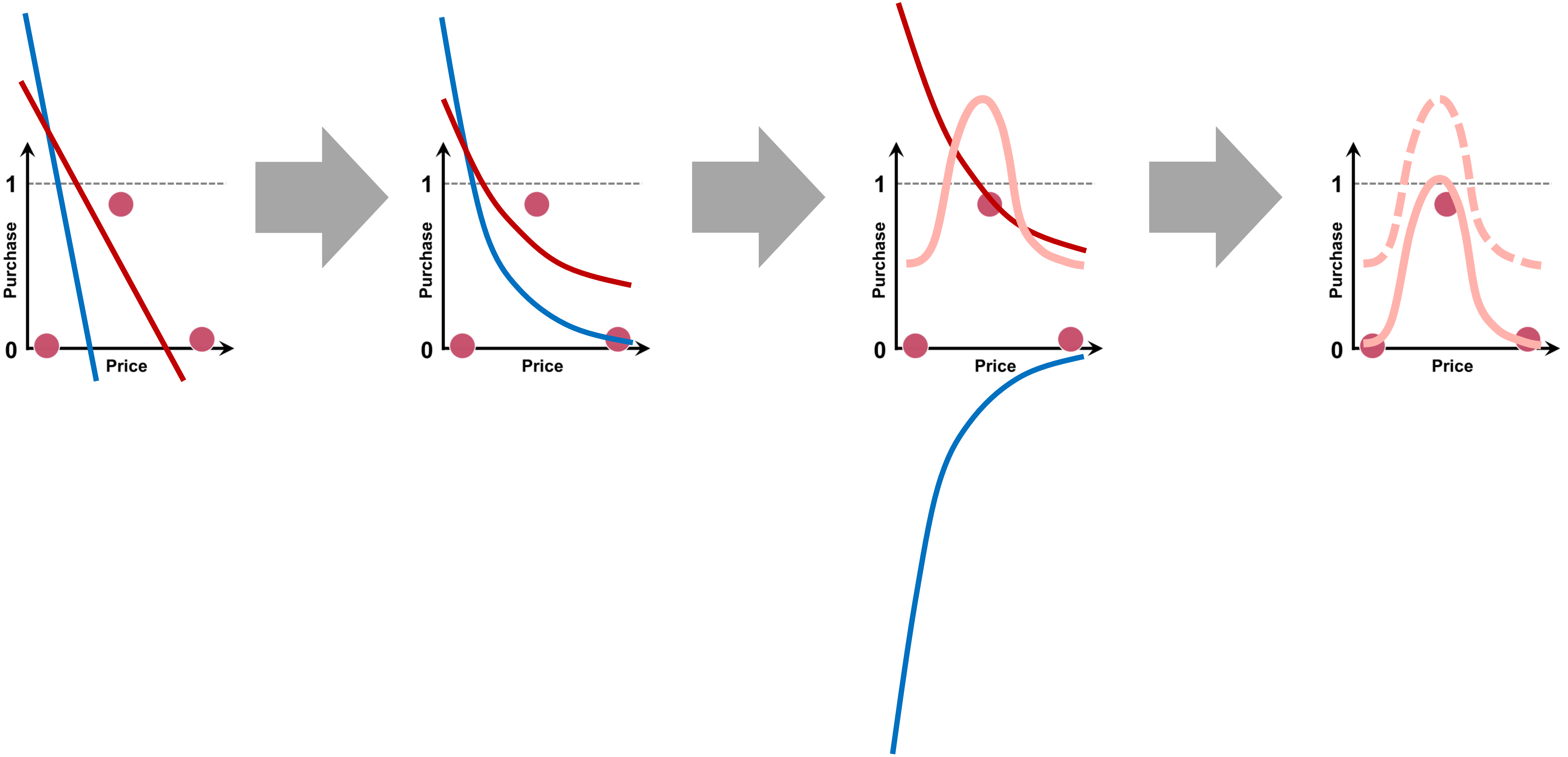
- **Application to the Basic Example**




(1) Linear function
1
0
Purchase
Price
(3) Weighting
1
0
Purchase
Price
Softplus: $f(Z_1) = log(1 + e^{Z_1})$
$+ 2.14$
$\times (-34.4)$
$\times (-1.30)$
Price
X
Purchase
Y
Weighted Sum
$+ (-0.58)$
1
0
Purchase
Price
(2) Activate Function


Price
X
Purchase
Y
Weighted Sum
$+\ (-0.58)$
$\times\ (-2.52)$
$+\ 1.29$
$\times\ 2.28$
1
0
Purchase
Price
(1) Linear function
(2) Activate Function
(3) Weighting

(1) Linear function   (2) Activate Function   (3) Weighting   (4) Adding Bias



Chain Rule

- After calculating the output which is to predict the values and estimate the loss, we should adjust the weights and biases to minimize the loss.
- In doing so, we apply **backpropagation** based on **gradient descent**.
- To understand gradient descent, you need the **chain rule**:

Lagrange's notation

For function $h(x) = f(g(x))$,

$$h'(x) = f'\big(g(x)\big) \cdot g'(x)$$

Leibniz's notation

If a variable z depends on the variable y, which itself depends on the variable x,

$$\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$$

Chain Rule

- **Example: Chicken Price and Fried Chicken Sales**

Chain Rule

- **Example: Chicken Price and Fried Chicken Sales**

We can rewrite the relationship as follows:

$$\textit{Fried Chicken Sales} = f(\textit{Fried Chicken Price})$$

$$\textit{Fried Chicken Price} = g(\textit{Chicken Price})$$

$$\textit{Fried Chicken Sales} = f(g(\textit{Chicken Price}))$$

Chain Rule

- **Example: Chicken Price and Fried Chicken Sales**

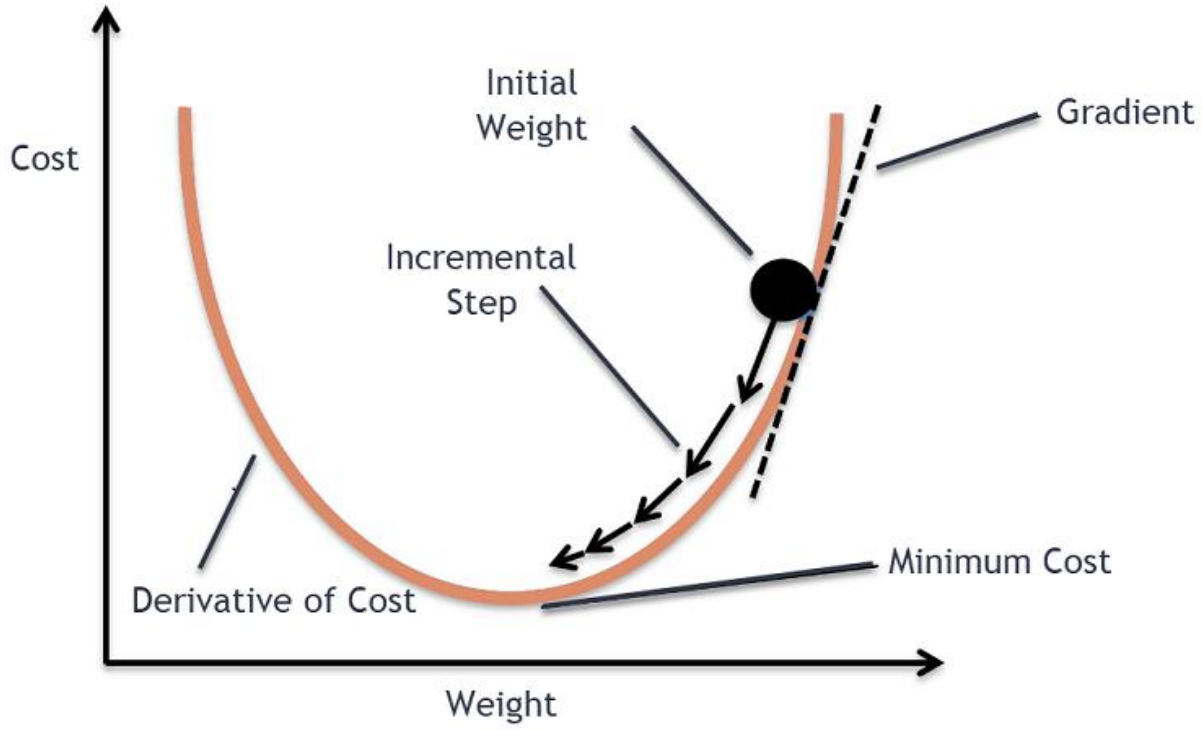
$$\frac{d\ \textit{Fried Chicken Sales}}{d\ \textit{Chicken Price}} = \underbrace{\frac{d\ \textit{Fried Chicken Sales}}{d\ \textit{Fried Chicken Price}}}_{\text{Demand-side response}} \cdot \underbrace{\frac{d\ \textit{Fried Chicken Price}}{d\ \textit{Chicken Price}}}_{\text{Supply-side response}}$$

"Are consumers price-sensitive?"

"Are sellers margin-sensitive?"

Gradient Descent

- A first-order iterative optimization for finding a local minimum of a differentiable function



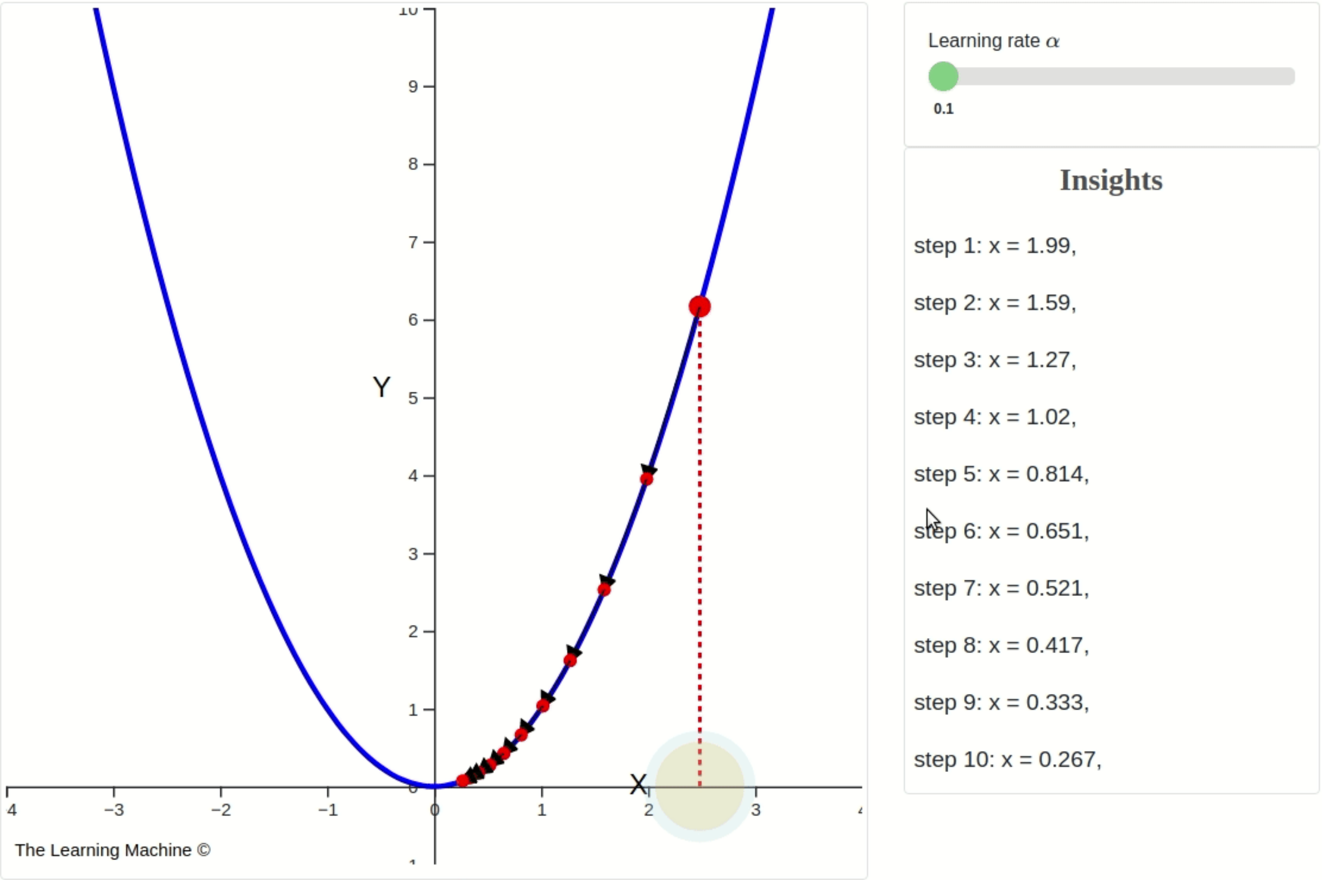
- **Step 1:** Take the derivative of the **loss function** for each parameter in it.
- **Step 2:** Pick random values for the parameters.
- **Step 3:** Plug the parameter values into the derivatives (i.e., the **gradient**).
- **Step 4:** Calculate the step sizes as
  - **Step Size** = **Slope** x **Learning Rate**
- **Step 5:** Calculate the new parameters as
  - **New** = **Old** – **Step Size**
  - Going back to **Step 3**

Gradient Descent

- **Loss Function**
  - **Loss** is the penalty for a bad prediction, or a number indicating how bad the model's prediction was on a single example.
  - **Regression losses:**
    - Mean squared error (MSE)/Quadratic loss/L2 loss
      - $MSE = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}$
    - Mean absolute error (MAE)/L1 loss
      - $MAE = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}$
  - **Classification losses:**
    - Cross entropy loss/negative log likelihood
      - $CrossEntropyLoss = -(y_i \log(\hat{y}_i) + (1 - y_i)\log(1 - \hat{y}_i))$

Gradient Descent

- **Learning Rates**
  - The **learning rate** (α) is a **hyperparameter** used to govern the pace at which an algorithm updates or learns the values of a parameter estimate.
  - A desirable learning rate is **low enough** for the network to converge on something useful while yet being **high enough** to train in a reasonable length of time.
  - Smaller learning rates necessitate more training epochs because of the fewer changes.
  - Larger learning rates result in faster changes and a suboptimal final set of weights frequently.

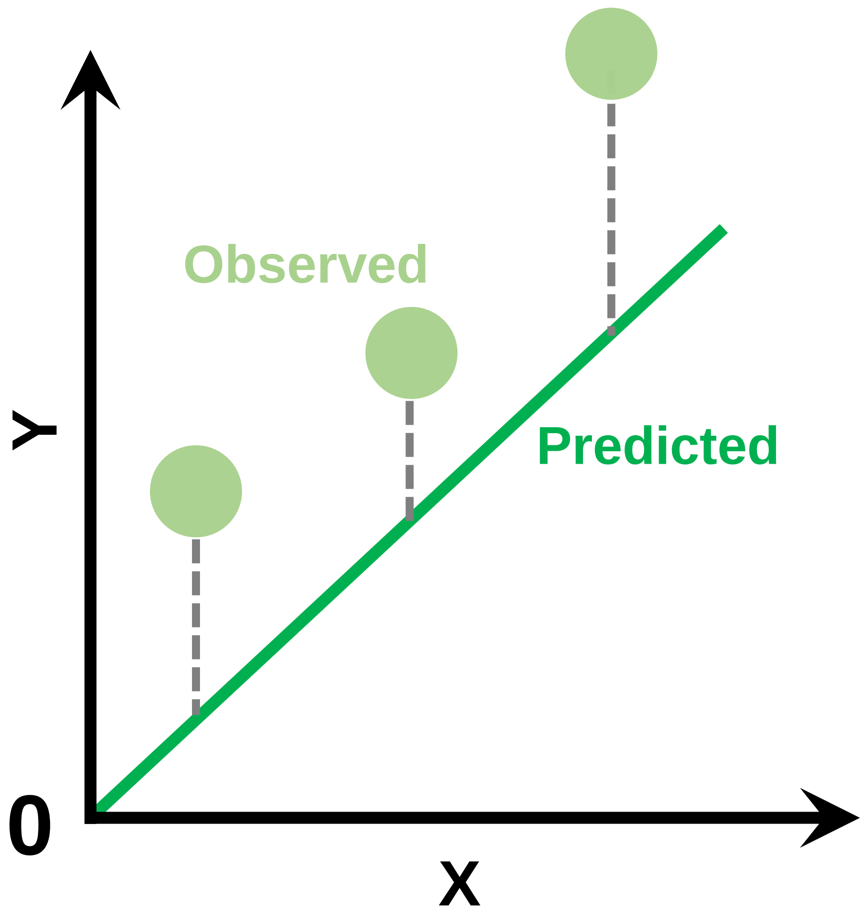


$\min x^2$

Drag the orange circle to change starting point

Gradient Descent

- **A Simple Example**



Sum of squared residuals (SSR)

$$= \Sigma\,(\text{Observed} - (\text{Predicted}))^2$$

$$\begin{aligned}
&= (Y_1 - (\text{intercept} + \text{slope} \times X_1))^2 \\
&+ (Y_2 - (\text{intercept} + \text{slope} \times X_2))^2 \\
&+ (Y_3 - (\text{intercept} + \text{slope} \times X_3))^2
\end{aligned}$$

$$\begin{aligned}
&= (1.4 - (\text{intercept} + 0.64 \times 0.5))^2 \\
&+ (1.9 - (\text{intercept} + 0.64 \times 2.3))^2 \\
&+ (3.2 - (\text{intercept} + 0.64 \times 2.9))^2
\end{aligned}$$

Gradient Descent

- **A Simple Example**

Chain Rule: $\frac{dz}{dx} = \frac{dz}{dy} \cdot \frac{dy}{dx}$

$$\frac{\partial\, SSR}{\partial\, intercept} = \frac{\partial}{\partial\, intercept}(1.4 - (\text{intercept} + 0.64 \times 0.5))^2 + \frac{\partial}{\partial\, intercept}(1.9 - (\text{intercept} + 0.64 \times 2.3))^2 + \frac{\partial}{\partial\, intercept}(3.2 - (\text{intercept} + 0.64 \times 2.9))^2$$

$$= 2 \times (1.4 - (\text{intercept} + 0.64 \times 0.5)) \times (-1) + 2 \times (1.9 - (\text{intercept} + 0.64 \times 2.3)) \times (-1) + 2 \times (3.2 - (\text{intercept} + 0.64 \times 2.9)) \times (-1)$$

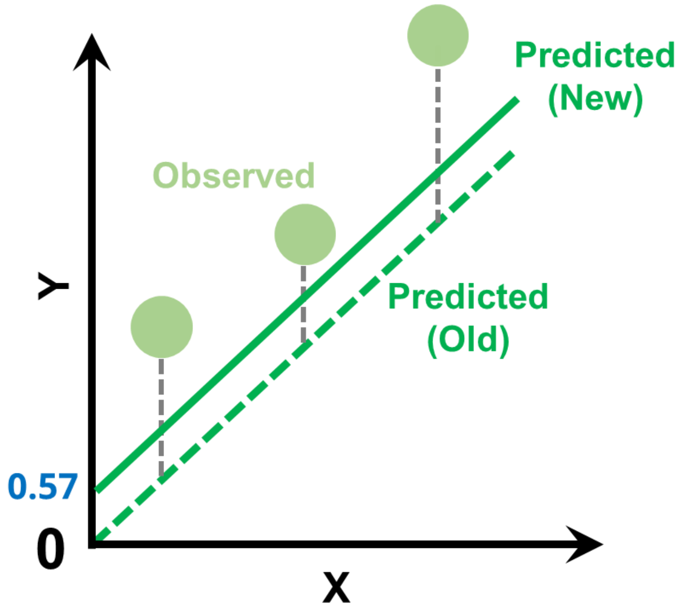
$$= \mathbf{-5.7}$$

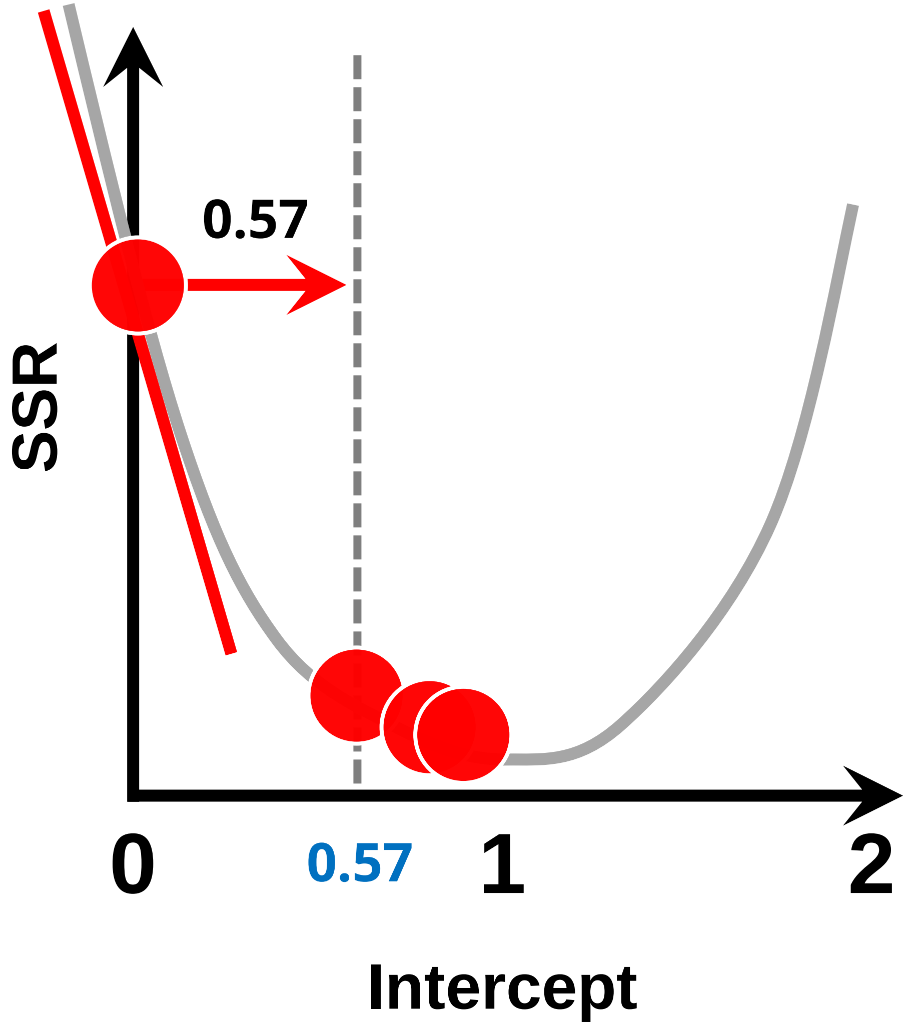
Gradient Descent

- **A Simple Example**
  - **Step Size** = **Slope** x **Learning Rate**

    = **– 5.7** x **0.1**

    = **– 0.57**
  - **New Intercept = 0 – (– 0.57) = 0.57**



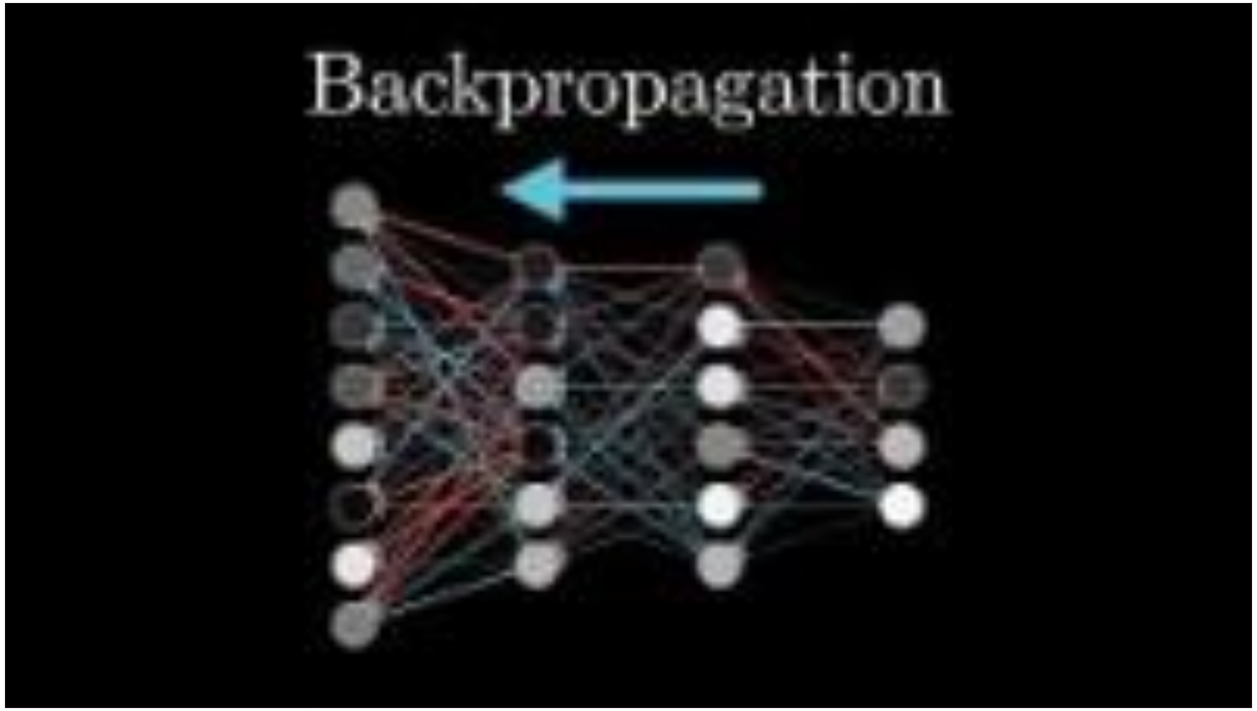


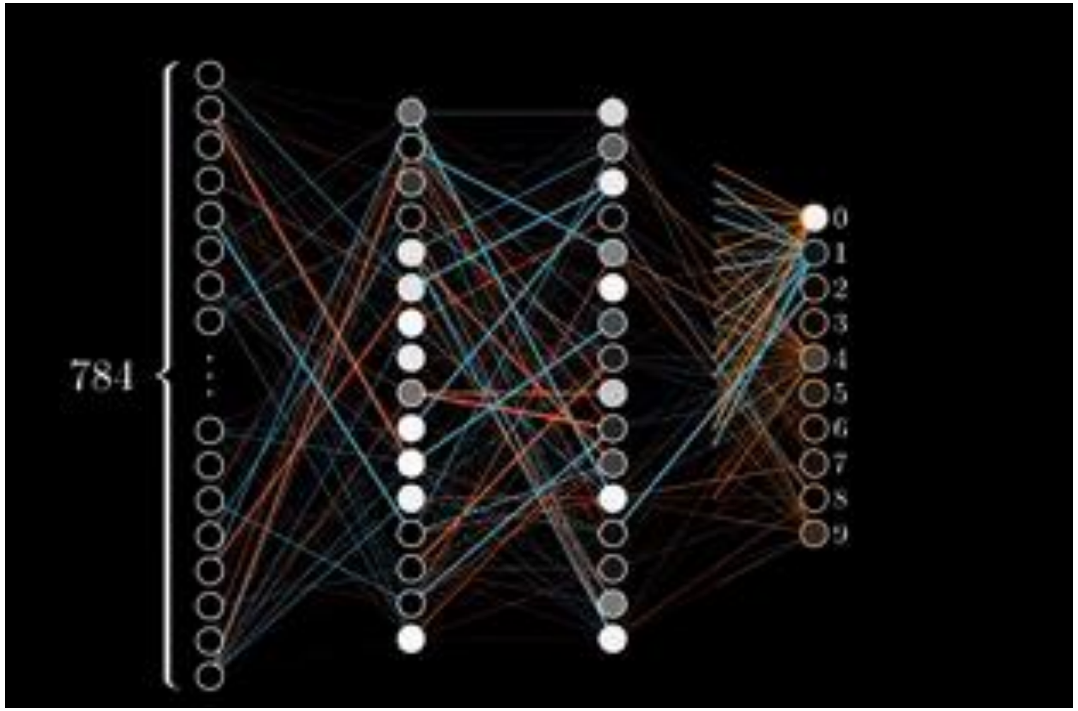
Backpropagation

- Backpropagation works by **propagating the error backwards through the network**, beginning at the output layer and progressing backwards towards the input layer.
- The **gradient of the loss function** with respect to the weights is calculated using the **chain rule**, which allows the error at the output layer to be backpropagated to the hidden layers.
- The weights are then updated using the **gradient descent** algorithm, which adjusts the weights in the direction that minimizes the loss function.
- This process is repeated over many iterations until the network converges to a set of weights that minimize the loss function.

Backpropagation

- **Awesome Visualization of Backpropagation**





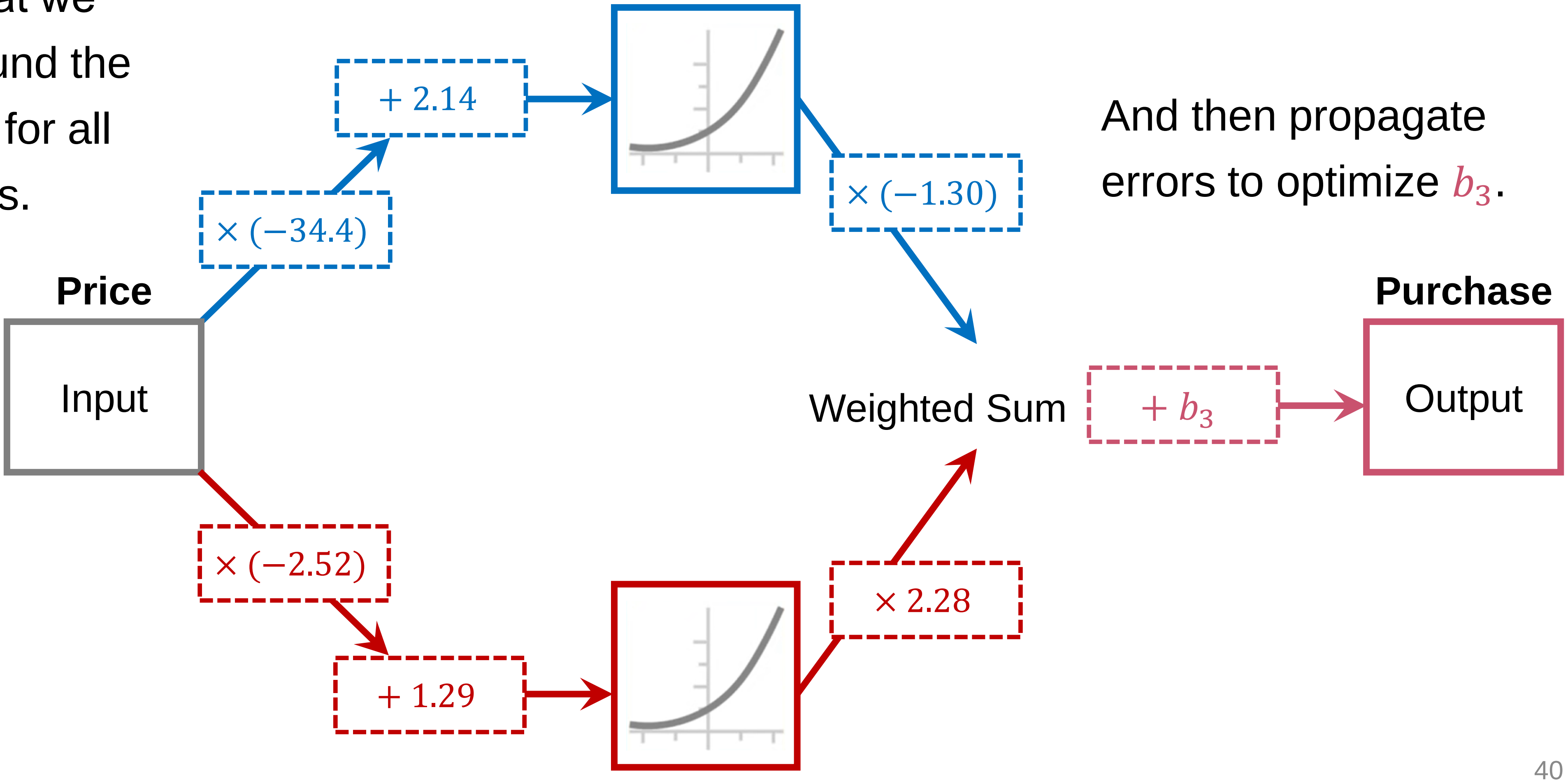
Source: https://www.youtube.com/watch?v=Ilg3gGewQ5U

Backpropagation

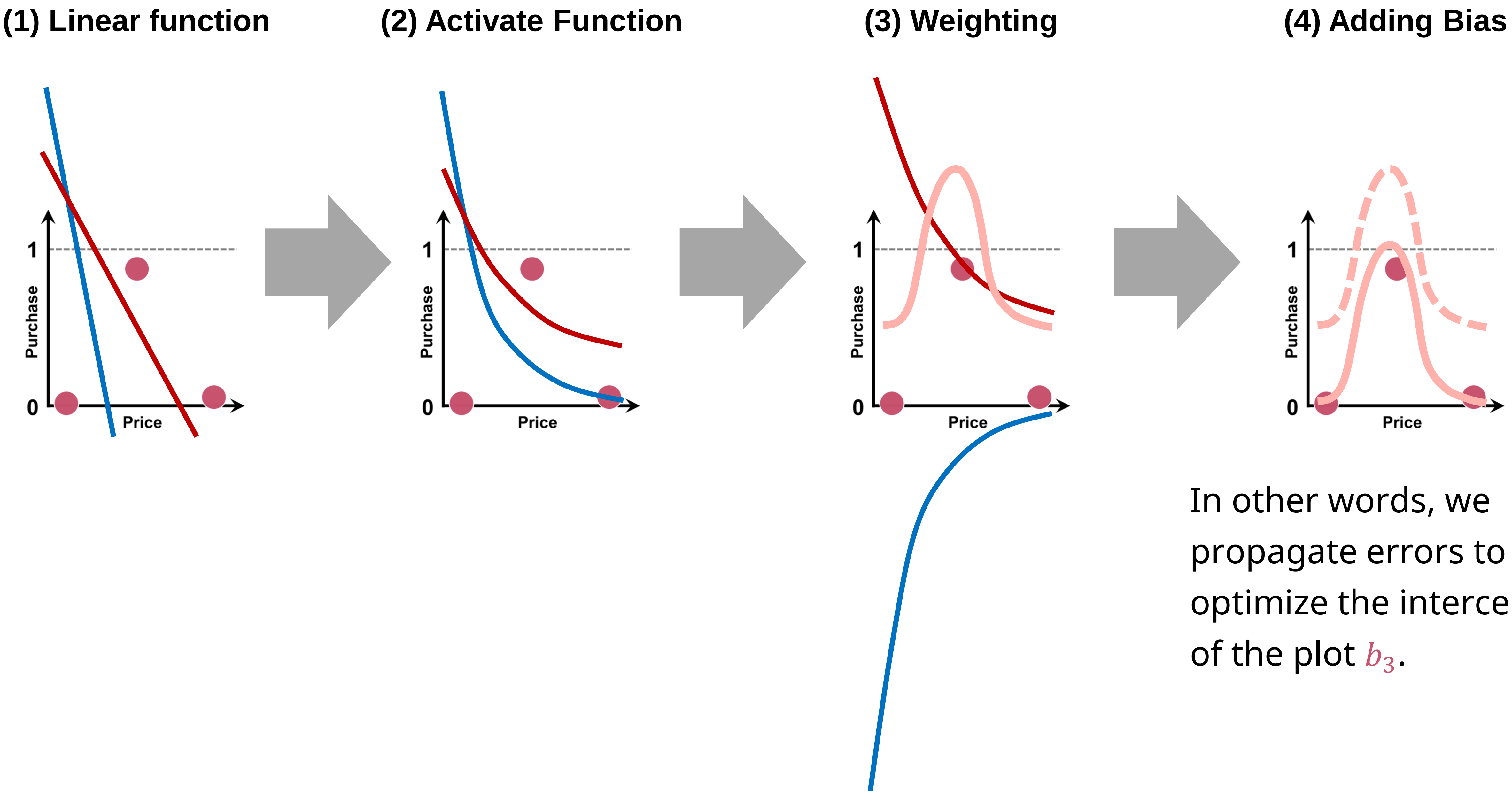
- **Application to the Basic Example**

Let's assume that we have already found the optimum values for all other parameters.

Softplus: $f(Z_1) = log(1 + e^{Z_1})$



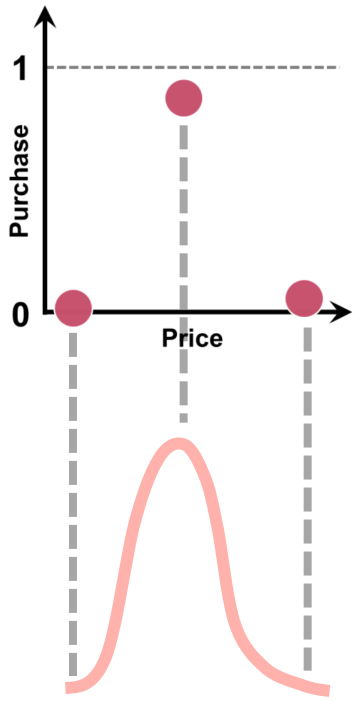
And then propagate errors to optimize b_3.



In other words, we propagate errors to optimize the intercept of the plot b_3.

Backpropagation

- **Application to the Basic Example**



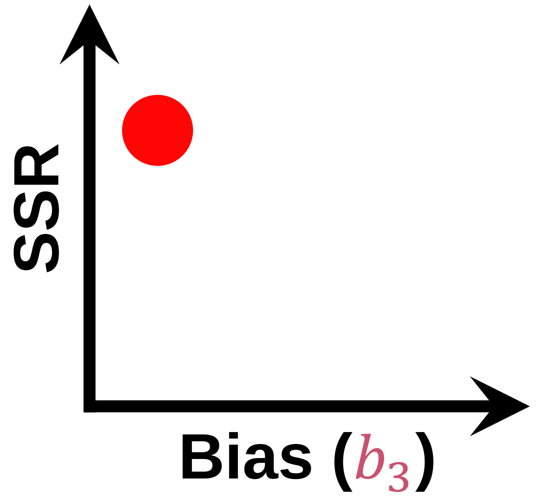
Sum of squared residuals (SSR)

$$= \Sigma\,(\text{Observed} - (\text{Predicted}))^2$$

$$= (Y_1 - (\text{intercept} + f(X_1)))^2 + (Y_2 - (\text{intercept} + f(X_2)))^2 + (Y_3 - (\text{intercept} + f(X_3)))^2$$

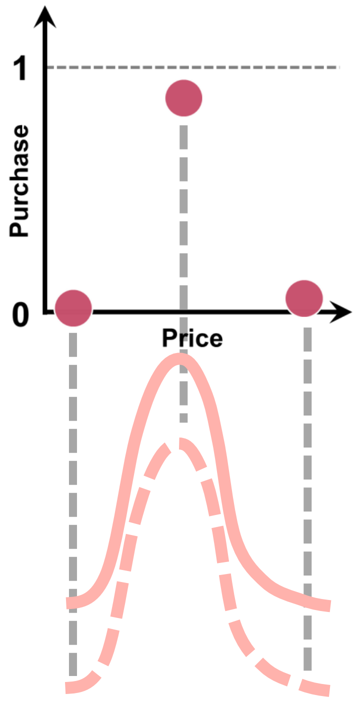
$$= (0 - (-2.6))^2 + (1 - (-1.61))^2 + (0 - (-2.61))^2$$

$$= \mathbf{20.38}$$



Backpropagation

- **Application to the Basic Example**



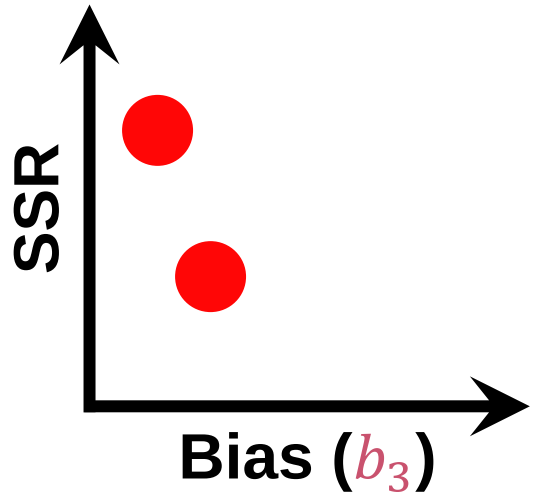
Sum of squared residuals (SSR)

$= \Sigma\,(\textbf{Observed} - (\textbf{Predicted}))^2$

$= (Y_1 - (\text{intercept} + f(X_1)))^2$
$+ (Y_2 - (\text{intercept} + f(X_2)))^2$
$+ (Y_3 - (\text{intercept} + f(X_3)))^2$

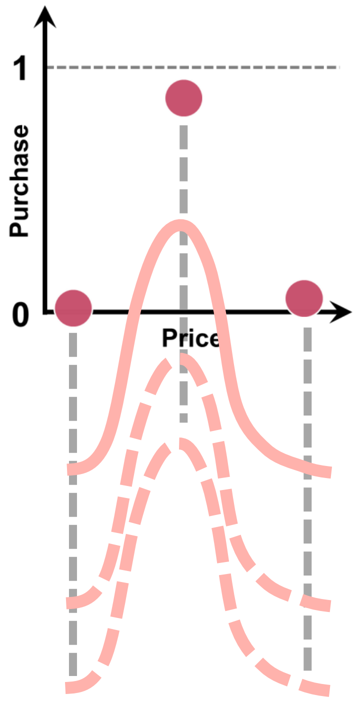
$= (0 - (-1.6))^2$
$+ (1 - (-0.61))^2$
$+ (0 - (-1.61))^2$

$= \mathbf{7.74}$



Backpropagation

- **Application to the Basic Example**



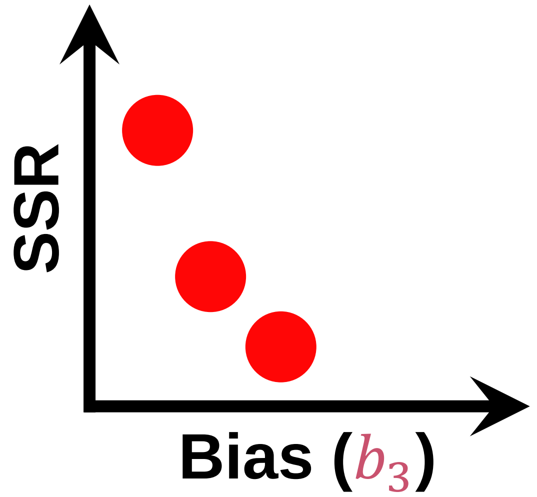
Sum of squared residuals (SSR)

$= \Sigma\,(\text{Observed} - (\text{Predicted}))^2$

$= (Y_1 - (\text{intercept} + f(X_1)))^2$
$+ (Y_2 - (\text{intercept} + f(X_2)))^2$
$+ (Y_3 - (\text{intercept} + f(X_3)))^2$

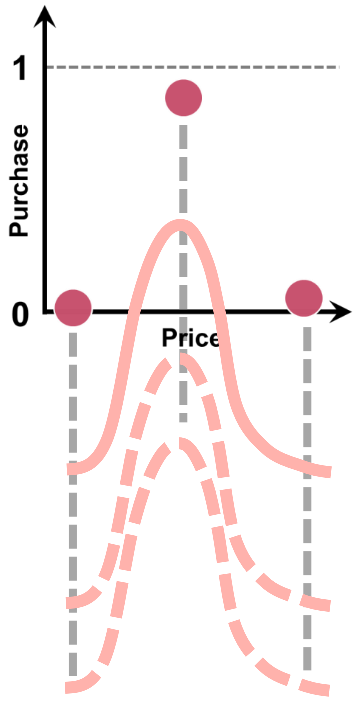
$= (0 - (-0.6))^2$
$+ (1 - (0.61))^2$
$+ (0 - (-0.61))^2$

$= \mathbf{0.88}$



Backpropagation

- **Application to the Basic Example**



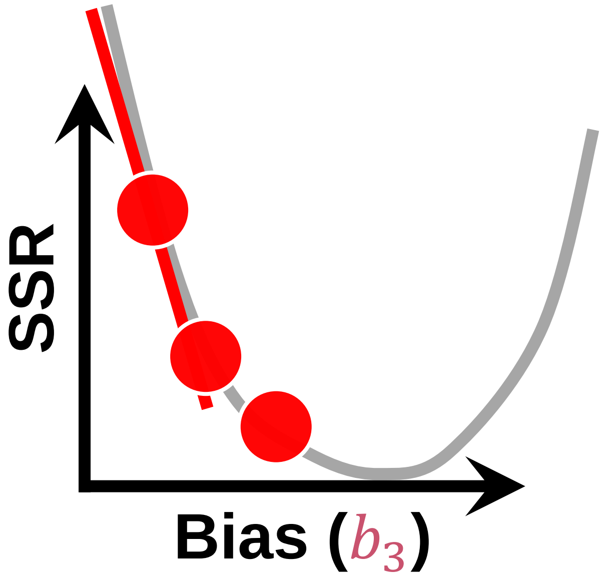
$$\frac{\partial\, SSR}{\partial\, b_3} = \frac{\partial\, SSR}{\partial\, Predicted} \cdot \frac{\partial\, Predicted}{\partial\, b_3}$$

$$= \frac{\partial}{\partial\, Predicted} \cdot \sum_{i=1}^{n=3} (Observed_i - Predicted_i)^2$$

$$= 2 \cdot \sum_{i=1}^{n=3} (Observed_i - Predicted_i) \cdot (-1)$$

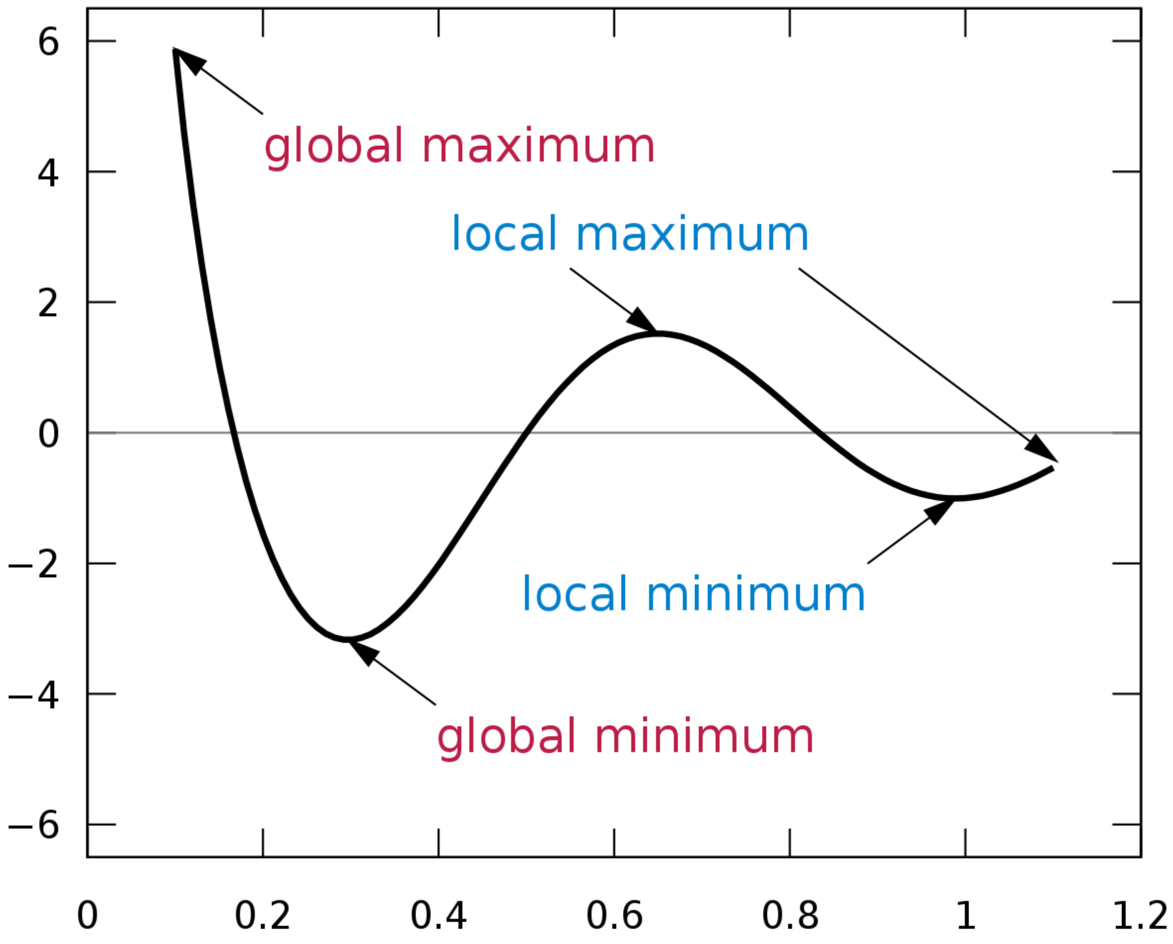
$$\text{Step Size} = \frac{\partial\, SSR}{\partial\, b_3} \cdot \text{Learning Rate}$$

$$\text{New } b_3 = \text{Old } b_3 - \text{Step Size}$$



Backpropagation

- Gradient descent with backpropagation is not guaranteed to find the global minimum of the error function, but only a local minimum; also, it has trouble crossing plateaus in the error function landscape.
- This issue, caused by the non-convexity of error functions in neural networks, was long thought to be a major drawback, but LeCun et al. (2015) argue that “recent theoretical and empirical results strongly suggest that local minima are not a serious issue in general.”



LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton. "Deep Learning." *Nature* 521, no. 7553 (2015): 436-444.

References

- Activation Functions in Neural Networks by Towards Data Science
- Common Loss functions in machine learning by Towards Data Science
- Deep Learning Basics by Google Colab
- Deep Learning Tutorial for Beginners by Simplilearn
- Estimation of Neurons and Forward Propagation in Neural Net by Analytics Vidhya
- Learning Rate in Machine Learning by Deepchecks
- Neural Networks / Deep Learning by StatQuest
- Python Deep Learning Tutorial by Tutorials Point
- Softplus and softminus

Until Next Class…

What you need to do, what you will do

Upcoming Schedules

- **Feb 21 (Fri): Hands-on Assignment #1 Due**
- **Feb 27 (Thu): Next Week**
  - Deep Learning: Basics (Remaining Part)
  - Short Session for Mid-term Preparation
- **Mar 6 (Thu): Mid-term Exam**
- **Mar 13 (Thu)**
  - Feedback for Hands-on Assignment #1
  - Deep Learning: Advanced (Concept + Hands-on)
  - Hands-on Assignment #2 is out.
- **Mar 20 (Thu):** No Class (*Spring Recess*)

Jaeung Sim

Assistant Professor

School of Business, University of Connecticut

jaeung.sim@uconn.edu