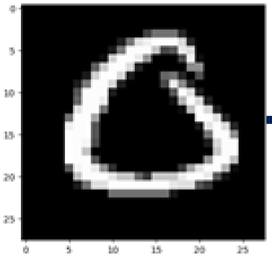
Multilayer Perceptron (MLP)

Rowel Atienza, PhD
University of the Philippines
github.com/roatienza
2023

Problem Definition (Supervised Learning)

Given a dataset $\mathcal{D} = (\boldsymbol{x}, \boldsymbol{y})$, find a function $f(\boldsymbol{x}, \boldsymbol{\theta}): \boldsymbol{x} \in \mathbb{R}^N \rightarrow \boldsymbol{y} \in \mathbb{R}^M$



$$f(\boldsymbol{x}, \boldsymbol{\theta})$$

$$\boldsymbol{y} \in \mathbb{R}^{10}$$

$$\boldsymbol{x} \in \mathbb{R}^{28 \times 28 \times 1}$$

What is $f(\cdot)$?

$f(\cdot)$ is generally a non-linear function that maps an input distribution $\boldsymbol{x} \sim p(\boldsymbol{x})$ to an output distribution $\boldsymbol{y} = p(\boldsymbol{y}|\boldsymbol{x})$:

$$\boldsymbol{y} = f(\boldsymbol{x}) = p(\boldsymbol{y}|\boldsymbol{x})$$

$f(\cdot)$ is an estimator of density $p(\boldsymbol{y}|\boldsymbol{x})$

General Function Approximator

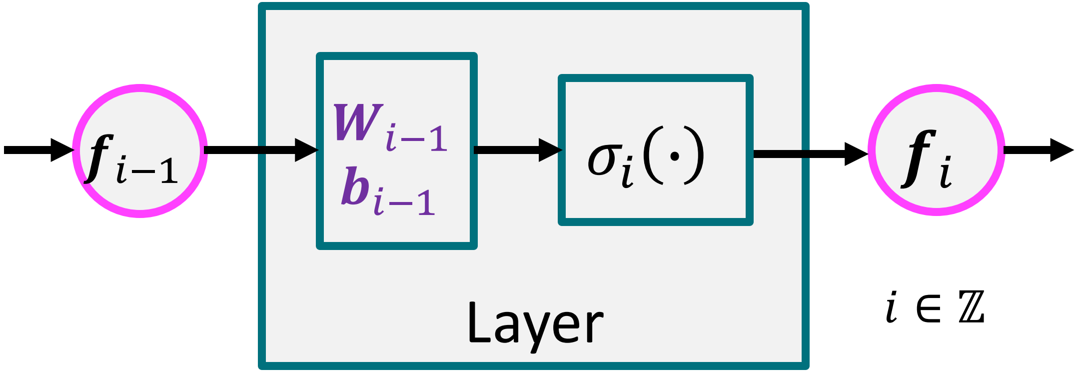
Theorem: Any function $f(\cdot)$ can be approximated by a composition of several smaller functions f_i:

$$\boldsymbol{y} = f(\boldsymbol{x}) \approx f_n \circ f_{n-1} \circ f_{n-2} \circ \cdots \circ f_1(\boldsymbol{x})$$

$\ni f_0 = \boldsymbol{x},\ n \in \mathbb{Z}$

f_i: Keras Dense Layer (`Dense`) or PyTorch Linear (`Linear`) + Activation

$$\boldsymbol{f}_i(\boldsymbol{f}_{i-1};\, \boldsymbol{\theta}_{i-1}) = \sigma_i(\boldsymbol{W}_{i-1}\boldsymbol{f}_{i-1} + \boldsymbol{b}_{i-1})$$

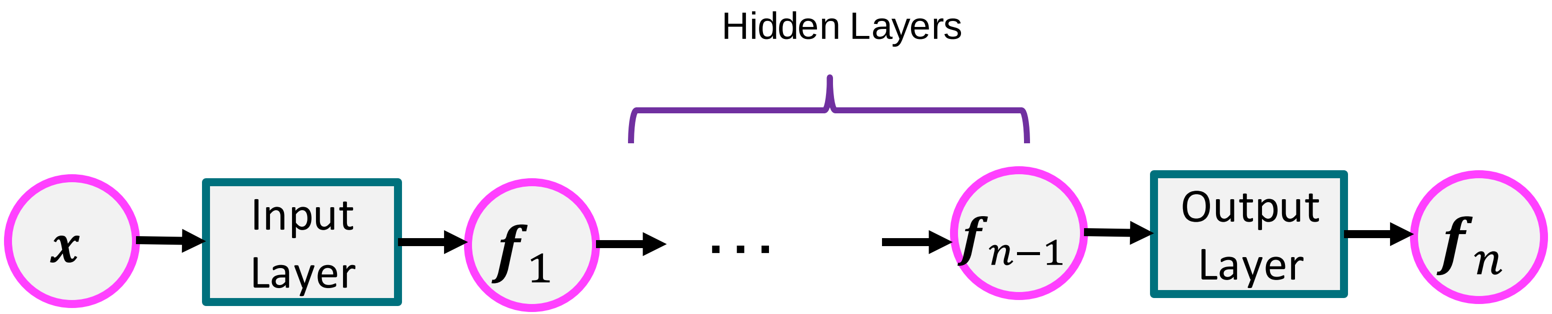


Weights: $\boldsymbol{W} = \{\boldsymbol{W}_0, \boldsymbol{W}_1, \dots, \boldsymbol{W}_{n-1}\}$ Biases: $\boldsymbol{b} = \{\boldsymbol{b}_0, \boldsymbol{b}_1, \dots, \boldsymbol{b}_{n-1}\}$

Weights, Biases := Parameters: $\boldsymbol{\theta} = \{\boldsymbol{\theta}_0, \boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_{n-1}\}$ $\boldsymbol{\theta}_{i-1} = \{\boldsymbol{W}_{i-1}, \boldsymbol{b}_{i-1}\}$

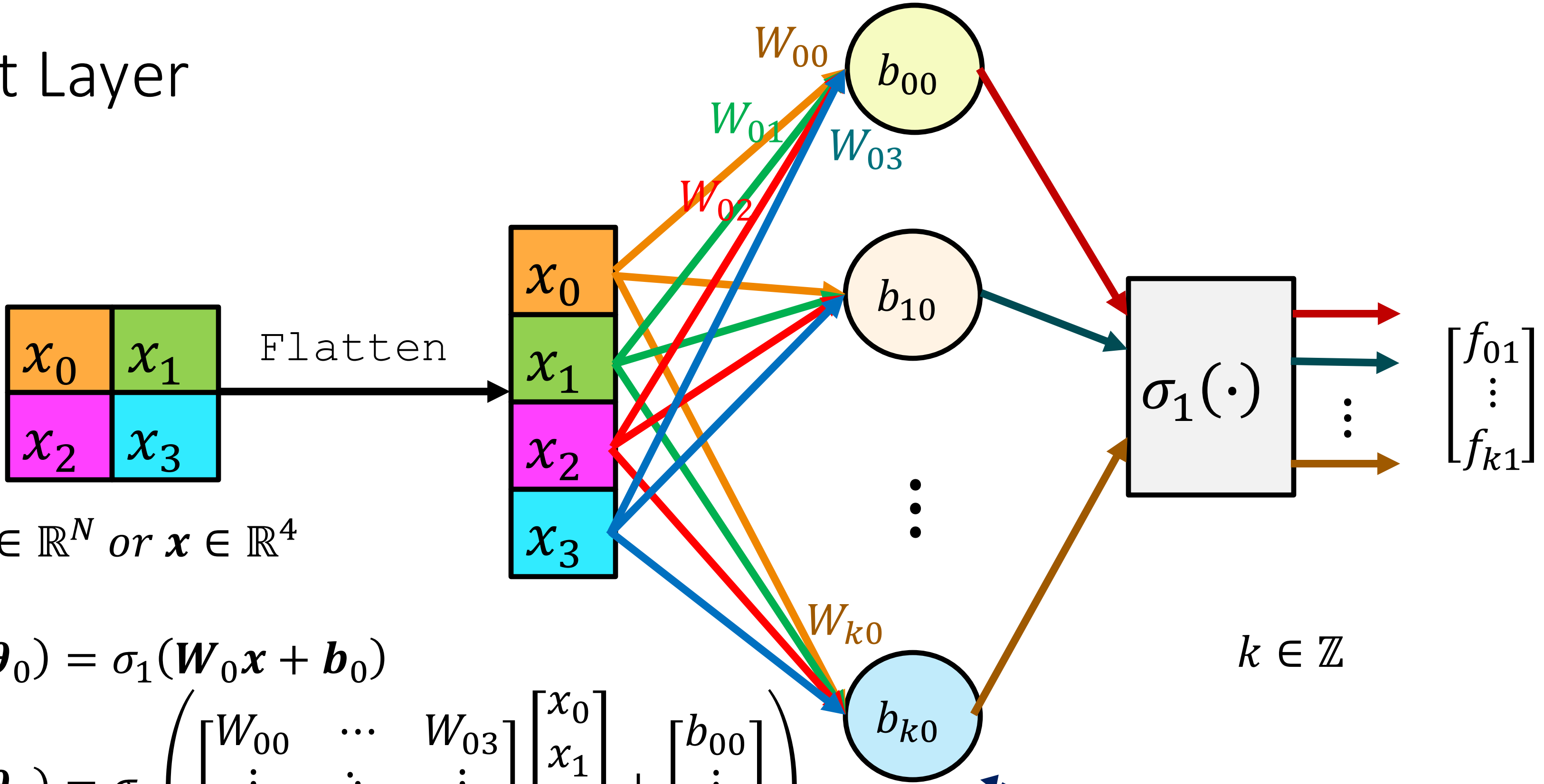
Activation function: $\sigma(\cdot)$

MLP: Function Approximator Implementation



$$\boldsymbol{y} = f(\boldsymbol{x}) \approx f_n \circ f_{n-1} \circ f_{n-2} \circ \cdots \circ f_1\,(\boldsymbol{x})$$
$$\ni f_0 = \boldsymbol{x},\ n \in \mathbb{Z}$$

Input Layer

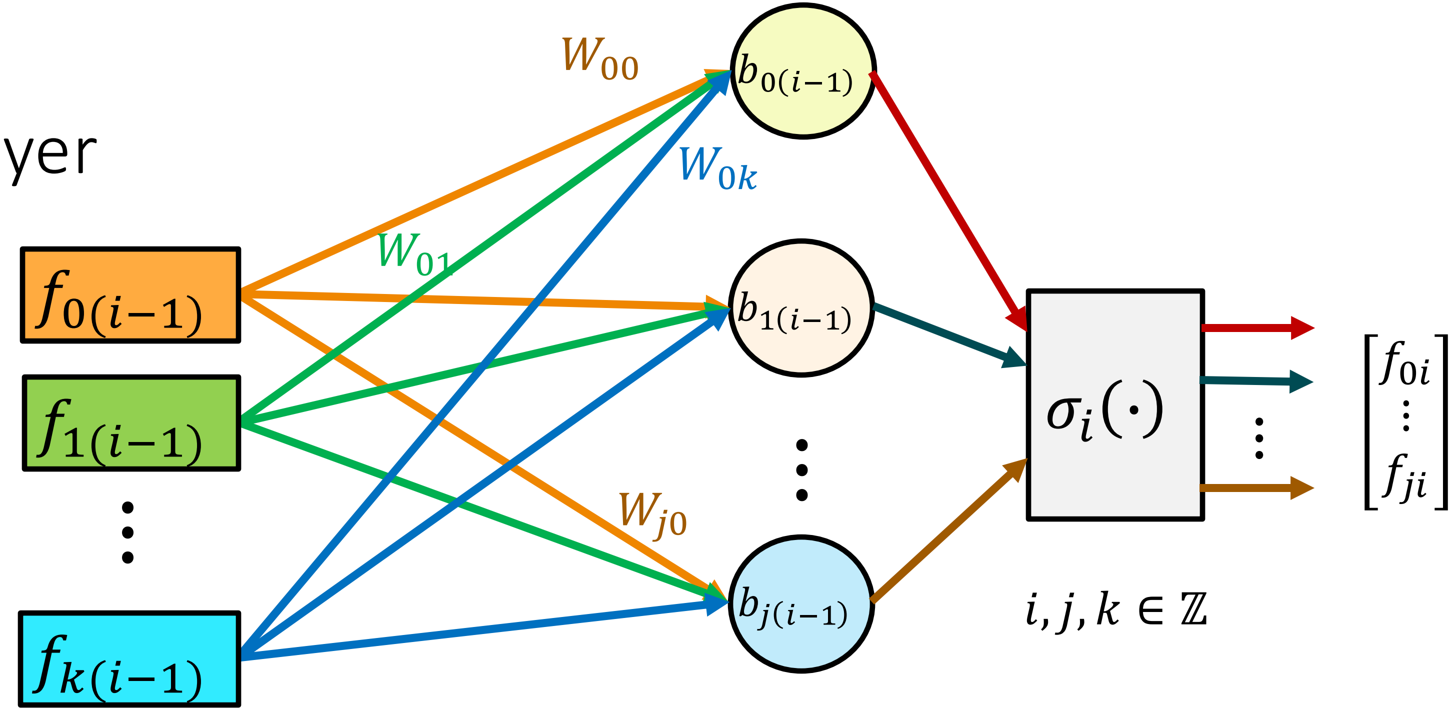


$\boldsymbol{x} \in \mathbb{R}^N$ *or* $\boldsymbol{x} \in \mathbb{R}^4$

$$\boldsymbol{f}_1(\boldsymbol{x};\ \boldsymbol{\theta}_0) = \sigma_1(\boldsymbol{W}_0\boldsymbol{x} + \boldsymbol{b}_0)$$

$$\boldsymbol{f}_1(\boldsymbol{x};\ \boldsymbol{\theta}_0) = \sigma_1\left(\begin{bmatrix} W_{00} & \cdots & W_{03} \\ \vdots & \ddots & \vdots \\ W_{k0} & \cdots & W_{k3} \end{bmatrix}\begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{bmatrix} + \begin{bmatrix} b_{00} \\ \vdots \\ b_{k0} \end{bmatrix}\right)$$

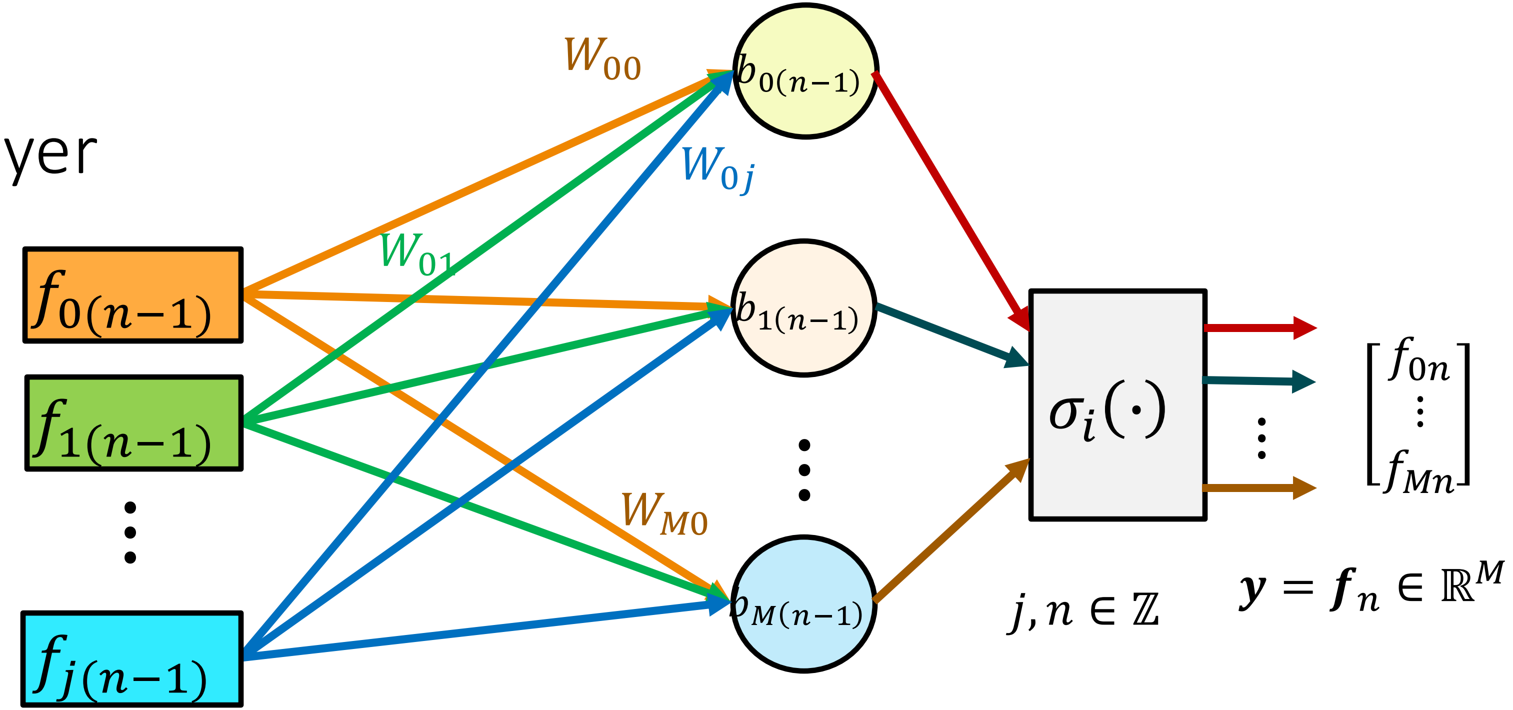
Hidden Layer



$i, j, k \in \mathbb{Z}$

$$\boldsymbol{f}_i(\boldsymbol{f}_{i-1};\, \boldsymbol{\theta}_{i-1}) = \sigma_i\left(\begin{bmatrix} W_{00} & \cdots & W_{0k} \\ \vdots & \ddots & \vdots \\ W_{j0} & \cdots & W_{jk} \end{bmatrix}\begin{bmatrix} f_{0(i-1)} \\ f_{1(i-1)} \\ \vdots \\ f_{k(i-1)} \end{bmatrix} + \begin{bmatrix} b_{0(i-1)} \\ \vdots \\ b_{j(i-1)} \end{bmatrix}\right)$$

Output Layer

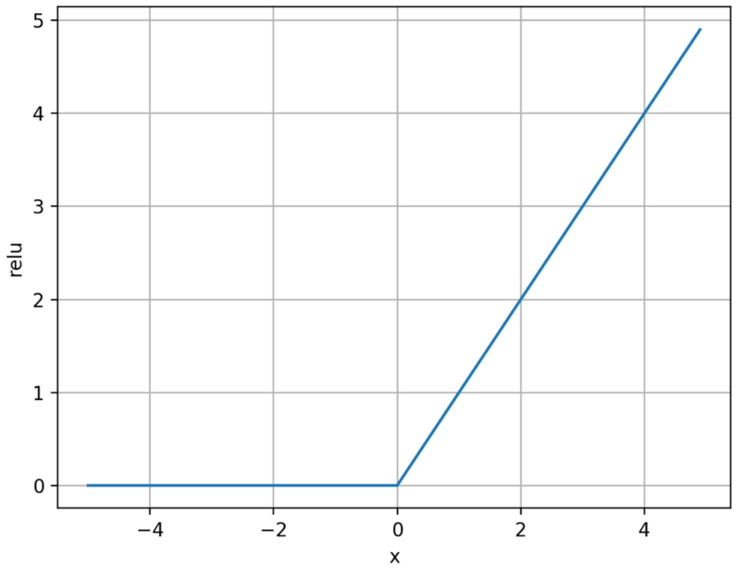


$$\boldsymbol{f}_n(\boldsymbol{f}_{n-1};\, \boldsymbol{\theta}_{n-1}) = \sigma_n\left(\begin{bmatrix} W_{00} & \cdots & W_{0j} \\ \vdots & \ddots & \vdots \\ W_{M0} & \cdots & W_{Mj} \end{bmatrix}\begin{bmatrix} f_{0(n-1)} \\ f_{1(n-1)} \\ \vdots \\ f_{j(n-1)} \end{bmatrix} + \begin{bmatrix} b_{0(n-1)} \\ \vdots \\ b_{M(n-1)} \end{bmatrix}\right)$$

Activation Function: $\sigma_i(\cdot)$

Rectified Linear Unit:

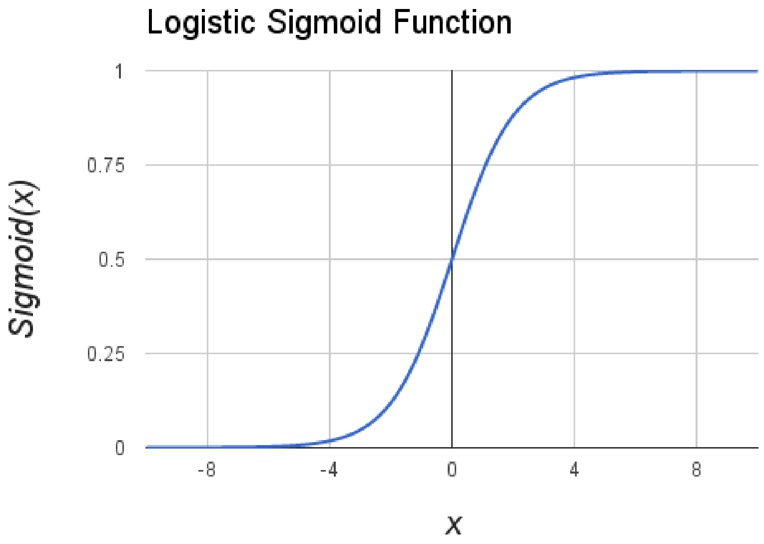
$$\sigma(x) = ReLU(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$



Activation Function: $\sigma_i(\cdot)$

Sigmoid:

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



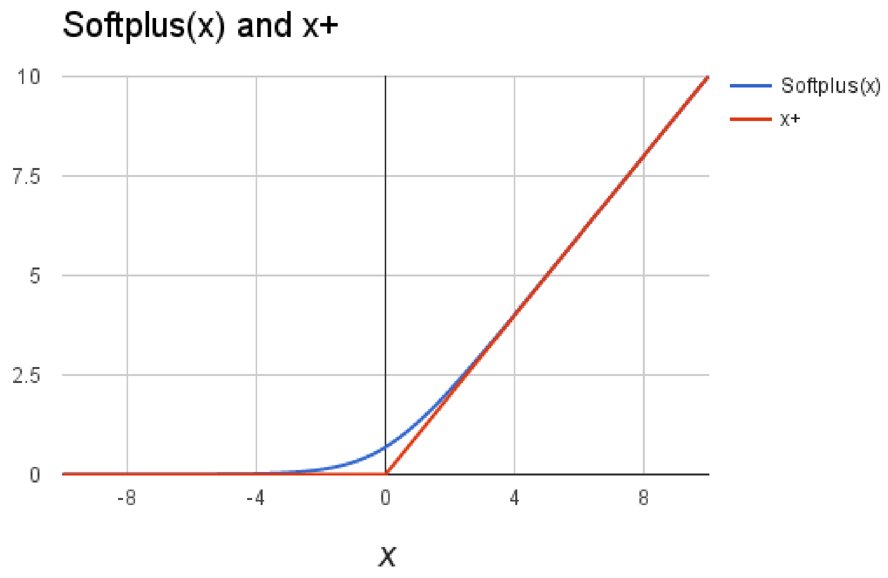
Activation Function: $\sigma_i(\cdot)$

Hyperbolic tangent:

$$\sigma(x) = \tanh x$$

Softplus:

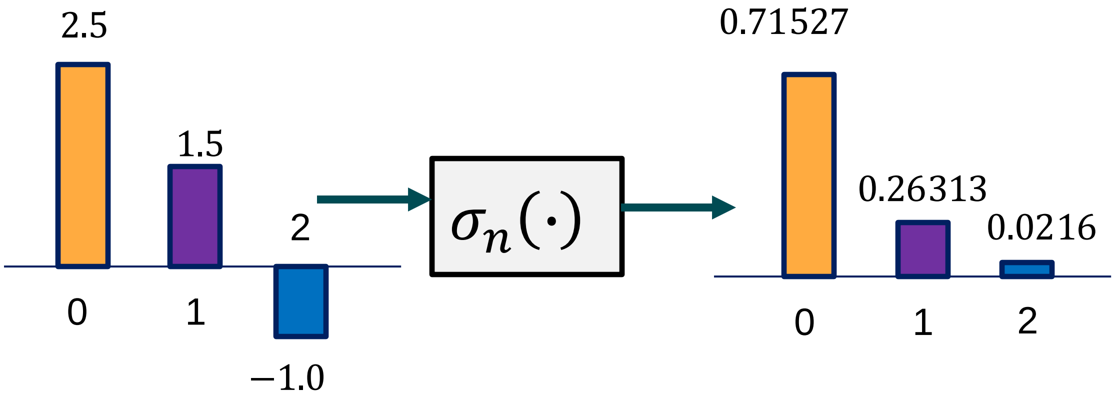
$$\sigma(x) = \ln(1 + e^x)$$



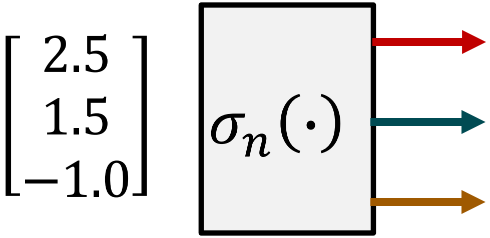
Activation Function: $\sigma_i(\cdot)$

Softmax:

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{k=0}^{C} e^{x_k}}$$



Probabilities
Sum is 1.0



$$\sigma_n\left(\begin{bmatrix} f_{0n} \\ f_{1n} \\ f_{3n} \end{bmatrix}\right) = softmax\left(\begin{bmatrix} 2.5 \\ 1.5 \\ -1.0 \end{bmatrix}\right) = \begin{bmatrix} 0.71527 \\ 0.26313 \\ 0.02160 \end{bmatrix}$$

Which activation to use?

Input and Hidden Layers

ReLU, GELU – inject non-linearity

Softplus – used in deep reinforcement learning

Linear – pass through

Output Layer

Sigmoid – Bernoulli Distribution, Normalized Linear Regression

Softmax – Logistic Regression

Linear – Un-normalized Linear Regression

How to learn $f(\cdot)$ from data?

Recall: Norms, Metrics, Distances from ML

Objective is to reduce the distance of the *prediction* $\boldsymbol{y} = f(\boldsymbol{x})$ from the ground truth label $\widetilde{\boldsymbol{y}}$

This distance, norm, or metric is oftentimes called a ***Loss Function*** or an ***Objective Function***

Loss Function	Equation
Mean Squared Error (MSE)	$\sum_{i=1}^{categories} \left(y_i^{label} - y_i^{prediction}\right)^2$
Mean Absolute Error (MAE)	$\sum_{i=1}^{categories} \left\|y_i^{label} - y_i^{prediction}\right\|$
Categorical Cross Entropy (CE)	$-\sum_{i=1}^{categories} y_i^{label} \log y_i^{prediction}$
Binary Cross Entropy (BCE)	$-y_1^{label} \log y_1^{prediction} - \left(1 - y_1^{label}\right) \log\left(1 - y_1^{prediction}\right)$

Optimization

Given the dataset $\{\mathcal{D}_{train}, \mathcal{D}_{test}\} = \{(\boldsymbol{x}_n, \boldsymbol{y}_n), (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$, we minimize the loss function on $\mathcal{D}_{train}$ and we measure the performance on $\mathcal{D}_{test}$

Optimization Algorithm: Stochastic Gradient Descent (SGD)

Variants of SGD: Adam, AdamW

Optimization Recipe

Initialize all weights by random values

Better initializers: Kaiming, Glorot, Uniform, Normal, LeCun,

Biases by zero or small positive values

Usually, default initialization algorithms are good enough

Preprocessing of Data

Input

Normalize such that $x_i \in [0., 1.]$

Adjust such that inputs has zero mean and unit variance

Output

In logistic regression, convert all labels to one-vectors

Example: In MNIST, digit 8 label is $\tilde{y} = [0,0,0,0,0,0,0,1,0,0]^T$

In linear regression, normalize outputs such that such that $y_i \in [0., 1.]$ or such that $y_i \in [-1., 1.]$

Hyper-parameters

Tunable network parameters

Depth or value of n in f_n

Width values of k and j in the input and hidden layers

Tunable training parameters

Learning rate

Learning rate scheduler

Warm-up

Batch size, Epochs

Optimization algorithm

In Summary

MLP is an implementation of the general function approximator

MLP is made of layers as building blocks

Design choices such as hyper-parameters, activation functions, etc

END