

# Multilayer Perceptron (MLP)

Rowel Atienza

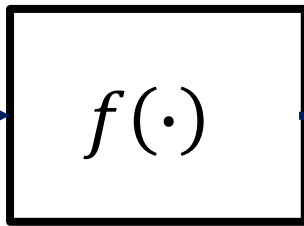
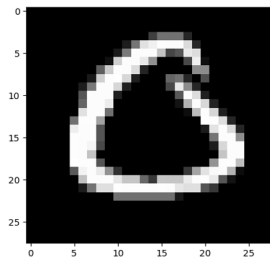
rowel@eee.upd.edu.ph

*University of the Philippines*

*Updated: 19 Sept 2020*

# Problem Definition (Supervised Learning)

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



$$\mathbf{y} = 0 \in \mathbb{R}^1$$

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

# What is $f(\cdot)$ ?

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

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

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

# General Function Approximator

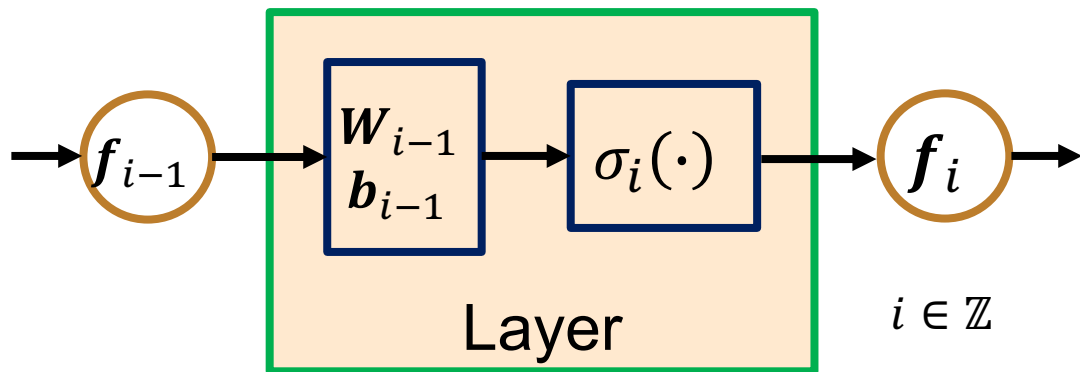
*Theorem:* Any function  $f(\cdot)$  can be approximated by a composition of several smaller functions  $f_i$ :

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

$$\exists f_0 = \mathbf{x}, n \in \mathbb{Z}$$

$f_i$ : Dense Layer (`tf.keras`) or Fully-Connected Layer (PyTorch)

$$f_i(f_{i-1}; \theta_{i-1}) = \sigma_i(W_{i-1}f_{i-1} + b_{i-1})$$

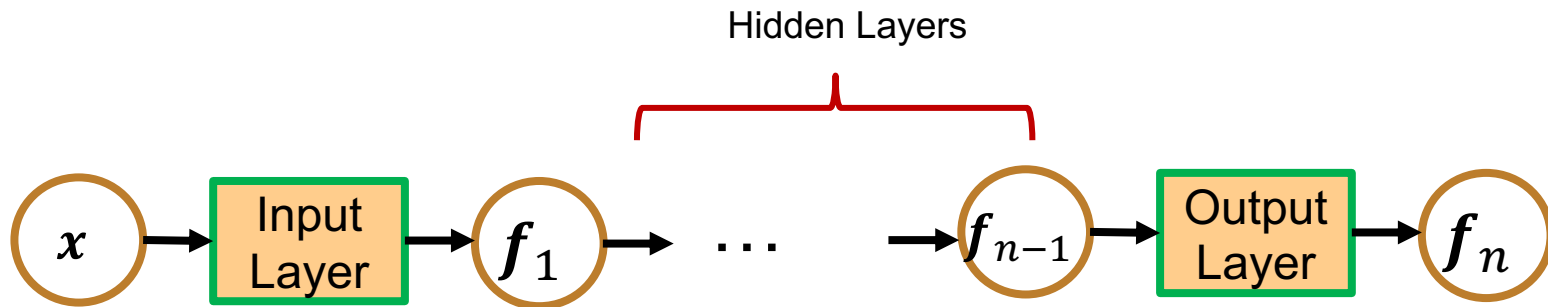


Weights:  $W = \{W_0, W_1, \dots, W_{n-1}\}$       Biases:  $b = \{b_0, b_1, \dots, b_{n-1}\}$

Weights, Biases := Parameters:  $\theta = \{\theta_0, \theta_1, \dots, \theta_{n-1}\}$        $\theta_{i-1} = \{W_{i-1}, b_{i-1}\}$

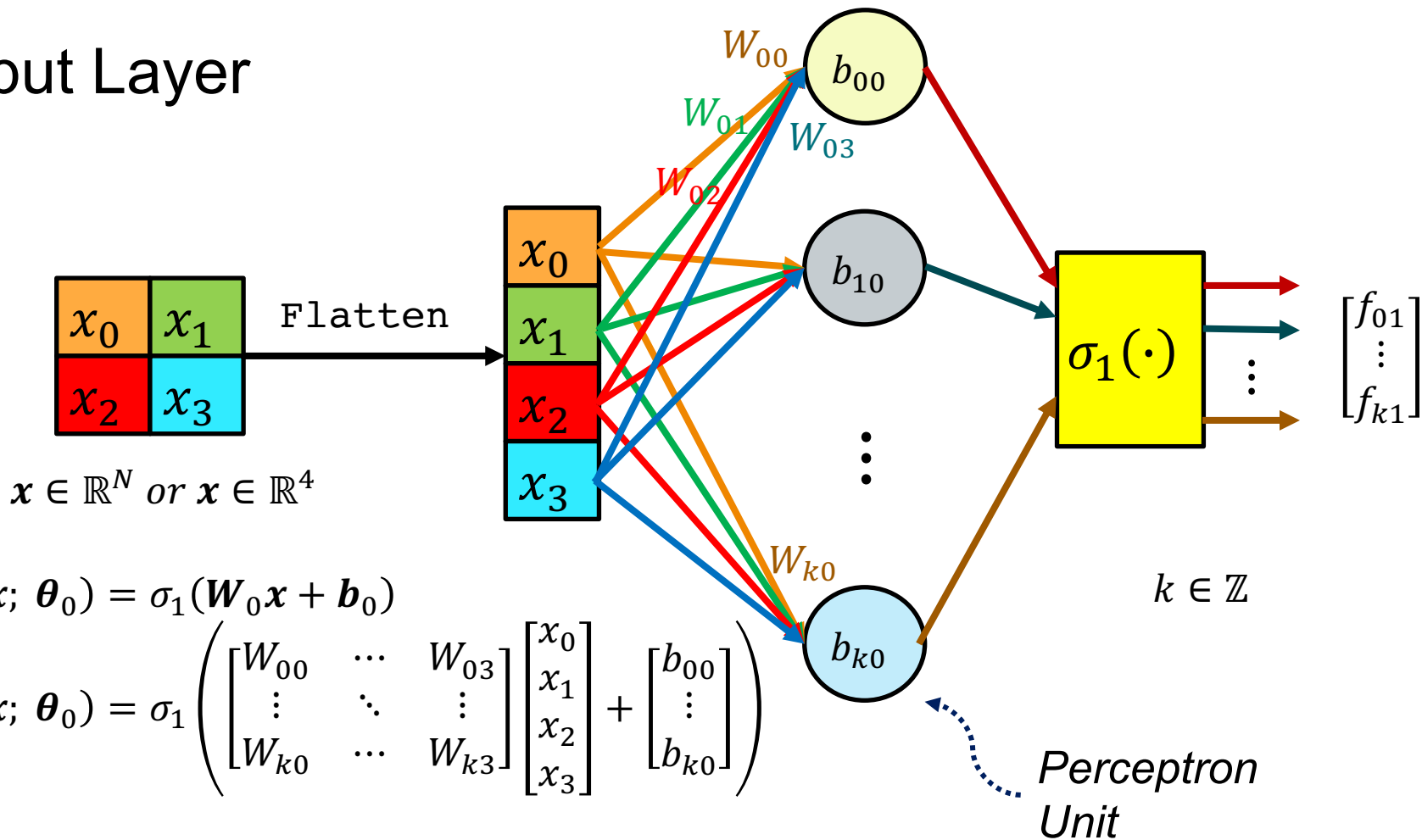
Activation function:  $\sigma(\cdot)$

# MLP: Function Approximator Implementation

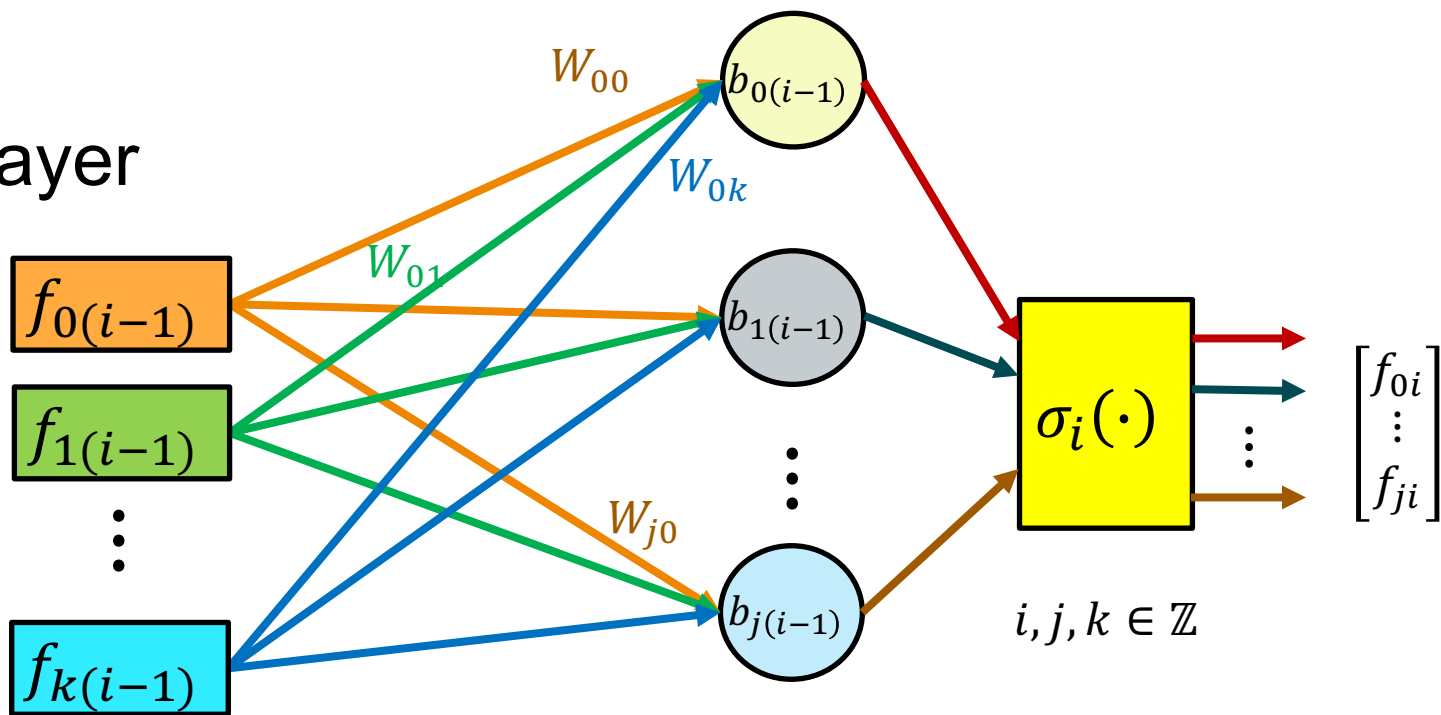


$$\mathbf{y} = f(\mathbf{x}) \approx f_n \circ f_{n-1} \circ f_{n-2} \circ \dots \circ f_1 (\mathbf{x})$$
$$\ni f_0 = \mathbf{x}, n \in \mathbb{Z}$$

# Input Layer



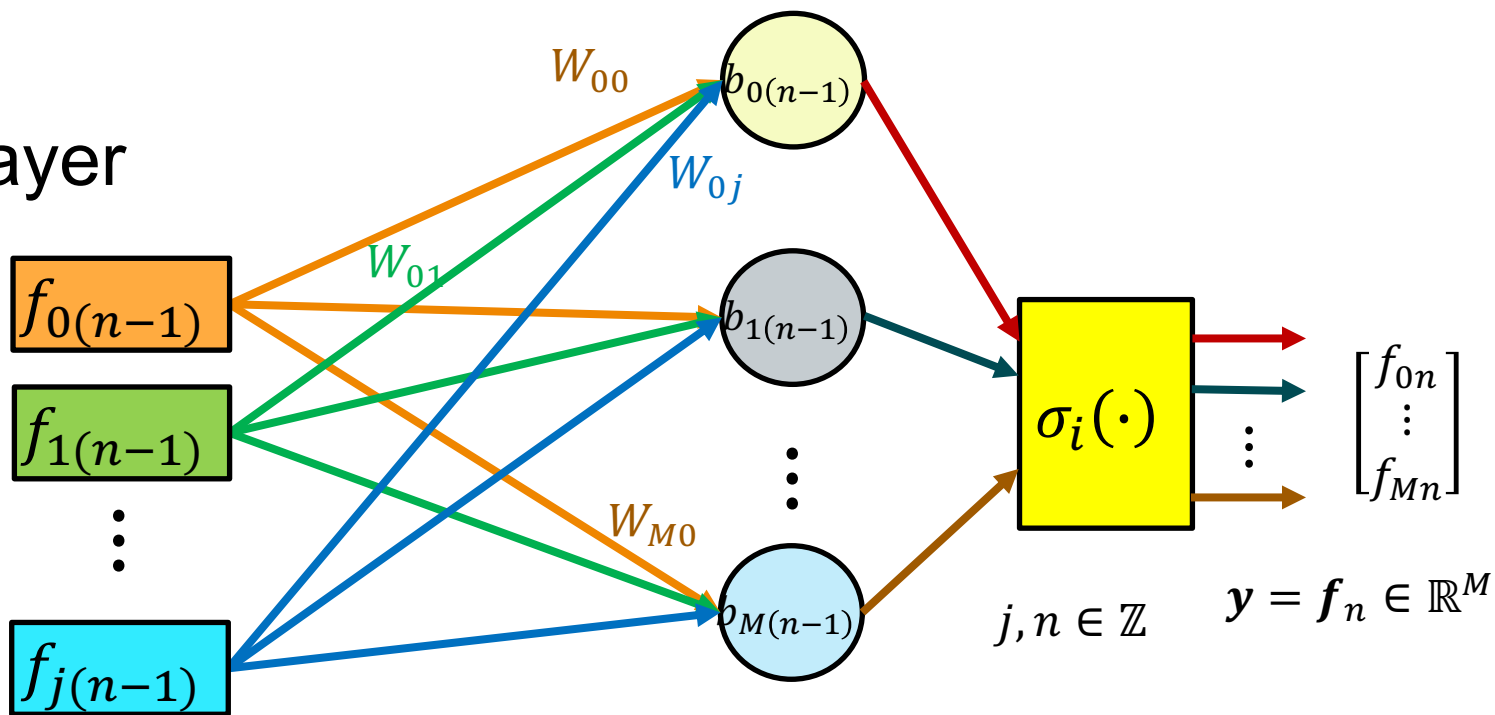
# Hidden Layer



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



# Output Layer



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

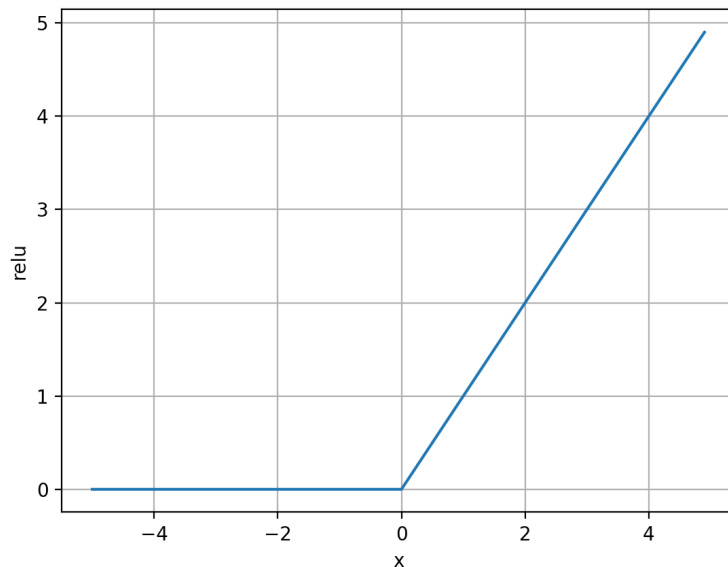
# Activation Function: $\sigma_i(\cdot)$

Identity or Linear:

$$\sigma(x) = x$$

Rectified Linear Unit:

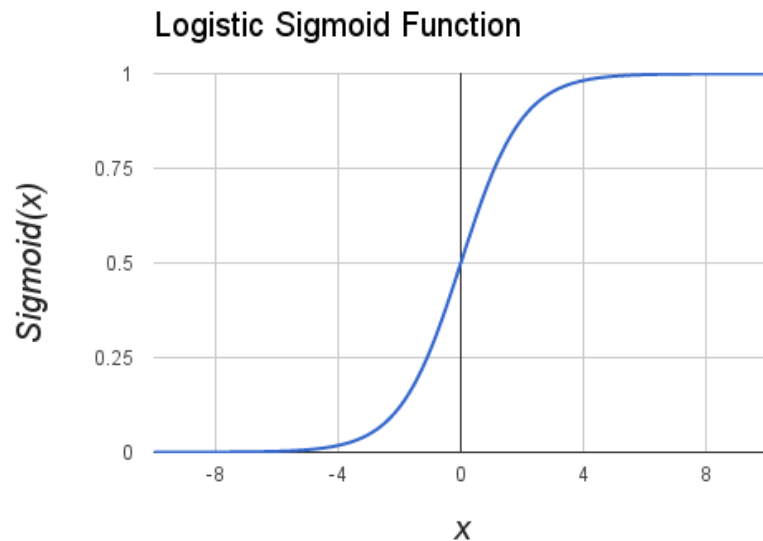
$$\sigma(x) = \text{ReLU}(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$



# Activation Function: $\sigma_i(\cdot)$

Sigmoid:

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



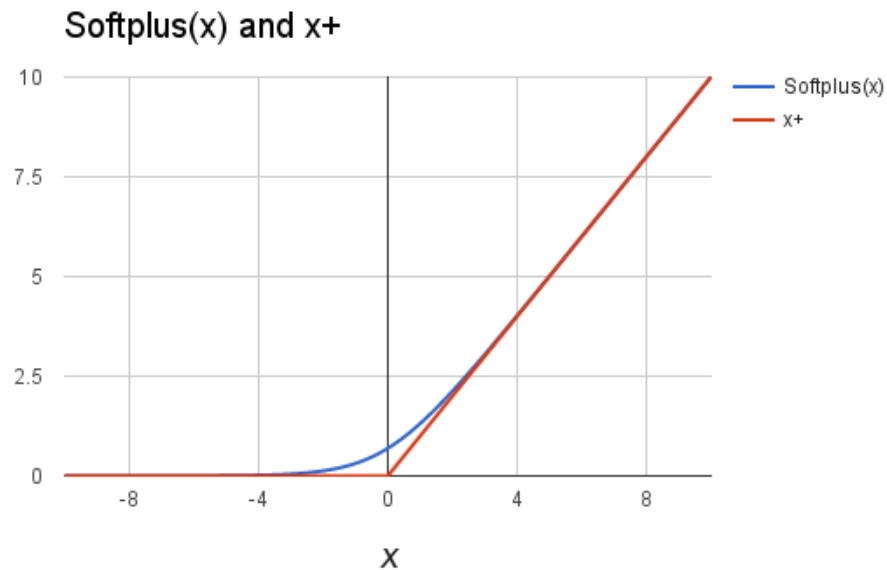
# Activation Function: $\sigma_i(\cdot)$

Hyperbolic tangent:

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

Softplus:

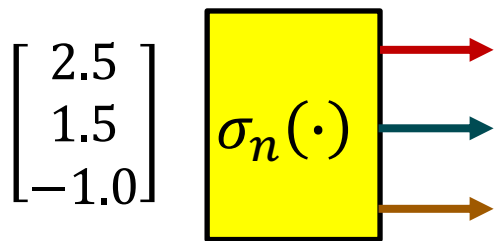
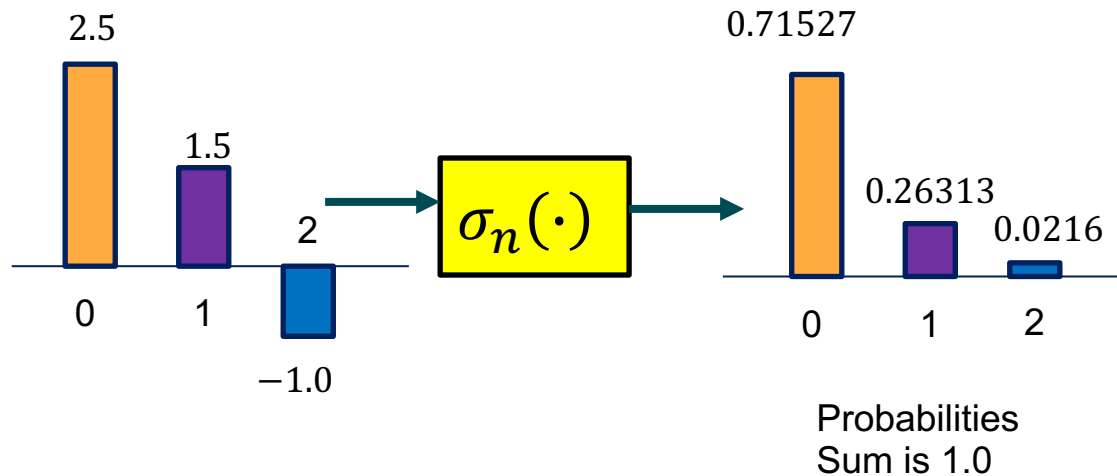
$$\sigma(x) = \ln(1 + e^x)$$



# Activation Function: $\sigma_i(\cdot)$

Softmax:

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



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

# Which activation to use?

## Input and Hidden Layers

*Linear* – recommended

*ReLU* – recommended

*Softplus* – complex, avoid!!!

## Output Layer

*Linear* – Un-normalized Linear Regression

*Sigmoid* – Bernoulli Distribution, Normalized Linear Regression

*Softmax* – Logistic Regression

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

*Recall:* Norms, Metrics,  
Distances from ML

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

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

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

# Optimization

Given the dataset  $\mathcal{D} = (\mathbf{x}, \mathbf{y}) = (\tilde{\mathbf{x}}, \tilde{\mathbf{y}}) = \{(\tilde{\mathbf{x}}_{train}, \tilde{\mathbf{y}}_{train}), (\tilde{\mathbf{x}}_{test}, \tilde{\mathbf{y}}_{test})\} = \{\mathcal{D}_{train}, \mathcal{D}_{test}\}$ , 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, RMSprop



# SGD Optimization Recipe

Initialize all weights by random values

Biases by zero or small positive values

Better initializers: [Glorot](#), Uniform, Normal, LeCun, He

# 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  $\tilde{y} = [0,0,0,0,0,0,0,0,1,0,0]^T$*

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

Batch size, Epochs

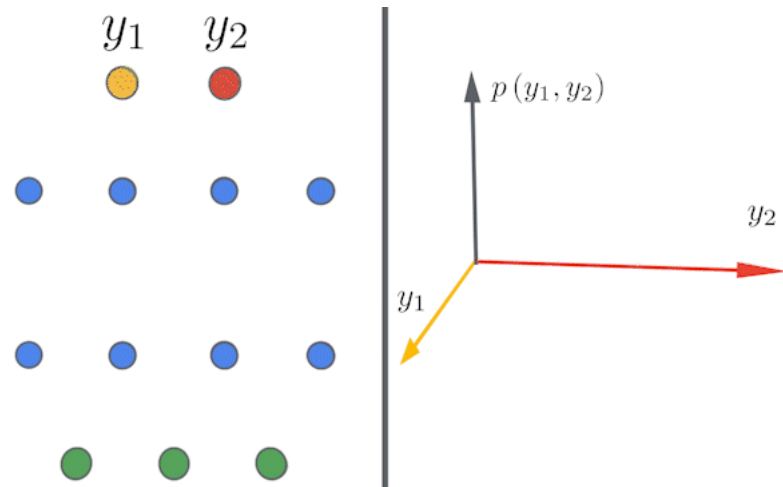
Optimization algorithm

# 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



Left: A schematic showing how deep neural networks induce simple input / output maps as they become infinitely wide. Right: As the width of a neural network increases , we see that the distribution of outputs over different random instantiations of the network becomes Gaussian.

<https://ai.googleblog.com/2020/03/fast-and-easy-infinitely-wide-networks.html>