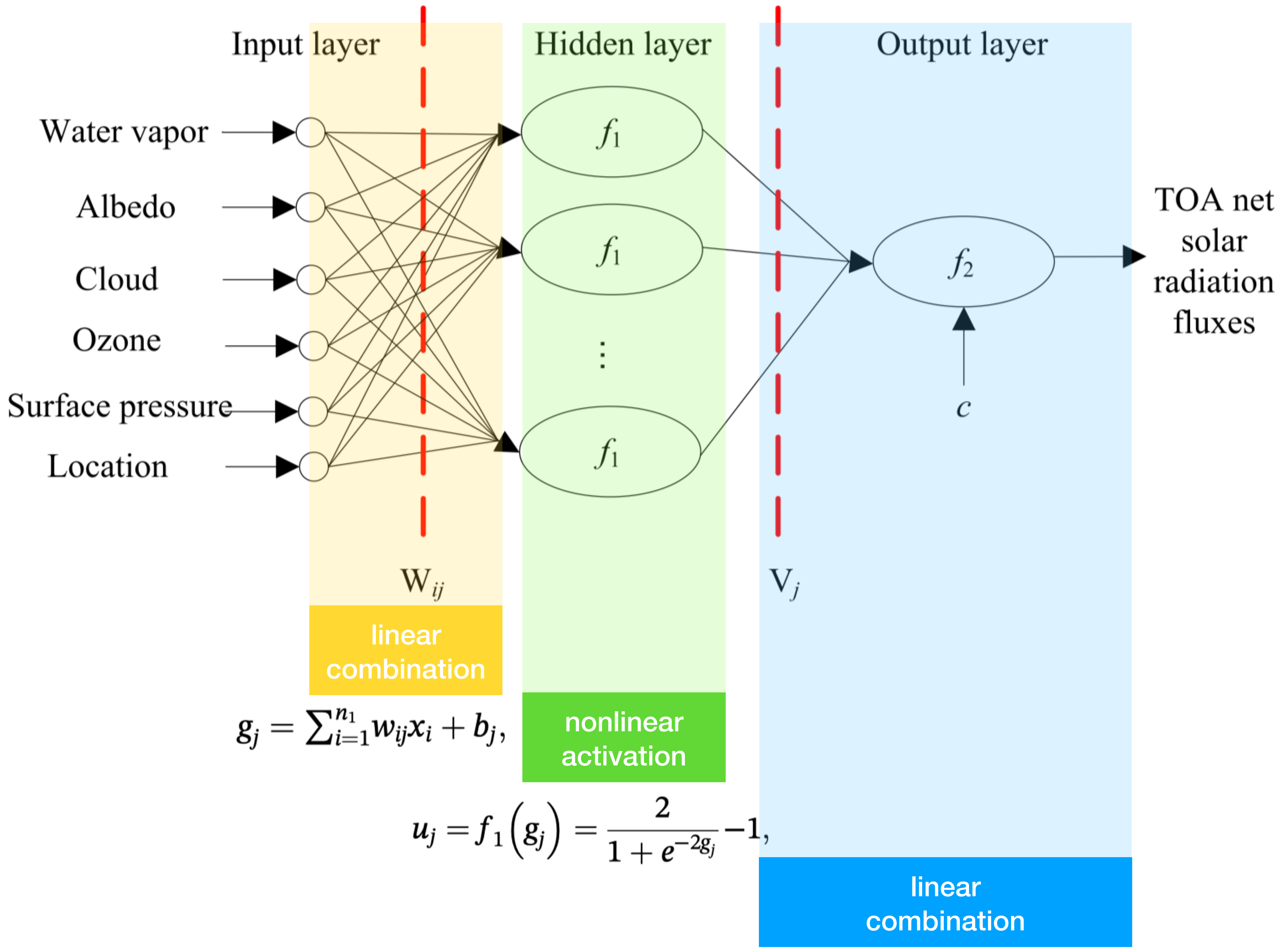
Some Basics of Neural Network Model

Wen

2022-05-20

NN model in Zhu et al. (2018)



$$g_j = \sum_{i=1}^{n_1} w_{ij} x_i + b_j,$$

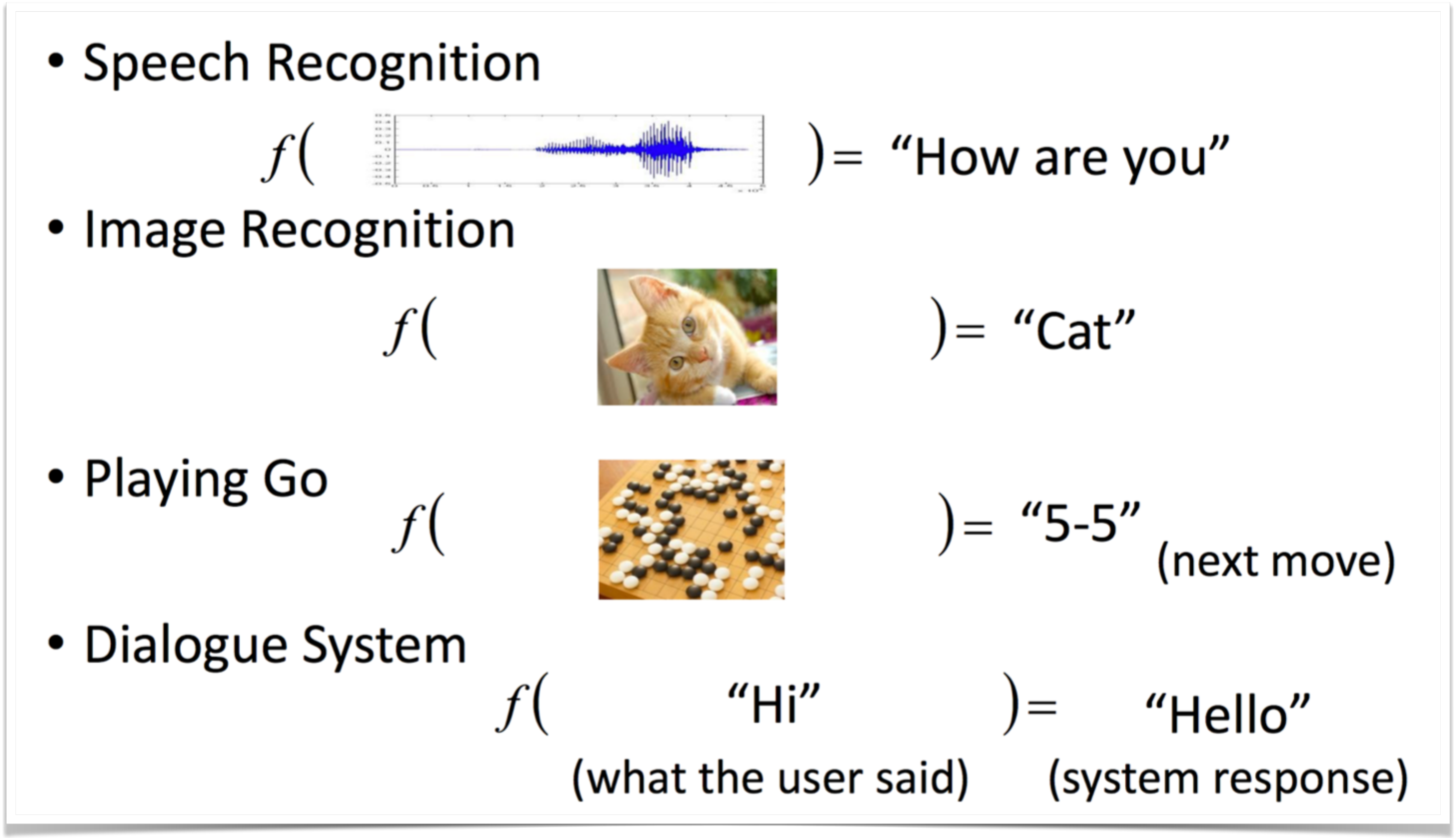
$$u_j = f_1(g_j) = \frac{2}{1 + e^{-2g_j}} - 1,$$

$$y = f_2(u_j) + c = \sum_{j=1}^{n_2} v_j u_j + c,$$

Outputs	Inputs
SSRC/TSRC	TCWV, SP, TCO3, FAL, Loc
SSR/TSR	TCWV, SP, TCO3, FAL, TCIW, TCLW, HCC, MCC, LCC, Loc
STRC	SKT, T10, T200, T500, TCWV, Loc
STR	SKT, T10, T200, T500, TCWV, HCC, MCC, LCC, Loc
TTRC	SKT, T10, T200, T500, Q200, Q500, Q700, Loc
TTR	SKT, T10, T200, T500, Q200, Q500, Q700, HCC, MCC, LCC, Loc

The Network is a Function

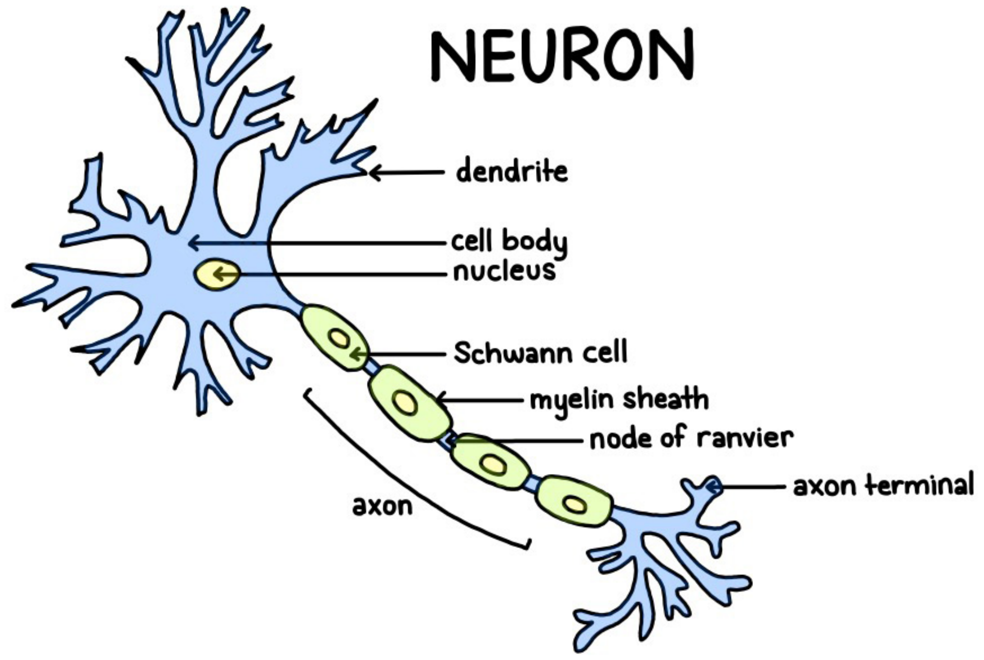
Or the “black box”

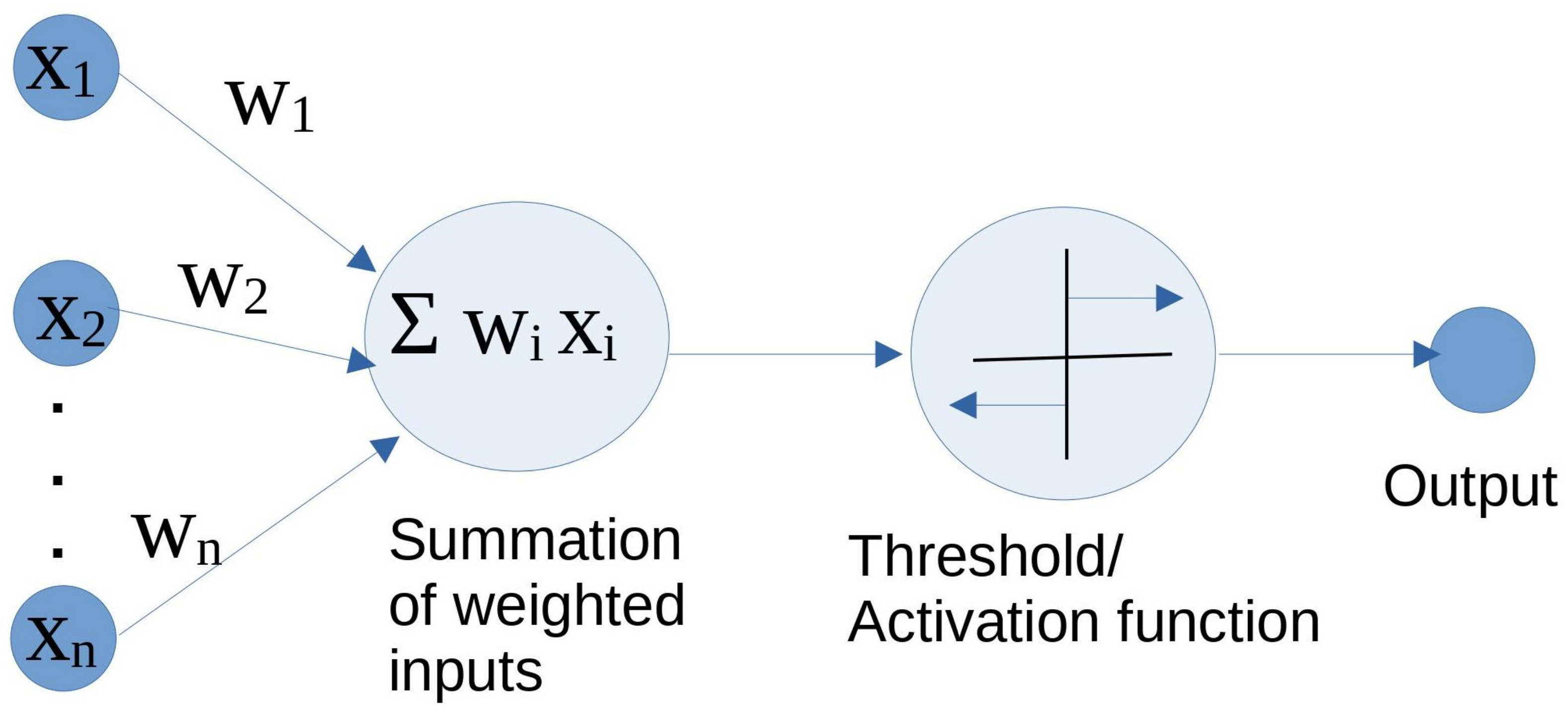


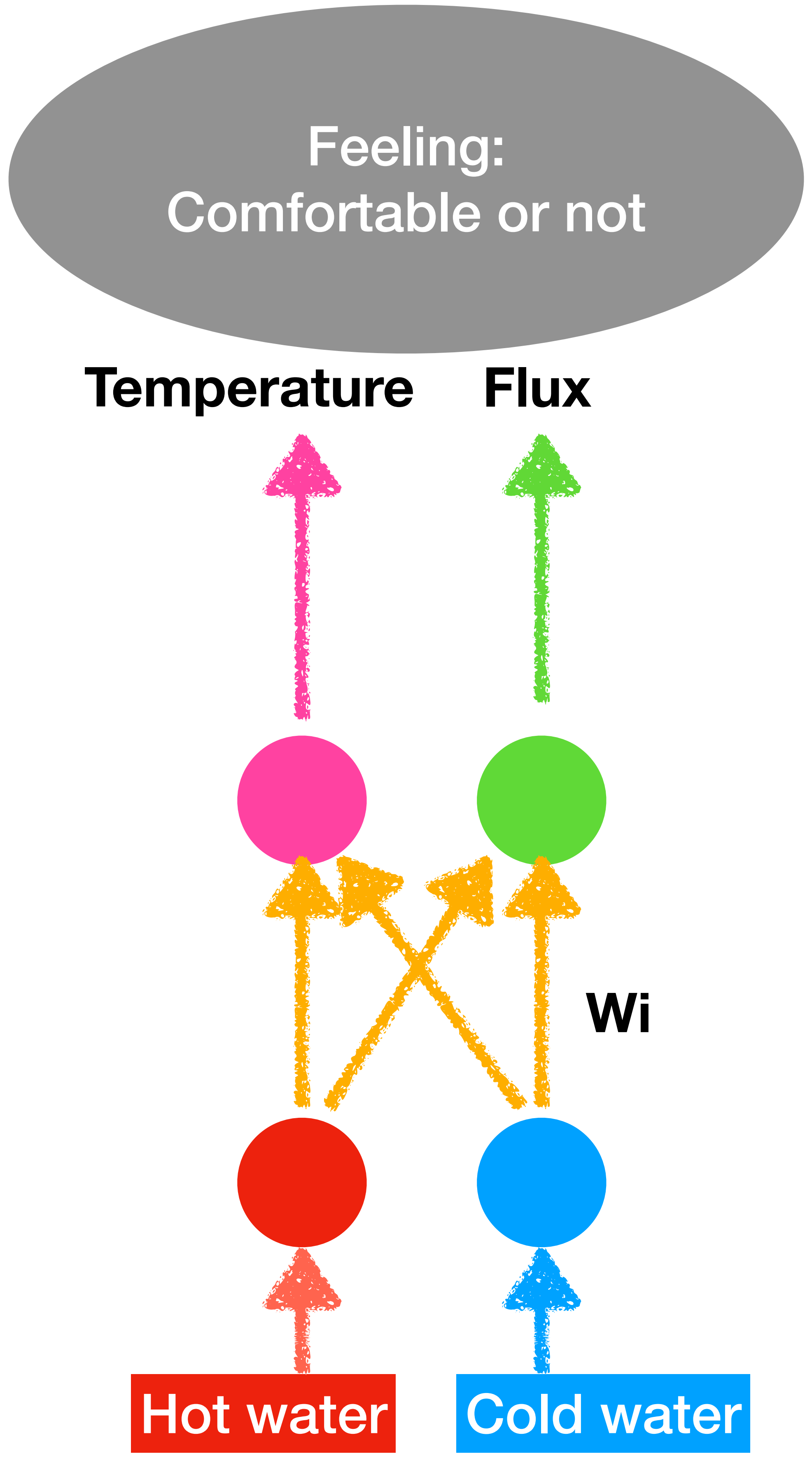
- Radiative kernels:

  f (meteorological variables) = TOA radiation flux

The Basic Unit: Perception

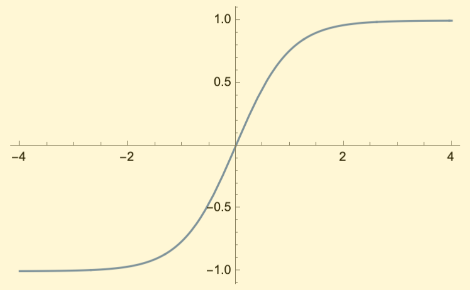






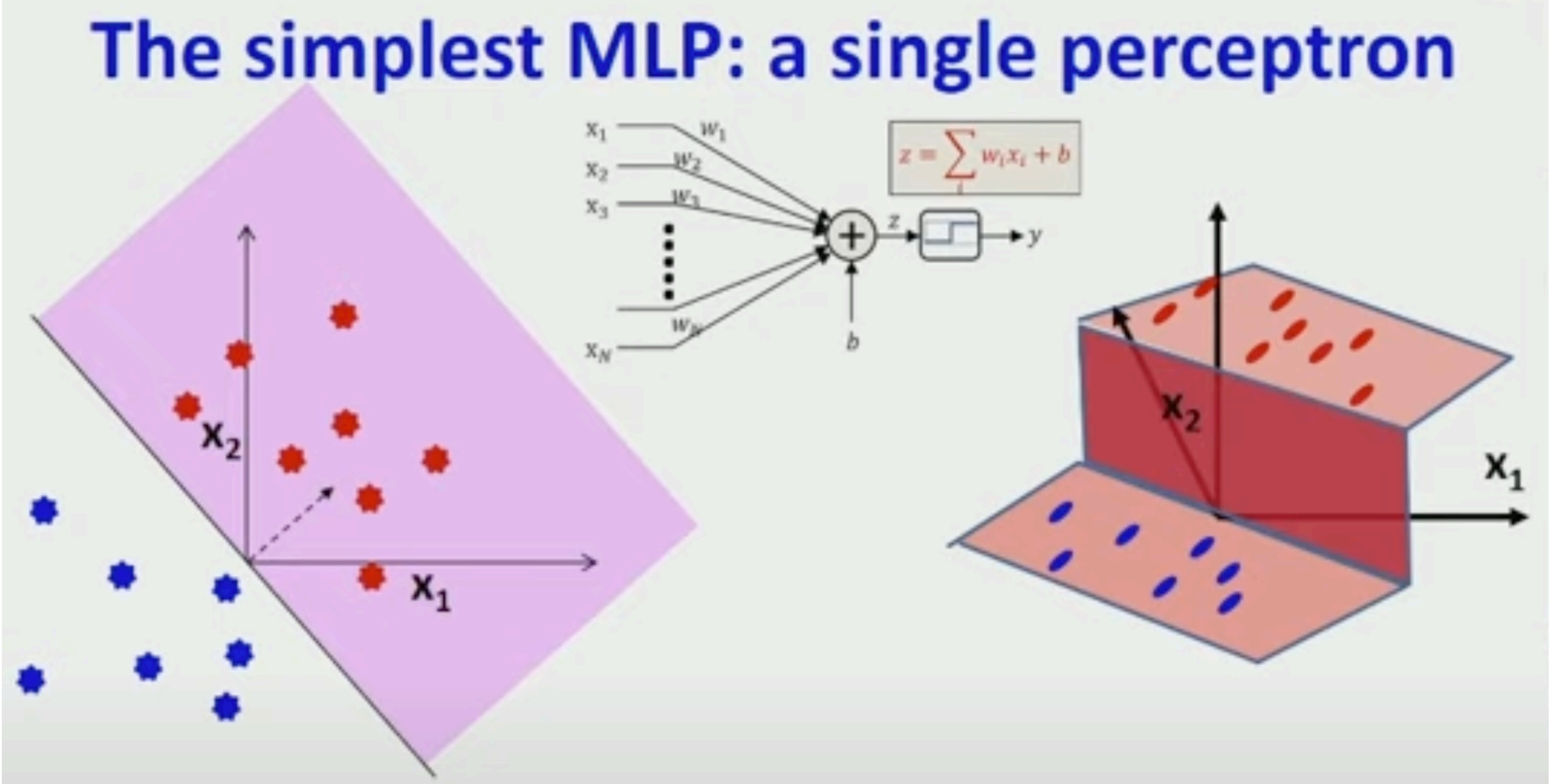
$$g_j = \sum_{i=1}^{n_1} w_{ij} x_i + b_j,$$

$$u_j = f_1\left(g_j\right) = \frac{2}{1+e^{-2g_j}} - 1,$$



- More generalized form:

$$o = \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) = \sigma(\vec{w} \cdot \vec{x} + c)$$

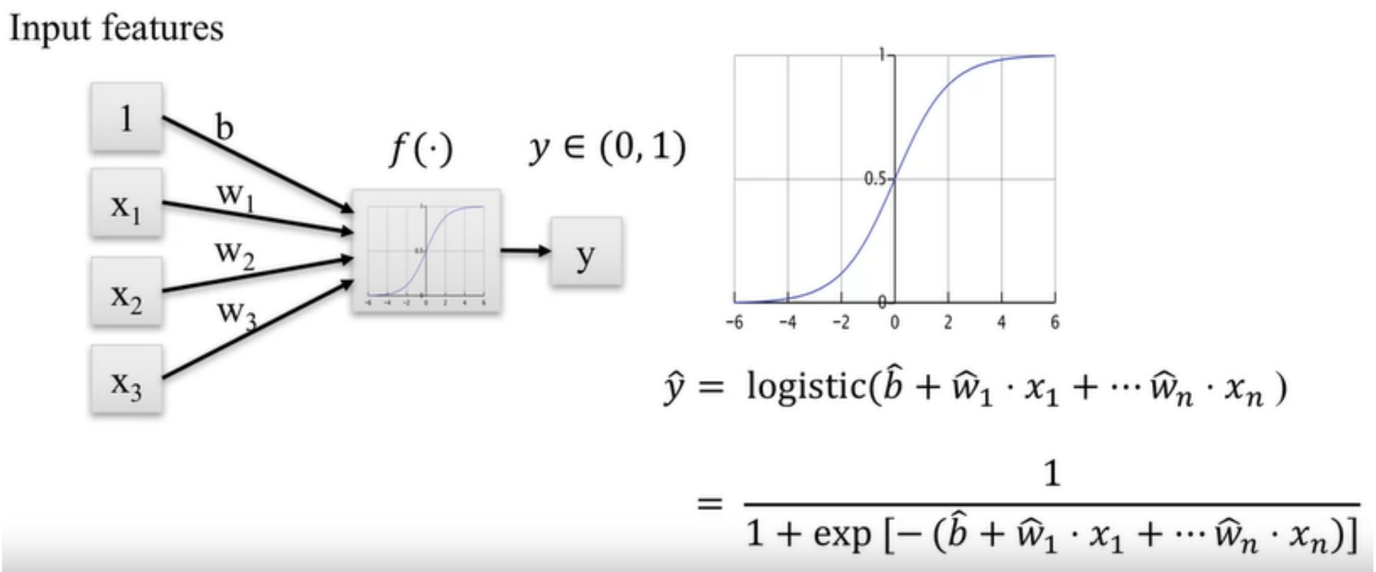


From Neuron to Network

- Linear Regression

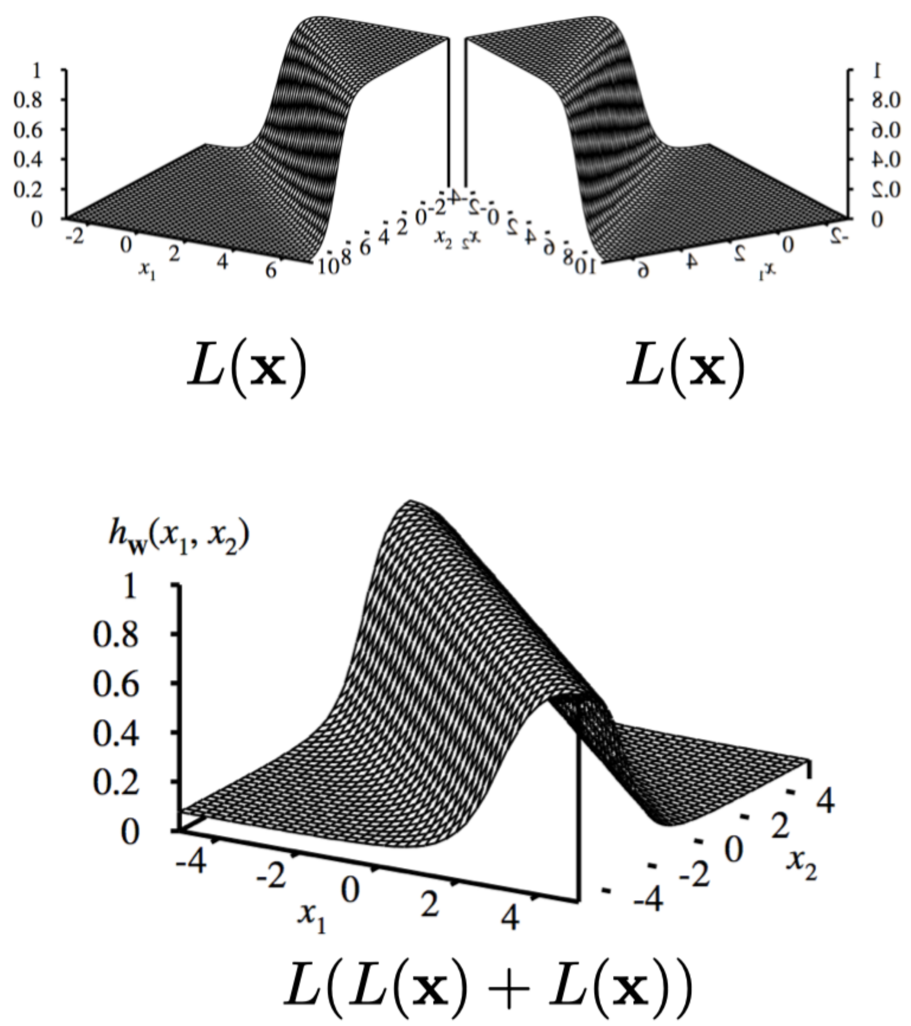
$$y_i = x_i^T \beta + \epsilon_i, \quad i = 1, \cdots, n$$

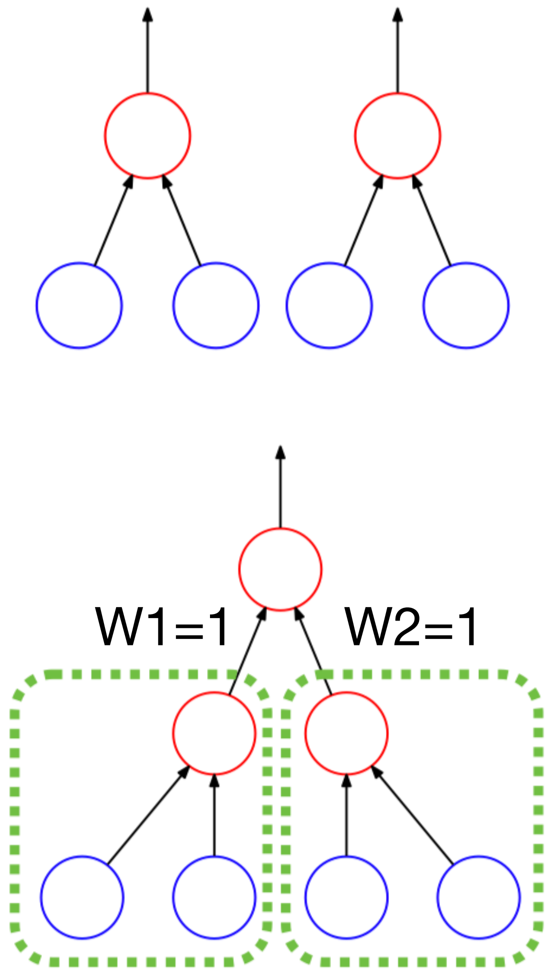
- Logistic Regression



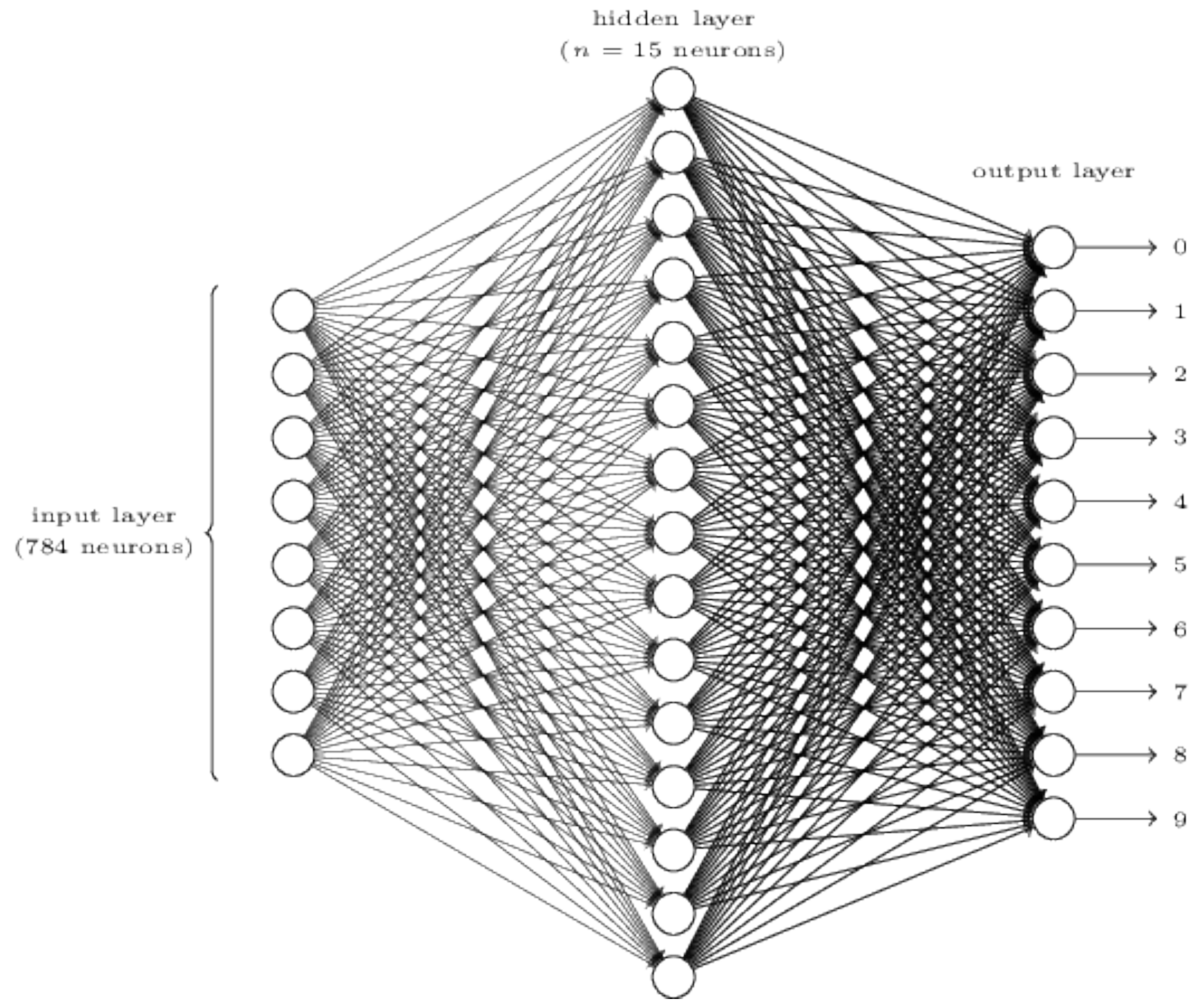
- Combination or composition of perceptrons can represent more functions

$f + g$ or $f \circ g$





- Multi Layer Perceptron



Universal Approximation Theorem

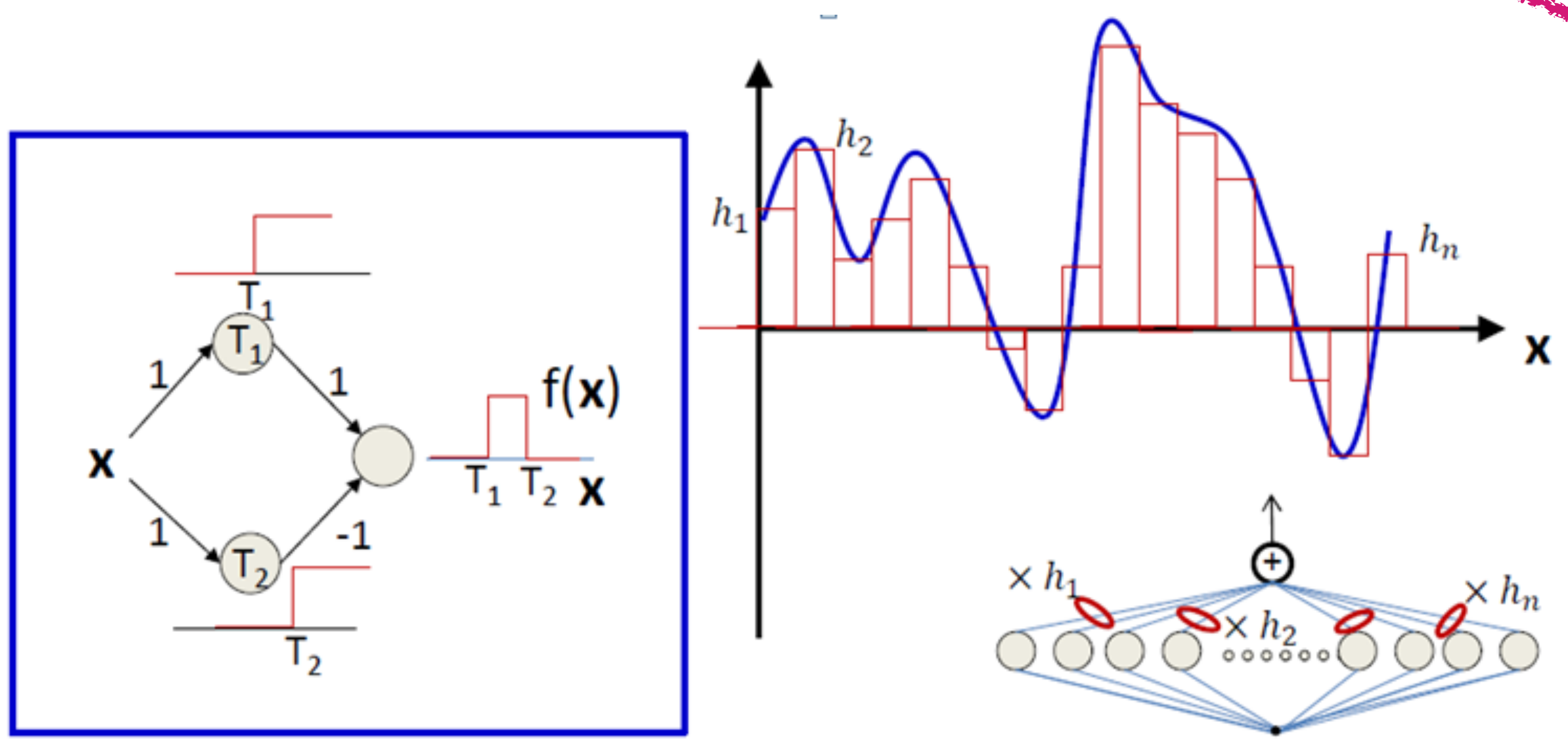
> a three-level MLP with one hidden layer can approximate any bounded continuous function with enough samples, appropriate weights, and suitable number of the hidden nodes

- Arbitrary-width case: sufficient nodes
- Arbitrary-depth case: deep network

- But how many nodes?

Table 1: A summary of known upper/lower bounds on minimum width for universal approximation. In the table, $\mathcal{K} \subset \mathbb{R}^{d_x}$ denotes a compact domain, and $p \in [1, \infty)$. "Conti." is short for continuous.

Reference	Function class	Activation ρ	Upper / lower bounds
Lu et al. (2017)	$L^1(\mathbb{R}^{d_x}, \mathbb{R})$	ReLU	$d_x + 1 \leq w_{\min} \leq d_x + 4$
	$L^1(\mathcal{K}, \mathbb{R})$	ReLU	$w_{\min} \geq d_x$
Hanin and Sellke (2017)	$C(\mathcal{K}, \mathbb{R}^{d_y})$	ReLU	$d_x + 1 \leq w_{\min} \leq d_x + d_y$
Johnson (2019)	$C(\mathcal{K}, \mathbb{R})$	uniformly conti.†	$w_{\min} \geq d_x + 1$
Kidger and Lyons (2020)	$C(\mathcal{K}, \mathbb{R}^{d_y})$	conti. nonpoly‡	$w_{\min} \leq d_x + d_y + 1$
	$C(\mathcal{K}, \mathbb{R}^{d_y})$	nonaffine poly	$w_{\min} \leq d_x + d_y + 2$
	$L^p(\mathbb{R}^{d_x}, \mathbb{R}^{d_y})$	ReLU	$w_{\min} \leq d_x + d_y + 1$
Ours (Theorem 1)	$L^p(\mathbb{R}^{d_x}, \mathbb{R}^{d_y})$	ReLU	$w_{\min} = \max\{d_x + 1, d_y\}$
Ours (Theorem 2)	$C([0,1], \mathbb{R}^2)$	ReLU	$w_{\min} = 3 > \max\{d_x + 1, d_y\}$
Ours (Theorem 3)	$C(\mathcal{K}, \mathbb{R}^{d_y})$	ReLU+Step	$w_{\min} = \max\{d_x + 1, d_y\}$
Ours (Theorem 4)	$L^p(\mathcal{K}, \mathbb{R}^{d_y})$	conti. nonpoly‡	$w_{\min} \leq \max\{d_x + 2, d_y + 1\}$

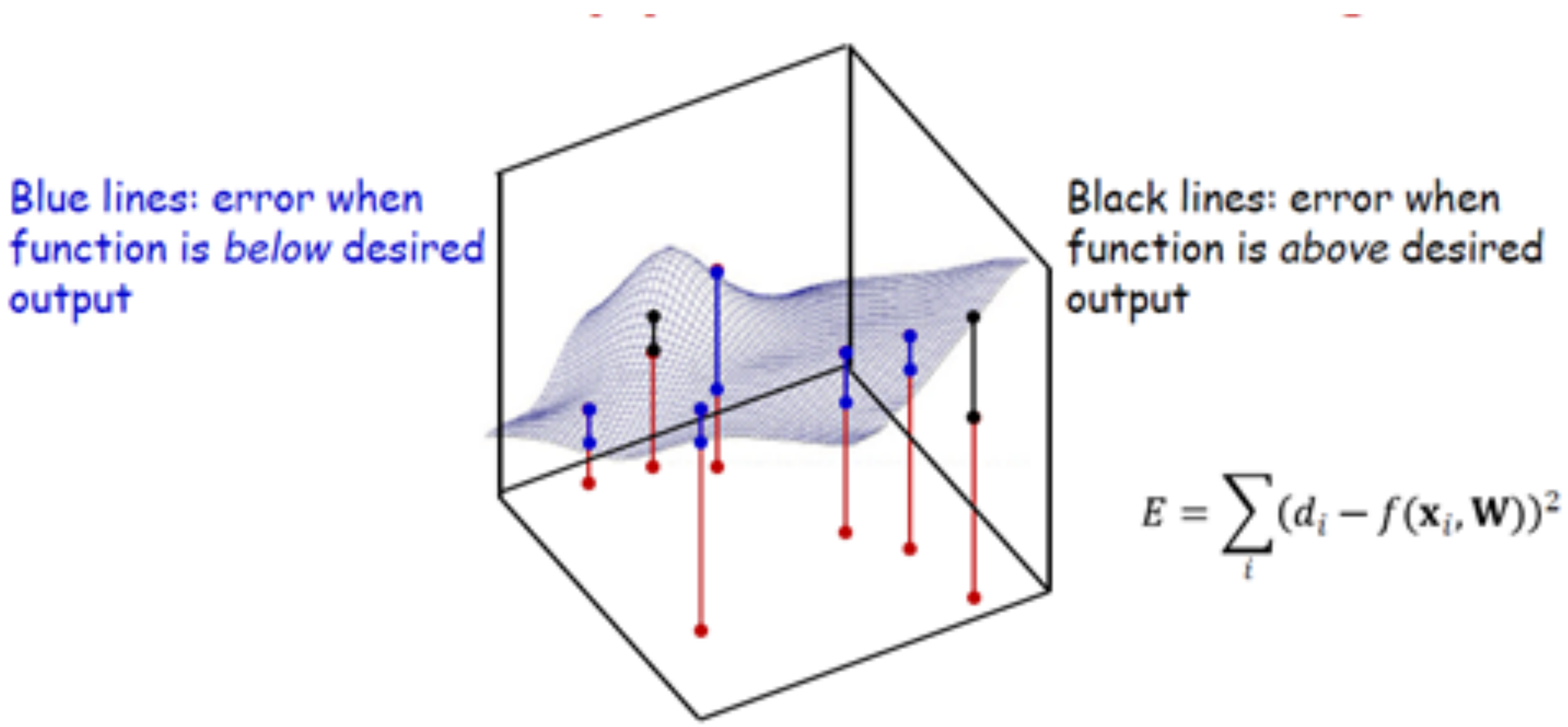


How to determine the weights?

- update the weights whenever the outputs are wrong

Learning the Parameters

Update the weights whenever the outputs are wrong



- Define a *divergence* between the ***actual*** network output for any parameter value and the *desired* output
  - Typically L2 divergence or KL divergence

- Mean Square Error used in this paper

$$E(s) = \frac{1}{N} \times \sum_{k=1}^{N} \left(y_k(s) - y_k^t\right)^2, \quad (7)$$

- Minimize the divergence/"Loss"

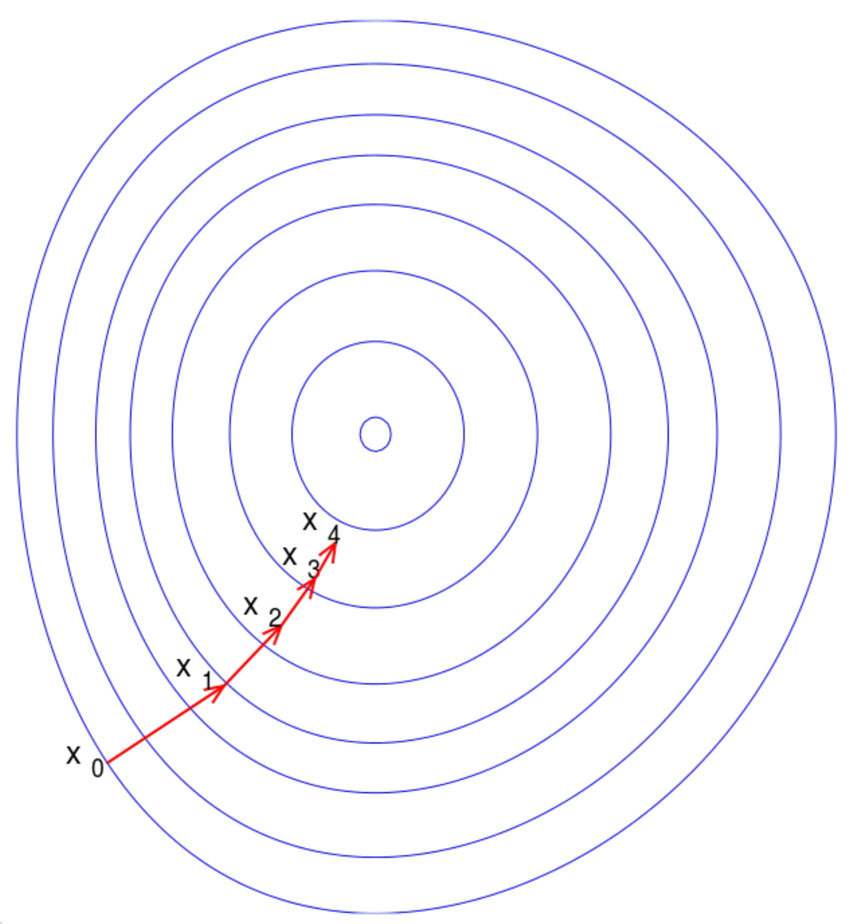
$$p(s+1) = p(s) - l \times \frac{\partial E}{\partial p},$$

- More frequently used form:

$$^{*}W_x = W_x - a \left(\frac{\partial Error}{\partial W_x}\right)$$

Old weight; Derivative of Error with respect to weight; New weight; Learning rate

- Visualization for the 2-dim case

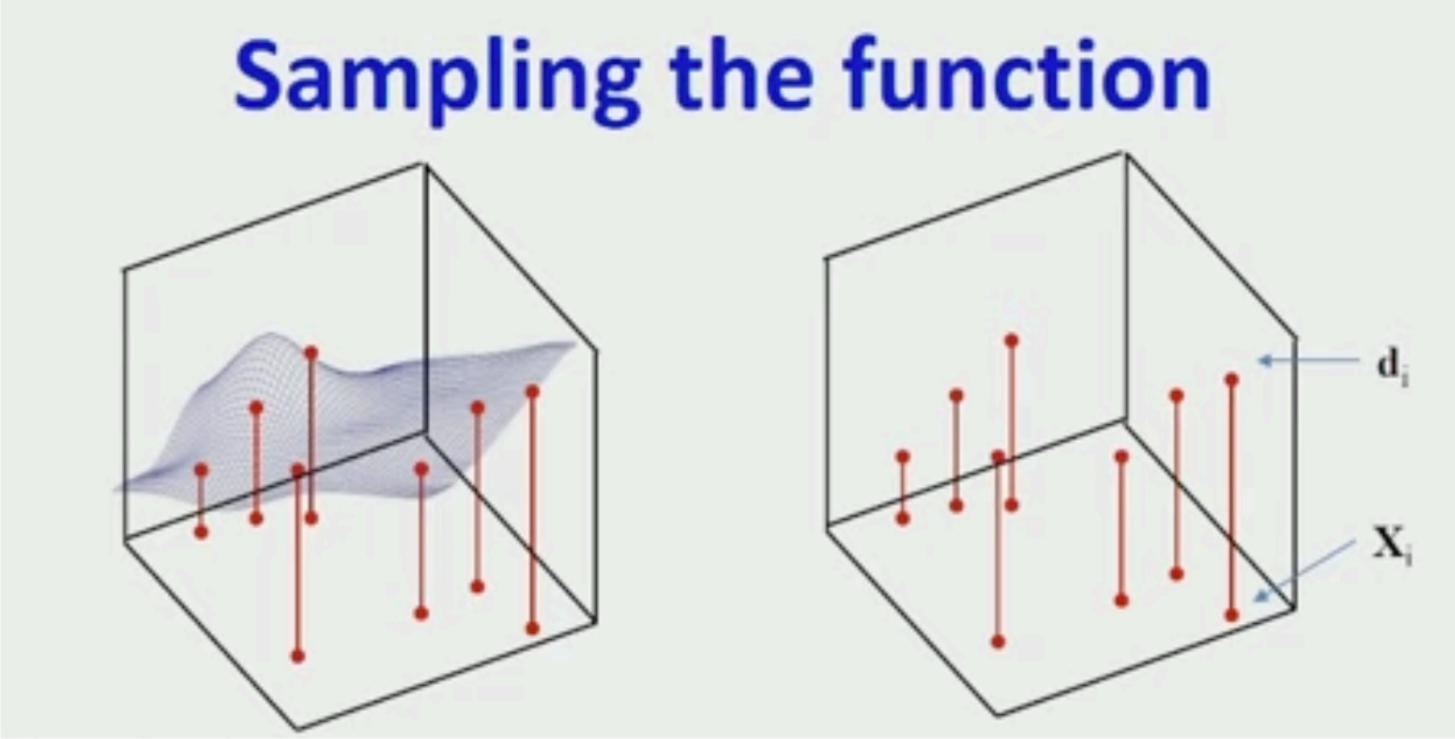


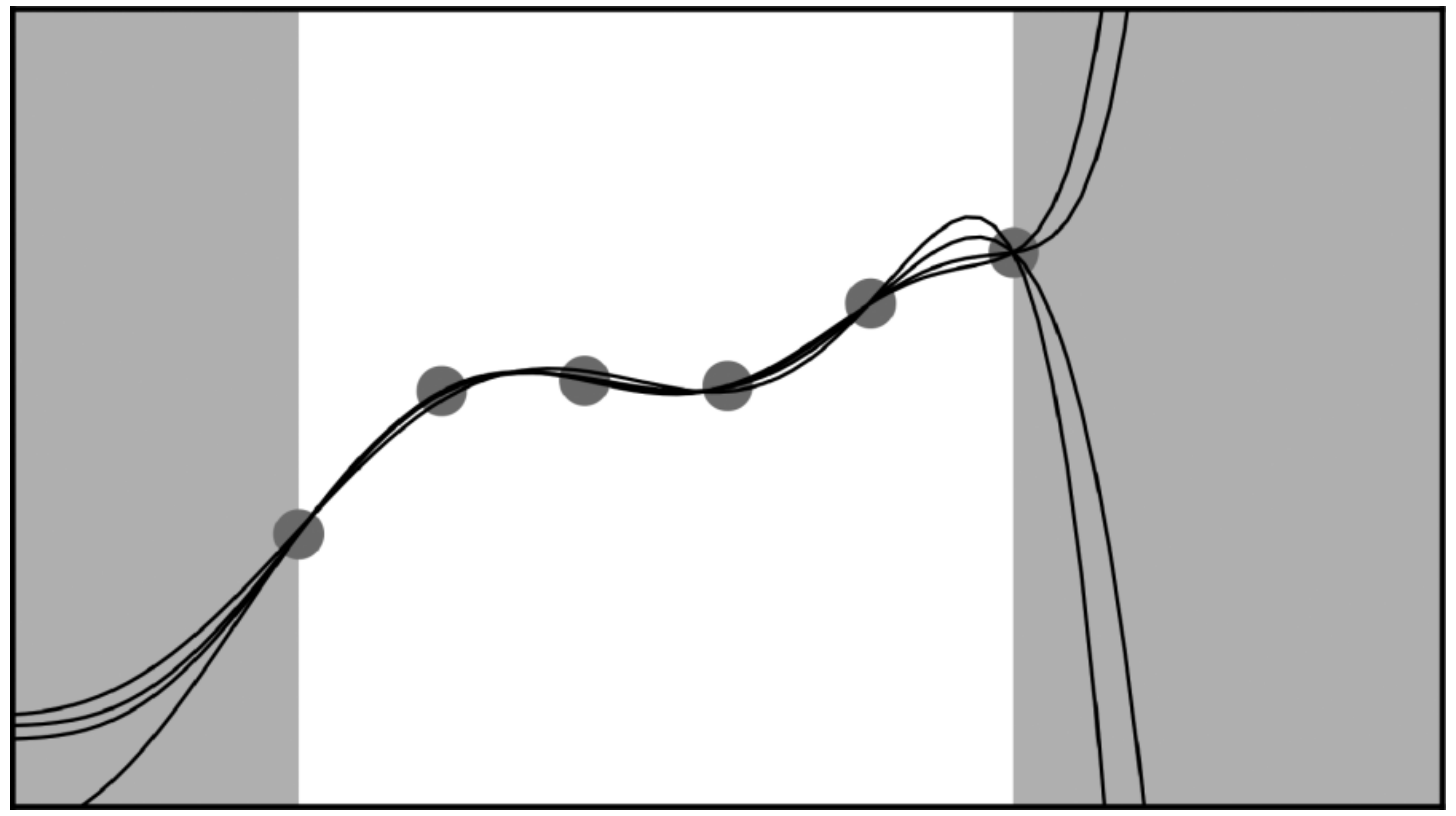
Further Noteworthy

- Data normalization

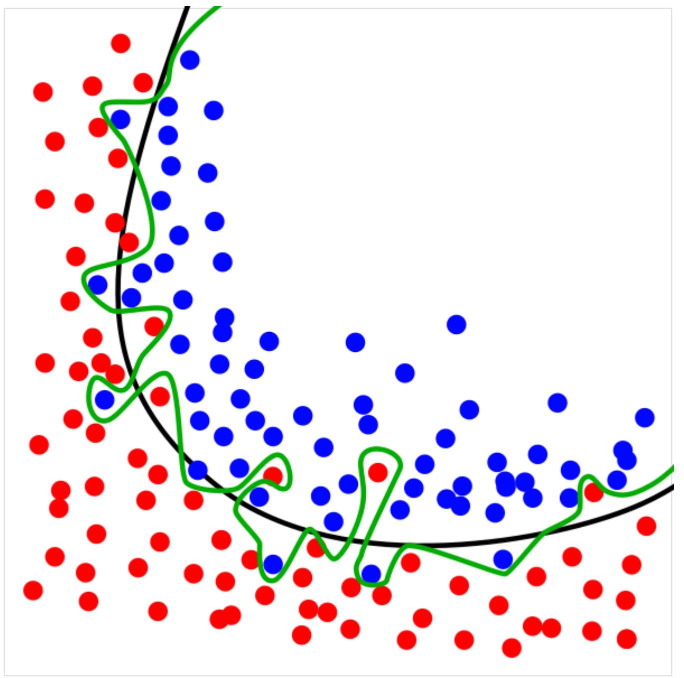
$$X = 2\times\frac{Z-\min(Z)}{\max(Z)-\min(Z)}-1.$$

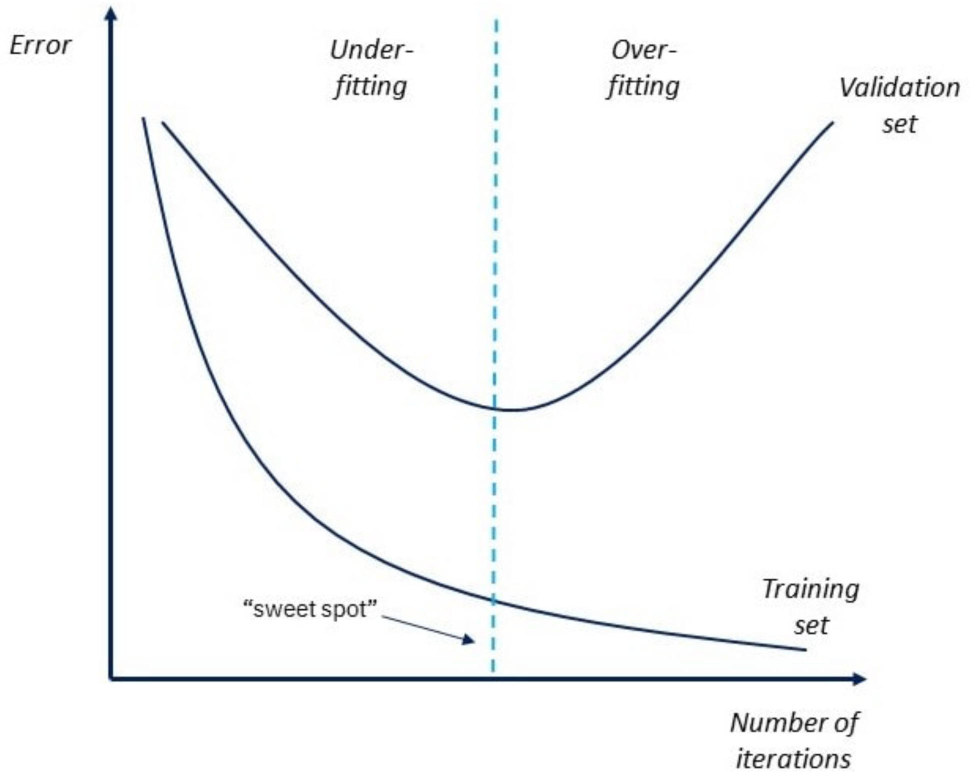
- Sampling problem: size of dataset, extrapolation





- Overfitting and early stop





Thank you