Popular Machine Learning Methods: Idea, Practice and Math

Part 2, Chapter 2, Section 5: Shallow Neural Networks

Yuxiao Huang

Data Science, Columbian College of Arts & Sciences
George Washington University

Fall 2024

Reference

- This set of slices was largely built on the following 7 wonderful books and a wide range of fabulous papers:

  - **HML** Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow (2nd Edition)
  - **PML** Python Machine Learning (3rd Edition)
  - **ESL** The Elements of Statistical Learning (2nd Edition)
  - **PRML** Pattern Recognition and Machine Learning
  - **NND** Neural Network Design (2nd Edition)
  - **LFD** Learning From Data
  - **RL** Reinforcement Learning: An Introduction (2nd Edition)

- For most materials covered in the slides, we will specify their corresponding books and papers for further reference.

Code Example & Case Study

- See related code examples in github repository: /p2_c2_s3_shallow_neural_networks/code_example
- See related case studies of Kaggle Competition in github repository: /p2_c2_s4_shallow_neural_networks/case_study

Table of Contents

Learning Objectives: Expectation

- It is expected to understand
  - the idea and implementation of Single-Layer Perceptron and Perceptron Learning Rule
  - the idea and implementation of Multi-Layer Perceptron and Backpropagation
  - the good practice for using sklearn Single-Layer Perceptron and Multi-Layer Perceptron

Learning Objectives: Recommendation

- It is recommended to understand
  - the math of Single-Layer Perceptron and Perceptron Learning Rule
  - the math of Multi-Layer Perceptron and Backpropagation

The Logical AND Data

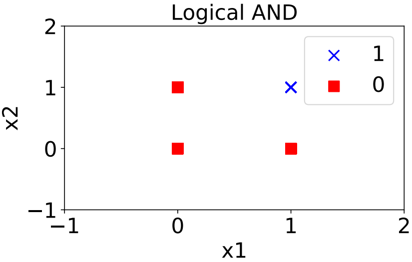

Figure 1: The scatter plot of the Logical AND data summarized in table 1.

Table 1: The Logical AND Data.

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

The Logical OR Data

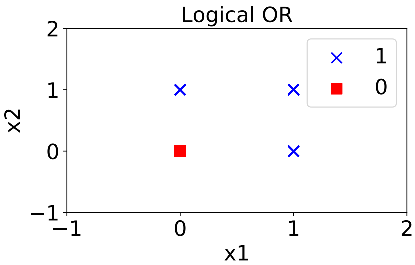

Figure 2: The scatter plot of the Logical OR data summarized in table 2.

Table 2: The Logical OR Data.

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

The Logical XOR Data

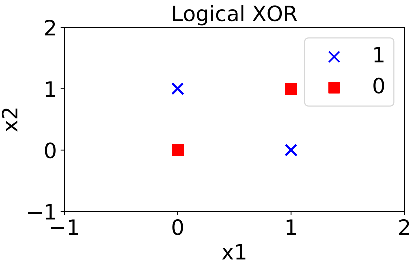

Figure 3: The scatter plot of the Logical XOR data summarized in table 3.

Table 3: The Logical XOR data.

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

Kaggle Competition: Digit Recognizer

Figure 4: Kaggle competition: digit recognizer. Picture courtesy of Kaggle.

- Modified National Institute of Standards and Technology (MNIST) dataset:
  - features: flattened 28×28 (i.e., 784) pixels (taking value in $[0, 255]$) in the image of a digit
  - target: the digit in each image, taking value in $[0, 9]$

Single-Layer Perceptron (SLP)

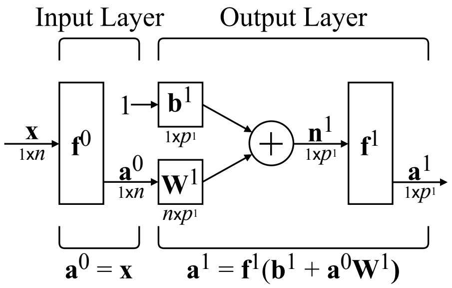

Figure 5: A single-layer perceptron.

- $\mathbf{x}$ is a $1 \times n$ input vector on the input layer (with n being the number of features)
- $\mathbf{f}^0$ is the activation function (*Identity* function) on the input layer:

$$\mathbf{f}(x) = x. \tag{1}$$

- $\mathbf{a}^0$ is a $1 \times n$ output vector on the input layer:

$$\mathbf{a}^0 = \mathbf{f}^0(\mathbf{x}) = \mathbf{x}. \tag{2}$$

- $\mathbf{b}^1$ is a $1 \times p^1$ bias vector on the output layer (with p^1 being the number of perceptrons on the output layer)
- $\mathbf{W}^1 = \left[\mathbf{w}_1^1 \cdots \mathbf{w}_n^1\right]^\intercal$ is a $n \times p^1$ weight matrix on the output layer

Single-Layer Perceptron (SLP)

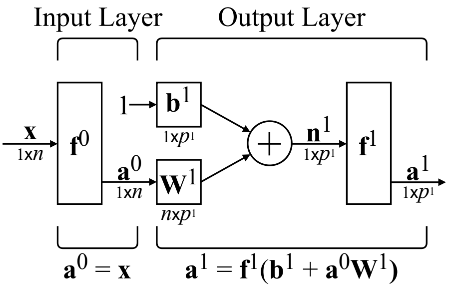

Figure 5: A single-layer perceptron.

- $\mathbf{n}^1$ is a $1 \times p^1$ net input vector on the output layer:

$$\mathbf{n}^1 = \mathbf{b}^1 + \mathbf{a}^0 \mathbf{W}^1. \tag{3}$$

- $\mathbf{f}^1$ is the activation function (e.g., *HardLimit* function) on the output layer:

$$\mathbf{f}(x) = \begin{cases} 1, & \text{if} \quad x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

- $\mathbf{a}^1$ is a $1 \times p^1$ output vector on the output layer:

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{n}^1) = \mathbf{f}^1(\mathbf{b}^1 + \mathbf{a}^0 \mathbf{W}^1). \tag{5}$$

The Logical AND Data

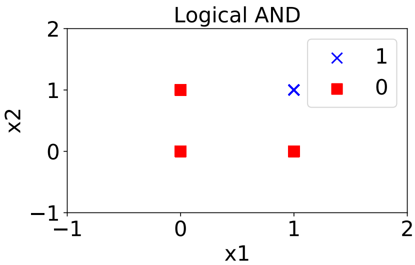

Figure 1: The scatter plot of the Logical AND data summarized in table 1.

Table 1: The Logical AND Data.

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

A Corresponding SLP

- **Q:** Can you design a SLP that perfectly separates the logic AND data?

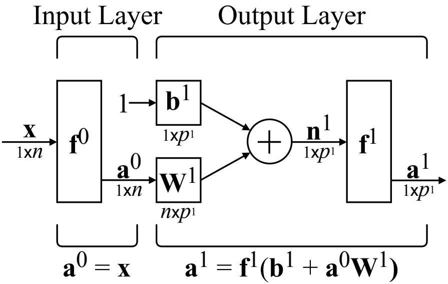

Figure 5: A single-layer perceptron.

A Corresponding SLP

- **Q:** Can you design a SLP that perfectly separates the logic AND data?

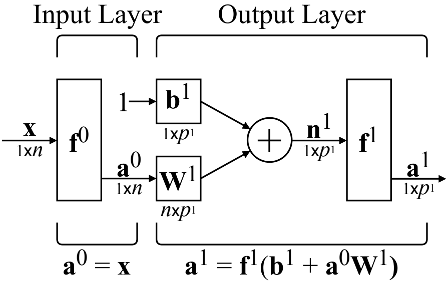

Figure 5: A single-layer perceptron.

- **A:** A SLP in fig. 5 with the following parameter settings:
  - $\mathbf{W}^1 = \begin{bmatrix}1 & 1\end{bmatrix}^\intercal$
  - $\mathbf{b}^1 = -1.5$
  - $\mathbf{f}^1$ is HardLimit function:

$$\mathbf{f}^1(x) = \begin{cases} 1, & \text{if} \quad x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

The Logical OR Data

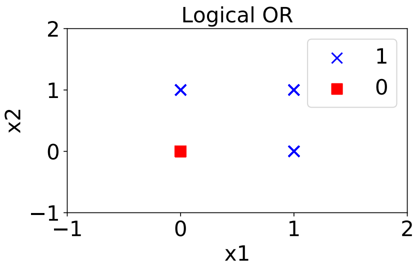

Figure 2: The scatter plot of the Logical OR data summarized in table 2.

Table 2: The Logical OR Data.

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	1
1	0	1
1	1	1

A Corresponding SLP

- **Q:** Can you design a SLP that perfectly separates the logic OR data?

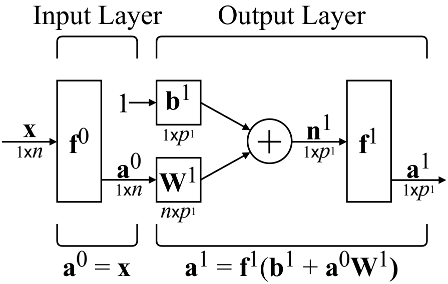

Figure 5: A single-layer perceptron.

A Corresponding SLP

- **Q:** Can you design a SLP that perfectly separates the logic OR data?

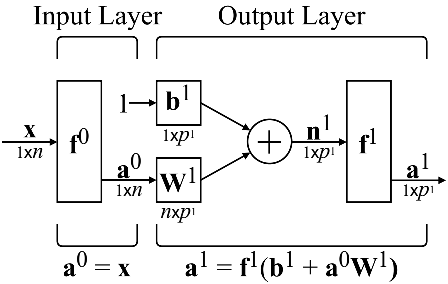

Figure 5: A single-layer perceptron.

- **A:** A SLP in fig. 5 with the following parameter settings:
  - $\mathbf{w}^1 = \begin{bmatrix} 1 & 1 \end{bmatrix}^\intercal$
  - $\mathbf{b}^1 = -0.5$
  - $\mathbf{f}^1$ is HardLimit function:

$$\mathbf{f}^1(x) = \begin{cases} 1, & \text{if} \quad x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Sklearn Single-Layer Perceptron: Code Example

- See /p2_c2_s4_shallow_neural_networks/code_example/:
  1. cells 17 to 23

The Motivation

- As shown in the previous two problems (logic AND and logic OR), designing the parameters of a SLP manually may not be easy.
- Would not it be great if we can estimate the parameters of SLP automatically, as what we did in linear and logistic regression?
- It turns out that we can use the *Perceptron Learning Rule* to do so.

Perceptron Learning Rule: The Idea + Math

- For mathematical convenience, here we use Stochastic Gradient Descent (SGD) to explain the perceptron learning rule.
- For each sample, the parameters of a SLP, $\boldsymbol{\theta}$, are updated using the perceptron learning rule:

$$\boldsymbol{\theta} = \boldsymbol{\theta} + \eta \begin{bmatrix} 1 & \mathbf{x} \end{bmatrix}^\intercal (\mathbf{y} - \mathbf{a}^1). \tag{6}$$

  Here:

  - $\boldsymbol{\theta}$ is a $(n+1) \times p_1$ parameter matrix (with n being the number of features):

$$\boldsymbol{\theta} = \begin{bmatrix} \mathbf{b}^1 & \mathbf{w}_1^1 \cdots \mathbf{w}_n^1 \end{bmatrix}^\intercal \tag{7}$$

  - $\mathbf{b}^1$ is a $1 \times p^1$ bias vector on the output layer (with p^1 being the number of perceptrons on the output layer)
  - $\mathbf{W}^1 = \begin{bmatrix} \mathbf{w}_1^1 \cdots \mathbf{w}_n^1 \end{bmatrix}^\intercal$ is a $n \times p^1$ weight matrix on the output layer
  - $\mathbf{x}$ is the $1 \times n$ feature vector of the sample
  - $\mathbf{y}$ is the $1 \times p_1$ real target vector of the sample
  - $\mathbf{a}^1$ is the $1 \times p_1$ output vector on the output layer, given in eq. (5)

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{n}^1) = \mathbf{f}^1(\mathbf{b}^1 + \mathbf{a}^0\mathbf{W}^1) \tag{5}$$

SLP: The Implementation

- See /models/p2_shallow_learning:
  1. cell 6

Further Reading

- See NND: Chap 4 for a very nice explanation of why the perceptron learning rule works, from a linear algebra perspective.
- See NND: Chap 4 for the proof of convergence for the perceptron learning rule (i.e., the proof that SLP can always perfectly separate linearly-separable data).

The Logical XOR Data

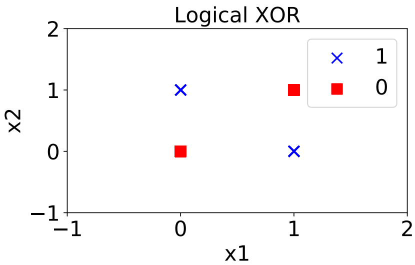

Figure 3: The scatter plot of the Logical XOR data summarized in table 3.

Table 3: The Logical XOR data.

x_1	x_2	$x_1 \oplus x_2$
0	0	0
0	1	1
1	0	1
1	1	0

A Corresponding SLP

- **Q:** Can you design a SLP that perfectly separates the logic XOR data?

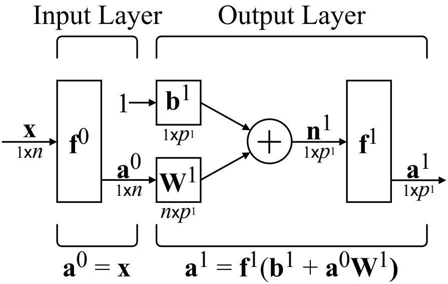

Figure 5: A single-layer perceptron.

.

A Corresponding SLP

- **Q:** Can you design a SLP that perfectly separates the logic XOR data?

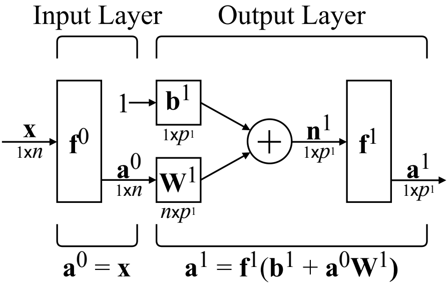

Figure 5: A single-layer perceptron.

- **A:** Unfortunately, no, since the data is not linearly-separable.

Multi-Layer Perceptron (MLP): Input Layer

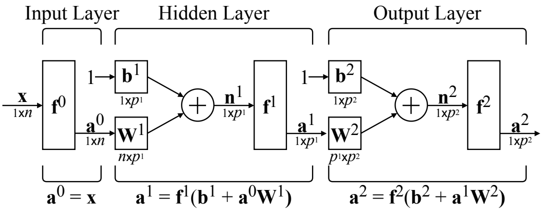

Figure 6: A fully connected three-layer MLP.

- $\mathbf{x}$ is a $1 \times n$ input vector on the input layer (with n being the number of features)
- $\mathbf{f}^0$ is the activation function (*Identity* function) on the input layer:

$$\mathbf{f}(x) = x. \tag{8}$$

- $\mathbf{a}^0$ is a $1 \times n$ output vector on the input layer, given in eq. (2):

$$\mathbf{a}^0 = \mathbf{f}^0(\mathbf{x}) = \mathbf{x}. \tag{2}$$

Multi-Layer Perceptron (MLP): Hidden Layer

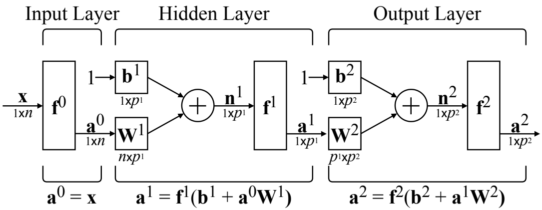

Figure 6: A fully connected three-layer MLP.

- $\mathbf{b}^1$ is a $1 \times p^1$ bias vector on the hidden layer (with p^1 being the number of perceptrons on the hidden layer)
- $\mathbf{W}^1 = \left[\mathbf{w}_1^1 \cdots \mathbf{w}_n^1\right]^\intercal$ is a $n \times p^1$ weight matrix on the hidden layer
- $\mathbf{n}^1$ is a $1 \times p^1$ net input vector on the hidden layer, given in eq. (3):

$$\mathbf{n}^1 = \mathbf{b}^1 + \mathbf{a}^0\mathbf{W}^1. \tag{3}$$

- $\mathbf{f}^1$ is the activation function (e.g., *HardLimit* function) on the hidden layer, given in eq. (4):

$$\mathbf{f}(x) = \begin{cases} 1, & \text{if} \quad x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

- $\mathbf{a}^1$ is a $1 \times p^1$ output vector on the hidden layer, given in eq. (5):

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{n}^1) = \mathbf{f}^1(\mathbf{b}^1 + \mathbf{a}^0\mathbf{W}^1). \tag{5}$$

Multi-Layer Perceptron (MLP): Output Layer

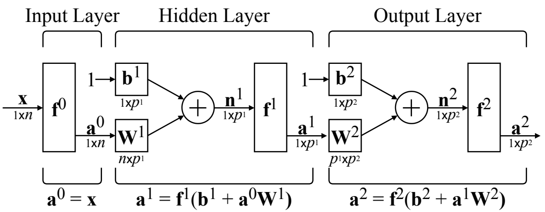

Figure 6: A fully connected three-layer MLP.

- $\mathbf{b}^2$ is a $1 \times p^2$ bias vector on the output layer (with p^2 being the number of perceptrons on the output layer)
- $\mathbf{W}^2 = \left[\mathbf{w}_1^2 \cdots \mathbf{w}_{p^1}^2\right]^\intercal$ is a $p^1 \times p^2$ weight matrix on the hidden layer
- $\mathbf{n}^1$ is a $1 \times p^1$ net input vector on the hidden layer:
$$\mathbf{n}^2 = \mathbf{b}^2 + \mathbf{a}^1\mathbf{W}^2. \tag{9}$$
- $\mathbf{f}^2$ is the activation function (e.g., *HardLimit* function) on the output layer, given in eq. (4):
$$\mathbf{f}(x) = \begin{cases} 1, & \text{if} \quad x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$
- $\mathbf{a}^2$ is a $1 \times p^2$ output vector on the output layer:
$$\mathbf{a}^2 = \mathbf{f}^2(\mathbf{n}^2) = \mathbf{f}^2(\mathbf{b}^2 + \mathbf{a}^1\mathbf{W}^2). \tag{10}$$

A MLP for the Logical XOR Data

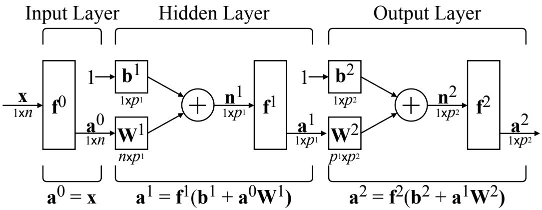

Figure 6: A fully connected three-layer MLP.

- A MLP in fig. 6 with the following parameter settings:
  - $\mathbf{W}^1 = \begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$, $\mathbf{b}^1 = \begin{bmatrix} -0.5 & -0.5 \end{bmatrix}$
  - $\mathbf{W}^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}^\intercal$, $\mathbf{b}^2 = -0.5$
  - $\mathbf{f}^1$ and $\mathbf{f}^2$ are the Hardlimit function, given in eq. (4):

$$\mathbf{f}(x) = \begin{cases} 1, & \text{if} \quad x \geq 0, \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

MLP with Multiple Hidden Layers

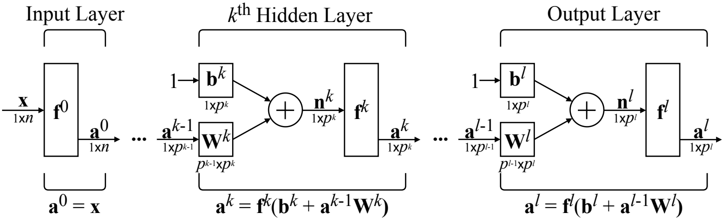

Figure 7: A fully connected MLP with $l-1$ hidden layers.

- $\mathbf{x}$ is a $1 \times n$ input vector on the input layer (with n being the number of features).
- $\mathbf{a}^0$ is a $1 \times n$ output vector on the input layer.
- $\mathbf{a}^{k-1}$ is the $1 \times p^{k-1}$ input vector on the $k^{\mathbf{th}}$ hidden layer (with p^{k-1} being the number of perceptrons on the $(k-1)^{\mathbf{th}}$ hidden layer).
- $\mathbf{a}^k$ is the $1 \times p^k$ output vector on the $k^{\mathbf{th}}$ hidden layer (with p^k being the number of perceptrons on the $k^{\mathbf{th}}$ hidden layer).
- $\mathbf{a}^{l-1}$ is the $1 \times p^{l-1}$ input vector on the output layer (with p^{l-1} being the number of perceptrons on the last hidden layer).
- $\mathbf{a}^l$ is the $1 \times p^l$ output vector on the output layer (with p^l being the number of perceptrons on the output layer).

Sklearn Multi-Layer Perceptron: Code Example

- See /p2_c2_s4_shallow_neural_networks/code_example/:
  1. cells 34 to 37

The Motivation

- Since manually designing the parameters of a SLP is not easy, doing so for a MLP is much more difficult, if not impossible.
- Unfortunately, we cannot simply use perceptron learning rule to automatically train a MLP (we will see the reason later).
- Instead, we can use the *Backpropagation* to do so.

Backpropagation: The Idea

- Backpropagation is an iterative process.
- There are three steps (in order) in each iteration:
  1. *Forward Pass*: sends information from input layer up to higher layers
  2. *Backward Pass*: sends information from output layer down to lower layers
  3. *Gradient Descent*: updates the parameters using information obtained in the forward and backward pass

Forward Pass: The Idea

- Forward pass works as follows:
  1. passes the input to the input layer
  2. from the first hidden layer up to the output layer, calculates the output of the current layer (e.g., layer k) from the output of the previous layer (e.g., layer $k - 1$)
- In forward pass, the net input and activation of each layer are preserved and reused in backward pass and gradient descent.

Backward Pass: The Idea

- Backward pass works as follows:
  1. calculates the loss function (e.g., Mean Squared Error) based on the target and the output of the output layer
  2. calculates the *Sensitivity* (more on this later) of the output layer
  3. from the last hidden layer down to the first hidden layer, calculates the sensitivity of the current layer (e.g., layer k) from the sensitivity of the next layer (e.g., layer $k + 1$)
- In backward pass, the sensitivity of each layer is preserved and reused in gradient descent

Gradient Descent: The Idea

- Gradient descent works as follows:
  1. uses the output (obtained in forward pass) and sensitivity (obtained in backward pass) of each layer to update the parameters of the layer

Backpropagation: The Math

- Let us take a look at the math underlying the three steps in backpropagation (forward pass, backward pass and gradient descent)
- The order in which the steps work:
  1. forward pass
  2. backward pass
  3. gradient descent
- The order in which we will discuss the steps (which better explains the dependence between the steps):
  1. gradient descent
  2. forward pass
  3. backward pass

The Loss Function

- Similar to what we did for perceptron learning rule, for mathematical convenience we will also use stochastic gradient descent to explain backpropagation.
- Since stochastic gradient descent processes one sample at a time, the loss function (Mean Squared Error), $\mathcal{L}(\boldsymbol{\theta})$, can be written as

$$\mathcal{L}(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{a}^{l})^2. \tag{11}$$

Here:

- $\mathbf{y}$ is the $1 \times p^l$ target vector of the sample (with p^l being the number of perceptrons on the output layer)
- $\mathbf{a}^l$ is the $1 \times p^l$ output vector on the output layer:

$$\mathbf{a}^l = \mathbf{f}^l(\mathbf{n}^l) = \mathbf{f}^l(\mathbf{b}^l + \mathbf{a}^{l-1}\mathbf{W}^l) = \mathbf{f}^l\left(\begin{bmatrix}1 & \mathbf{a}^{l-1}\end{bmatrix}\boldsymbol{\theta}^l\right) \tag{12}$$

- $\mathbf{b}^l$ is a $1 \times p^l$ bias vector on the output layer
- $\mathbf{W}^l = \begin{bmatrix}\mathbf{w}_1^l \cdots \mathbf{w}_{p^{l-1}}^l\end{bmatrix}^\intercal$ is a $p^{l-1} \times p^l$ weight matrix on the output layer
- $\boldsymbol{\theta}^l$ is the $(p^{l-1} + 1) \times p^l$ parameter vector on the output layer:

$$\boldsymbol{\theta}^l = \begin{bmatrix}\mathbf{b}^l & \mathbf{w}_1^l \cdots \mathbf{w}_{p^{l-1}}^l\end{bmatrix}^\intercal \tag{13}$$

Gradient Descent: The Math

- Using gradient descent, the parameters of layer k, $\boldsymbol{\theta}^k$, can be updated as
$$\boldsymbol{\theta}^k = \boldsymbol{\theta}^k - \eta \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^k}. \tag{14}$$
- We cannot directly calculate the gradient in eq. (14) since, as shown in eqs. (11) and (12), while $\mathcal{L}(\boldsymbol{\theta})$ is an explicit function of parameters of the output layer, $\boldsymbol{\theta}^l$, it is not an explicit function of parameters of the hidden layers, $\boldsymbol{\theta}^k$ (where $k < l$).
- This is why we cannot use the perceptron learning rule for MLP, as we did for SLP (where there is no hidden layer).
- This is also why we use the chain rule to rewrite eq. (14):
$$\boldsymbol{\theta}^k = \boldsymbol{\theta}^k - \eta \frac{\partial \mathbf{n}^k}{\partial \boldsymbol{\theta}^k} \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^k}, \tag{15}$$
where $\mathbf{n}^k$ is the net input on layer k (and an explicit function of $\boldsymbol{\theta}^k$):
$$\mathbf{n}^k = \mathbf{b}^k + \mathbf{a}^{k-1}\mathbf{W}^k = \begin{bmatrix} 1 & \mathbf{a}^{k-1} \end{bmatrix} \boldsymbol{\theta}^k. \tag{16}$$

Gradient Descent: The Math

- Based on eq. (16)

$$\mathbf{n}^k = \mathbf{b}^k + \mathbf{a}^{k-1}\mathbf{W}^k = \begin{bmatrix}1 & \mathbf{a}^{k-1}\end{bmatrix} \boldsymbol{\theta}^k. \tag{16}$$

we rewrite the first derivation in eq. (15), $\frac{\partial \mathbf{n}^k}{\partial \boldsymbol{\theta}^k}$, as

$$\frac{\partial \mathbf{n}^k}{\partial \boldsymbol{\theta}^k} = \frac{\partial \left(\begin{bmatrix}1 & \mathbf{a}^{k-1}\end{bmatrix} \boldsymbol{\theta}^k\right)}{\partial \boldsymbol{\theta}^k} = \begin{bmatrix}1 & \mathbf{a}^{k-1}\end{bmatrix}^\intercal. \tag{17}$$

- We define the second derivation in eq. (15), $\frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^k}$, as the *Sensitivity* of $\mathcal{L}(\boldsymbol{\theta})$ to the changes of $\mathbf{n}^k$, denoted by $\mathbf{s}^k$:

$$\mathbf{s}^k = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^k}. \tag{18}$$

Gradient Descent: The Math

- By substituting eqs. (17) and (18)

$$\frac{\partial \mathbf{n}^k}{\partial \boldsymbol{\theta}^k} = \frac{\partial \left(\begin{bmatrix} 1 & \mathbf{a}^{k-1} \end{bmatrix} \boldsymbol{\theta}^k \right)}{\partial \boldsymbol{\theta}^k} = \begin{bmatrix} 1 & \mathbf{a}^{k-1} \end{bmatrix}^\intercal \quad \text{and} \quad \mathbf{s}^k = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^k} \tag{17,18}$$

into eq. (15)

$$\boldsymbol{\theta}^k = \boldsymbol{\theta}^k - \eta \frac{\partial \mathbf{n}^k}{\partial \boldsymbol{\theta}^k} \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^k}, \tag{15}$$

we can update the parameters as

$$\boldsymbol{\theta}^k = \boldsymbol{\theta}^k - \eta \begin{bmatrix} 1 & \mathbf{a}^{k-1} \end{bmatrix}^\intercal \mathbf{s}^k. \tag{19}$$

- We use forward pass to calculate $\mathbf{a}^{k-1}$ in eq. (19).
- We use backward pass to calculate $\mathbf{s}^k$ in eq. (19).

Gradient Descent: The Idea + Math

- Gradient descent works as follows:
  1. uses eq. (19), the output (obtained in forward pass) and sensitivity (obtained in backward pass) of each layer to update the parameters of the layer:
  $$\boldsymbol{\theta}^k = \boldsymbol{\theta}^k - \eta \begin{bmatrix} 1 & \mathbf{a}^{k-1} \end{bmatrix}^\intercal \mathbf{s}^k. \tag{19}$$

Forward Pass: The Math

- The key in forward pass is a recursive relationship between the output of two consecutive layers.
- Since the output on layer k, $\mathbf{a}^k$, is

$$\mathbf{a}^k = \mathbf{f}^k(\mathbf{n}^k), \tag{20}$$

  where $\mathbf{f}^k$ is the activation function on layer k and $\mathbf{n}^k$ the net input on layer k, given in eq. (16)

$$\mathbf{n}^k = \mathbf{b}^k + \mathbf{a}^{k-1}\mathbf{W}^k = \begin{bmatrix} 1 & \mathbf{a}^{k-1} \end{bmatrix} \boldsymbol{\theta}^k. \tag{16}$$

  we can rewrite eq. (20) as

$$\mathbf{a}^k = \mathbf{f}^k(\mathbf{n}^k) = \mathbf{f}^k(\mathbf{b}^k + \mathbf{a}^{k-1}\mathbf{W}^k) = \mathbf{f}^k(\begin{bmatrix} 1 & \mathbf{a}^{k-1} \end{bmatrix} \boldsymbol{\theta}^k). \tag{21}$$

- Eq. (21) shows that the output on layer k, $\mathbf{a}^k$, relies on the output on layer $k-1$, $\mathbf{a}^{k-1}$.
- This is why we use *forward* pass to calculate the output from lower layers (e.g., layer $k-1$) up to higher layers (e.g., layer k).

Forward Pass: The Idea + Math

- Forward pass works as follows:
  1. passes the input to the input layer, given in eq. (2):

$$\mathbf{a}^0 = \mathbf{f}^0(\mathbf{x}) = \mathbf{x} \tag{2}$$

  2. from the first hidden layer (layer 1) up to the output layer (layer l), uses eq. (21) to calculate the output of the current layer (e.g., layer k) from the output of the previous layer (e.g., layer $k-1$):

$$\mathbf{a}^k = \mathbf{f}^k(\mathbf{n}^k) = \mathbf{f}^k(\begin{bmatrix}1 & \mathbf{a}^{k-1}\end{bmatrix}\boldsymbol{\theta}^k), \quad \text{where} \quad 1 \leq k \leq l \tag{21}$$

Backward Pass: The Math

- Similar to forward pass, the key in backward pass is also a recursive relationship.
- Unlike forward pass where the relationship is between output of two consecutive layers, the relationship in backward pass is between sensitivity of two consecutive layers.
- Concretely, by applying the chain rule to eq. (18)

$$\mathbf{s}^k = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^k}, \tag{18}$$

  we have

$$\mathbf{s}^k = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^k} = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^{k+1}} \frac{\partial \mathbf{n}^{k+1}}{\partial \mathbf{n}^k} = \mathbf{s}^{k+1} \frac{\partial \mathbf{n}^{k+1}}{\partial \mathbf{n}^k}. \tag{22}$$

- Eq. (22) shows that the sensitivity of layer k, $\mathbf{s}^k$, relies on the sensitivity of layer $k+1$, $\mathbf{s}^{k+1}$.
- This is why we use *backward* pass to calculate the sensitivity from higher layers (e.g., layer $k+1$) down to lower layers (e.g., layer k).

The Sensitivity

- Based on eq. (18)

$$\mathbf{s}^k = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^k}, \tag{18}$$

the sensitivity of the output layer (with index l), $\mathbf{s}^l$, is

$$\mathbf{s}^l = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^l}. \tag{23}$$

- By substituting eqs. (11) and (12)

$$\mathcal{L}(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{a}^l)^2 \quad \text{and} \quad \mathbf{a}^l = \mathbf{f}^l(\mathbf{n}^l) \tag{11,12}$$

into eq. (23), we have

$$\mathbf{s}^l = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^l} = \frac{\partial (\mathbf{y} - \mathbf{a}^l)^2}{\partial \mathbf{n}^l} = -2(\mathbf{y} - \mathbf{a}^l)\frac{\partial \mathbf{f}^l(\mathbf{n}^l)}{\partial \mathbf{n}^l} = -2(\mathbf{y} - \mathbf{a}^l)\dot{\mathbf{f}}^l(\mathbf{n}^l). \tag{24}$$

The Jacobian Matrix

- The last piece in backward pass is the derivation in eq. (22)

$$\mathbf{s}^k = \mathbf{s}^{k+1} \frac{\partial \mathbf{n}^{k+1}}{\partial \mathbf{n}^k}, \tag{22}$$

where the derivation, $\frac{\partial \mathbf{n}^{k+1}}{\partial \mathbf{n}^k}$, is a Jacobian matrix:

$$\frac{\partial \mathbf{n}^{k+1}}{\partial \mathbf{n}^k} = \begin{bmatrix} \frac{\partial n_1^{k+1}}{\partial n_1^k} & \frac{\partial n_1^{k+1}}{\partial n_2^k} & \cdots & \frac{\partial n_1^{k+1}}{\partial n_{p^k}^k} \\ \frac{\partial n_2^{k+1}}{\partial n_1^k} & \frac{\partial n_2^{k+1}}{\partial n_2^k} & \cdots & \frac{\partial n_2^{k+1}}{\partial n_{p^k}^k} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{p^{k+1}}^{k+1}}{\partial n_1^k} & \frac{\partial n_{p^{k+1}}^{k+1}}{\partial n_2^k} & \cdots & \frac{\partial n_{p^{k+1}}^{k+1}}{\partial n_{p^k}^k} \end{bmatrix}. \tag{25}$$

Here:

- entry (i, j) in the matrix is $\frac{\partial n_i^{k+1}}{\partial n_j^k}$
- n_i^{k+1} is the net input of the i^{th} perceptron on layer $k+1$
- n_j^k is the net input of the j^{th} perceptron on layer k
- p^{k+1} and p^k are the number of perceptrons on layer $k+1$ and k

The Jacobian Matrix

- In order to derive the equation of entry (i, j) in the Jacobian matrix, $\frac{\partial n_i^{k+1}}{\partial n_j^k}$, we first derive the equation of the net input of the i^{th} perceptron on layer $k+1$, n_i^{k+1}.
- Since the i^{th} perceptron on layer $k+1$ is fully connected with all the p^k perceptrons on layer k, we can write n_i^{k+1} as

$$n_i^{k+1} = b_i^{k+1} + \sum_{q=1}^{p^k} a_q^k w_{qi}^{k+1}. \tag{26}$$

Here:

- w_{qi}^{k+1} is the connecting weight between the q^{th} perceptron on layer k and the i^{th} perceptron on layer $k+1$
- a_q^k is the output of the q^{th} perceptron on layer k
- b_i^{k+1} is the bias of the i^{th} perceptron on layer $k+1$

The Jacobian Matrix

- By substituting eq. (26)

$$n_i^{k+1} = b_i^{k+1} + \sum_{q=1}^{p^k} a_q^k w_{qi}^{k+1} \tag{26}$$

into $\frac{\partial n_i^{k+1}}{\partial n_j^k}$, we have

$$\frac{\partial n_i^{k+1}}{\partial n_j^k} = \frac{\partial \left(b_i^{k+1} + \sum_{q=1}^{p^k} a_q^k w_{qi}^{k+1}\right)}{\partial n_j^k}. \tag{27}$$

- Since b_i^{k+1} (the bias of the i^{th} perceptron on layer $k+1$) is not related to n_j^k, we can rewrite eq. (27) as

$$\frac{\partial n_i^{k+1}}{\partial n_j^k} = \frac{\partial \sum_{q=1}^{p^k} a_q^k w_{qi}^{k+1}}{\partial n_j^k}. \tag{28}$$

The Jacobian Matrix

- The output of the q^{th} perceptron on layer k, a_q^k, is

$$a_q^k = f_q^k(n_q^k), \tag{29}$$

where f_q^k and n_q^k are the activation function and net input of the q^{th} perceptron on layer k.

- Based on eq. (29), a_q^k is only related to n_j^k when $q = j$, eq. (28)

$$\frac{\partial n_i^{k+1}}{\partial n_j^k} = \frac{\partial \sum_{q=1}^{p^k} a_q^k w_{qi}^{k+1}}{\partial n_j^k} \tag{28}$$

can be written as

$$\frac{\partial n_i^{k+1}}{\partial n_j^k} = \frac{\partial a_j^k w_{ij}^{k+1}}{\partial n_j^k} = w_{ij}^{k+1} \frac{\partial f_j^k(n_j^k)}{\partial n_j^k} = w_{ij}^{k+1} \dot{f}_j^k(n_j^k). \tag{30}$$

The Jacobian Matrix

- By substituting eq. (30) into the Jacobian matrix in eq. (25)

$$\frac{\partial \mathbf{n}^{k+1}}{\partial \mathbf{n}^k} = \begin{bmatrix} \frac{\partial n_1^{k+1}}{\partial n_1^k} & \frac{\partial n_1^{k+1}}{\partial n_2^k} & \cdots & \frac{\partial n_1^{k+1}}{\partial n_{p^k}^k} \\ \frac{\partial n_2^{k+1}}{\partial n_1^k} & \frac{\partial n_2^{k+1}}{\partial n_2^k} & \cdots & \frac{\partial n_2^{k+1}}{\partial n_{p^k}^k} \\ \vdots & \vdots & & \vdots \\ \frac{\partial n_{p^{k+1}}^{k+1}}{\partial n_1^k} & \frac{\partial n_{p^{k+1}}^{k+1}}{\partial n_2^k} & \cdots & \frac{\partial n_{p^{k+1}}^{k+1}}{\partial n_{p^k}^k} \end{bmatrix}, \tag{25}$$

we have

$$\frac{\partial \mathbf{n}^{k+1}}{\partial \mathbf{n}^k} = \left(\mathbf{W}^{k+1}\right)^\intercal \dot{\mathbf{F}}^k(\mathbf{n}^k), \tag{31}$$

where $\dot{\mathbf{F}}^k(\mathbf{n}^k)$ is a diagonal matrix:

$$\dot{\mathbf{F}}^k(\mathbf{n}^k) = \begin{bmatrix} \dot{f}_1^k(n_1^k) & 0 & \cdots & 0 \\ 0 & \dot{f}_2^k(n_2^k) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \dot{f}_{p^k}^k(n_{p^k}^k) \end{bmatrix}. \tag{32}$$

The Jacobian Matrix

- By substituting eq. (31)

$$\frac{\partial \mathbf{n}^{k+1}}{\partial \mathbf{n}^{k}} = \left(\mathbf{W}^{k+1}\right)^{\intercal} \dot{\mathbf{F}}^{k}(\mathbf{n}^{k}) \tag{31}$$

into eq. (22)

$$\mathbf{s}^{k} = \mathbf{s}^{k+1} \frac{\partial \mathbf{n}^{k+1}}{\partial \mathbf{n}^{k}}, \tag{22}$$

we have

$$\mathbf{s}^{k} = \mathbf{s}^{k+1} \left(\mathbf{W}^{k+1}\right)^{\intercal} \dot{\mathbf{F}}^{k}(\mathbf{n}^{k}), \tag{33}$$

where $\dot{\mathbf{F}}^{k}(\mathbf{n}^{k})$ is given in eq. (32)

$$\dot{\mathbf{F}}^{k}(\mathbf{n}^{k}) = \begin{bmatrix} \dot{f}_1^{k}(n_1^{k}) & 0 & \cdots & 0 \\ 0 & \dot{f}_2^{k}(n_2^{k}) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \dot{f}_{p^k}^{k}(n_{p^k}^{k}) \end{bmatrix}. \tag{32}$$

Backward Pass: The Idea + Math

- Backward pass works as follows:
  1. calculates the loss function (Mean Squared Error) based on the target and the output of the output layer using eq. (11):
  $$\mathcal{L}(\boldsymbol{\theta}) = (\mathbf{y} - \mathbf{a}^l)^2 \tag{11}$$
  2. calculates the sensitivity of the output layer (i.e., the last layer with index l) using eq. (24):
  $$\mathbf{s}^l = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^l} = -2(\mathbf{y} - \mathbf{a}^l)\dot{\mathbf{f}}^l(\mathbf{n}^l) \tag{24}$$
  3. from the last hidden layer down to the first hidden layer, uses eq. (33) to calculate the sensitivity of the current layer (e.g., layer k) from the sensitivity of the next layer (e.g., layer $k+1$):
  $$\mathbf{s}^k = \mathbf{s}^{k+1}\left(\mathbf{W}^{k+1}\right)^\intercal \dot{\mathbf{F}}^k(\mathbf{n}^k), \quad \text{where} \quad l-1 \geq k \geq 1 \tag{33}$$

Summary

1. Forward pass
   1. passes the input to the input layer:

$$\mathbf{a}^0 = \mathbf{f}^0(\mathbf{x}) = \mathbf{x} \tag{2}$$

   2. calculates the output from the first hidden layer up to the output layer:

$$\mathbf{a}^k = \mathbf{f}^k(\mathbf{n}^k) = \mathbf{f}^k\left(\begin{bmatrix}1 & \mathbf{a}^{k-1}\end{bmatrix}\boldsymbol{\theta}^k\right), \quad \text{where} \quad 1 \leq k \leq l \tag{21}$$

2. Backward pass
   1. calculates the sensitivity of the output layer:

$$\mathbf{s}^l = \frac{\partial \mathcal{L}(\boldsymbol{\theta})}{\partial \mathbf{n}^l} = -2(\mathbf{y} - \mathbf{a}^l)\dot{\mathbf{f}}^l(\mathbf{n}^l) \tag{24}$$

   2. calculates the sensitivity from the last hidden layer down to the first:

$$\mathbf{s}^k = \mathbf{s}^{k+1}\left(\mathbf{W}^{k+1}\right)^\intercal \dot{\mathbf{F}}^k(\mathbf{n}^k), \quad \text{where} \quad l-1 \geq k \geq 1 \tag{33}$$

3. Gradient descent
   1. updates the parameters of each layer:

$$\boldsymbol{\theta}^k = \boldsymbol{\theta}^k - \eta \begin{bmatrix}1 & \mathbf{a}^{k-1}\end{bmatrix}^\intercal \mathbf{s}^k \tag{19}$$

Some Popular Activation Functions

- Here are some popular activation functions used in MLP:
  - HardLimit:

$$HardLimit(x) = \begin{cases} 1, & \text{if} \quad x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{34}$$

  - Rectified Linear Unit (ReLU):

$$ReLU(x) = \begin{cases} x, & \text{if} \quad x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{35}$$

  - Linear (a.k.a., Identity):

$$Linear(x) = x \tag{36}$$

  - Sigmoid (σ):

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{37}$$

  - Hyperbolic Tangent ($tanh$):

$$tanh(x) = 2\sigma(2x) - 1 \tag{38}$$

- We will discuss some of these activation functions in much greater detail (and introduce some new activation functions) in /p3_c2_s2_training_deep_neural_networks.

MLP: The Implementation

- See /models/p2_shallow_learning:
  1. cell 7

Example

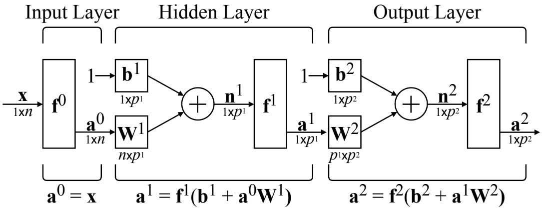

Figure 6: A fully connected three-layer MLP.

- Here is the first iteration of backpropagation on MLP in fig. 6, where:
  - $\mathbf{x} = \begin{bmatrix}1 & 1\end{bmatrix}$, $\mathbf{y} = 0$, $n = 2$
  - $\mathbf{W}^1 = \begin{bmatrix}1 & 1 \\ 1 & 1\end{bmatrix}$, $\mathbf{b}^1 = \begin{bmatrix}0 & 0\end{bmatrix}$, $p^1 = 2$
  - $\mathbf{W}^2 = \begin{bmatrix}1 & 1\end{bmatrix}^\top$, $\mathbf{b}^2 = 0$, $p^2 = 1$
  - $\mathbf{f}^1 = \text{ReLU}, \mathbf{f}^2 = \text{Linear}$

Forward Pass

- Pass the input to the input layer:

$$\mathbf{a}^0 = \mathbf{x} = \begin{bmatrix} 1 & 1 \end{bmatrix}. \tag{39}$$

- Calculate the output of the hidden layer:

$$\mathbf{n}^1 = \mathbf{b}^1 + \mathbf{a}^0\mathbf{W}^1 = \begin{bmatrix} 0 & 0 \end{bmatrix} + \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 \end{bmatrix}, \tag{40}$$

$$\mathbf{a}^1 = \mathbf{f}^1(\mathbf{n}^1) = \mathrm{ReLU}\left(\begin{bmatrix} 2 & 2 \end{bmatrix}\right) = \begin{bmatrix} 2 & 2 \end{bmatrix}. \tag{41}$$

- Calculate the output of the output layer:

$$\mathbf{n}^2 = \mathbf{b}^2 + \mathbf{a}^1\mathbf{W}^2 = 0 + \begin{bmatrix} 2 & 2 \end{bmatrix} \begin{bmatrix} 1 & 1 \end{bmatrix}^\intercal = 4, \tag{42}$$

$$\mathbf{a}^2 = \mathbf{f}^2(\mathbf{n}^2) = \mathrm{Linear}(4) = 4. \tag{43}$$

Backward Pass

- Calculate the sensitivity of the output layer:

$$\dot{\mathbf{f}}^2(\mathbf{n}^2) = \dot{\text{Linear}}(4) = 1, \tag{44}$$

$$\mathbf{s}^2 = -2(\mathbf{y} - \mathbf{a}^2)\dot{\mathbf{f}}^2(\mathbf{n}^2) = -2 \times (0 - 4) \times 1 = 8. \tag{45}$$

- Calculate the sensitivity of the hidden layer:

$$\dot{\mathbf{F}}^1(\mathbf{n}^1) = \begin{bmatrix} \dot{f}_1^1(n_1^1) & 0 \\ 0 & \dot{f}_2^1(n_2^1) \end{bmatrix} = \begin{bmatrix} \dot{\text{ReLU}}(2) & 0 \\ 0 & \dot{\text{ReLU}}(2) \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \tag{46}$$

$$\mathbf{s}^1 = \mathbf{s}^2 \left(\mathbf{W}^2\right)^\intercal \dot{\mathbf{F}}^1(\mathbf{n}^1) = 8 \times \begin{bmatrix} 1 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 8 & 8 \end{bmatrix}. \tag{47}$$

Gradient Descent

- Update the parameters of the hidden layer (with $\eta = 0.1$):

$$\mathbf{W}^1 = \mathbf{W}^1 - \eta \left(\mathbf{a}^0\right)^\intercal \mathbf{s}^1 = \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} - 0.1 \times \begin{bmatrix} 1 & 1 \end{bmatrix}^\intercal \begin{bmatrix} 8 & 8 \end{bmatrix} = \begin{bmatrix} 0.2 & 0.2 \\ 0.2 & 0.2 \end{bmatrix}, \tag{48}$$

$$\mathbf{b}^1 = \mathbf{b}^1 - \eta \mathbf{s}^1 = \begin{bmatrix} 0 & 0 \end{bmatrix} - 0.1 \times \begin{bmatrix} 8 & 8 \end{bmatrix} = \begin{bmatrix} -0.8 & -0.8 \end{bmatrix}. \tag{49}$$

- Update the parameters of the output layer (with $\eta = 0.1$):

$$\mathbf{W}^2 = \mathbf{W}^2 - \eta \left(\mathbf{a}^1\right)^\intercal \mathbf{s}^2 = \begin{bmatrix} 1 & 1 \end{bmatrix}^\intercal - 0.1 \times \begin{bmatrix} 2 & 2 \end{bmatrix}^\intercal \times 8 = \begin{bmatrix} -0.6 & -0.6 \end{bmatrix}^\intercal, \tag{50}$$

$$\mathbf{b}^2 = \mathbf{b}^2 - \eta \mathbf{s}^2 = 0 - 0.1 \times 8 = -0.8. \tag{51}$$