Binary Classification

生醫光電所 吳育德

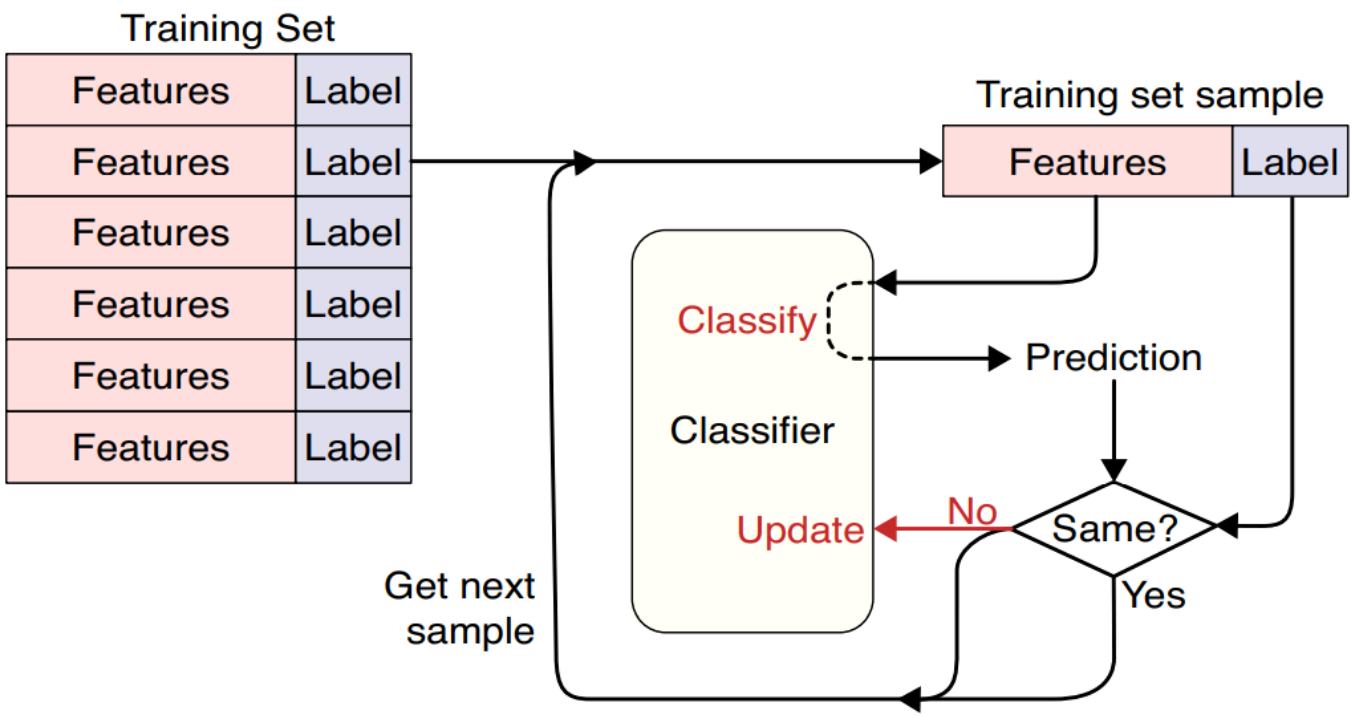
Paolo Perrotta, Programming Machine Learning: From Coding to Deep Learning (2020)

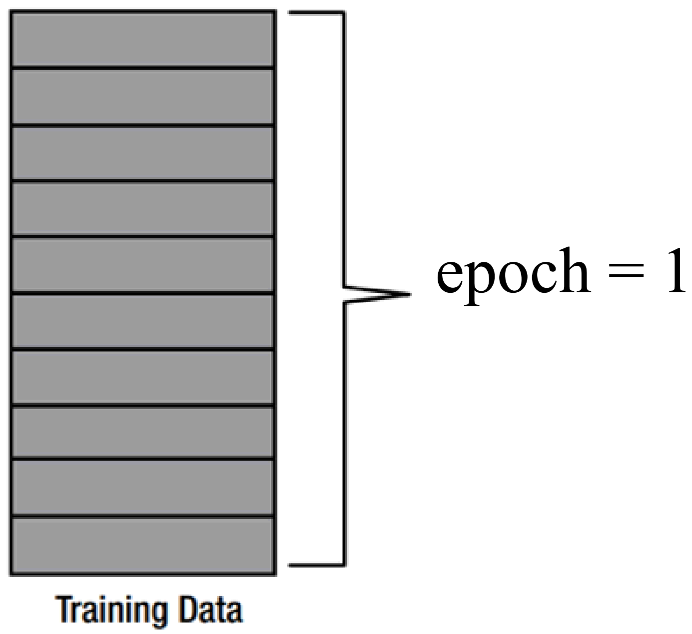
Outline

- Leap from multiple linear regression to classification.
- Use a binary classifier to recognize a single digit in the MNIST dataset.
- Multiclass classification in recognizing all MNIST characters.

The Essence of Training Classifier and Epoch

- Epoch is the **number of completed training cycles for all the training data**.

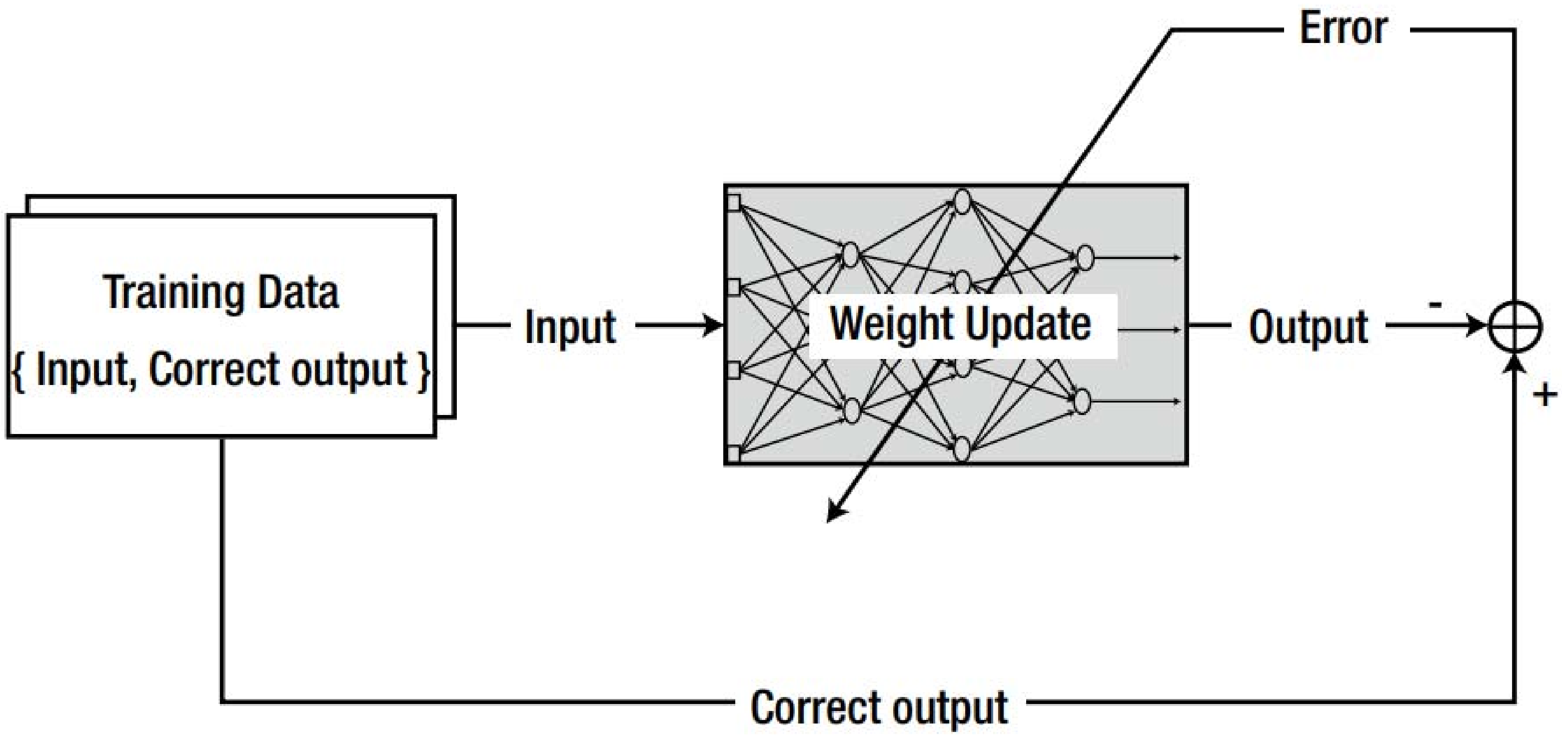




Andrew Glassner - Deep Learning_ A Visual Approach-No Starch Press (2021)

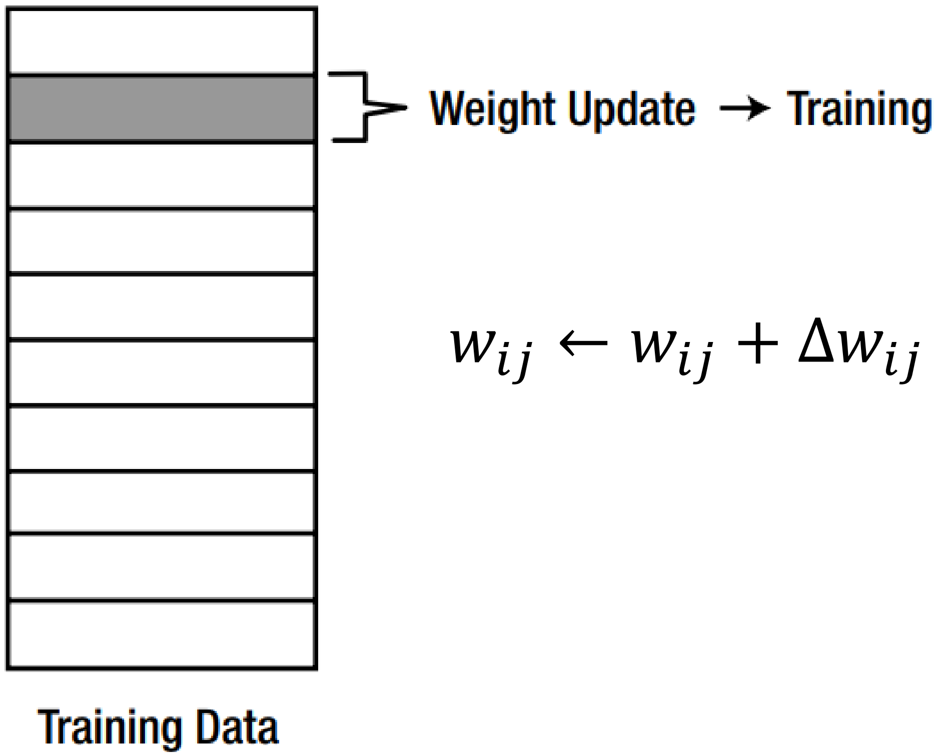
Phil Kim - MatLab Deep Learning with Machine Learning, Neural Networks and Artificial Intelligence 2017, Apress

Supervised Learning of a Neural Network



Stochastic Gradient Descent (SGD) with Batch Size 1

- Assume 99 training data, the weights are updated 99 iterations for each epoch
- Training 10 epochs needs 990 iterations

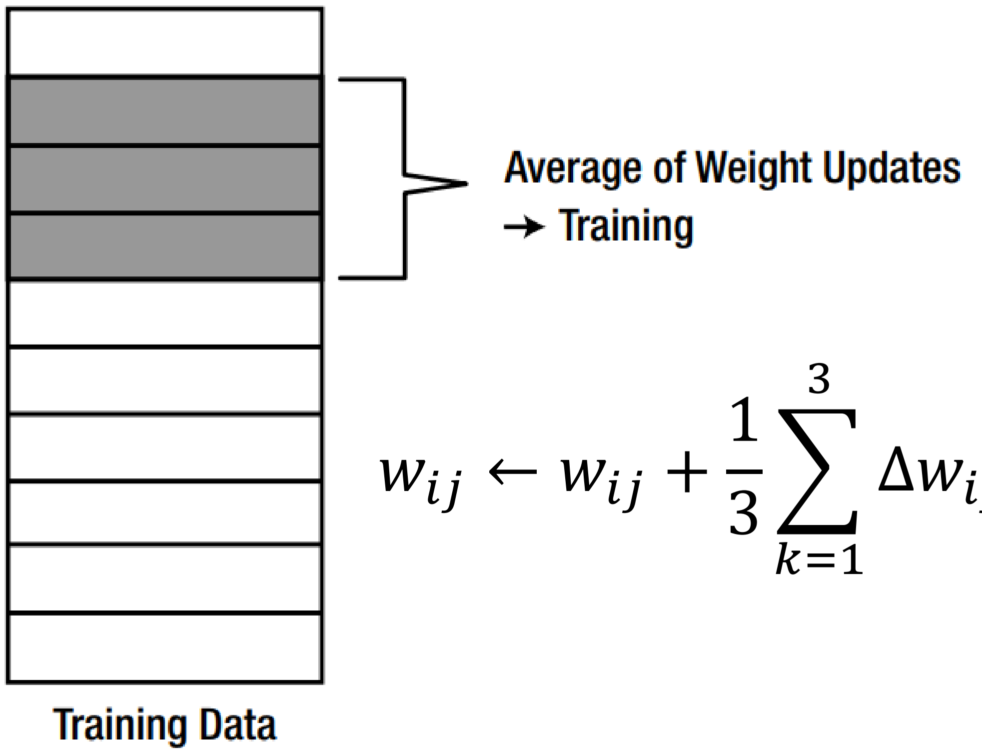


$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij}$$

Phil Kim - MatLab Deep Learning with Machine Learning, Neural Networks and Artificial Intelligence 2017, Apress

Mini-batch Gradient descent with Batch Size 3

- Assume 99 training data, the weights are updated 33 iterations for each epoch
- Training 10 epochs needs 330 iterations

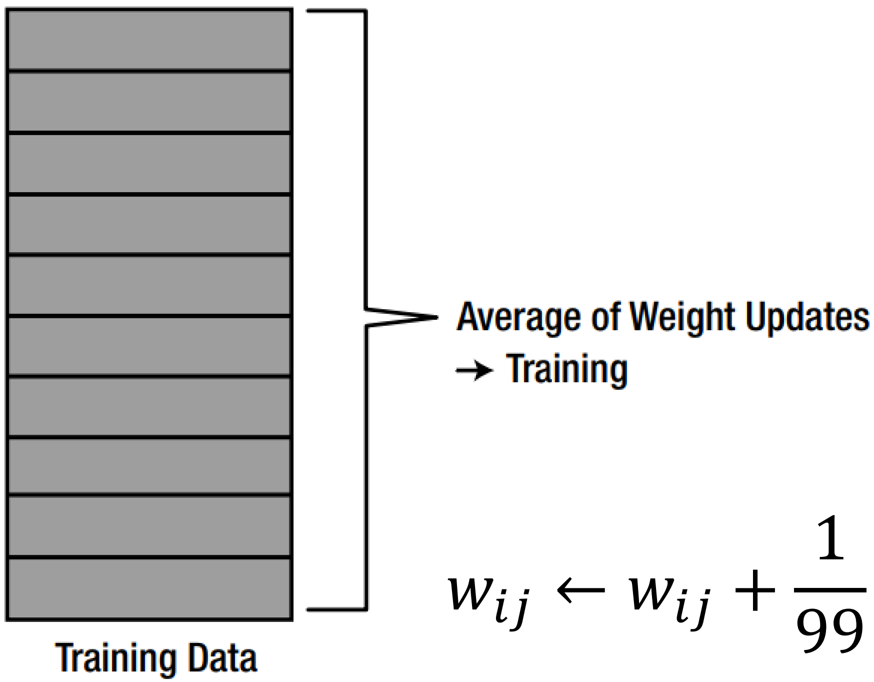


$$w_{ij} \leftarrow w_{ij} + \frac{1}{3}\sum_{k=1}^{3} \Delta w_{ij}\,(k)$$

Phil Kim - MatLab Deep Learning with Machine Learning, Neural Networks and Artificial Intelligence 2017, Apress

Batch Gradient descent

- Assume 99 training data, the weights are updated 1 iteration for each epoch
- Training 10 epochs needs 10 iterations



$$w_{ij} \leftarrow w_{ij} + \frac{1}{99}\sum_{k=1}^{99} \Delta w_{ij}\,(k)$$

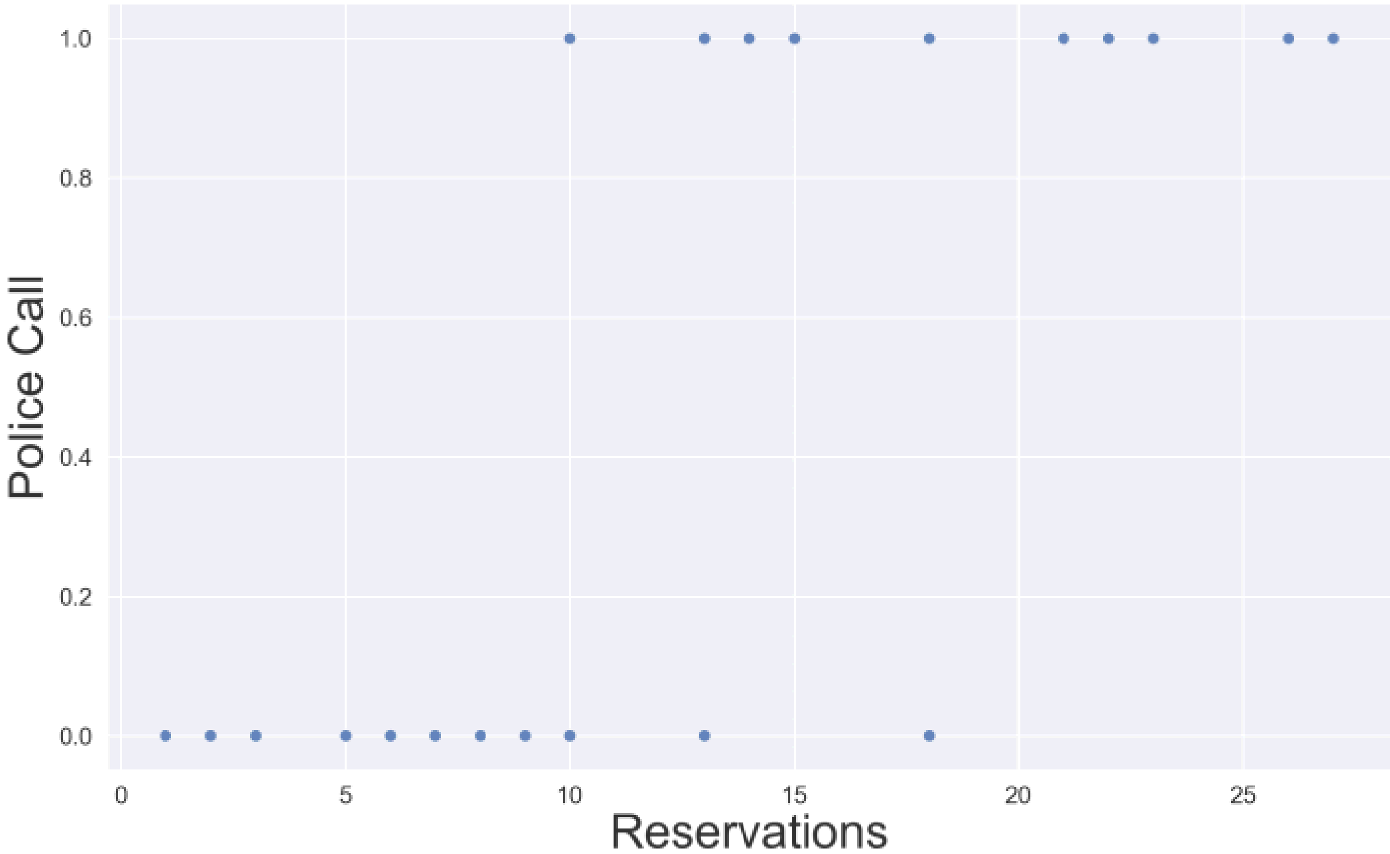
Phil Kim - MatLab Deep Learning with Machine Learning, Neural Networks and Artificial Intelligence 2017, Apress

Where Linear Regression Fails: Binary Classification

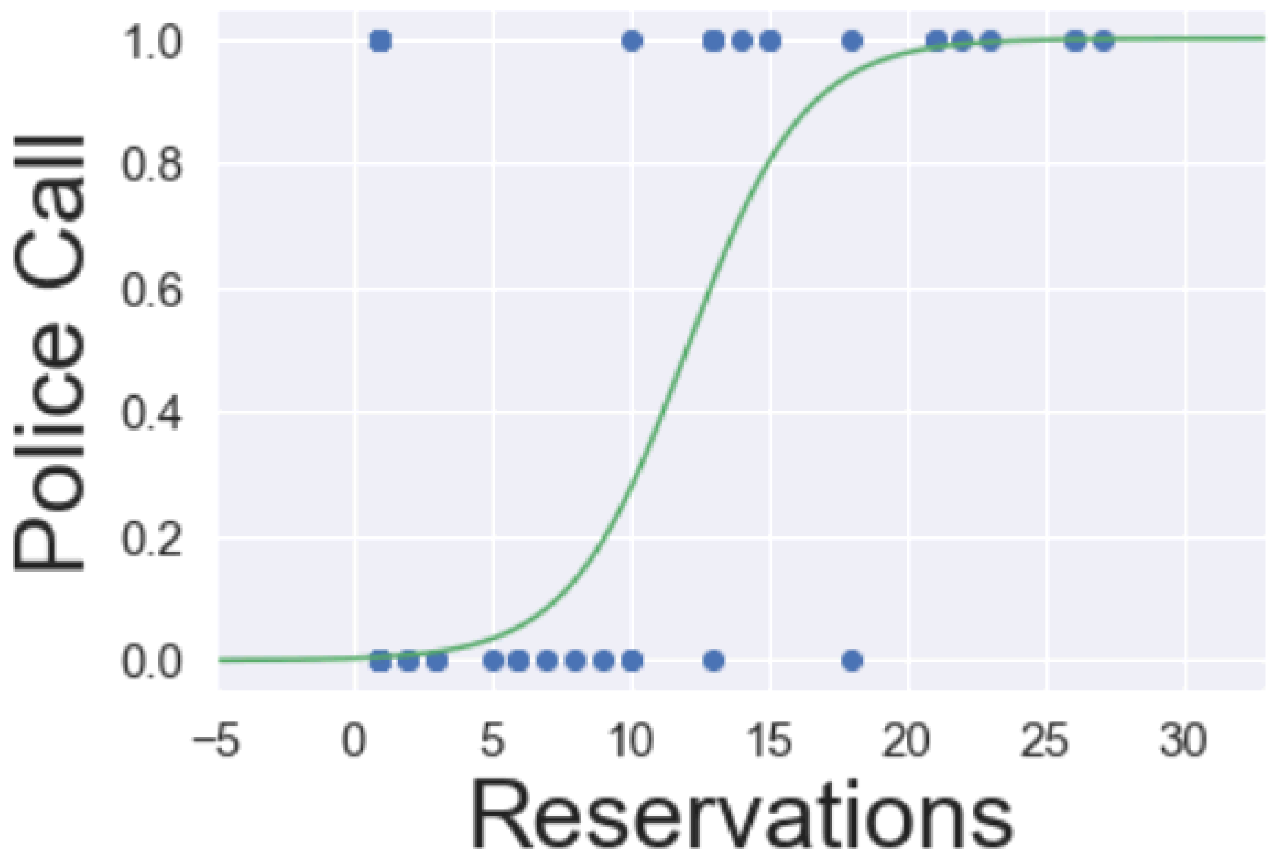
- On busy nights, it's common for noisy customers to hang out in front of the pizzeria, until the neighbors eventually call the police.
- Roberto suspects that the same input variables such as temperature and tourists, also affect the number of loud customers by the entrance, and hence the likelihood of a police call.
- He wants to know in advance whether a police call is likely to happen, so that he can set up countermeasures—such as walk outside and beg people to lower their voices.

Reservations	Temperature	Tourists	Police
13	26	9	1
2	14	6	0
14	20	3	1
23	25	9	1
13	24	8	1
1	13	2	0
18	23	9	1
10	18	10	1
26	24	3	1
3	14	1	0
3	12	3	0
21	27	5	1
7	17	3	0
22	21	1	1
2	14	4	0
27	26	2	1
6	15	4	0
10	21	7	0
18	18	3	0
15	26	8	1
9	20	6	0
26	25	9	1
8	21	10	0
15	22	7	1
10	20	2	0
21	21	1	1
5	12	7	0
6	14	9	0
13	19	4	1
13	20	3	0

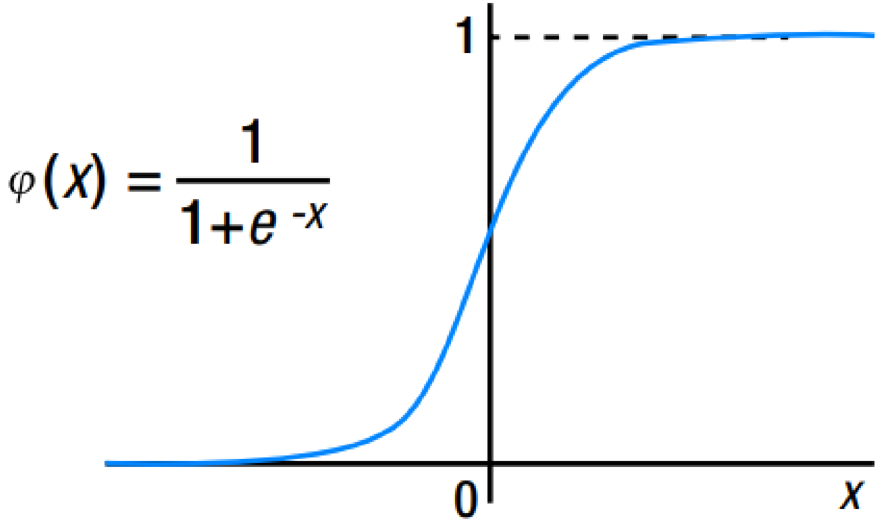
“Reservations” column against the label



“Reservations” column against the label



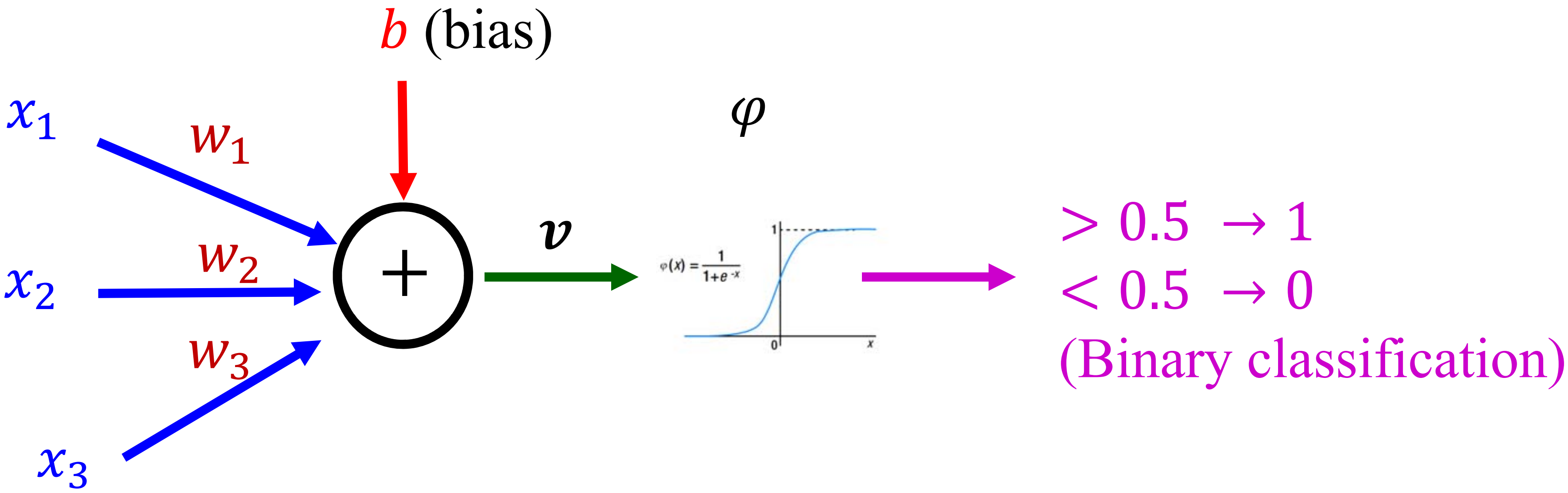
Using the sigmoid function as an activation function



$$\frac{d(1+e^{-x})^{-1}}{dx} = -(1+e^{-x})^{-2}(-e^{-x}) = \frac{1}{1+e^{-x}}\left(1 - \frac{1}{1+e^{-x}}\right)$$

$$\varphi'(x) = \varphi(x)\big(1 - \varphi(x)\big)$$

Multiple inputs and single output for Binary Classification



$$v = x_1 w_1 + x_2 w_2 + x_3 w_3 + b = \boldsymbol{x}\boldsymbol{w} + b$$

$$\text{where } \boldsymbol{x} = [x_1\ x_2\ x_3],\ \boldsymbol{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix}$$

$y = \varphi(v)$, φ is the sigmoid activation function

When $b \equiv w_1 \neq 0$ and $y = \varphi(v)$ with MSE

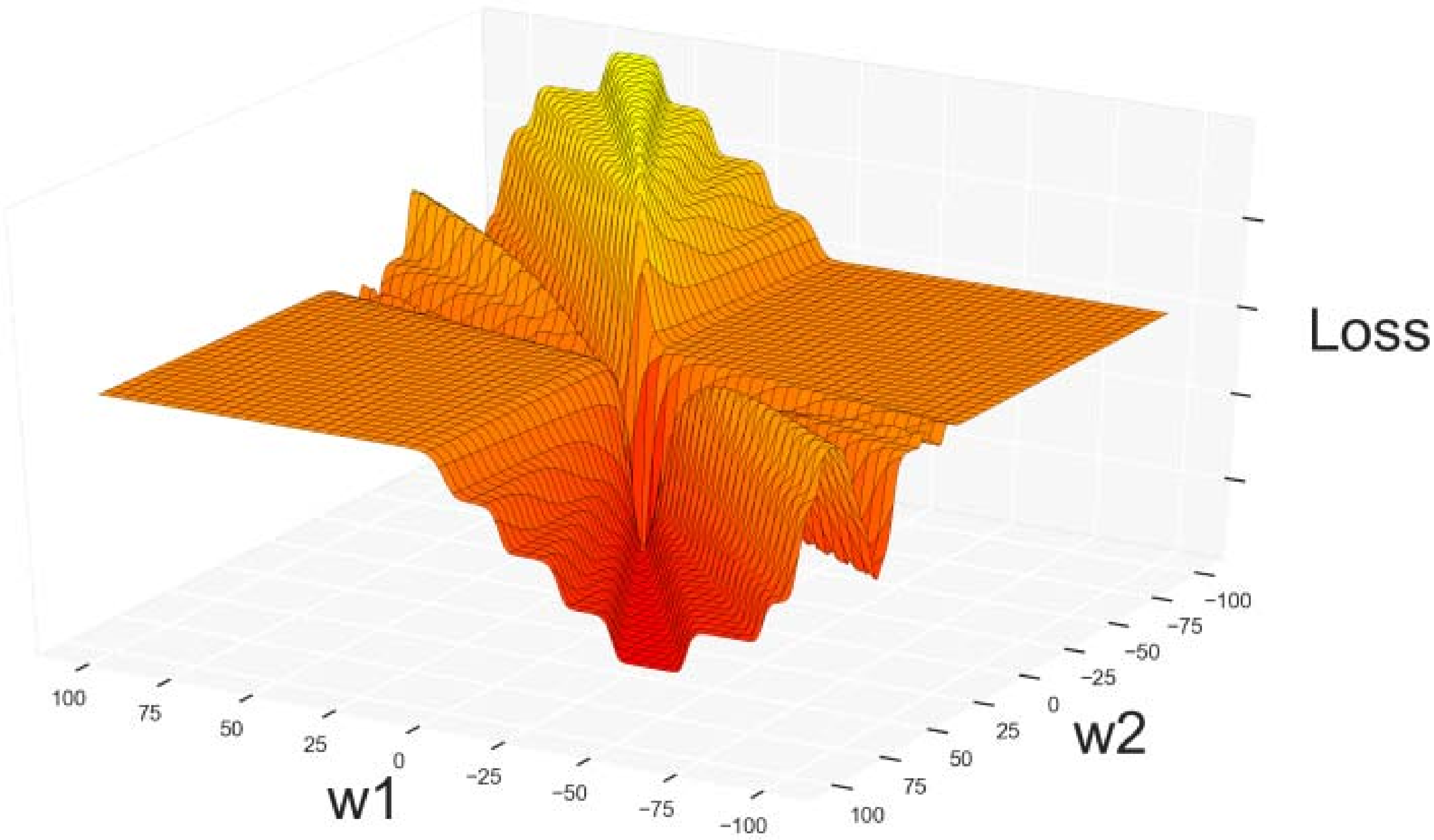
$$\hat{v}_1 = [1\ x_{12}\ x_{13}\ x_{14}]\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \Rightarrow \quad e_1 = \varphi(\hat{v}_1) - y_1 = \hat{y}_1 - y_1$$

$$\vdots$$

$$\hat{v}_N = [1\ x_{N2}\ x_{N3}\ x_{N4}]\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \Rightarrow \quad e_N = \varphi(\hat{v}_N) - y_N = \hat{y}_N - y_N$$

$$\mathrm{Loss}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}(\varphi(\hat{v}_n) - y_n)^2, \hat{\boldsymbol{y}}_{N\times 1} = \varphi([\mathbf{1}_{N\times 1}\ \ \boldsymbol{X}_{N\times 3}] \times \boldsymbol{w}_{4\times 1})$$

Mean Squared Error Has Too Many local Minimum



04_plot_losses.jpynb

Minimize Mean Squared Error Using Gradient Descent

- The loss function for output

$$\text{Loss}(\boldsymbol{w}) = \frac{1}{N}\sum_{n=1}^{N}(\varphi(w_1 + x_{n2}w_2 + x_{n3}w_3 + x_{n4}w_4) - y_n)^2 = \frac{1}{N}\sum_{n=1}^{N} e_N^2$$

- Minimize the loss function *Loss* w.r.t w_1, w_2, w_3 and w_4

$$\frac{\partial Loss}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial Loss}{\partial w_1} \\ \frac{\partial Loss}{\partial w_2} \\ \frac{\partial Loss}{\partial w_3} \\ \frac{\partial Loss}{\partial w_4} \end{bmatrix} = \begin{bmatrix} \frac{\partial Loss}{\partial \hat{y}_n}\frac{\partial \hat{y}_n}{\partial \hat{v}_n}\frac{\partial \hat{v}_n}{\partial w_1} \\ \frac{\partial Loss}{\partial \hat{y}_n}\frac{\partial \hat{y}_n}{\partial \hat{v}_n}\frac{\partial \hat{v}_n}{\partial w_2} \\ \frac{\partial Loss}{\partial \hat{y}_n}\frac{\partial \hat{y}_n}{\partial \hat{v}_n}\frac{\partial \hat{v}_n}{\partial w_3} \\ \frac{\partial Loss}{\partial \hat{y}_n}\frac{\partial \hat{y}_n}{\partial \hat{v}_n}\frac{\partial \hat{v}_n}{\partial w_4} \end{bmatrix} \begin{bmatrix} \frac{2}{N}\sum_{n=1}^{N} e_n\varphi'(\hat{v}_n) \\ \frac{2}{N}\sum_{n=1}^{N} e_n\varphi'(\hat{v}_n)x_{n2} \\ \frac{2}{N}\sum_{n=1}^{N} e_n\varphi'(\hat{v}_n)x_{n3} \\ \frac{2}{N}\sum_{n=1}^{N} e_n\varphi'(\hat{v}_n)x_{n4} \end{bmatrix} = \frac{2}{N}\left[\mathbf{1}_{N\times 1}\ \ \boldsymbol{X}_{N\times 3}\right]^{\mathrm{T}}_{4\times\mathrm{N}} e_{N\times 1}\varphi'$$

- The Batch Gradient Decent method

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha\frac{\partial Loss}{\partial \boldsymbol{w}}$$

$$[\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 3}]^{\mathrm{T}}_{4\times \mathrm{N}} \qquad \times \qquad e_{N\times 1}\varphi'$$

x_1		1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
x_2	Reservations	13 2 14 23 13 1 18 10 26 3 3 21 7 22 2 27 6 10 18 15 9 26 8 15 10 21 5 6 13 13
x_3	Temperature	26 14 20 25 24 12 23 18 24 14 12 27 17 21 12 26 15 21 18 26 20 25 21 22 20 21 12 14 19 20
x_4	Pizzas	44 23 28 60 42 5 51 44 42 9 14 43 22 34 16 46 26 33 29 43 37 62 47 38 22 29 34 38 30 28

$$\begin{bmatrix} e_1\varphi'(\hat{v}_1) \\ e_2\varphi'(\hat{v}_2) \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ e_{N-1}\varphi'(\hat{v}_{N-1}) \\ e_N\varphi'(\hat{v}_N) \end{bmatrix}$$

Cross Entropy

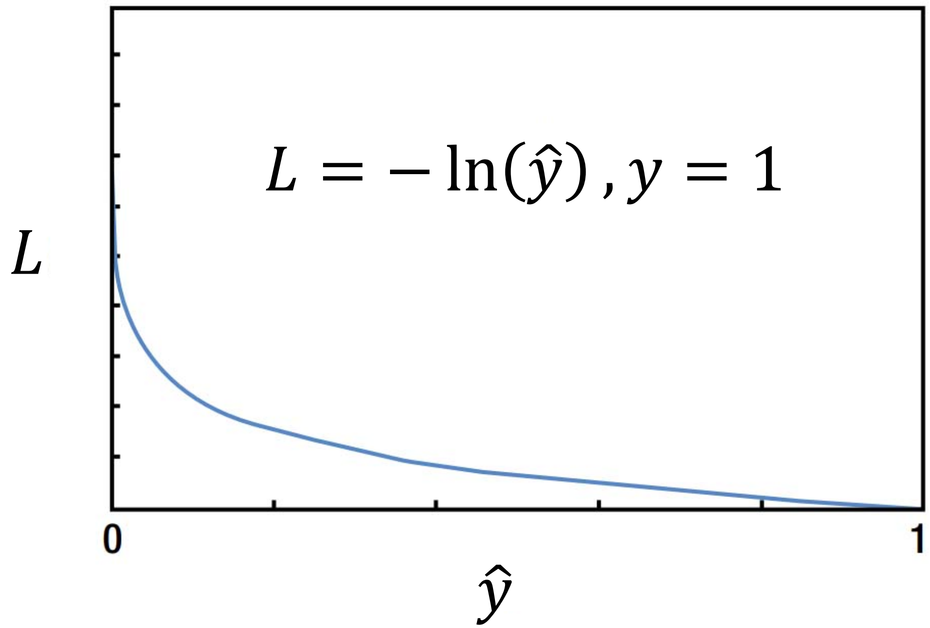
- $L = -y\ln(\hat{y}) - (1-y)\ln(1-\hat{y})$ is the concatenation of the following two equations：

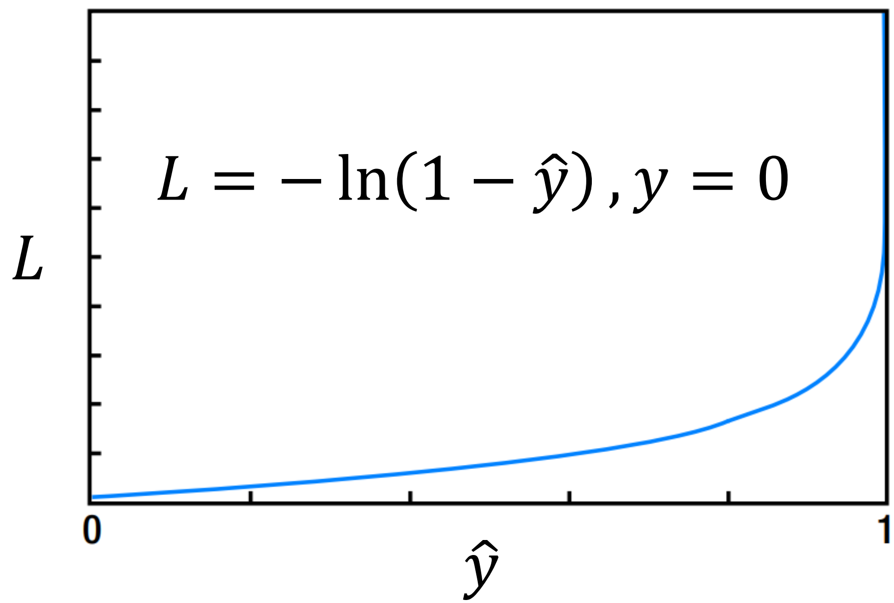
$$L = \begin{cases} -\ln(\hat{y}) & y = 1 \\ -\ln(1-\hat{y}) & y = 0 \end{cases}$$

- Therefore, the cross entropy cost function often teams up with **sigmoid** and **softmax** activation functions in the neural network.

Cross Entropy

- This cost function is proportional to the error.
- The cross entropy function is much more sensitive to the error than quadratic function.





$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial(-y\ln(\hat{y}) - (1-y)\ln(1-\hat{y}))}{\partial \hat{y}}$$

$$= -y\frac{1}{\hat{y}} - (1-y)\frac{-1}{1-\hat{y}}$$

$$= \frac{-y(1-\hat{y}) + (1-y)\hat{y}}{\hat{y}(1-\hat{y})}$$

$$= \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})}$$

$$\hat{y} = \varphi(\hat{v})$$

$$\frac{\partial \hat{y}}{\partial \hat{v}} = \varphi(\hat{v})\big(1-\varphi(\hat{v})\big) = \hat{y}(1-\hat{y})$$

$$\color{red}{\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \hat{v}}} = \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})} \times \hat{y}(1-\hat{y}) = -(y-\hat{y}) = \hat{y} - y = \varphi(\hat{v}) - y \rightarrow \color{red}{e}$$

When $b \equiv w_1 \neq 0$ and $y = \varphi(v)$ with CE loss

$$\hat{v}_1 = [1\ x_{12}\ x_{13}\ x_{14}]\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \Rightarrow \quad e_1 = \varphi(\hat{v}_1) - y_1 = \hat{y}_1 - y_1$$

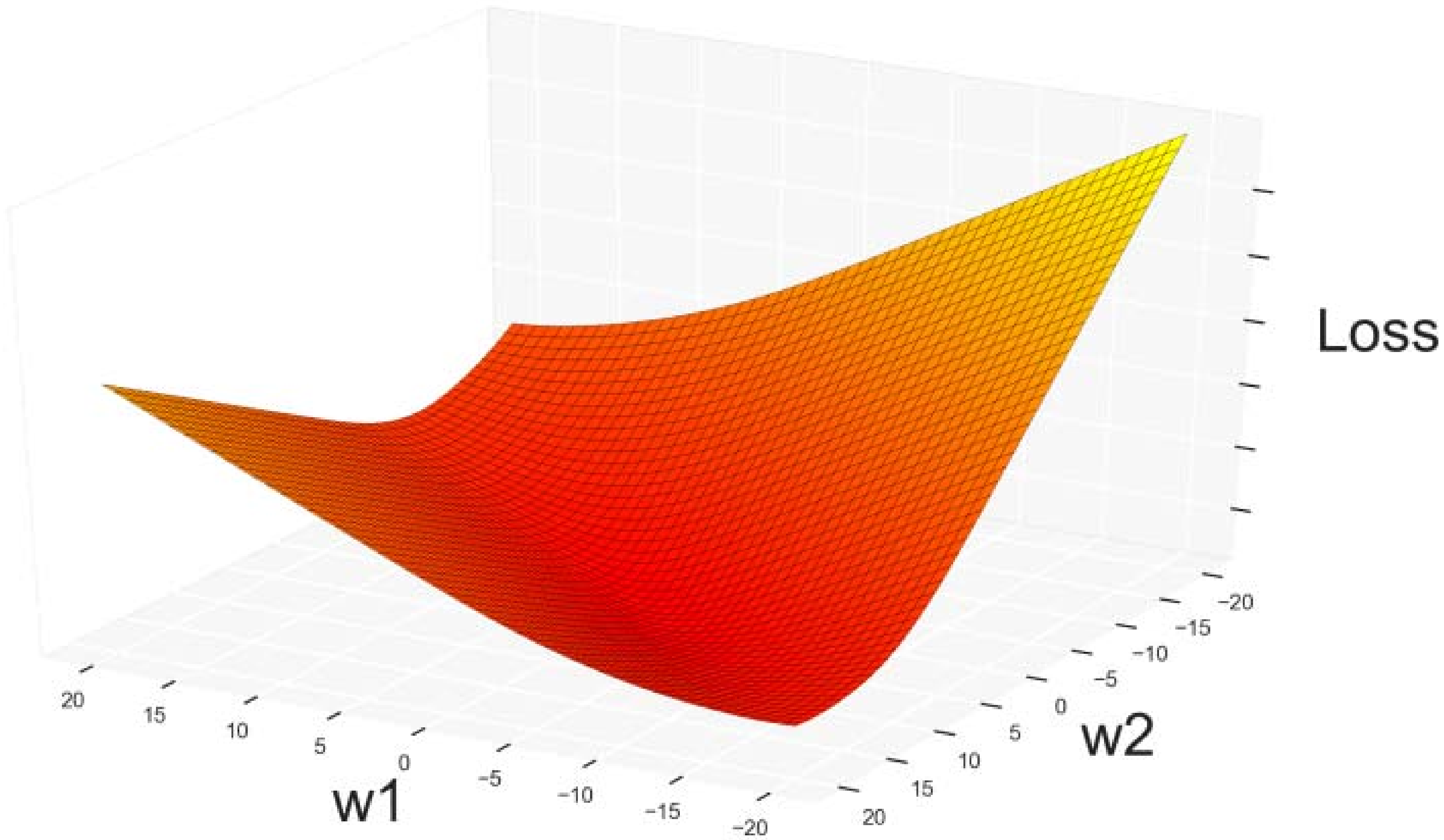
$$\vdots$$

$$\hat{v}_N = [1\ x_{N2}\ x_{N3}\ x_{N4}]\begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} \Rightarrow \quad e_N = \varphi(\hat{v}_N) - y_N = \hat{y}_N - y_N$$

$$\mathrm{Loss}(\boldsymbol{w}) = \frac{-1}{N}\sum_{n=1}^{N}\left(y_n \ln(\hat{y}_n) + (1 - y_n)\ln(1 - \hat{y}_n)\right),$$

$$\hat{\boldsymbol{y}}_{N\times 1} = \varphi([\mathbf{1}_{N\times 1}\ \ \boldsymbol{X}_{N\times 3}] \times \boldsymbol{w}_{4\times 1})$$

Cross Entropy Error is Nice and Smooth



04_plot_losses.jpynb

Minimize Mean Cross Entropy Using Gradient Descent

- The loss function for output

$$\text{Loss}(\boldsymbol{w}) = \frac{-1}{N}\sum_{n=1}^{N}(y_n \ln(\hat{y}_n) + (1 - y_n)\ln(1 - \hat{y}_n))$$

- Minimize the loss function $Loss$ w.r.t w_1, w_2, w_3 and w_4

$$\frac{\partial Loss}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial Loss}{\partial w_1} \\ \frac{\partial Loss}{\partial w_2} \\ \frac{\partial Loss}{\partial w_3} \\ \frac{\partial Loss}{\partial w_4} \end{bmatrix} = \begin{bmatrix} \frac{\partial Loss}{\partial \hat{y}_n}\frac{\partial \hat{y}_n}{\partial \hat{v}_n}\frac{\partial \hat{v}_n}{\partial w_1} \\ \frac{\partial Loss}{\partial \hat{y}_n}\frac{\partial \hat{y}_n}{\partial \hat{v}_n}\frac{\partial \hat{v}_n}{\partial w_2} \\ \frac{\partial Loss}{\partial \hat{y}_n}\frac{\partial \hat{y}_n}{\partial \hat{v}_n}\frac{\partial \hat{v}_n}{\partial w_3} \\ \frac{\partial Loss}{\partial \hat{y}_n}\frac{\partial \hat{y}_n}{\partial \hat{v}_n}\frac{\partial \hat{v}_n}{\partial w_4} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{n=1}^{N} e_n \\ \frac{1}{N}\sum_{n=1}^{N} e_n x_{n2} \\ \frac{1}{N}\sum_{n=1}^{N} e_n x_{n3} \\ \frac{1}{N}\sum_{n=1}^{N} e_n x_{n4} \end{bmatrix} = \frac{2}{N}[\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 3}]^{\mathrm{T}}_{4\times \mathrm{N}} e_{N\times 1}$$

- The Batch Gradient Decent method

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha\frac{\partial Loss}{\partial \boldsymbol{w}}$$

$$[\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 3}]^{\mathrm{T}}_{4\times \mathrm{N}} \qquad \times \qquad e_{N\times 1}$$

x_1		1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
x_2	Reservations	13	2	14	23	13	1	18	10	26	3	3	21	7	22	2	27	6	10	18	15	9	26	8	15	10	21	5	6	13	13
x_3	Temperature	26	14	20	25	24	12	23	18	24	14	12	27	17	21	12	26	15	21	18	26	20	25	21	22	20	21	12	14	19	20
x_4	Pizzas	44	23	28	60	42	5	51	44	42	9	14	43	22	34	16	46	26	33	29	43	37	62	47	38	22	29	34	38	30	28

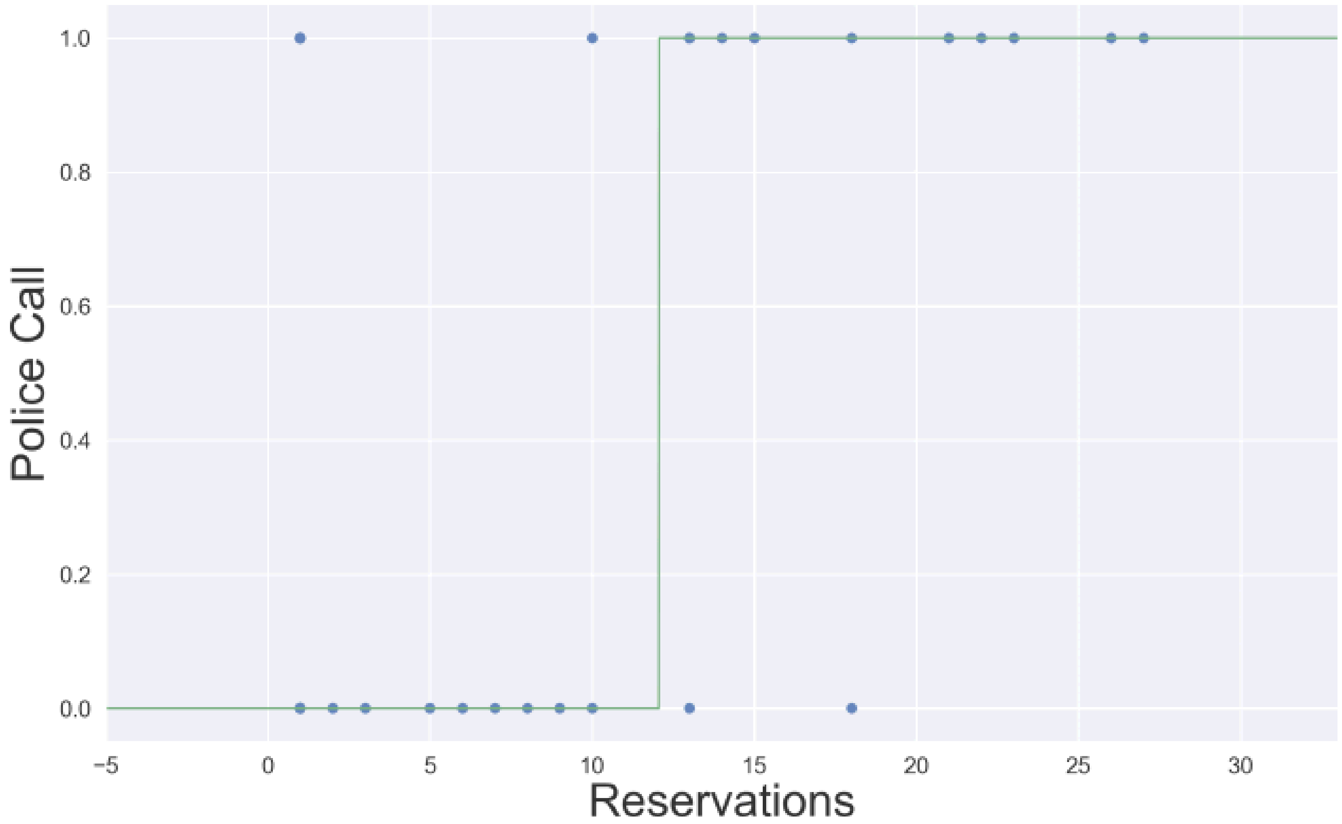
$$\begin{bmatrix} e_1 = \varphi(\hat{v}_1) - y_1 \\ e_2 = \varphi(\hat{v}_2) - y_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ e_N = \varphi(\hat{v}_N) - y_N \end{bmatrix}$$

```python
import numpy as np

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def forward(X, w):
    weighted_sum = np.matmul(X, w)
    return sigmoid(weighted_sum)

def classify(X, w):
    return np.round(forward(X, w))

def loss(X, Y, w):
    y_hat = forward(X, w)
    first_term = Y * np.log(y_hat)
    second_term = (1 - Y) * np.log(1 - y_hat)
    return -np.average(first_term + second_term)

def gradient(X, Y, w):
    return np.matmul(X.T, (forward(X, w) - Y)) / X.shape[0]
```

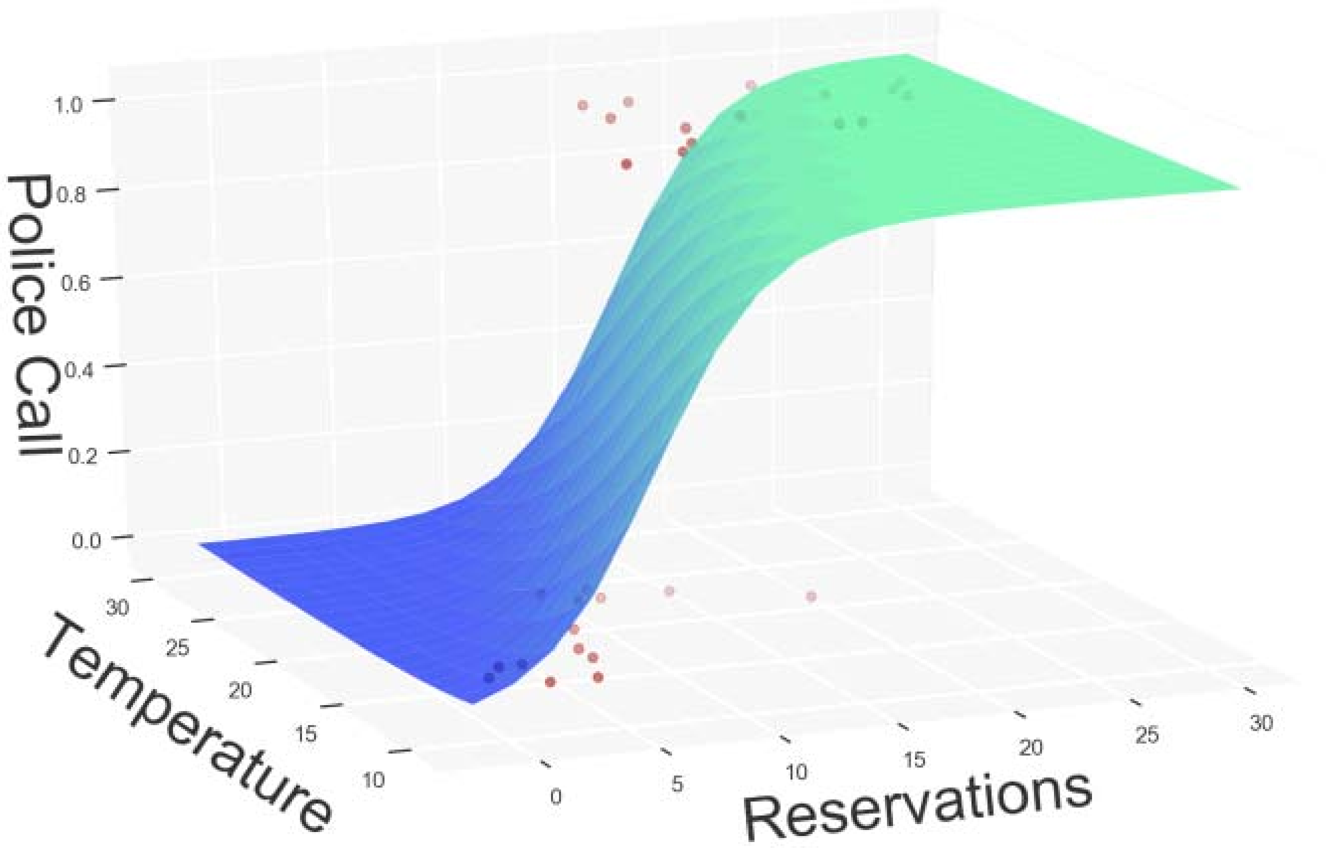
```python
def train(X, Y, iterations, lr):
    w = np.zeros((X.shape[1], 1))
    for i in range(iterations):
        print("Iteration %4d => Loss: %.20f" % (i, loss(X, Y, w)))
        w -= gradient(X, Y, w) * lr
    return w

def test(X, Y, w):
    total_examples = X.shape[0]
    correct_results = np.sum(classify(X, w) == Y)
    success_percent = correct_results * 100 / total_examples
    print("\nSuccess: %d/%d (%.2f%%)" %
          (correct_results, total_examples, success_percent))

# Prepare data
x1, x2, x3, y = np.loadtxt("police.txt", skiprows=1, unpack=True)
X = np.column_stack((np.ones(x1.size), x1, x2, x3))
Y = y.reshape(-1, 1)
w = train(X, Y, iterations=10000, lr=0.001)

# Test it
test(X, Y, w)
```

Binary Classification

Below a certain threshold around 12, the value of the model is under 0.5, and classify() rounds it to 0. Above that threshold, the value of the model is over 0.5, and classify() rounds it to 1. The result is a stepwise model function.

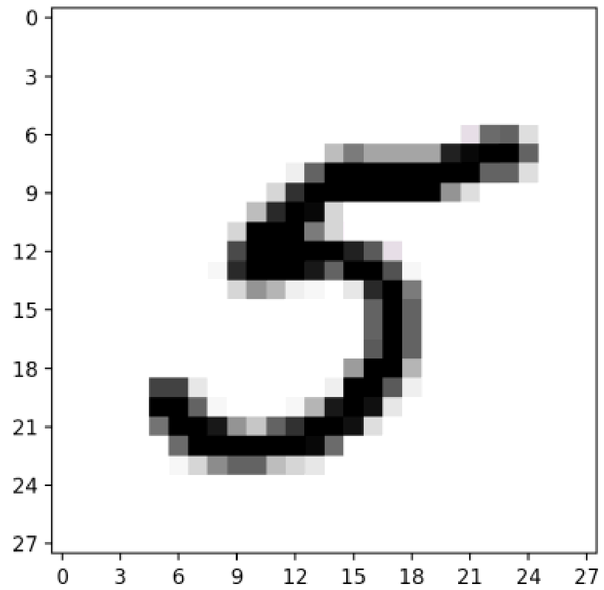


Visualization of Classifier's Model

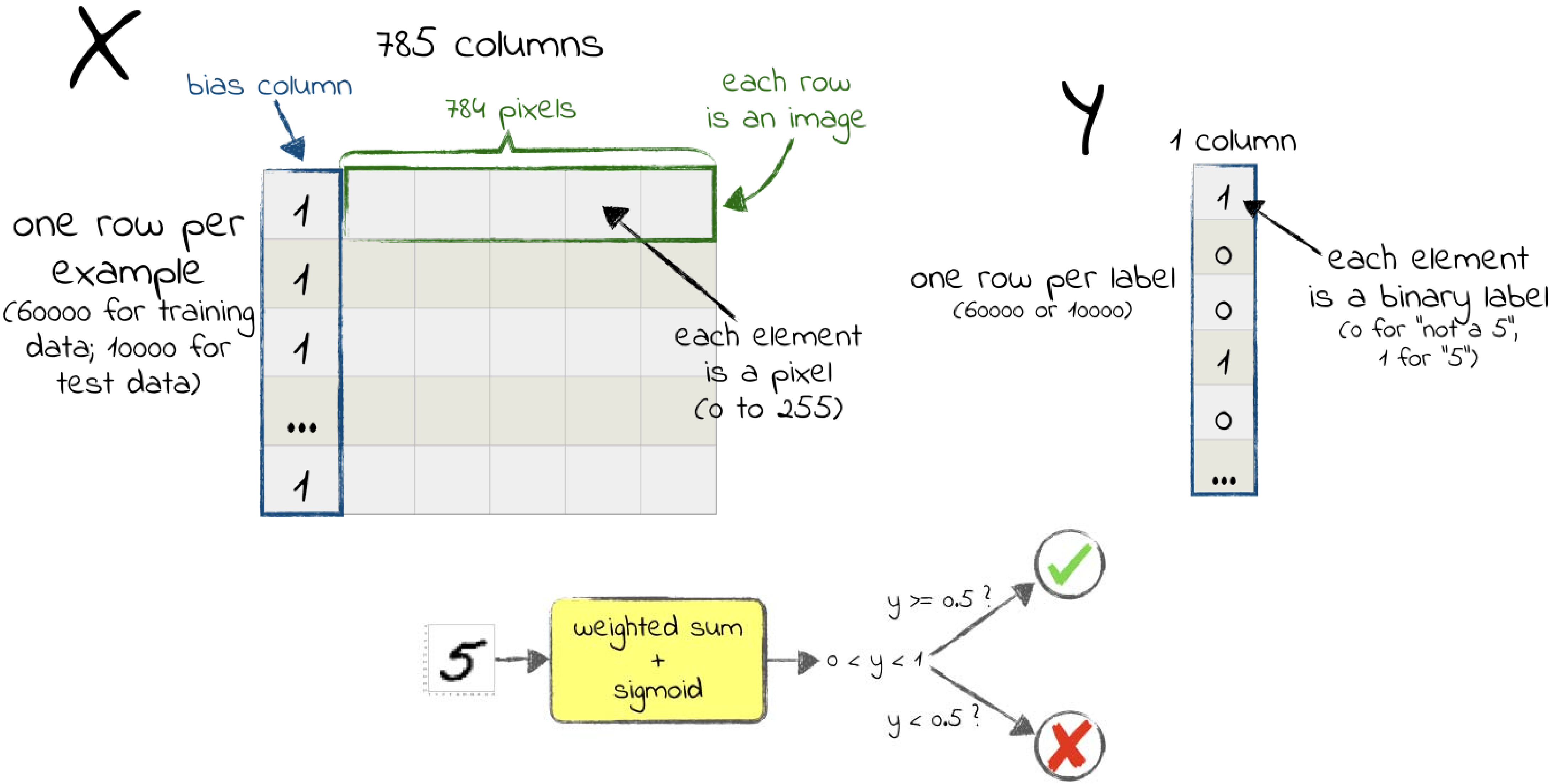


Binary Classification of MNIST

- Digits are made up of 28 by 28 grayscale pixels, each represented by one byte.
- In MNIST’s grayscale, 0 stands for “perfect background white,” and 255 stands for “perfect foreground black.”
- It contains 70,000 examples, neatly partitioned into 7,000 examples for each digit from 0 to 9.



MNIST Training and Testing Data

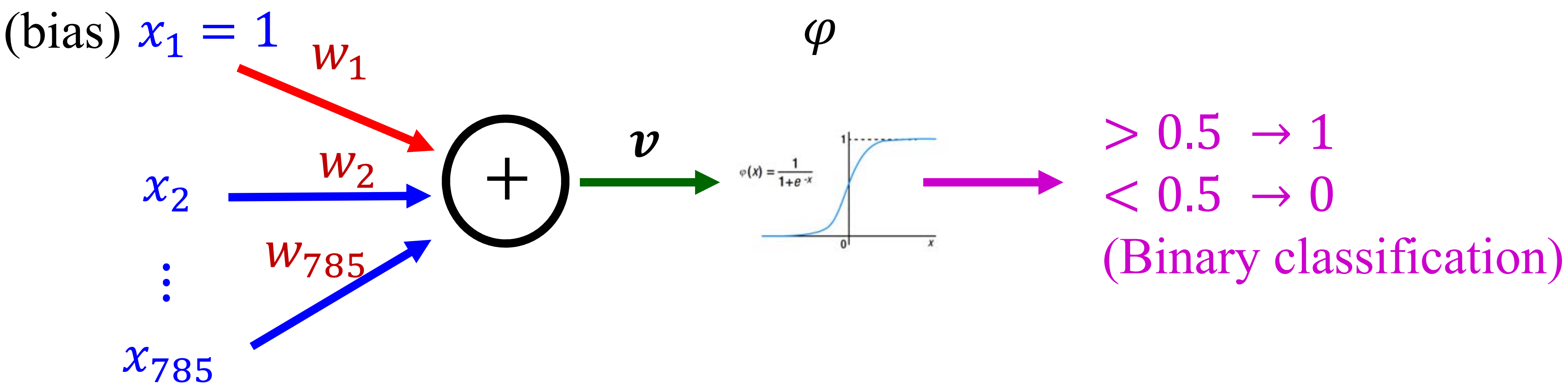


```python
import numpy as np
import gzip
import struct

def load_images(filename):
    # Open and unzip the file of images:
    with gzip.open(filename, 'rb') as f:
        # Read the header information into a bunch of variables
        _ignored, n_images, columns, rows = struct.unpack('>IIII', f.read(16))
        # Read all the pixels into a NumPy array of bytes:
        all_pixels = np.frombuffer(f.read(), dtype=np.uint8)
        # Reshape the pixels into a matrix where each line is an image:
        return all_pixels.reshape(n_images, columns * rows)

def prepend_bias(X):
    # Insert a column of 1s in the position 0 of X.
    # ("axis=1" stands for: "insert a column, not a row")
    return np.insert(X, 0, 1, axis=1)

# 60000 images, each 785 elements (1 bias + 28 * 28 pixels)
X_train = prepend_bias(load_images("../data/mnist/train-images-idx3-ubyte.gz"))

# 10000 images, each 785 elements, with the same structure as X_train
X_test = prepend_bias(load_images("../data/mnist/t10k-images-idx3-ubyte.gz"))
```

```python
def load_labels(filename):
    # Open and unzip the file of images:
    with gzip.open(filename, 'rb') as f:
        # Skip the header bytes:
        f.read(8)
        # Read all the labels into a list:
        all_labels = f.read()
        # Reshape the list of labels into a one-column matrix:
        return np.frombuffer(all_labels, dtype=np.uint8).reshape(-1, 1)

def encode_fives(Y):
    # Convert all 5s to 1, and everything else to 0
    return (Y == 5).astype(int)

# 60K labels, each with value 1 if the digit is a five, and 0 otherwise
Y_train = encode_fives(load_labels("../data/mnist/train-labels-idx1-ubyte.gz"))

# 10000 labels, with the same encoding as Y_train
Y_test = encode_fives(load_labels("../data/mnist/t10k-labels-idx1-ubyte.gz"))
```

reshape(-1, 1) means: “Arrange these data into a matrix with one column, and however many rows you need.”

Multiple inputs and single output for Binary Classification



$$v = x_1 w_1 + x_2 w_2 + x_{3w_3} + \cdots x_{785} w_{785} = \boldsymbol{xw}$$

$$\text{where } \boldsymbol{x} = [x_1 \ \cdots \ x_{785}],\ \boldsymbol{w} = \begin{bmatrix} w_1 \\ \vdots \\ w_{785} \end{bmatrix}$$

$y = \varphi(v)$, φ is the sigmoid activation function

When $b \equiv w_1 \neq 0$ and $y = \varphi(v)$ with CE loss

$$\hat{v}_1 = [1\ x_{12}\ \cdots\ x_{1785}]\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{785} \end{bmatrix} \Rightarrow\ e_1 = \varphi(\hat{v}_1) - y_1 = \hat{y}_1 - y_1$$

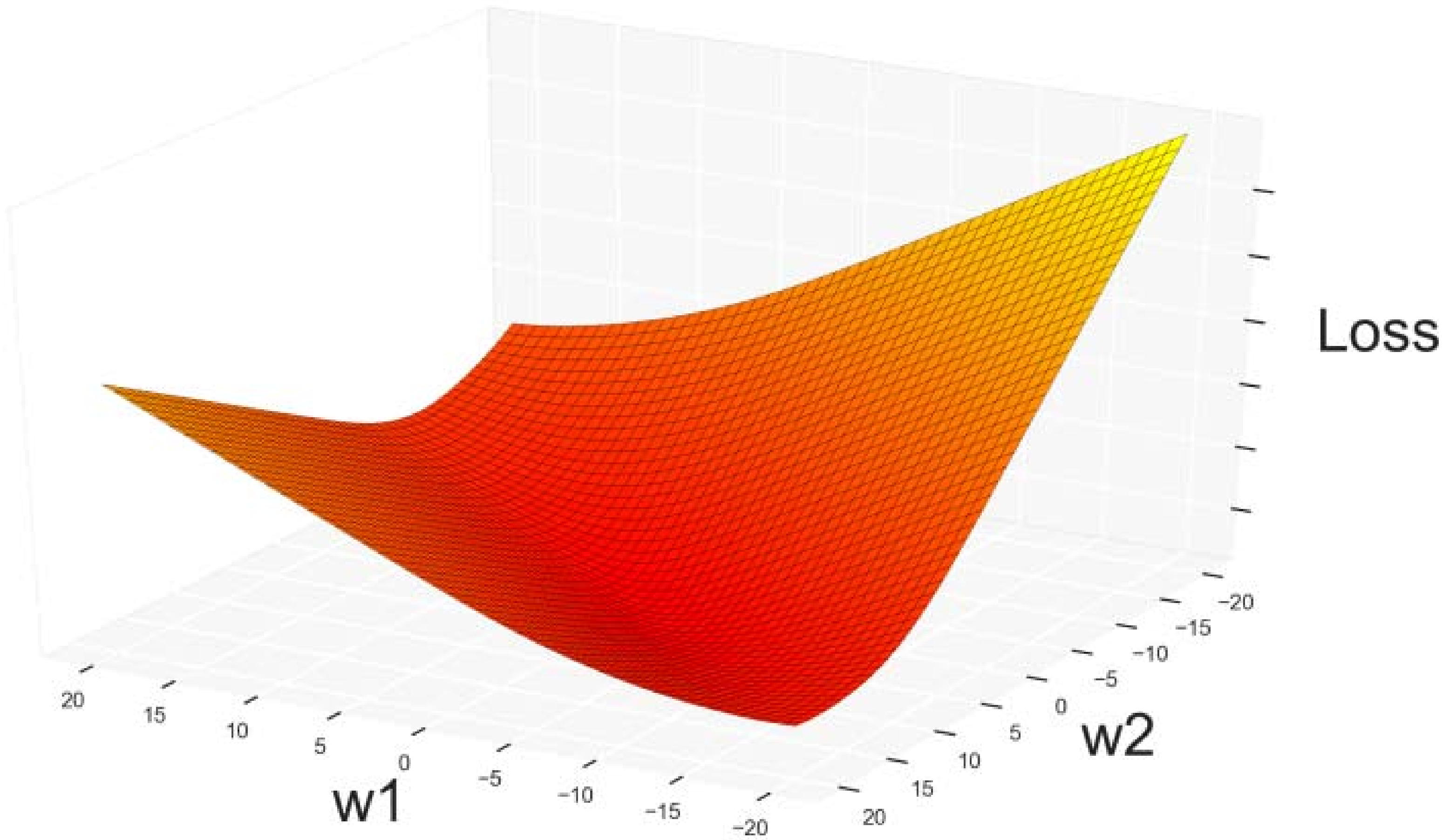
$$\vdots$$

$$\hat{v}_N = [1\ x_{N2} \cdots x_{N785}]\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_{785} \end{bmatrix} \Rightarrow\ e_N = \varphi(\hat{v}_N) - y_N = \hat{y}_N - y_N$$

$$\text{Loss}(\boldsymbol{w}) = \frac{-1}{N}\sum_{n=1}^{N}\left(y_n \ln(\hat{y}_n) + (1 - y_n)\ln(1 - \hat{y}_n)\right),$$

$$\hat{\boldsymbol{y}}_{N\times 1} = \varphi([\boldsymbol{1}_{N\times 1}\ \ \boldsymbol{X}_{N\times 784}] \times \boldsymbol{w}_{785\times 1})$$

Cross Entropy Error is Nice and Smooth



04_plot_losses.jpynb

Minimize Mean Cross Entropy Using Gradient Descent

- The loss function for output

$$\text{Loss}(\boldsymbol{w}) = \frac{-1}{N}\sum_{n=1}^{N}\left(y_n \ln(\hat{y}_n) + (1-y_n)\ln(1-\hat{y}_n)\right)$$

- Minimize the loss function *Loss* w.r.t $w_1, w_2, \cdots w_{785}$

$$\frac{\partial Loss}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial Loss}{\partial w_1} \\ \frac{\partial Loss}{\partial w_2} \\ \vdots \\ \frac{\partial Loss}{\partial w_{785}} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{n=1}^{N} e_n \\ \frac{1}{N}\sum_{n=1}^{N} e_n x_{n2} \\ \vdots \\ \frac{1}{N}\sum_{n=1}^{N} e_n x_{n785} \end{bmatrix} = \frac{1}{N}\left[\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 784}\right]^{\mathrm{T}}_{785\times \mathrm{N}} e_{N\times 1}$$

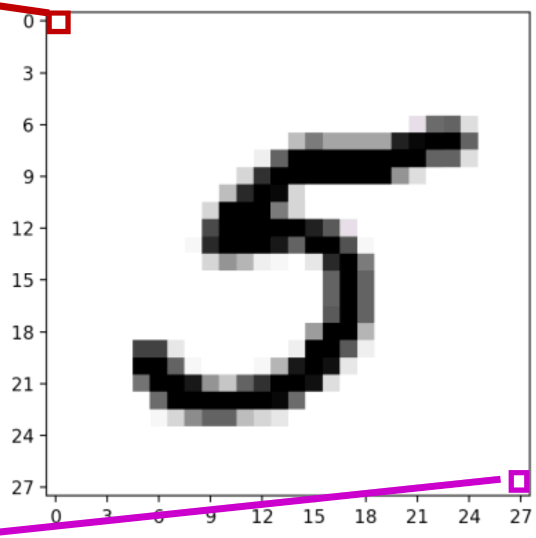
- The Batch Gradient Decent method

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha \frac{\partial Loss}{\partial \boldsymbol{w}}$$

$$[\mathbf{1}_{N\times1}\ \ \boldsymbol{X}_{N\times3}]^{\mathrm{T}}_{785\times\mathrm{N}} \quad \times \quad e_{N\times1}$$

$\boldsymbol{x}_1$ 1的列向量 共有 N 個

$\boldsymbol{x}_2$ N 個影像中左上角第一個畫素灰階形成的列向量



$\vdots$

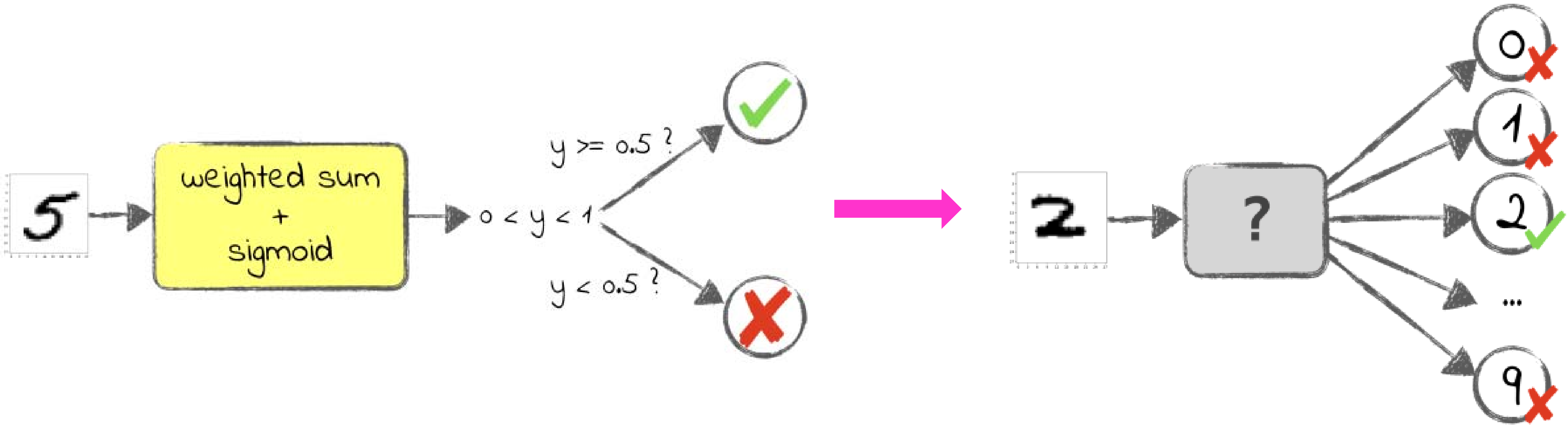
$\boldsymbol{x}_{785}$ N 個影像中右下角第一個畫素灰階形成的列向量

$$\begin{bmatrix} e_1 = \varphi(\hat{v}_1) - y_1 \\ e_2 = \varphi(\hat{v}_2) - y_2 \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ \vdots \\ e_N = \varphi(\hat{v}_N) - y_N \end{bmatrix}$$

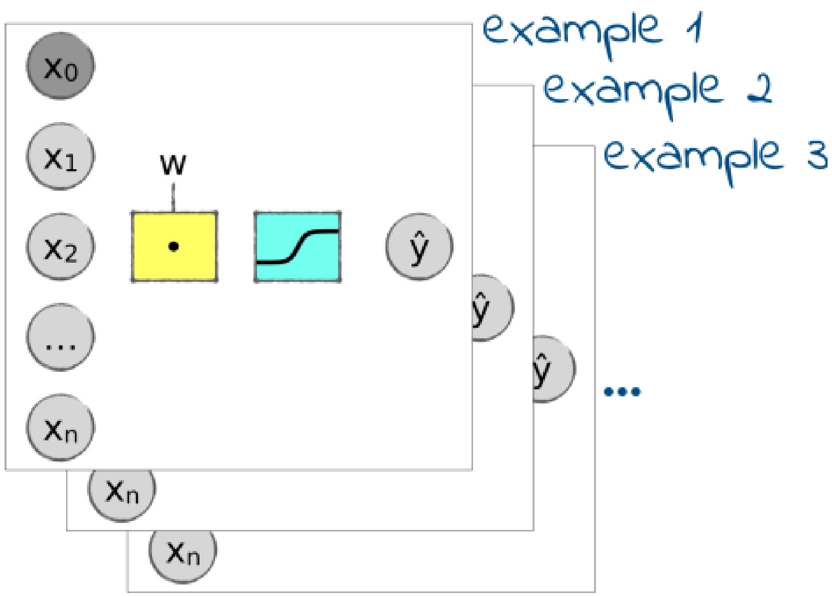
```python
import numpy as np

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def forward(X, w):
    weighted_sum = np.matmul(X, w)
    return sigmoid(weighted_sum)

def classify(X, w):
    return np.round(forward(X, w))

def loss(X, Y, w):
    y_hat = forward(X, w)
    first_term = Y * np.log(y_hat)
    second_term = (1 - Y) * np.log(1 - y_hat)
    return -np.average(first_term + second_term)

def gradient(X, Y, w):
    return np.matmul(X.T, (forward(X, w) - Y)) / X.shape[0]
```

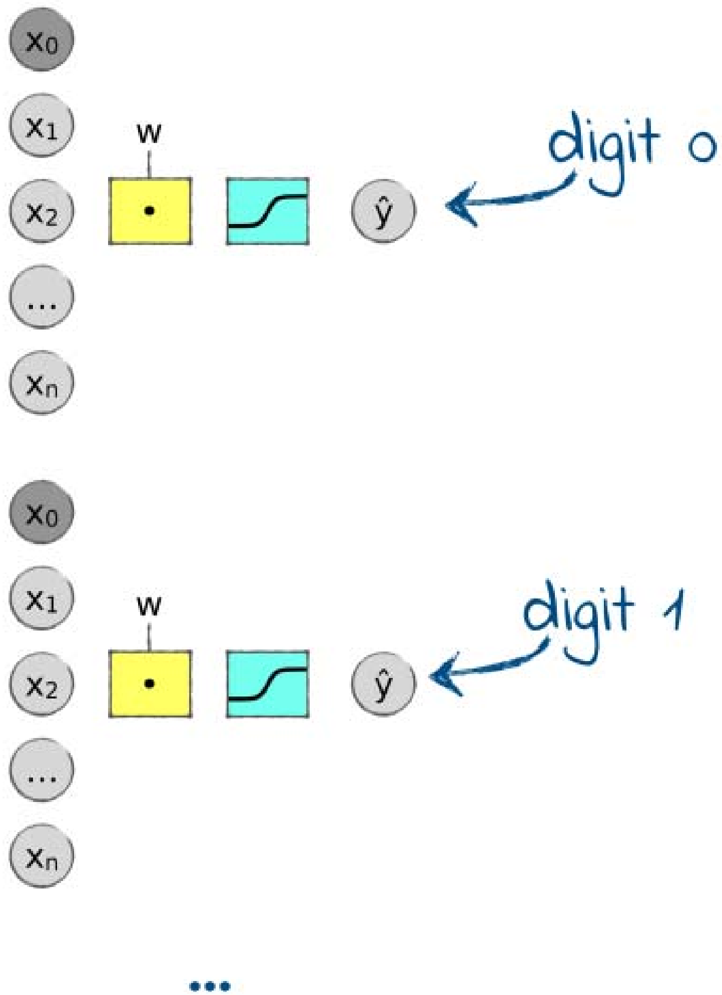
```python
def train(X, Y, iterations, lr):
    w = np.zeros((X.shape[1], 1))
    for i in range(iterations):
        print("Iteration %4d => Loss: %.20f" % (i, loss(X, Y, w)))
        w -= gradient(X, Y, w) * lr
    return w

def test(X, Y, w):
    total_examples = X.shape[0]
    correct_results = np.sum(classify(X, w) == Y)
    success_percent = correct_results * 100 / total_examples
    print("\nSuccess: %d/%d (%.2f%%)" %
          (correct_results, total_examples, success_percent))

w = train(data.X_train, data.Y_train, iterations=100, lr=1e-5)
test(data.X_test, data.Y_test, w)
```

Multiple Classification Using Sigmoid

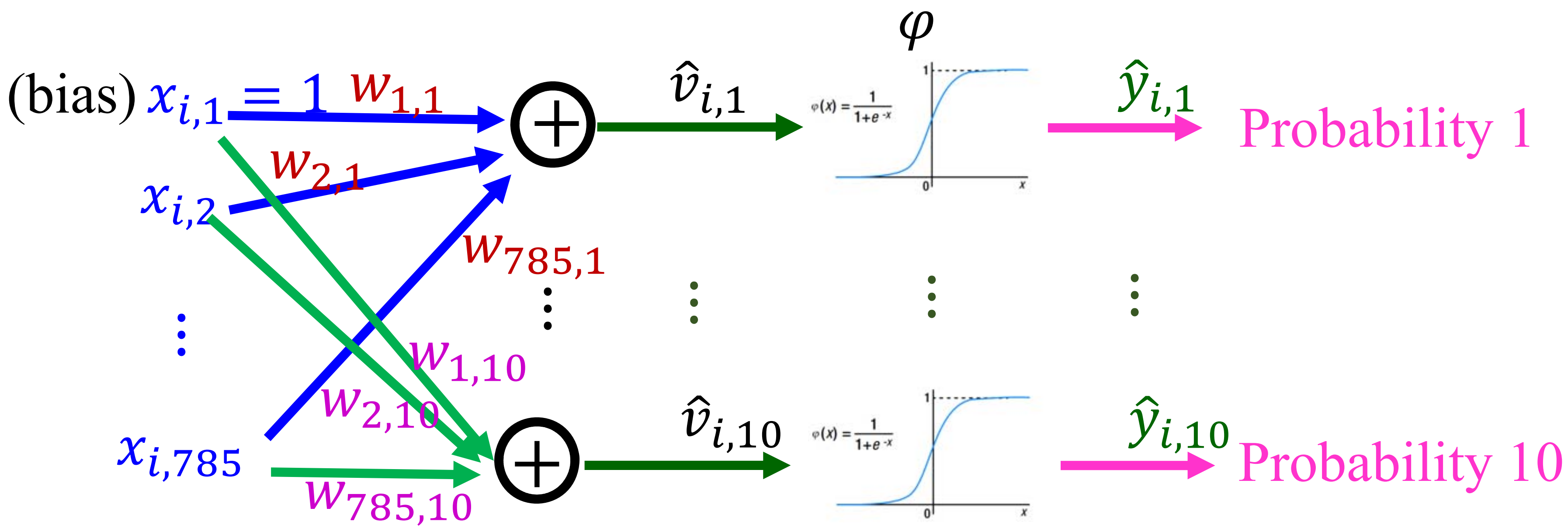


Multiple Classification Using Sigmoid





Multiple inputs and outputs for Multiple Classification



$$\hat{v}_{i,j} = x_{i,1}w_{1,j} + x_{i,2}w_{2,j} + x_{i,3}w_{3,j} + \cdots x_{i,785}w_{785,j} = \boldsymbol{x}_i\boldsymbol{w}_j,$$

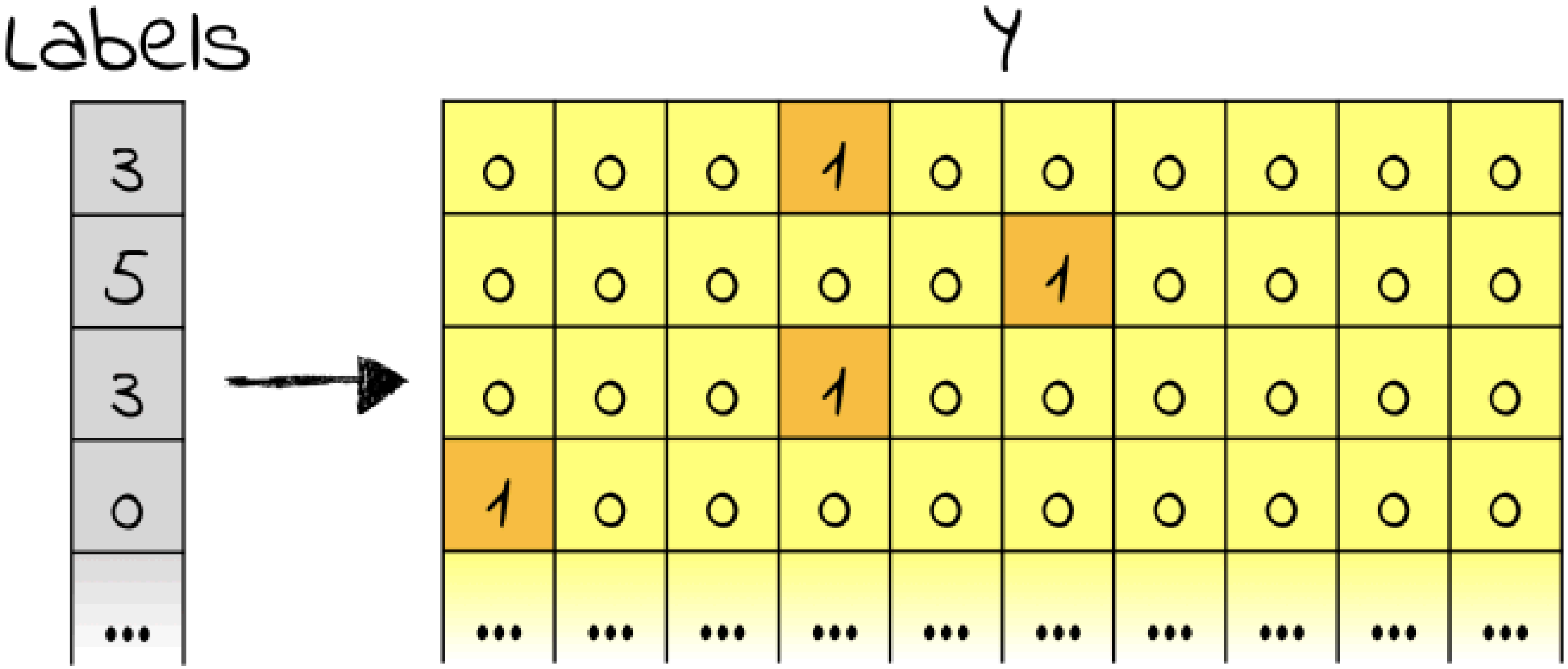
where $\boldsymbol{x}_i = \begin{bmatrix} x_{i,1} & \cdots & x_{i,785} \end{bmatrix}, \boldsymbol{w_j} = \begin{bmatrix} w_{1,j} \\ \vdots \\ w_{785,j} \end{bmatrix}, i = 1, \cdots, N, j = 1, \cdots 10$

$\hat{y}_{i,j} = \varphi(v_{i,j})$, φ is the sigmoid activation function

One-Hot Encoding

A possible output $\hat{y}_1, \cdots, \hat{y}_{10}$

"0"	"1"	"2"	"3"	"4"	"5"	"6"	"7"	"8"	"9"
0.111	0.005	0.787	0.170	0.001	0.176	0.352	0.001	0.073	0.003



```python
def one_hot_encode(Y):
    n_labels = Y.shape[0]
    n_classes = 10
    encoded_Y = np.zeros((n_labels, n_classes))
    for i in range(n_labels):
        label = Y[i]
        encoded_Y[i][label] = 1
    return encoded_Y
```

- one_hot_encode() initializes a matrix of zeros with one row per label, and one column per class
- Y.shape[0] means “the number of rows in Y”
- Then it walks through the matrix, flipping the “hot” values to 1.

```python
# 60K labels, each a single digit from 0 to 9
Y_train_unencoded = load_labels("../data/mnist/train-labels-idx1-ubyte.gz")

# 60K labels, each consisting of 10 one-hot encoded elements
Y_train = one_hot_encode(Y_train_unencoded)

# 10000 labels, each a single digit from 0 to 9
Y_test = load_labels("../data/mnist/t10k-labels-idx1-ubyte.gz")
```

When $b \equiv w_1 \neq 0$ and $\hat{y} = \varphi(\hat{v})$ with CE loss

$$\begin{bmatrix} 1 & x_{1,2} & \cdots & x_{1,785} \\ 1 & x_{2,2} & \cdots & x_{2,785} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N,2} & \cdots & x_{N,785} \end{bmatrix} \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,10} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,10} \\ \vdots & \vdots & \vdots & \vdots \\ w_{785,1} & w_{785,2} & \cdots & w_{785,10} \end{bmatrix} = \begin{bmatrix} \hat{v}_{1,1} & \cdots & \hat{v}_{1,10} \\ \hat{v}_{2,1} & \cdots & \hat{v}_{2,10} \\ \vdots & \vdots & \vdots \\ \hat{v}_{N,1} & \cdots & \hat{v}_{N,10} \end{bmatrix}_{N\times\mathbf{10}}$$

$$\Rightarrow \hat{\boldsymbol{v}}_{N\times 10} = [\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 784}] \times \boldsymbol{w}_{785\times 10}$$
$$\Rightarrow \hat{\boldsymbol{y}}_{N\times 10} = \varphi([\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 784}] \times \boldsymbol{w}_{785\times 10})$$

$$\Rightarrow \quad e_{1,j} = \varphi\left(\hat{v}_{1,j}\right) - y_{1,j} = \hat{y}_{1,j} - y_{1,j}, j = 1, \cdots 10$$
$$\vdots$$
$$\Rightarrow e_{N,j} = \varphi\left(\hat{v}_{N,j}\right) - y_{N,j} = \hat{y}_{N,j} - y_{N,j}, j = 1, \cdots 10$$
$$\Rightarrow \hat{\boldsymbol{e}}_{N\times 10} = \varphi([\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 784}] \times \boldsymbol{w}_{785\times 10}) - Y_{N\times 10}$$

Minimize Mean Cross Entropy Using Gradient Descent

- The loss function for output

$$\text{Loss}(\boldsymbol{w}) = \sum_{j=1}^{10}\left\{\frac{-1}{N}\sum_{n=1}^{N}\left(y_{n,j}\ln\left(\hat{y}_{n,j}\right) + \left(1-y_{n,j}\right)\ln\left(1-\hat{y}_{n,j}\right)\right)\right\}$$

Minimize $Loss(\boldsymbol{w})$ w.r.t $w_{1,1}, w_{2,1}, \cdots w_{785,1}, \cdots, w_{1,10}, w_{2,10}, \cdots w_{785,10}$

$$\frac{\partial Loss}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial Loss}{\partial w_{1,1}} & \frac{\partial Loss}{\partial w_{1,2}} & \cdots & \frac{\partial Loss}{\partial w_{1,10}} \\ \frac{\partial Loss}{\partial w_{2,1}} & \frac{\partial Loss}{\partial w_{2,2}} & \cdots & \frac{\partial Loss}{\partial w_{2,10}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial Loss}{\partial w_{785,1}} & \frac{\partial Loss}{\partial w_{785,2}} & \cdots & \frac{\partial Loss}{\partial w_{785,10}} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{n=1}^{N} e_{n,1} & \frac{1}{N}\sum_{n=1}^{N} e_{n,2} & \cdots & \frac{1}{N}\sum_{n=1}^{N} e_{n,10} \\ \frac{1}{N}\sum_{n=1}^{N} e_{n,1}x_{n,2} & \frac{1}{N}\sum_{n=1}^{N} e_{n,2}x_{n,2} & \cdots & \frac{1}{N}\sum_{n=1}^{N} e_{n,10}x_{n,2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{N}\sum_{n=1}^{N} e_{n,1}x_{n,785} & \frac{1}{N}\sum_{n=1}^{N} e_{n,2}x_{n,785} & \cdots & \frac{1}{N}\sum_{n=1}^{N} e_{n,10}x_{n,785} \end{bmatrix}$$

$$= \frac{1}{N}\left[\mathbf{1}_{N\times 1}\quad \boldsymbol{X}_{N\times 284}\right]^{\mathrm{T}}_{785\times \mathrm{N}}\boldsymbol{e}_{N\times 10}$$

- The steepest (gradient) decent method

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha\frac{\partial Loss}{\partial \boldsymbol{w}}$$

```python
import numpy as np

def sigmoid(z):
    return 1 / (1 + np.exp(-z))

def forward(X, w):
    weighted_sum = np.matmul(X, w)
    return sigmoid(weighted_sum)

def classify(X, w):
    y_hat = forward(X, w)
    labels = np.argmax(y_hat, axis=1)
    return labels.reshape(-1, 1)

def loss(X, Y, w):
    y_hat = forward(X, w)
    first_term = Y * np.log(y_hat)
    second_term = (1 - Y) * np.log(1 - y_hat)
    return -np.sum(first_term + second_term) / X.shape[0]
```

weighted_sum.shape → (N, 10)

labels.shape → (N, 1)

→ (N, 10)

→ (N, 10)

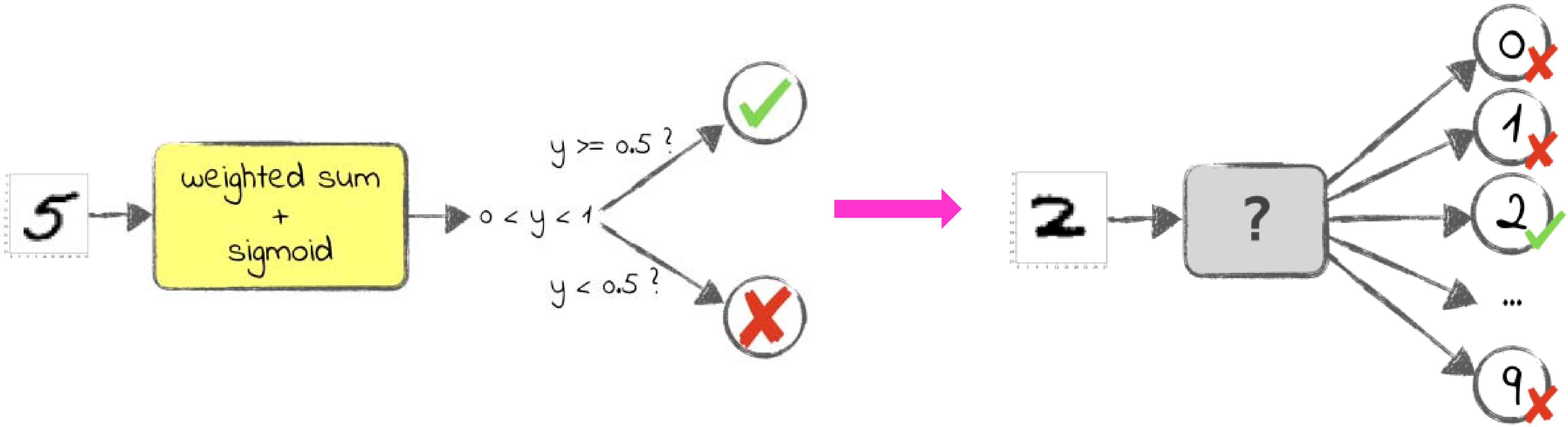
```python
def gradient(X, Y, w):
    return np.matmul(X.T, (forward(X, w) - Y)) / X.shape[0]
```

→ (785, 10)

```python
def report(iteration, X_train, Y_train, X_test, Y_test, w):
    matches = np.count_nonzero(classify(X_test, w) == Y_test)
    n_test_examples = Y_test.shape[0]
    matches = matches * 100.0 / n_test_examples
    training_loss = loss(X_train, Y_train, w)
    print("%d - Loss: %.20f, %.2f%%" % (iteration, training_loss, matches))

def train(X_train, Y_train, X_test, Y_test, iterations, lr):
    w = np.zeros((X_train.shape[1], Y_train.shape[1]))
    for i in range(iterations):
        report(i, X_train, Y_train, X_test, Y_test, w)
        w -= gradient(X_train, Y_train, w) * lr
    report(iterations, X_train, Y_train, X_test, Y_test, w)
    return w
```

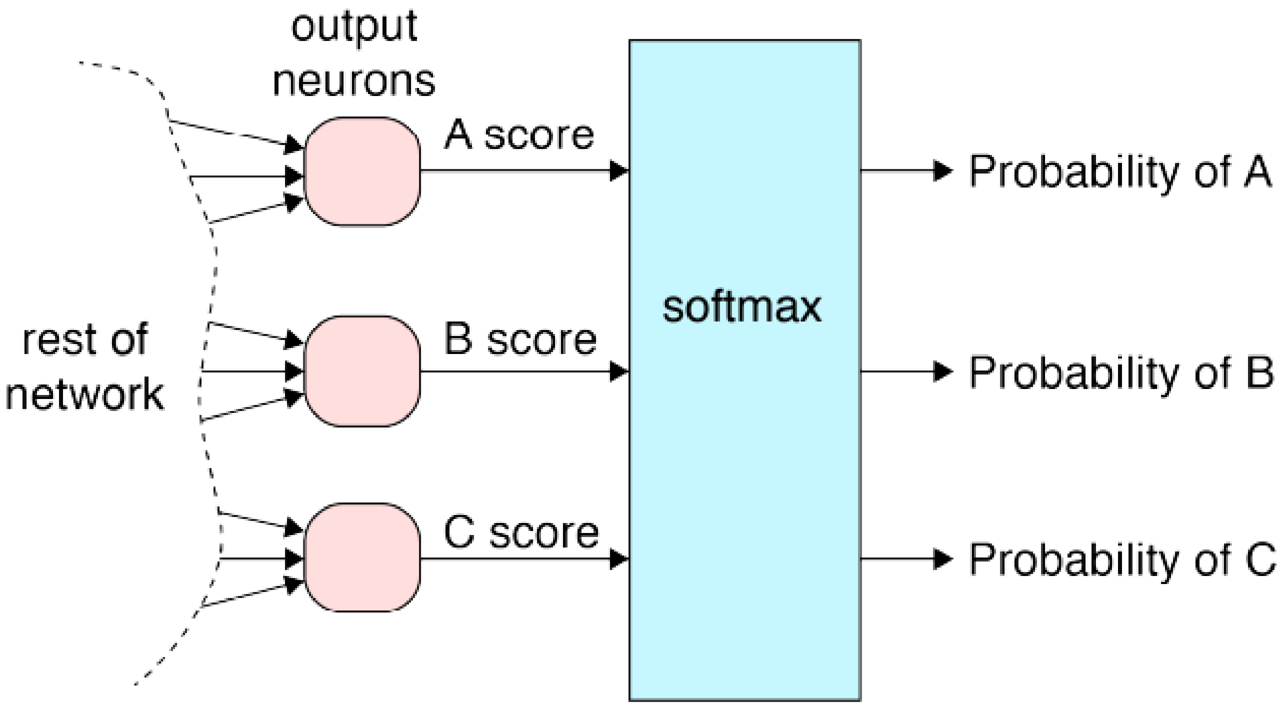
```python
w = train(X_train, Y_train,
          X_test, Y_test,
          iterations=200, lr=1e-5)
```

Multiple Classification Using Softmax



The softmax function turns all the scores into probabilities

In general, multiclass classifiers employ the softmax function as the activation function of the output node.



The softmax function turns all the scores into probabilities

- The output from the *i*-th output node of the softmax function is：

$$y_i = \sigma(v_i) = \frac{e^{v_i}}{e^{v_1} + e^{v_2} + e^{v_3} + \cdots + e^{v_M}} = \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}}$$

v_i is the weighted sum of the *i*-th output node.

M is the number of output nodes.

$$\sigma(v_1) + \sigma(v_2) + \sigma(v_3) + \cdots + \sigma(v_M) = 1$$

- Example:

$$v = \begin{bmatrix} 2 \\ 1 \\ 0.1 \end{bmatrix} \Rightarrow \sigma(v) = \begin{bmatrix} \dfrac{e^2}{e^2 + e^1 + e^{0.1}} \\ \dfrac{e^1}{e^2 + e^1 + e^{0.1}} \\ \dfrac{e^{0.1}}{e^2 + e^1 + e^{0.1}} \end{bmatrix} = \begin{bmatrix} 0.6590 \\ 0.2424 \\ 0.0986 \end{bmatrix}$$

Derivative of Softmax function

$$y_i = \sigma(v_i) = \frac{e^{v_i}}{e^{v_1} + e^{v_2} + e^{v_3} + \cdots + e^{v_M}} = \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}} = e^{v_i}\left(\sum_{k=1}^{M} e^{v_k}\right)^{-1}$$

$$\frac{\partial y_i}{\partial v_i} = \frac{\partial e^{v_i}}{\partial v_i}\left(\sum_{k=1}^{M} e^{v_k}\right)^{-1} + e^{v_i}\frac{\partial\left(\sum_{k=1}^{M} e^{v_k}\right)^{-1}}{\partial v_i},$$

$$= \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}} - \frac{e^{v_i}e^{v_i}}{\left(\sum_{k=1}^{M} e^{v_k}\right)^2}$$

$$= \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}}\left(1 - \frac{e^{v_i}}{\sum_{k=1}^{M} e^{v_k}}\right)$$

$$= \sigma(v_i)\big(1 - \sigma(v_i)\big)$$

$$= y_i(1 - y_i)$$

note $\frac{d(f(x)g(x))}{dx} = \frac{d(f(x))}{dx} g(x) + f(x)\frac{d(g(x))}{dx}$

Derivative of the Binary Cross-Entropy Loss

$$\frac{\partial L}{\partial \hat{y}} = \frac{\partial(-y\ln(\hat{y}) - (1-y)\ln(1-\hat{y}))}{\partial \hat{y}}$$

$$= -y\frac{1}{\hat{y}} - (1-y)\frac{-1}{1-\hat{y}}$$

$$= \frac{-y(1-\hat{y}) + (1-y)\hat{y}}{\hat{y}(1-\hat{y})}$$

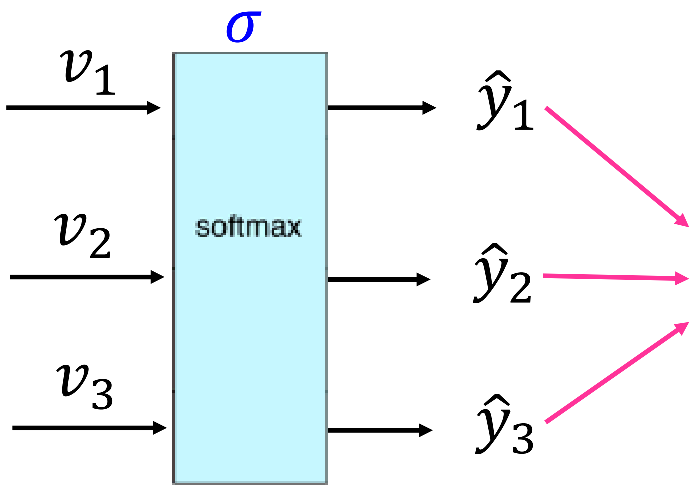
$$= \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})}$$

$$\hat{y} = \sigma(\hat{v})$$

$$\frac{\partial \hat{y}}{\partial \hat{v}} = \sigma(\hat{v})\big(1 - \sigma(\hat{v})\big) = \hat{y}(1-\hat{y})$$

$$\frac{\partial L}{\partial \hat{y}}\frac{\partial \hat{y}}{\partial \hat{v}} = \frac{-(y-\hat{y})}{\hat{y}(1-\hat{y})} \times \hat{y}(1-\hat{y}) = -(y-\hat{y}) = \hat{y} - y = \sigma(\hat{v}) - y \rightarrow e$$

Derivative of the Categorical Cross-Entropy Loss



$$L(v_1, v_2, v_2) = -\sum_{k=1}^{3} y_k \ln(\hat{y}_k),$$

One-Hot Encoding

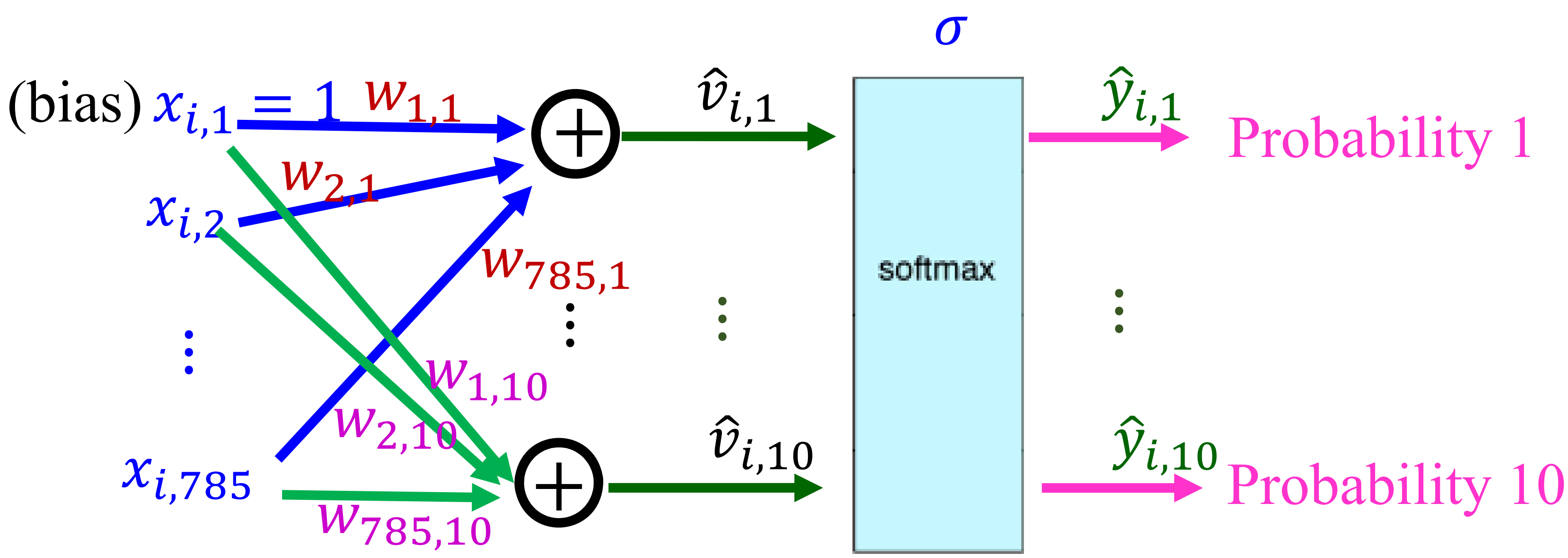
$$\begin{aligned}&[y_1 \quad y_2 \quad y_3]\\ &= [1 \quad 0 \quad 0],\\ &= [0 \quad 1 \quad 0],\\ &= [0 \quad 0 \quad 1]\end{aligned}$$

$$\frac{\partial L}{\partial v_j} = \frac{\partial}{\partial v_j}\left(-\sum_{k=1}^{3} y_k \ln(\hat{y}_k)\right) = -\sum_{k=1}^{3} y_k \frac{\partial \ln(\hat{y}_k)}{\partial v_j}, \qquad \hat{y}_k = \frac{e^{v_k}}{e^{v_1} + e^{v_2} + e^{v_3}}$$

$$\ln(\hat{y}_k) = v_k - \ln\left(\sum_{i=1}^{3} e^{v_i}\right) \Rightarrow \frac{\partial \ln(\hat{y}_k)}{\partial v_j} = \frac{\partial v_k}{\partial v_j} - \frac{\partial}{\partial v_j}\ln\left(\sum_{i=1}^{3} e^{v_i}\right) = 1\{k = j\} - \frac{e^{v_j}}{\sum_{i=1}^{3} e^{v_i}}$$

$$\Rightarrow \frac{\partial L}{\partial v_j} = -\sum_{k=1}^{3} y_k \left(1\{k = j\} - \hat{y}_j\right) = -\sum_{k=1}^{3} y_k 1\{k = j\} + \hat{y}_j \sum_{k=1}^{3} y_k = -y_j + \hat{y}_j \times 1 \equiv e_j$$

Multiple inputs and outputs for Multiple Classification



$$\hat{v}_{i,j} = x_{i,1}w_{1,j} + x_{i,2}w_{2,j} + x_{i,3}w_{3,j} + \cdots x_{i,785}w_{785,j} = \boldsymbol{x}_i\boldsymbol{w}_j,$$

where $\boldsymbol{x}_i = \begin{bmatrix} x_{i,1} & \cdots & x_{i,785} \end{bmatrix}, \boldsymbol{w}_j = \begin{bmatrix} w_{1,j} \\ \vdots \\ w_{785,j} \end{bmatrix}, i = 1, \cdots, N, j = 1, \cdots 10$

$\hat{y}_{i,j} = \sigma(v_{i,j})$, σ is the softmax activation function

When $b \equiv w_1 \neq 0$ and $\hat{y} = \sigma(\hat{v})$ with CE loss

$$\begin{bmatrix} 1 & x_{1,2} & \cdots & x_{1,785} \\ 1 & x_{2,2} & \cdots & x_{2,785} \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_{N,2} & \cdots & x_{N,785} \end{bmatrix} \begin{bmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,10} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,10} \\ \vdots & \vdots & \vdots & \vdots \\ w_{785,1} & w_{785,2} & \cdots & w_{785,10} \end{bmatrix} = \begin{bmatrix} \hat{v}_{1,j} & \cdots & \hat{v}_{1,10} \\ \hat{v}_{2,j} & \cdots & \hat{v}_{2,10} \\ \vdots & \vdots & \vdots \\ \hat{v}_{N,j} & \cdots & \hat{v}_{N,10} \end{bmatrix}_{N\times\mathbf{10}}$$

$$\Rightarrow \hat{\boldsymbol{v}}_{N\times 10} = [\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 784}] \times \boldsymbol{w}_{785\times 10}$$

$$\Rightarrow \hat{\boldsymbol{y}}_{N\times 10} = \sigma([\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 784}] \times \boldsymbol{w}_{785\times 10})$$

$$\Rightarrow e_{1,j} = \sigma(\hat{v}_{1,j}) - y_{1,j} = \hat{y}_{1,j} - y_{1,j}, j = 1, \cdots 10$$

$$\vdots$$

$$\Rightarrow e_{N,j} = \sigma(\hat{v}_{N,j}) - y_{N,j} = \hat{y}_{N,j} - y_{N,j}, j = 1, \cdots 10$$

$$\Rightarrow \hat{\boldsymbol{e}}_{N\times 10} = \sigma([\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 784}] \times \boldsymbol{w}_{785\times 10})$$

Minimize Mean Cross Entropy Using Gradient Descent

- The loss function for output

$$\text{Loss}(\boldsymbol{w}) = \sum_{j=1}^{10} \left\{ \frac{-1}{N} \sum_{n=1}^{N} y_{n,j} \ln(\hat{y}_{n,j}) \right\}, \qquad \frac{\partial Loss}{\partial w_{i,j}} = \frac{\partial Loss}{\partial \hat{v}_{n,j}} \frac{\partial \hat{v}_{n,j}}{\partial w_{i,j}}$$

Minimize $Loss(\boldsymbol{w})$ w.r.t $w_{1,1}, w_{2,1}, \cdots w_{785,1}, \cdots, w_{1,10}, w_{2,10}, \cdots w_{785,10}$

$$\frac{\partial Loss}{\partial \boldsymbol{w}} = \begin{bmatrix} \frac{\partial Loss}{\partial w_{1,1}} & \frac{\partial Loss}{\partial w_{1,2}} & \cdots & \frac{\partial Loss}{\partial w_{1,10}} \\ \frac{\partial Loss}{\partial w_{2,1}} & \frac{\partial Loss}{\partial w_{2,2}} & \cdots & \frac{\partial Loss}{\partial w_{2,10}} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{\partial Loss}{\partial w_{785,1}} & \frac{\partial Loss}{\partial w_{785,2}} & \cdots & \frac{\partial Loss}{\partial w_{785,10}} \end{bmatrix} = \begin{bmatrix} \frac{1}{N}\sum_{n=1}^{N} e_{n,1} & \frac{1}{N}\sum_{n=1}^{N} e_{n,2} & \cdots & \frac{1}{N}\sum_{n=1}^{N} e_{n,10} \\ \frac{1}{N}\sum_{n=1}^{N} e_{n,1}x_{n,2} & \frac{1}{N}\sum_{n=1}^{N} e_{n,2}x_{n,2} & \cdots & \frac{1}{N}\sum_{n=1}^{N} e_{n,10}x_{n,2} \\ \vdots & \vdots & \vdots & \vdots \\ \frac{1}{N}\sum_{n=1}^{N} e_{n,1}x_{n,785} & \frac{1}{N}\sum_{n=1}^{N} e_{n,2}x_{n,785} & \cdots & \frac{1}{N}\sum_{n=1}^{N} e_{n,10}x_{n,785} \end{bmatrix}$$

$$= \frac{1}{N} [\mathbf{1}_{N\times 1} \quad \boldsymbol{X}_{N\times 284}]^{\mathrm{T}}_{785\times \mathrm{N}} \boldsymbol{e}_{N\times 10}$$

- The Batch Gradient Decent method

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \alpha \frac{\partial Loss}{\partial \boldsymbol{w}}$$

Homework # 8-1, 8-2

- Re-write the Lecture 08_4_MNIST_10_classes_sigmoid_classification.jpynb

in object-oriented format.

Hint: Homework 8_1_MINST_10_classes_classification_using_logistic_oop_to_do.jpynb

- Re-write the Lecture 08_5_MNIST_10_classes_softmax_classification.jpynb

in object-oriented format.

Hint: Homework 8_2_MINST_10_classes_classification_using_softmax_oop_to_do.jpynb

Deadline of Homework #8: 2022/11/21 3:30pm