

Softmax Regression

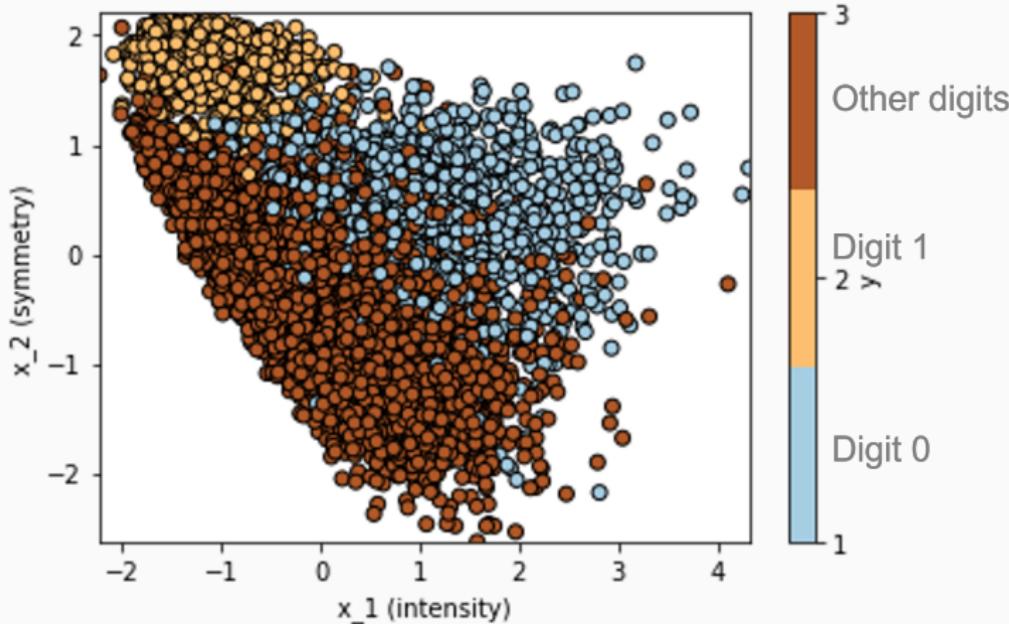
Last update: October 28, 2024



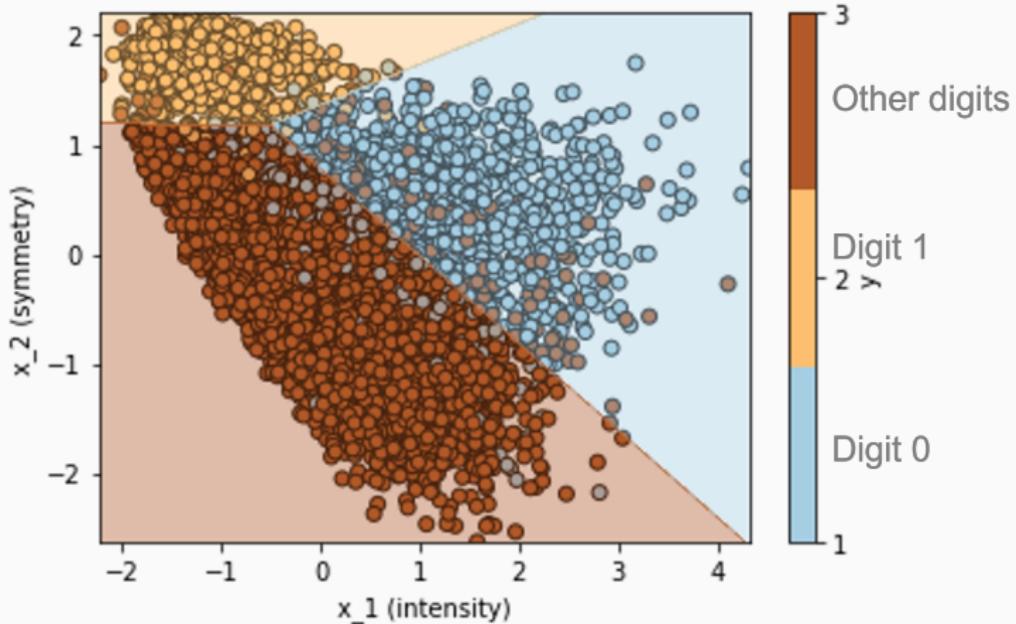
fit@hcmus

KHOA CÔNG NGHỆ THÔNG TIN
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Softmax Regression — idea



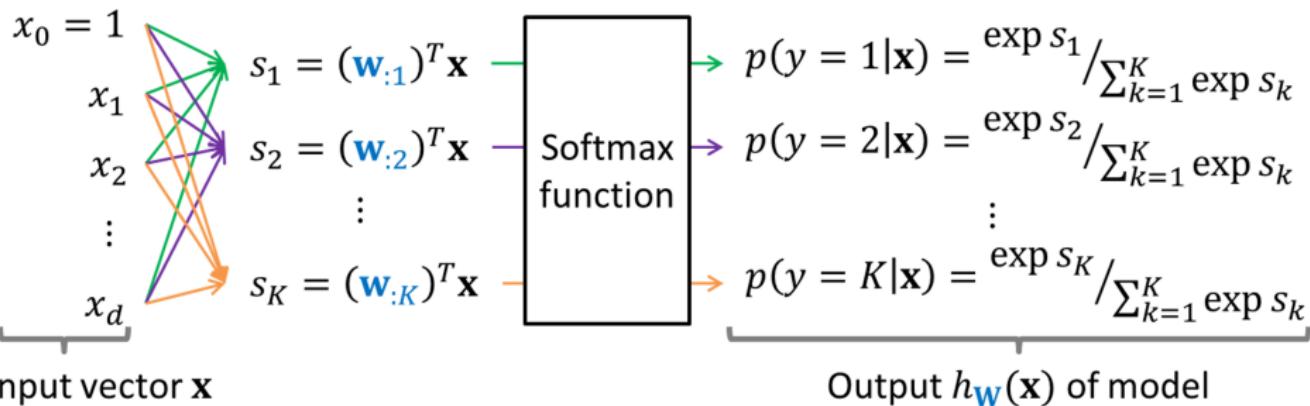
Look at training data and choose *lines* (planes in 3D space, hyper-planes in >3D space), each line separates two classes well



Softmax Regression — model form

$\mathbf{w}_{:k}$: k^{th} column of matrix \mathbf{W}
 $s_k \in (-\infty, \infty)$

$p(y = k|x) \in [0, 1]$
and $\sum_{k=1}^K p(y = k|x) = 1$



Softmax Regression — training

Given training data: $\{(\mathbf{x}^{(1)}, y^{(1)}), \dots, (\mathbf{x}^{(N)}, y^{(N)})\}$
where $\mathbf{x}^{(n)} \in \mathbb{R}^{d+1}$ and $y^{(n)} \in \{1, 2, \dots, K\}$

We should choose model $h_{\mathbf{W}}(\mathbf{x})$ with $\mathbf{W} = ?$

Steps to find \mathbf{W} :

1. Define $E_{train}(\mathbf{W})$ — error function on training data
2. $\min_{\mathbf{W}} E_{train}(\mathbf{W})$

Step 1: Define $E_{train}(\mathbf{W})$ — error function on training data

People often use *Cross-entropy Error*

Cross-entropy Error on a data point (\mathbf{x}, y) :

$$e(h_{\mathbf{W}}(\mathbf{x}), y) = -\ln(h_{\mathbf{W}}(\mathbf{x})_y)$$

Equivalently:

$$e(h_{\mathbf{W}}(\mathbf{x}), y) = -\sum_{j=1}^K \text{onehot}(y)_j \ln(h_{\mathbf{W}}(\mathbf{x})_j)$$

Intuition: with correct class y ,

- If $h_{\mathbf{W}}(\mathbf{x})_y \rightarrow 0$ then $e \rightarrow \infty$
- If $h_{\mathbf{W}}(\mathbf{x})_y \rightarrow 1$ then $e \rightarrow 0$

Cross-entropy Error on training data:

$$E_{train}(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^N e(h_{\mathbf{W}}(\mathbf{x}^{(n)}), y^{(n)})$$

Step 2: $\min_{\mathbf{w}} E_{train}(\mathbf{w})$

We can use iterative gradient-based algorithms such as Gradient Descent

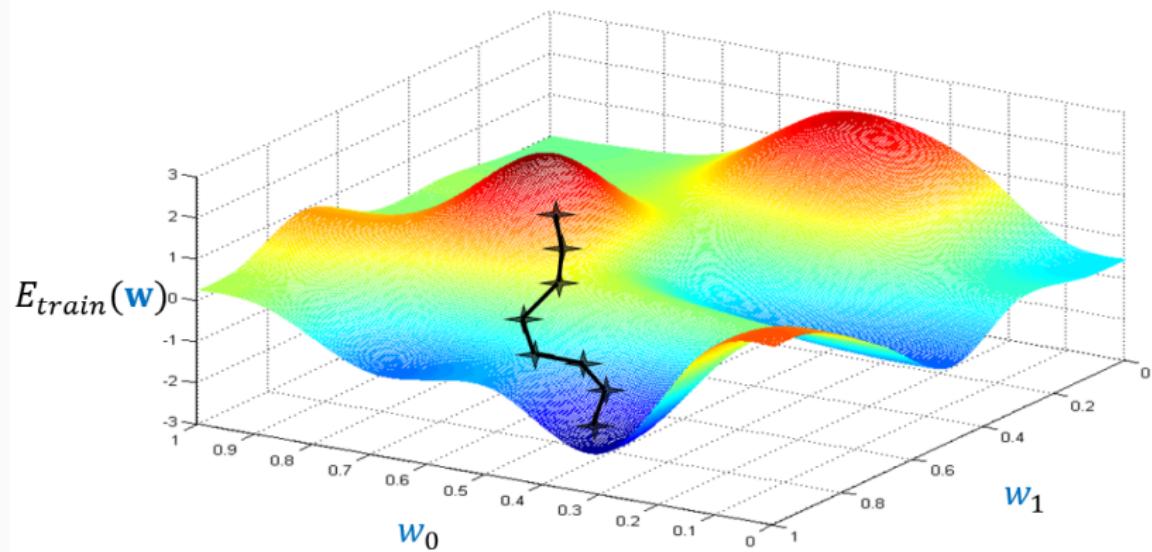


Image source: Andrew Ng, Machine Learning, MOOC

Step 2: $\min_{\mathbf{W}} E_{train}(\mathbf{W})$

Gradient Descent algorithm:

1. Initialize \mathbf{W}
2. Repeat until termination criteria are satisfied:

$$\mathbf{W} \leftarrow \mathbf{W} - \alpha \nabla E_{train}(\mathbf{W})$$

where:

- α is a number > 0 ; it is called “learning rate” and is a hyper-parameter which partially controls step size
- $\nabla E_{train}(\mathbf{W})$ is a matrix containing partial derivatives of E_{train} w.r.t each w_{ij} of \mathbf{W} ; it is called “gradient”

$$\nabla E_{train}(\mathbf{W}) = \begin{bmatrix} \frac{\partial E_{train}}{\partial w_{11}} & \dots & \frac{\partial E_{train}}{\partial w_{1K}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E_{train}}{\partial w_{(d+1)1}} & \dots & \frac{\partial E_{train}}{\partial w_{(d+1)K}} \end{bmatrix}$$

$$\nabla E_{train}(\mathbf{w}) = ?$$

After doing difficult math with pencil and paper:

$$\frac{\partial E_{train}}{\partial w_{ij}} = \frac{1}{N} \sum_{n=1}^N x_i^{(n)} (h_{\mathbf{w}}(\mathbf{x}^{(n)})_j - \text{onehot}(y^{(n)})_j)$$

Based on this formula, we can write Python/Numpy code to compute $\nabla E_{train}(\mathbf{w})$ with **3 nested loops** :-)

$$\nabla E_{train}(\mathbf{W}) = ?$$

After thinking hard from previous formula:

$$\nabla E_{train}(\mathbf{W}) = \frac{1}{N} \sum_{n=1}^N \mathbf{x}^{(n)} \left(h_{\mathbf{W}}(\mathbf{x}^{(n)}) - \text{onehot}(y^{(n)}) \right)^T$$

Based on this formula, we can write Python/Numpy code to compute $\nabla E_{train}(\mathbf{W})$ with 1 loop

$$\nabla E_{train}(\mathbf{W}) = ?$$

After thinking hard from previous formula (your job):

$$\nabla E_{train}(\mathbf{W}) = \dots \mathbf{X} \dots \hat{\mathbf{Y}} \dots \mathbf{Y} \dots$$

where:

$$\mathbf{X} = \begin{bmatrix} - & (\mathbf{x}^{(1)})^T & - \\ - & \vdots & - \\ - & (\mathbf{x}^{(N)})^T & - \end{bmatrix} \quad \hat{\mathbf{Y}} = \begin{bmatrix} - & (h_{\mathbf{W}}(\mathbf{x}^{(1)}))^T & - \\ - & \vdots & - \\ - & (h_{\mathbf{W}}(\mathbf{x}^{(N)}))^T & - \end{bmatrix}$$
$$\mathbf{Y} = \begin{bmatrix} - & (onehot(y^{(1)}))^T & - \\ - & \vdots & - \\ - & (onehot(y^{(N)}))^T & - \end{bmatrix}$$

Now we can write Python/Numpy code to compute
 $\nabla E_{train}(\mathbf{W})$ with **0 loop** :-)

Done!