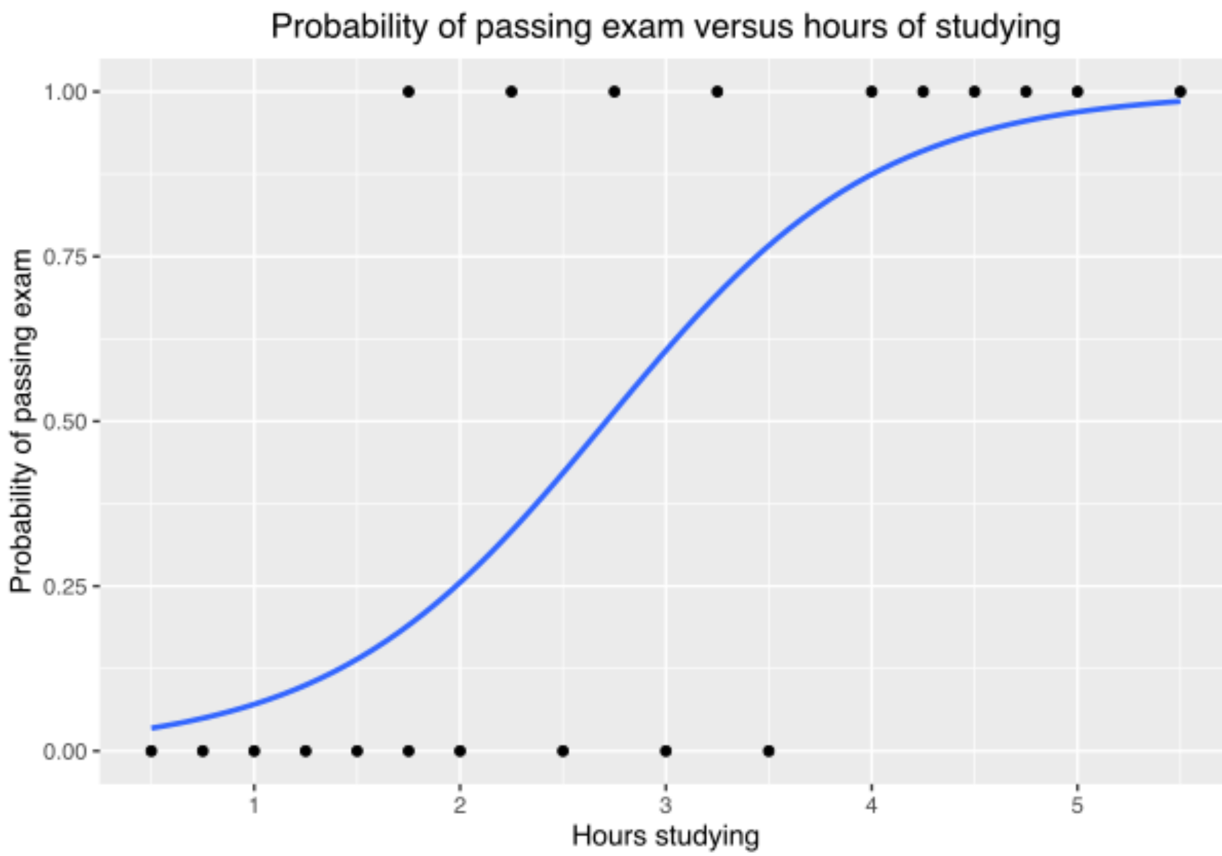
CS440/ECE448 Lecture 11: Softmax

Mark Hasegawa-Johnson, 2/2024



Outline

- Linear Classifier: Review
- Probabilities: Softmax and logistic sigmoid
- Training criterion: Cross-entropy

Linear classifier

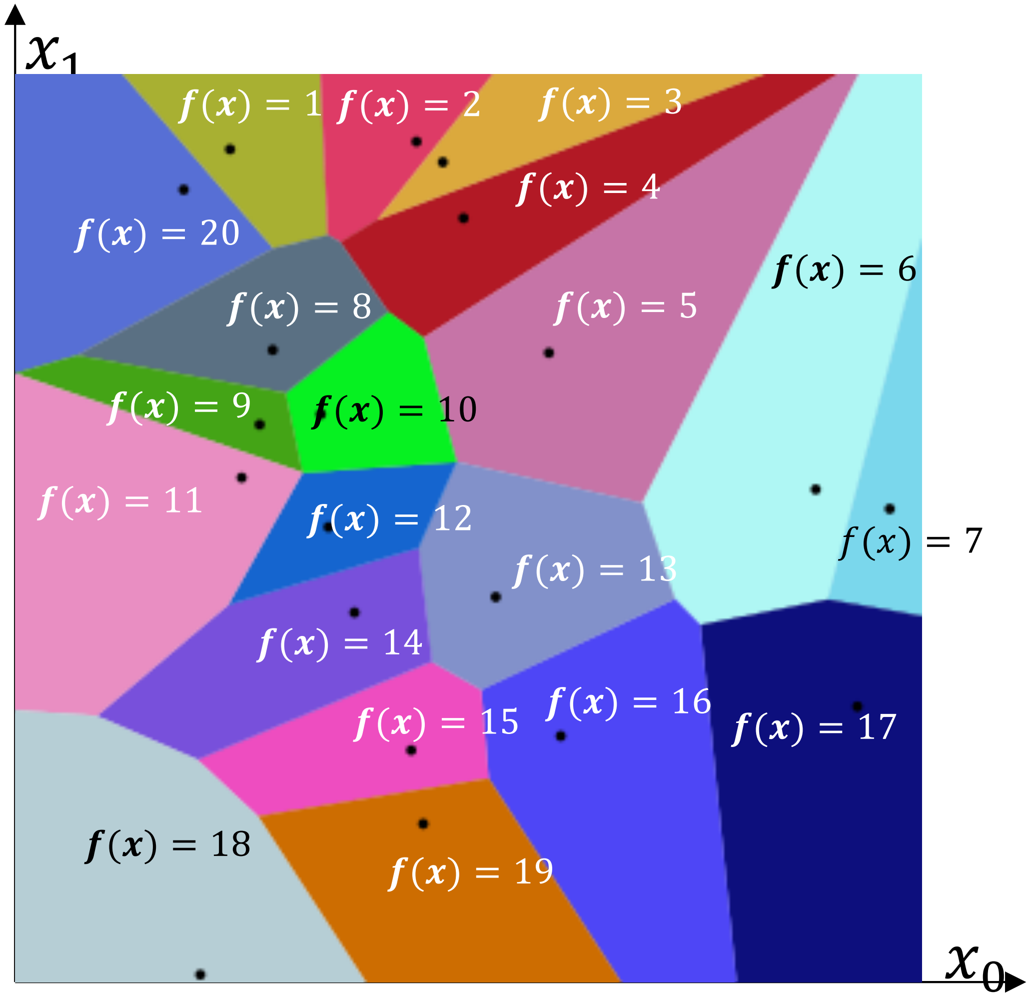
In a linear classifier,

$$f(\boldsymbol{x}) = \operatorname{argmax} \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

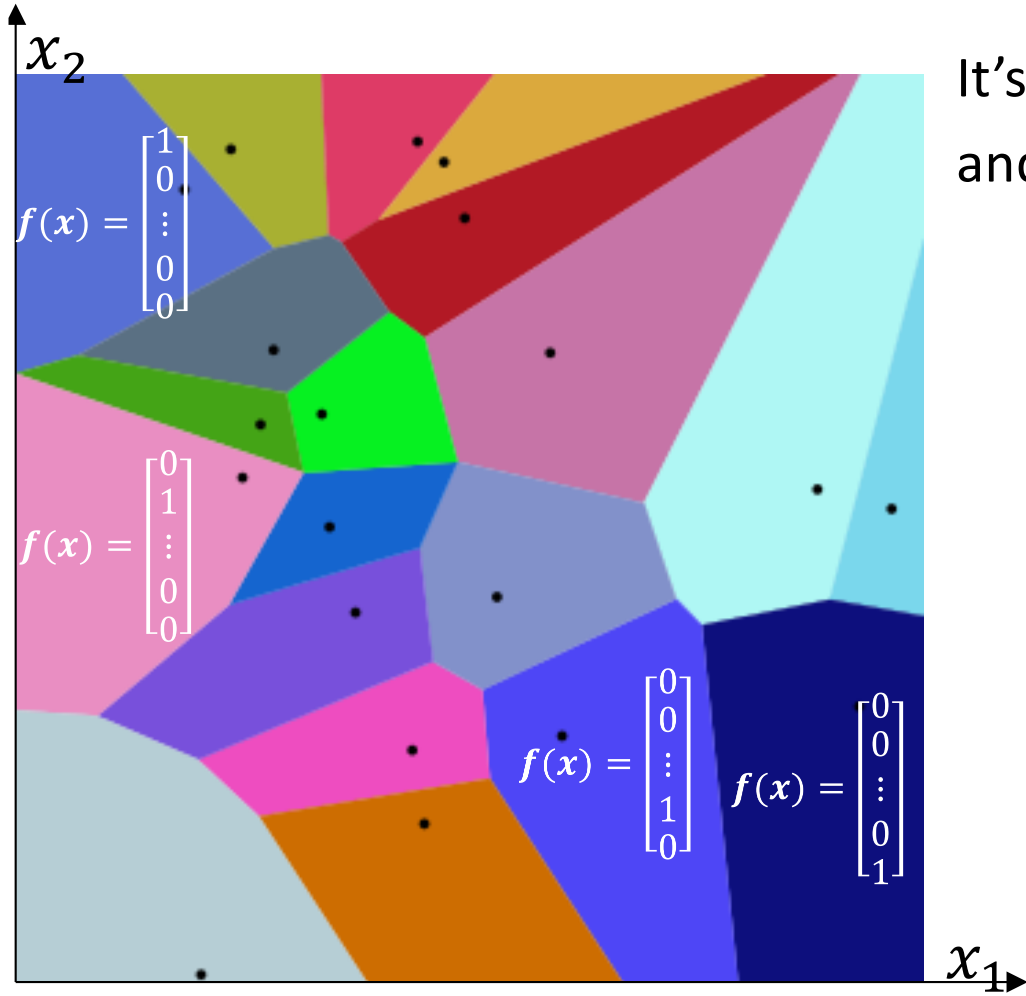
The boundary between class k and class l is the line (or plane, or hyperplane) given by the equation

$$(\boldsymbol{w}_k - \boldsymbol{w}_l)^T \boldsymbol{x} + (b_k - b_l) = 0$$

… where $\boldsymbol{w}_k{}^T$ is the k^th^ row of $\boldsymbol{W}$, and b_k is the k^th^ element of $\boldsymbol{b}$.



One-hot vectors



It's often useful to convert the labels $f(\boldsymbol{x})$ and y into one-hot vectors $\boldsymbol{f}(\boldsymbol{x})$ and $\boldsymbol{y}$:

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_v \end{bmatrix} = \begin{bmatrix} \mathbb{I}_{y=1} \\ \vdots \\ \mathbb{I}_{y=v} \end{bmatrix} \in \{0,1\}^v,$$

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_v(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \mathbb{I}_{f(\boldsymbol{x})=1} \\ \vdots \\ \mathbb{I}_{f(\boldsymbol{x})=v} \end{bmatrix} \in \{0,1\}^v$$

The perceptron learning algorithm

1. Compute the classifier output $\hat{y} = \underset{\boldsymbol{k}}{\operatorname{argmax}}\left(\boldsymbol{w}_k^T \boldsymbol{x} + b_k\right)$
2. Update the weight vectors as:

$$\boldsymbol{w}_k \leftarrow \begin{cases} \boldsymbol{w}_k - \eta \boldsymbol{x} & k = \hat{y} \\ \boldsymbol{w}_k + \eta \boldsymbol{x} & k = y \\ \boldsymbol{w}_k & \text{otherwise} \end{cases}$$

where $\eta \approx 0.01$ is the learning rate.

Outline

- Linear Classifier: Review
- Probabilities: Softmax and logistic sigmoid
- Training criterion: Cross-entropy

Key idea: $f_c(\boldsymbol{x})$ =posterior probability of class $\boldsymbol{c}$

- A perceptron has a one-hot output vector, in which $f_c(\boldsymbol{x}) = 1$ if the neural net thinks *c* is the most likely value of *y*, and 0 otherwise
- A softmax computes $f_c(\boldsymbol{x}) \approx \Pr(Y = c|\boldsymbol{x})$. The conditions for this to be true are:
  1. It needs to satisfy the axioms of probability:
  $$0 \leq f_c(\boldsymbol{x}) \leq 1, \qquad \sum_{c=1}^{v} f_c(\boldsymbol{x}) = 1$$
  2. The weight matrix, $\boldsymbol{W}$, is trained using a loss function that encourages $\boldsymbol{f}(\boldsymbol{x})$ to approximate posterior probability of the labels on some training dataset:
  $$f_c(\boldsymbol{x}) \approx \Pr(Y = c|\boldsymbol{x})$$

Softmax satisfies the axioms of probability

- Axiom #1, probabilities are non-negative ($f_k(\boldsymbol{x}) \geq 0$). There are many ways to do this, but one way that works is to choose:

$$f_c(\boldsymbol{x}) \propto \exp(\boldsymbol{w}_c^T \boldsymbol{x} + b_c)$$

- Axiom #2, probabilities should sum to one ($\sum_{k=1}^{v} f_k(\boldsymbol{x}) = 1$). This can be done by normalizing:

$$f_c(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_c^T \boldsymbol{x} + b_c)}{\sum_{k=0}^{V-1} \exp(\boldsymbol{w}_k^T \boldsymbol{x} + b_k)}$$

The softmax function

This is called the softmax function:

$$\boldsymbol{f}(\boldsymbol{x}) = [f_1(\boldsymbol{x}), \dots, f_v(\boldsymbol{x})]^T$$

$$f_c(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_c^T \boldsymbol{x} + b_c)}{\sum_{k=1}^{v} \exp(\boldsymbol{w}_k^T \boldsymbol{x} + b_k)}$$

…where $\boldsymbol{w}_k^T$ is the kth row of the matrix $\boldsymbol{W}$.

Quiz

Go to https://us.prairielearn.com/pl/course_instance/147925/assessment/2397335, and try the quiz!

The logistic sigmoid function

For a two-class classifier, we don't really need the vector label. If we define $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_2$ and $b = b_1 - b_2$, then the softmax simplifies to:

$$\boldsymbol{f}(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}) = \begin{bmatrix} \Pr(Y = 1|\boldsymbol{x}) \\ \Pr(Y = 2|\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \frac{1}{1+e^{-(\boldsymbol{w}^T x+b)}} \\ \frac{e^{-(\boldsymbol{w}^T x+b)}}{1+e^{-(\boldsymbol{w}^T x+b)}} \end{bmatrix} = \begin{bmatrix} \sigma(\boldsymbol{w}^T \boldsymbol{x} + b) \\ 1 - \sigma(\boldsymbol{w}^T \boldsymbol{x} + b) \end{bmatrix}$$

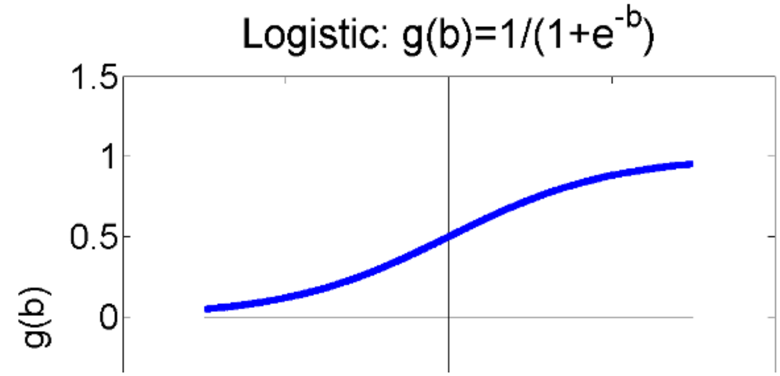
… so instead of the softmax, we use a scalar function called the logistic sigmoid function:

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

This function is called sigmoid because it is S-shaped.

For $z \rightarrow -\infty$, $\quad \sigma(z) \rightarrow 0$

For $z \rightarrow +\infty$, $\quad \sigma(z) \rightarrow 1$



Outline

- Linear Classifier: Review
- Probabilities: Softmax and logistic sigmoid
- Training criterion: Cross-entropy

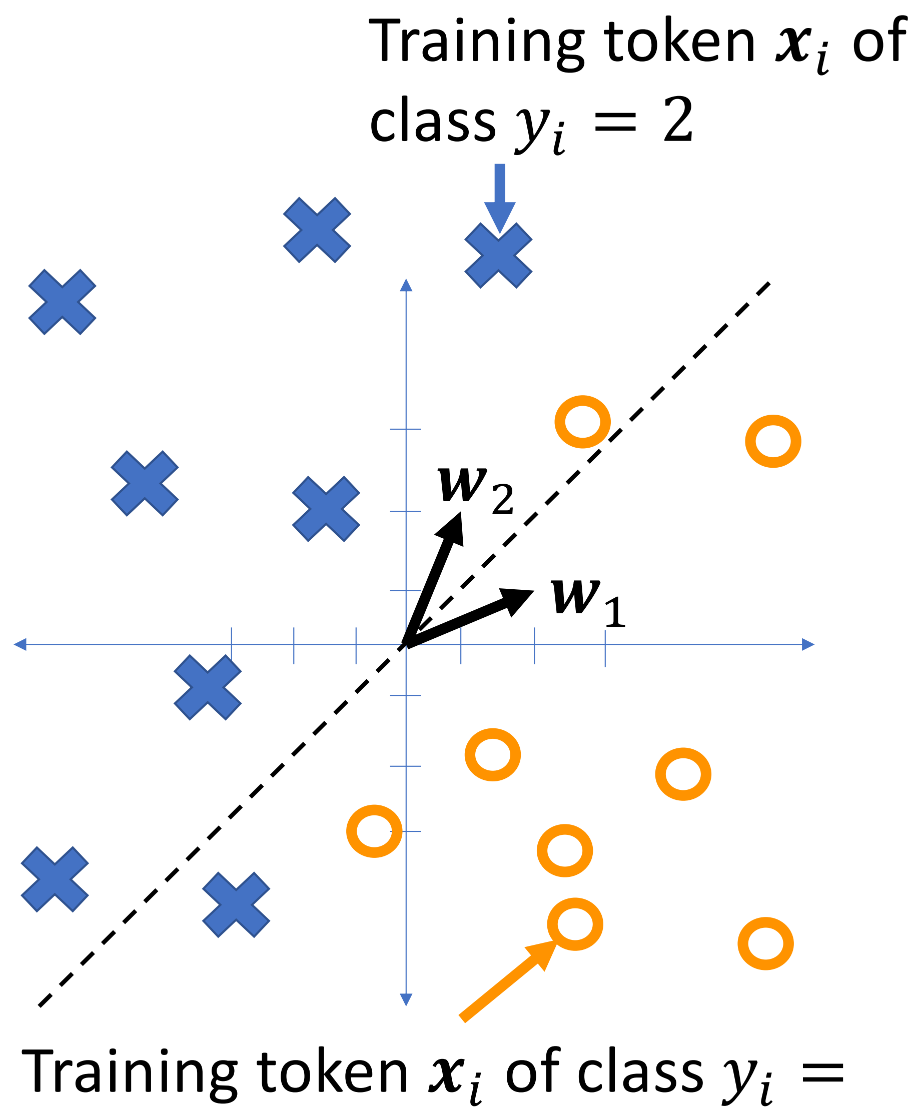
Gradient descent

Suppose we have training tokens (x_i, y_i), and we have some initial class vectors w_1 and w_2. We want to update them as

$$\boldsymbol{w}_1 \leftarrow \boldsymbol{w}_1 - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_1}$$

$$\boldsymbol{w}_2 \leftarrow \boldsymbol{w}_2 - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_2}$$

…where $\mathcal{L}$ is some loss function. What loss function makes sense?



Zero-one loss function

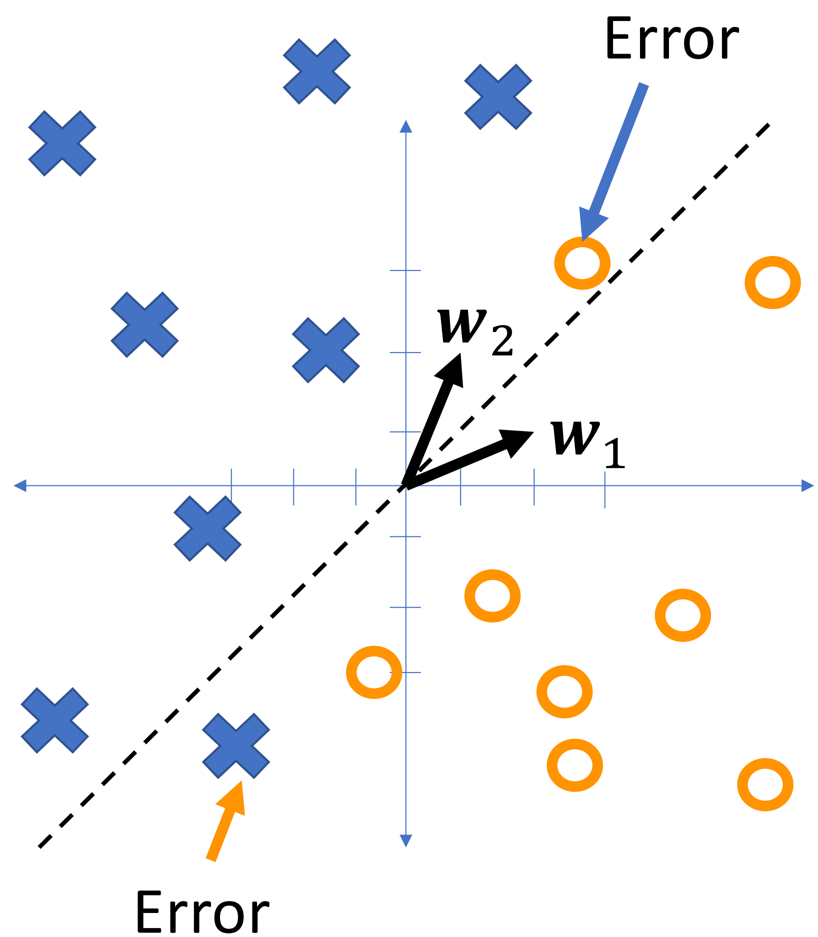
The most obvious loss function for a classifier is its classification error rate,

$$\mathcal{L} = \frac{1}{n}\sum_{i=1}^{n} \ell(f(\boldsymbol{x}_i), y_i)$$

Where $\ell(\hat{y}, y)$ is the zero-one loss function,

$$\ell(f(\boldsymbol{x}), y) = \begin{cases} 0 & f(\boldsymbol{x}) = y \\ 1 & f(\boldsymbol{x}) \neq y \end{cases}$$

The problem with zero-one loss is that it's not differentiable.



A loss function that learns probabilities

Suppose we have a softmax output, so we want $f_c(\boldsymbol{x}) \approx \Pr(Y = c|\boldsymbol{x})$. We can train this by learning $\boldsymbol{W}$ and $\boldsymbol{b}$ to maximize the probability of the training corpus. If we assume all training tokens are independent, we get:

$$\boldsymbol{W}, \boldsymbol{b} = \underset{\boldsymbol{W},\boldsymbol{b}}{\operatorname{argmax}} \prod_{i=1}^{n} \Pr(Y = y_i|\boldsymbol{x}_i) = \underset{\boldsymbol{W},\boldsymbol{b}}{\operatorname{argmax}} \sum_{i=1}^{\boldsymbol{n}} \ln \Pr(Y = y_i|\boldsymbol{x}_i)$$

But remember that $f_c(\boldsymbol{x}) \approx \Pr(Y = c|\boldsymbol{x})$! Therefore, maximizing the log probability of training data is the same as minimizing the cross entropy between the neural net and the ground truth:

$$\boldsymbol{W}, \boldsymbol{b} = \underset{\boldsymbol{W},\boldsymbol{b}}{\operatorname{argmin}} \frac{1}{n} \sum_{i=1}^{n} \mathcal{L}_i\,, \qquad \mathcal{L}_i = -\log f_{y_i}(\boldsymbol{x}_i)$$

Cross-entropy

This loss function:

$$\mathcal{L} = -\ln f_y(\boldsymbol{x})$$

is called cross-entropy. It measures the difference in randomness between:

- Truth: $Y = y$ with probability 1.0, $\ln(1.0) = 0$, minus the
- Neural net estimate: $Y = y$ with probability $f_y(\boldsymbol{x})$.
- Thus $\mathcal{L} = 0 - \ln f_y(\boldsymbol{x})$

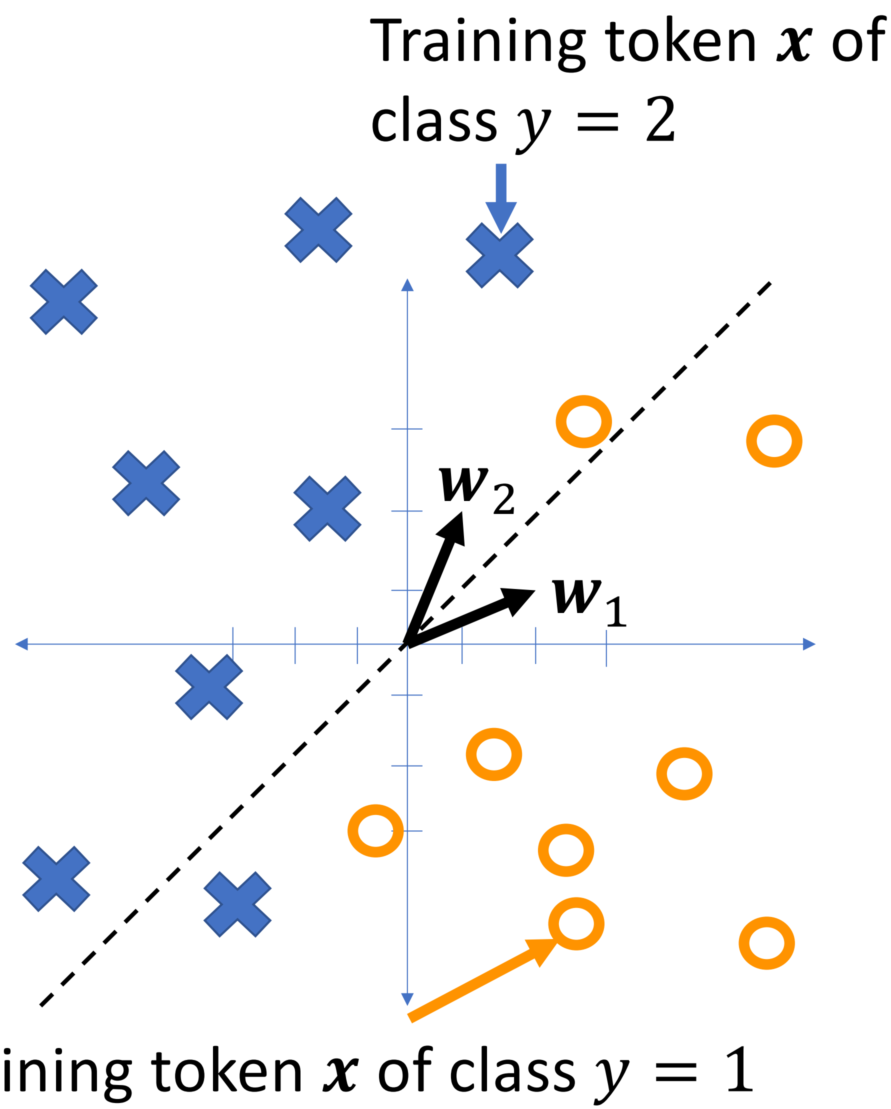
Stochastic gradient descent

Suppose we have a training example $(\boldsymbol{x}, y)$. We want to find

$$\boldsymbol{w}_c \leftarrow \boldsymbol{w}_c - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_c}$$

Now we know that $\mathcal{L} = -\ln f_y(\boldsymbol{x})$,

and $f_y(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_y^T \boldsymbol{x} + b_y)}{\sum_{k=1}^{v} \exp(\boldsymbol{w}_k^T \boldsymbol{x} + b_k)}$. What is $\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_c}$?



Gradient of the cross-entropy of a softmax

Suppose we define $z_c = \boldsymbol{w}_c^T \boldsymbol{x} + b_c$. Then we can write:

$$\mathcal{L} = -\ln f_y(\boldsymbol{x}) = -\ln\left(\frac{e^{z_y}}{\sum_{k=1}^{v} e^{z_k}}\right) = \ln\left(\sum_{k=1}^{v} e^{z_k}\right) - z_y$$

…and…

$$\frac{\partial \mathcal{L}}{\partial z_c} = \begin{cases} \frac{e^{z_c}}{\sum_{k=1}^{v} e^{z_k}} - 1 & c = y \\ \frac{e^{z_c}}{\sum_{k=1}^{v} e^{z_k}} & c \neq y \end{cases}$$

Gradient of the cross-entropy of the softmax

Since we have these definitions:

$$\mathcal{L} = -\ln f_y(\boldsymbol{x}), \qquad f_y(\boldsymbol{x}) = \frac{\exp(z_y)}{\sum_{k=1}^{v} \exp(z_k)}, \qquad z_c = \boldsymbol{w}_c^T \boldsymbol{x} + b_c$$

Then:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_c} = \left(\frac{\partial \mathcal{L}}{\partial z_c}\right)\left(\frac{\partial z_c}{\partial \boldsymbol{w}_c}\right) = \left(\frac{\partial \mathcal{L}}{\partial z_c}\right)\boldsymbol{x}$$

…where:

$$\frac{\partial \mathcal{L}}{\partial z_c} = \begin{cases} f_c(\boldsymbol{x}_i) - 1 & c = y \\ f_c(\boldsymbol{x}_i) - 0 & c \neq y \end{cases}$$

Similarity to linear regression

For linear regression, we had:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = \epsilon \boldsymbol{x}, \qquad \epsilon = f(\boldsymbol{x}) - y$$

For the softmax classifier with cross-entropy loss, we have

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_c} = \epsilon_c \boldsymbol{x}$$

$$\epsilon_c = \begin{cases} f_c(\boldsymbol{x}_i) - 1 & c = y \text{ (output should be 1)} \\ f_c(\boldsymbol{x}_i) - 0 & \text{otherwise(output should be 0)} \end{cases}$$

Similarity to perceptron

Suppose we have a training token $(\boldsymbol{x}, y)$, and we have some initial class vectors $\boldsymbol{w}_c$. Using softmax and cross-entropy loss, we can update the weight vectors as

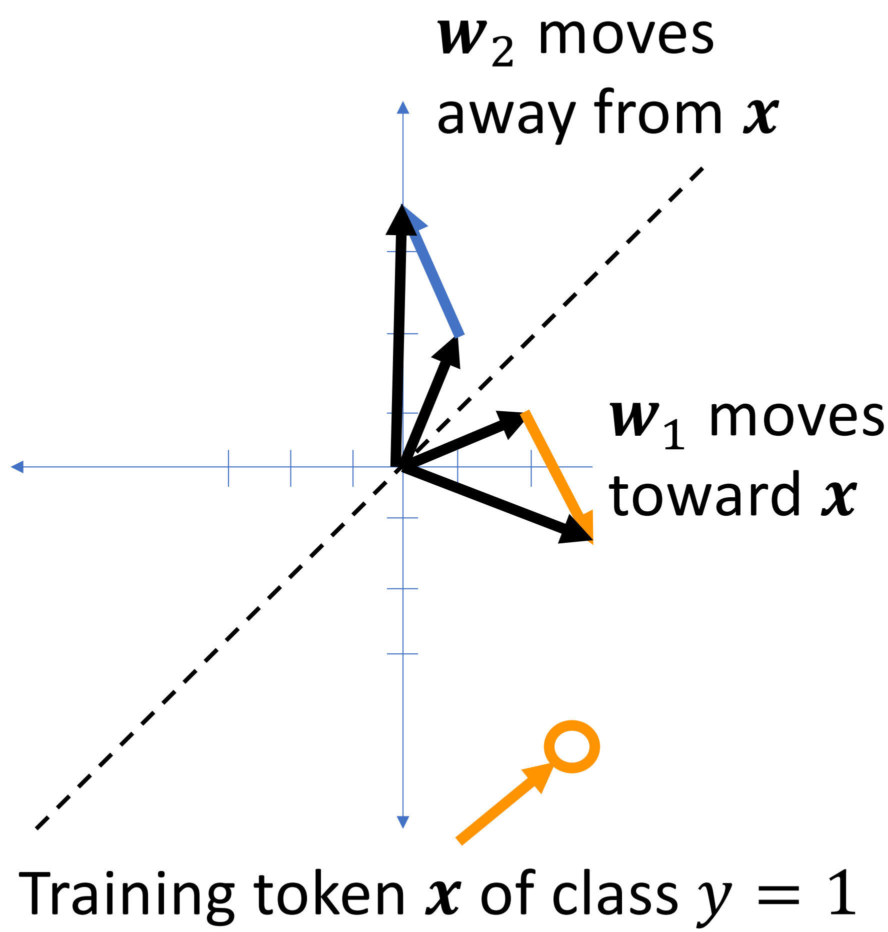
$$\boldsymbol{w}_c \leftarrow \boldsymbol{w}_c - \eta\epsilon_c\boldsymbol{x}$$

…where

$$\epsilon_c = \begin{cases} f_c(\boldsymbol{x}_i) - 1 & c = y_i \\ f_c(\boldsymbol{x}_i) - 0 & \text{otherwise} \end{cases}$$

In other words, like a perceptron,

$$= \begin{cases} \epsilon_c < 0 & c = y_i \\ \epsilon_c > 0 & \text{otherwise} \end{cases}$$



Outline

- Softmax: $f_c(\boldsymbol{x}) = \frac{\exp(\boldsymbol{w}_c^T \boldsymbol{x} + b_c)}{\sum_{k=1}^{v} \exp(\boldsymbol{w}_k^T \boldsymbol{x} + b_k)} \approx \Pr(Y = c|\boldsymbol{x})$

- Cross-entropy: $\mathcal{L} = -\ln f_y(\boldsymbol{x})$

- Derivative of the cross-entropy of a softmax:

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_c} = \epsilon_c \boldsymbol{x}, \qquad \epsilon_c = \begin{cases} f_c(\boldsymbol{x}_i) - 1 & c = y \text{ (output should be 1)} \\ f_c(\boldsymbol{x}_i) - 0 & \text{otherwise(output should be 0)} \end{cases}$$

- Gradient descent:

$$\boldsymbol{w}_c \leftarrow \boldsymbol{w}_c - \eta \epsilon_c \boldsymbol{x}$$