CS440/ECE448 Lecture 10: Perceptron

Mark Hasegawa-Johnson, 2/2024

Outline

- Linear Classifiers
- Gradient descent
- One-hot vectors and the perceptron loss function
- Perceptron learning algorithm

Linear classifier: Notation

- The observation $\boldsymbol{x}^T = [x_1, \dots, x_d]$ is a real-valued vector (d is the number of feature dimensions)
- The class label $y \in \mathcal{Y}$ is drawn from some finite set of class labels.
- Usually the output vocabulary, $\mathcal{Y}$, is some set of strings. For convenience, though, we usually map the class labels to a sequence of integers, $\mathcal{Y} = \{1, \dots, v\}$, where v is the vocabulary size

Linear classifier: Definition

A linear classifier is defined by

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

where:

$$\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} = \begin{bmatrix} w_{1,1} & \cdots & w_{1,d} \\ \vdots & \ddots & \vdots \\ w_{v,1} & \cdots & w_{v,d} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_v \end{bmatrix} = \begin{bmatrix} w_1^T x + b_1 \\ \vdots \\ w_v^T x + b_v \end{bmatrix}$$

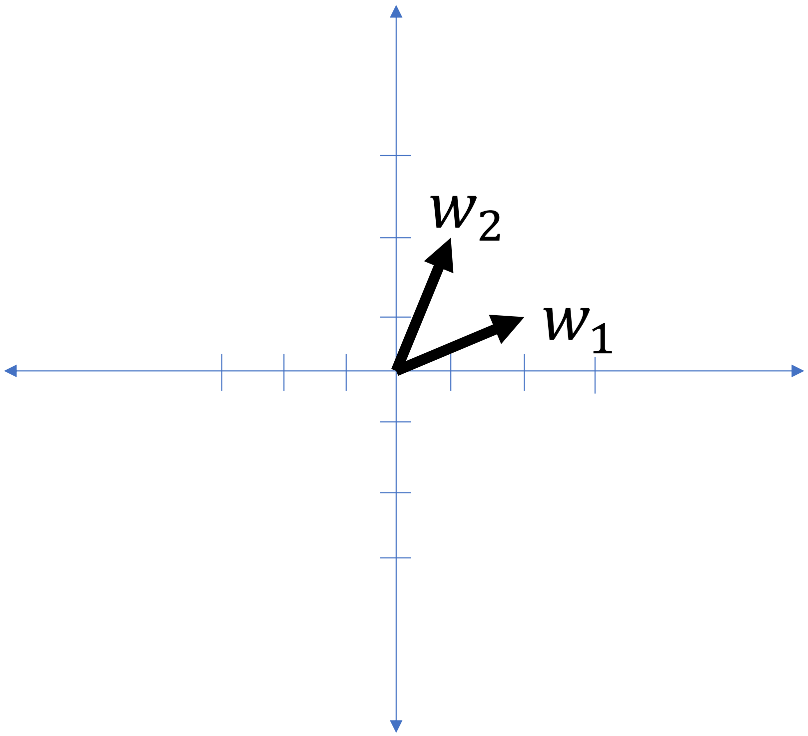
$\boldsymbol{w}_k, b_k$ are the **weight vector** and **bias** corresponding to **class k**, and the argmax function finds the element of the vector wx with the largest value.

There are a total of $v(d+1)$ trainable parameters: the elements of the matrix w.

Example

Consider a two-class classification problem, with

$$\boldsymbol{w}_1^T = [w_{1,1}, w_{1,2}] = [2,1]$$
$$\boldsymbol{w}_2^T = [w_{2,1}, w_{2,2}] = [1,2]$$



Example

Notice that in the two-class case, the equation

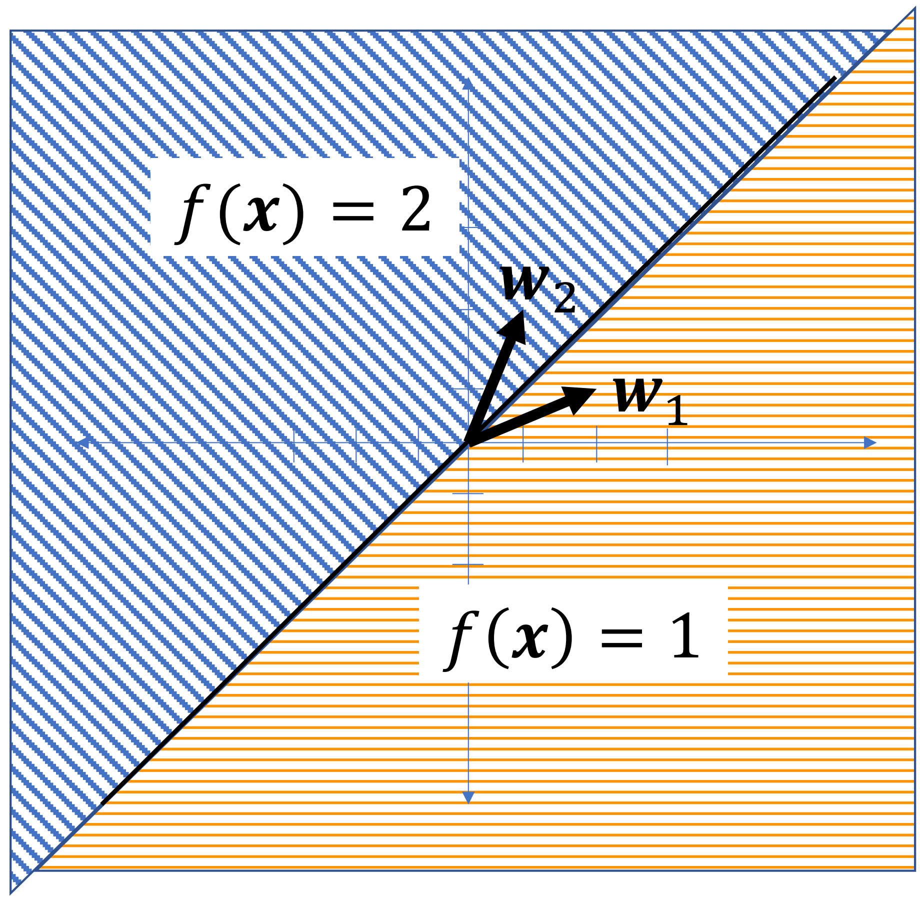
$$f(\boldsymbol{x}) = \text{argmax}\, \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$$

Simplifies to

$$f(\boldsymbol{x}) = \begin{cases} 1 & \boldsymbol{w}_1^T\boldsymbol{x} + b_1 > \boldsymbol{w}_2^T\boldsymbol{x} + b_2 \\ 2 & \boldsymbol{w}_1^T\boldsymbol{x} + b_1 < \boldsymbol{w}_2^T\boldsymbol{x} + b_2 \end{cases}$$

The class boundary is the line whose equation is

$$(\boldsymbol{w}_2 - \boldsymbol{w}_1)^T x + (b_2 - b_1) = 0$$



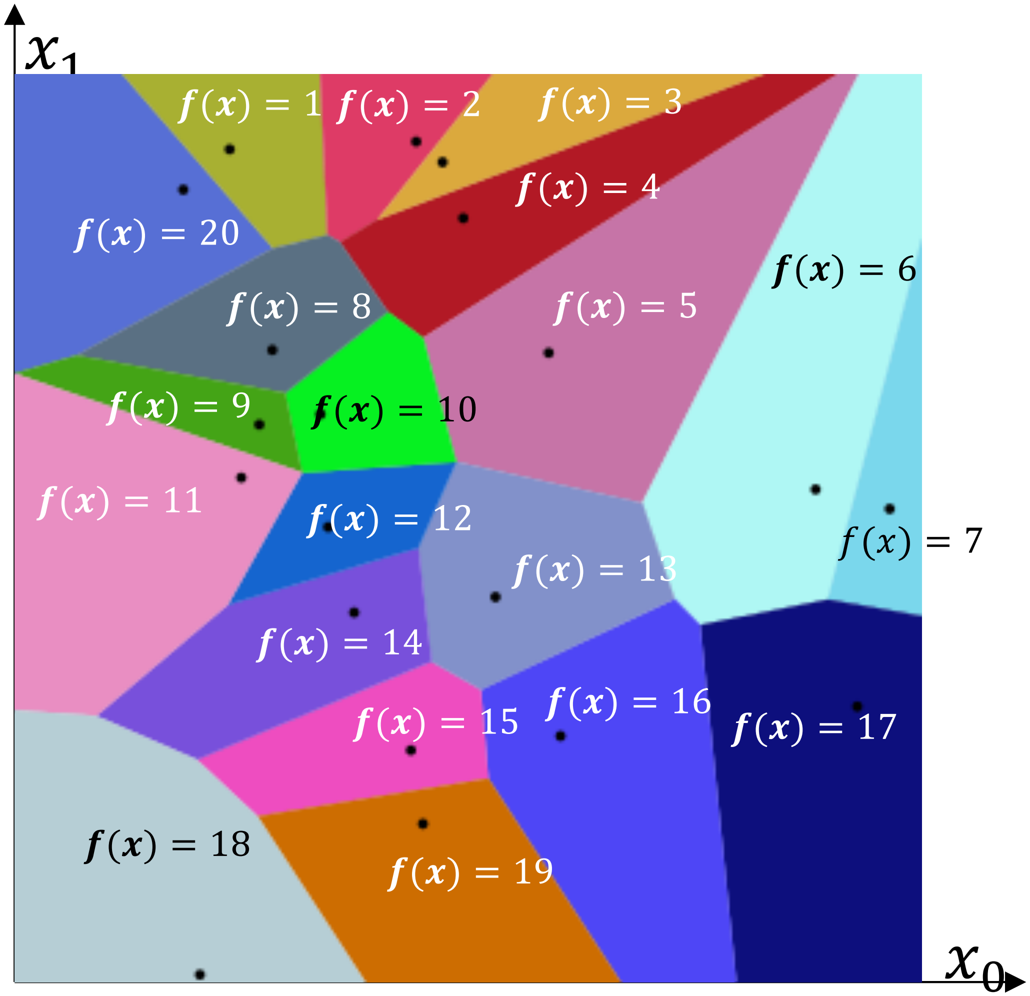
Multi-class linear classifier

In a general multi-class 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

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

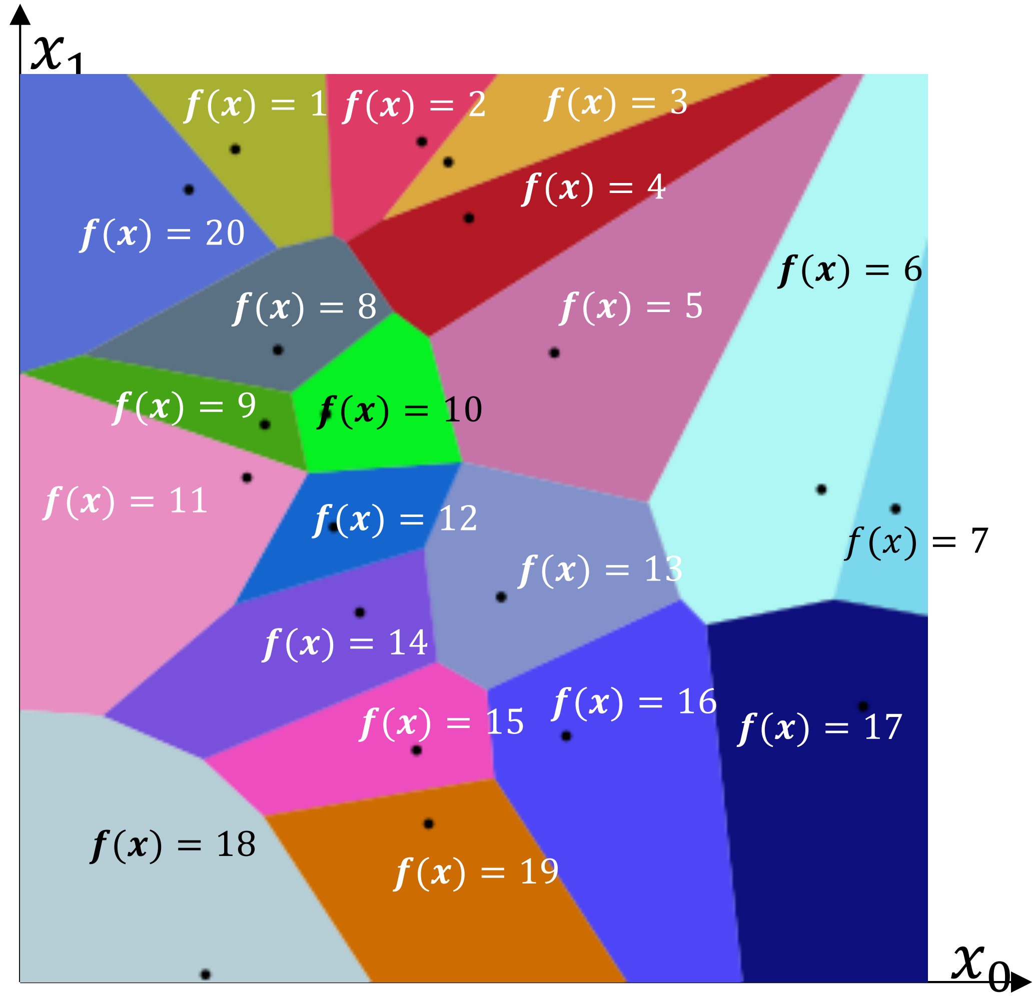


Voronoi regions

The classification regions in a linear classifier are called Voronoi regions.

A **Voronoi region** is a region that is

- Convex (if $\boldsymbol{u}$ and $\boldsymbol{v}$ are points in the region, then every point on the line segment $\overline{\boldsymbol{uv}}$ connecting them is also in the region)
- Bounded by piece-wise linear boundaries



Outline

- Linear Classifiers
- Gradient descent
- One-hot vectors and the perceptron loss function
- Perceptron learning algorithm

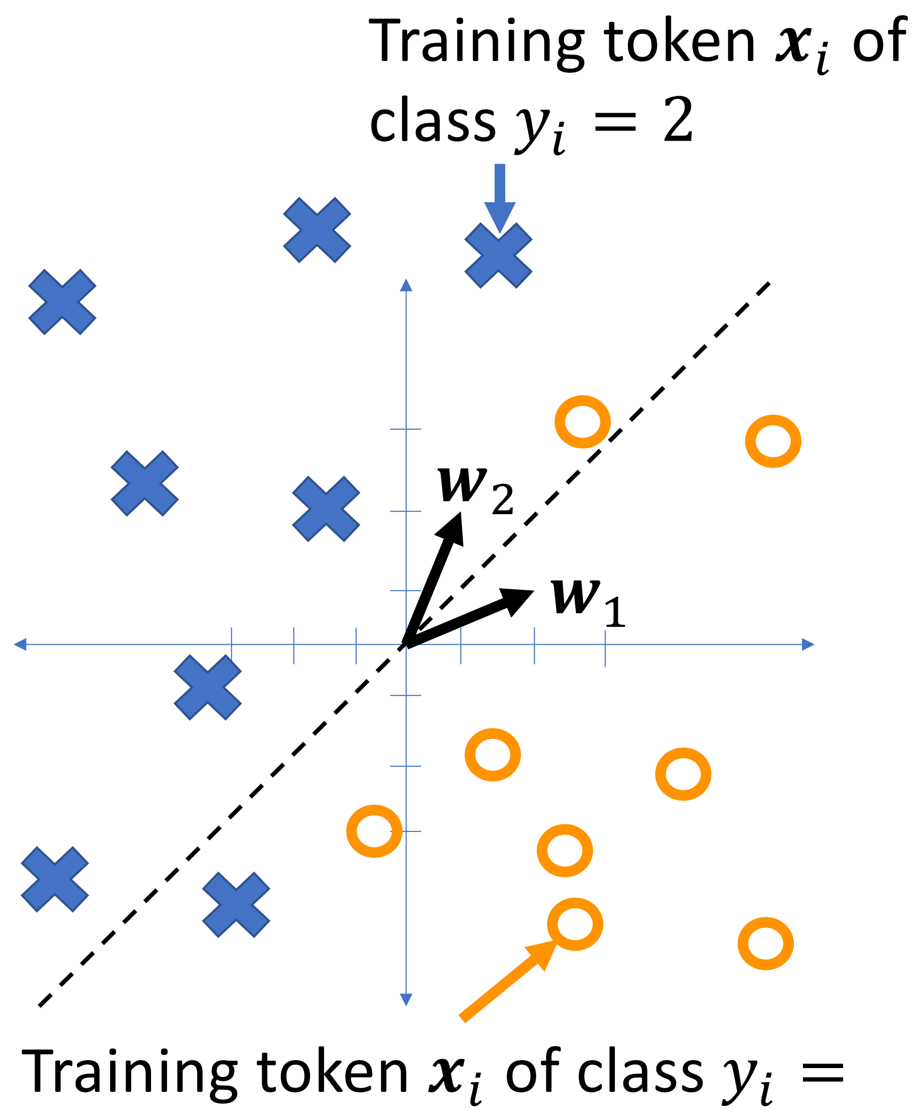
Gradient descent

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?



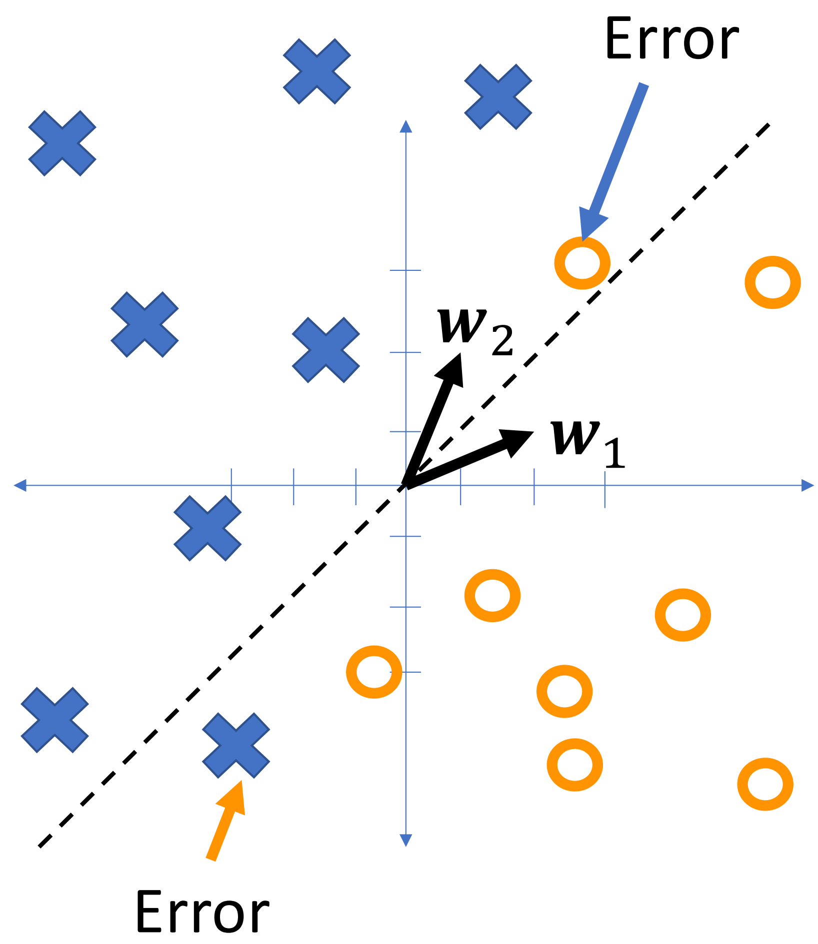
Zero-one loss function

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,

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

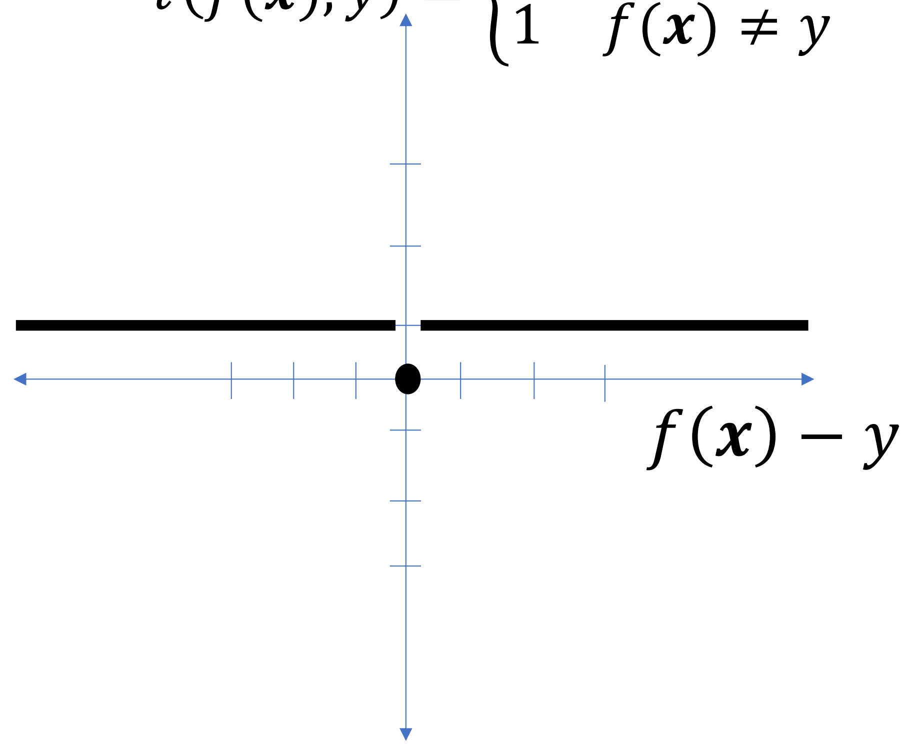


Non-differentiable!

The problem with the zero-one loss function is that it's not differentiable:

$$\frac{\partial \ell(f(\boldsymbol{x}), y)}{\partial f(\boldsymbol{x})} = \begin{cases} 0 & f(\boldsymbol{x}) \neq y \\ +\infty & f(\boldsymbol{x}) = y^{+} \\ -\infty & f(\boldsymbol{x}) = y^{-} \end{cases}$$

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



Outline

- Linear Classifiers: multi-class and 2-class
- Gradient descent
- One-hot vectors and the perceptron loss function
- Perceptron learning algorithm

One-hot vectors

A **one-hot vector** is a binary vector in which all elements are 0 except for a single element that's equal to 1.

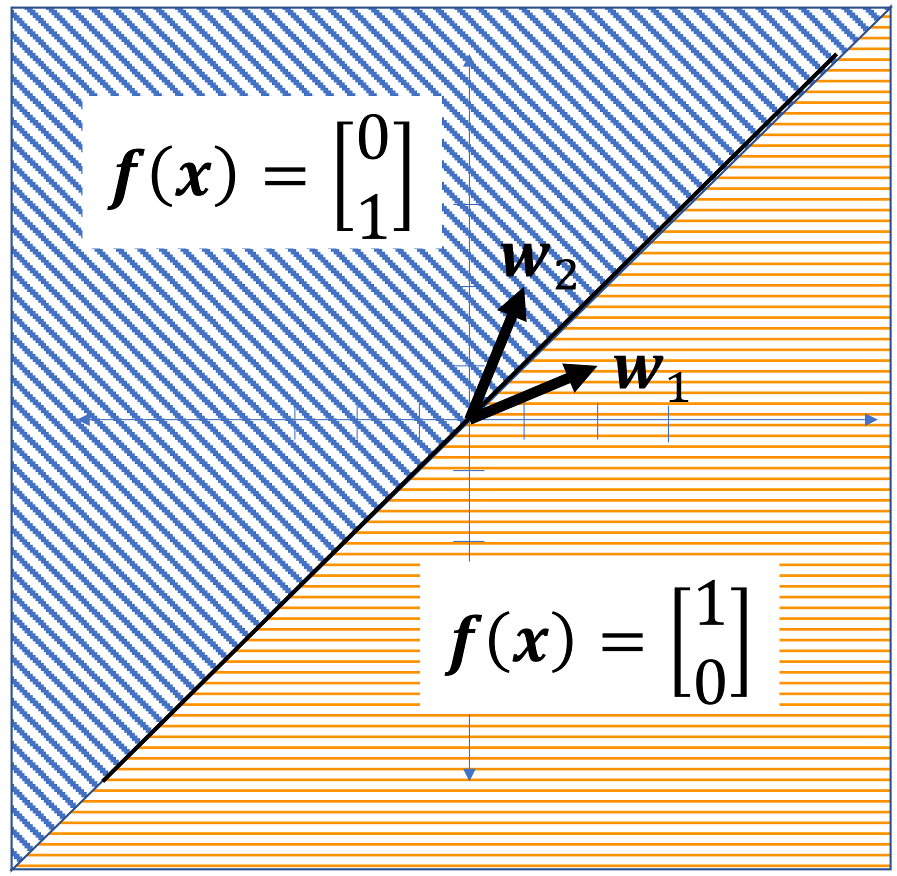
Example: Binary classifier

Consider the classifier

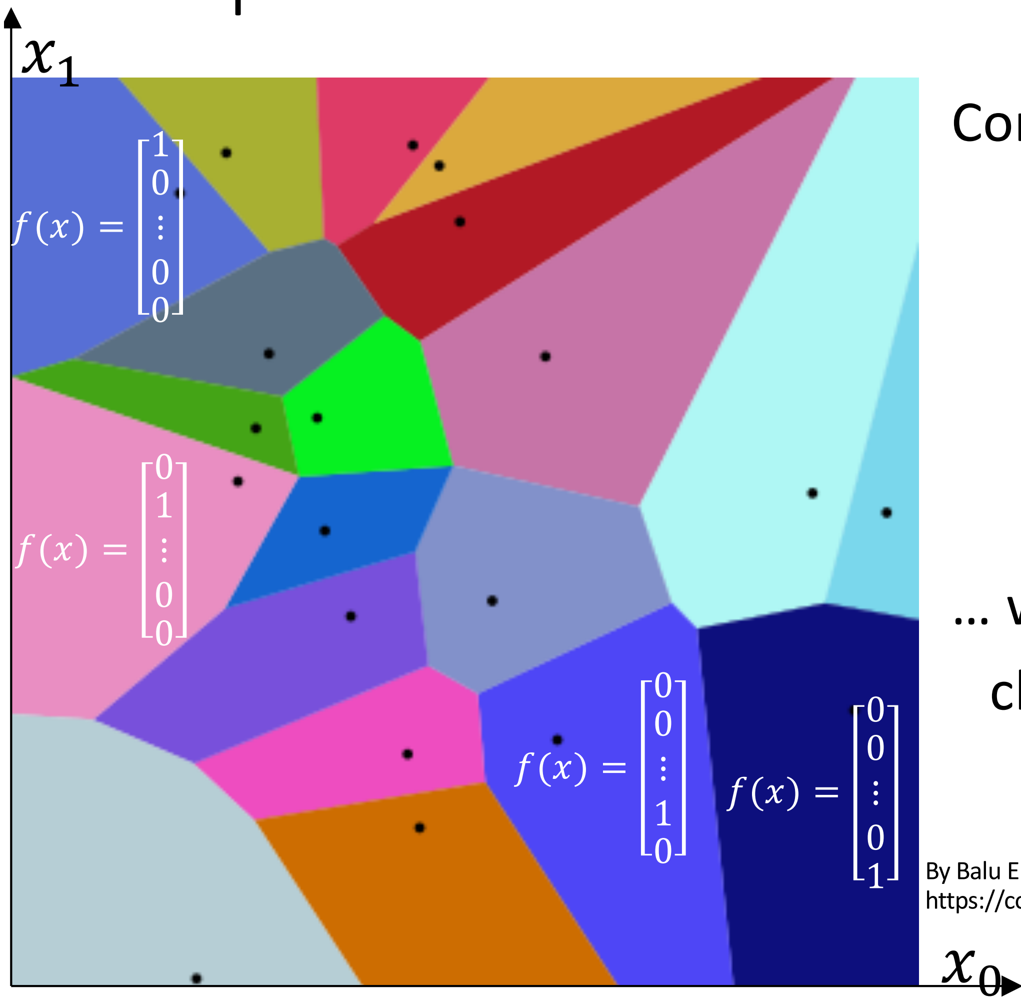
$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ f_2(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \mathbb{1}_{\mathrm{argmax}\, \boldsymbol{W}\boldsymbol{x}=1} \\ \mathbb{1}_{\mathrm{argmax}\, \boldsymbol{W}\boldsymbol{x}=2} \end{bmatrix}$$

…where $\mathbb{1}_P$ is called the “indicator function,” and it means:

$$\mathbb{1}_P = \begin{cases} 1 & P \text{ is true} \\ 0 & P \text{ is false} \end{cases}$$



Example: Multi-Class



Consider the classifier

$$\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_v(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \mathbb{I}_{\text{argmax}\, \boldsymbol{W}\boldsymbol{x}=1} \\ \vdots \\ \mathbb{I}_{\text{argmax}\, \boldsymbol{W}\boldsymbol{x}=v} \end{bmatrix}$$

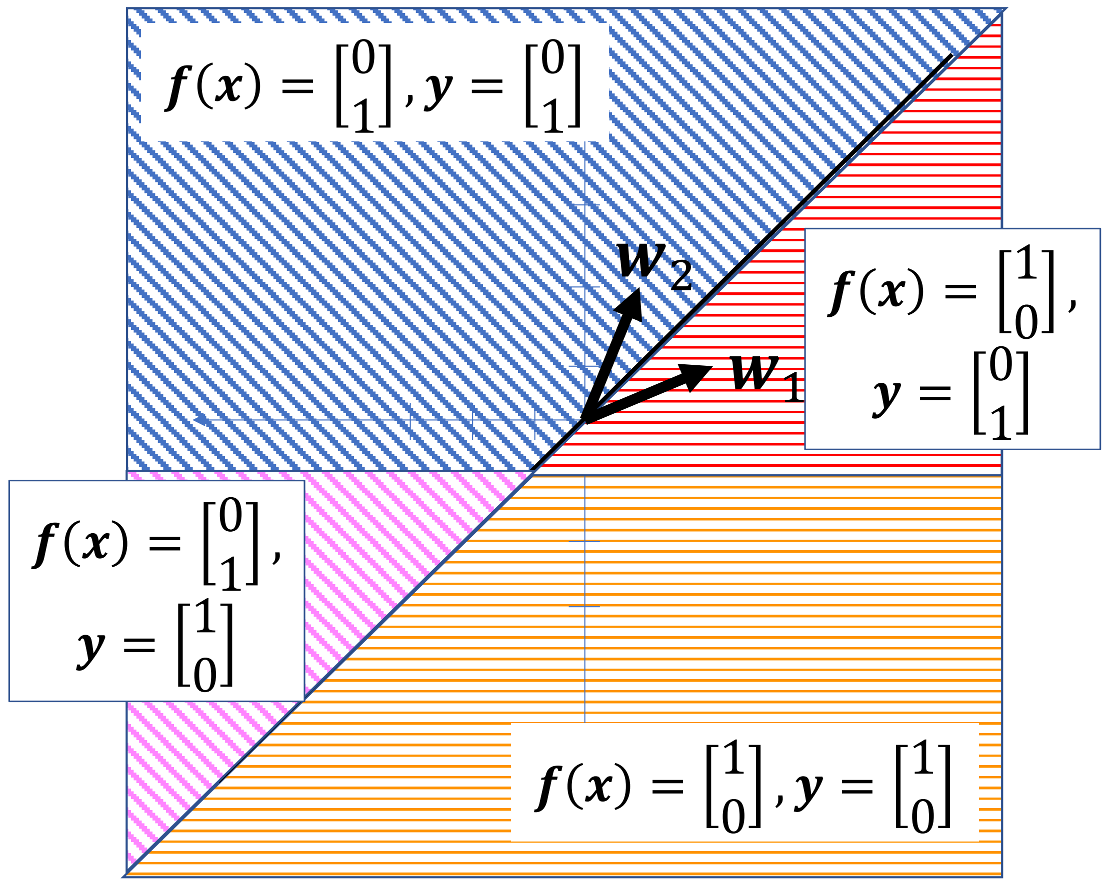
… with 20 classes. Then some of the classifications might look like this.

One-hot ground truth

We can also use one-hot vectors to describe the ground truth. Let's call the one-hot vector $\boldsymbol{y}$, and the integer label y, thus

$$\boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} \mathbb{1}_{y=1} \\ \mathbb{1}_{y=2} \end{bmatrix}$$

Ground truth might differ from classifier output. For example, they might be as shown here:



Counting errors using one-hot vectors

- An error occurs if $\boldsymbol{f}(\boldsymbol{x}) \neq \boldsymbol{y}$.
- So, to determine whether an error has occurred, we could just check:

$$\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y} = \begin{cases} \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix} & \text{no error occurred} \\ \text{anything else} & \text{an error occurred} \end{cases}$$

The perceptron loss

Instead of a one-zero loss, the perceptron uses a weird loss function that gives great results when differentiated. The perceptron loss function is:

$$\ell(\boldsymbol{x}, \boldsymbol{y}) = (\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{y})^T (\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b})$$

$$= [f_1(\boldsymbol{x}) - y_1, \quad \cdots, \quad f_v(\boldsymbol{x}) - y_v] \left(\begin{bmatrix} w_{1,1} & \cdots & w_{1,d} \\ \vdots & \ddots & \vdots \\ w_{v,1} & \cdots & w_{v,d} \end{bmatrix} \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix} + \begin{bmatrix} b_1 \\ \vdots \\ b_v \end{bmatrix} \right)$$

$$= \sum_{k=1}^{v} (f_k(\boldsymbol{x}) - y_k)\left(\boldsymbol{w}_k^T \boldsymbol{x} + b_k\right)$$

The perceptron loss

$$\ell(\boldsymbol{x}, \boldsymbol{y}) = \sum_{k=1}^{v} (f_k(\boldsymbol{x}) - y_k)\left(\boldsymbol{w}_k^T \boldsymbol{x} + b_k\right)$$

Notice that:

$$(f_k(\boldsymbol{x}) - y_k) = \begin{cases} +1 & f_k(\boldsymbol{x}) = 1, y_k = 0 \\ -1 & f_k(\boldsymbol{x}) = 0, y_k = 1 \\ 0 & \text{otherwise} \end{cases}$$

The perceptron loss

So what the loss really means is:

$$\ell(\boldsymbol{x}, \boldsymbol{y}) = \left(\boldsymbol{w}_{\hat{y}}^T \boldsymbol{x} + b_{\hat{y}}\right) - \left(\boldsymbol{w}_y^T \boldsymbol{x} + b_y\right)$$

Where:

- y is the correct class label for this training token
- $\hat{y} = \underset{\boldsymbol{k}}{\operatorname{argmax}}\left(\boldsymbol{w}_k^T \boldsymbol{x} + b_k\right)$ is the classifier output
- $\ell(\boldsymbol{x}, \boldsymbol{y}) > 0$ if $\hat{y} \neq y$
- $\ell(\boldsymbol{x}, \boldsymbol{y}) = 0$ if $\hat{y} = y$

Outline

- Linear Classifiers: multi-class and 2-class
- Gradient descent
- One-hot vectors and the perceptron loss function
- Perceptron learning algorithm

Gradient of the perceptron loss

$$\ell(\boldsymbol{x}, \boldsymbol{y}) = \left(\boldsymbol{w}_{\hat{y}}^{T}\boldsymbol{x} + b_{\hat{y}}\right) - \left(\boldsymbol{w}_{y}^{T}\boldsymbol{x} + b_{y}\right)$$

Its derivative is:

$$\frac{\partial \ell(\boldsymbol{x}, \boldsymbol{y})}{\partial \boldsymbol{w}_k} = \begin{cases} x & k = \hat{y} \\ -x & k = y \\ 0 & \text{otherwise} \end{cases}$$

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:

$$\boldsymbol{w}_k \leftarrow \boldsymbol{w}_k - \eta \frac{\partial \ell(\boldsymbol{x}, \boldsymbol{y})}{\partial \boldsymbol{w}_k} = \begin{cases} \boldsymbol{w}_k - \eta \boldsymbol{x} & k = \hat{y} \\ \boldsymbol{w}_k + \eta \boldsymbol{x} & k = y \\ 0 & \text{otherwise} \end{cases}$$

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

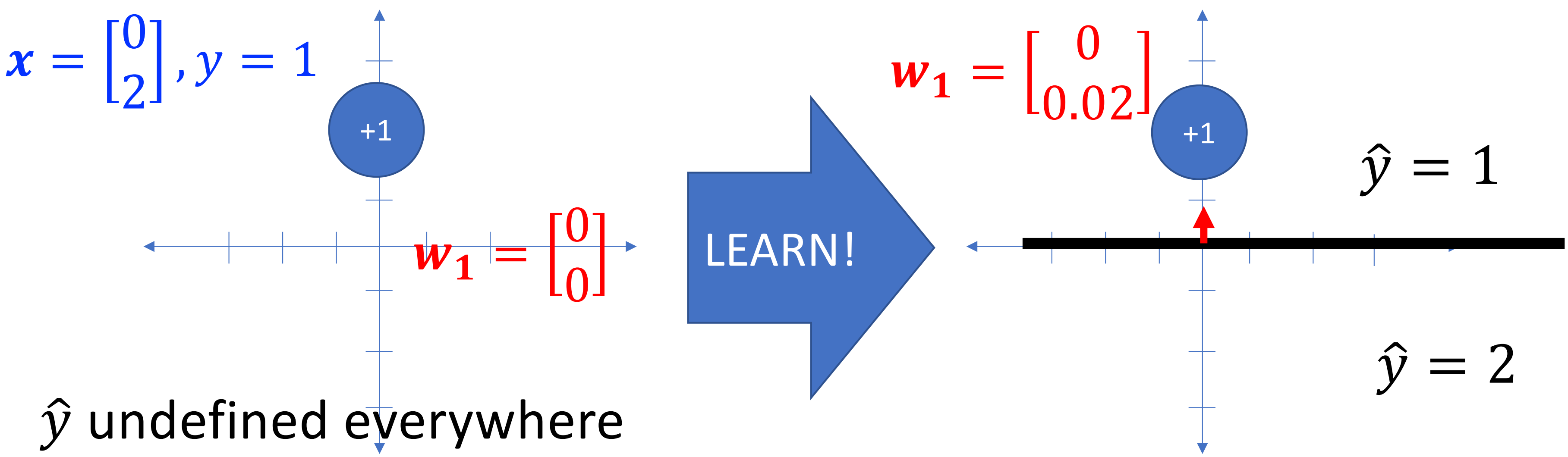
Example

Start with $\boldsymbol{w}_k = [0,0]^T$ for both classes.

Suppose that $x = [0,2]^T$, with the label $y = 1$.

$\hat{y} = \underset{\boldsymbol{k}}{\operatorname{argmax}}\left(\boldsymbol{w}_k^T \boldsymbol{x}\right)$ is undefined, since $\boldsymbol{w}_k^T \boldsymbol{x} = \boldsymbol{0}$ for both classes, so we only update

$$\boldsymbol{w}_1 \leftarrow \boldsymbol{w}_1 + \eta \boldsymbol{x} = \begin{bmatrix} 0 \\ 0.02 \end{bmatrix}$$



Example

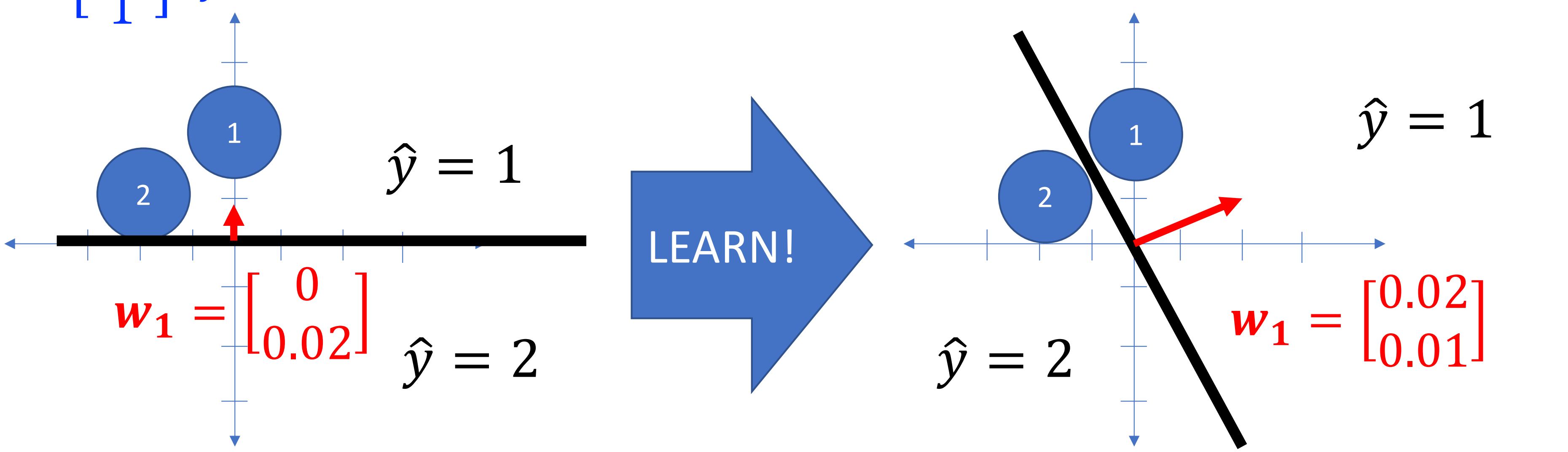
Now $\boldsymbol{w}_1 = [0,0.02]^T$, but $\boldsymbol{w}_2 = [0,0]^T$.

Suppose the next $\boldsymbol{x} = [-2,1]^T$, with the label $y = 2$.

$\hat{y} = \underset{\boldsymbol{k}}{\operatorname{argmax}}\left(\boldsymbol{w}_k^T \boldsymbol{x}\right) = 1$ which is wrong, so we update

$$\boldsymbol{w}_1 \leftarrow \boldsymbol{w}_1 - \eta \boldsymbol{x} = \begin{bmatrix} 0.02 \\ 0.01 \end{bmatrix}, \qquad \boldsymbol{w}_2 \leftarrow \boldsymbol{w}_2 + \eta \boldsymbol{x} = \begin{bmatrix} -0.02 \\ 0.01 \end{bmatrix}$$

$\boldsymbol{x} = \begin{bmatrix} -2 \\ 1 \end{bmatrix}, y = 2$



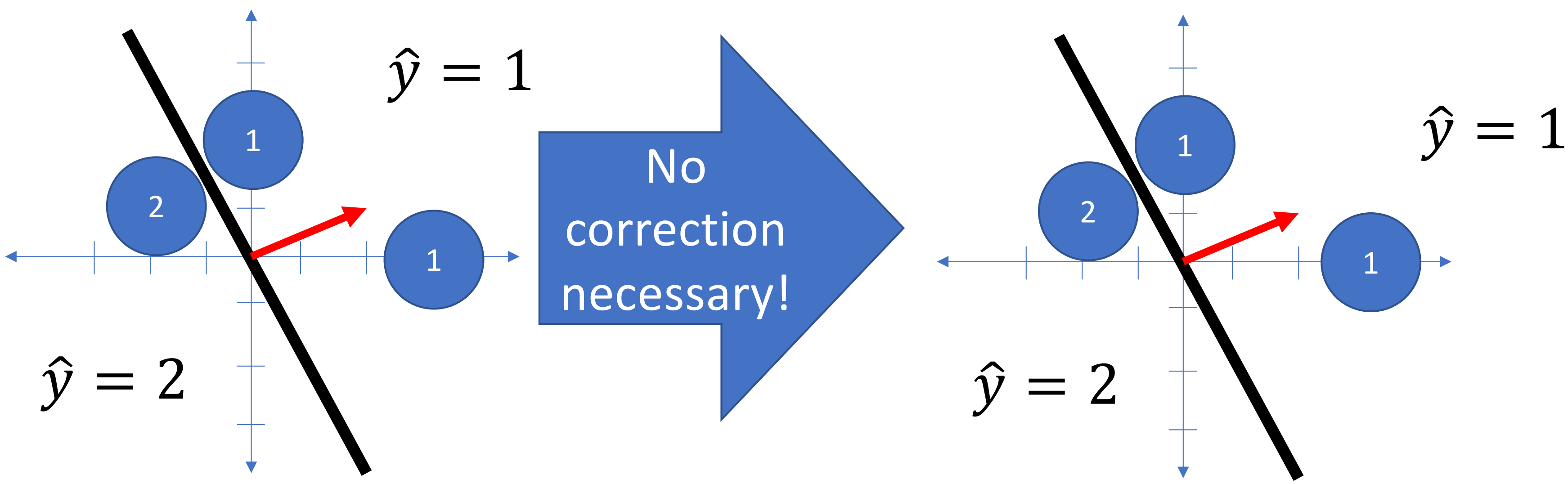
Example

Suppose the next token is $\boldsymbol{x} = [3,0]^T$, with the label $y = 1$. Since $\hat{y}$ is right, the weights don't need to be updated:

$$\boldsymbol{w}_k \leftarrow \boldsymbol{w}_k + 0$$

$\boldsymbol{x} = \begin{bmatrix} 3 \\ 1 \end{bmatrix}, y = 1$

$\boldsymbol{w}_k$ unchanged



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:

$$\boldsymbol{w}_k \leftarrow \begin{cases} \boldsymbol{w}_k - \eta \boldsymbol{x} & k = \hat{y} \\ \boldsymbol{w}_k + \eta \boldsymbol{x} & k = y \\ 0 & \text{otherwise} \end{cases}$$

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

Try the quiz!

Try the quiz:
https://us.prairielearn.com/pl/course_instance/147925/assessment/2395719

Special case: two classes

If there are only two classes, then we only need to learn one weight vector, $\boldsymbol{w} = \boldsymbol{w}_1 - \boldsymbol{w}_2$. We can learn it as:

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:

$$\boldsymbol{w} \leftarrow \begin{cases} \boldsymbol{w} - \eta \boldsymbol{x} & \hat{y} \neq y, y = 2 \\ \boldsymbol{w} + \eta \boldsymbol{x} & \hat{y} \neq y, y = 1 \\ 0 & \hat{y} = y \end{cases}$$

where $\eta \approx 0.01$ is the learning rate. Sometimes we say $y \in \{1, -1\}$ instead of $y \in \{1,2\}$.

Outline

- Linear Classifiers: $f(\boldsymbol{x}) = \operatorname{argmax} \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b}$
- Gradient descent: $\boldsymbol{w}_c \leftarrow \boldsymbol{w}_c - \eta \frac{\partial \mathcal{L}}{\partial \boldsymbol{w}_c}$
- One-hot vectors: $\boldsymbol{f}(\boldsymbol{x}) = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_v(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} \mathbb{1}_{\operatorname{argmax} \boldsymbol{W}\boldsymbol{x}=1} \\ \vdots \\ \mathbb{1}_{\operatorname{argmax} \boldsymbol{W}\boldsymbol{x}=v} \end{bmatrix}, \ \boldsymbol{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \end{bmatrix} = \begin{bmatrix} \mathbb{1}_{y=1} \\ \mathbb{1}_{y=2} \\ \vdots \end{bmatrix}$
- Perceptron learning algorithm:

$$\boldsymbol{w}_c \leftarrow \begin{cases} \boldsymbol{w}_c - \eta \boldsymbol{x} & c = \hat{y} \\ \boldsymbol{w}_c + \eta \boldsymbol{x} & c = y \\ 0 & \text{otherwise} \end{cases}$$