

Introduction to Machine Learning

Perceptron

Nan Jiang
CS PhD at Purdue University

Feb 2025

Perceptron

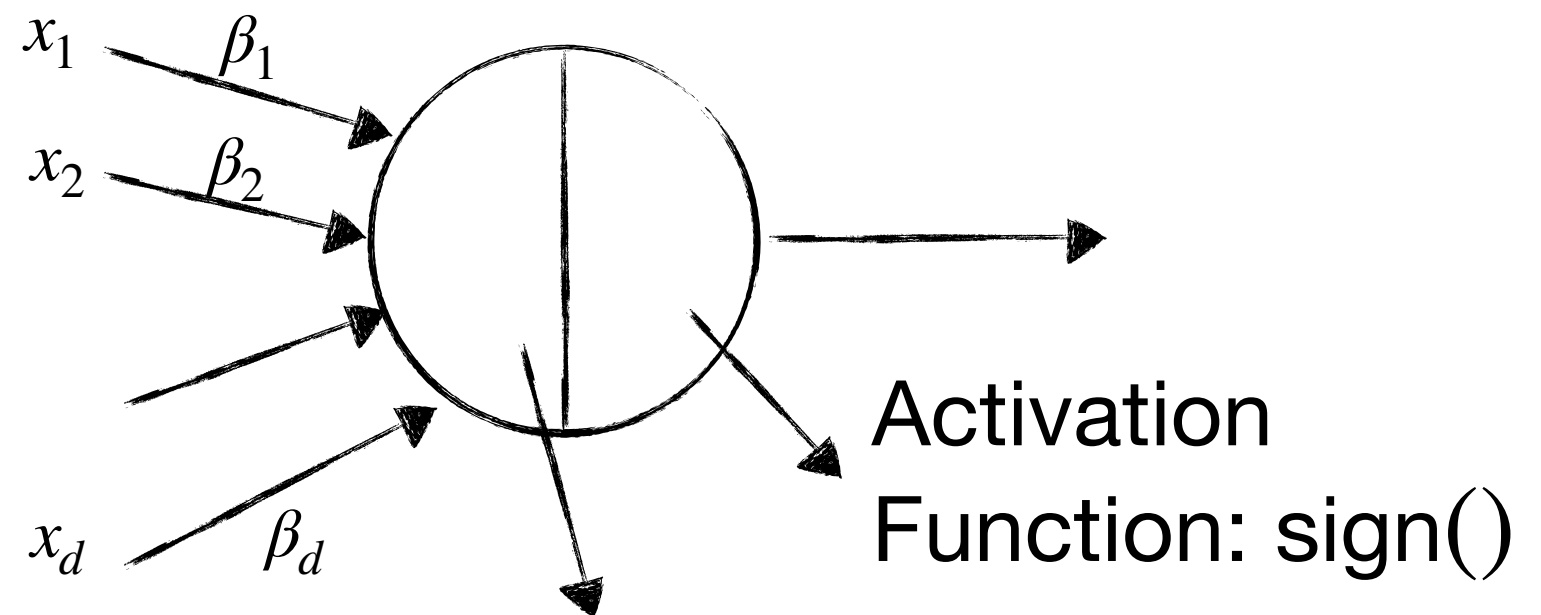
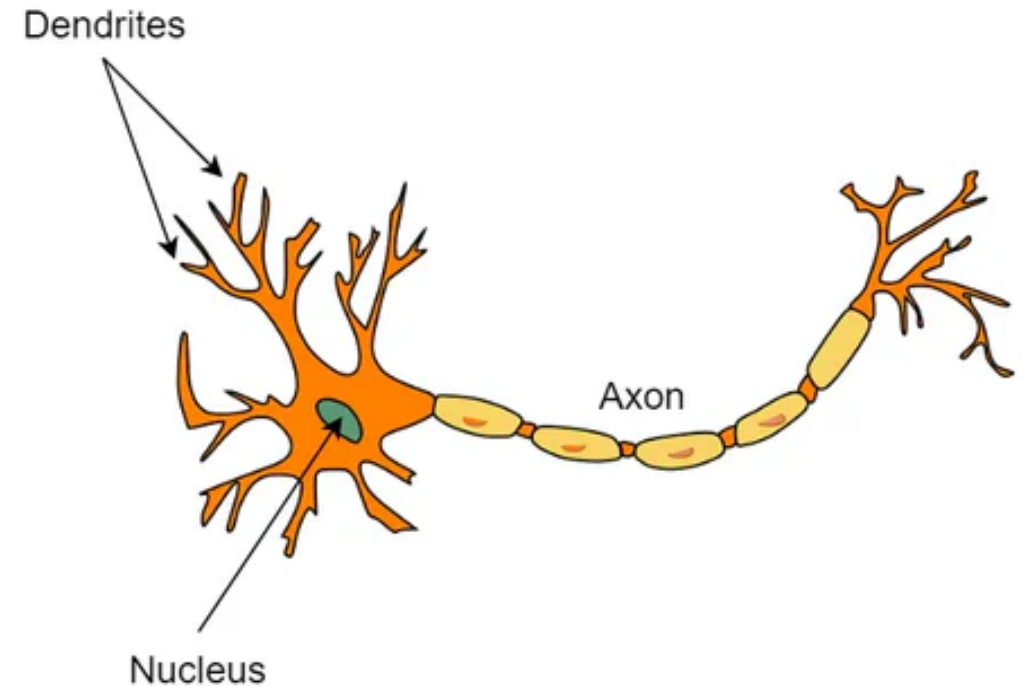
- Simple Linear Classifier

- Input $\underline{x} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_d \end{bmatrix} \in \mathbb{R}^d$,

- Weight $\underline{\beta} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \dots \\ \beta_d \end{bmatrix} \in \mathbb{R}^d$

- bias $\beta_0 \in \mathbb{R}$

- Output $y \in \{-1, 1\}$

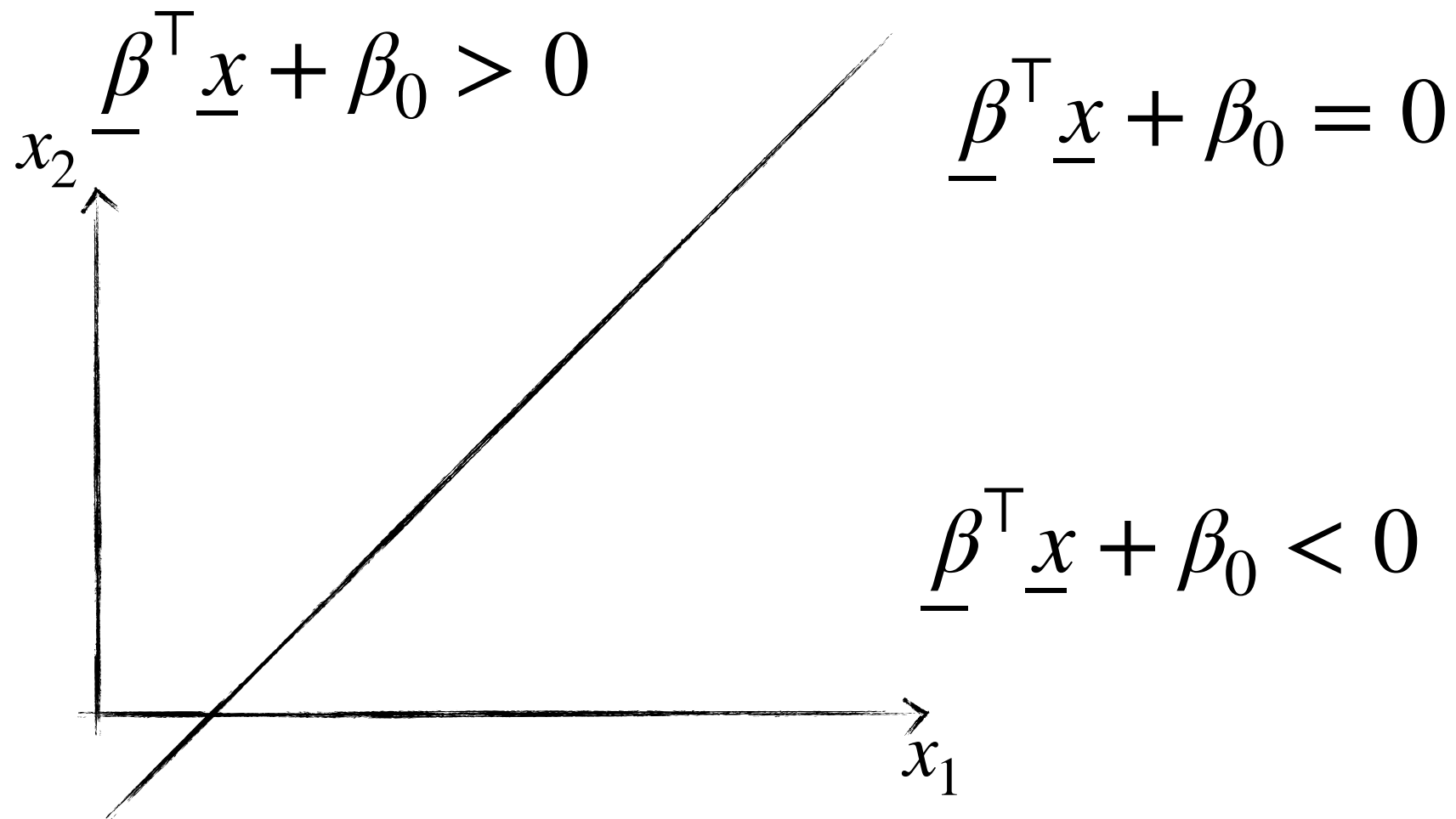


Linear Function $\sum_{i=1}^d \beta_i x_i + \beta_0$

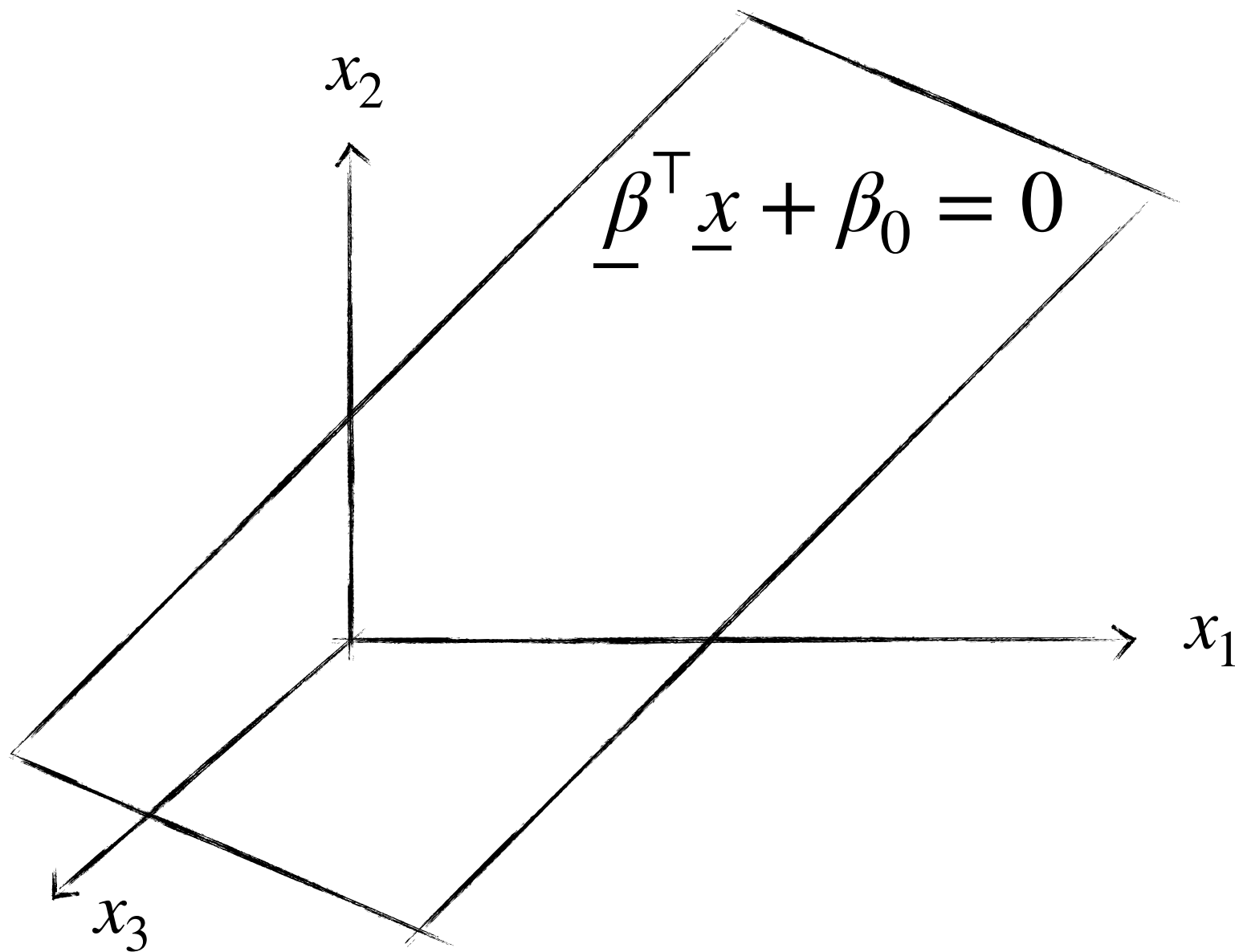
Vector inner product notation

- $\underline{\beta}^\top \underline{x}_1 = [\beta_1 \quad \beta_2 \quad \dots \quad \beta_d] \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_d \end{bmatrix} = \sum_{i=1}^d \beta_i x_i \in \mathbb{R}$

The decision boundary (when $d=2$)



The decision boundary (when $d=3$)

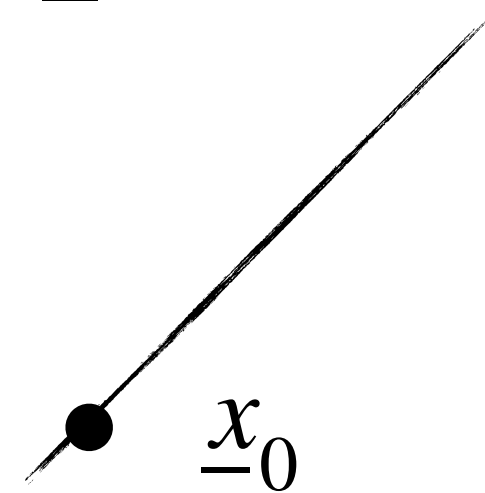


- For any \underline{x}_0 on this line (or hyper plane)

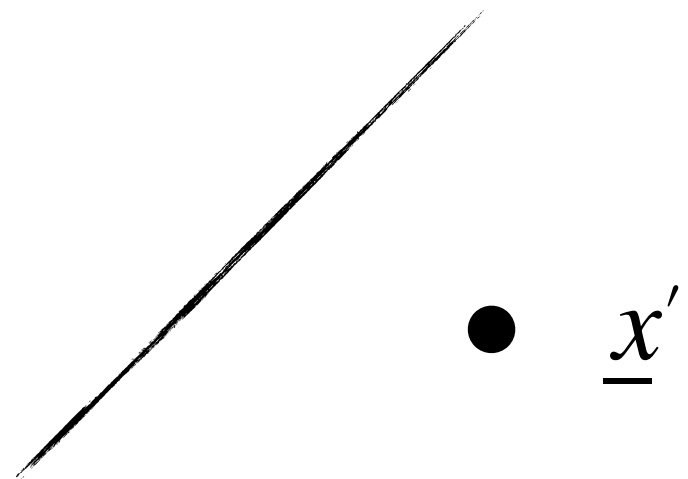
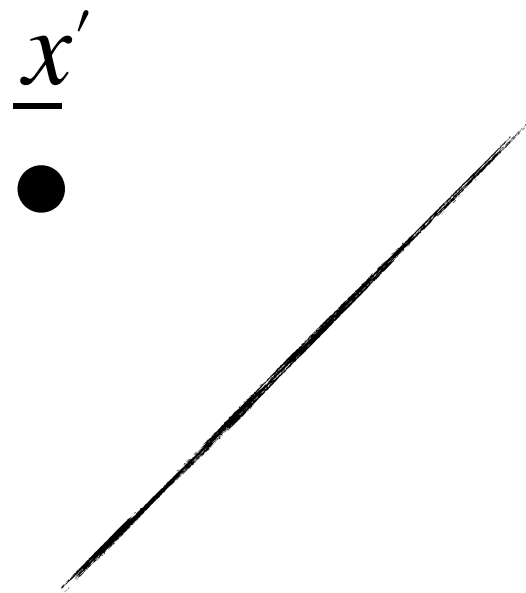
- $\underline{\beta}^\top \underline{x}_0 + \beta_0 = 0$

- $\rightarrow \beta_0 = -\underline{\beta}^\top \underline{x}_0$

$$\underline{\beta}^\top \underline{x} + \beta_0 = 0$$



- For any $\underline{x'}$ not on this line (or hyper plane)



- For any \underline{x}_1 and \underline{x}_2 on this line (or hyper plane):

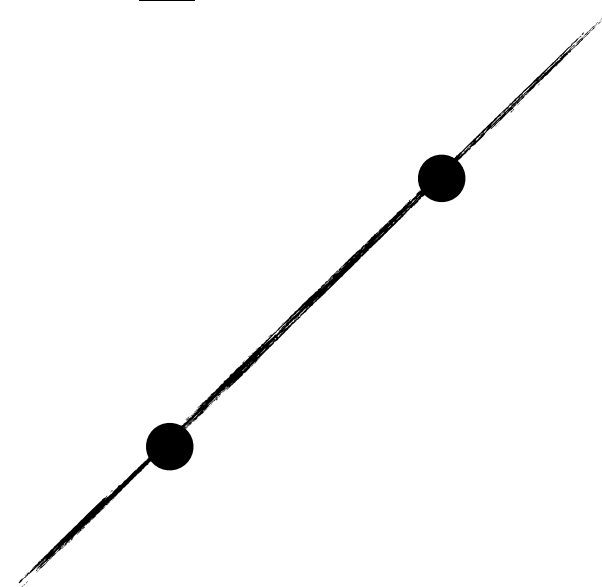
- $\underline{\beta}^\top \underline{x}_1 + \beta_0 = \underline{\beta}^\top \underline{x}_2 + \beta_0 = 0$

- $\underline{\beta}^\top \underline{x}_1 - \underline{\beta}^\top \underline{x}_2 = 0$

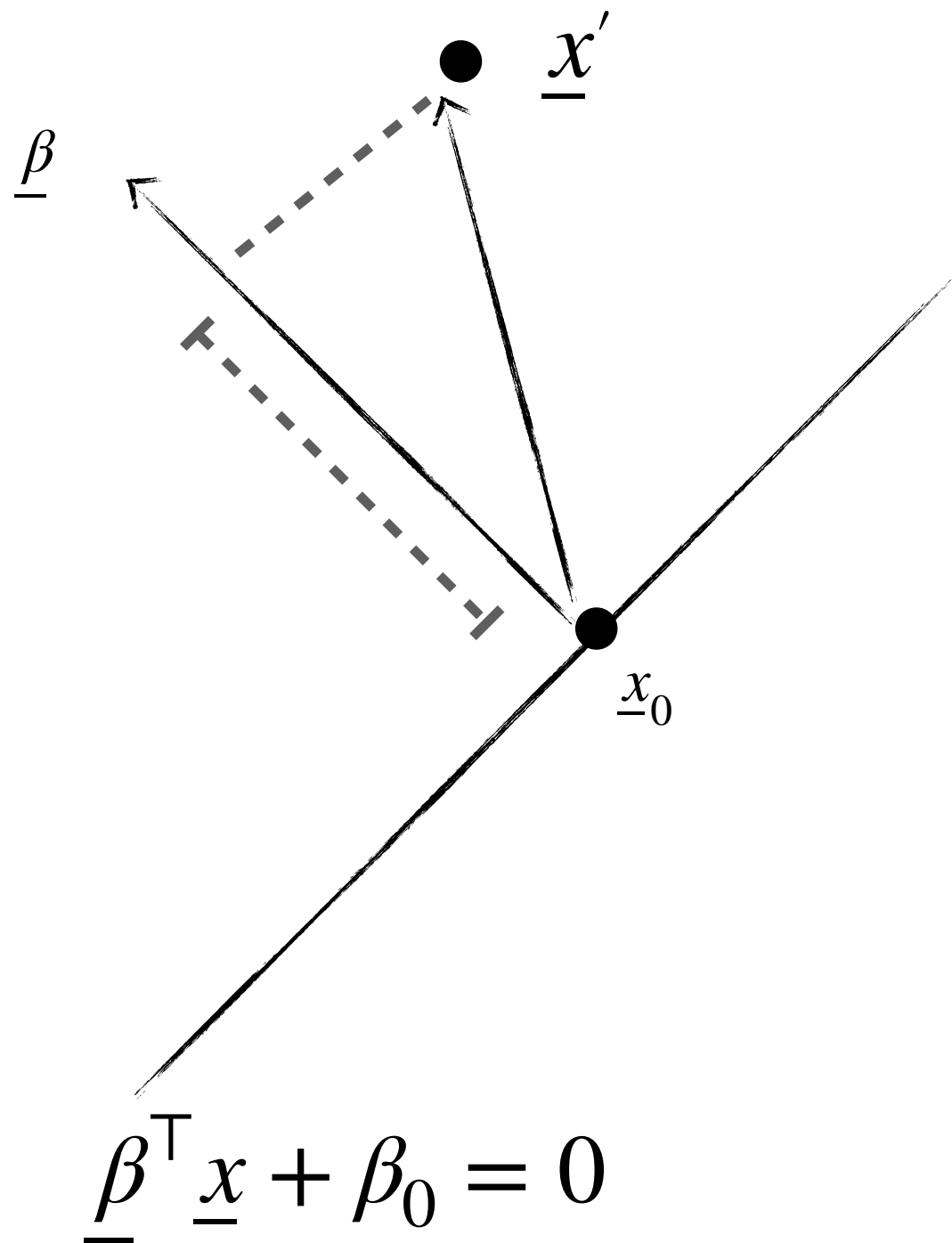
- $\underline{\beta}^\top (\underline{x}_1 - \underline{x}_2) = 0$

- vector $\underline{\beta}^\top \perp$ vector $(\underline{x}_1 - \underline{x}_2)$

$$\underline{\beta}^\top \underline{x} + \beta_0 = 0$$



The distance from the point to the line



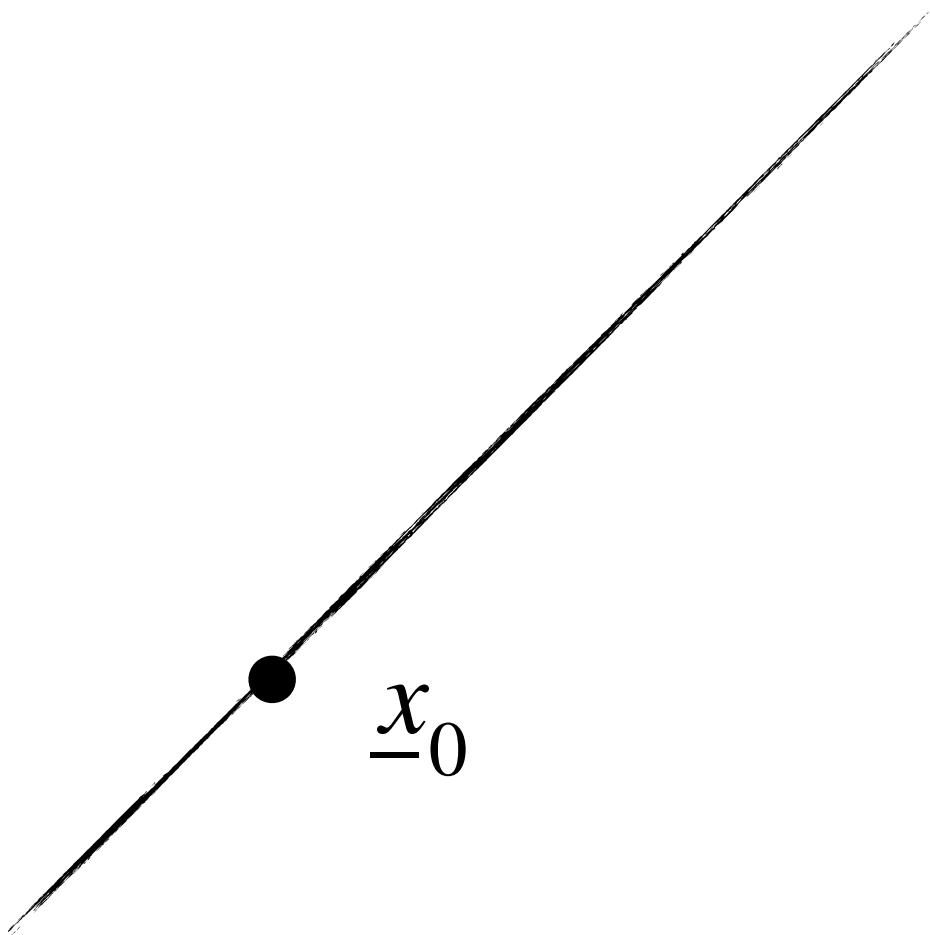
The distance from
the point to the line

$$\begin{aligned}\underline{\beta}^\top (\underline{x}' - \underline{x}_0) \\&= \underline{\beta}^\top \underline{x}' - \underline{\beta}^\top \underline{x}_0 \\&= \underline{\beta}^\top \underline{x}' + \beta_0\end{aligned}$$

The distance from the point to the line

The distance $\underline{\beta}^\top \underline{x}_i + \beta_0$, The output $y_i \in \{+1, -1\}$

● \underline{x}'

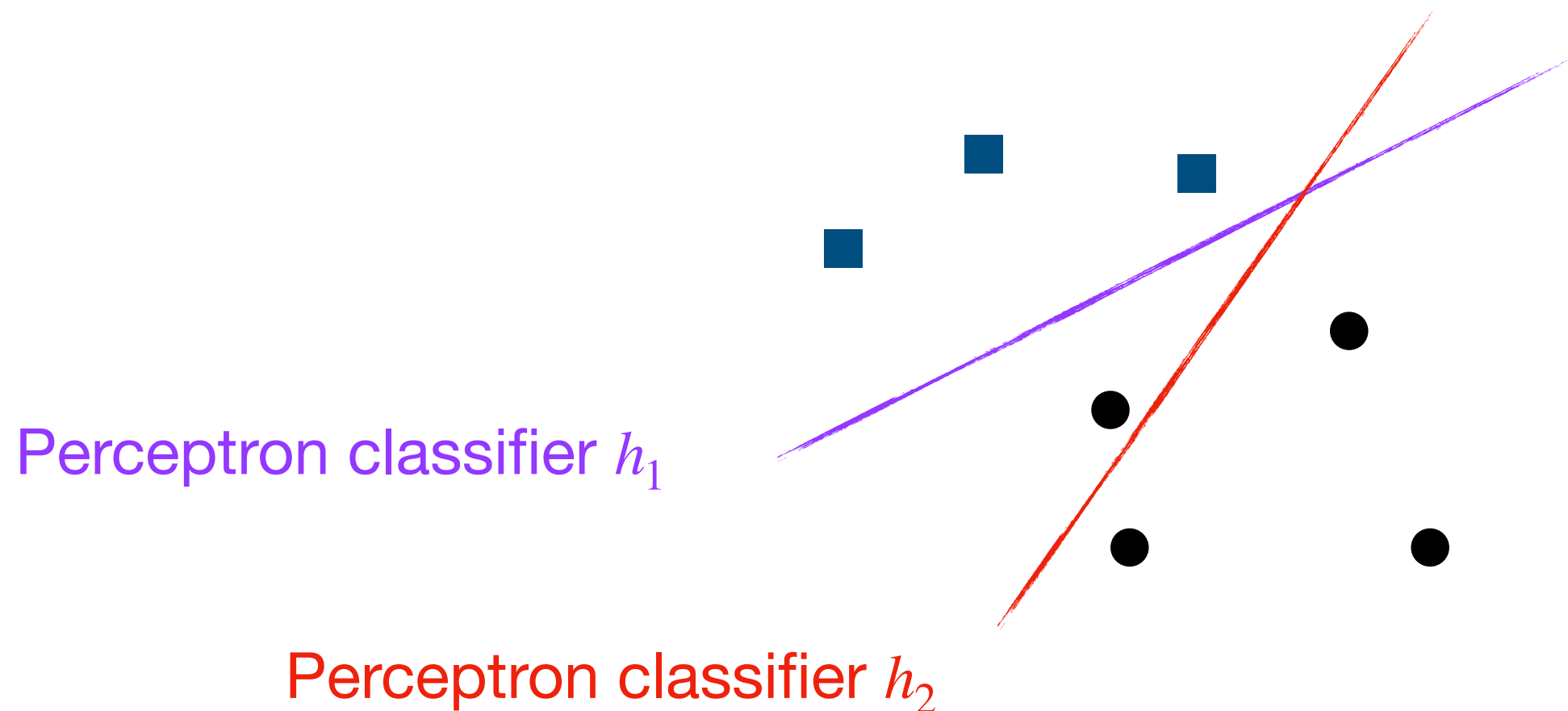


● \underline{x}''

The distance
 $y_i \left(\underline{\beta}^\top \underline{x}_i + \beta_0 \right)$

Which classifier is better?

- Given a dataset, quantify which classifier is better?

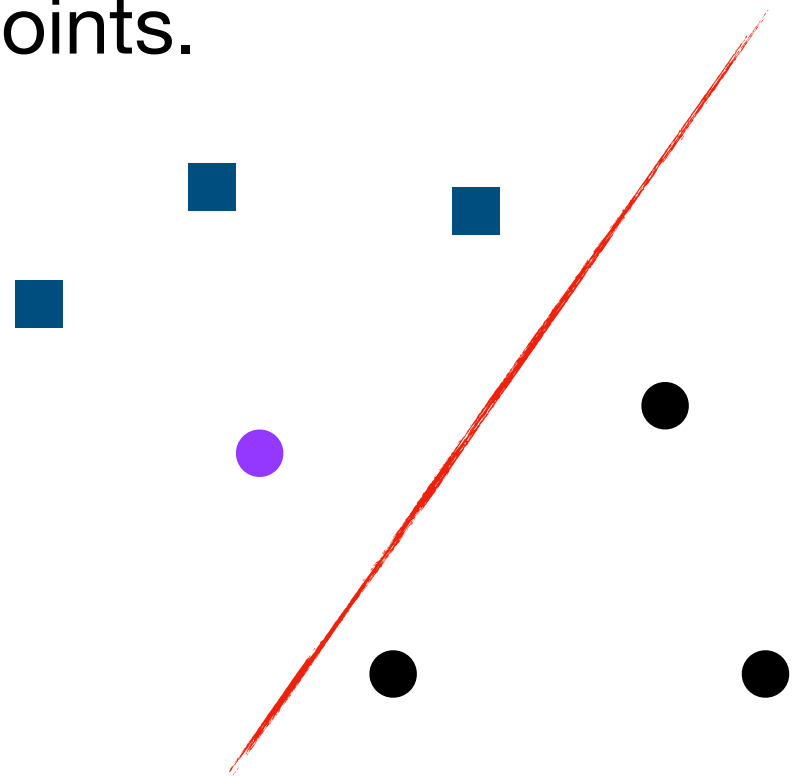


Cost Function

The distance $y_i \left(\underline{\beta}^\top \underline{x}_i + \beta_0 \right)$

Let M be the number of misclassified points.

$$\ell(\underline{\beta}, \beta_0) = \sum_{i=1}^M y_i \left(\underline{\beta}^\top \underline{x}_i + \beta_0 \right)$$



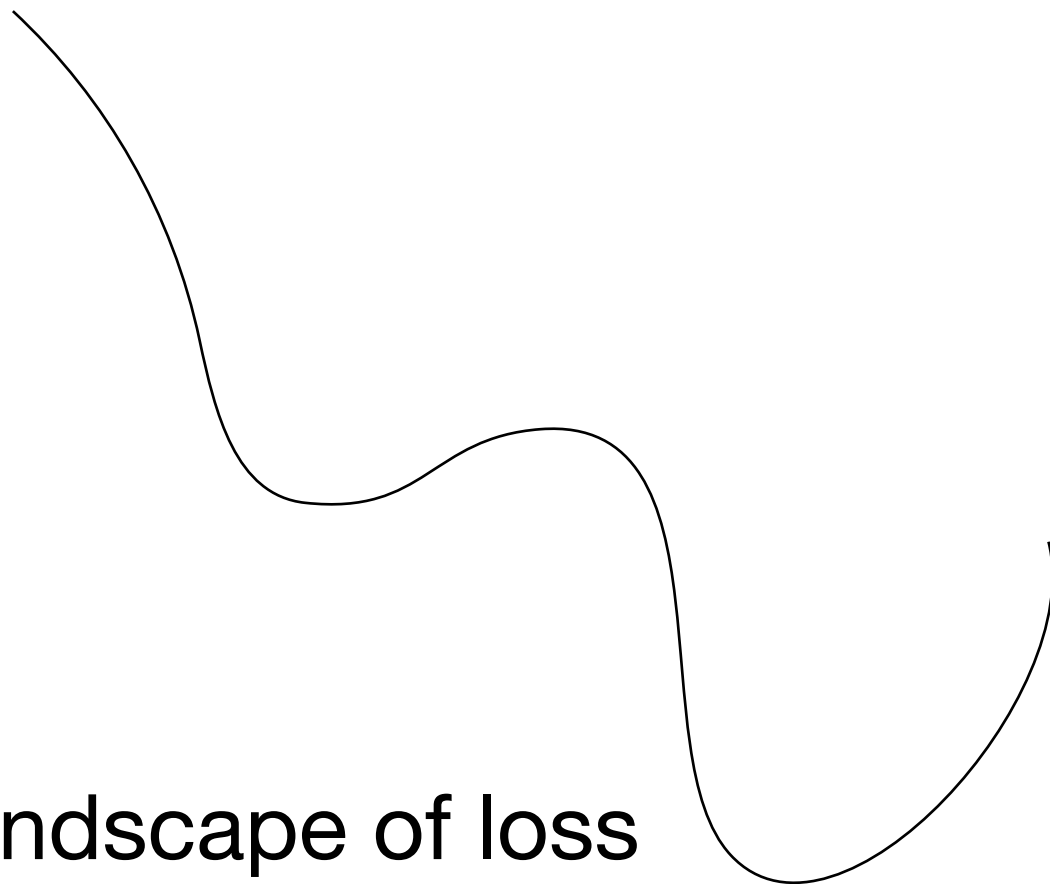
Update Rule of Perceptron

Gradient Descent:

- Let β be the parameters of the model and $0 < \gamma \leq 1$ be the learning rate, we compute:

$$\beta_{new} \leftarrow \beta_{old} - \gamma \frac{\partial \ell}{\partial W}$$

The landscape of loss



Compute the gradient

Let M be the number of misclassified points.

The loss function

$$\ell(\underline{\beta}, \beta_0) = \sum_{i=1}^M y_i \left(\underline{\beta}^\top \underline{x}_i + \beta_0 \right)$$

The gradient is:

$$\frac{\partial \ell}{\partial \underline{\beta}} = - \sum_{i=1}^M y_i \underline{x}_i$$

$$\frac{\partial \ell}{\partial \beta_0} = - \sum_{i=1}^M y_i$$

Update Rule of Perceptron

- In the Perceptron algorithm, we update parameters as follow:

$$\frac{\partial \ell}{\partial \beta} = - \sum_{i=1}^M y_i x_i$$
$$\frac{\partial \ell}{\partial \beta_0} = - \sum_{i=1}^M y_i$$

$$\beta_{new} \leftarrow \beta_{old} - \gamma \frac{\partial \ell}{\partial W}$$

$$\beta_{new} \leftarrow \beta_{old} + \gamma y_i x_i$$

$$\beta_{new} \leftarrow \beta_{old} + \gamma y_i$$

Cost Function

- Quantify how good of the classifier.

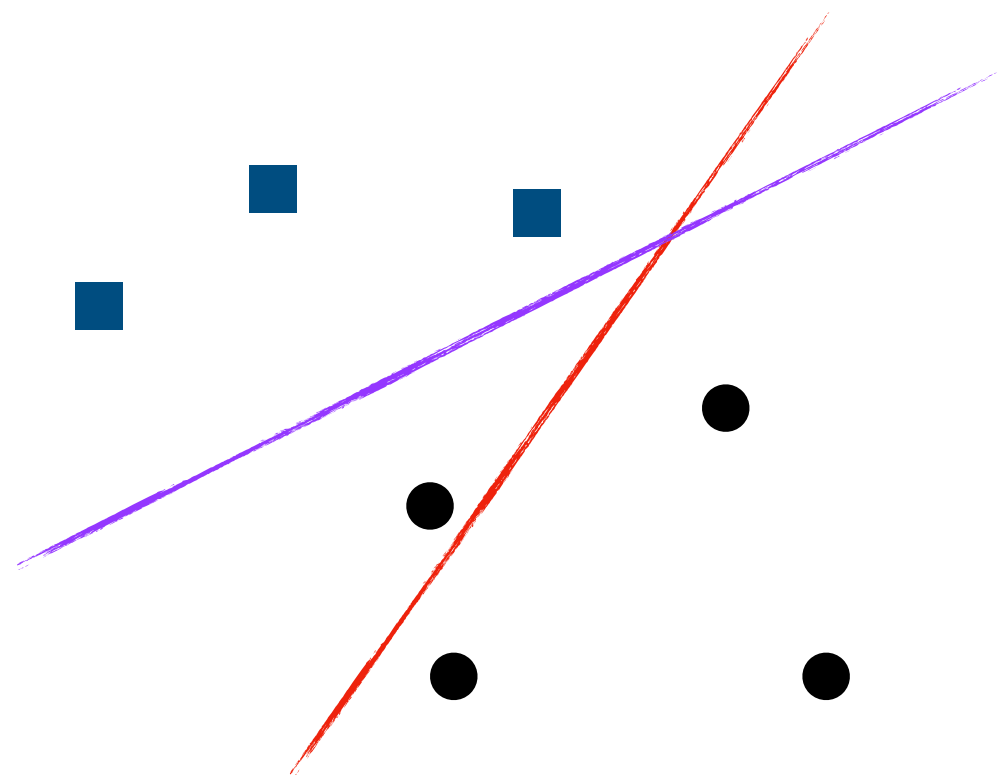
$$\ell(\underline{\beta}, \beta_0) = - \sum_{i=1}^M y_i \left(\underline{\beta}^\top \underline{x}_i + \beta_0 \right)$$

M is the number of misclassified points.

$$\frac{\partial \ell}{\partial \underline{\beta}} = - \sum_{i=1}^M y_i \underline{x}_i$$

$$\frac{\partial \ell}{\partial \beta_0} = - \sum_{i=1}^M y_i$$

Which line is better?



The Whole Pipeline of Perceptron

- For $t=1$ to T
 - For $l=1$ to N
 - If $y_i \left(\underline{\beta}^\top \underline{x} + \beta_0 \right) \leq 0$,
 - $\underline{\beta} \leftarrow \underline{\beta} + \rho y_i \underline{x}_i$
 - $\underline{\beta} \leftarrow \underline{\beta} + \rho y_i$

The Whole Pipeline of Perceptron

For t=1 to T • total learning iterations
 For i=1 to N • total data points
 # incorrect prediction

If $\underline{y}_i \left(\underline{\beta}^\top \underline{x} + \beta_0 \right) \leq 0$,
 # Update parameters of perceptron

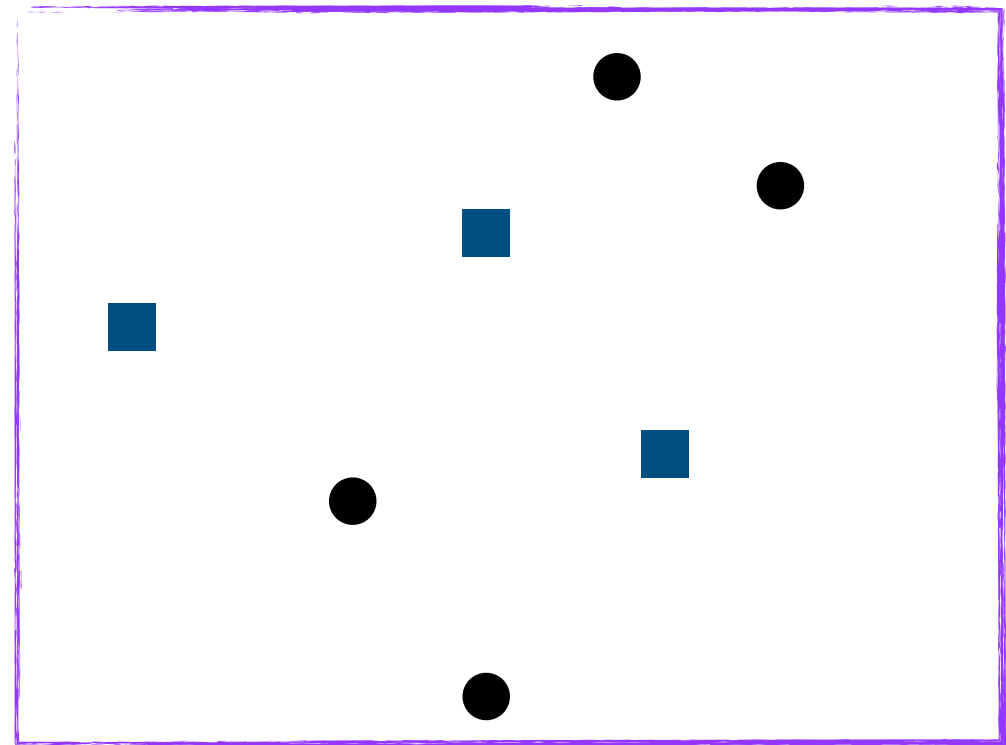
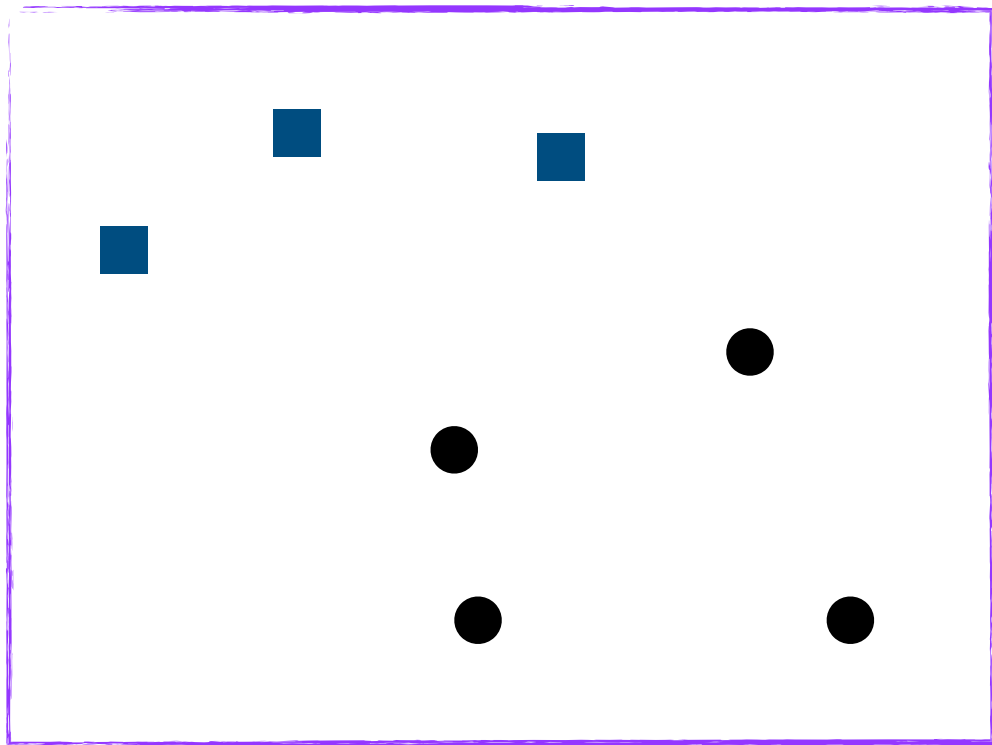
$$\underline{\beta} \leftarrow \underline{\beta} + \rho y_i \underline{x}_i$$

$$\underline{\beta} \leftarrow \underline{\beta} + \rho y_i$$

Linear separable assumption

- There exists $\delta > 0, \underline{\beta}, \beta_0$, such that:

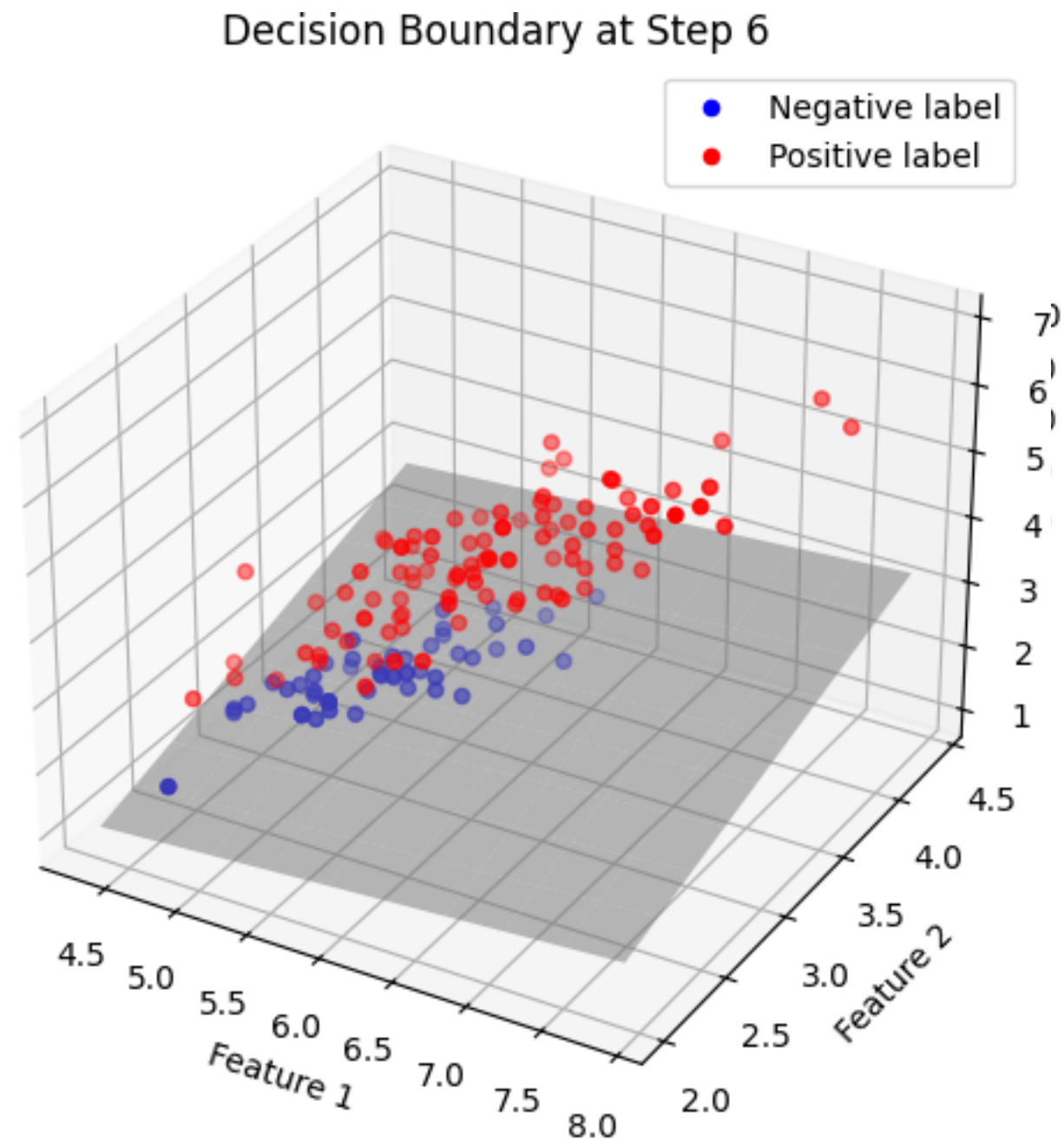
$$y_i \left(\underline{\beta}^\top \underline{x}_i + \beta_0 \right) \geq \delta, \quad \text{for all } y_i, \underline{x}_i$$



Code Implementation Demo

```
def Perceptron(X, y, gamma=0.1, T=50):  
    # N = num_samples, d = num_features  
    N, d = X.shape  
  
    # Randomly initialize weights and bias  
    beta = np.random.uniform(-1.0, 1.0, size=d)  
    beta0 = np.random.uniform(-1.0, 1.0)  
  
    # Iterate over the dataset for T epochs  
    for t in range(T):  
        for xi, yi in zip(X, y):  
            if yi * (np.dot(xi, beta) + beta0) <= 0:  
                # Update weights  
                beta += gamma * yi * xi  
                beta0 += gamma * yi  
    return beta, beta0
```

Visualize the process



<https://github.com/jiangnanhugo/intro-to-ML/>

Thank you!