

# Machine Learning & Data Mining

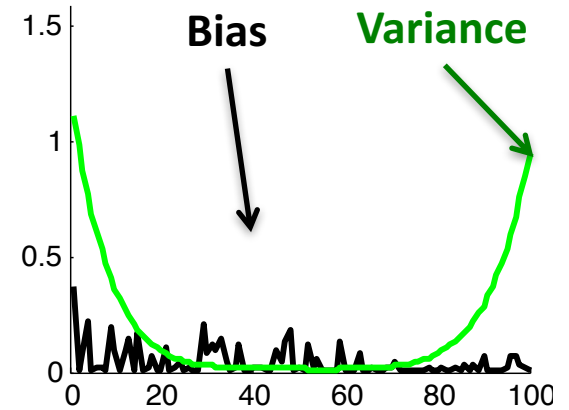
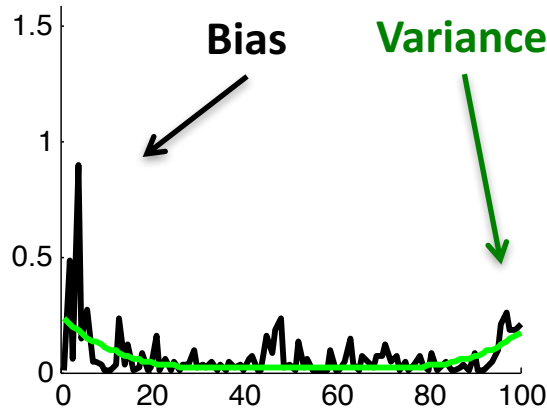
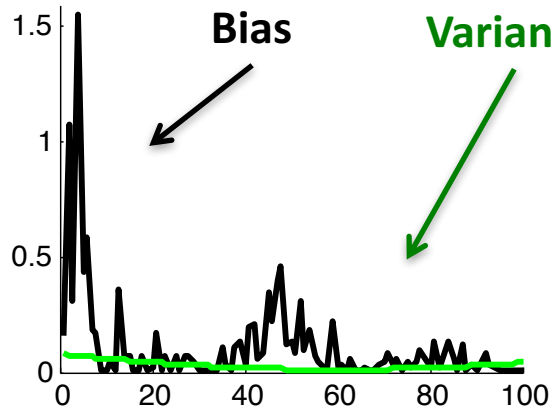
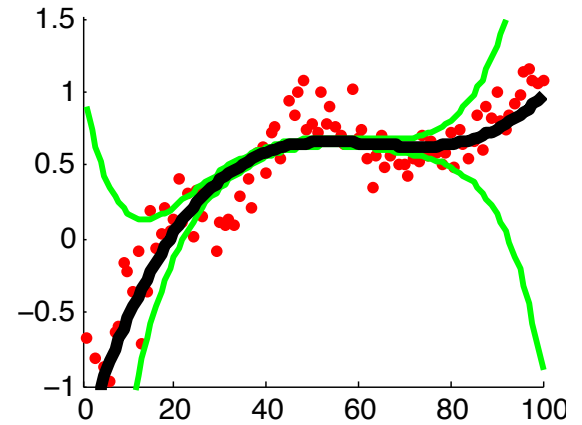
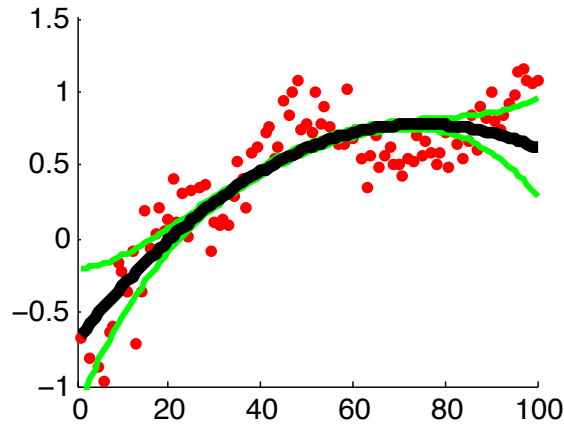
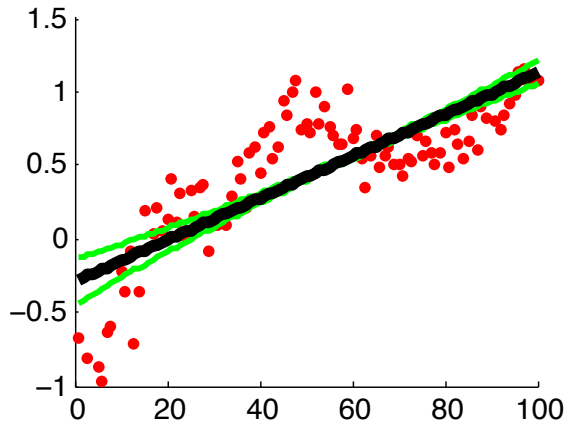
## **CMS/CS/CNS/EE 155**

Lecture 2:  
Perceptron & Stochastic Gradient  
Descent

# Recap: Basic Recipe (supervised)

- Training Data:  $S = \{(x_i, y_i)\}_{i=1}^N$   $x \in R^D$   
 $y \in \{-1, +1\}$
- Model Class:  $f(x | w, b) = w^T x - b$  **Linear Models**
- Loss Function:  $L(a, b) = (a - b)^2$  **Squared Loss**
- Learning Objective:  $\operatorname{argmin}_{w, b} \sum_{i=1}^N L(y_i, f(x_i | w, b))$   
**Optimization Problem**

# Recap: Bias-Variance Trade-off



# Recap: Complete Pipeline

$$S = \{(x_i, y_i)\}_{i=1}^N$$

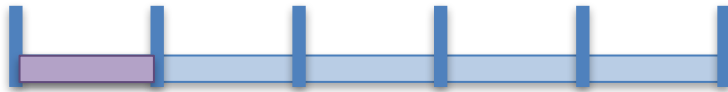
Training Data

$$f(x | w, b) = w^T x - b$$

Model Class(es)

$$L(a, b) = (a - b)^2$$

Loss Function



$$\operatorname{argmin}_{w, b} \sum_{i=1}^N L(y_i, f(x_i | w, b))$$

Cross Validation & Model Selection



Profit!

# Today

- **Two Basic Learning Algorithms**
- Perceptron Algorithm
- (Stochastic) Gradient Descent
  - Aka, actually solving the optimization problem

# The Perceptron

- One of the earliest learning algorithms
  - 1957 by Frank Rosenblatt
- Still a great algorithm
  - Fast
  - Clean analysis
  - Precursor to Neural Networks



Frank Rosenblatt  
with the Mark 1  
Perceptron Machine

# Perceptron Learning Algorithm

## (Linear Classification Model)

- $w^1 = 0, b^1 = 0$

$$f(x | w) = \text{sign}(w^T x - b)$$

- For  $t = 1 \dots$

- Receive example  $(x, y)$

- If  $f(x | w^t) = y$

- $[w^{t+1}, b^{t+1}] = [w^t, b^t]$

- Else

- $w^{t+1} = w^t + yx$

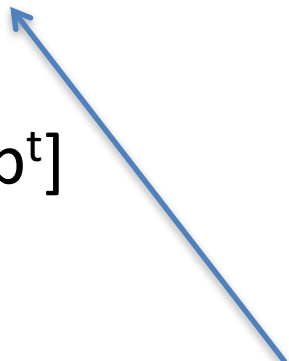
- $b^{t+1} = b^t - y$

**Training Set:**

$$S = \{(x_i, y_i)\}_{i=1}^N$$

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

Go through training set  
in arbitrary order  
(e.g., randomly)

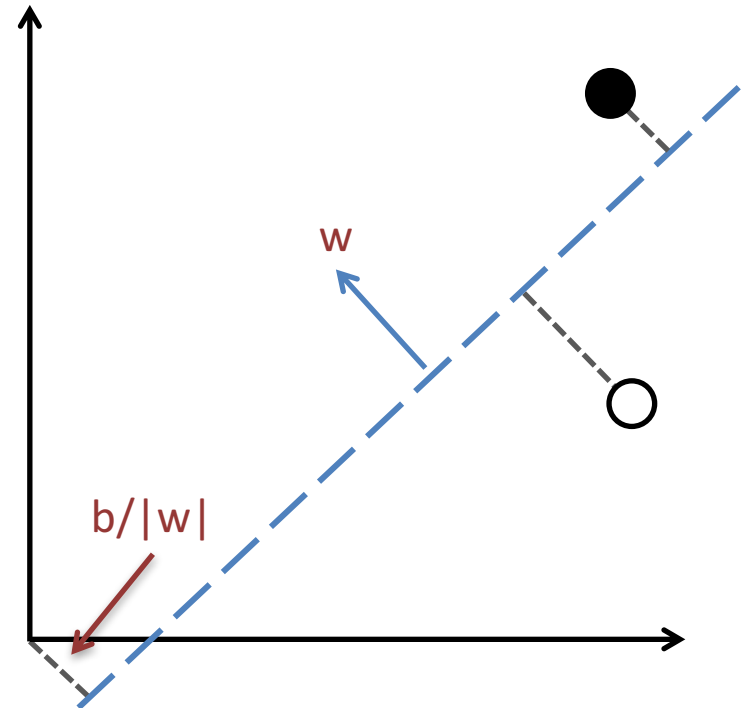


# Aside: Hyperplane Distance

- Line is a 1D, Plane is 2D
- Hyperplane is many D
  - Includes Line and Plane
- Defined by  $(w, b)$

- Distance: 
$$\frac{|w^T x - b|}{\|w\|}$$

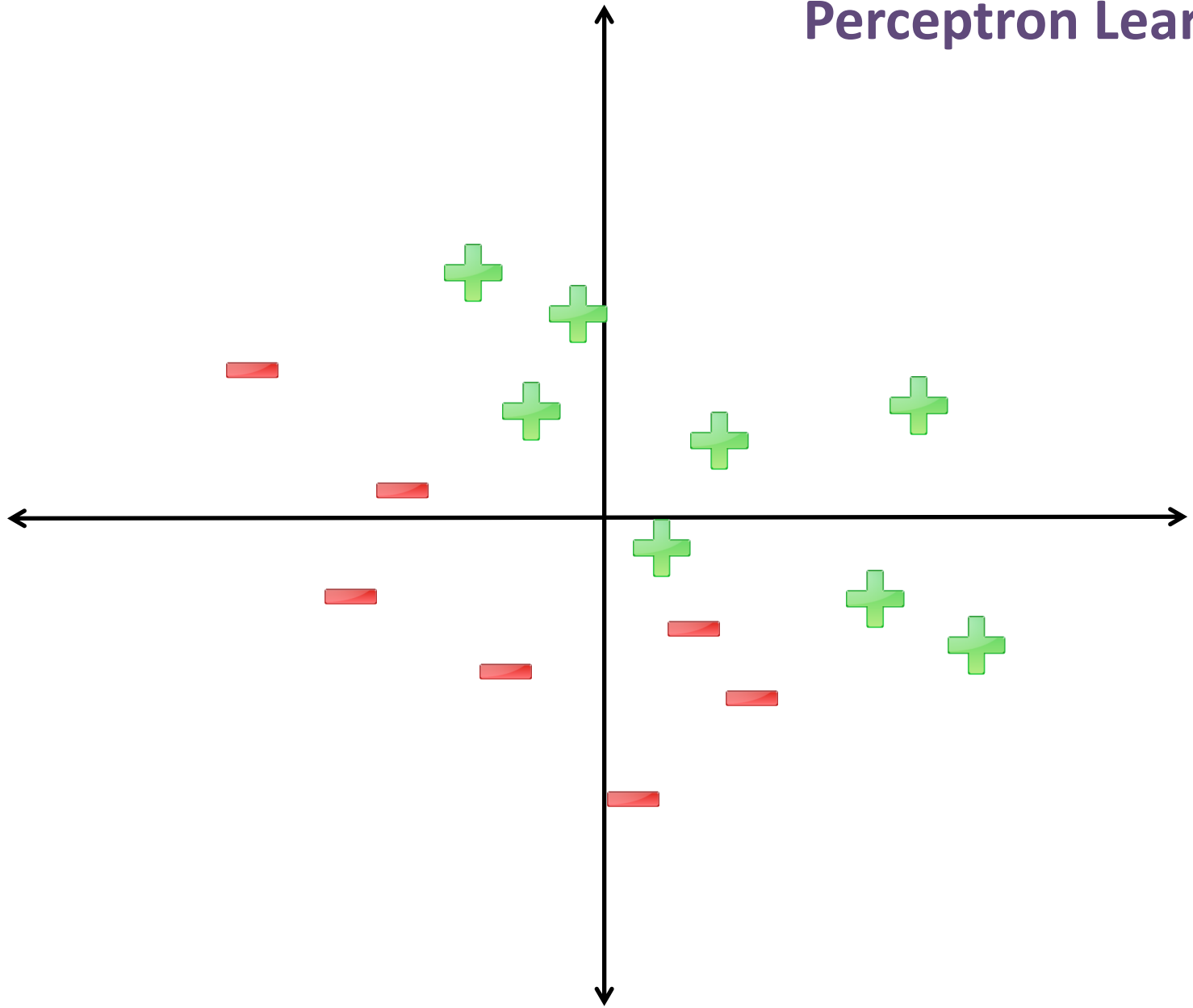
- Signed Distance: 
$$\frac{w^T x - b}{\|w\|}$$



**Linear Model = un-normalized signed distance!**

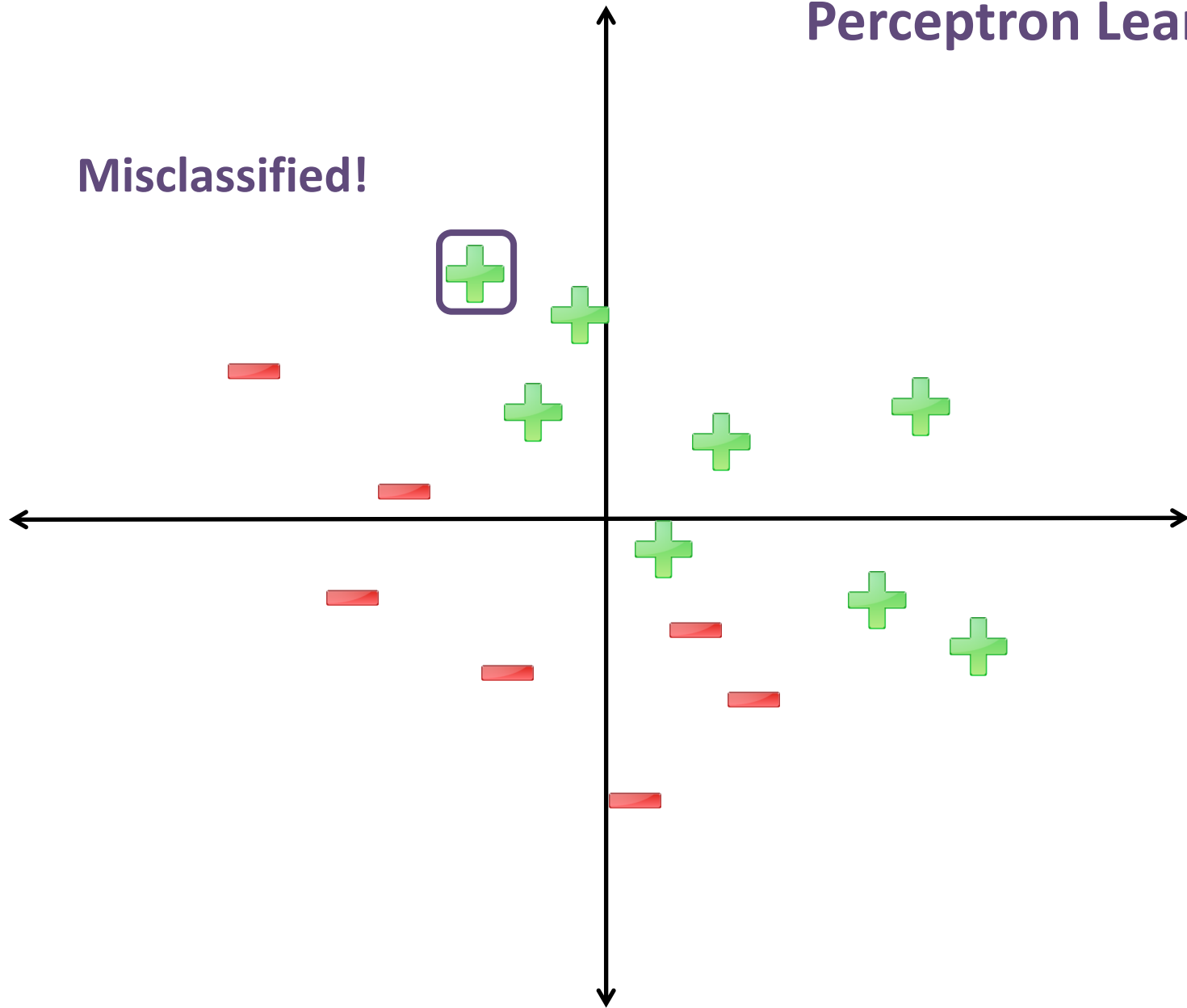


# Perceptron Learning

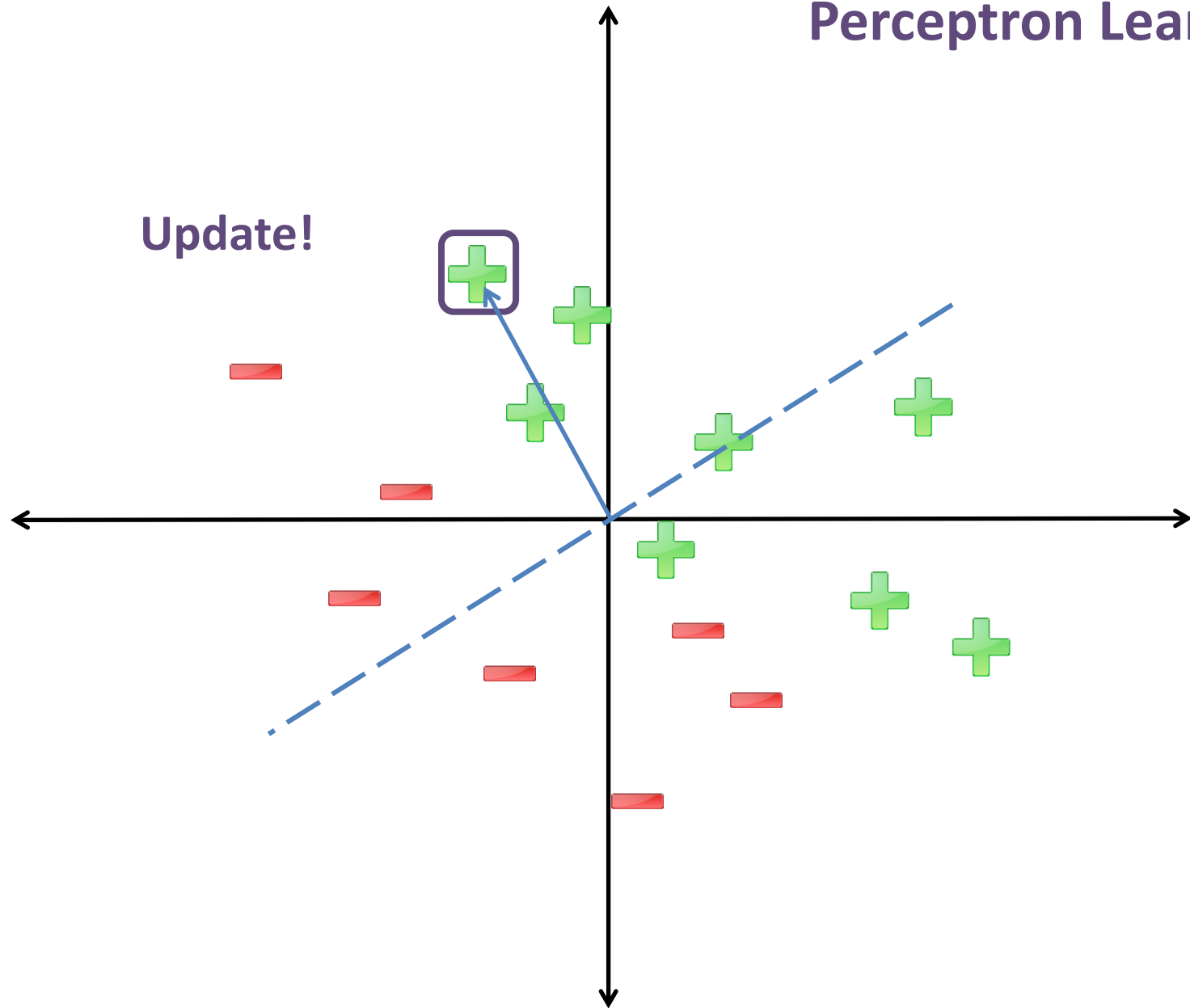


# Perceptron Learning

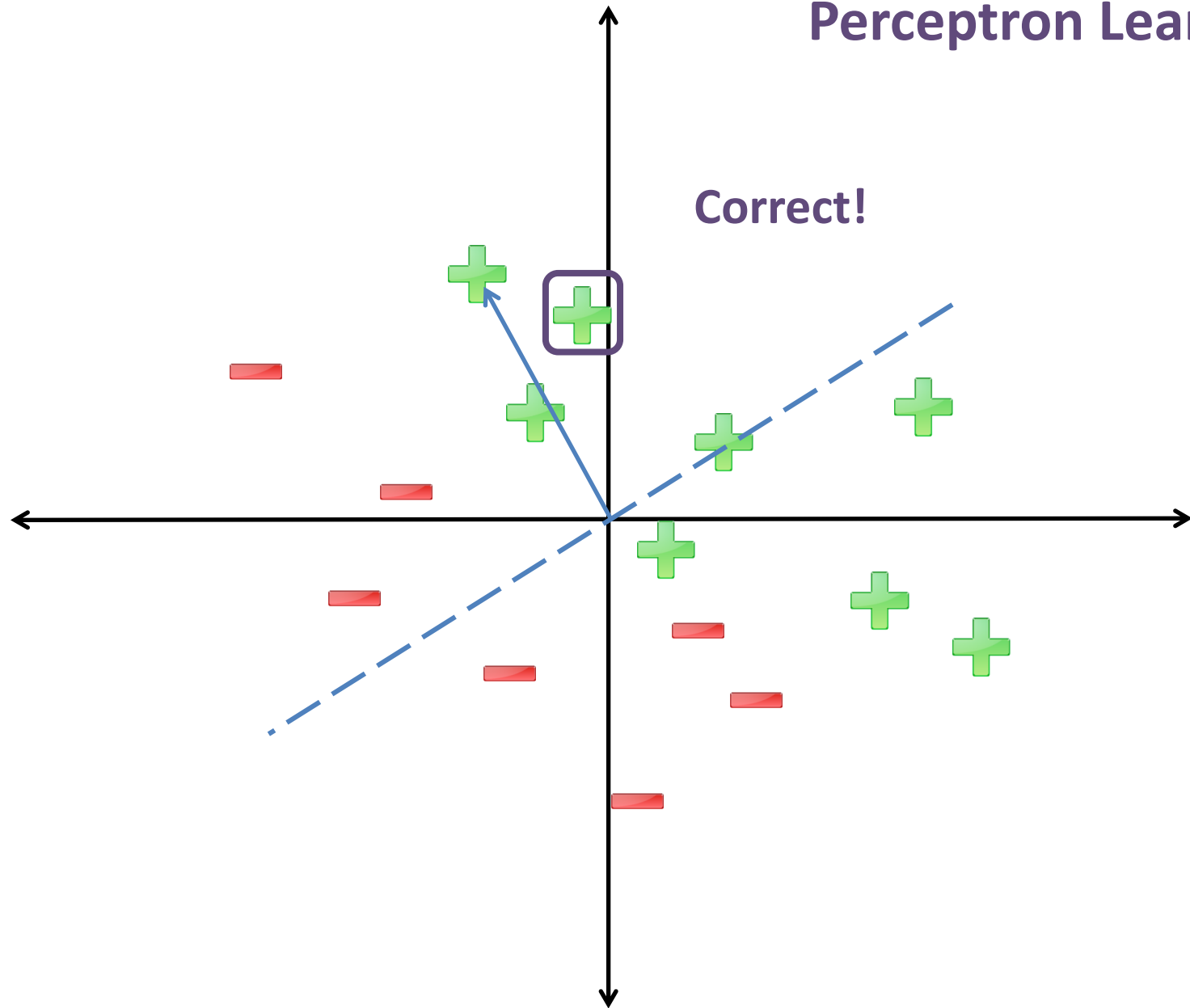
Misclassified!



# Perceptron Learning

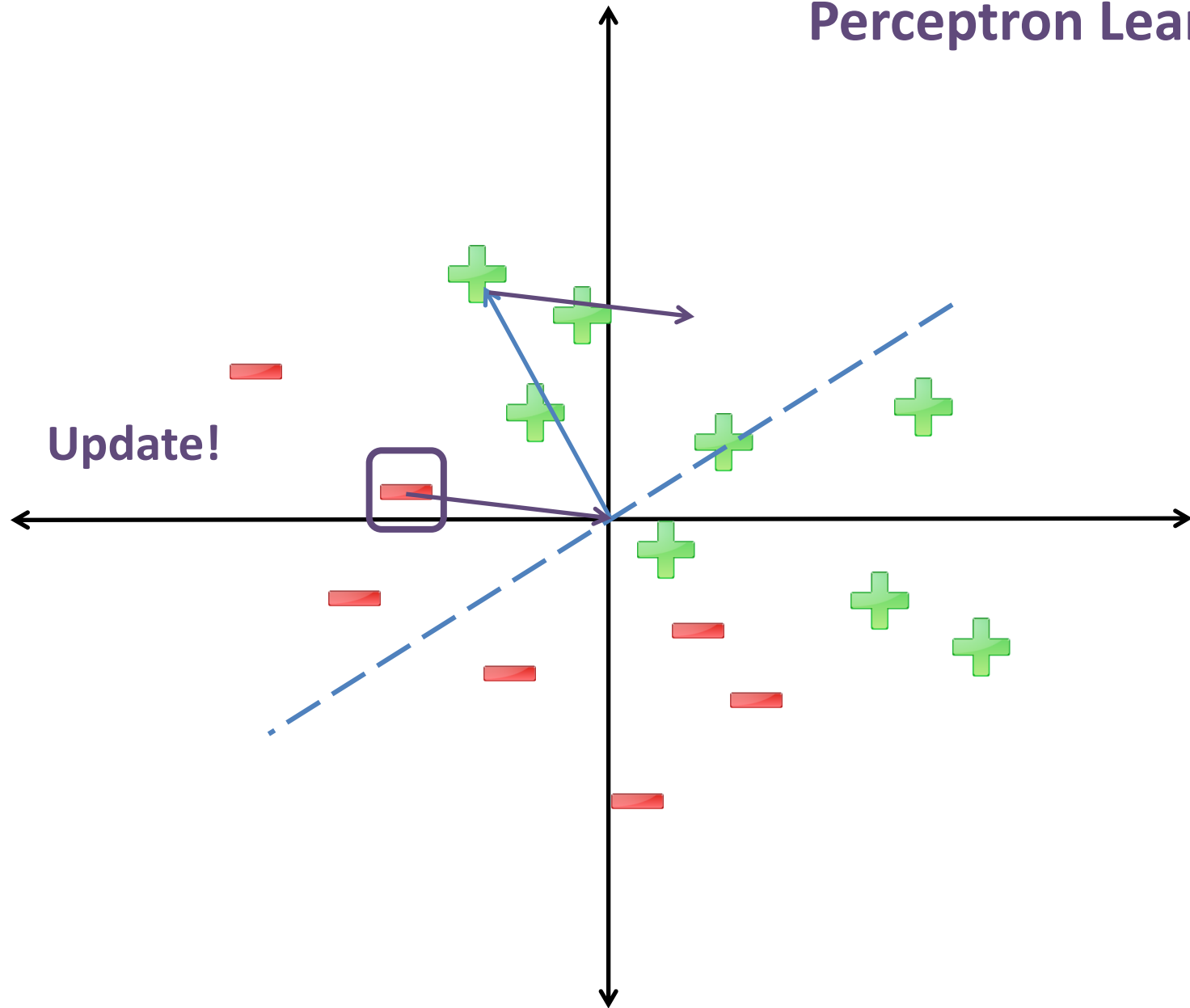


# Perceptron Learning

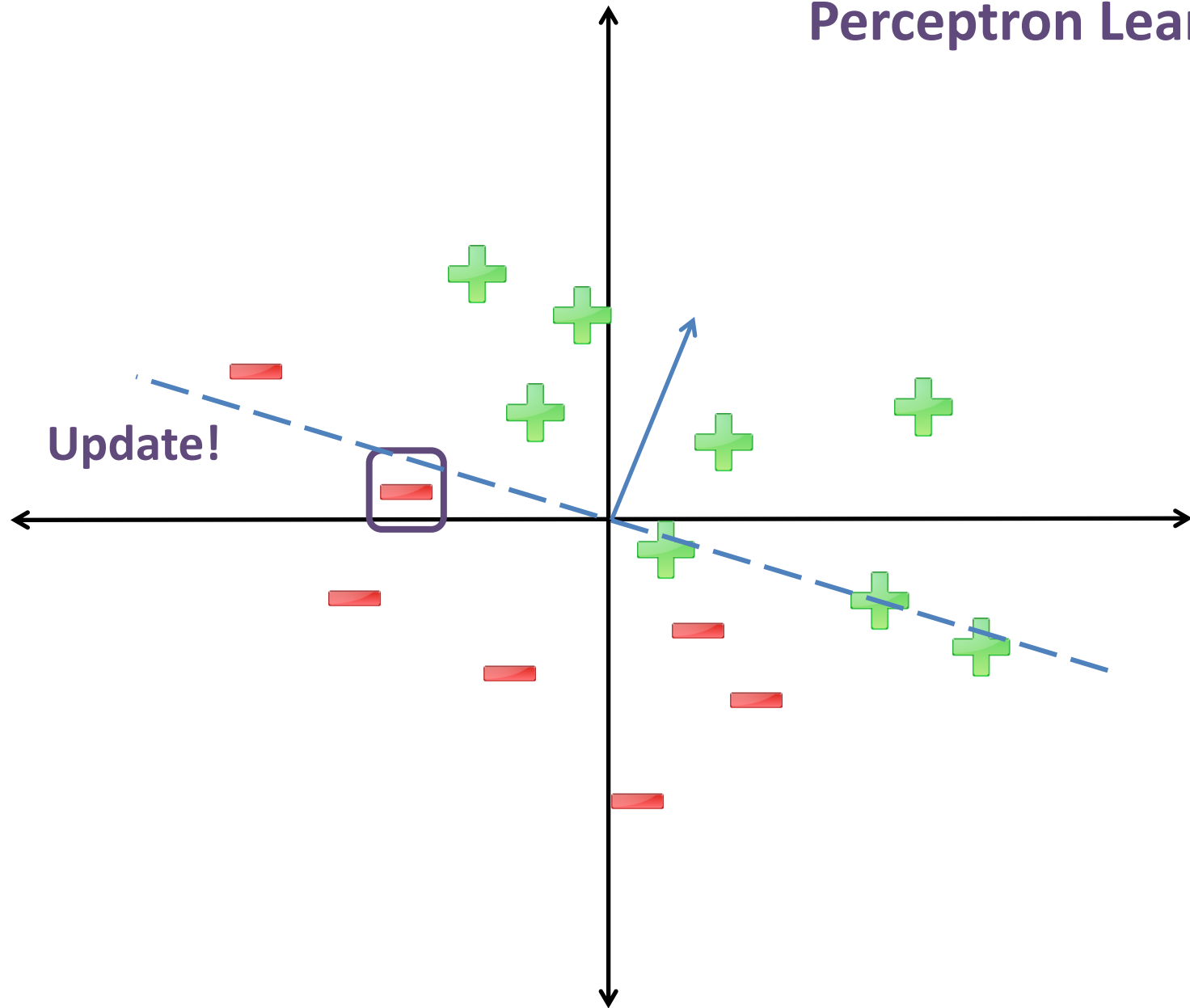


The diagram shows a 2D coordinate system with a horizontal x-axis and a vertical y-axis. A dashed blue line, representing the decision boundary, passes through the origin and extends from the bottom-left to the top-right. Data points are represented by red minus signs (-) and green plus signs (+). There are 10 red minus signs and 10 green plus signs scattered across the plot. A solid blue vector originates from the origin and points towards a red minus sign located in the upper-left quadrant. This red minus sign is enclosed in a purple rectangular box. To the left of this box, the text "Misclassified!" is written in a bold, purple font. In the top-right corner of the plot area, the text "Perceptron Learning" is written in a purple font.

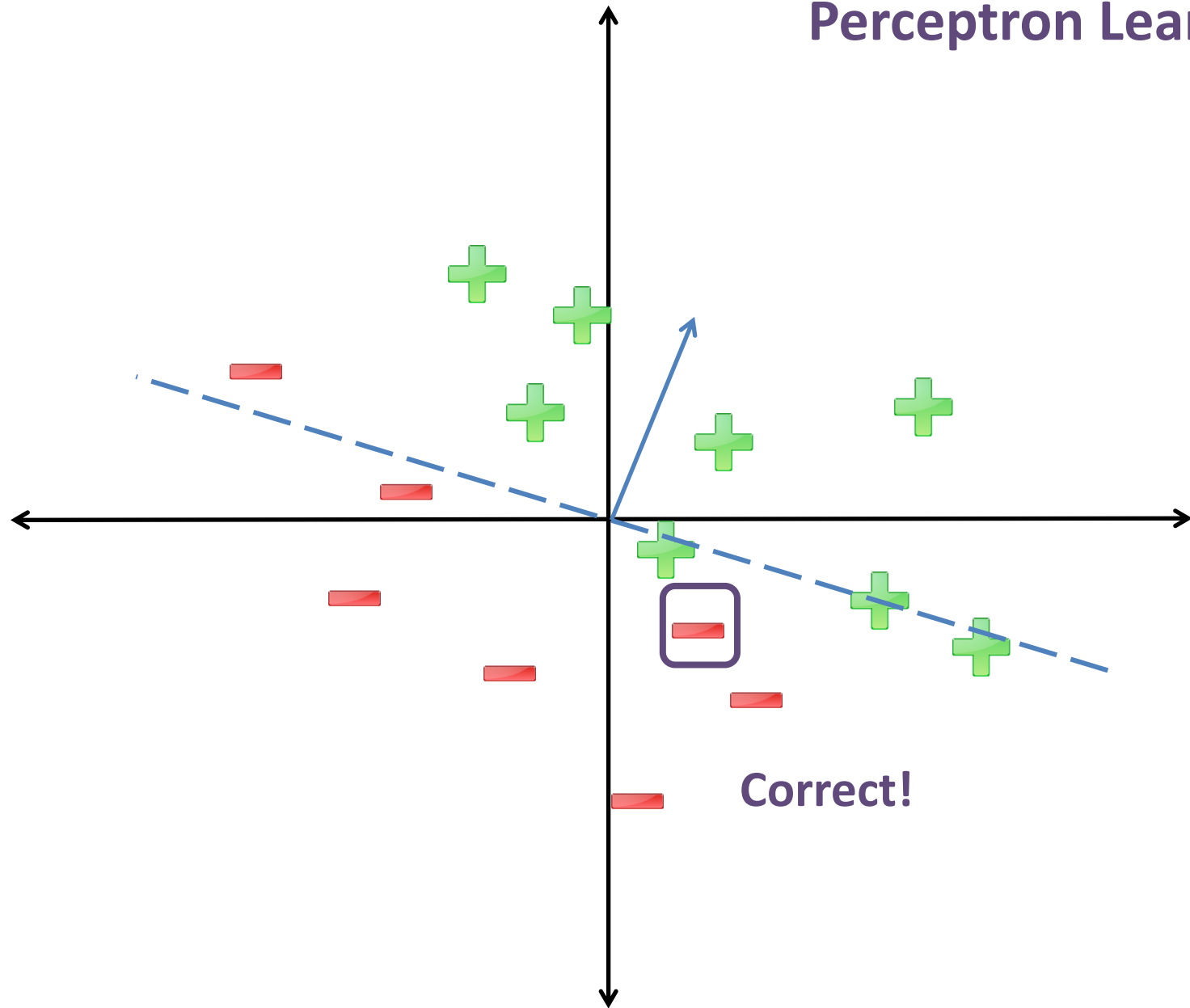
# Perceptron Learning



# Perceptron Learning

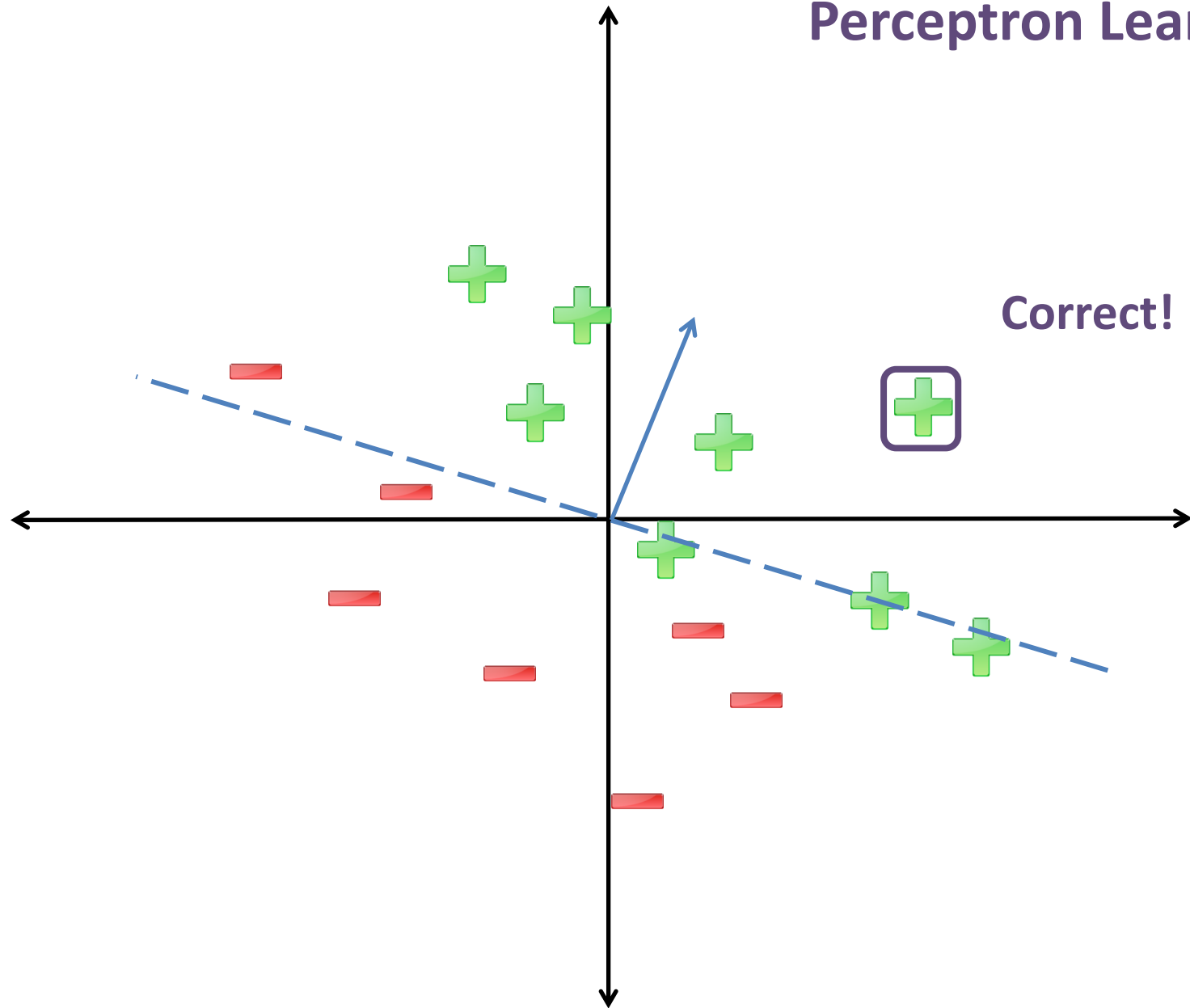


# Perceptron Learning

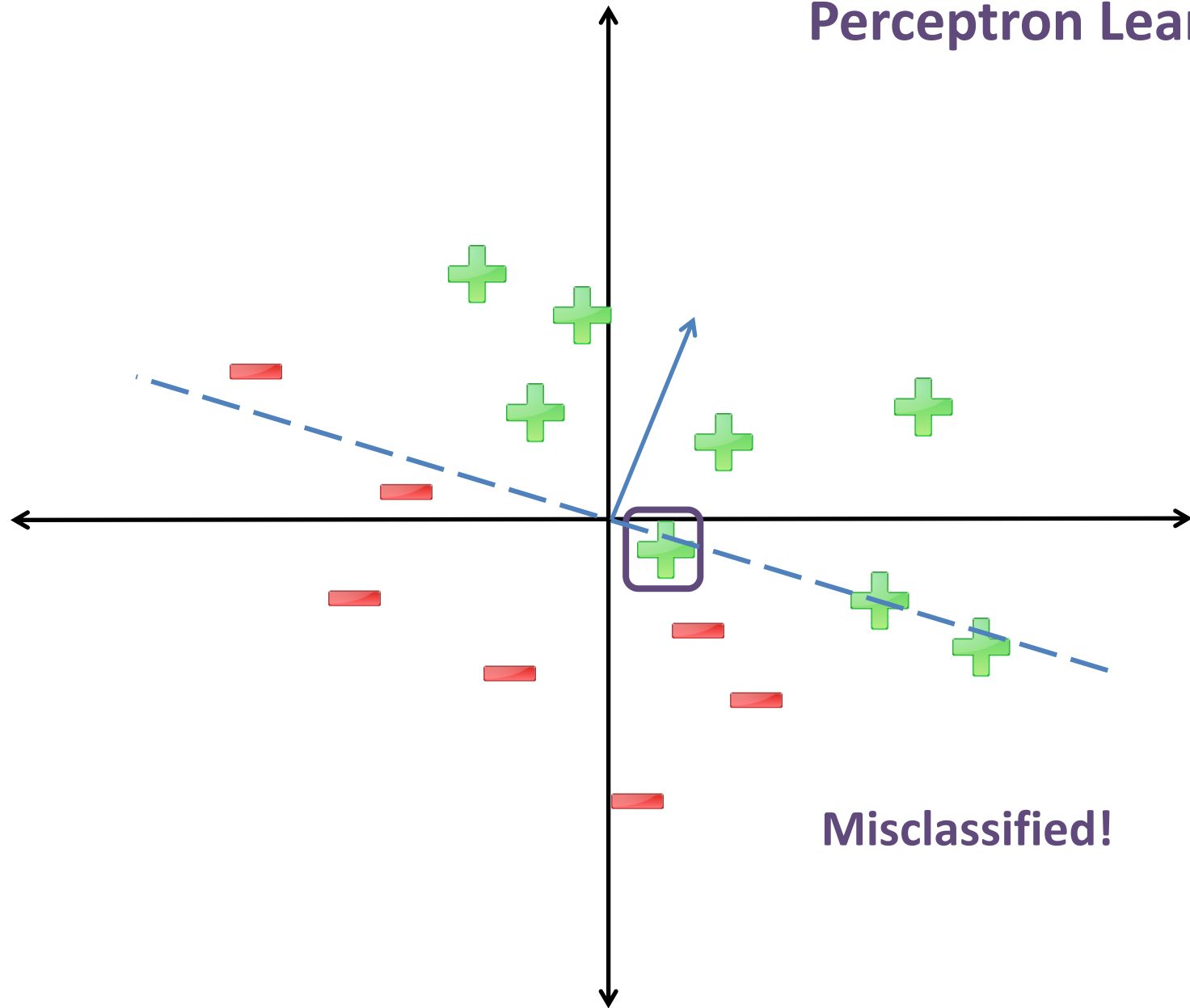




# Perceptron Learning



# Perceptron Learning



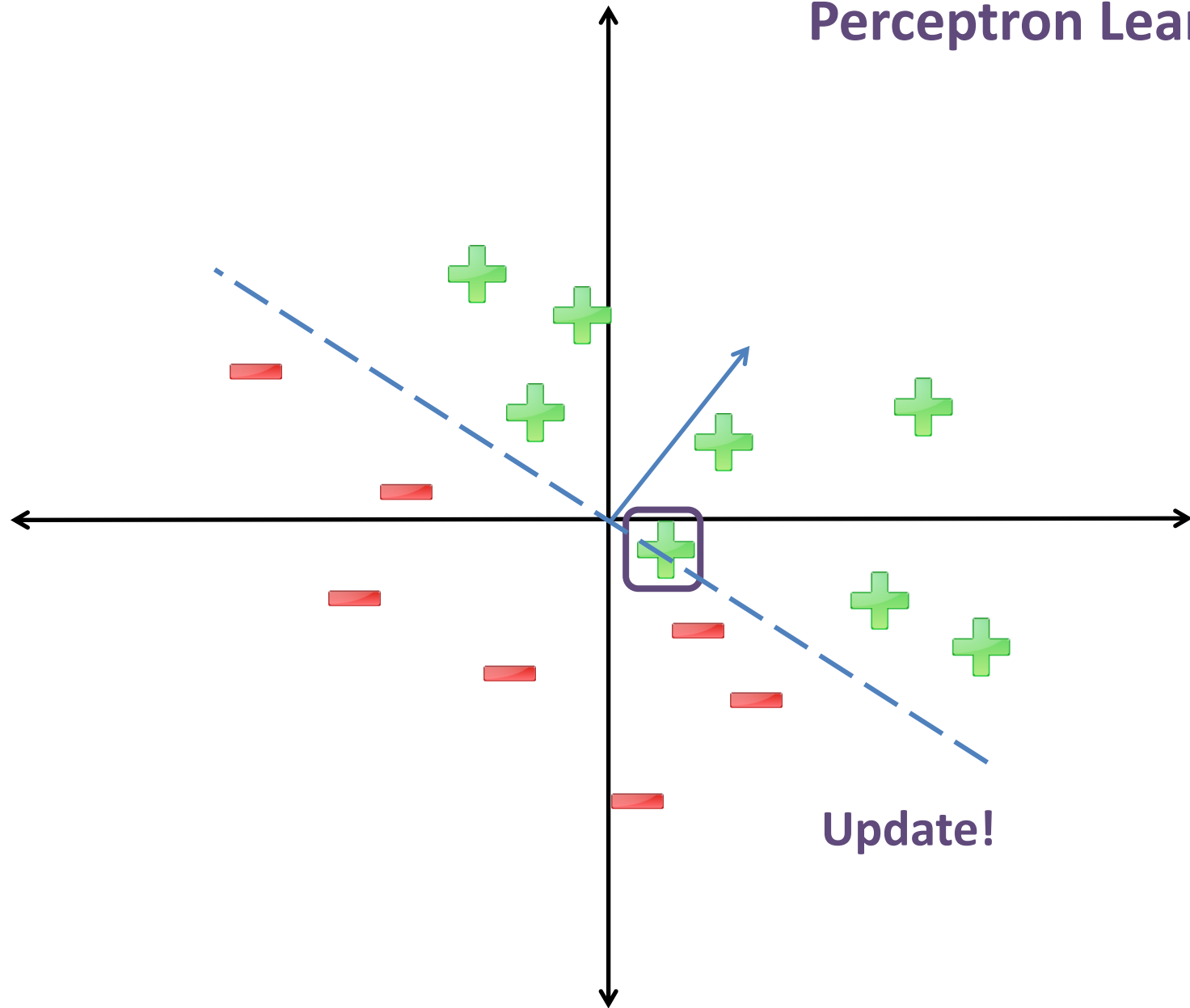
**Misclassified!**

Perceptron Learning

Update!

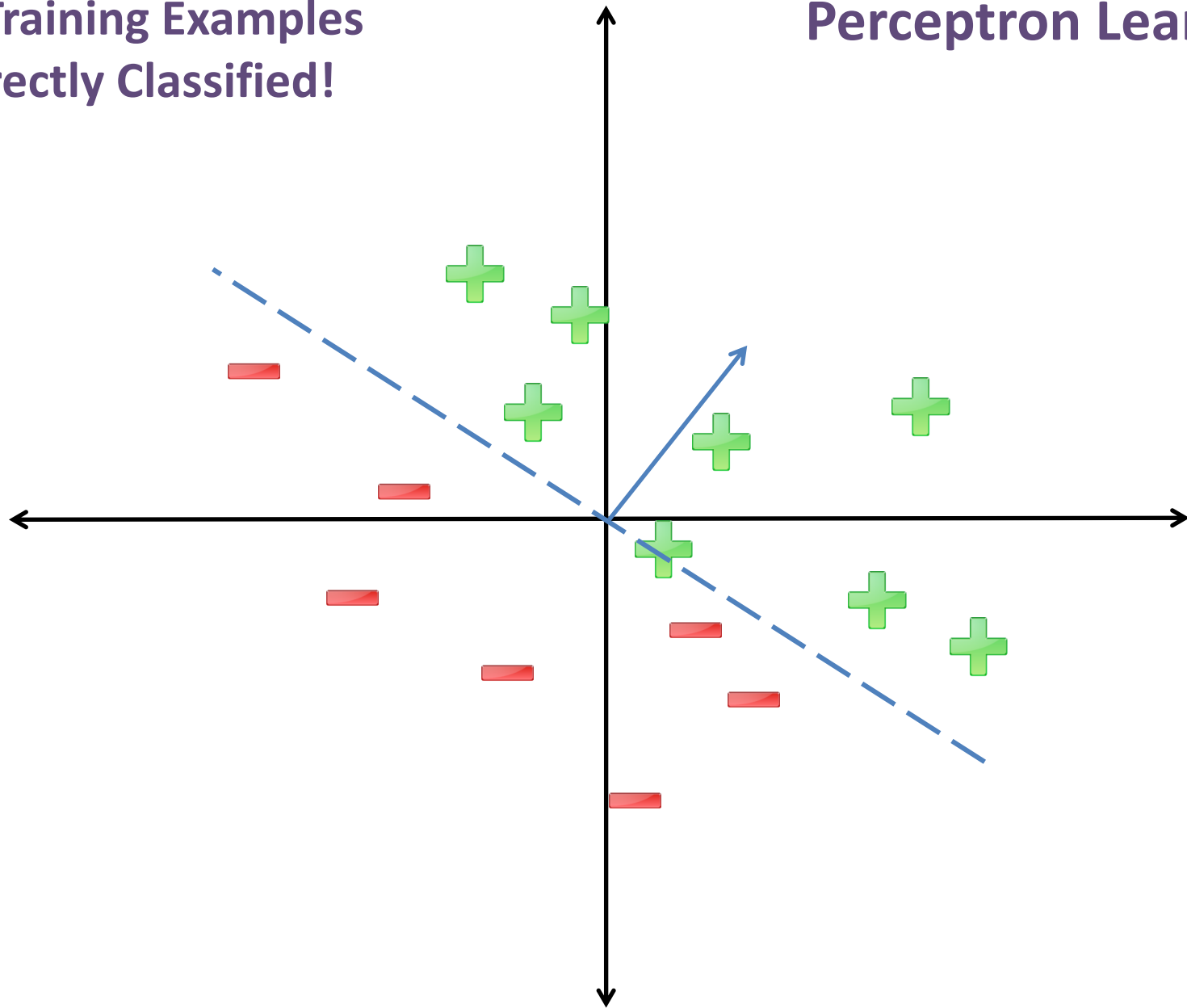
## Update!

# Perceptron Learning



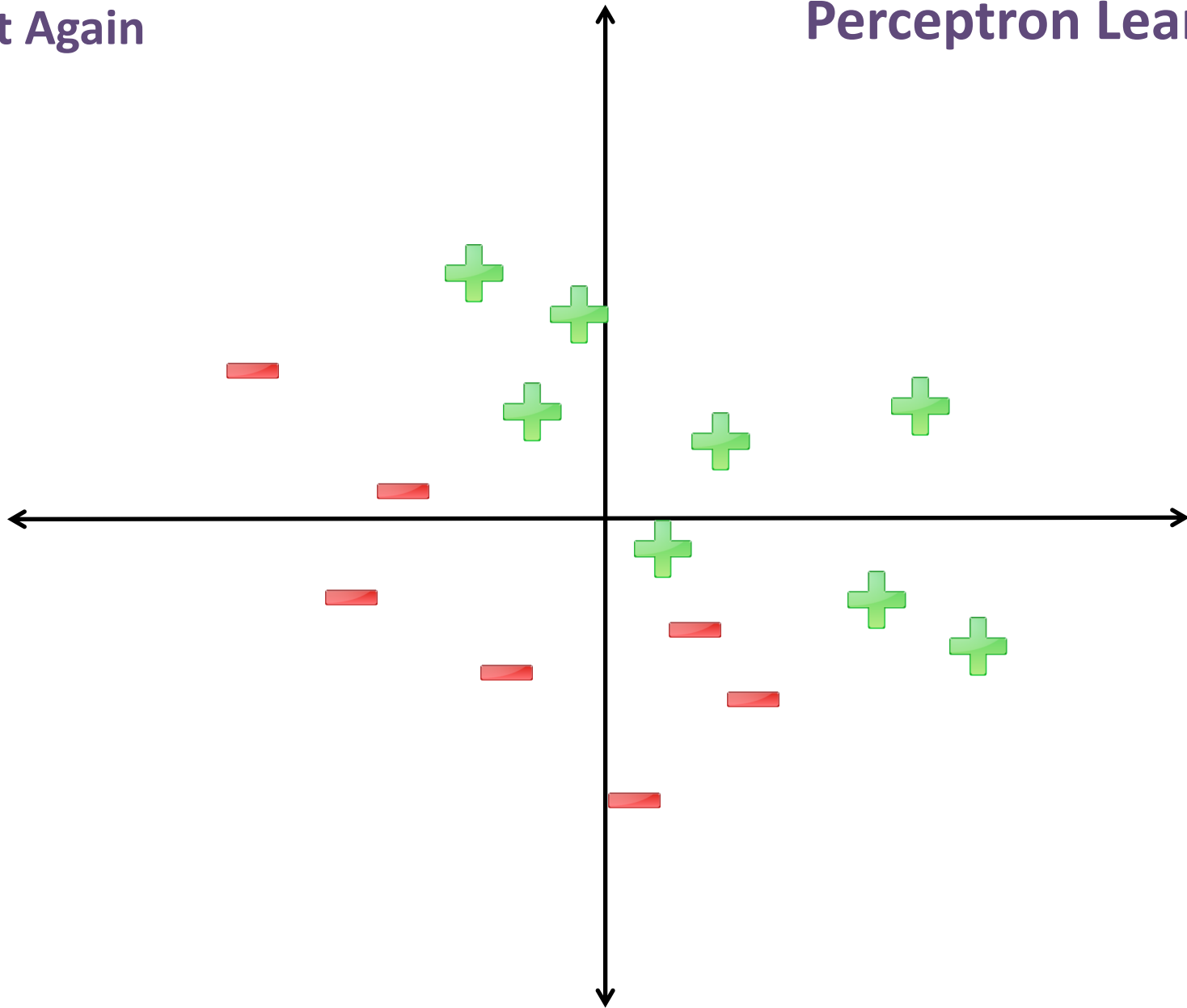
All Training Examples  
Correctly Classified!

## Perceptron Learning

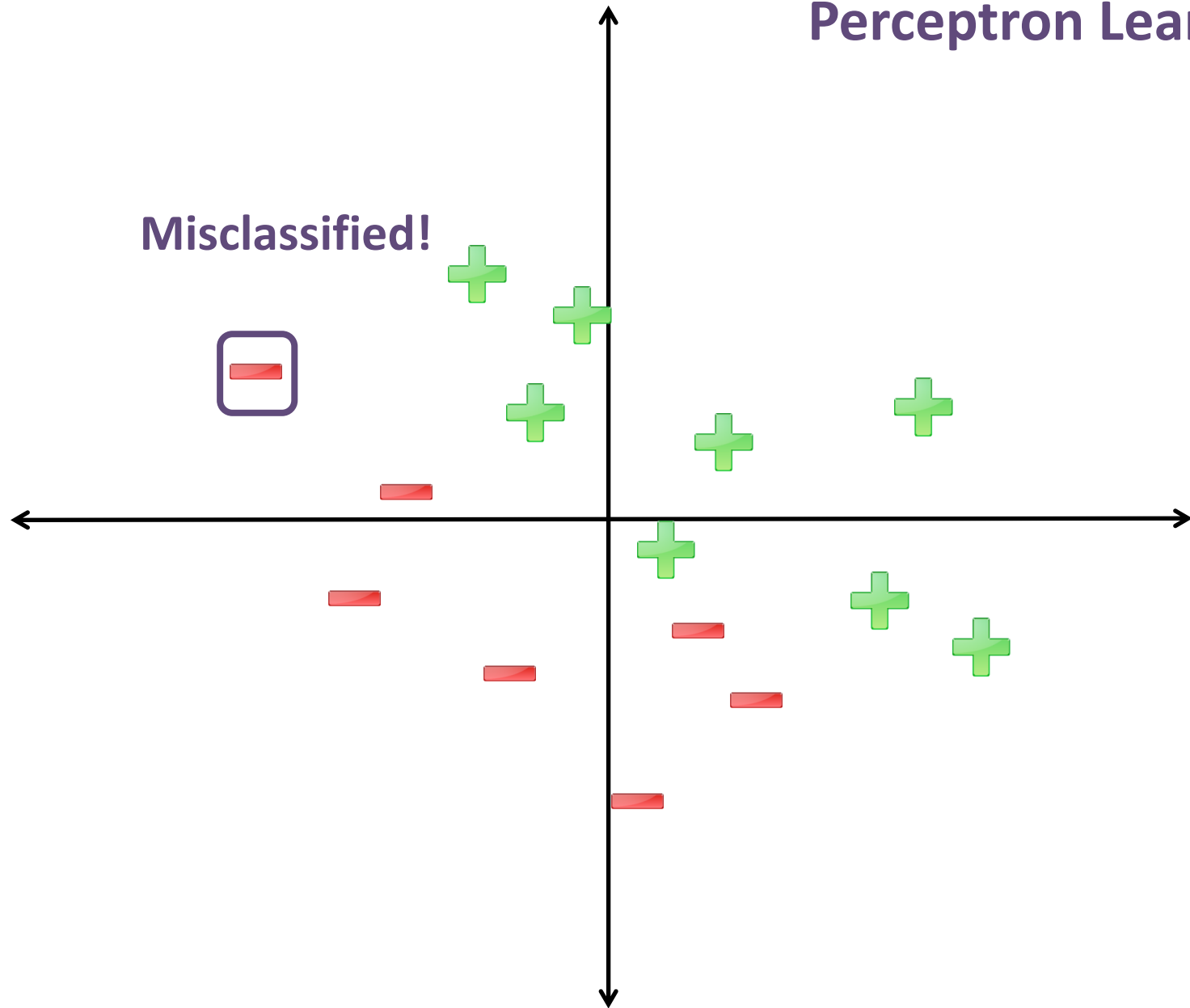


Start Again

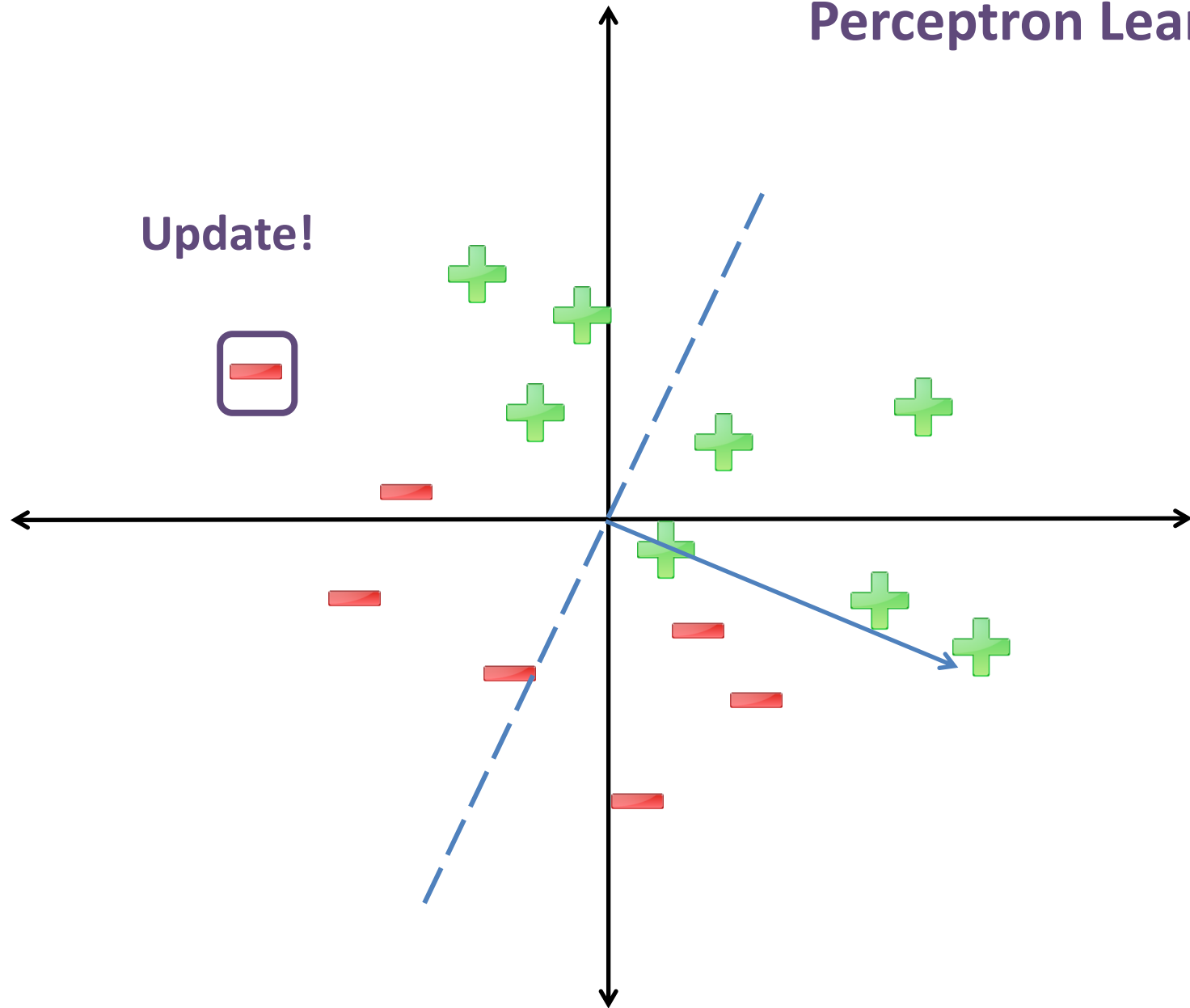
Perceptron Learning



# Perceptron Learning

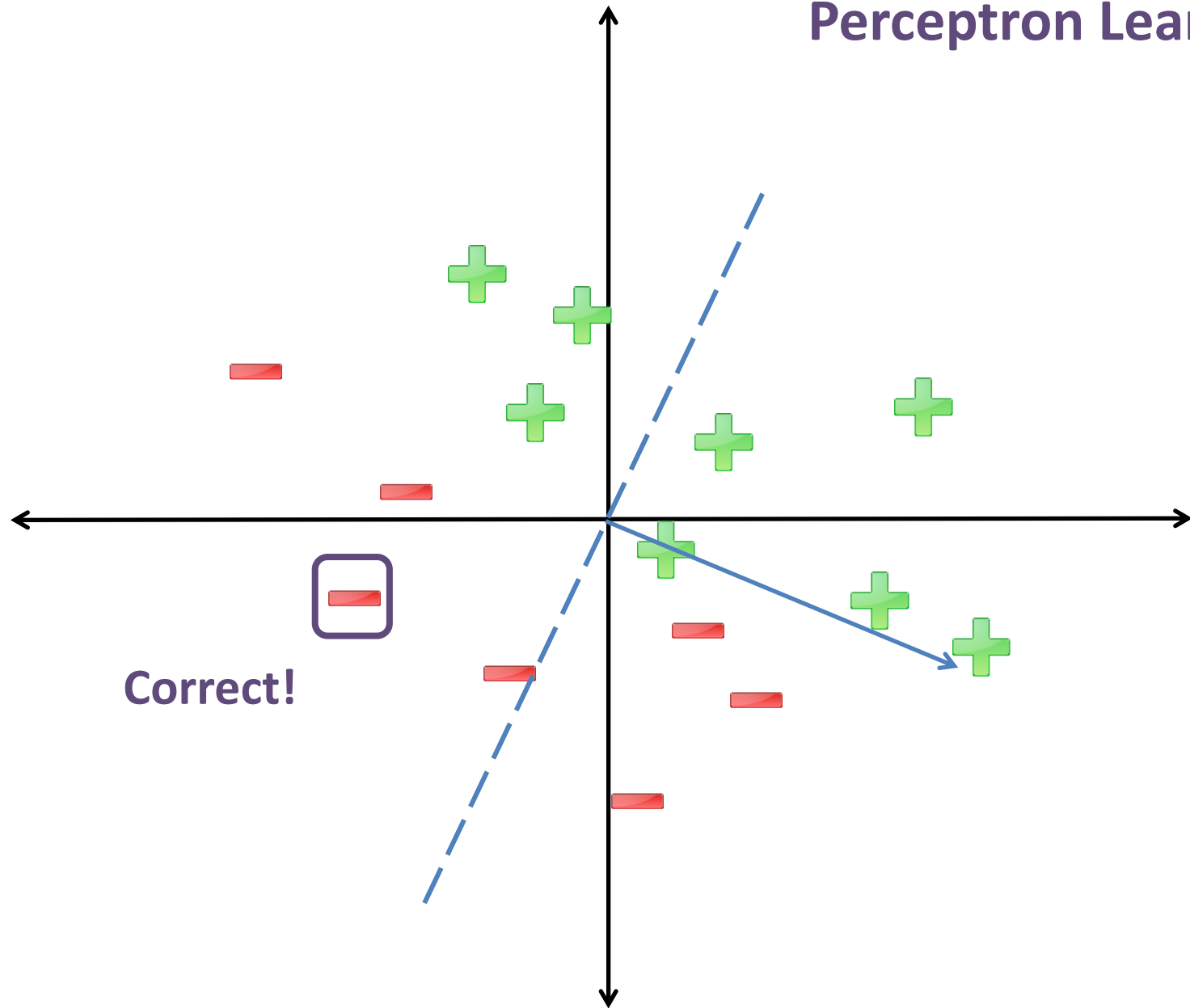


# Perceptron Learning



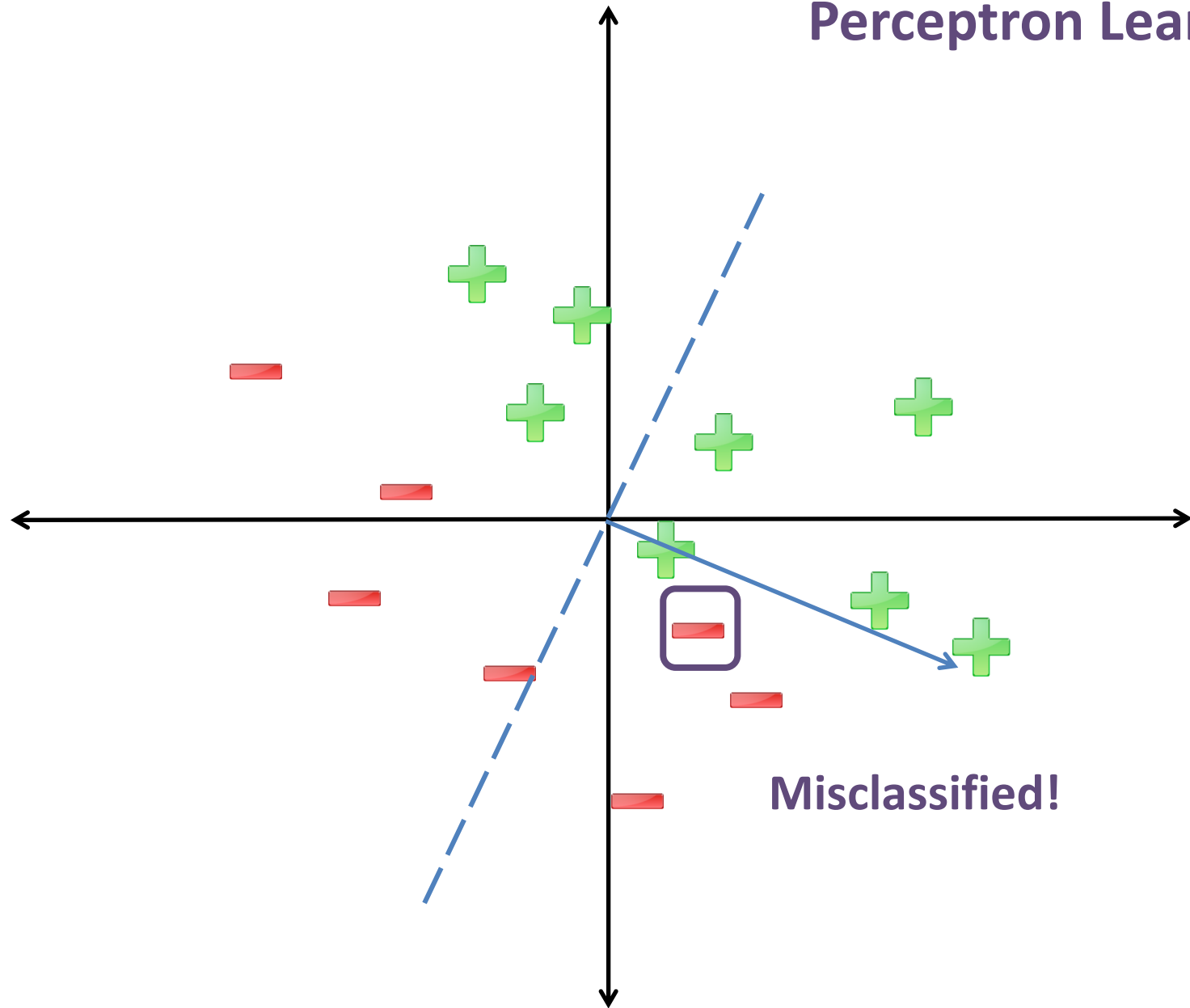


# Perceptron Learning

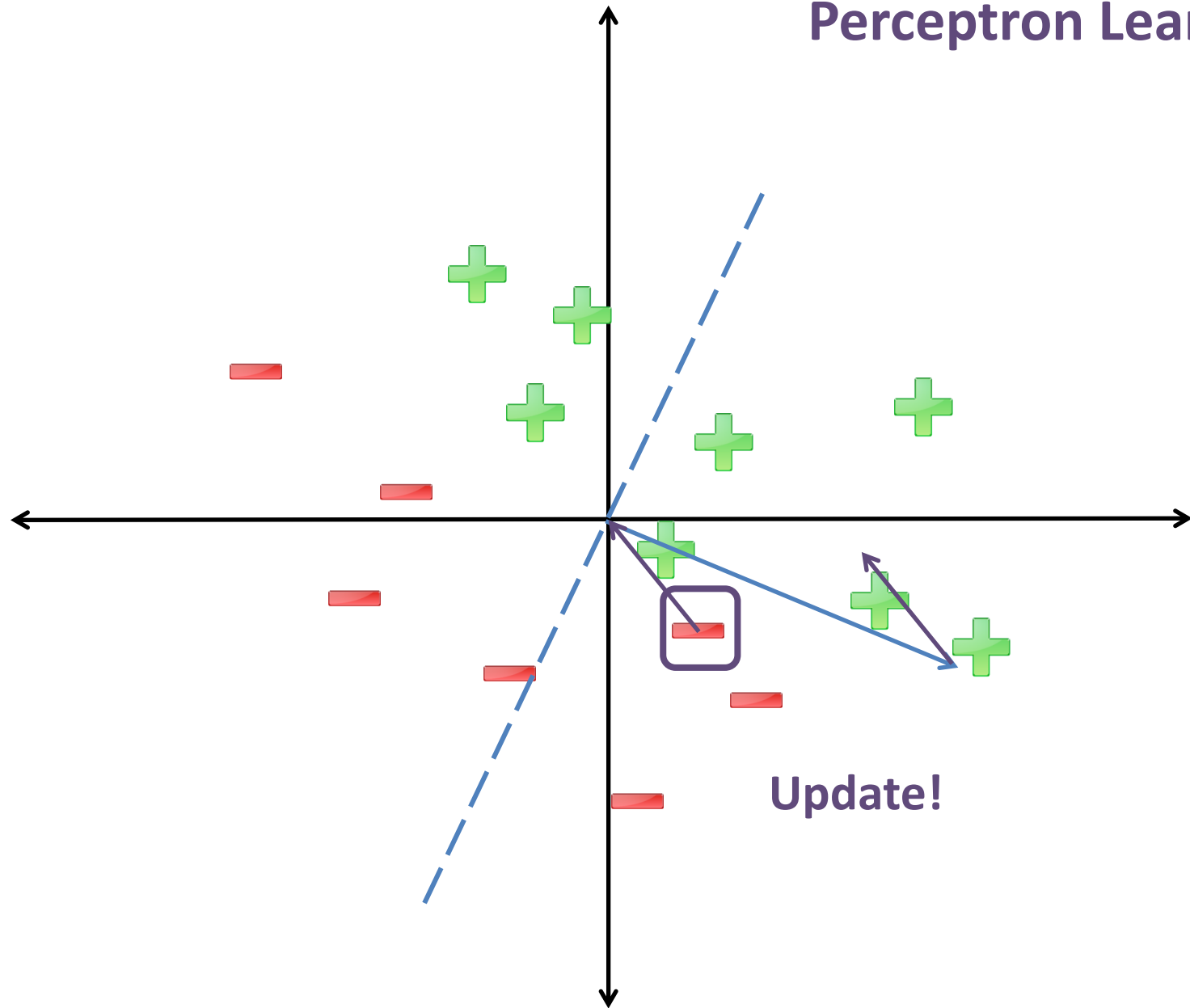


The diagram shows a 2D coordinate system with a horizontal and vertical axis, both ending in arrows. A dashed blue line, representing the decision boundary, passes through the origin and extends from the bottom-left to the top-right. There are 10 green '+' symbols and 10 red '-' symbols scattered across the plot. The '+' symbols are primarily located in the upper-left and upper-right quadrants, while the '-' symbols are primarily in the lower-left and lower-right quadrants. A solid blue vector originates from the origin and points towards the upper-right quadrant, ending near a '+' symbol. The text "Correct!" is written in a bold, black font near the tip of this vector. One '+' symbol, located in the upper-right quadrant, is enclosed in a black square box.

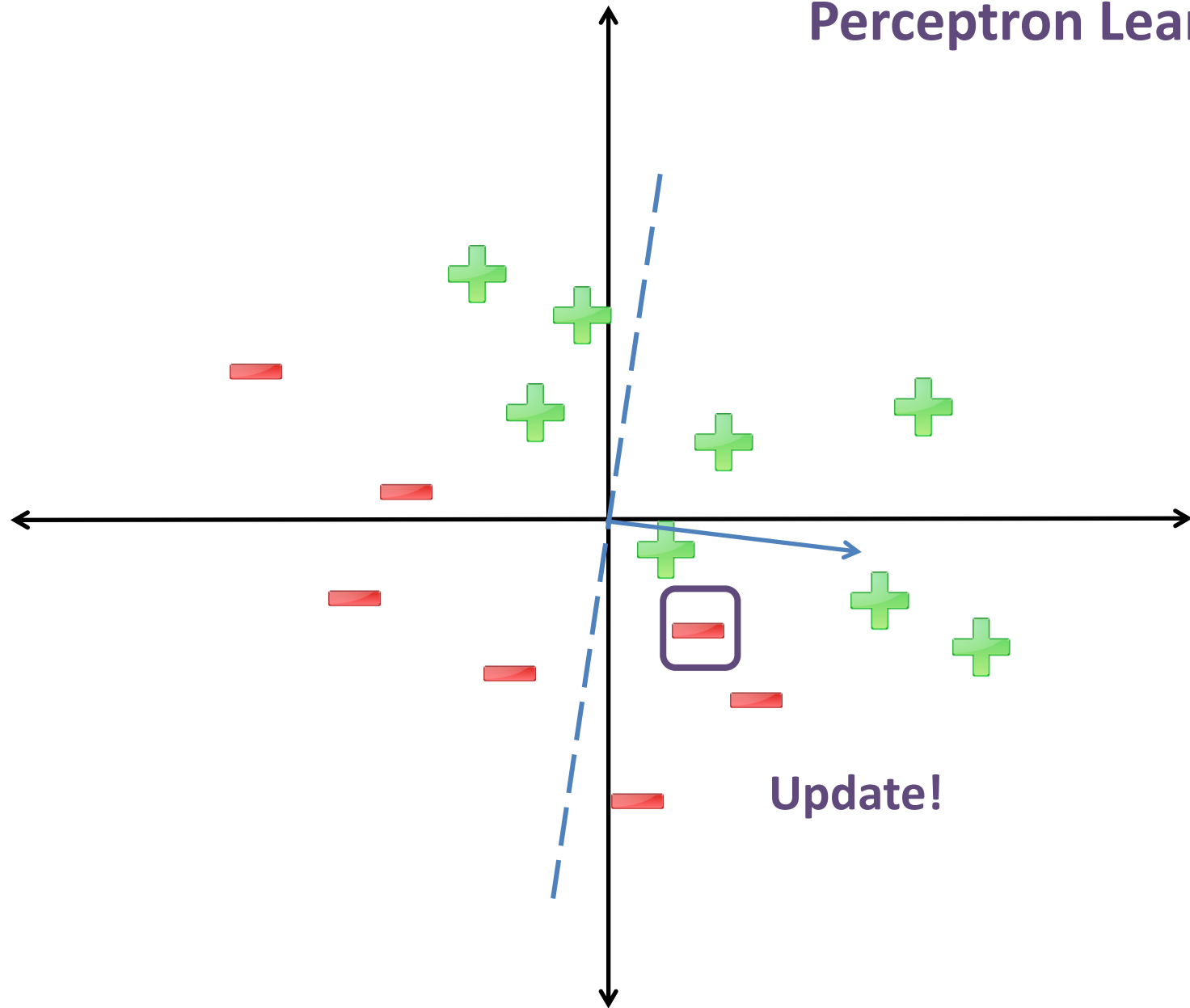
# Perceptron Learning



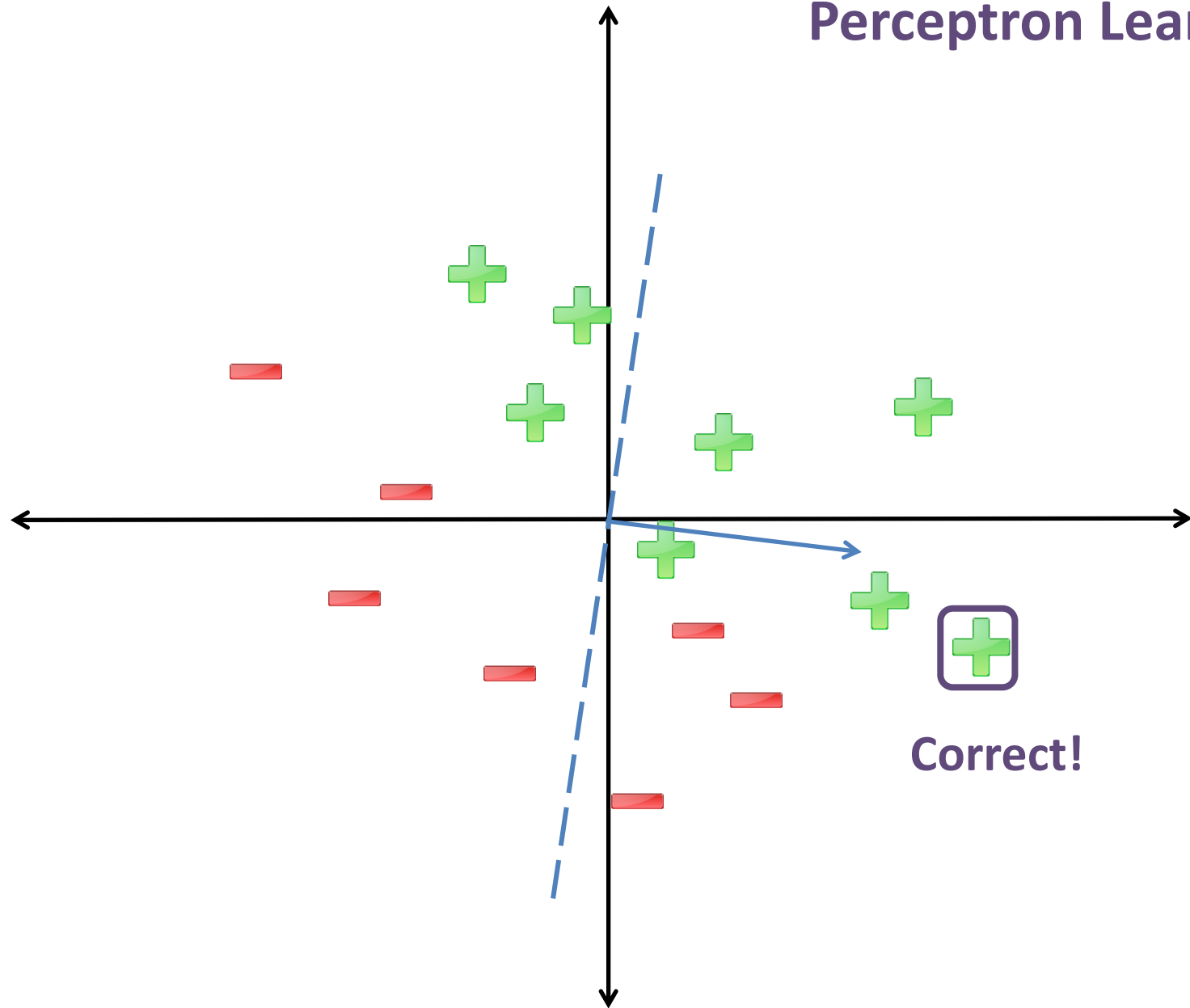
# Perceptron Learning



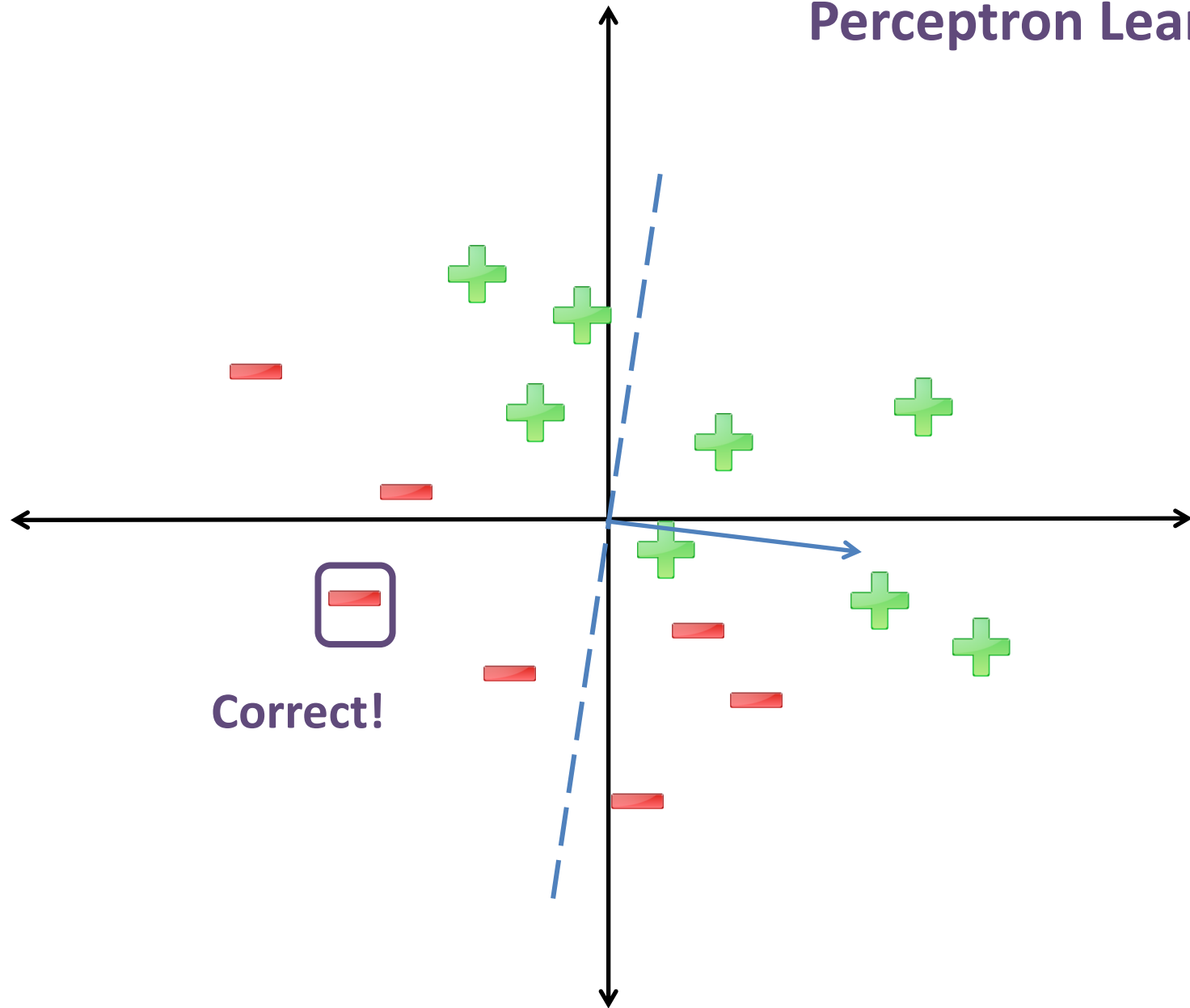
# Perceptron Learning



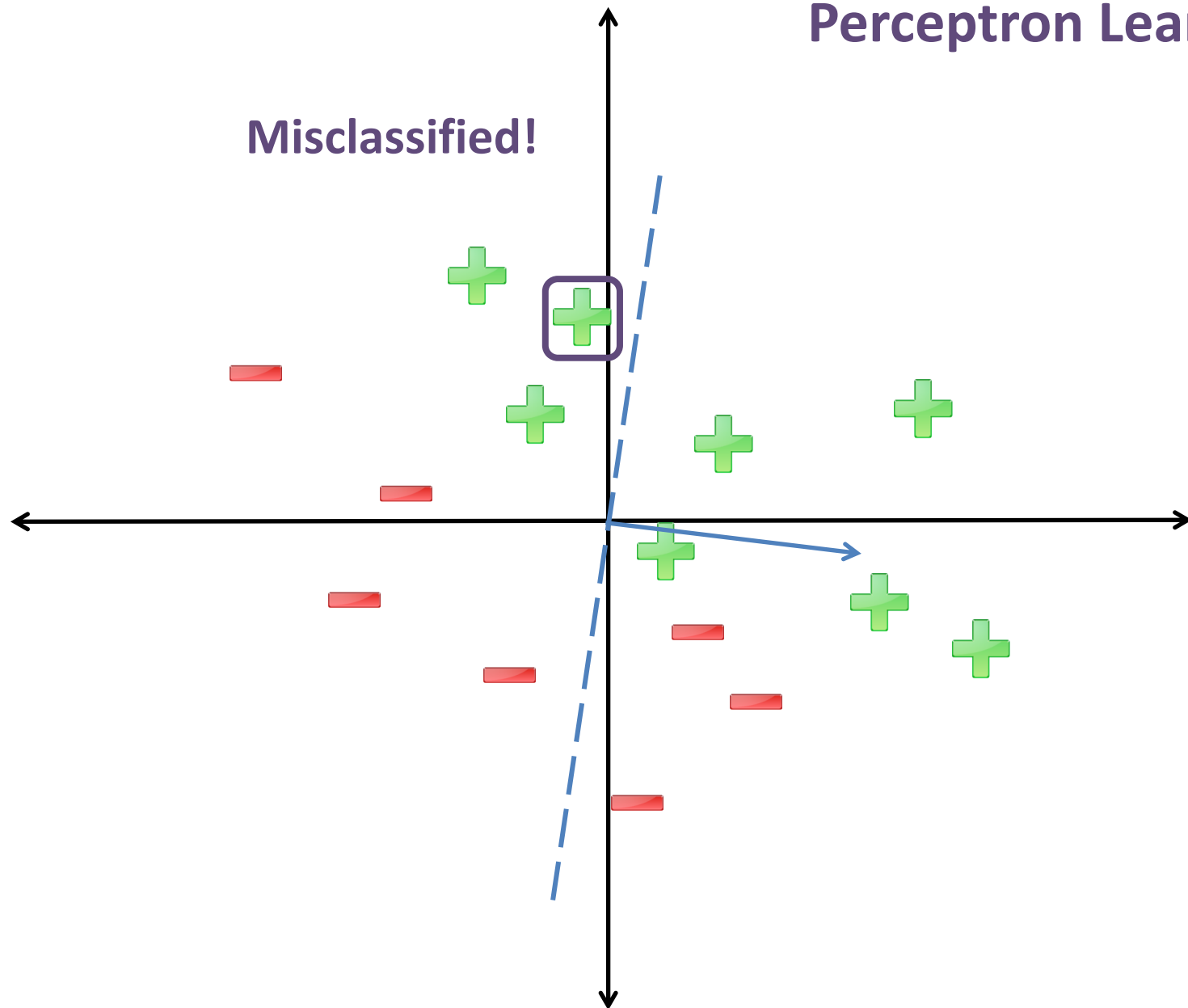
# Perceptron Learning



# Perceptron Learning



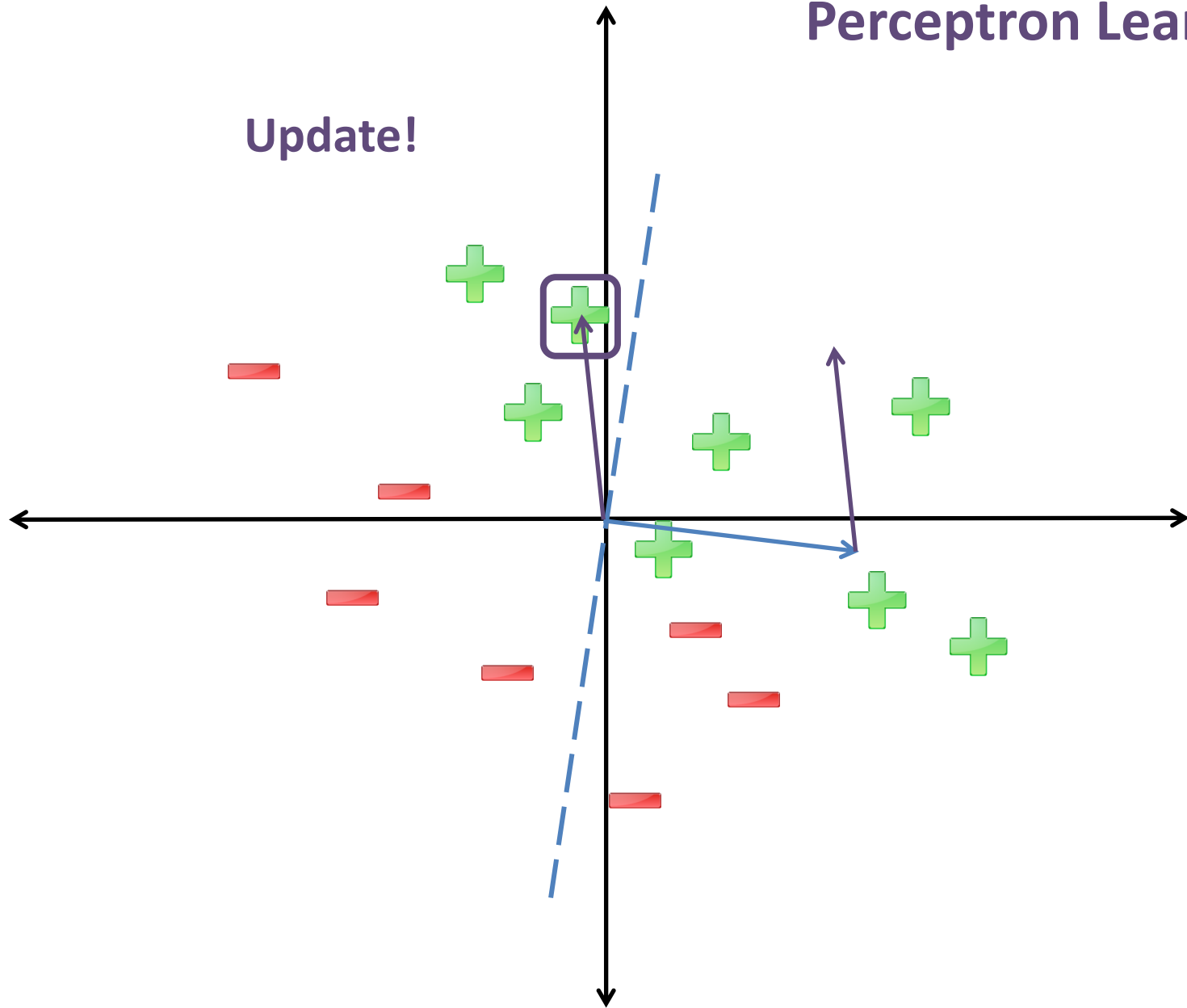
# Perceptron Learning



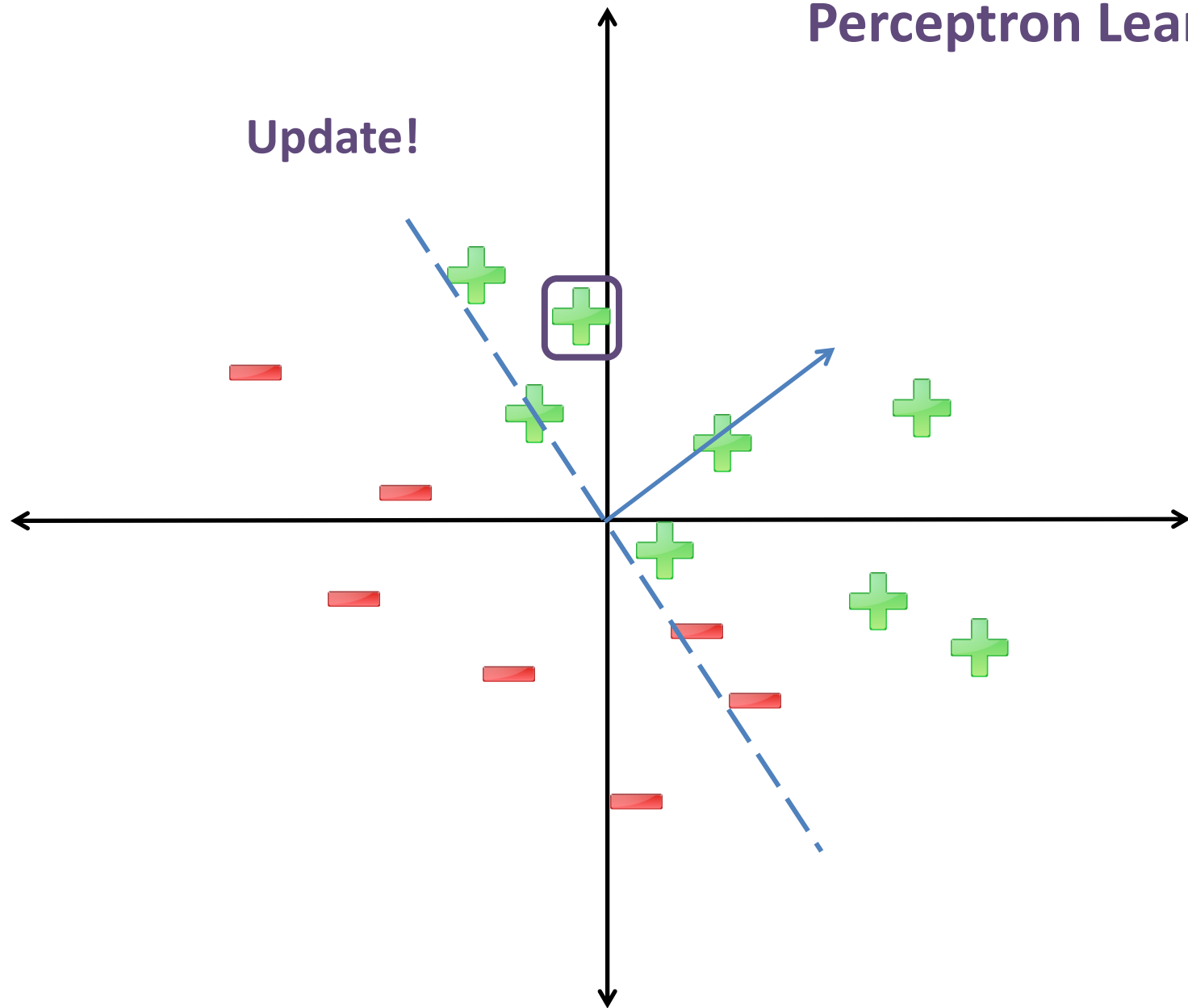


# Perceptron Learning

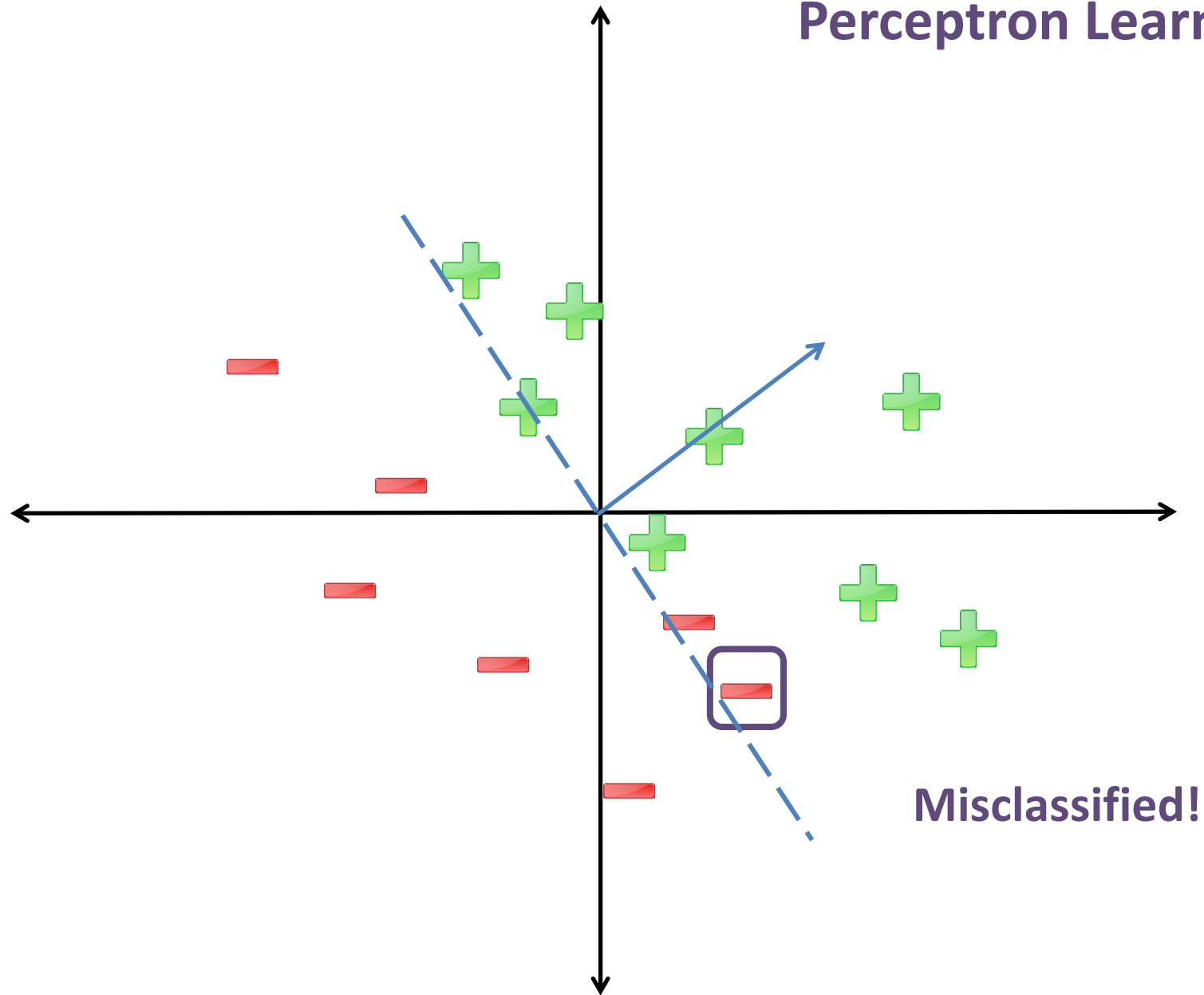
Update!



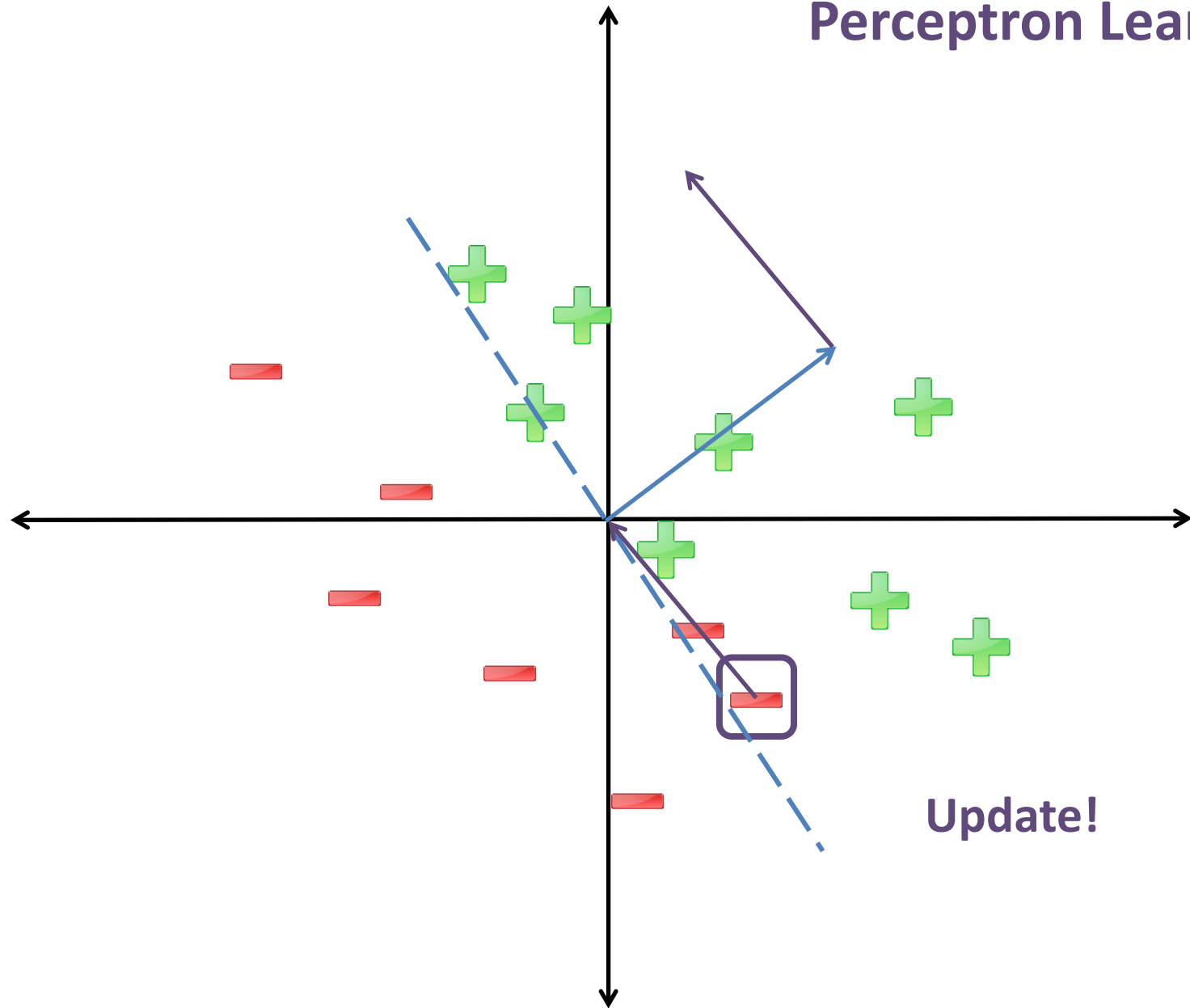
# Perceptron Learning



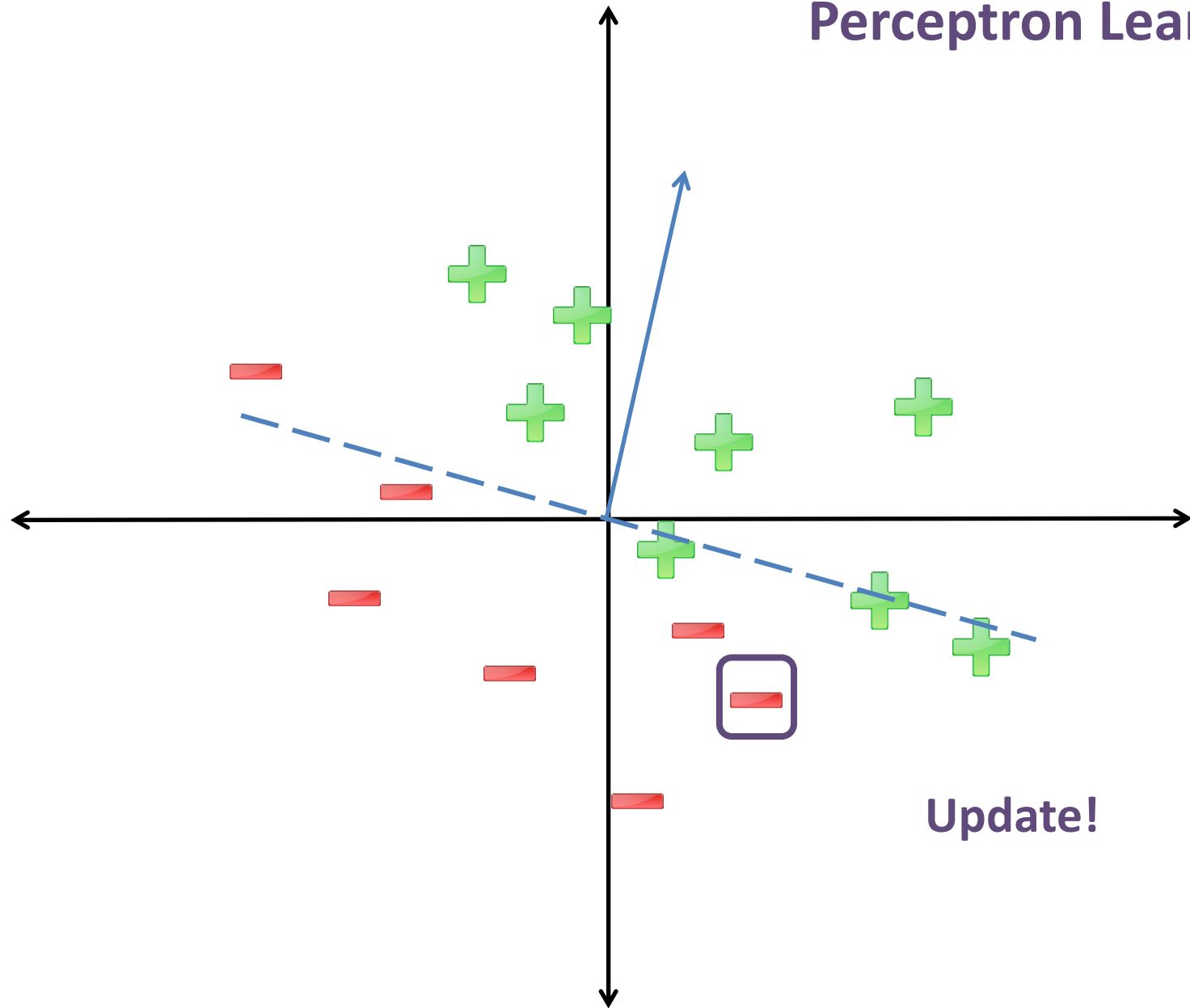
# Perceptron Learning



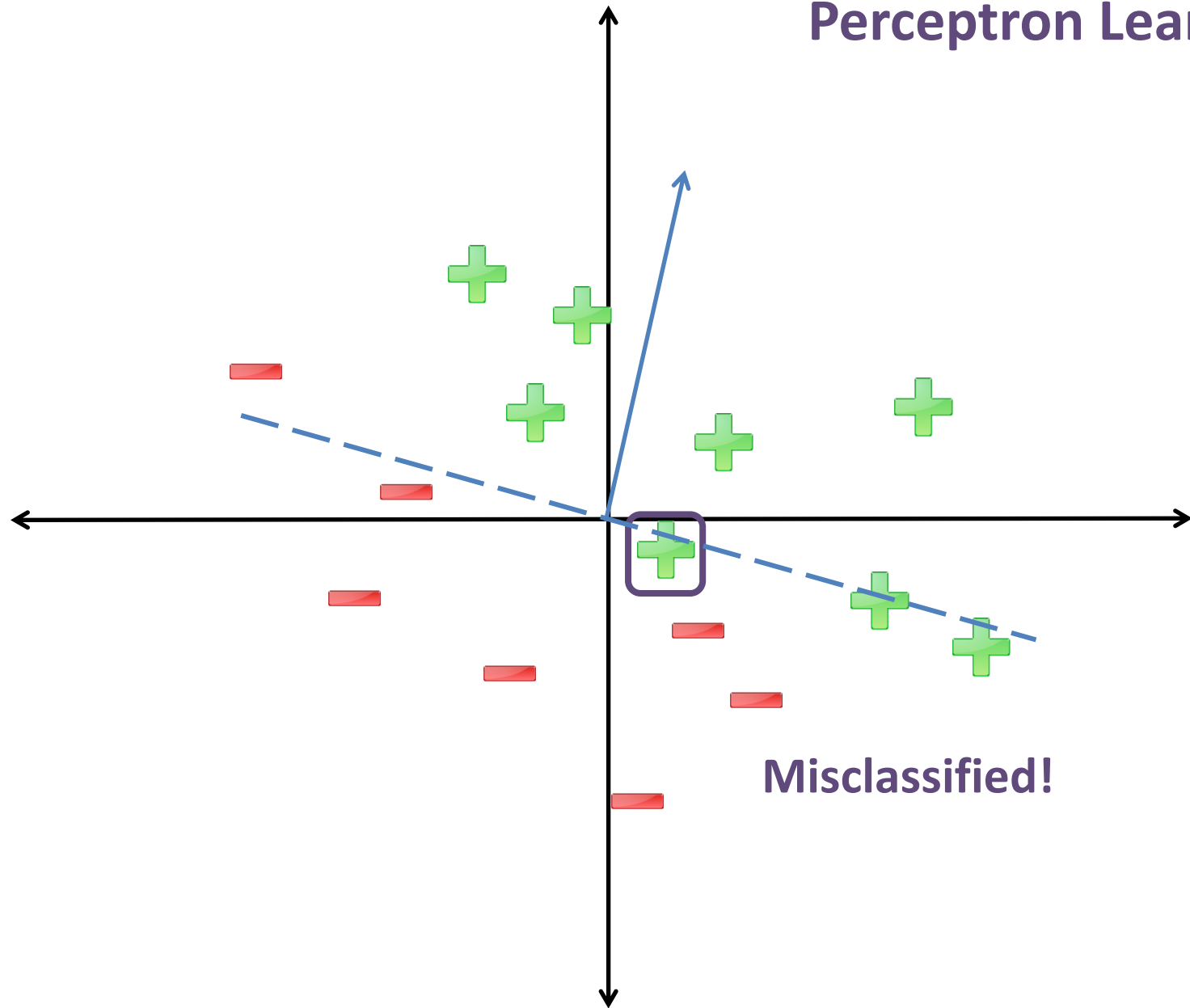
# Perceptron Learning



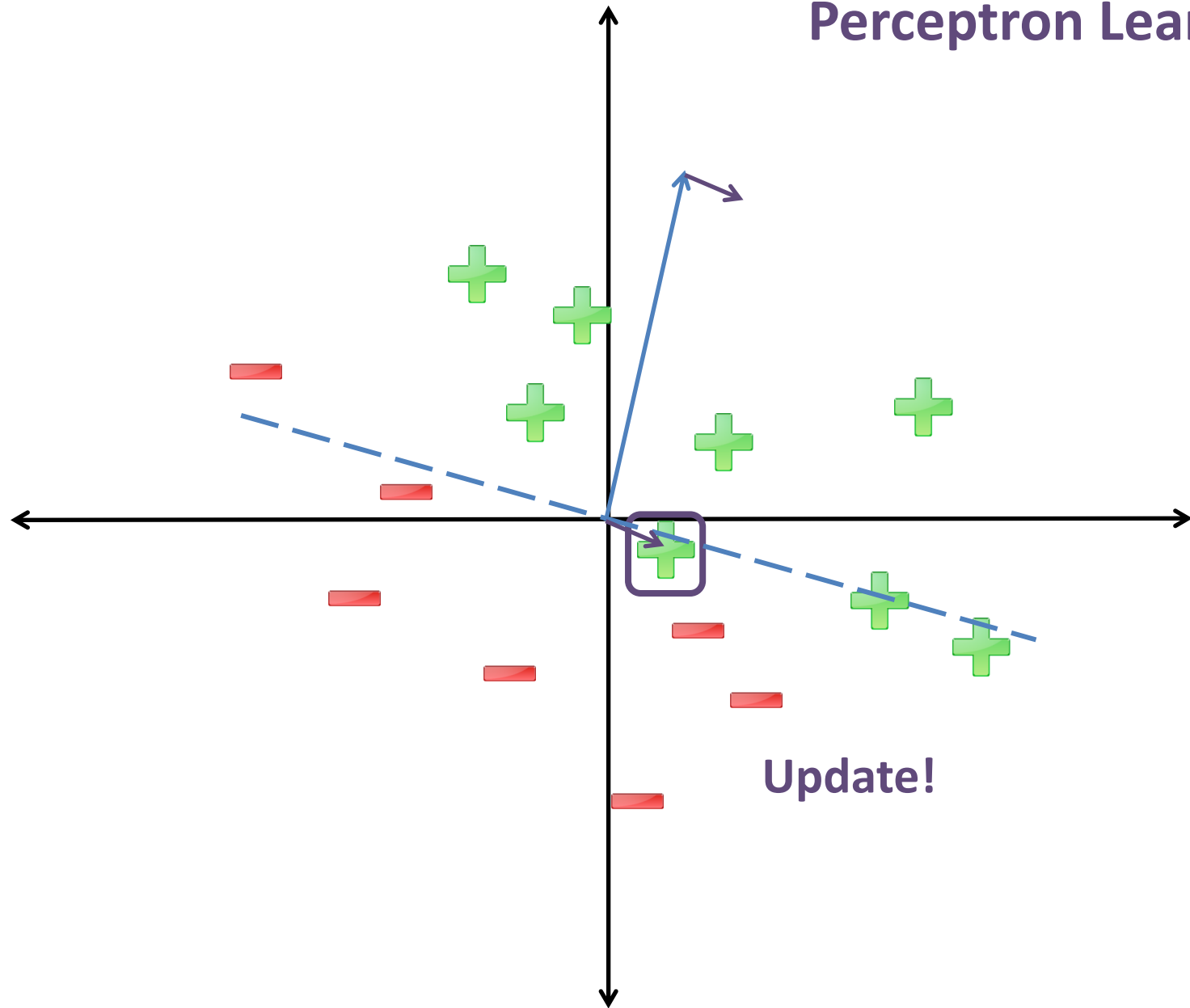
# Perceptron Learning



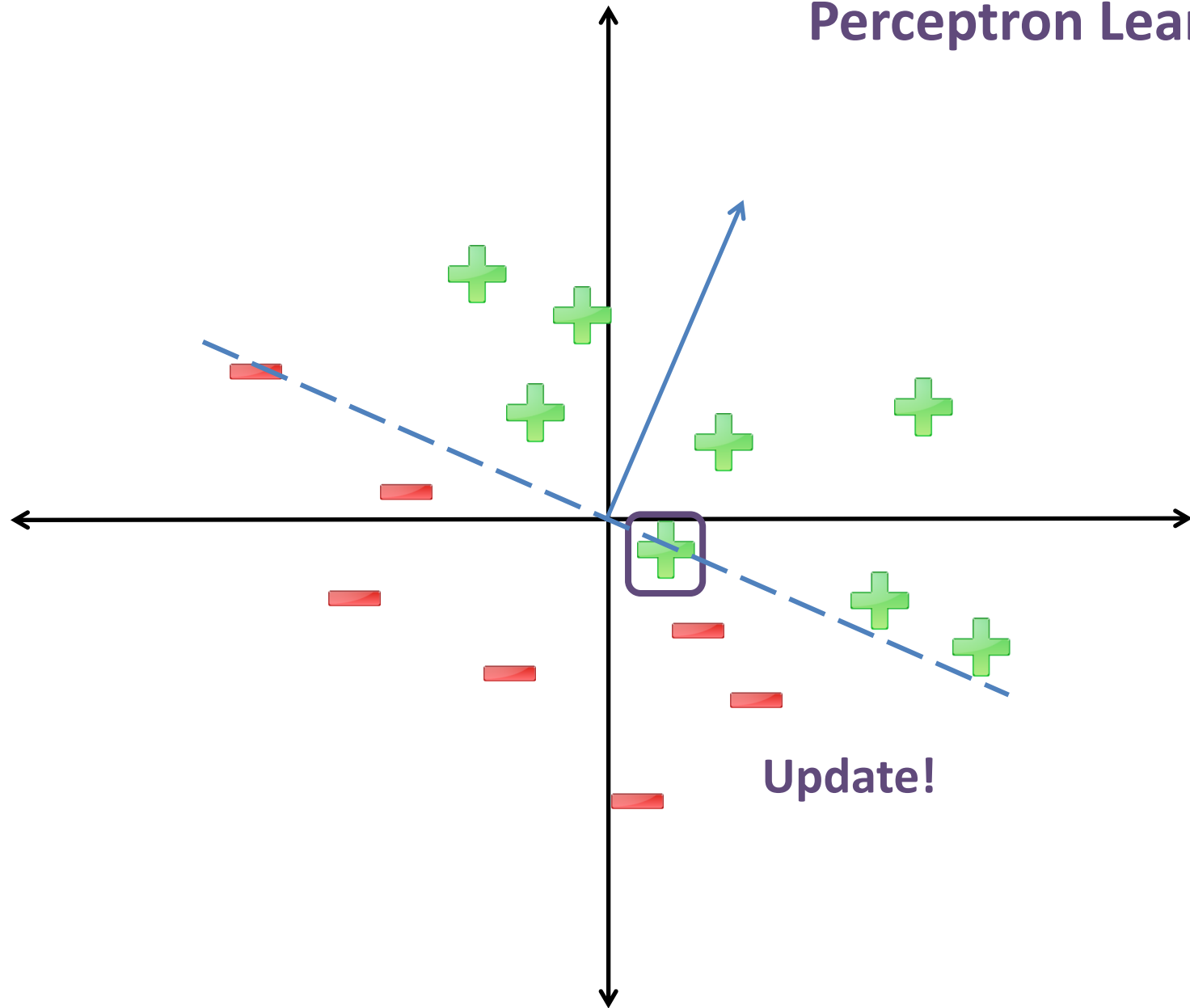
# Perceptron Learning



# Perceptron Learning

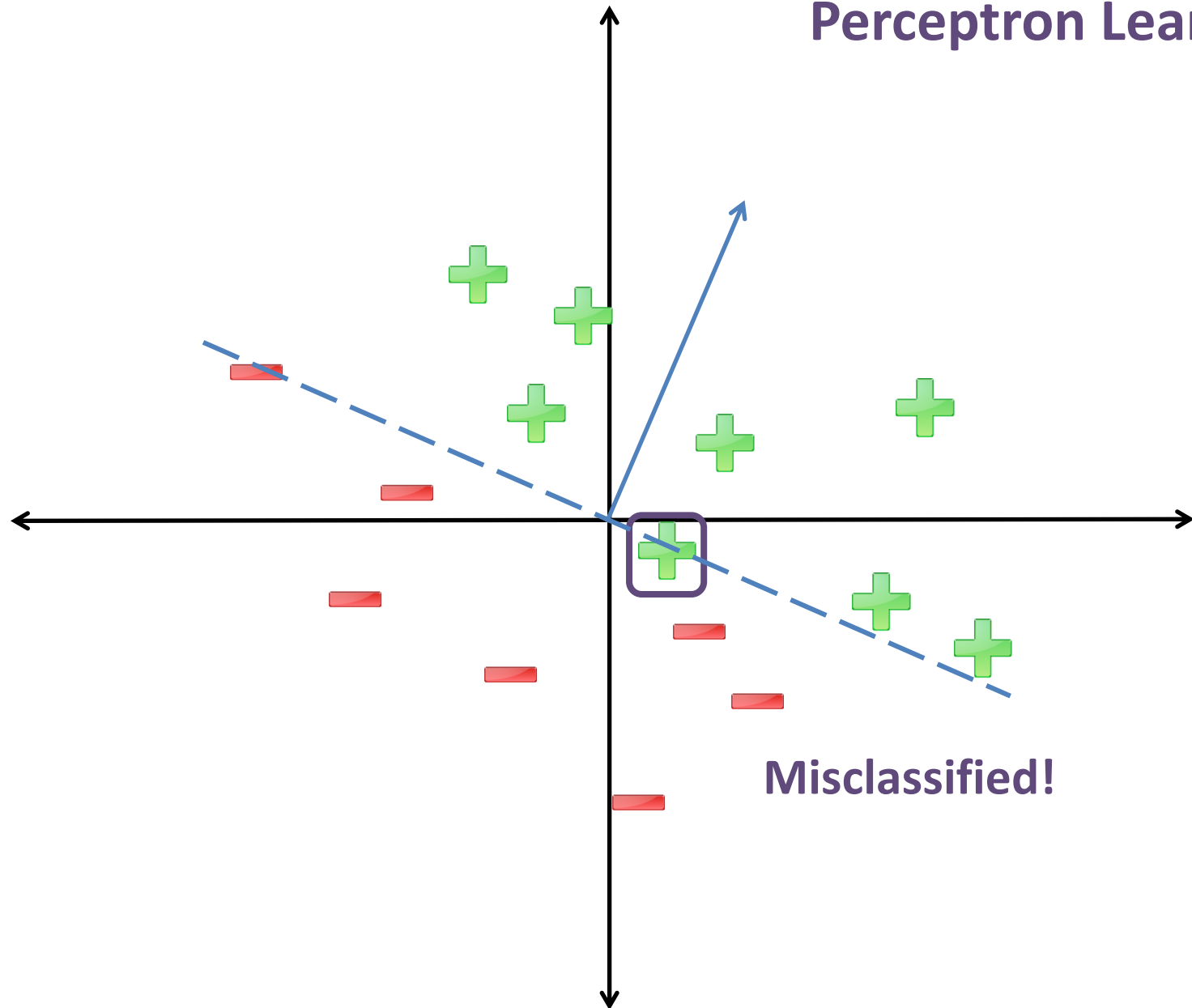


# Perceptron Learning

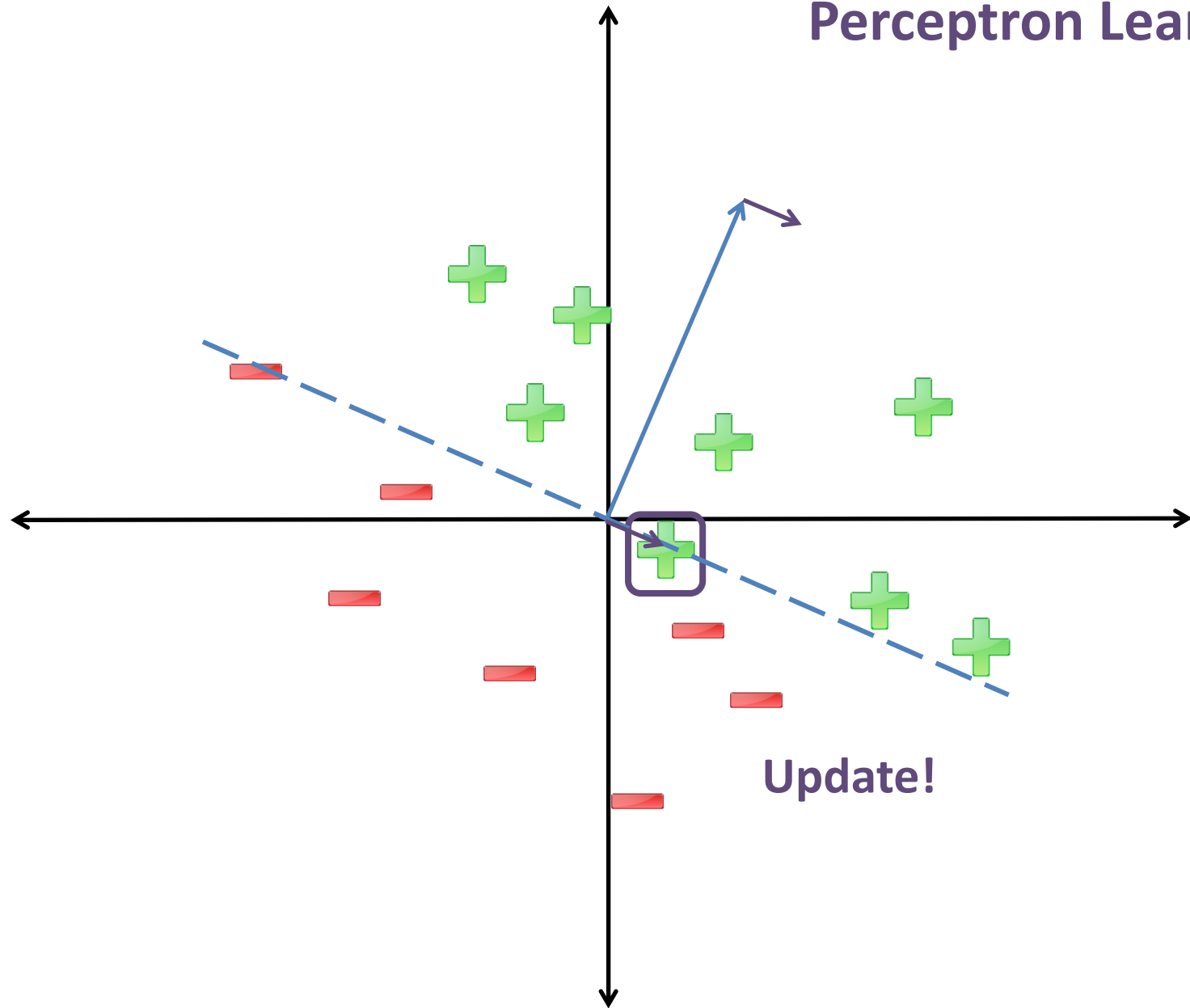




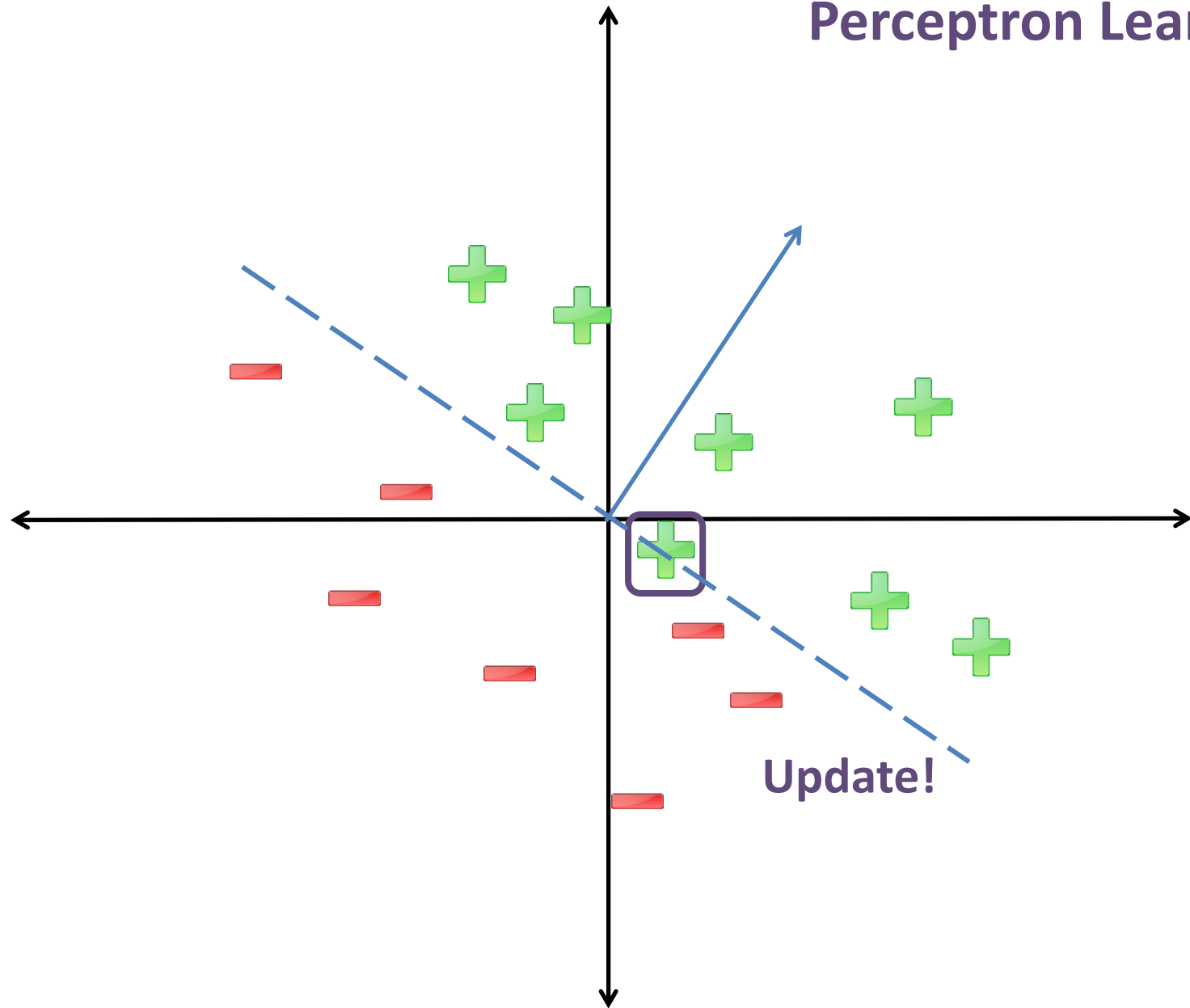
# Perceptron Learning



# Perceptron Learning

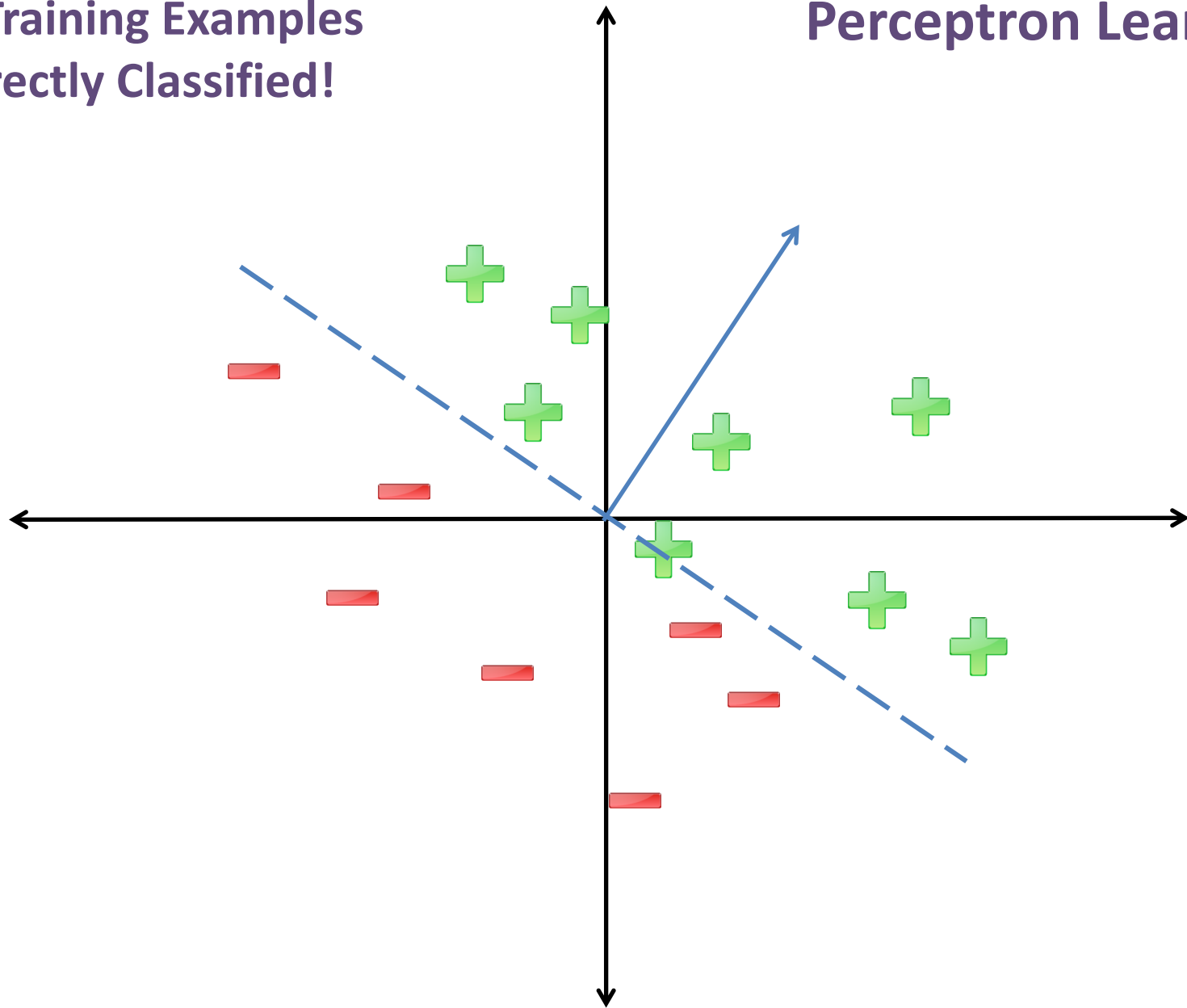


# Perceptron Learning



All Training Examples  
Correctly Classified!

## Perceptron Learning



# Recap: Perceptron Learning Algorithm

## (Linear Classification Model)

- $w^1 = 0, b^1 = 0$

$$f(x | w) = \text{sign}(w^T x - b)$$

- For  $t = 1 \dots$

- Receive example  $(x, y)$

- If  $f(x | w^t) = y$

- $[w^{t+1}, b^{t+1}] = [w^t, b^t]$

- Else

- $w^{t+1} = w^t + yx$

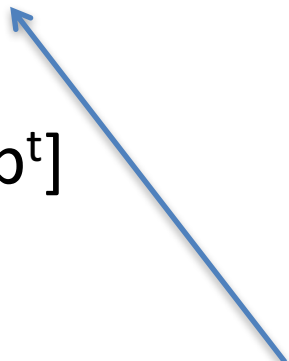
- $b^{t+1} = b^t - y$

**Training Set:**

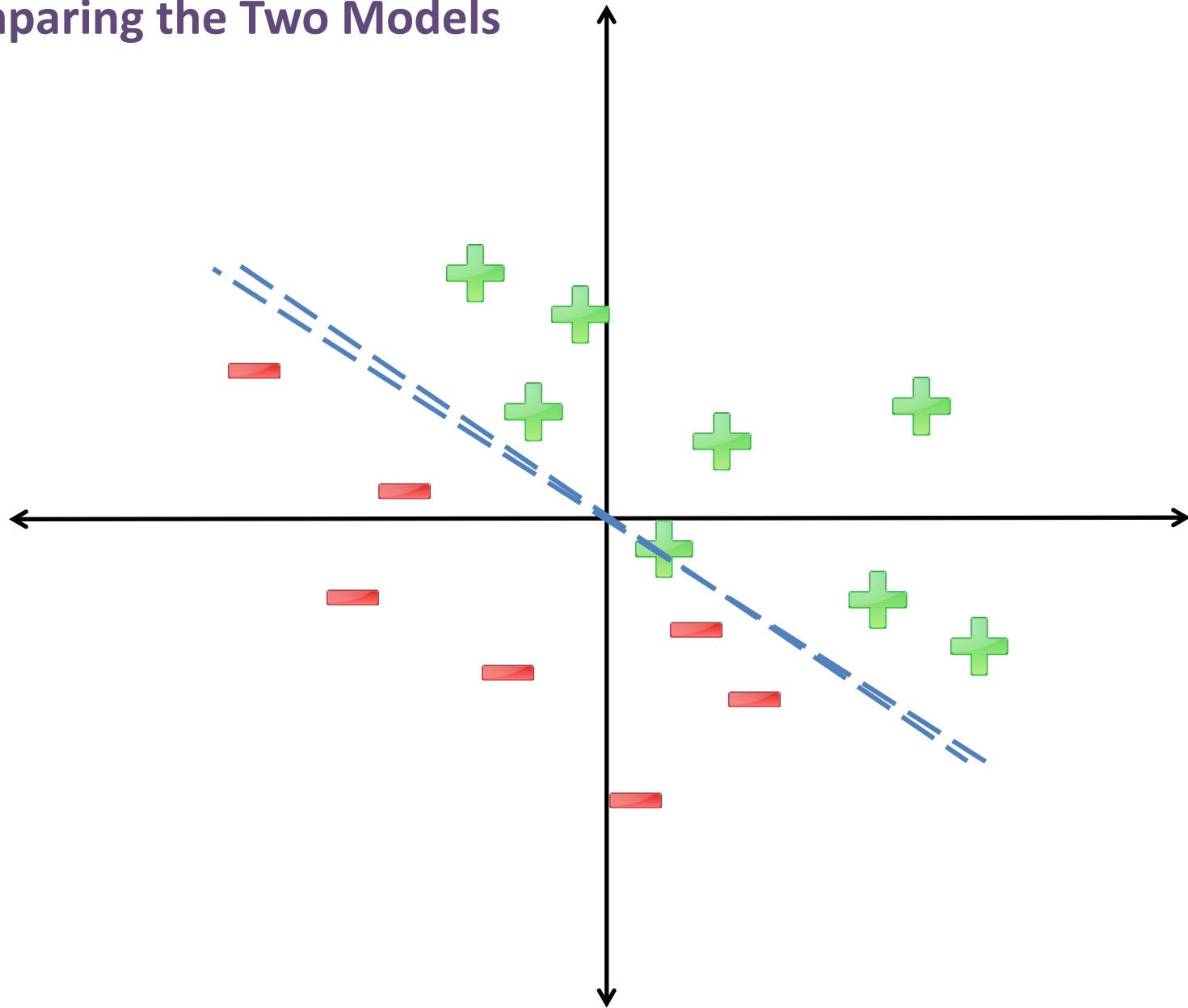
$$S = \{(x_i, y_i)\}_{i=1}^N$$

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

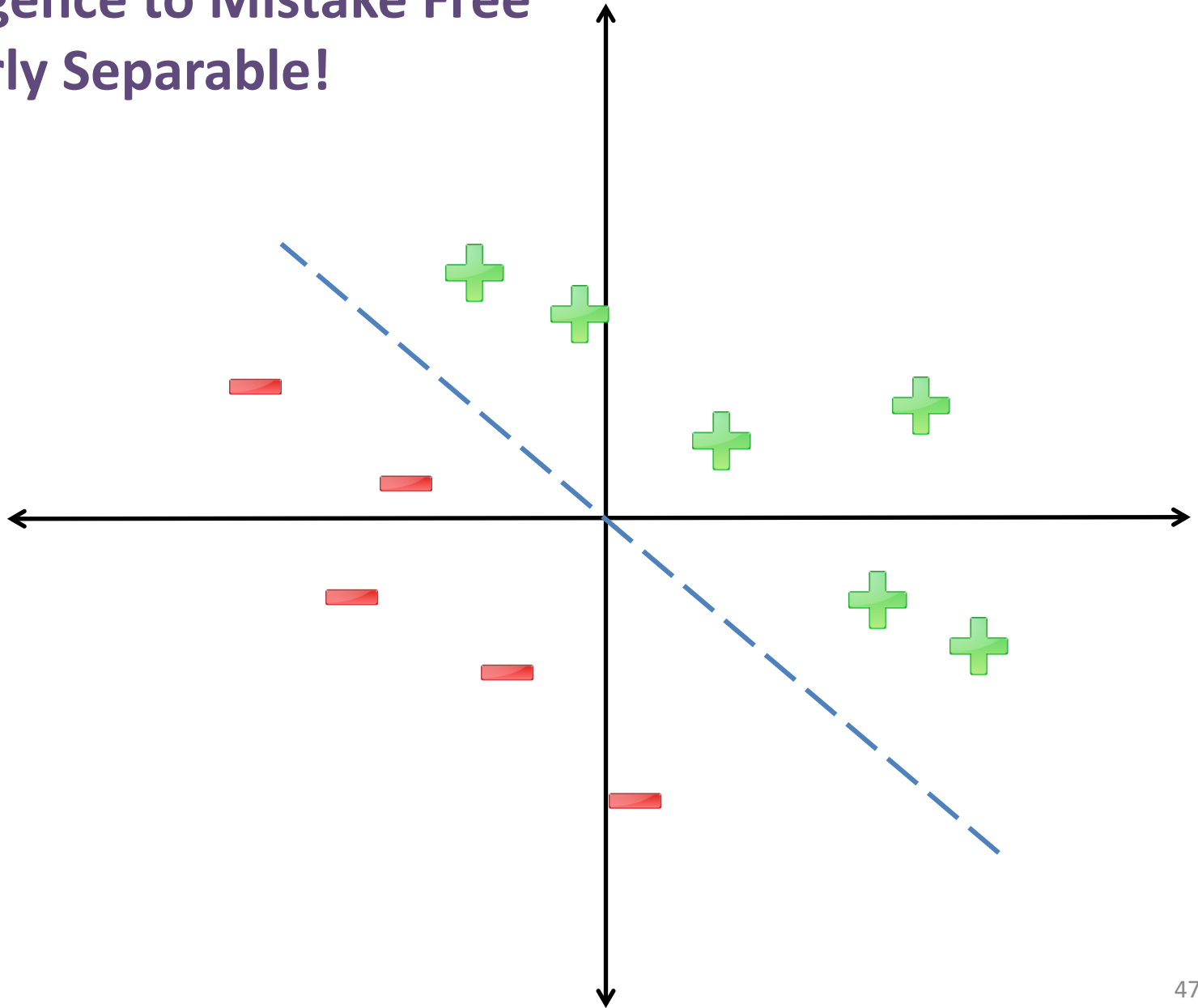
Go through training set  
in arbitrary order  
(e.g., randomly)



## Comparing the Two Models

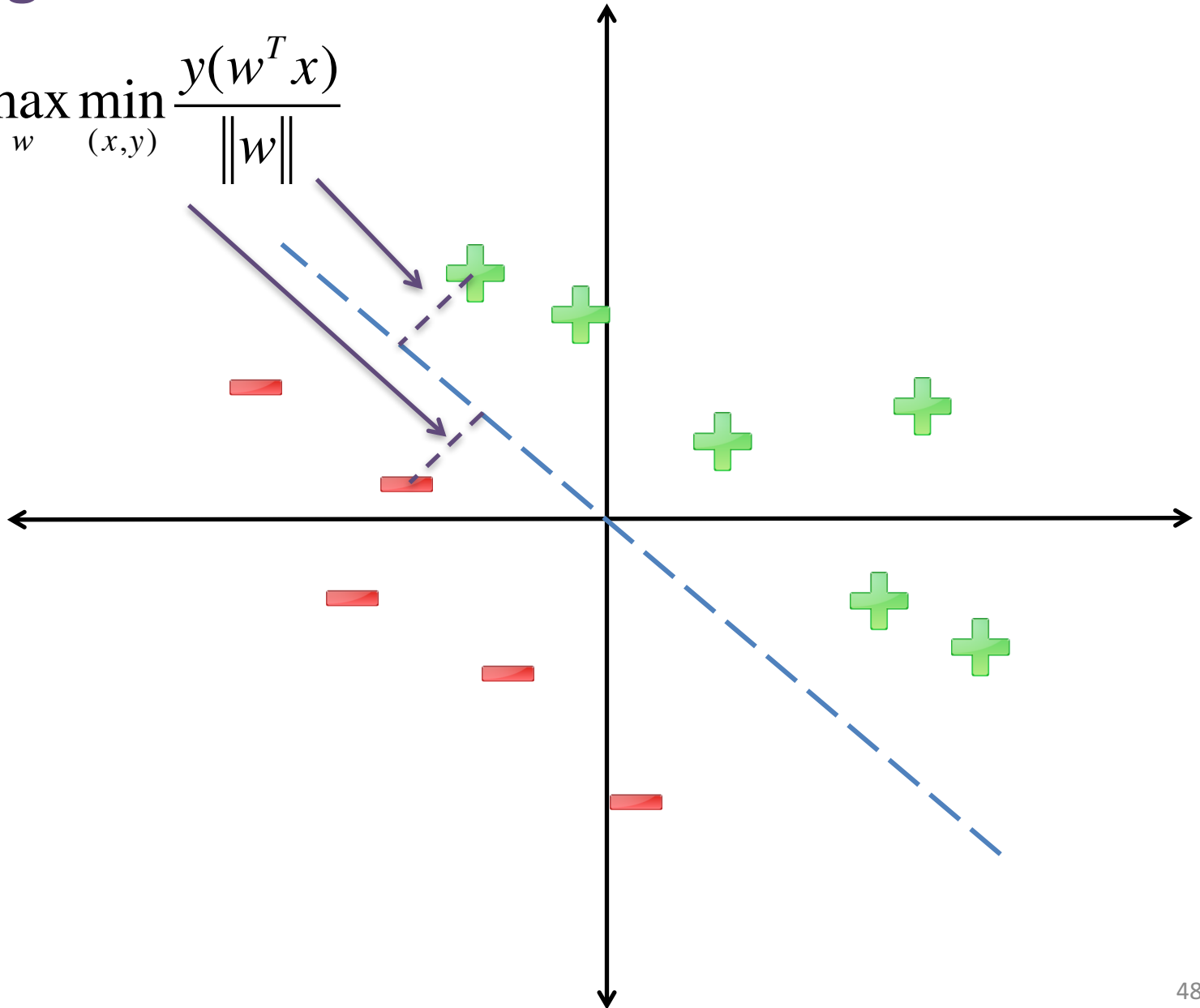


**Convergence to Mistake Free  
= Linearly Separable!**



# Margin

$$\gamma = \max_w \min_{(x,y)} \frac{y(w^T x)}{\|w\|}$$





# Linear Separability

- A classification problem is Linearly Separable:
  - Exists  $w$  with perfect classification accuracy
- Separable with Margin  $\gamma$ :

$$\gamma = \max_w \min_{(x,y)} \frac{y(w^T x)}{\|w\|}$$

- Linearly Separable:  $\gamma > 0$

# Perceptron Mistake Bound

Holds for any ordering  
of training examples!

“Radius” of Feature Space

$$R = \max_x \|x\|$$

#Mistakes Bounded By:  $\frac{R^2}{\gamma^2}$

Margin

\*\*If Linearly Separable

More Details: <http://www.cs.nyu.edu/~mohri/pub/pmb.pdf>

# In the Real World...

- Most problems are NOT linearly separable!
- May never converge...
- So what to do?
- **Use validation set!**

# Early Stopping via Validation

- Run Perceptron Learning on Training Set
- Evaluate current model on Validation Set
- Terminate when validation accuracy stops improving

[https://en.wikipedia.org/wiki/Early\\_stopping](https://en.wikipedia.org/wiki/Early_stopping)

# Online Learning vs Batch Learning

- Online Learning:
  - Receive a stream of data  $(x,y)$
  - Make incremental updates (typically)
  - Perceptron Learning is an instance of Online Learning
- Batch Learning
  - Given all the data up front
  - Can use online learning algorithms for batch learning
  - E.g., stream the data to the learning algorithm

# Recap: Perceptron

- One of the first machine learning algorithms
- **Benefits:**
  - Simple and fast
  - Clean analysis
- **Drawbacks:**
  - Might not converge to a very good model
  - What is the objective function?

# (Stochastic) Gradient Descent

# Back to Optimizing Objective Functions

- Training Data:  $S = \{(x_i, y_i)\}_{i=1}^N$   $x \in \mathbb{R}^D$   
 $y \in \{-1, +1\}$
- Model Class:  $f(x | w, b) = w^T x - b$  **Linear Models**
- Loss Function:  $L(a, b) = (a - b)^2$  **Squared Loss**
- Learning Objective:  $\operatorname{argmin}_{w, b} \sum_{i=1}^N L(y_i, f(x_i | w, b))$   
**Optimization Problem**



# Back to Optimizing Objective Functions

$$\operatorname{argmin}_{w,b} L(w,b) \equiv \sum_{i=1}^N L(y_i, f(x_i | w, b))$$

- Typically, requires optimization algorithm.
- Simplest: **Gradient Descent**
- This Lecture: stick with squared loss
  - Talk about various loss functions next lecture

# Gradient Review for Squared Loss

$$\partial_w L(w, b) = \partial_w \sum_{i=1}^N L(y_i, f(x_i | w, b))$$

$$= \sum_{i=1}^N \partial_w L(y_i, f(x_i | w, b))$$

Linearity of Differentiation

$$= \sum_{i=1}^N -2(y_i - f(x_i | w, b)) \partial_w f(x_i | w, b)$$

$$L(a, b) = (a - b)^2$$

Chain Rule

$$= \sum_{i=1}^N -2(y_i - f(x_i | w, b)) x_i$$

$$f(x | w, b) = w^T x - b$$

# Gradient Descent

- Initialize:  $w^1 = 0, b^1 = 0$
- For  $t = 1 \dots$

$$w^{t+1} = w^t - \eta^{t+1} \partial_w L(w^t, b^t)$$

$$b^{t+1} = b^t - \eta^{t+1} \partial_b L(w^t, b^t)$$

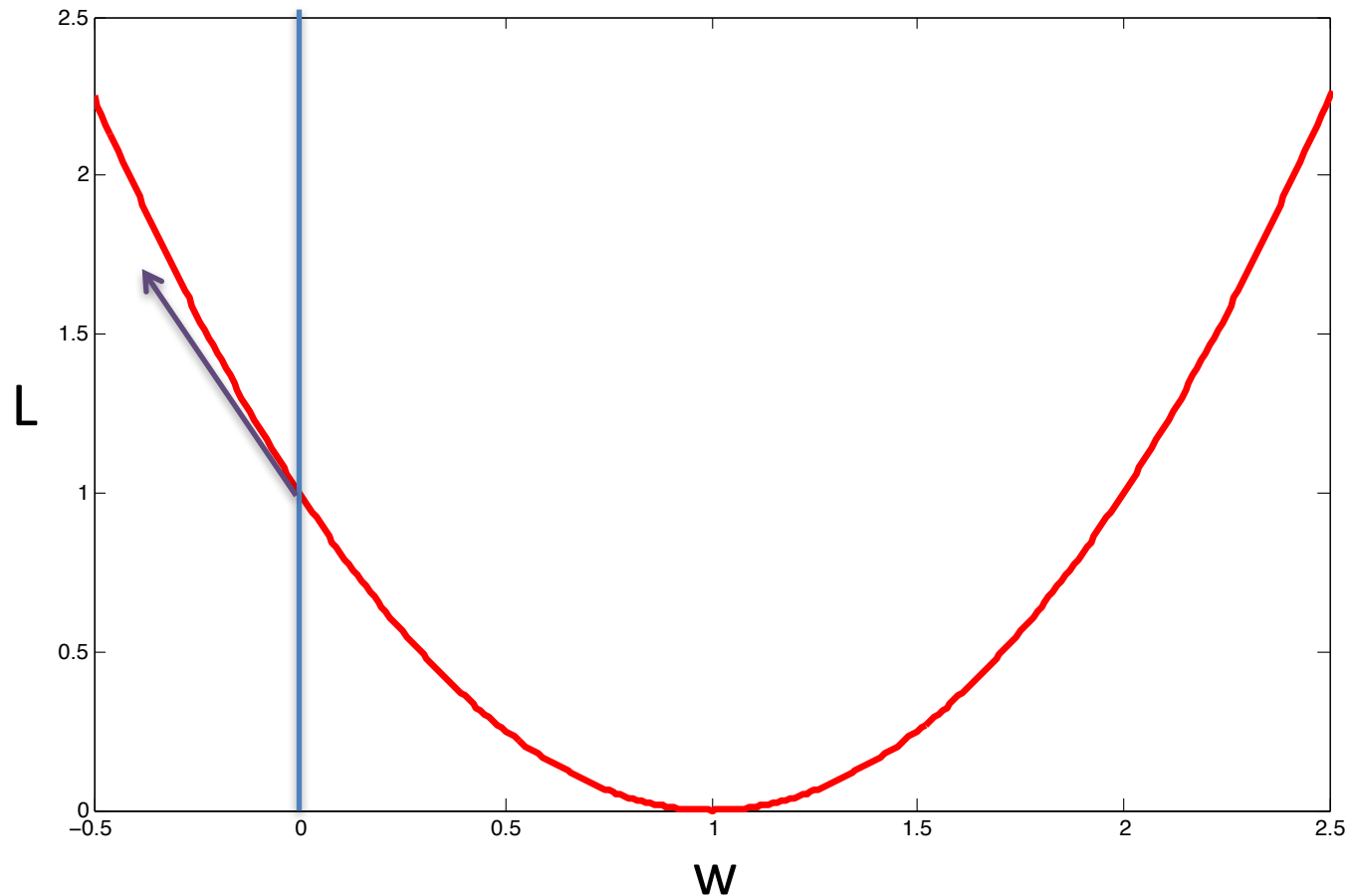


“Step Size”

# How to Choose Step Size?

$$\eta = 1$$

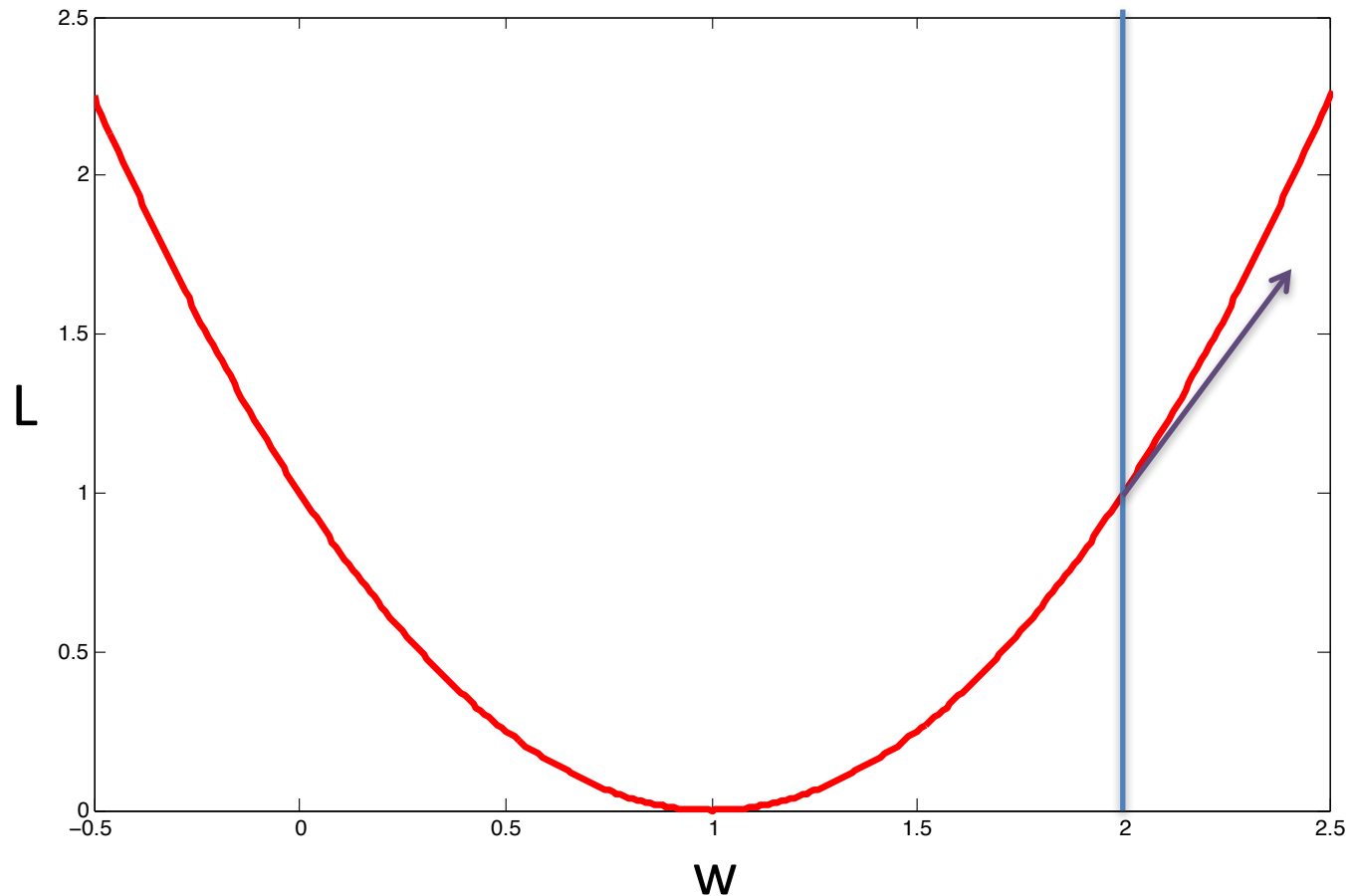
$$\partial_w L(w) = -2(1 - w)$$



# How to Choose Step Size?

$$\eta = 1$$

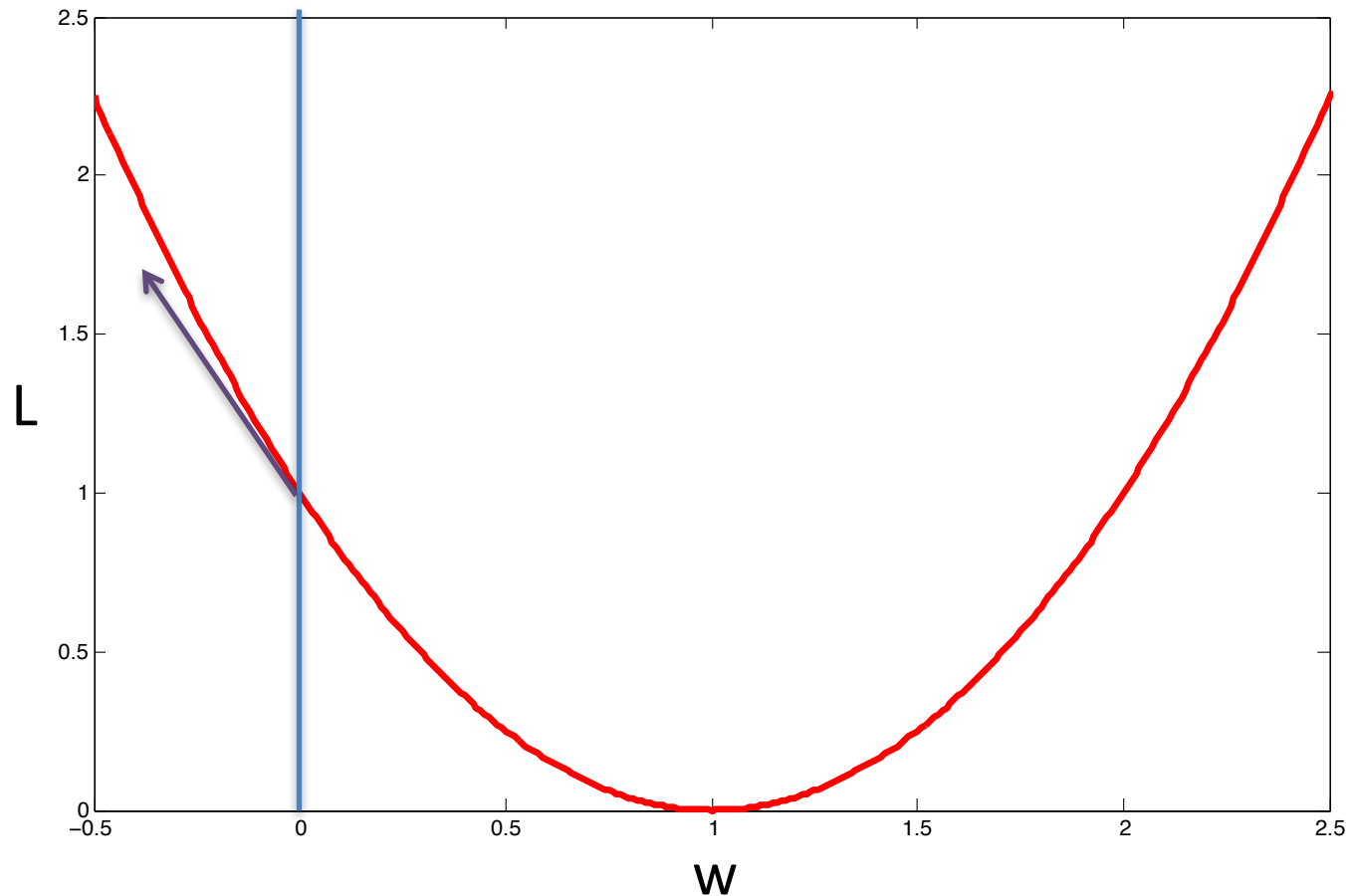
$$\partial_w L(w) = -2(1 - w)$$



# How to Choose Step Size?

$$\eta = 1$$

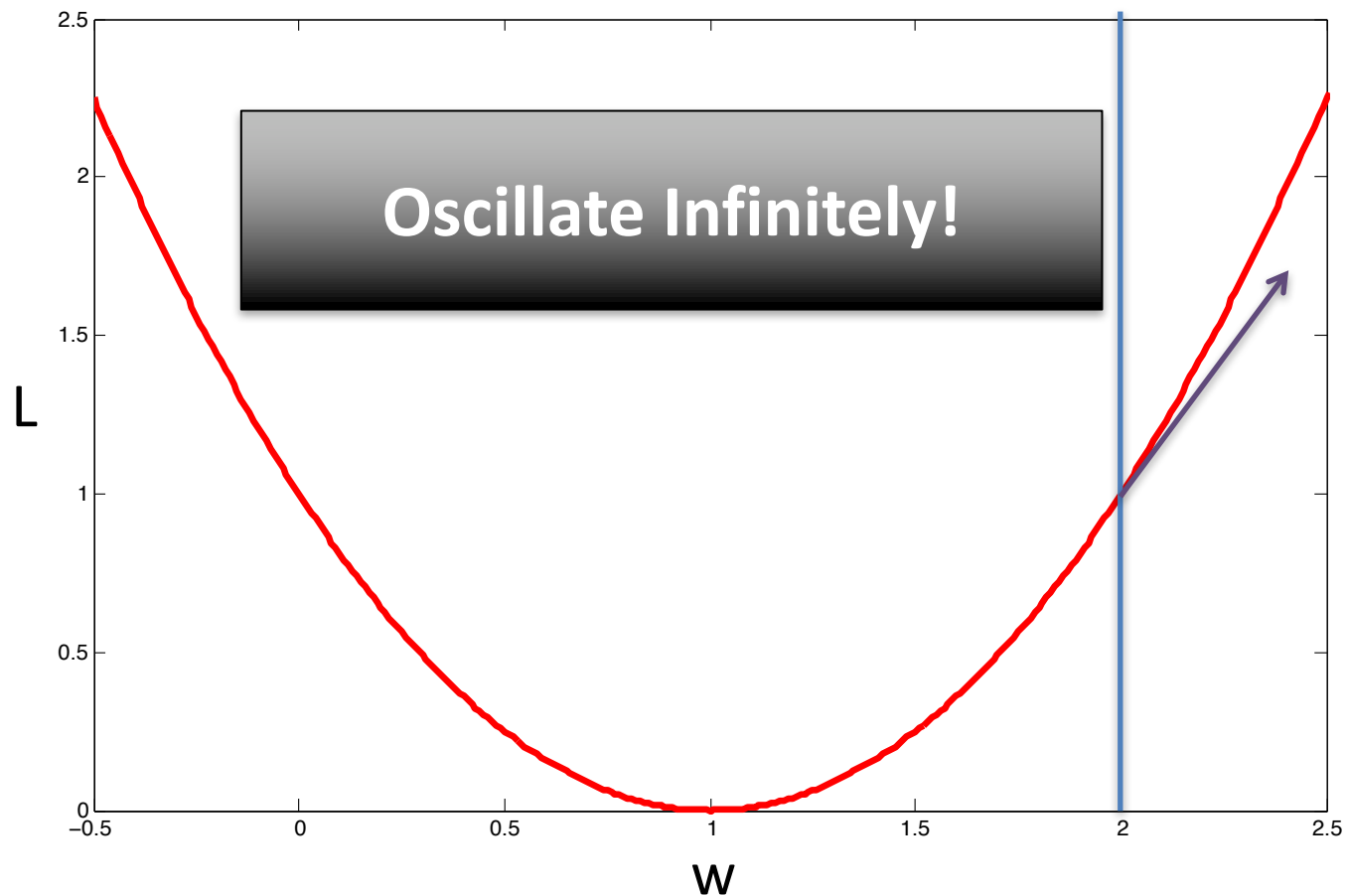
$$\partial_w L(w) = -2(1 - w)$$



# How to Choose Step Size?

$$\eta = 1$$

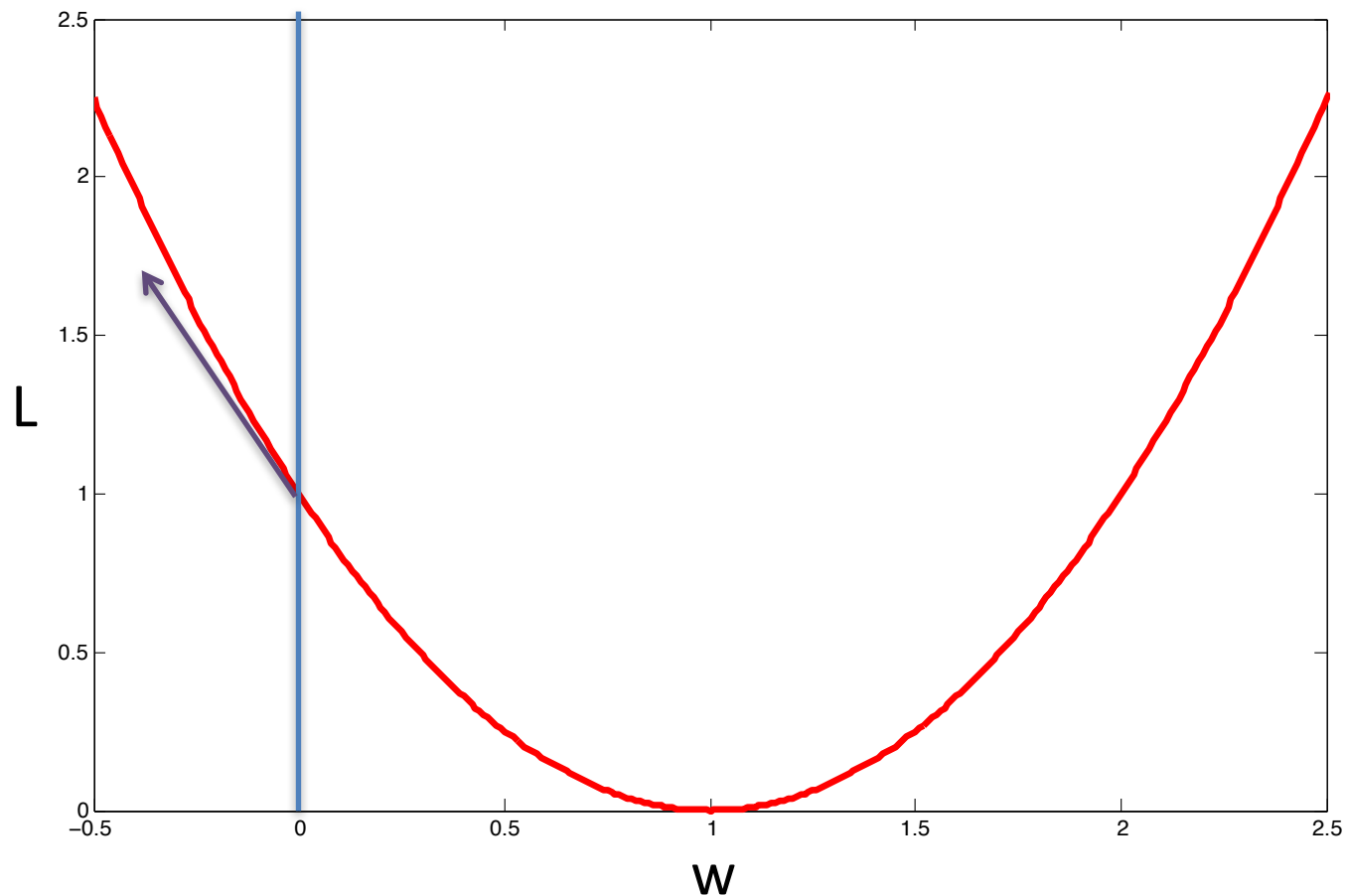
$$\partial_w L(w) = -2(1 - w)$$



# How to Choose Step Size?

$$\eta = 0.0001$$

$$\partial_w L(w) = -2(1 - w)$$

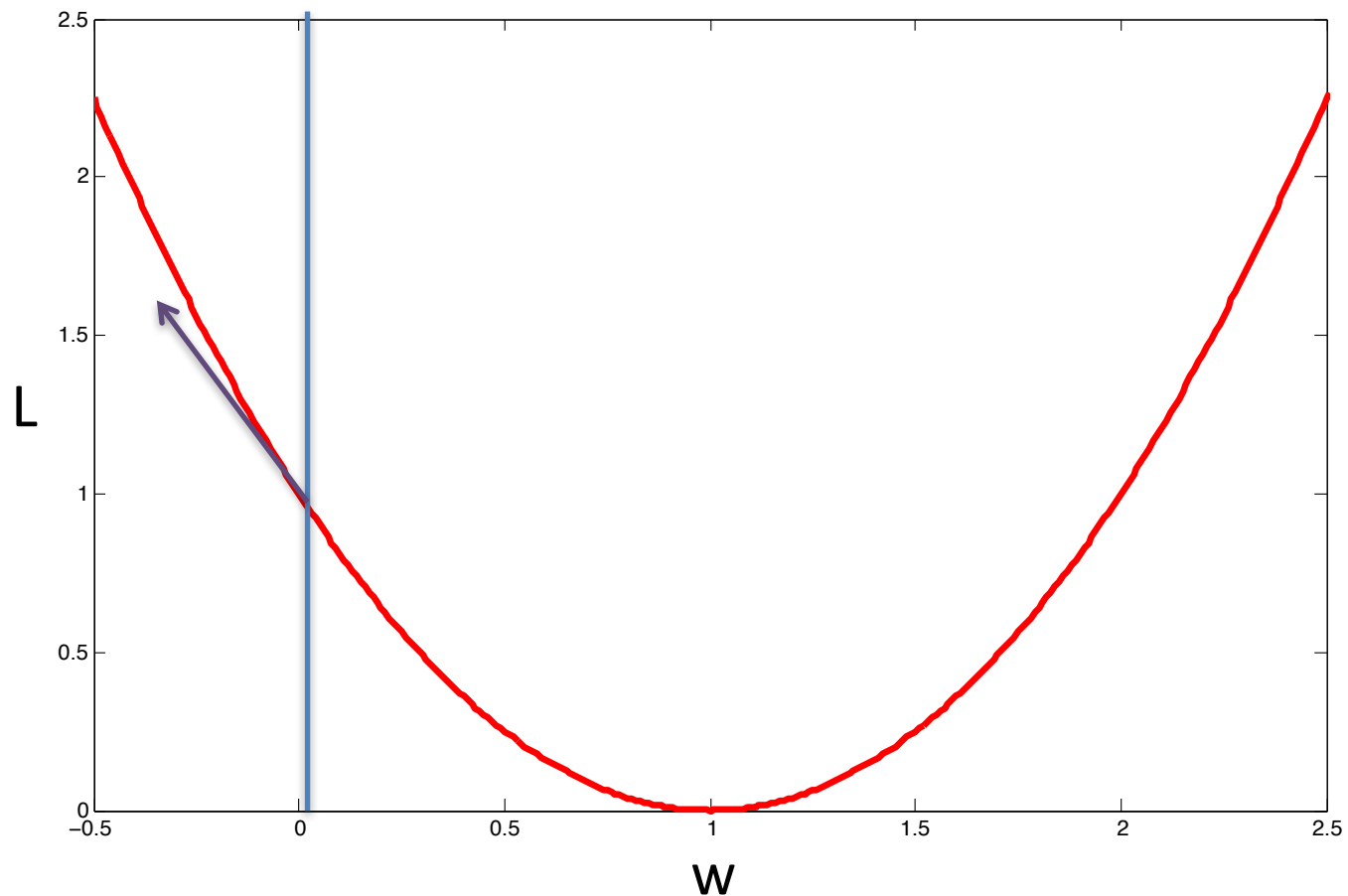




# How to Choose Step Size?

$$\eta = 0.0001$$

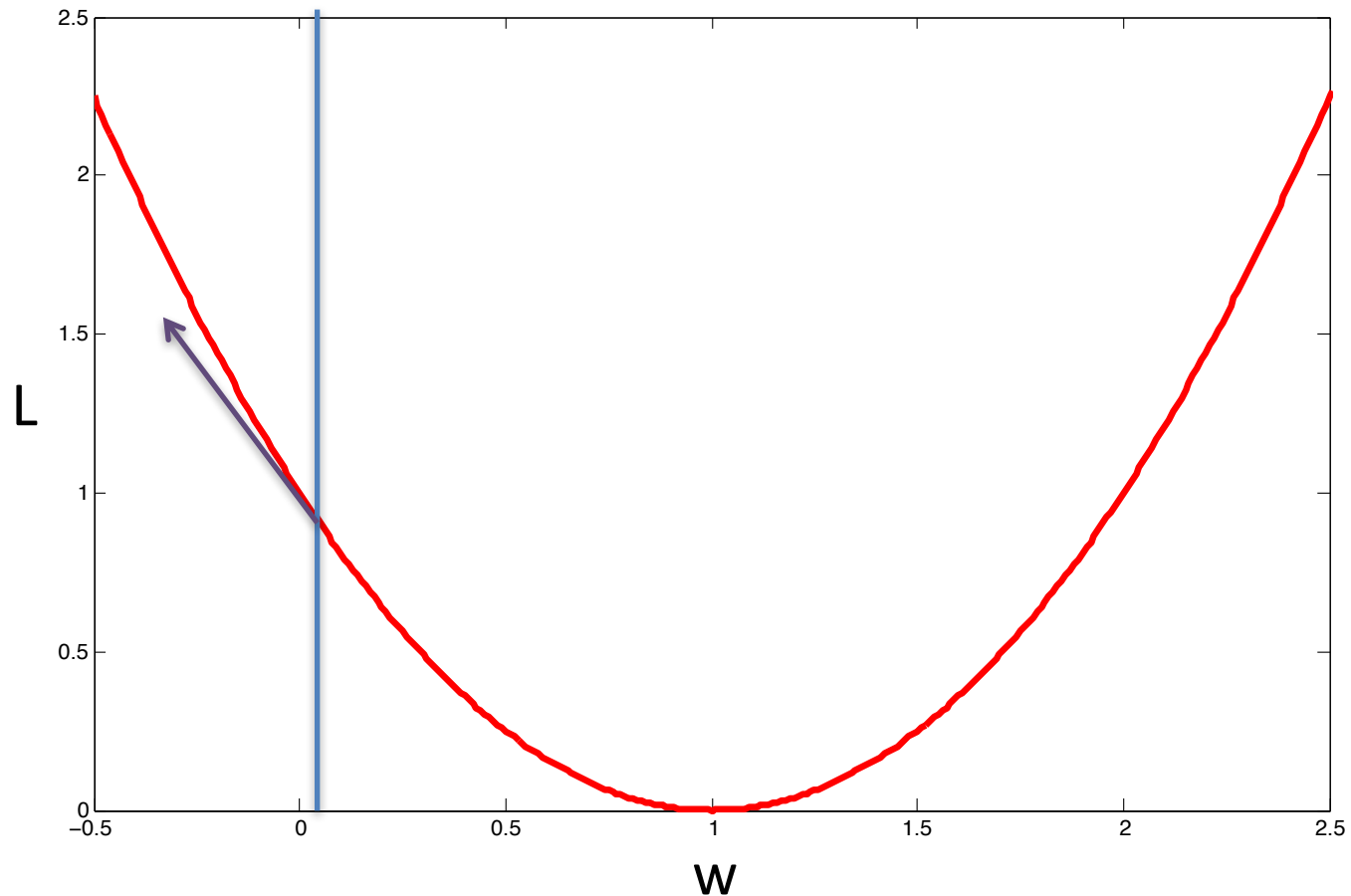
$$\partial_w L(w) = -2(1 - w)$$



# How to Choose Step Size?

$$\eta = 0.0001$$

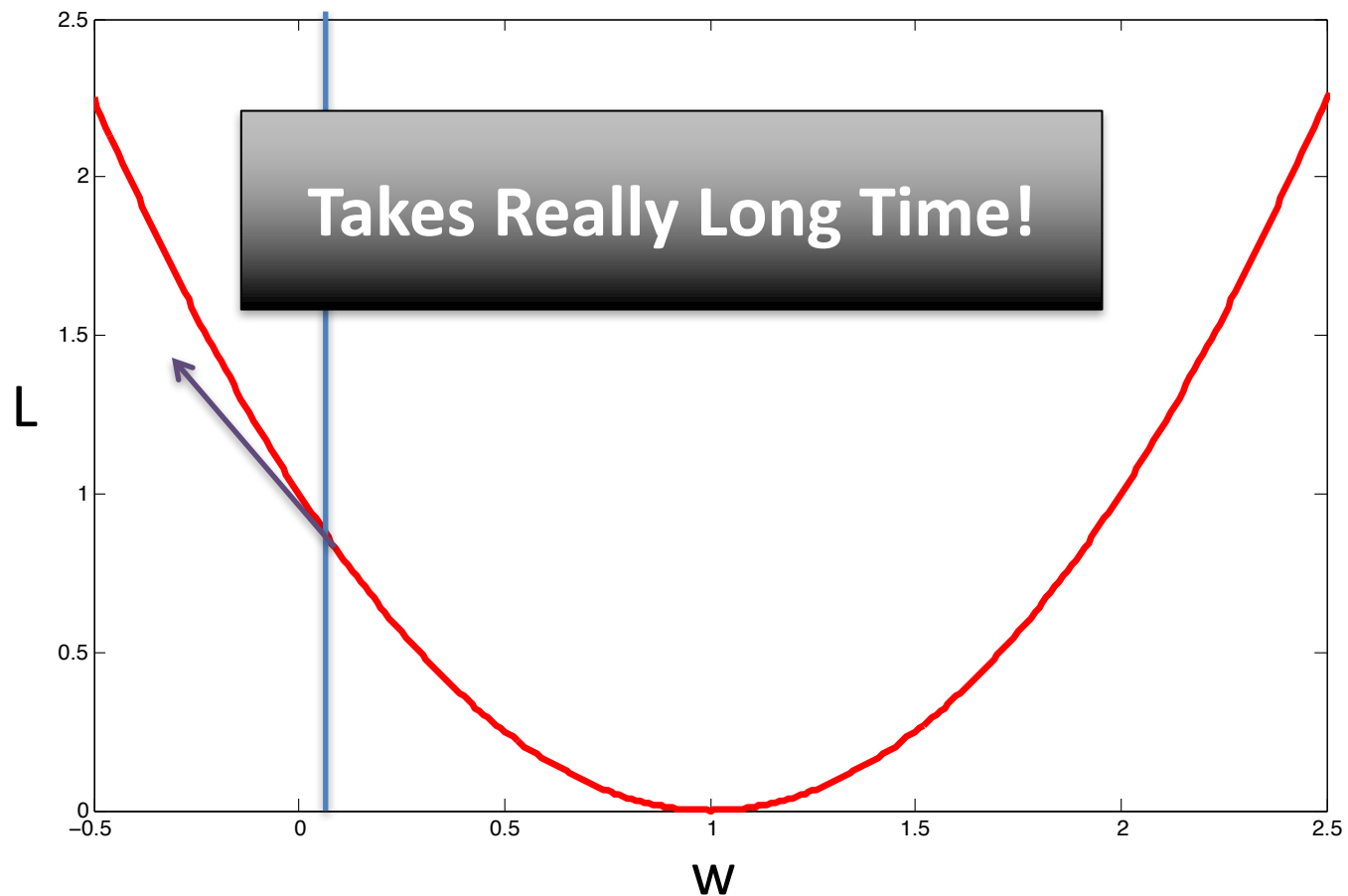
$$\partial_w L(w) = -2(1 - w)$$



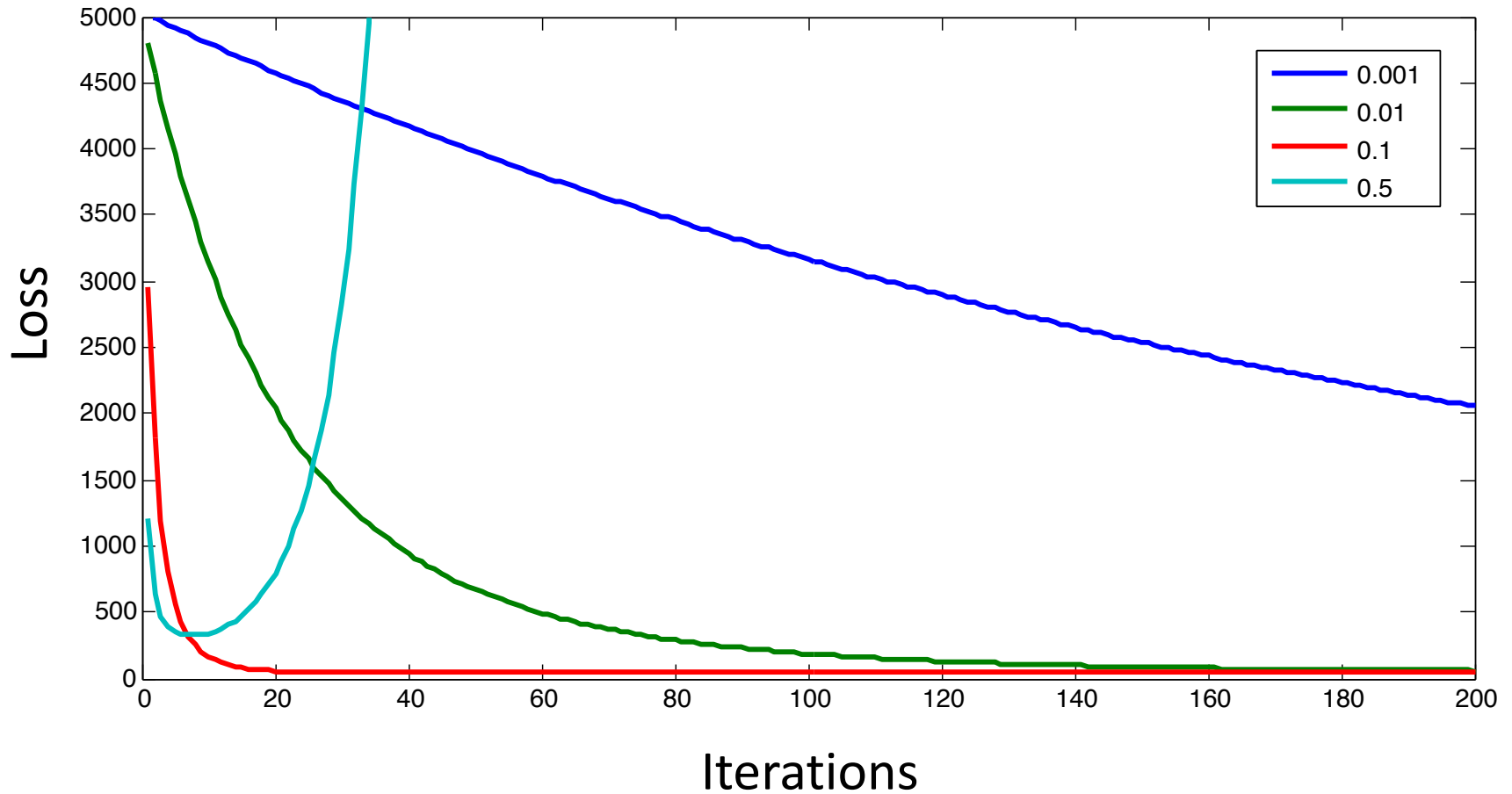
# How to Choose Step Size?

$$\eta = 0.0001$$

$$\partial_w L(w) = -2(1 - w)$$

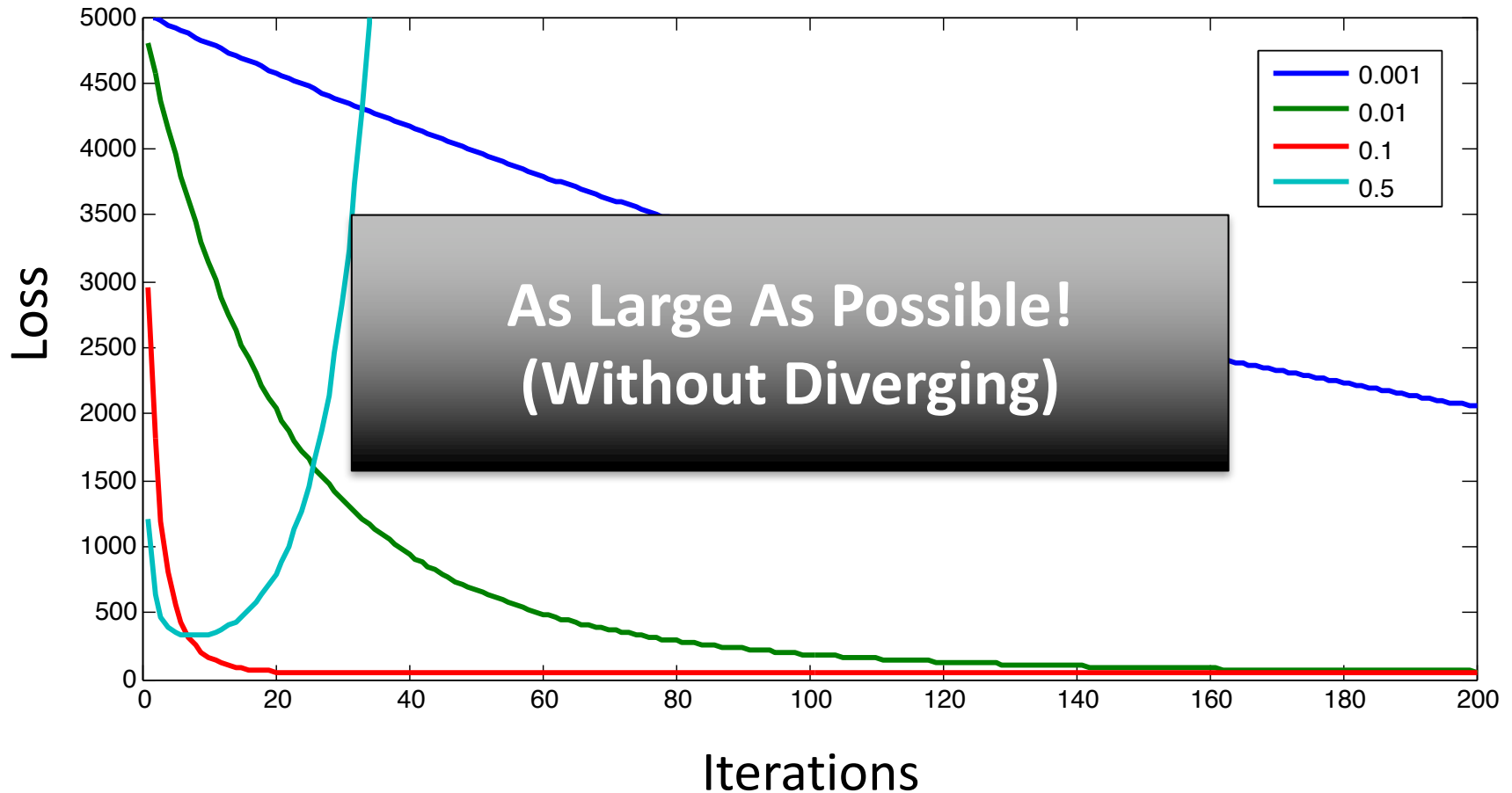


# How to Choose Step Size?



Note that the absolute scale is not meaningful  
Focus on the relative magnitude differences

# How to Choose Step Size?



Note that the absolute scale is not meaningful  
Focus on the relative magnitude differences

# Being Scale Invariant

- Consider the following two gradient updates:

$$w^{t+1} = w^t - \eta^{t+1} \partial_w L(w^t, b^t)$$

$$w^{t+1} = w^t - \hat{\eta}^{t+1} \partial_w \hat{L}(w^t, b^t)$$

- Suppose:  $\hat{L} = 1000L$ 
  - How are the two step sizes related?**

$$\hat{\eta}^{t+1} = \eta / 1000$$

# Practical Rules of Thumb

- Divide Loss Function by Number of Examples:

$$w^{t+1} = w^t - \left( \frac{\eta^{t+1}}{N} \right) \partial_w L(w^t, b^t)$$

- Start with large step size
  - If loss plateaus, divide step size by 2
  - (Can also use advanced optimization methods)
  - (Step size must decrease over time to guarantee convergence to global optimum)

# Aside: Convexity

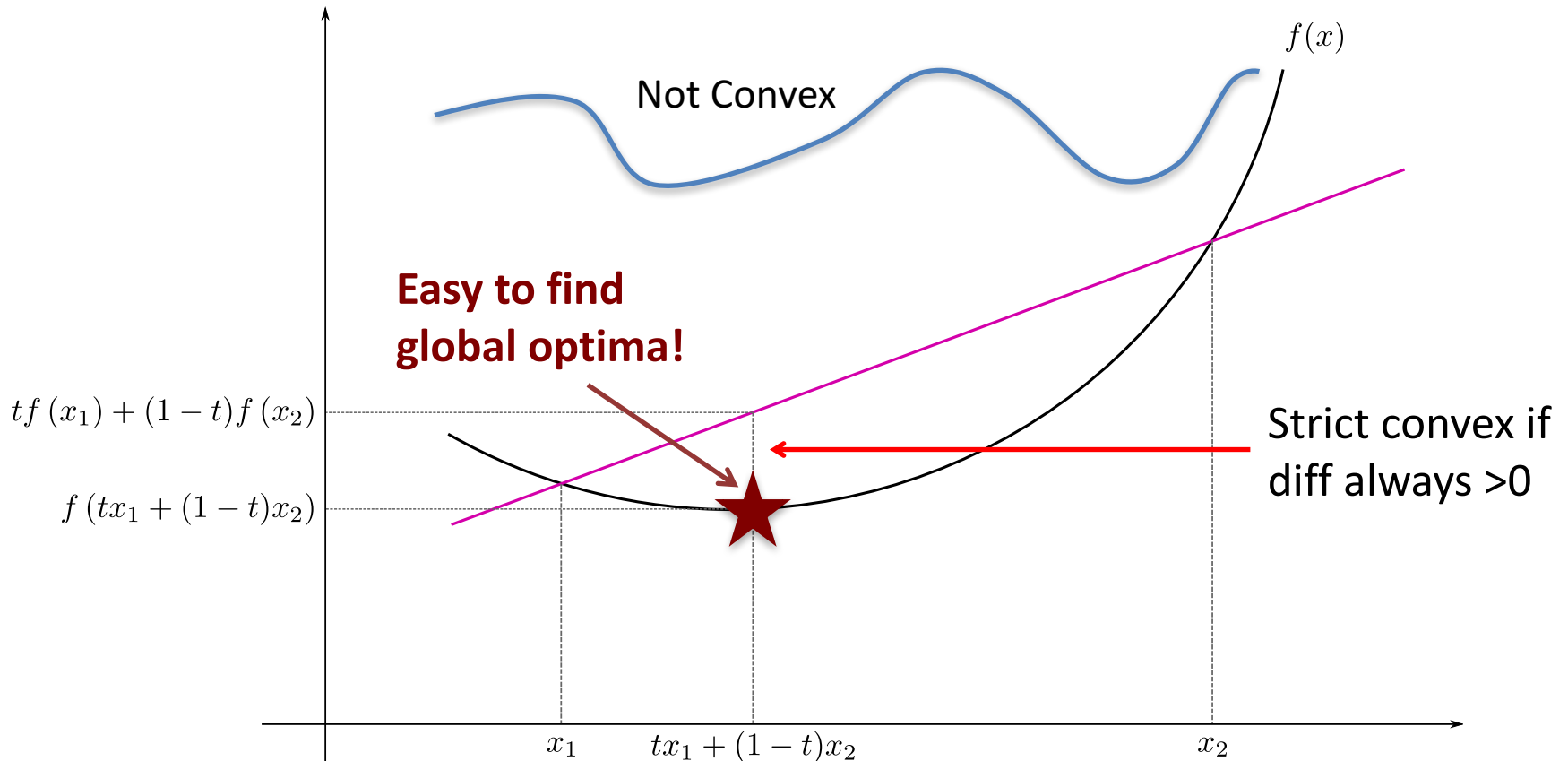


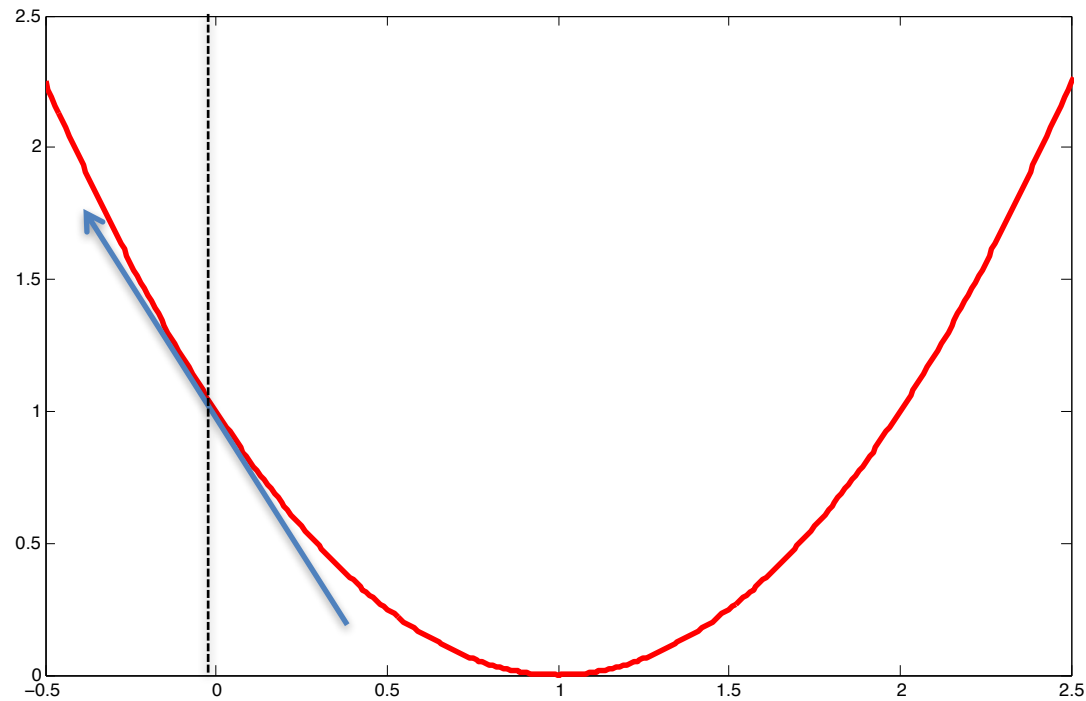
Image Source: [http://en.wikipedia.org/wiki/Convex\\_function](http://en.wikipedia.org/wiki/Convex_function)



# Aside: Convexity

$$L(x_2) \geq L(x_1) + \nabla L(x_1)^T (x_2 - x_1)$$

Function is always  
above the locally  
linear extrapolation



# Aside: Convexity

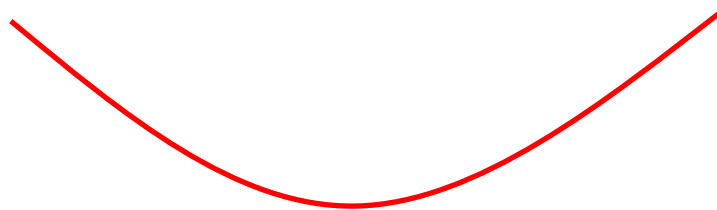
- All local optima are global optima:



Gradient Descent  
will find optimum




Assuming step  
size chosen safely

- Strictly convex: unique global optimum:



- Almost all standard objectives are (strictly) convex:
  - Squared Loss, SVMs, LR, Ridge, Lasso
  - We will see non-convex objectives later (e.g., deep learning)

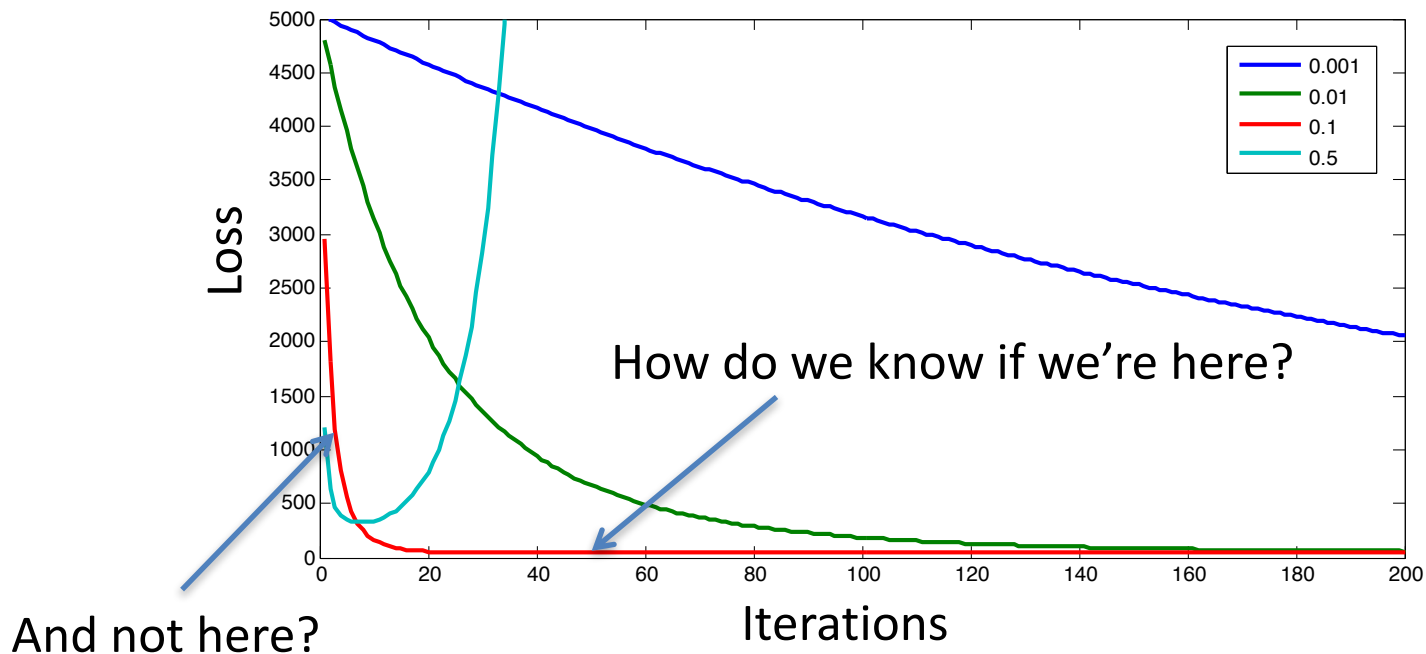
# Convergence

- Assume  $L$  is convex
- How many iterations to achieve:  $L(w) - L(w^*) \leq \varepsilon$
- If:  $|L(a) - L(b)| \leq \rho \|a - b\|$   L is “ $\rho$ -Lipschitz”
  - Then  $O(1/\varepsilon^2)$  iterations
- If:  $|\nabla L(a) - \nabla L(b)| \leq \rho \|a - b\|$   L is “ $\rho$ -smooth”
  - Then  $O(1/\varepsilon)$  iterations
- If:  $L(a) \geq L(b) + \nabla L(b)^T (a - b) + \frac{\rho}{2} \|a - b\|^2$   L is “ $\rho$ -strongly convex”
  - Then  $O(\log(1/\varepsilon))$  iterations

More Details: Bubeck Textbook Chapter 3

# Convergence

- In general, takes infinite time to reach global optimum.
- But in general, we don't care!
  - As long as we're close enough to the global optimum



# When to Stop?

- Convergence analyses = worst-case upper bounds
  - **What to do in practice?**
- Stop when progress is sufficiently small
  - E.g., relative reduction less than 0.001
- Stop after pre-specified #iterations
  - E.g., 100000
- Stop when validation error stops going down



Yisong prefers  
this option

# Limitation of Gradient Descent


- Requires full pass over training set per iteration

$$\partial_w L(w, b | S) = \partial_w \sum_{i=1}^N L(y_i, f(x_i | w, b))$$

- Very expensive if training set is huge
- **Do we need to do a full pass over the data?**

# Stochastic Gradient Descent

- Suppose Loss Function Decomposes Additively

$$L(w, b) = \frac{1}{N} \sum_{i=1}^N L_i(w, b)$$


Each  $L_i$  corresponds to a single data point

- Gradient = expected gradient of sub-functions

$$\partial_w L(w, b) = \partial_w \mathbb{E}_i [L_i(w, b)] = \mathbb{E}_i [\partial_w L_i(w, b)]$$

$$L_i(w, b) \equiv (y_i - f(x_i | w, b))^2$$


# Stochastic Gradient Descent

- Suffices to take random gradient update
  - So long as it matches the true gradient in expectation

- Each iteration t:

- Choose i at random

Expected Value is:  $\partial_w L(w, b)$


$$w^{t+1} = w^t - \eta^{t+1} \partial_w L_i(w, b)$$

$$b^{t+1} = b^t - \eta^{t+1} \partial_b L_i(w, b)$$

- **SGD is an online learning algorithm!**

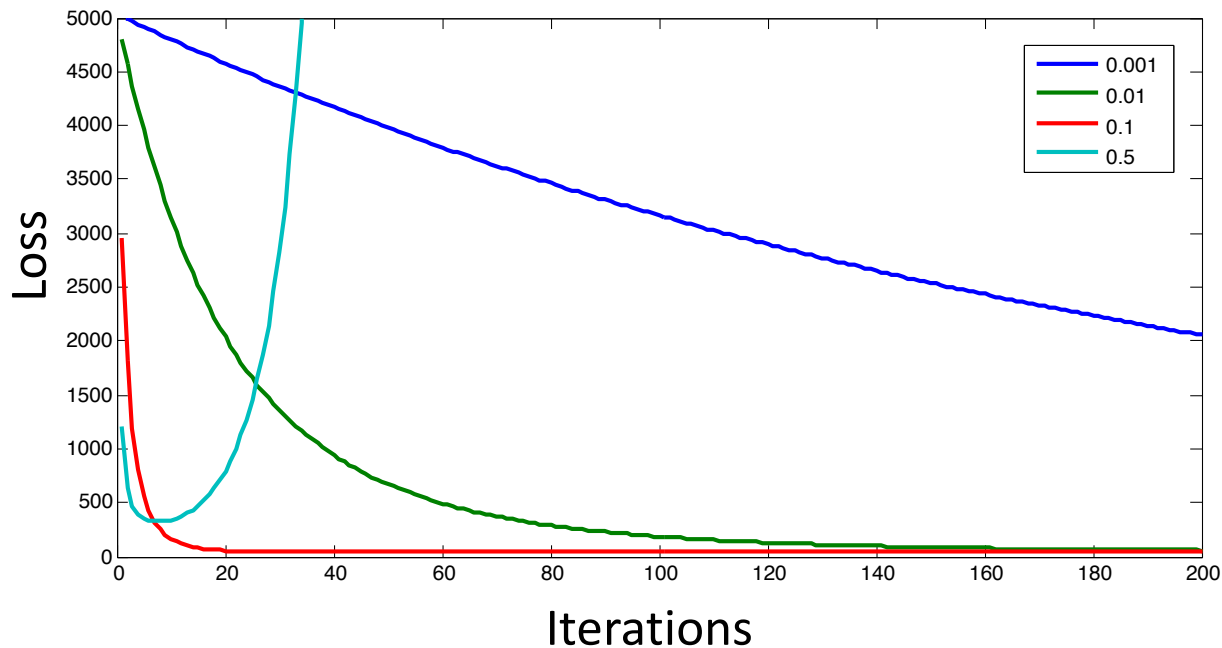


# Mini-Batch SGD

- Each  $L_i$  is a small batch of training examples
  - E.g., 500-1000 examples
  - Can leverage vector operations
  - Decrease volatility of gradient updates
- Industry state-of-the-art
  - Everyone uses mini-batch SGD
  - Often parallelized
    - (e.g., different cores work on different mini-batches)

# Checking for Convergence

- How to check for convergence?
  - Evaluating loss on entire training set seems expensive...



# Checking for Convergence

- How to check for convergence?
  - Evaluating loss on entire training set seems expensive...
- Don't check after every iteration
  - E.g., check every 1000 iterations
- Evaluate loss on a subset of training data
  - E.g., the previous 5000 examples.

# Recap: Stochastic Gradient Descent

- Conceptually:
  - Decompose Loss Function Additively
  - Choose a Component Randomly
  - Gradient Update
- Benefits:
  - Avoid iterating entire dataset for every update
  - Gradient update is consistent (in expectation)
- Industry Standard

# Perceptron Revisited

## (What is the Objective Function?)

- $w^1 = 0, b^1 = 0$

$$f(x | w) = \text{sign}(w^T x - b)$$

- For  $t = 1 \dots$

- Receive example  $(x, y)$

- If  $f(x | w^t) = y$

- $[w^{t+1}, b^{t+1}] = [w^t, b^t]$

- Else

- $w^{t+1} = w^t + yx$

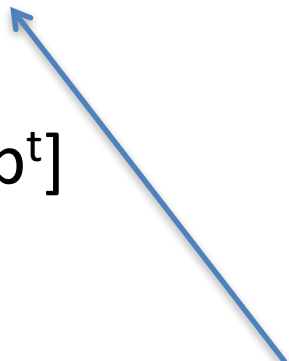
- $b^{t+1} = b^t - y$

**Training Set:**

$$S = \{(x_i, y_i)\}_{i=1}^N$$

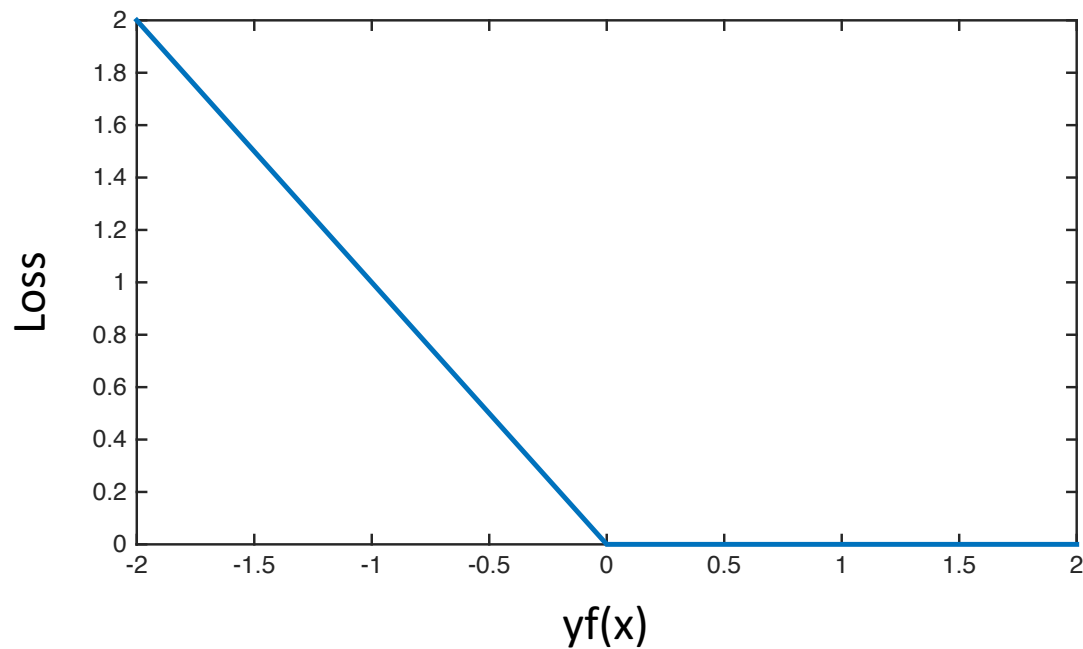
$$y \in \{+1, -1\}$$

Go through training set  
in arbitrary order  
(e.g., randomly)



# Perceptron (Implicit) Objective

$$L_i(w, b) = \max \{0, -y_i f(x_i | w, b)\}$$



# Recap: Complete Pipeline

$$S = \{(x_i, y_i)\}_{i=1}^N$$

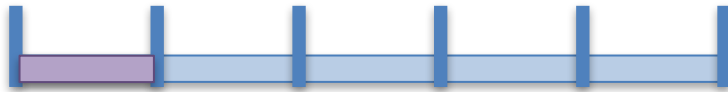
Training Data

$$f(x | w, b) = w^T x - b$$

Model Class(es)

$$L(a, b) = (a - b)^2$$

Loss Function



$$\operatorname{argmin}_{w, b} \sum_{i=1}^N L(y_i, f(x_i | w, b)) \quad \text{Use SGD!}$$

Cross Validation & Model Selection



Profit!

# Next Week

- Different Loss Functions
  - Hinge Loss (SVM)
  - Log Loss (Logistic Regression)
- Non-linear model classes
  - Neural Nets
- Regularization
- Thursday Recitation: Linear Algebra & Calculus