Convolutional Neural Networks (CNNs) and Support Vector Machines (SVMs)

CSCI-P556 Applied Machine Learning

Lecture 18

D.S. Williamson

Agenda and Learning Outcomes

Today’s Topics

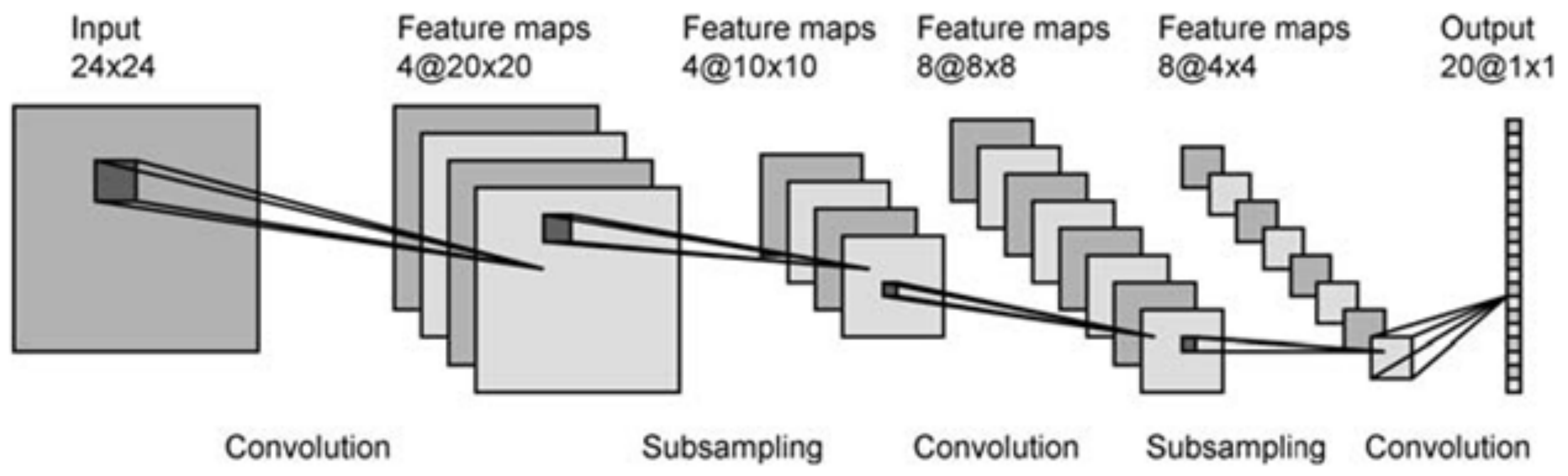
- **Topics**:
    - Convolutional Neural Networks (CNNs)
    - Support Vector Machines, Part I

- **Announcements**
    - Quiz #2 on Thursday
    - Project proposal comments (tomorrow)
    - hw#3 coming soon (today or tomorrow)

Convolutional Neural Networks

Convolutional Neural Networks (CNNs)

- Used for processing data with grid-like topology (i.e. images). Networks use convolution in place of general matrix multiplication
- Good at capturing local (short-term) dependencies and correlations (e.g. correlations amongst adjacent pixels in an image, or dependencies across nearby frequencies of an audio signal)
- There are four main operations in CNNs
    - Convolution
    - Nonlinear Activation Function (i.e. ReLU)
    - Pooling (or Subsampling)
    - Classification



Convolution

- Convolution is a linear mathematical operation on two functions

- Given functions *x(t)* and *w(t),* the convolution of *x(t)* and *w(t)* is as follows

$$s(t) = x(t) * w(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a)$$

- This step is used to extract “features” from an input, so it is also referred to as the ***feature map*** stage

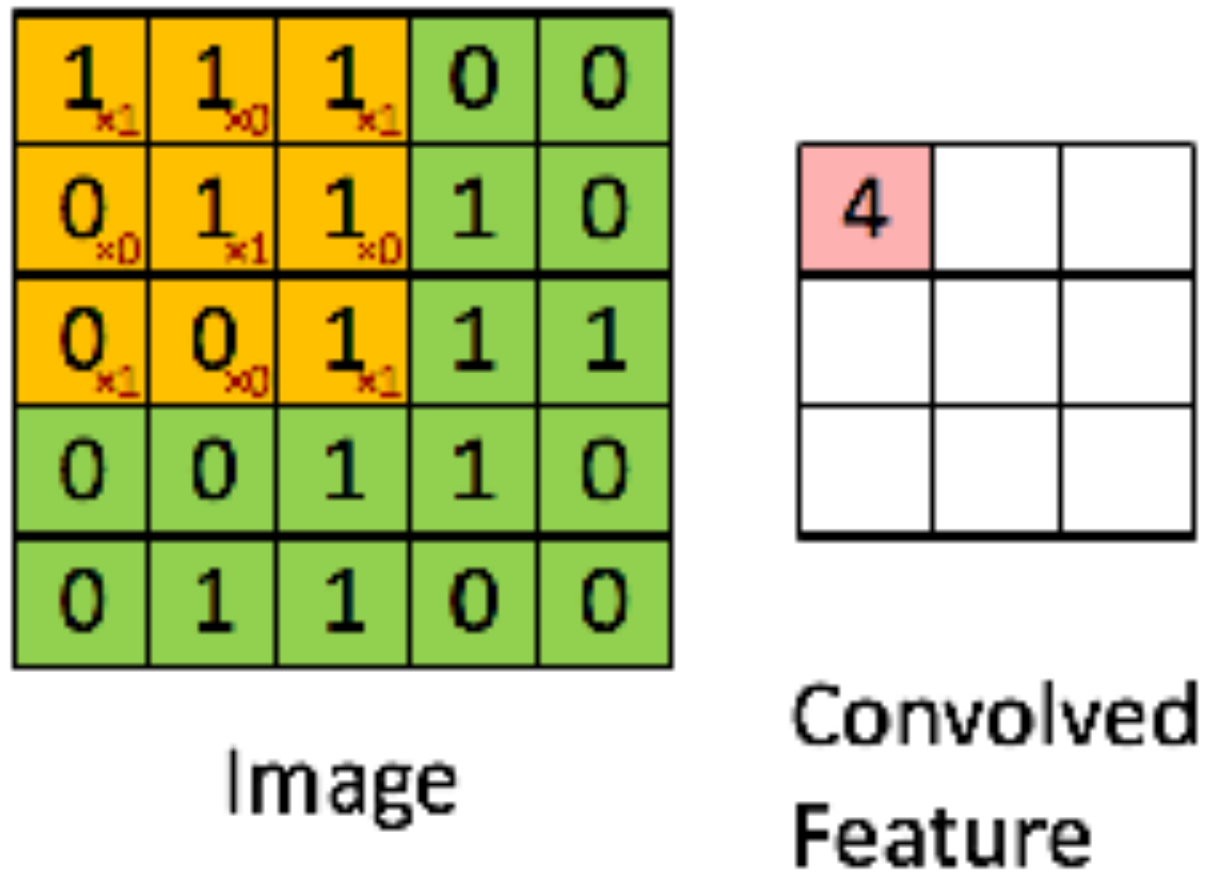
Example: Convolution on an Image

- Suppose you are given the following binary image, X

1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

- You want to convolve this image with matrix, W as shown below

1	0	1
0	1	0
1	0	1

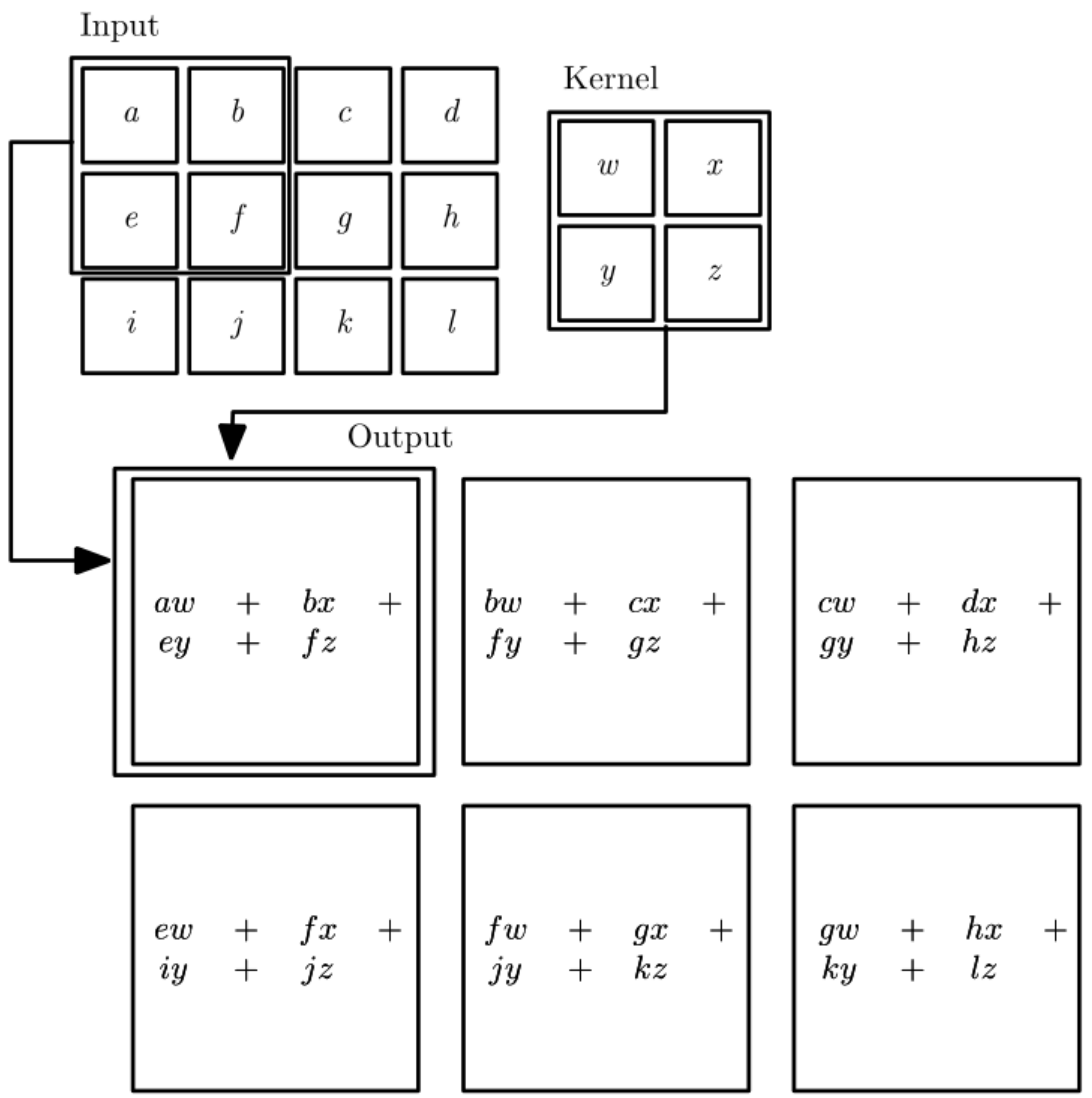


Images courtesy of *the data science blog*

Convolution Operation

- A mathematical depiction of the convolution operation on an input image

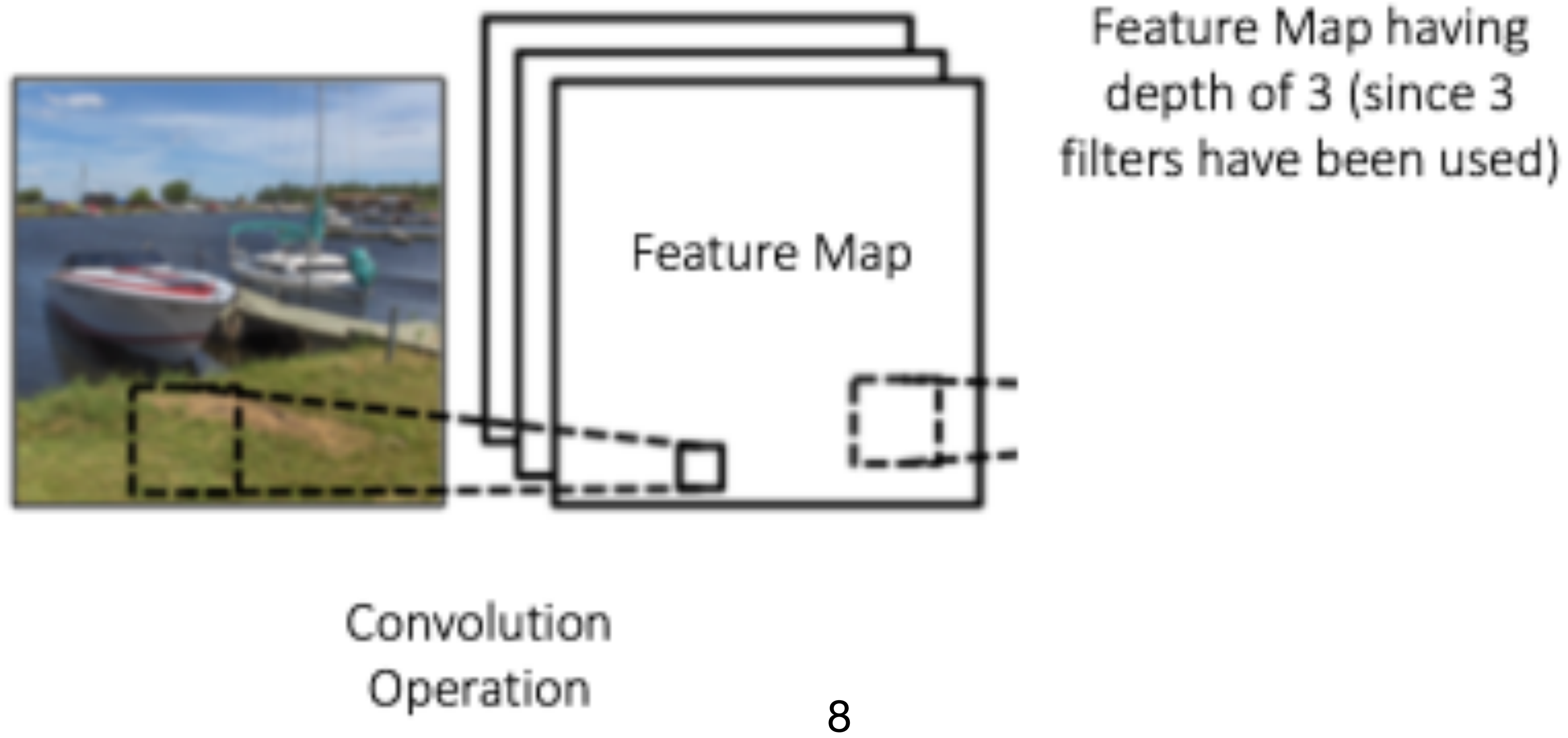
- A CNN learns the values of the filter (or kernel) on its own during the training process



Images courtesy of *(Goodfellow 2016)*

Feature Map

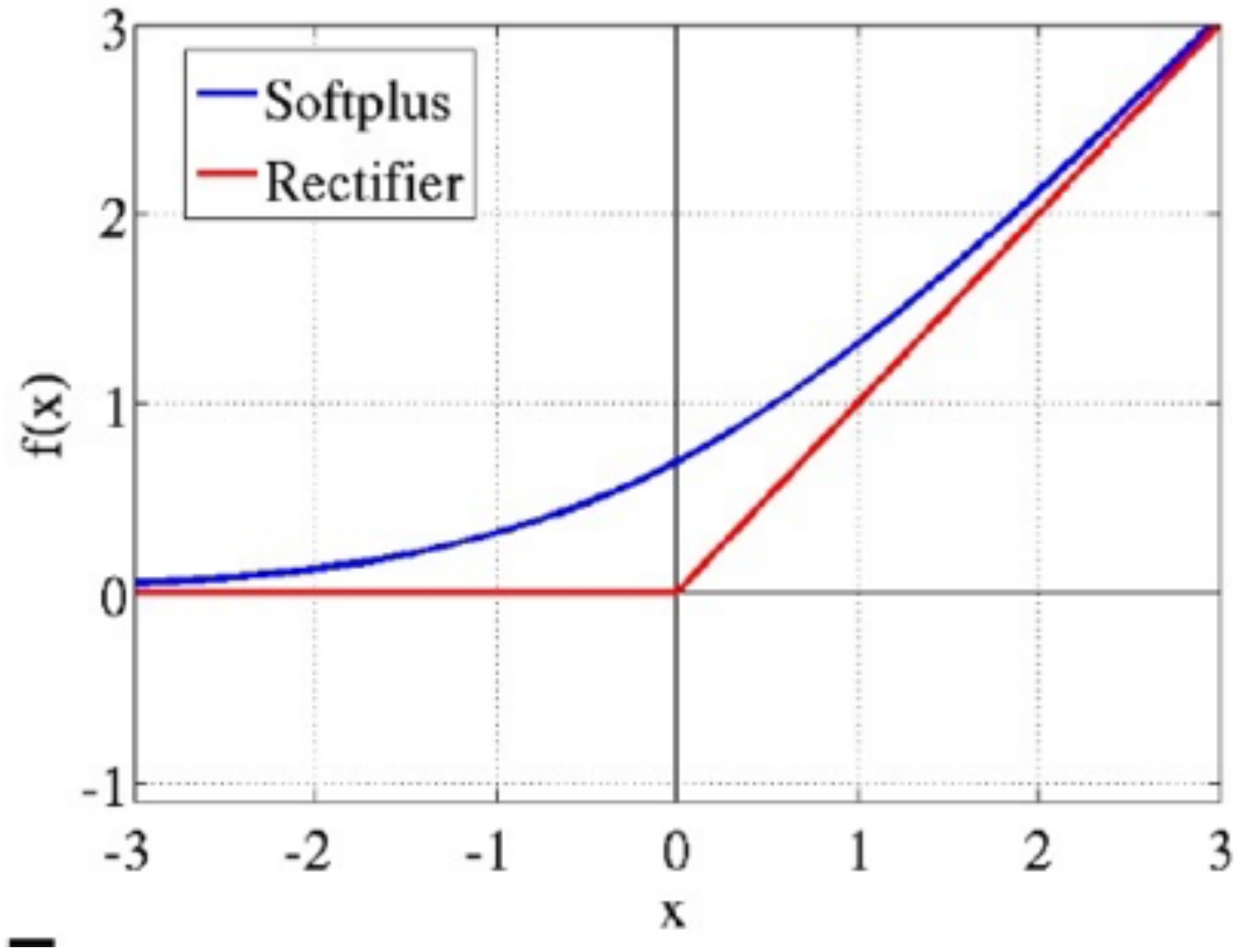
- The size of the Feature Map (resulting image after convolution) is controlled by three parameters
  - **Depth**: Number of different filters to use for the convolution operation
  - **Stride**: Number of pixels used to slide the filter across the input
  - **Zero-padding**: May pad the input with zeros around the border



Images courtesy of *the data science blog*

Rectified Linear (ReLU) Activation

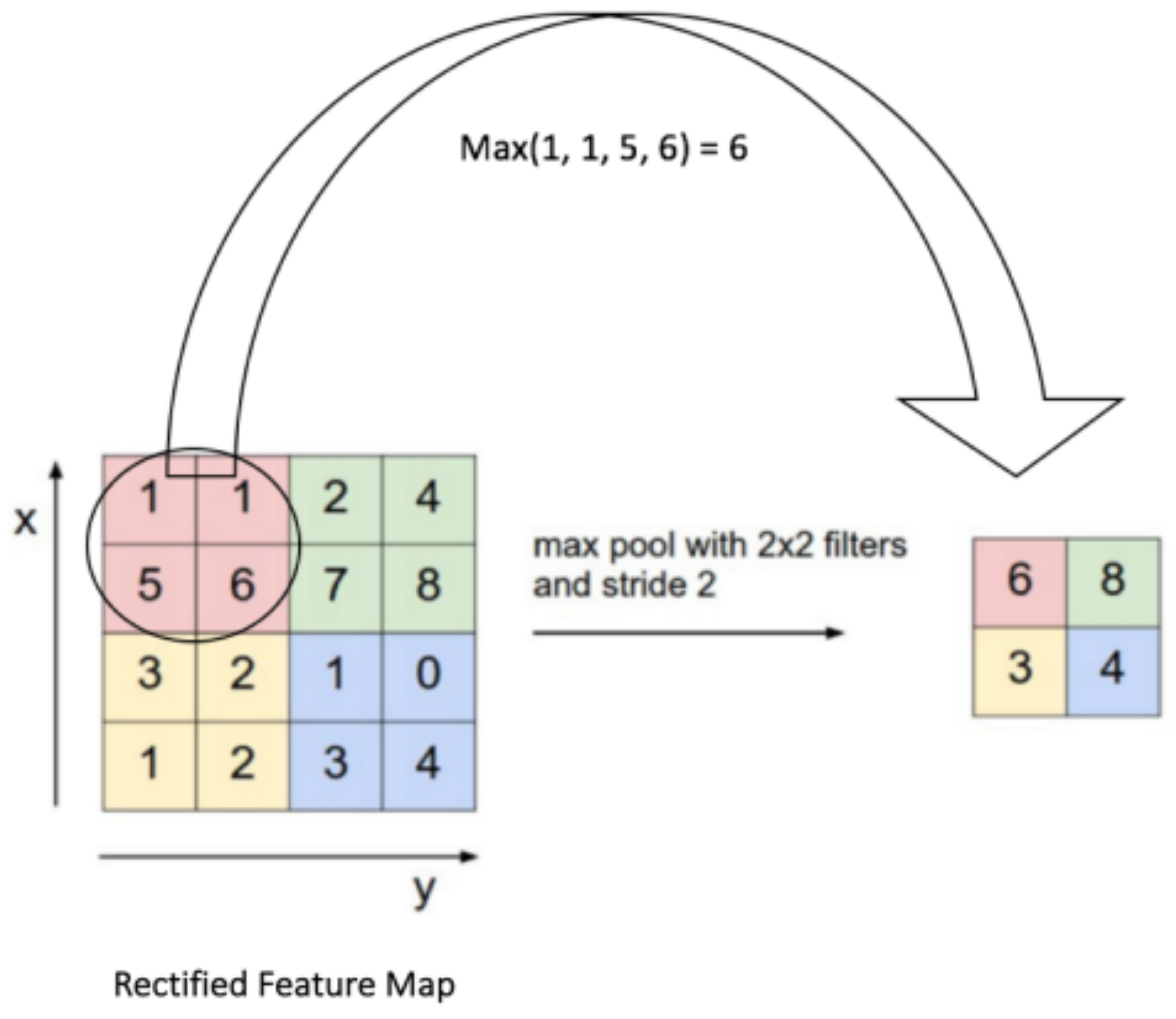
- A rectified linear (ReLU) activation operation may be applied after the convolution operation
  - Introduces nonlinearity to the network
  - ReLU(x) = max(0, x)
  - Applied to every element (pixel)
  - Negative values are replaced by 0



- Other nonlinear activation functions may be used instead

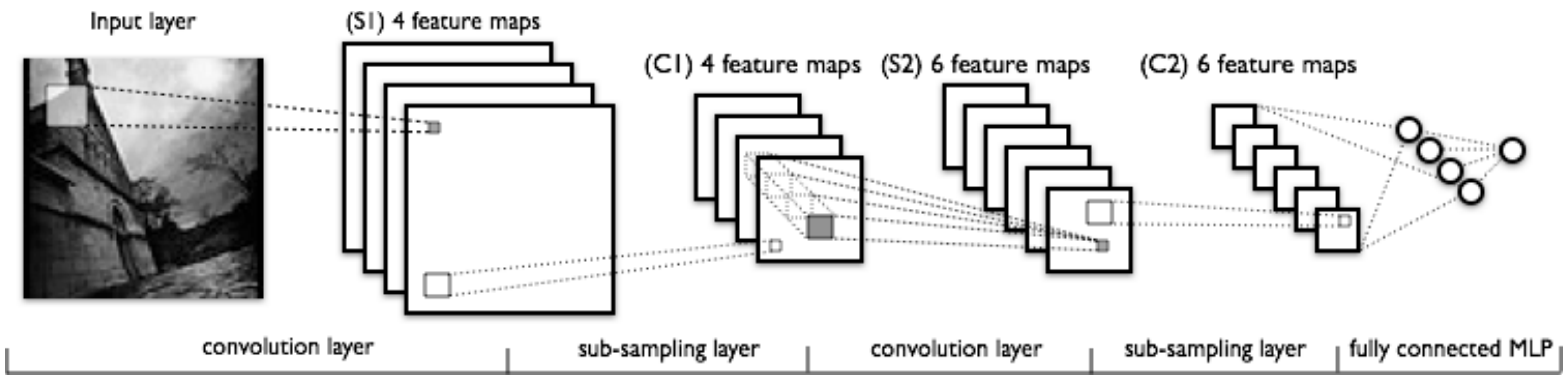
Pooling

- Pooling (aka spatial pooling, subsampling, or downsampling) is used to reduce the dimensionality of the feature map
- Different types of pooling include: Max, Average, Sum, etc.
- A window is defined, and the pooling operation is performed over the elements within that window
- The pooling window slides over the feature map by the stride amount
- It is applied to each feature map



CNN

- Multiple layers of Convolution, Activation, and Pooling may be used in a CNN
- These layers act as feature extraction, to find useful features from the input
- Generally, a final Fully Connected layer is added via a DNN for classification or regression purposes



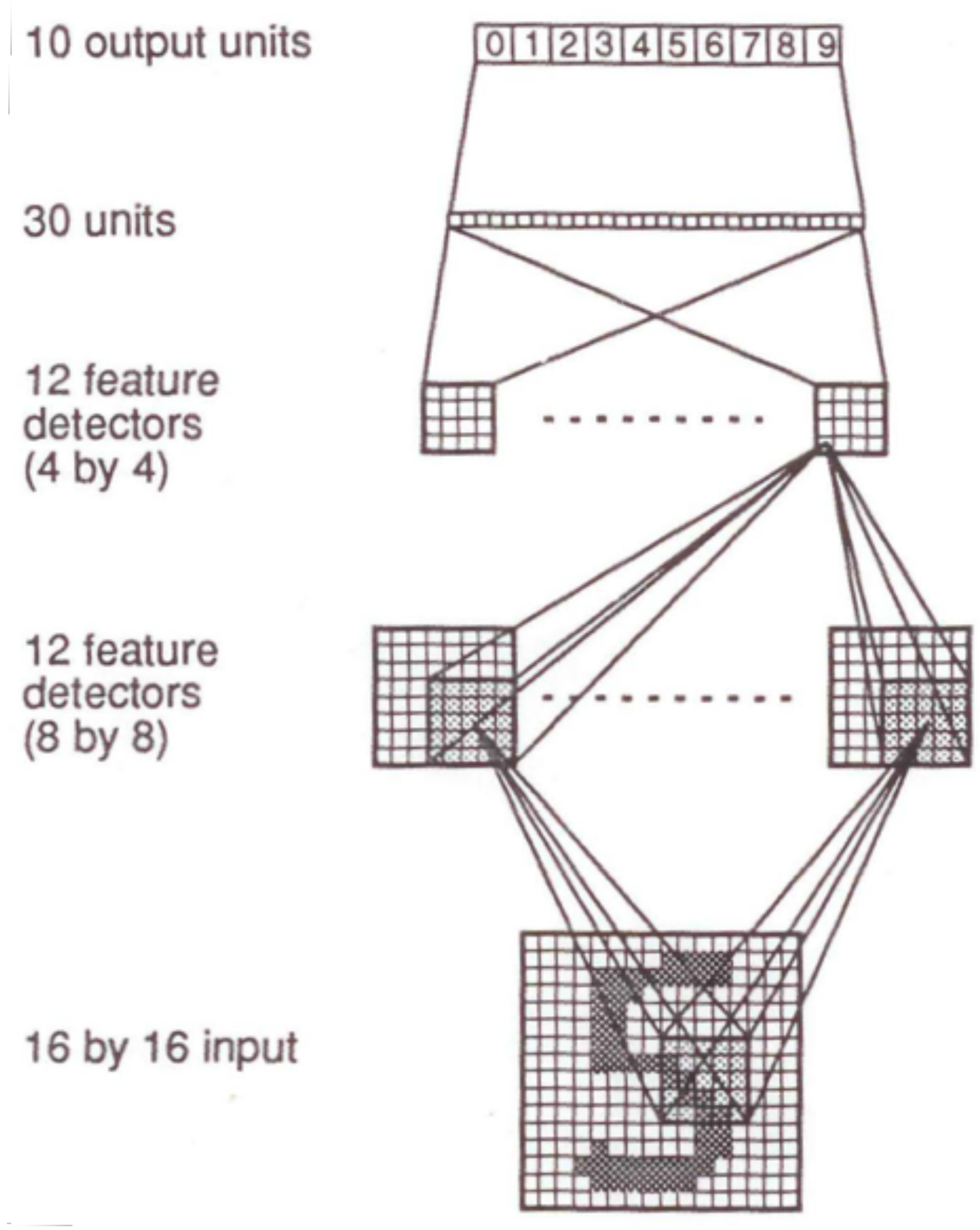
CNN Training

- The Backpropagation algorithm is used to train the parameters of a CNN
- Basic steps:
  - Randomly initialize all filters (or kernels) and weights
  - Propagate the input forward through the layers of the CNN (convolution, activation, pooling, DNN) to get an output(s)
  - Calculate the error between the actual output(s) and the desired output(s)
  - Use Backpropagation to calculate the gradients and deltas, and then update the filters and weights accordingly

An Early CNN Application

Handwritten Zip code Recognition (1989)

- **Input**: binary pixels for each digit (16 x 16 dimensional)
- Padded with -1.
- **Output**: 10 digits, using sigmoid activations
- Scaled hyperbolic tanget activation function
- **Architecture**: 4 layers (12x8x8 – 12x4x4 -30 -10)
    - Layer 1: kernel of size 5x5, stride of 2, depth of 12
    - Layer 2: kernel of size 5x5, stride of 2, depth of 12
    - Layer 3: Fully-connected layer with 30 units
    - Layer 4: 10 unit output layer
- **Performance**: Trained on 7300 digits and tested on 2000 new ones
    - Achieved 1% error on the training set and 5% error on the test set
    - If allowing rejection (no decision), 1% error on the test set
    - This task is not easy



CNN in PyTorch

- The following classes can be used within a defined class that implements the desired CNN architecture. (e.g. in the *__init__()* and *forward()* functions)
  - **Convolution**: Conv2d (or Conv1d) to perform convolution operation
  - **Pooling**: Maxpool2D, Avgpool2D,..

Docs > torch.nn > Conv2d

Conv2d

CONV2D

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros')` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

This module supports TensorFloat32.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of implicit padding on both sides for `padding` number of points for each dimension.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,

Docs > torch.nn > MaxPool2d

MAXPOOL2D

CLASS `torch.nn.MaxPool2d(kernel_size, strid` `return_indices=False, ceil_mode=False`

Applies a 2D max pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C, H, W), output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \max_{m=0,\ldots,kH-1} \max_{n=0,\ldots,kW-1} \text{input}(N_i, C_j, \text{stride}[0] \times h + m, \text{stride}[1] \times w + n)$$

If `padding` is non-zero, then the input is implicitly zero-padded o controls the spacing between the kernel points. It is harder to des `dilation` does.

Docs > torch.nn > AvgPool2d

AvgPool2c

AVGPOOL2D

CLASS `torch.nn.AvgPool2d(kernel_size, stride=None, padding=0, ceil_mode=False, count_include_pad=True, divisor_override=None)` [SOURCE]

Applies a 2D average pooling over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C, H, W), output (N, C, H_{out}, W_{out}) and `kernel_size` (kH, kW) can be precisely described as:

$$out(N_i, C_j, h, w) = \frac{1}{kH * kW} \sum_{m=0}^{kH-1} \sum_{n=0}^{kW-1} input(N_i, C_j, stride[0] \times h + m, stride[1] \times w + n)$$

If `padding` is non-zero, then the input is implicitly zero-padded on both sides for `padding` number of points.

NOTE

When ceil_mode=True, sliding windows are allowed to go off-bounds if they start within the left padding or the input. Sliding windows that would start in the right padding region are ignored.

Support Vector Machines

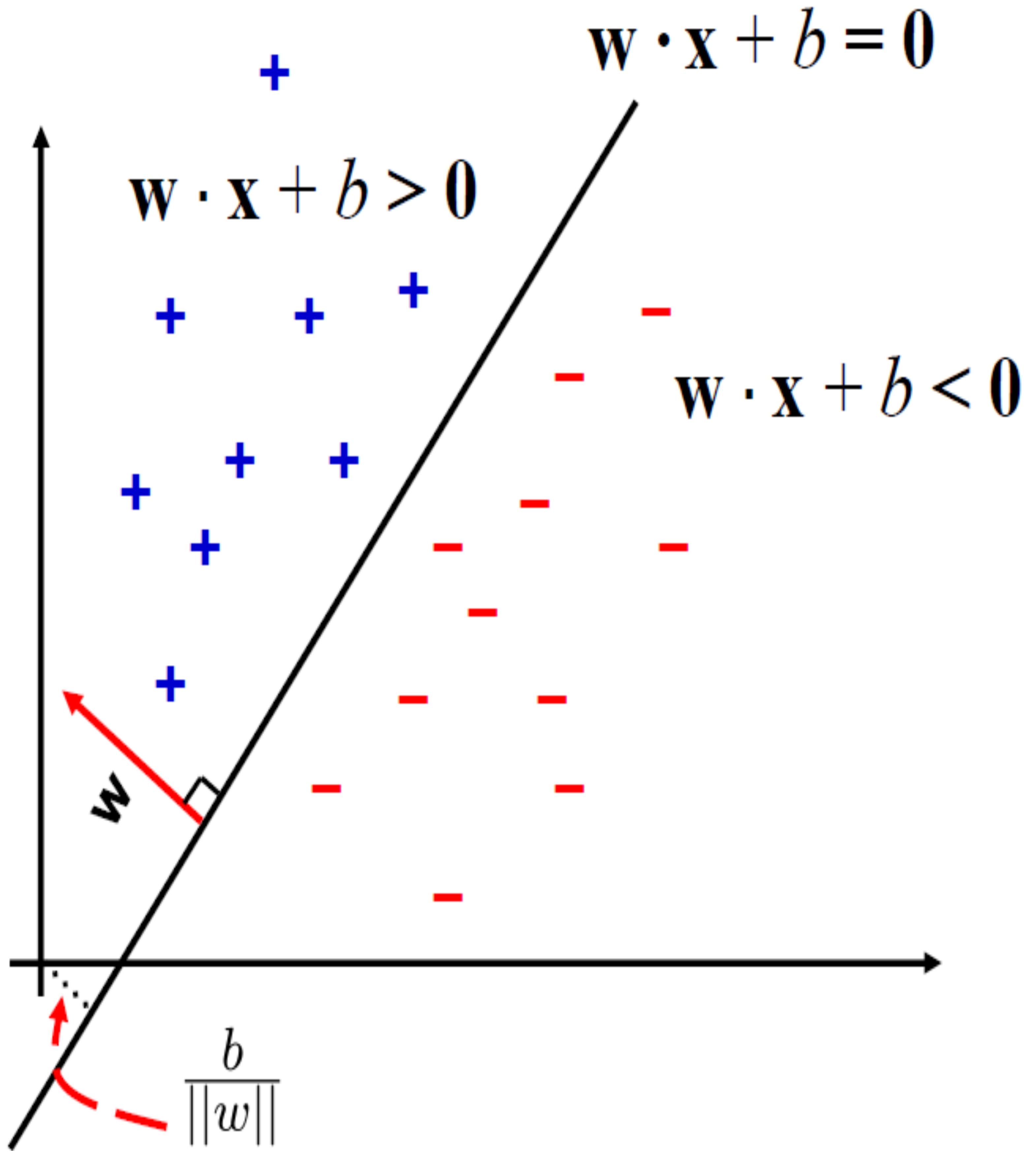
Warning!

- This is just a first introduction to SVMs – very basic version of SVM introduced
- We will discuss more on the advanced techniques later
  - Hence, Strong assumptions in this case

The Classification Problem

- ***Binary classification*** can be viewed as the task of ***separating classes in the feature space***

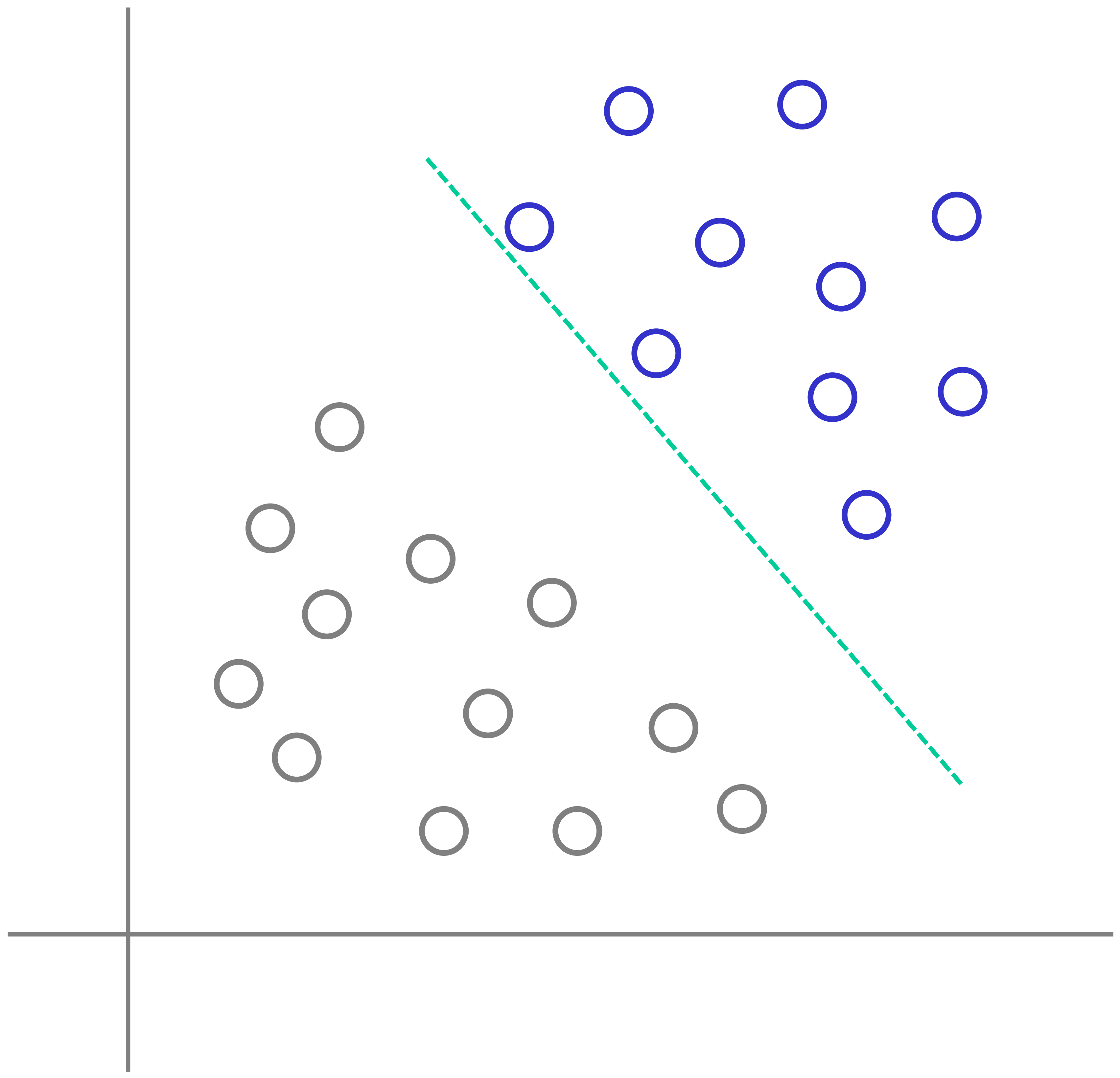
- This is accomplished by formulating a decision boundary
  - **w** is the slope of the line
  - **b** is the intercept



Recall: Perceptrons

- **Perceptron Convergence Theorem**: If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations

- The solution weight vector is **<u>not</u>** unique. There are ***<u>infinite possible solutions</u>*** and decision boundaries.
  - Perceptrons find any separating hyperplane
  - The hyperplane depends on initialization and ordering of training points

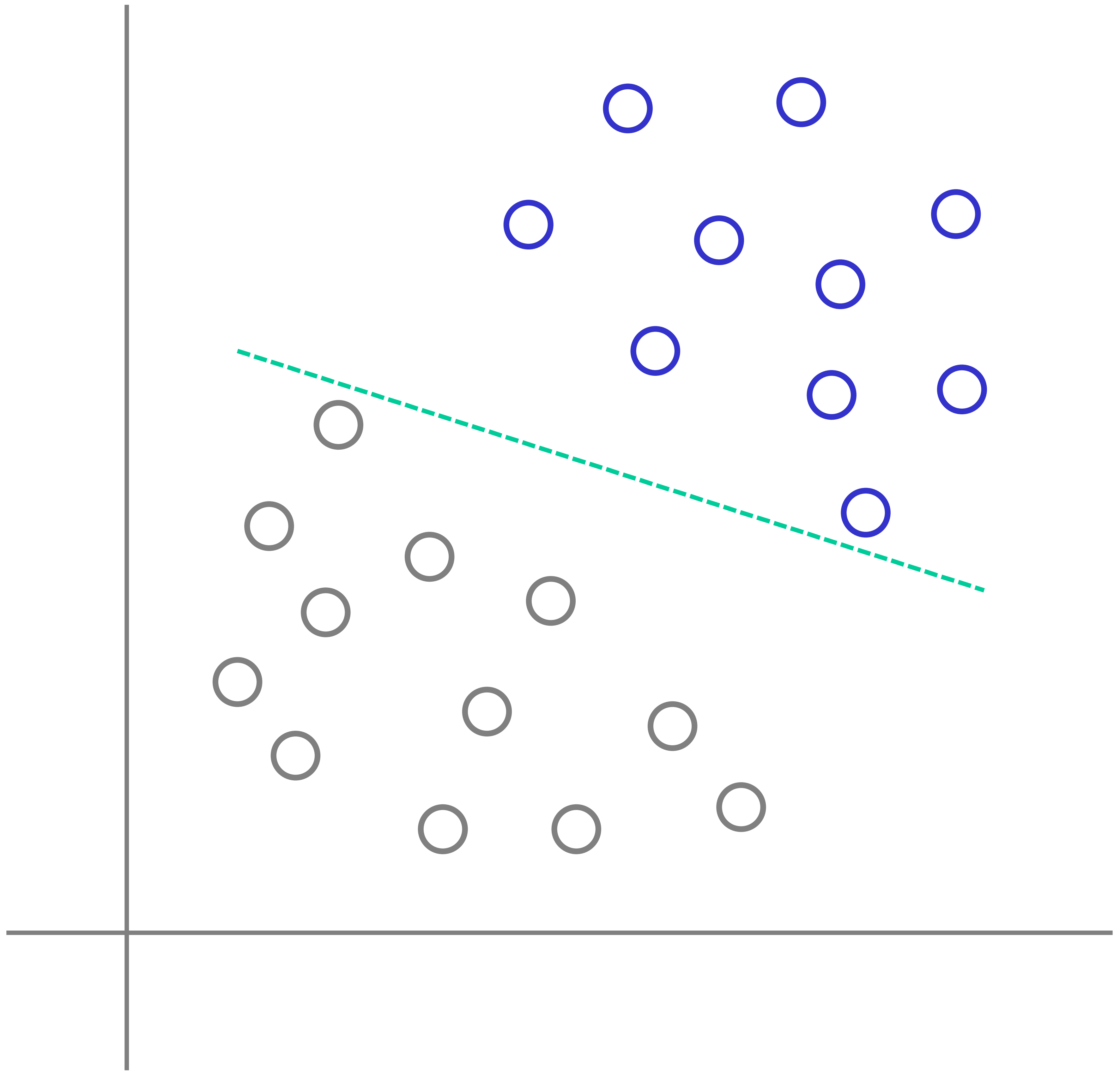


Recall: Perceptrons

- **Perceptron Convergence Theorem**: If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations

- The solution weight vector is **not** unique. There are ***infinite possible solutions*** and decision boundaries.
    - Perceptrons find any separating hyperplane
    - The hyperplane depends on initialization and ordering of training points

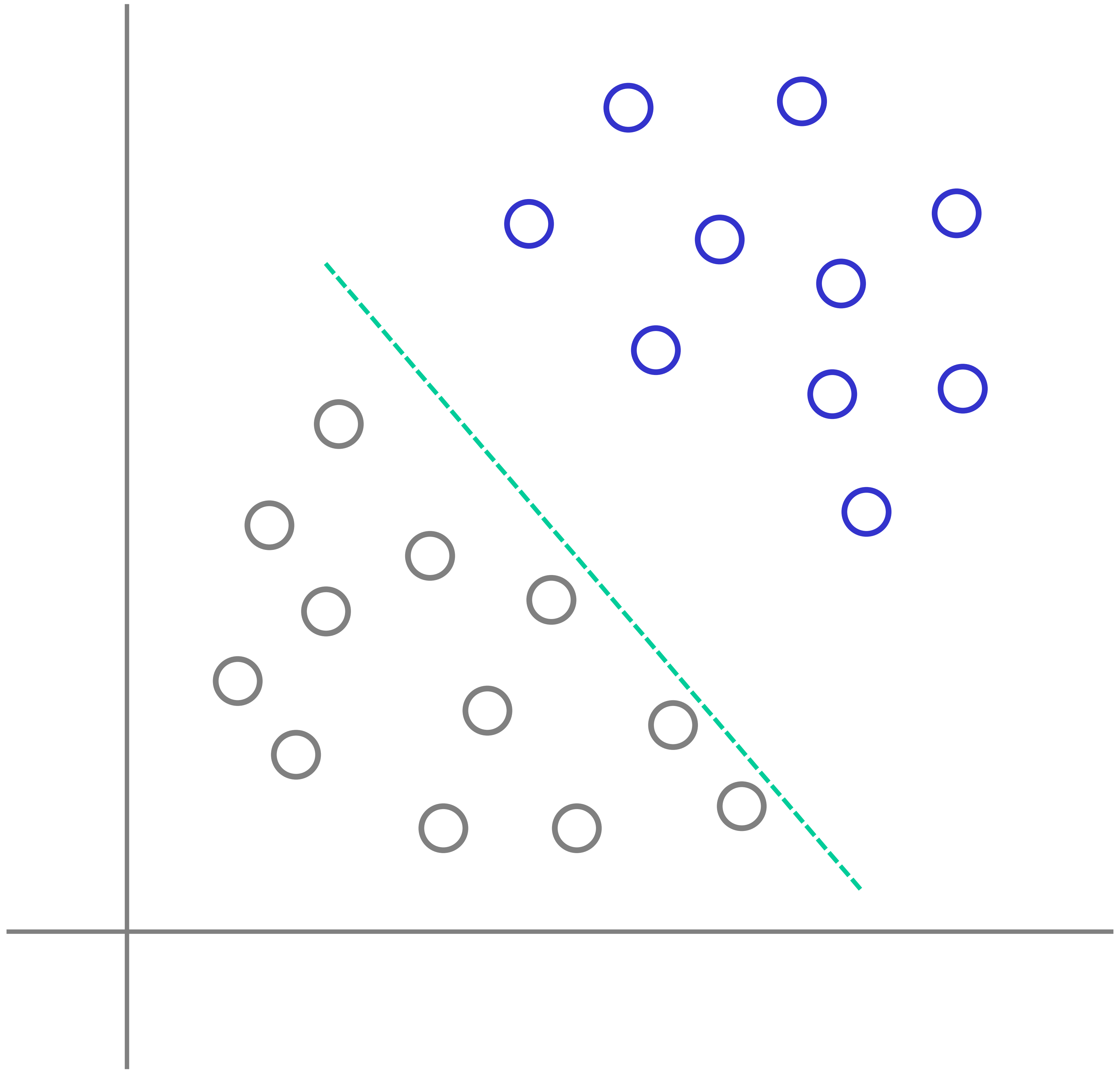
- ***If done differently***



Recall: Perceptrons

- **Perceptron Convergence Theorem**: If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations

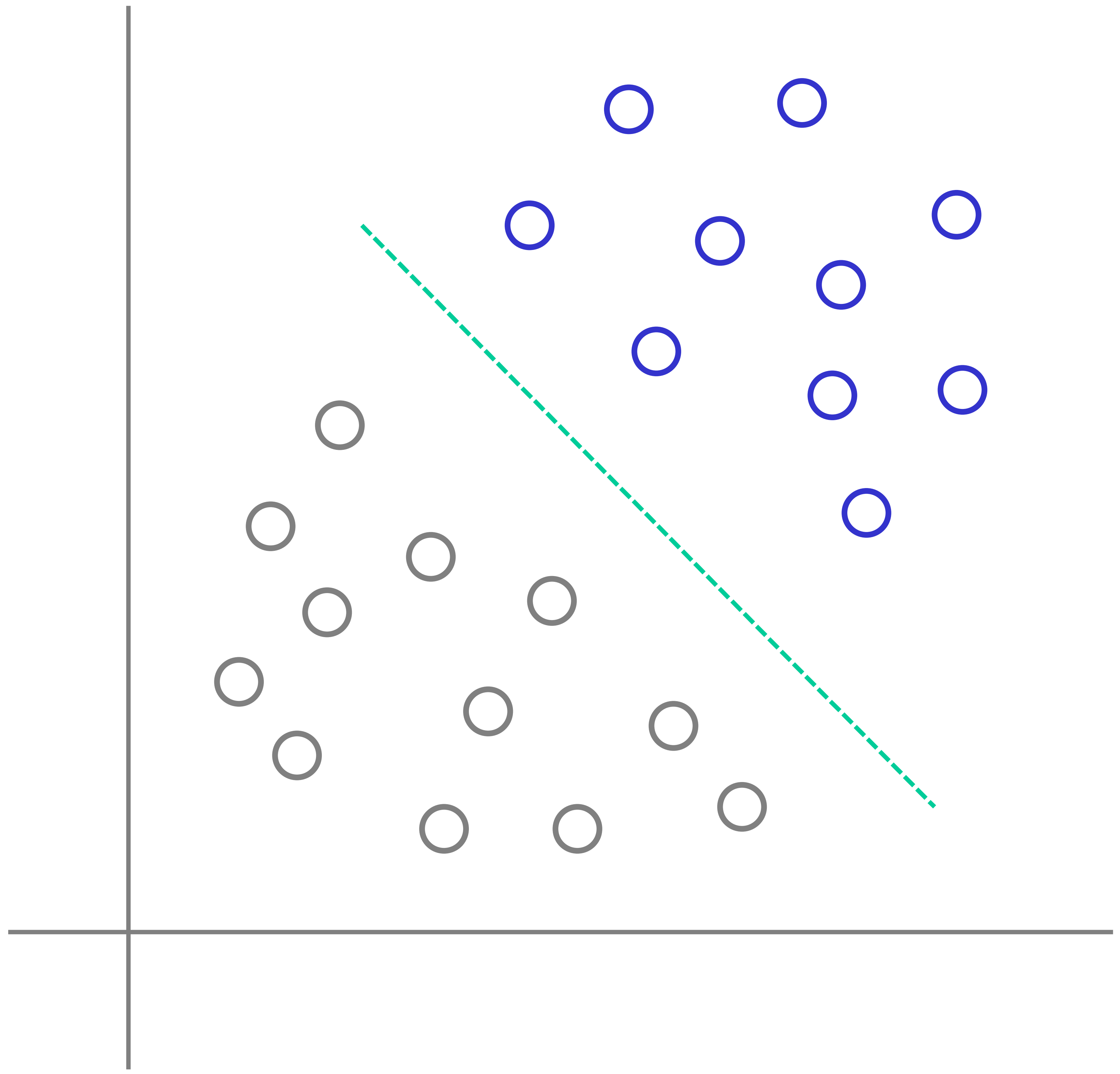
- The solution weight vector is **not** unique. There are ***infinite possible solutions*** and decision boundaries.
  - Perceptrons find any separating hyperplane
  - The hyperplane depends on initialization and ordering of training points
- ***If done differently….Again***



Recall: Perceptrons

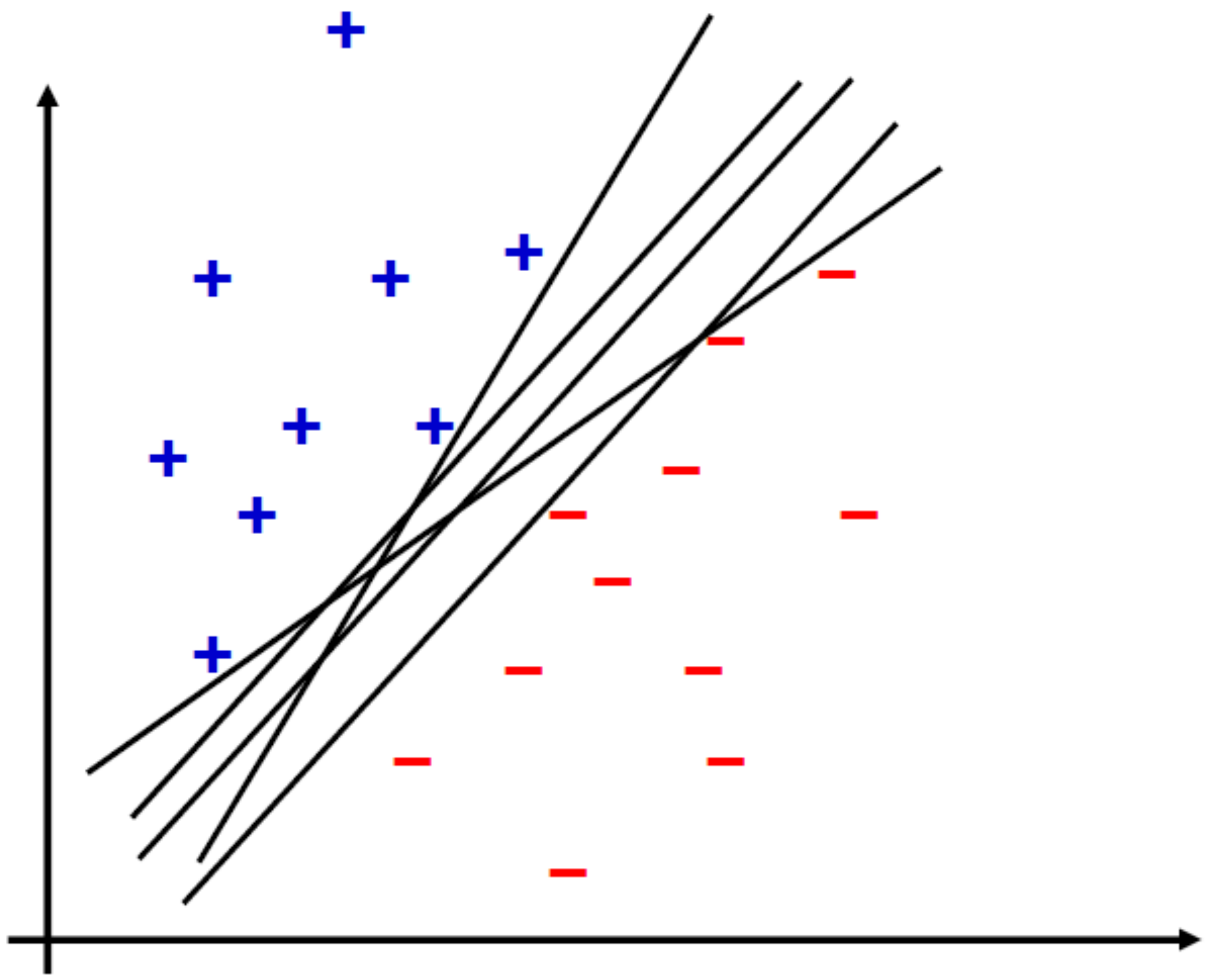
- **Perceptron Convergence Theorem**: If a classification problem is linearly separable, a perceptron will reach a solution in a finite number of iterations

- The solution weight vector is **<u>not</u>** unique. There are ***<u>infinite possible solutions</u>*** and decision boundaries.
  - Perceptrons find any separating hyperplane
  - The hyperplane depends on initialization and ordering of training points
- ***<u>If done differently….Again…And Again</u>***



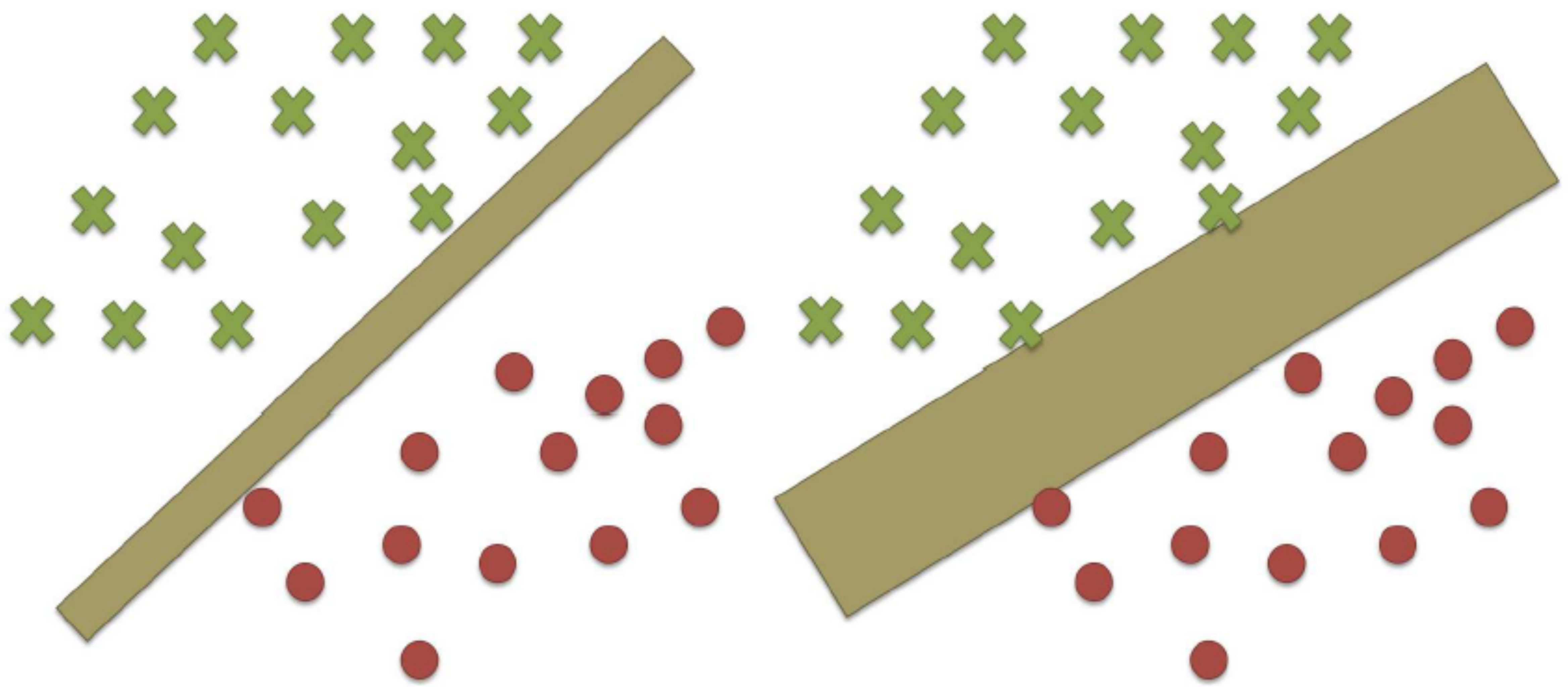
Linear Separators

- Which is the best linear separator?
  - Depends on the goal
  - Goal is to classify **<u>accurately</u>** and **<u>generalize</u>** to new examples.



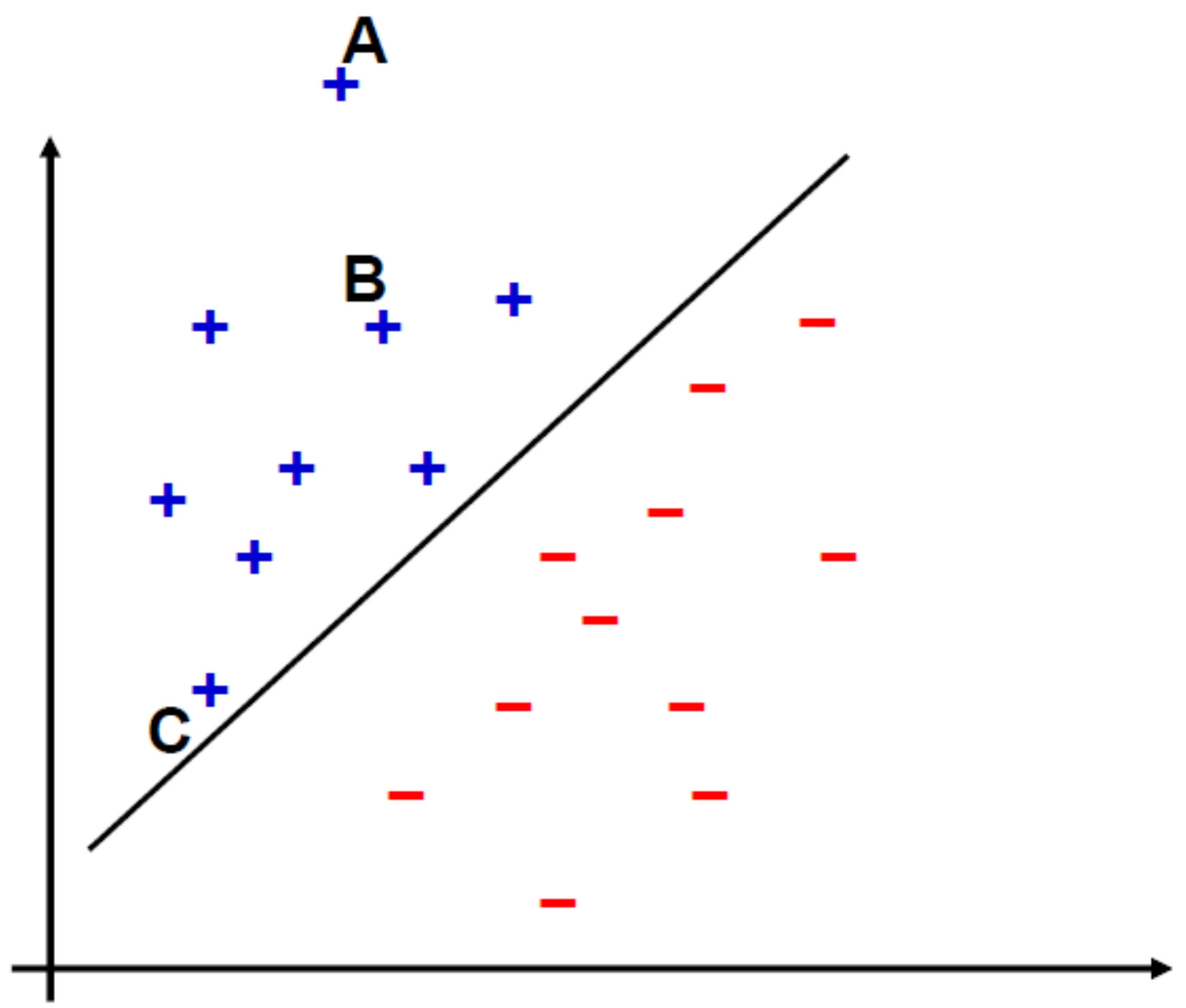
Notion of Margins

- Many different hyperplanes can classify the data. Which one will work best?
- The **hyperplane** that **maximizes** the **separation between the two classes** (the margin)



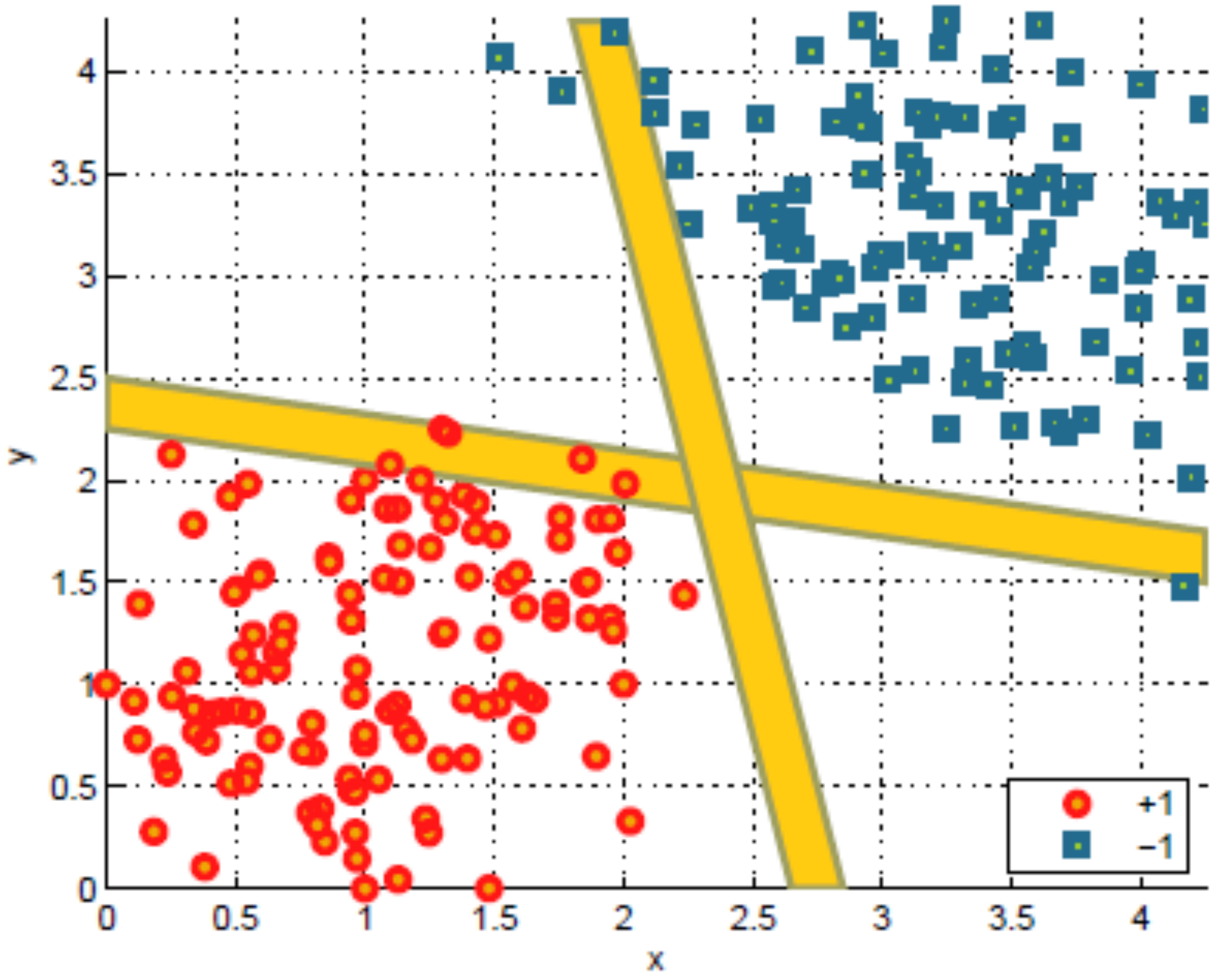
Intuition of a Margin

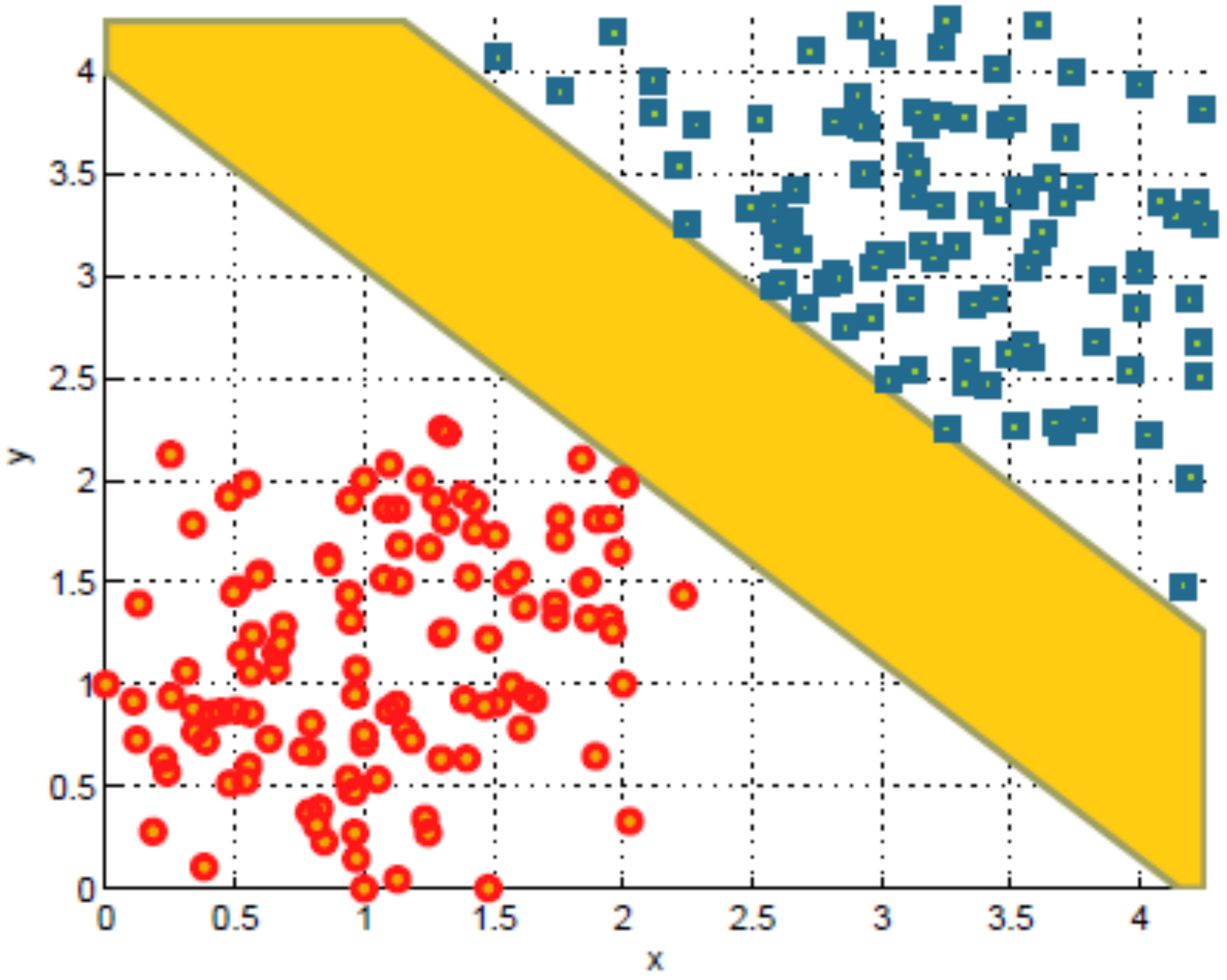
- Consider points A, B, and C
- We are quite confident in our prediction for A because it is far from the decision boundary.
- In contrast, we are not so confident in our prediction for C because a slight change in the decision boundary may flip the decision
- Given a training set, we would like to make all predictions correct and confident! This leads to the concept of margin



Why Max Margin?

- Minimizes generalization error. Works well on **Future data**
- Minimizes Complexity. **Fewer support vectors**
- Minimizes the capacity of the classifier. **Eliminates overfitting**





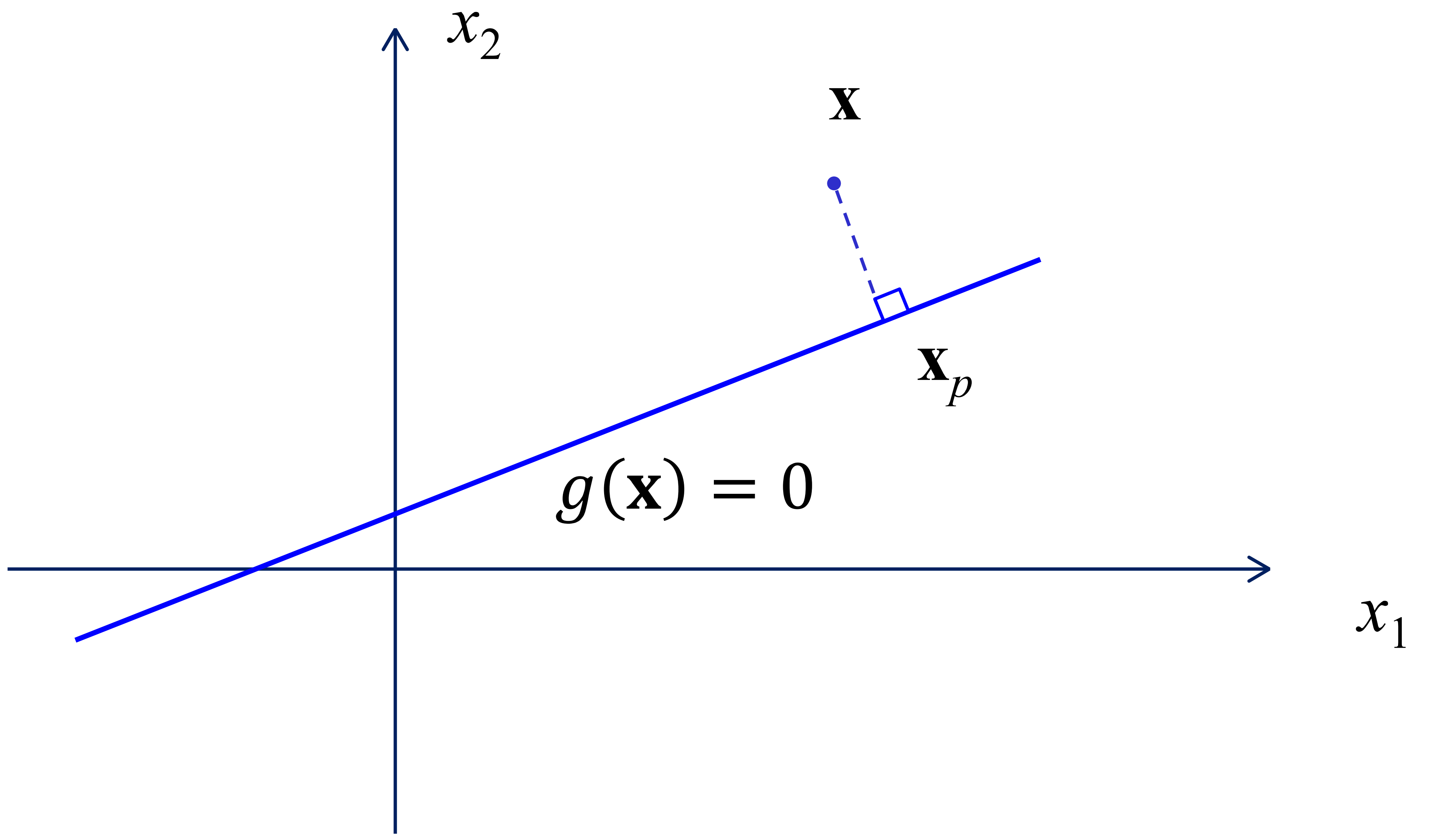
Decision Boundary

Need to know distance from point to the decision boundary

- Given a decision boundary (e.g. linear discriminant function)

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$$

- To find its distance to a given pattern **x**, project **x** onto the decision boundary



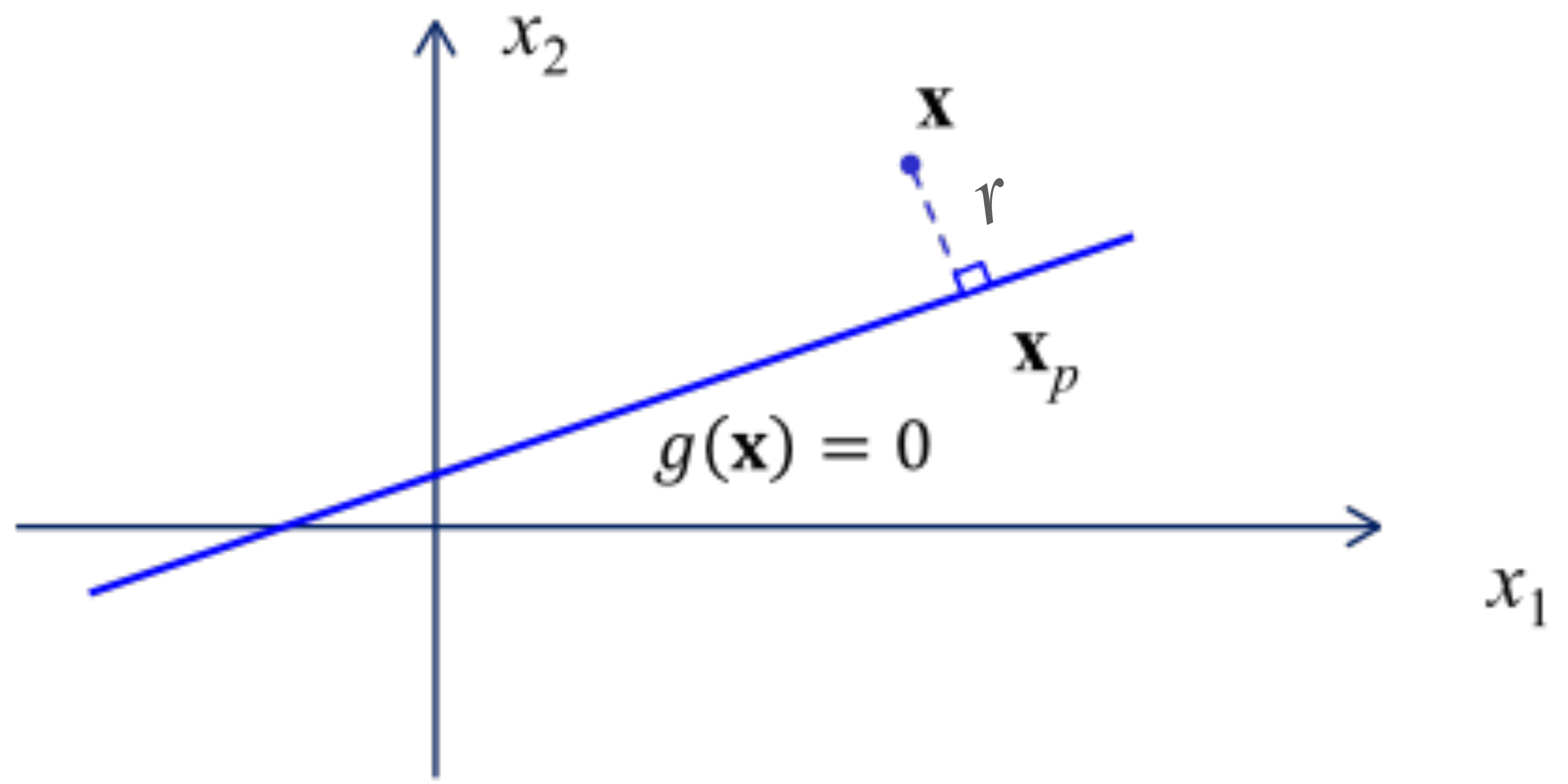
Decision Boundary (cont.)

- **x** can be re-written as a function of the projection and the weights

$$\mathbf{x} = \mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||}$$

- $\mathbf{x}_p$ is **x**'s projection
- The second term arises from the fact that the weight vector is perpendicular to the decision boundary
- The algebraic distance *r* is positive if **x** is on the positive side of the boundary and negative if **x** is on the negative side

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$$

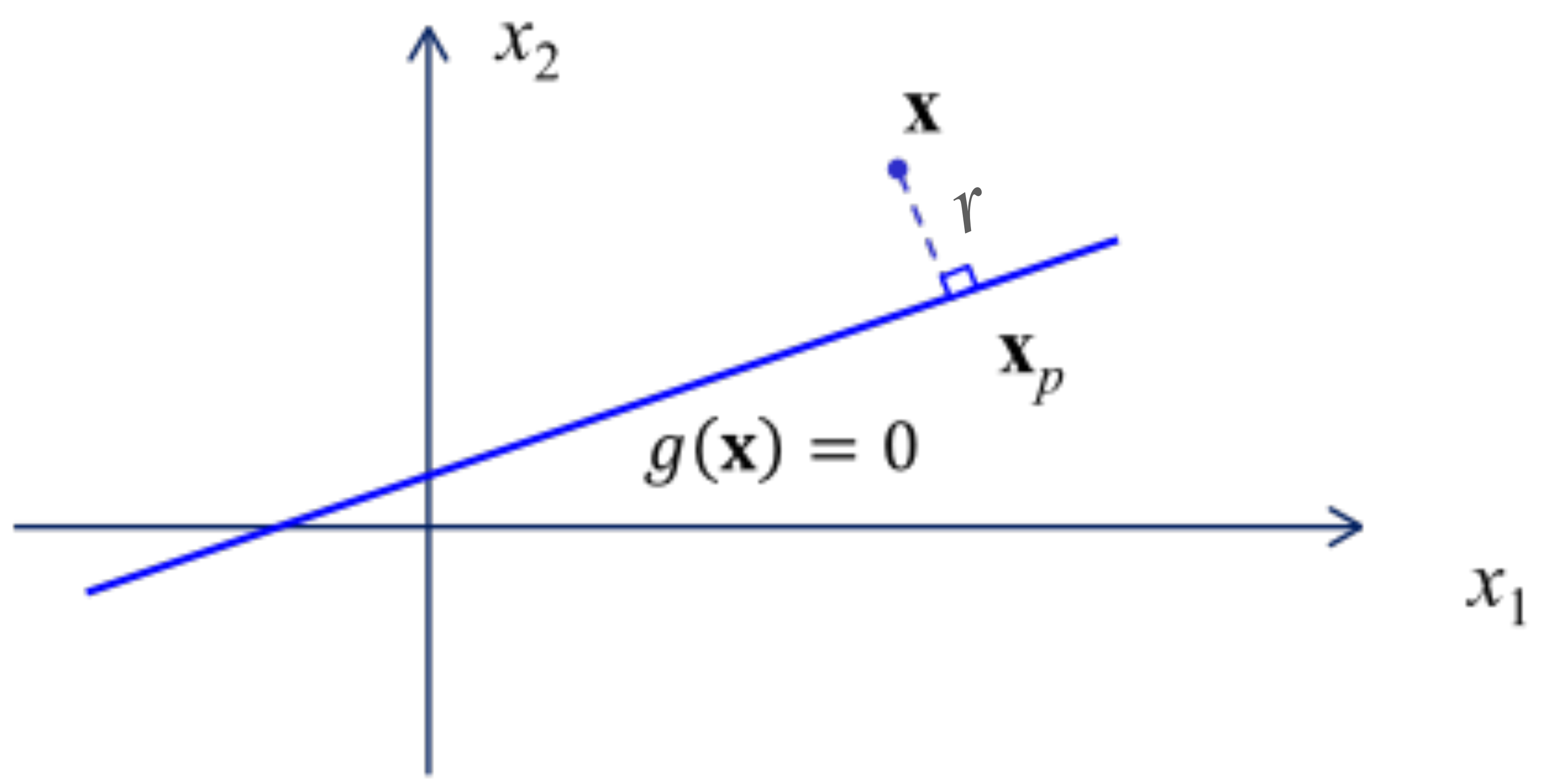


Decision Boundary (cont.)

- Since **x** can be written in terms of its projection and *r*, then the decision boundary can as well:

$$g(\mathbf{x}) = g(\mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||})$$
$$= \mathbf{w}^T(\mathbf{x}_p + r\frac{\mathbf{w}}{||\mathbf{w}||}) + b$$
$$= \mathbf{w}^T\mathbf{x}_p + b + r||\mathbf{w}||$$
$$= r||\mathbf{w}||$$

$$g(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0$$

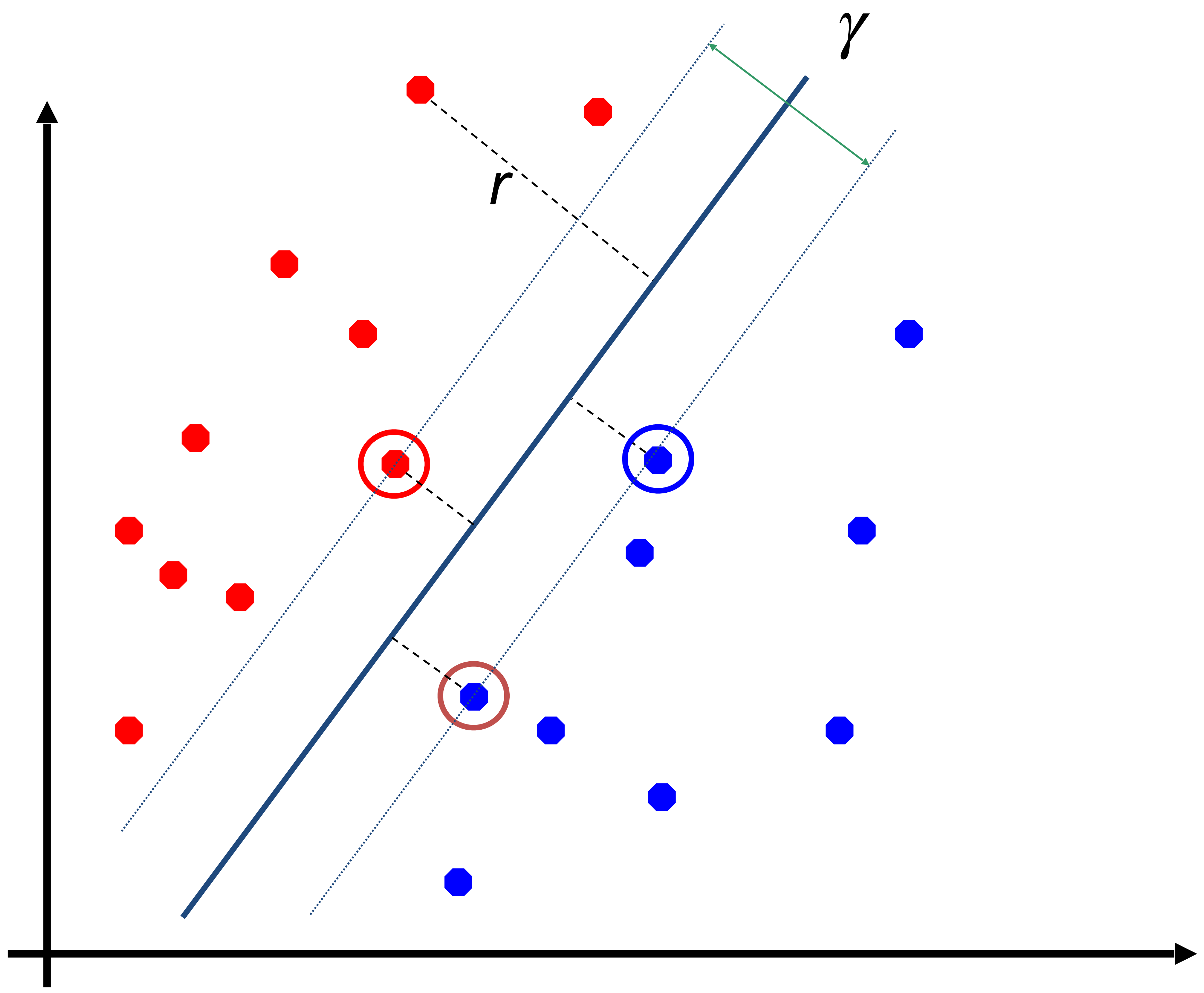


- Thus, $r = \dfrac{g(\mathrm{x})}{||\mathbf{w}||}$

As a special case, for the origin, r = b/||w||

Margins

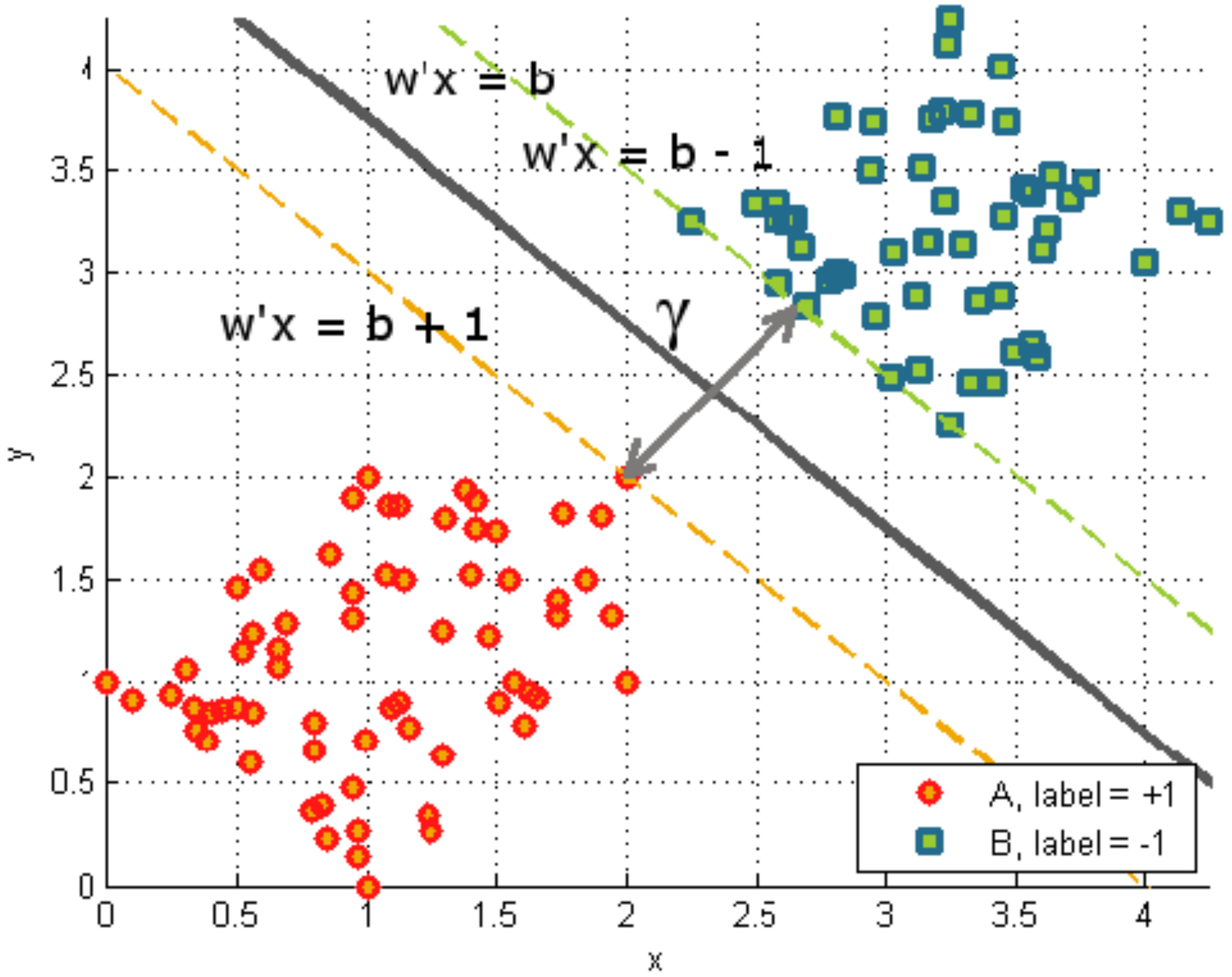
- Distance from example $\mathbf{x}_i$ to the separator is $r = \frac{\mathbf{w}^T \mathbf{x}_i + b}{\|\mathbf{w}\|}$
- Examples closest to the hyperplane are ***support vectors***.
- ***Margin*** γ of the separator is the distance between support vectors.



Notation

- We denote the classifier, $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$, for all $\mathbf{x} \in \mathbb{R}^n$
- **Assumptions**
  - Supporting hyperplanes $\mathbf{w}^T\mathbf{x} + b = \pm 1$
  - The distance between the two supporting hyperplanes is the margin, which is $\gamma = 2$
- To achieve **scale invariance** we divide the classifier by $||\mathbf{w}||_2$. Then the supporting hyperplanes are $\mathbf{w}^T\mathbf{x} + b = \pm \dfrac{1}{||\mathbf{w}||_2}$

margin is $\gamma = \dfrac{2}{||\mathbf{w}||_2}$.



Max margin Classifier

- Given a **linearly separable** training set $S = \{(\mathbf{x}^{(i)}, y^{(i)}) : i = 1, \cdots, N\}$, we would like to find a classifier with a maximum margin, γ

- This can be represented as an optimization problem.

$$\max_{\mathbf{w}, b, \gamma} \gamma \quad \text{subject to: } y^{(i)} \frac{(\mathbf{w}^T \mathbf{x}^{(i)} + b)}{||\mathbf{w}||} \geq \gamma, \quad i = 1, \cdots, N$$

Constraint ensures accurate classification

Nasty optimization problem! Let's make it look nicer!

- Let $\gamma' = \gamma ||\mathbf{w}||$, this is equivalent to

$$\max_{\mathbf{w}, b, \gamma'} \frac{\gamma'}{||\mathbf{w}||} \quad \text{subject to: } y^{(i)}(\mathbf{w}^T \mathbf{x}^{(i)} + b) \geq \gamma', \quad i = 1, \cdots, N$$

Max margin Classifier

- Note that rescaling $\mathbf{w}$ and b by $(1/\gamma')$ will not change the classifier, we can thus further reformulate the optimization problem

$$\max_{\mathbf{w},b,\gamma'} \frac{\gamma'}{||\mathbf{w}||} \quad \text{subject to: } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq \gamma', \quad i = 1,\cdots,N$$

$$\Downarrow$$

$$\max_{\mathbf{w},b} \frac{1}{||\mathbf{w}||} \quad \text{subject to: } y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \geq 1, \quad i = 1,\cdots,N$$

- Note that maximizing the geometric margin is equivalent to minimizing the magnitude of $\mathbf{w}$ subject to maintaining a functional margin of at least 1

Solving the problem

$$\boxed{\begin{array}{l} \displaystyle\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \\ \text{subject to}: \; y^i(\mathbf{w}\cdot\mathbf{x}^i + b) \geq 1, \quad i = 1, \cdots, N \end{array}}$$

- This results in a ***quadratic optimization problem*** with *linear inequality constraints*.
- This is a well-known class of mathematical programming problems for which several (non-trivial) algorithms exist.
    - One could solve for $\mathbf{w}$ using any of these methods
- We will see that it is useful to first formulate an equivalent dual optimization problem and solve it instead
    - This requires a bit of machinery.

Constrained Optimization

- The general optimization problem can be written as such, for generic functions

$$\min_x f(x) \quad \text{subject to: } g_i(x) \leq 0, \quad i = 1, \cdots, m$$

- To solve the above optimization problem, ***consider the following cost function known as the Lagrangian***

$$\mathscr{L}(x, \alpha) = f(x) + \sum_i \alpha_i g_i(x)$$

- Under certain conditions it can be shown that for a solution x' to the above problem, we have

$$f(x') = \underbrace{\min_x \max_\alpha \mathscr{L}(x, \alpha)}_{\text{Primal Form}} = \underbrace{\max_\alpha \min_x \mathscr{L}(x, \alpha)}_{\text{Dual Form subject to } \alpha_i \geq 0}$$

Dual Problem

- After simplifying the inequality, the problem becomes:

$$\min \frac{1}{2}||\mathbf{w}||^2 \quad \text{subject to: } 1 - y^{(i)}(\mathbf{w}^T\mathbf{x}^{(i)} + b) \leq 0, \quad i = 1,\cdots,N$$

- ***The Lagrangian is then***

$$\mathscr{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{i=1}^{N} \alpha_i \left[1 - y^{(i)}(\mathbf{w}^T\mathbf{x} + b)\right], \text{ subject to } \alpha_i \geq 0$$

- We want to solve $\max_{\alpha} \min_{x} \mathscr{L}(\mathbf{w}, b, \alpha)$ s.t. $\alpha_i \geq 0$

- ***Setting the gradient of $\mathscr{L}$ w.r.t. $\mathbf{w}$ and b to zero, we have***

$$\mathbf{w} - \sum_{i=1}^{N} \alpha_i y^i \mathbf{x}^i = 0 \qquad \mathbf{w} = \sum_{i=1}^{N} \alpha_i y^i \mathbf{x}^i$$

$$\sum_{i=1}^{N} \alpha_i y^i = 0$$

Dual Problem

- If we substitute $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y^i \mathbf{x}^i$ in $\mathscr{L}$, we have

$$
\begin{aligned}
L(\boldsymbol{\alpha}) &= \frac{1}{2}\mathbf{w}\cdot\mathbf{w} - \sum_{i=1}^{N}\alpha_i\{y^i(\mathbf{w}\cdot\mathbf{x}^i + b) - 1\} \\
&= \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^i y^j < \mathbf{x}^i\cdot\mathbf{x}^j > - \sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^i y^j < \mathbf{x}^i\cdot\mathbf{x}^j > - b\sum_{i=1}^{N}\alpha_i y^i + \sum_{i=1}^{N}\alpha_i \\
&= \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i\alpha_j y^i y^j < \mathbf{x}^i\cdot\mathbf{x}^j >
\end{aligned}
$$

- Note that $\sum_{i=1}^{N} \alpha_i y^i = 0$
- This is a function of α_i only

Dual Problem

- The new objective function is in terms of α_i only. It is known as the ***dual problem***: if we know all α_i, then we know $\mathbf{w}$
    - The original problem is known as at the primal problem
- **The objective function of the dual problem needs to be maximized!**

$$\max L(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^i y^j < \mathbf{x}^i \cdot \mathbf{x}^j >$$

$$\text{subject to} \quad \alpha_i \geq 0, i = 1, \ldots, n, \qquad \sum_{i=1}^{N} \alpha_i y^i = 0$$

Properties of α_i when we introduce the Lagrange multipliers

The result when we differentiate the original Lagrangian w.r.t. b

Dual Problem

$$\max L(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y^i y^j < \mathbf{x}^i \cdot \mathbf{x}^j >$$

$$\text{subject to} \quad \alpha_i \geq 0, i = 1, \ldots, n, \qquad \sum_{i=1}^{N} \alpha_i y^i = 0$$

- This is also a ***quadratic programming (QP) problem***. A global maximum of α_i can always be found using a QP solver (beyond scope of this class). Luckily, SVMs have been implemented within many platforms.
- $\mathbf{w}$ can be recovered by $\mathbf{w} = \sum_{i=1}^{N} \alpha_i y^i \mathbf{x}^i$
- b can also be recovered as well (just a minute)

Characteristics of the Solution

- Many of the α_i are zero. $\mathbf{w}$ is a linear combination of only a small number of data points
- In fact, optimization theory requires that the solution to satisfy the following KKT conditions:

$$\alpha_i \geq 0, i = 1, \ldots, n,$$

$$y^i \left(\sum_{j=1}^{N} \alpha_i y^j < \mathbf{x}^j \cdot \mathbf{x}^i > + \mathrm{b}\right) \geq 1$$

Functional margin ≥ 1

$$\alpha_i \{ y^i \left(\sum_{j=1}^{N} \alpha_i y^j < \mathbf{x}^j \cdot \mathbf{x}^i > + \mathrm{b}\right) - 1 \} = 0$$

α_i is nonzero only when functional margin = 1

- **$\mathbf{x}_i$ with non-zero α_i are called support vectors (SV)**
  - Let $t_j, (j = 1, \cdots, s)$ be the indices of the s support vectors. We can write $\mathbf{w} = \sum_{j=1}^{s} \alpha_{t_j} y^{t_j} \mathbf{x}^{t_j}$
  - The decision boundary is determined only by the SV

Solving for b, the bias

- Note that we know that for support vectors the functional margin = 1
- We can use this information to solve for b
- We can use any support vector to achieve this

$$y^i \left(\sum_{j=1}^{s} \alpha_{t_j} y^{t_j} < \mathbf{x}^{t_j} \cdot \mathbf{x}^i > + \mathrm{b}\right) = 1$$

- A numerically more stable solution is to use all support vectors (see a ML textbook)

Classifying new examples

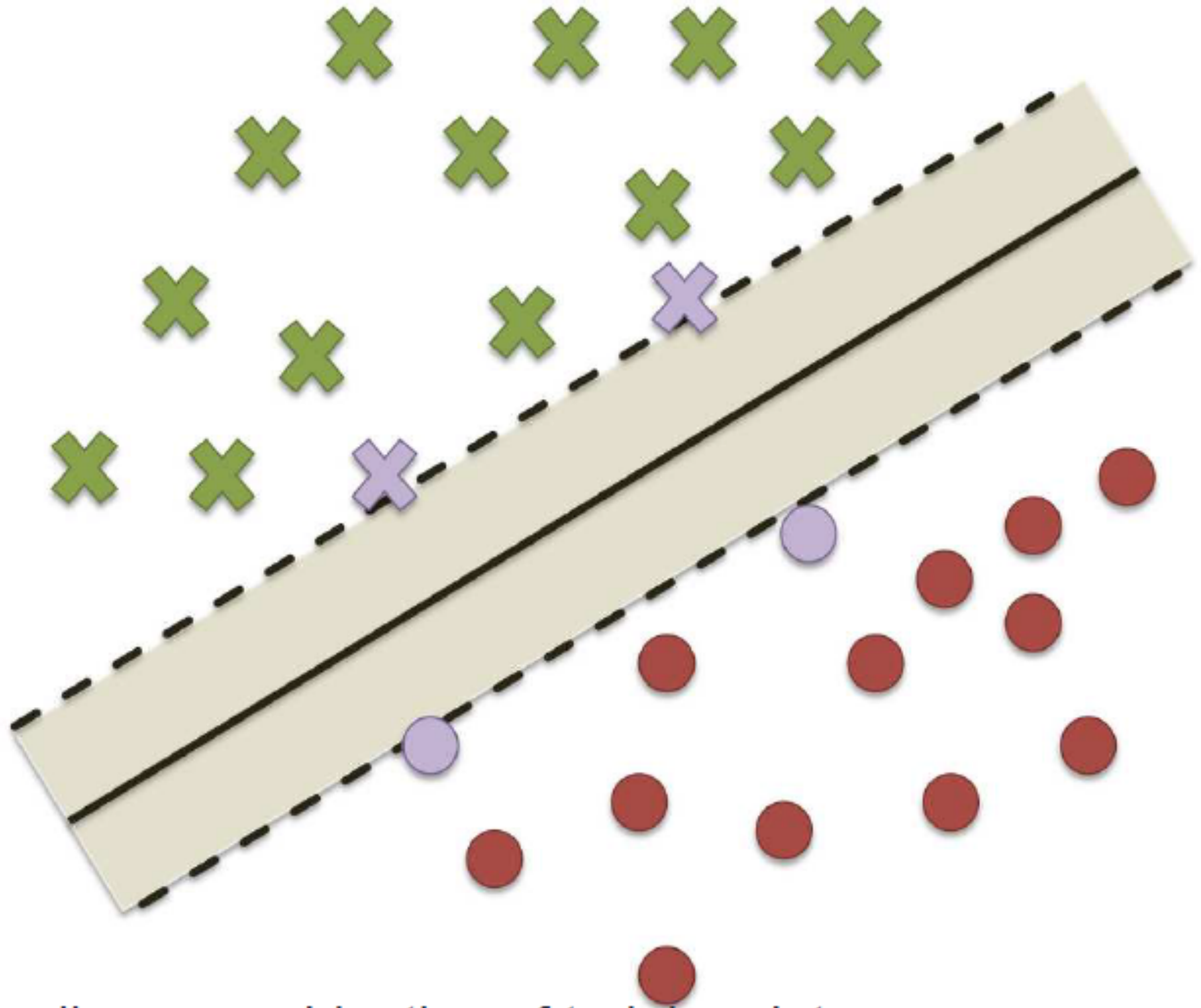
Represents inner product between the two (e.g. support vector and input)

- For classifying with a new input $\mathbf{z}$
    - Compute $\mathbf{w}^T\mathbf{z} + b = \sum_{j=1}^{2} \alpha_{t_j} y^{t_j} < \mathbf{x}^{t_j} \cdot \mathbf{z} > + b$
    - Classify $\mathbf{z}$ as positive if the sum is positive, and negative otherwise
- Note: $\mathbf{w}$ need not be formed explicitly, rather we can classify $\mathbf{z}$ by taking a weighted sum of the inner products with the support vectors
    - This is useful when we generalize from inner product to kernel functions later

Support vectors

- Only points, $\mathbf{x}_i$, that lie on the supporting hyperplanes have $\alpha_i > 0$. These are called the ***support vectors***. Complexity of the solution only depends on the number of support vectors

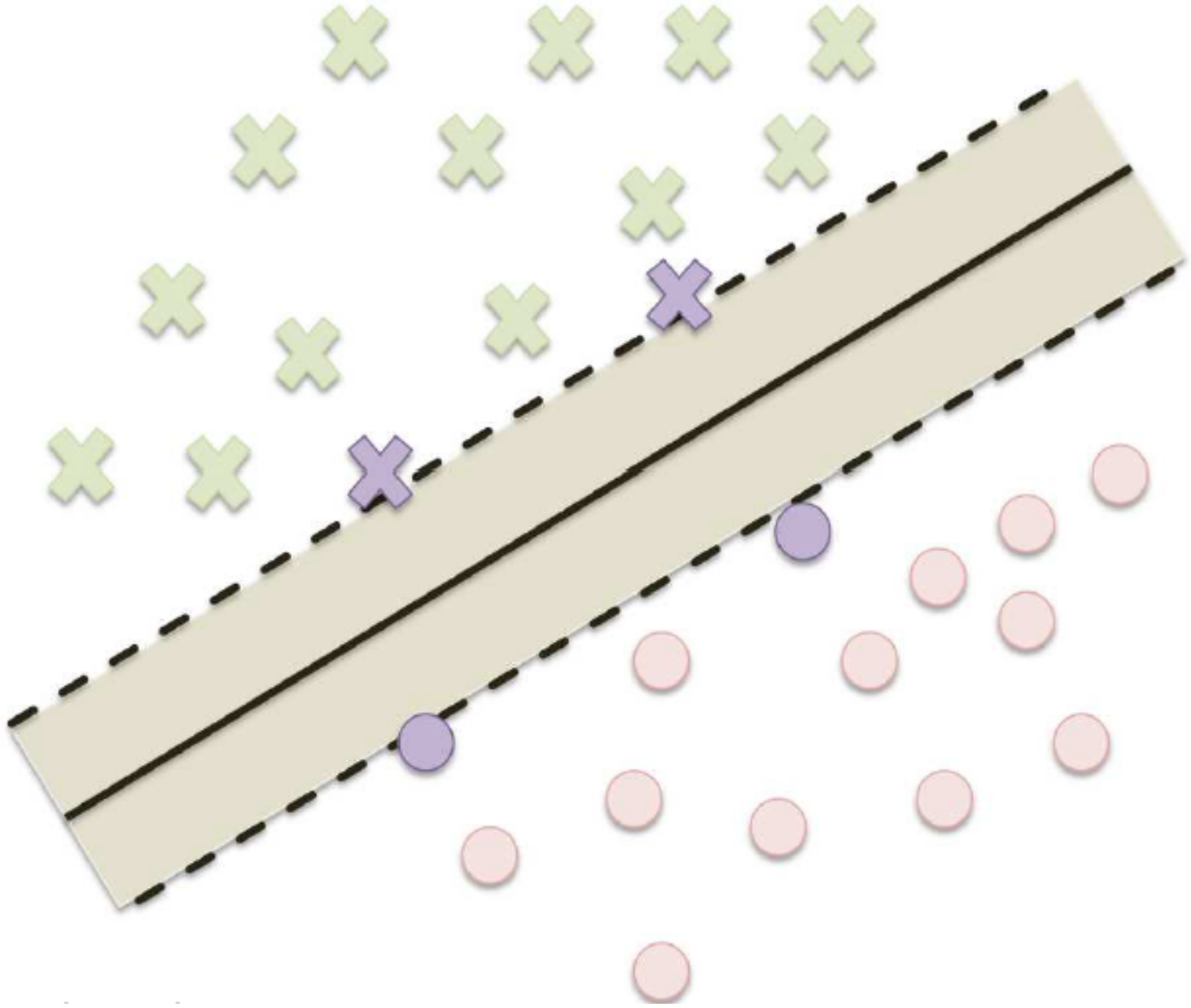
$$\mathbf{w} = \sum_{i=1}^{N} y_i \alpha_i \mathbf{x}_i = \sum_{\text{support vectors}} y_i \alpha_i \mathbf{x}_i$$



Recall that **w** is a linear combination of training data

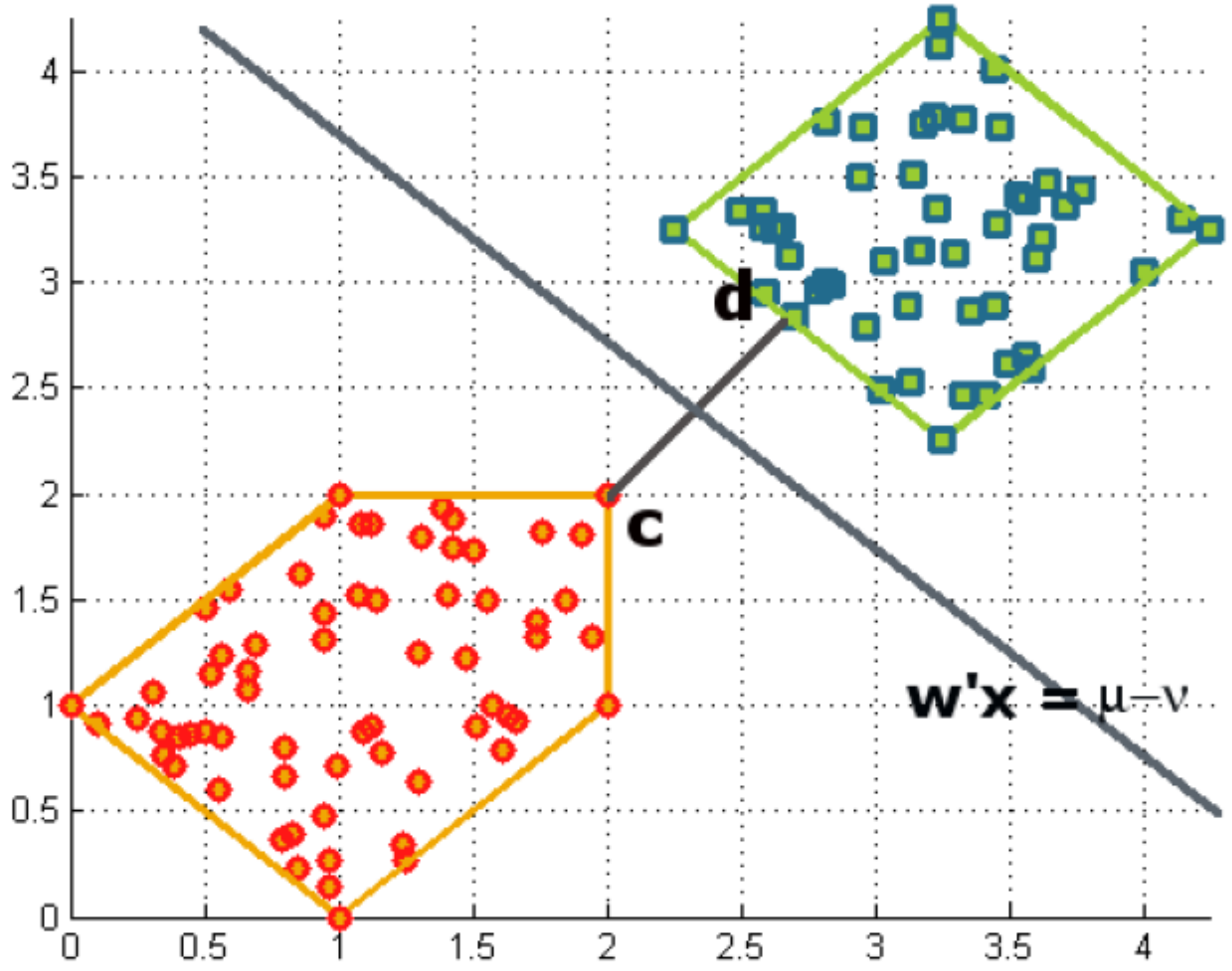
Support Vectors

- Learned model will not change if we delete all the data (e.g., only the support vectors matter)

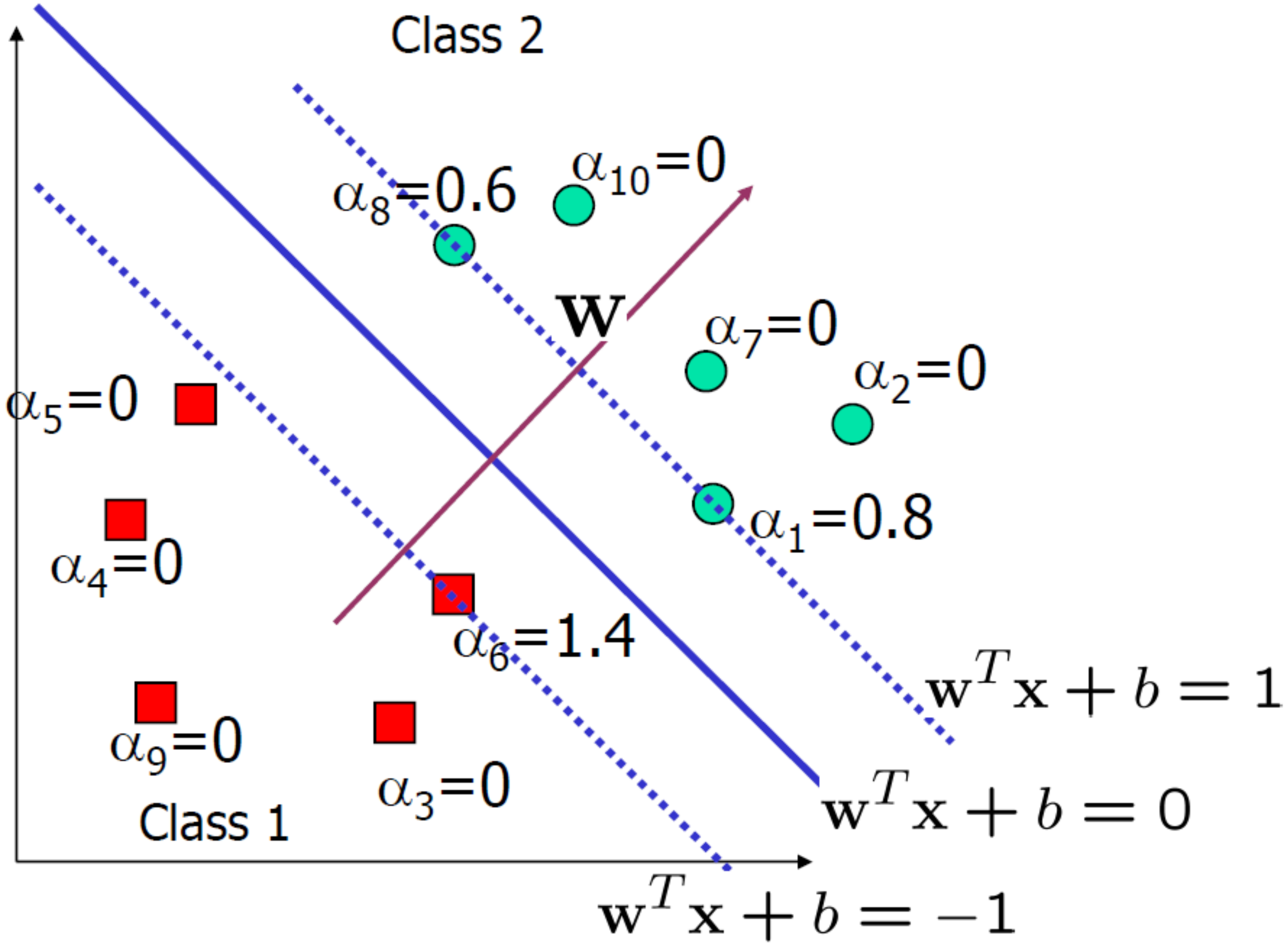


Geometric Perspective

- Maximizing margin is equivalent to ***maximizing the distance between the two closet points on the convex hulls of the two sets***



Geometric Perspective (2)



Summary

- We demonstrated that we prefer to have linear classifiers with large margin.
- We formulated the problem of finding the maximum margin linear classifier as a quadratic optimization problem
- This problem can be solved by solving its dual problem, and efficient QP algorithms are available.
- Problem solved?
- How about non-linear data? – Kernels
- How about noise? – Soft Margin SVMs

Next Class

Support Vector Machines II: soften assumptions