10-315 INTRODUCTION TO MACHINE LEARNING (SCS MAJORS)
LECTURE 2: THE PERCEPTRON

LEILA WEHBE
CARNEGIE MELLON UNIVERSITY
MACHINE LEARNING DEPARTMENT

Lecture based on **chapter 4** from Hal Daumé III, on Kilian Weinberger's **lecture 3**, on Tom Mitchell's **lecture 1** and Matt Gormley's **lecture 1**.

LECTURE OUTCOMES

- Definition of linear separator
- Perceptron algorithm
- Perceptron algorithm guarantees
- Definition of margin

LINKS (USE THE VERSION YOU NEED)

- **Notebook**
- **PDF slides**

SUPERVISED LEARNING PROBLEM STATEMENT

The goal is to learn a function c^* that maps input variables X to output variables y, based on a set of labeled training examples.

- classification: y is binary or multiclass
- regression: y is continuous

Training Data: Given a training set of n labeled examples:$\{(X_1, y_1), (X_2, y_2), \ldots, (X_n, y_n)\}$, where $X_i \in \mathcal{X}$ represents the input features and $y_i \in \mathcal{Y}$ represents the corresponding labels, the goal is to estimate the optimal function c^* that best predicts the labels for new, unseen data.

Hypothesis Space: The function c^* is chosen from a family of hypotheses $\mathcal{H}$. That is, $c^* \in \mathcal{H}$, where $\mathcal{H}$ represents the set of all possible functions that could map inputs to outputs.

Learning Rule: A learning rule is applied to select the optimal function c^* from the hypothesis space $\mathcal{H}$. The learning rule is typically defined based on an optimization algorithm that seeks to minimize a cost function over the training data.

SUPERVISED LEARNING PROBLEM STATEMENT (CONTINUED)

Loss Function: A loss function $L(y, \hat{y})$ quantifies the error between the predicted value $\hat{y} = c(X)$ and the actual value y for a single data point.

- Examples of Loss Functions
  - **0-1 Loss (for Classification)**: The 0-1 loss function is used in classification tasks and is defined for. single point as:

    $$L(y, \hat{y}) = \begin{cases} 0, & \text{if } y = \hat{y} \\ 1, & \text{if } y \neq \hat{y} \end{cases}$$

    This loss function counts the number of incorrect predictions, and the goal is to minimize the number of errors.
  - The loss over a dataset (also refered to as the error rate): $\frac{1}{n} \sum_{i=1}^{n} L(y_i, \hat{y}_i)$

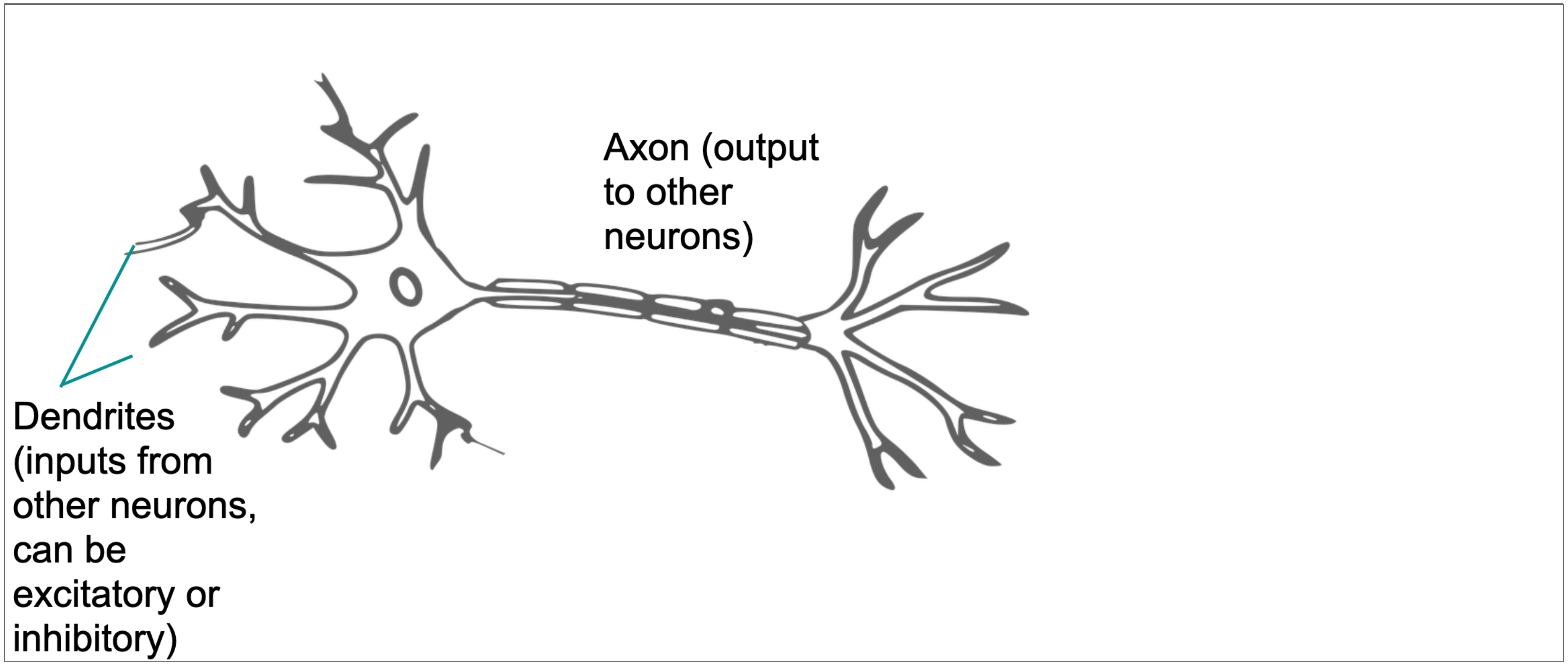
Test Data: A set of m labeled examples:$\{(X_j, y_j), j \in 1 \ldots m\}$, which is sampled from the same distribution as the training set.

CLASS OUTLINE:

- Supervised learning:
    - Perceptron

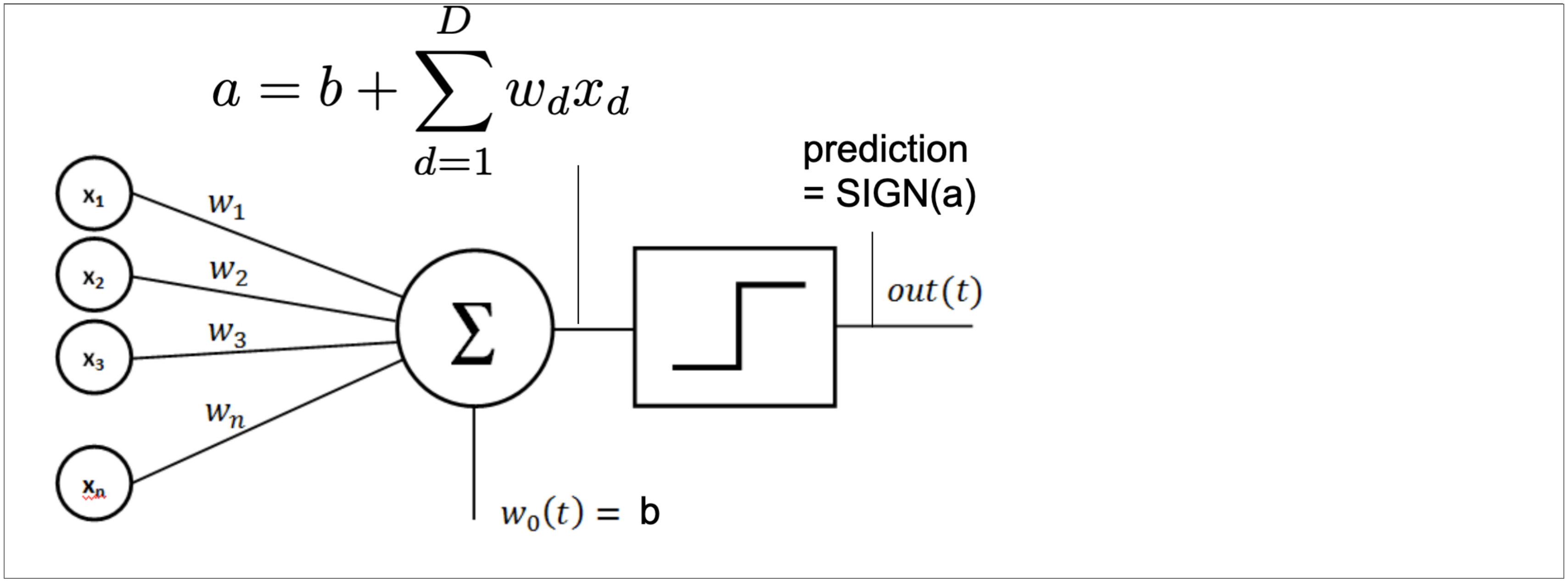
THE PERCEPTRON

- Introduced by Rosenblatt in 1958
- Inspired by real neurons



THE PERCEPTRON

- Introduced by Rosenblatt in 1958
- Inspired by real neurons



THE PERCEPTRON

- Assume data is binary
- Assume data is linearly separable:
    - there exist a hyperplane that perfectly divides the two classes

REFRESHER:

- Recall how to define a hyperplane:
    - a subspace whose dimension is one less than that of its ambient space
    - if x is d-dimensional:
        - $\langle \mathbf{x}, w \rangle + b = 0$
        - (w also d-dimensional)

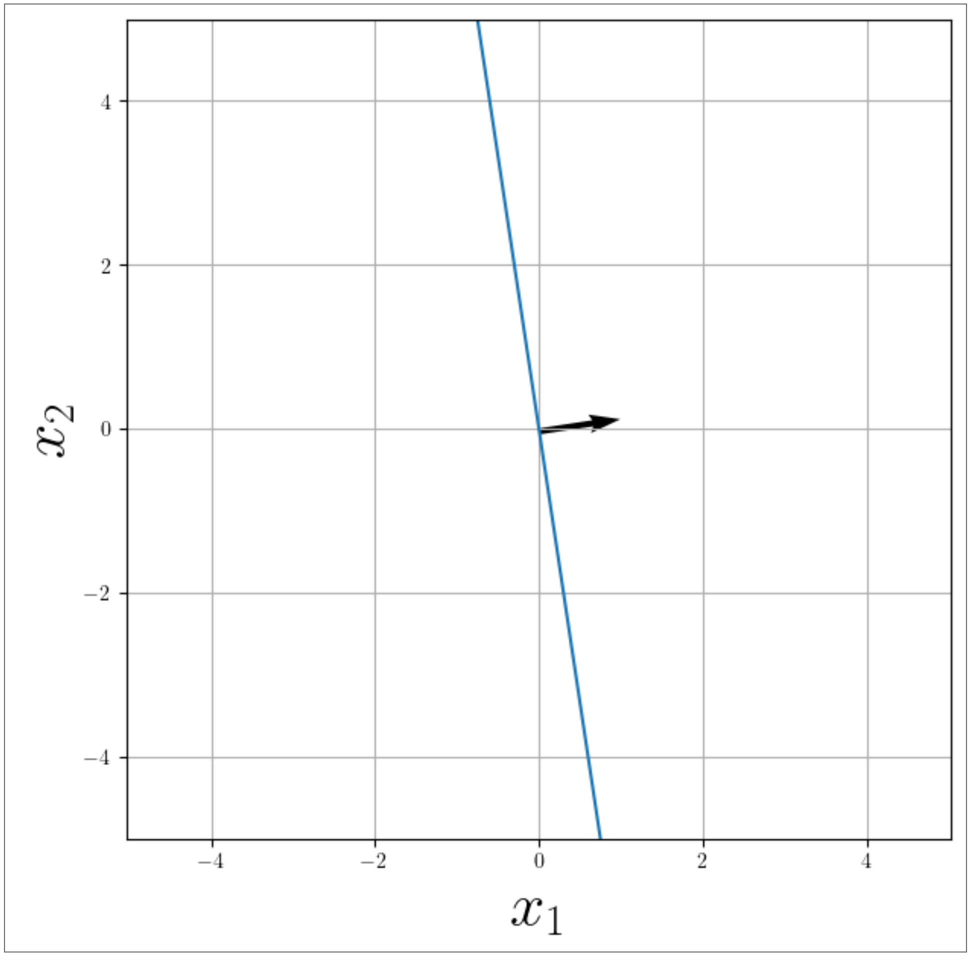
In [1]:

```
import matplotlib.pyplot as plt
%matplotlib inline
import numpy as np
plt.rcParams['text.usetex'] = True
import warnings
warnings.filterwarnings('ignore')
```

In [7]:

```
def plot_line(ax,xlims, w, do_norm=True):
    x1 = np.linspace(xlims[0],xlims[1],1000)
    x2_plot = (- w[2] - w[0]*x1)/w[1]  #  w[0]*x1 + w[1]*x2 + w[2] = 0
    ax.plot(x1,x2_plot)
    origin = x1[np.array(x1.shape[0]/2).astype(int)], x2_plot[int(x1.shape[0]/2)]
    nn = np.linalg.norm(w) if do_norm else 1
    ax.quiver(*origin, w[0]/nn,w[1]/nn, color='k',angles='xy', scale_units='xy', scale=1)
    ax.axis('equal')
    ax.axis([xlims[0],xlims[1],xlims[0],xlims[1]])
    ax.set_xlabel(r'$x_1$',fontsize=30); ax.set_ylabel(r'$x_2$',fontsize=30)
    ax.grid()

w = [4,0.6]
b = 0
f, ax = plt.subplots(figsize=(7,7))
plot_line(ax, [-5,5], [w[0],w[1],b])
```



4
2
0
−2
−4
−4
−2
0
2
4
x_1
x_2

THE PERCEPTRON

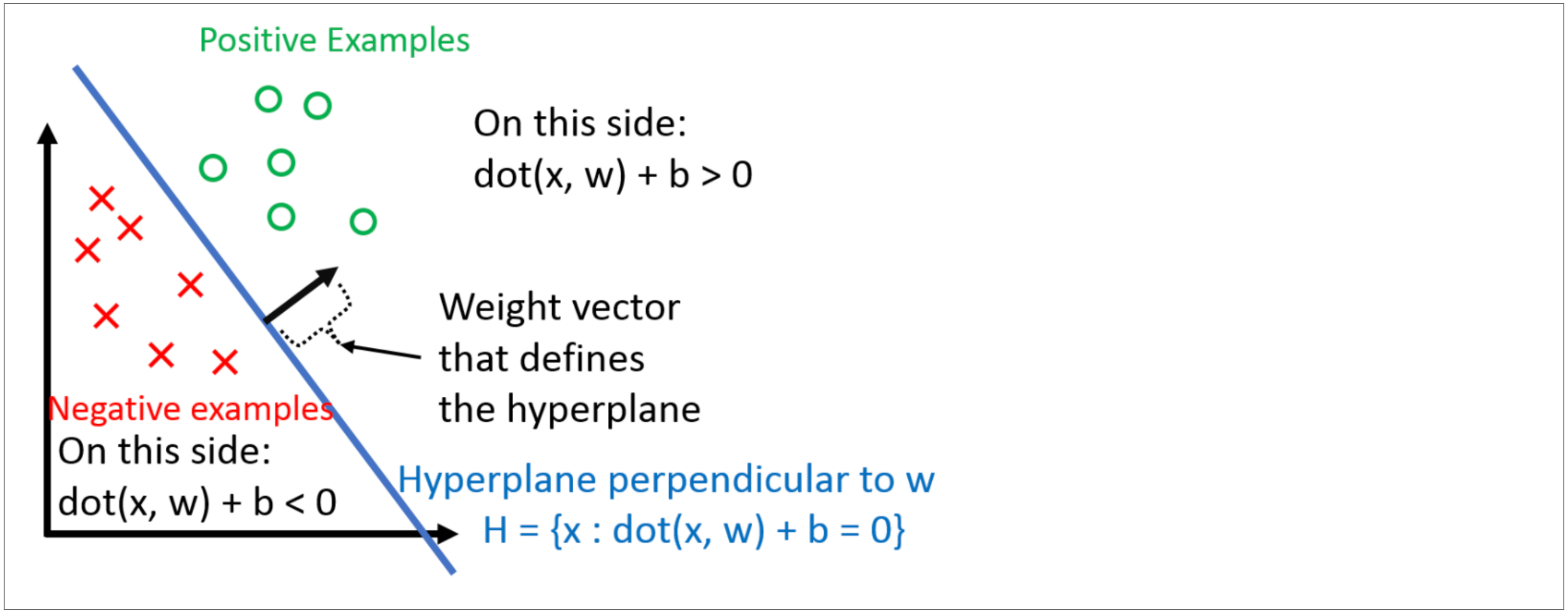
- Assume data is binary
- Assume data is linearly separable:
    - i.e, there exist a hyperplane that perfectly divides the two classes

$$\exists \mathbf{w}, b \quad \text{s.t.} \quad \forall (\mathbf{x}_i, y_i) \in D, \tag{1}$$

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) > 0 \tag{2}$$

$$\exists \mathbf{w}, b \quad \text{s.t.} \quad \forall(\mathbf{x}_i, y_i) \in D, \tag{3}$$

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) > 0 \tag{4}$$



source

WHAT IS A SEPARABLE DATASET?

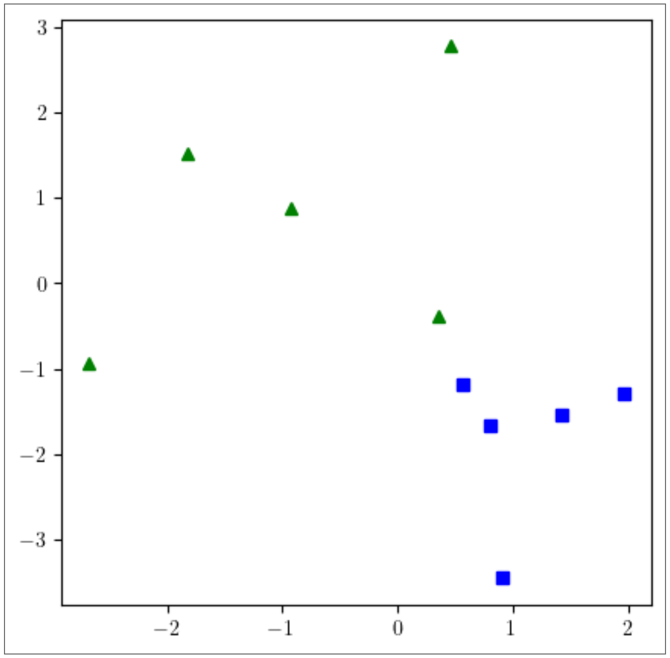
In [25]:

```
from sklearn import datasets

X, y = datasets.make_blobs(n_samples=10, centers=np.array([[-1,1],[1,-1]]),
                           n_features=2, center_box=(0, 10))
y[y==0] = -1

def plot_dataset(ax,X,y):
    ax.plot(X[:, 0][y == -1], X[:, 1][y == -1], 'g^')
    ax.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'bs');

f, ax = plt.subplots(figsize=(5,5))
plot_dataset(ax,X,y)
```


3
2
1
0
−1
−2
−3
−2
−1
0
1
2

SIMPLIFYING w AND b

- We can write $\mathbf{x}_i$ as:

$$\mathbf{x_i}' = \begin{bmatrix} \mathbf{x_i} \\ 1 \end{bmatrix} \tag{5}$$

- and incorporate b into $\mathbf{w}$:

$$\mathbf{w}' = \begin{bmatrix} \mathbf{w} \\ b \end{bmatrix} \tag{6}$$

- The same hyperplane is now defined by $\mathbf{w}'^{\top}\mathbf{x}' = 0$
- Why does this work?
- We will use $\mathbf{w}$ and $\mathbf{x}$ to refer to these vectors in the rest of the lecture.

THE PERCEPTRON TRAINING ALGORITHM

- Initialize $\mathbf{w} = \mathbf{0}$
- while TRUE do
    - m = 0
    - for $(\mathbf{x}_i, y_i) \in D$ do
        - if $y_i(\mathbf{w}^\top \mathbf{x}_i) \leq 0$ then   % if the tuple $(\mathbf{x}_i, y_i)$ is misclassified
            - $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}_i$ %update the weight vector $\mathbf{w}$
            - m = m+1   %count number of misclassified examples in this round
        - end if
    - end for   %if no examples were misclassified in this round
        - break   %break out of the loop
    - end if
- end while

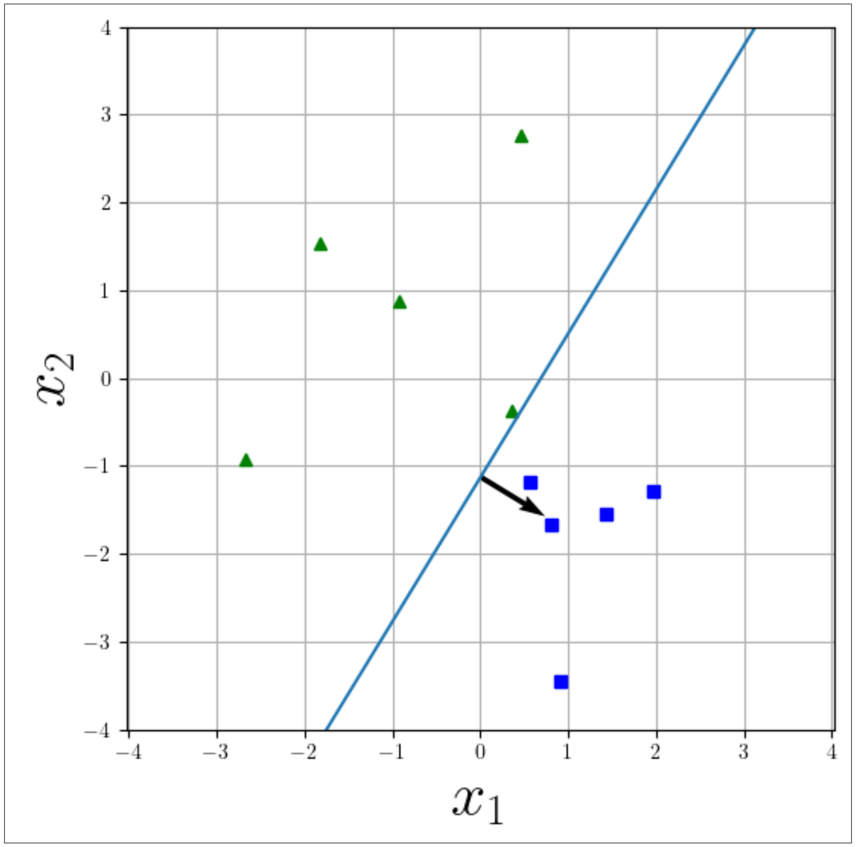
QUESTIONS:

- how does convergence happen?
- what happens if the data is not separable?

In [26]:

```
def perceptron_train(X,y,MaxIter=20):
    w = np.zeros((X.shape[1]))
    for i in range(MaxIter):
        m = 0
        for (xi,yi) in zip(X,y):
            if yi*w.T.dot(xi)<=0:
                w = w + yi*xi
                m = m + 1
        if m==0:
            break
    return w

f,ax = plt.subplots(figsize=(6,6))
plot_dataset(ax,X,y)
Xprime = np.hstack([X,np.ones((X.shape[0],1))])
w = perceptron_train(Xprime,y)
plot_line(ax,[-4,4], w)
```



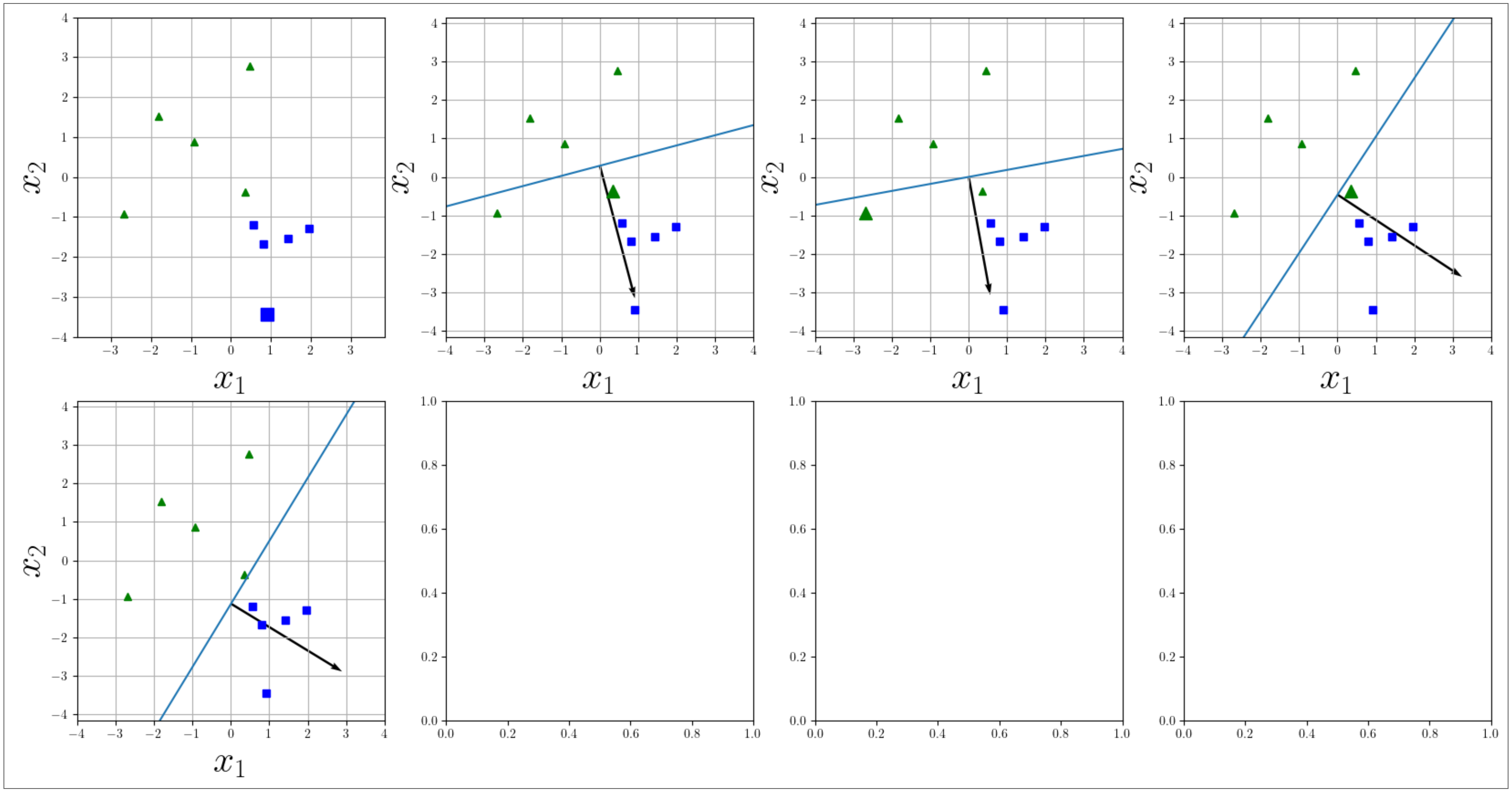
4
3
2
1
0
−1
−2
−3
−4
x_2
−4
−3
−2
−1
0
1
2
3
4
x_1

In [20]:

```python
def perceptron_train_and_plot(axs,X,y,MaxIter=20):
    w = np.zeros((X.shape[1]))
    plot_dataset(axs[0],X,y)
    plot_line(axs[0],[-4,4], w)
    plt_cnt = 0
    for i in range(MaxIter):
        m = 0
        for (xi,yi) in zip(X,y):
            if yi*w.T.dot(xi)<=0:
                w = w + yi*xi
                m = m + 1
                plt_cnt = plt_cnt + 1
                try:
                    if yi == -1:
                        axs[plt_cnt-1].plot([xi[0]],[xi[1]],'g^',markersize=10)
                    else:
                        axs[plt_cnt-1].plot([xi[0]],[xi[1]],'bs',markersize=10)
                    plot_dataset(axs[plt_cnt],X,y)
                    plot_line(axs[plt_cnt],[-4,4], w, do_norm=False)
                except:
                    print('not enough subplots')
        if m==0:
            break
    return w
```

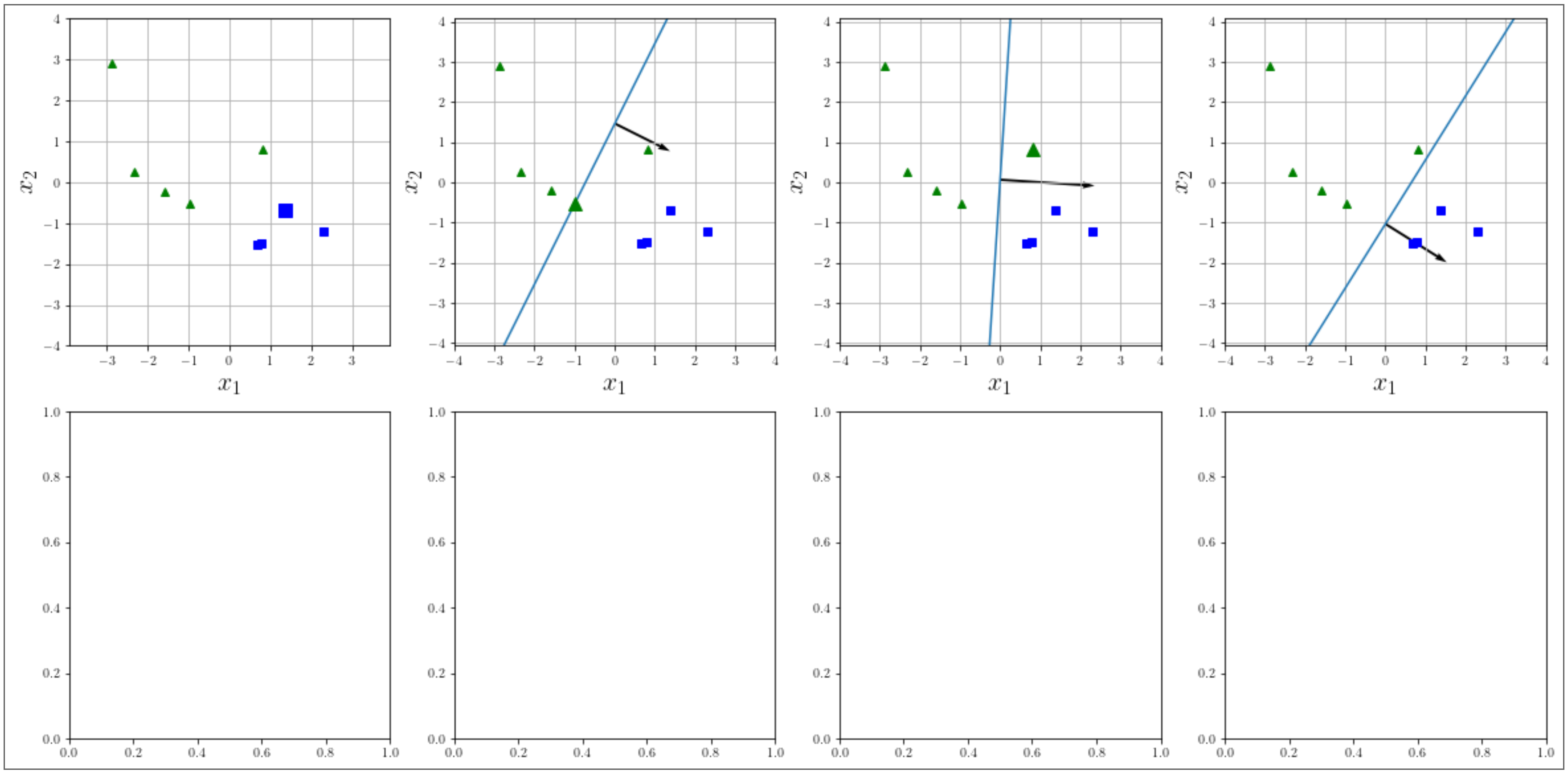
In [27]:

```
f,axs = plt.subplots(nrows=2,ncols=4,figsize=(20,10))
w = perceptron_train_and_plot(axs.reshape(-1),Xprime,y)
```



In [68]:

```
X, y = datasets.make_blobs(n_samples=10, centers=np.array([[-1,1],[1,-1]]),
                           n_features=2, center_box=(0, 10))
y[y==0] = -1
Xprime = np.hstack([X,np.ones((X.shape[0],1))])
f,axs = plt.subplots(nrows=2,ncols=4,figsize=(20,10))
w = perceptron_train_and_plot(axs.reshape(-1),Xprime,y)
```



WHY DOES THIS TRAINING ALGORITHM WORK?

- what happens if you missclassify a positive example?
  - this means $y_i(\mathbf{w^k}^\top \mathbf{x_i}) \leq 0$, and in other words $\mathbf{w^k}^\top \mathbf{x_i} < 0$)
  - the weights get updated:

$$\mathbf{w^{k+1}} = \mathbf{w^k} + \mathbf{x_i}$$

  - what happens to the dot product $(\mathbf{w}^\top \mathbf{x_i})$?

$$\begin{aligned} \mathbf{w^{k+1}}^\top \mathbf{x_i} &= (\mathbf{w^k} + \mathbf{x_i})^\top \mathbf{x_i} && (7) \\ &= \mathbf{w^k}^\top \mathbf{x_i} + \mathbf{x_i}^\top \mathbf{x_i} && (8) \\ &> \mathbf{w^k}^\top \mathbf{x_i} && (9) \end{aligned}$$

- The prediction becomes more positive (vice versa for a negative example).
- Thus the boundary is getting closer to correctly classiying $\mathbf{x_i}$.

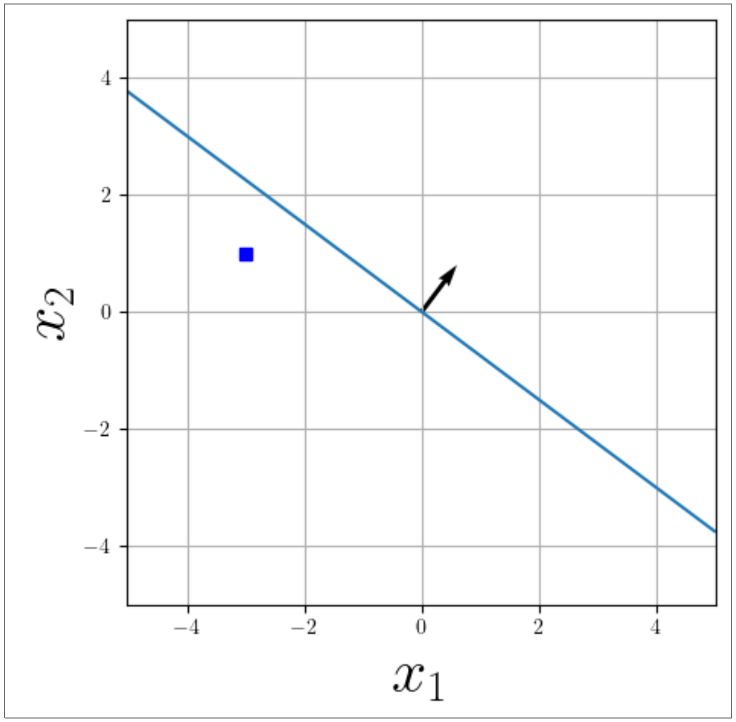
HOW MANY ITERATIONS ARE NEEDED WITH A DATASET WITH ONE EXAMPLE?

In [28]:

```
f, ax = plt.subplots(figsize=(5,5))
plot_line(ax, [-5,5], [3,4,0])
ax.plot([-3],[1],'bs')
```

Out[28]:

```
[<matplotlib.lines.Line2D at 0x1295a6610>]
```

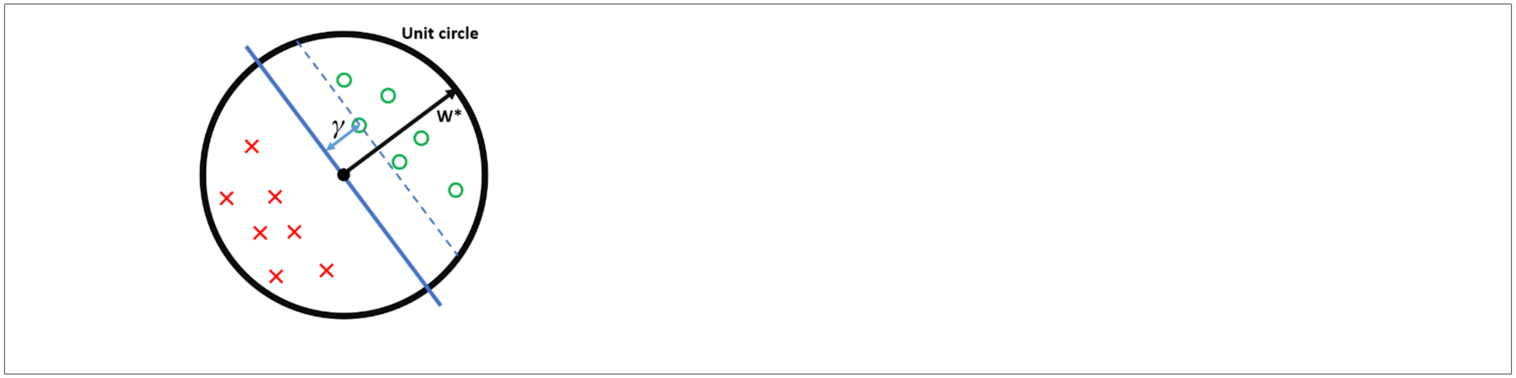


CONVERGENCE OF THE PERCEPTRON ALGORITHM

- The perceptron algorithm converges in $\frac{1}{\gamma^2}$ updates if the data is linearly separable.
- γ is the margin of the problem instance (defined on next slide).

NOTION OF MARGIN

- Assume there exists $\mathbf{w}^*$ such that $\forall(\mathbf{x_i}, y_i) \in D$, $y_i(\mathbf{x_i}^\top \mathbf{w}^*) > 0$
- Also assume we rescale $\mathbf{w}^*$ and the $\mathbf{x_i}$s such that:
  - $||\mathbf{w}^*|| = 1$ and $||\mathbf{x_i}|| \leq 1 \ \forall \mathbf{x_i}$ (how?)
- The margin γ of the hyperplane $\mathbf{w}^*$ is the minimum distance between one of the points and the hyperplane:
  - $\gamma = \min_{(\mathbf{x_i}, y_i) \in \mathcal{D}} |\mathbf{x_i}^\top \mathbf{w}^*|$ (since $\mathbf{w}^*$ is unit norm)



[source](#)

THEOREM

- Given:
    - All $\mathbf{x}_i$s are within the unit sphere
    - There exists a separating hyperplane $\mathbf{w}^*$, with $||\mathbf{w}^*|| = 1$
    - γ is the margin of hyperplane $\mathbf{w}^*$
- If all of the above holds, then the Perceptron algorithm makes at most $\frac{1}{\gamma^2}$ mistakes.

QUESTIONS:

- Is it easier to train when the margin is small or large?
- What types of datasets will converge quickly?

PROOF

source

Keeping what we defined above, consider the effect of an update ($\mathbf{w}$ becomes $\mathbf{w} + y\mathbf{x}$) on the two terms $\mathbf{w}^\top \mathbf{w}^*$ and $\mathbf{w}^\top \mathbf{w}$. We will use two facts:

- $y(\mathbf{x}^\top \mathbf{w}) \leq 0$: This holds because $\mathbf{x}$ is misclassified by $\mathbf{w}$ - otherwise we wouldn't make the update.
- $y(\mathbf{x}^\top \mathbf{w}^*) > 0$: This holds because $\mathbf{w}^*$ is a separating hyper-plane and classifies all points correctly.

1. Consider the effect of an update on $\mathbf{w}^\top \mathbf{w}^*$:

$$(\mathbf{w} + y\mathbf{x})^\top \mathbf{w}^* = \mathbf{w}^\top \mathbf{w}^* + y(\mathbf{x}^\top \mathbf{w}^*) \geq \mathbf{w}^\top \mathbf{w}^* + \gamma$$

The inequality follows from the fact that, for $\mathbf{w}^*$, the distance from the hyperplane defined by $\mathbf{w}^*$ to $\mathbf{x}$ must be at least γ (i.e. $y(\mathbf{x}^\top \mathbf{w}^*) = |\mathbf{x}^\top \mathbf{w}^*| \geq \gamma$).

This means that for each update, $\mathbf{w}^\top \mathbf{w}^*$ grows by **at least** γ.

2. Consider the effect of an update on $\mathbf{w}^\top \mathbf{w}$:

$$(\mathbf{w} + y\mathbf{x})^\top (\mathbf{w} + y\mathbf{x}) = \mathbf{w}^\top \mathbf{w} + \underbrace{2y(\mathbf{w}^\top \mathbf{x})}_{<0} + \underbrace{y^2(\mathbf{x}^\top \mathbf{x})}_{0\leq \ \leq 1} \leq \mathbf{w}^\top \mathbf{w} + 1$$

The inequality follows from the fact that

- $2y(\mathbf{w}^\top \mathbf{x}) < 0$ as we had to make an update, meaning $\mathbf{x}$ was misclassified
- $0 \leq y^2(\mathbf{x}^\top \mathbf{x}) \leq 1$ as $y^2 = 1$ and all $\mathbf{x}^\top \mathbf{x} \leq 1$ (because $\|\mathbf{x}\| \leq 1$).

3. Now we know that after M updates the following two inequalities must hold:

   (1) $\mathbf{w}^\top \mathbf{w}^* \geq M\gamma$

   (2) $\mathbf{w}^\top \mathbf{w} \leq M$.

We can then complete the proof:

$$\begin{aligned}
M\gamma &\leq \mathbf{w}^\top \mathbf{w}^* && \text{By (1)} \\
&= \|\mathbf{w}\| \cos(\theta) && \text{by definition of inner-product, where } \theta \text{ is the angle between } \mathbf{w} \text{ and } \mathbf{w}^*. \\
&\leq ||\mathbf{w}|| && \text{by definition of cos, we must have } \cos(\theta) \leq 1. \\
&= \sqrt{\mathbf{w}^\top \mathbf{w}} && \text{by definition of } \|\mathbf{w}\| \\
&\leq \sqrt{M} && \text{By (2)} \\
& \\
&\Rightarrow M\gamma \leq \sqrt{M} \\
&\Rightarrow M^2\gamma^2 \leq M \\
&\Rightarrow M \leq \frac{1}{\gamma^2} && \text{And hence, the number of updates } M \text{ is bounded from above by a constant.}
\end{aligned}$$

QUESTIONS

- What happens if the data is not separable?
- Does the order matter?
- Is the perceptron guaranteed to find an optimal solution?

ANSWERS:

- the algorithm doens't converge
- different orders of points might lead to faster or slower convergence, and to a different $\mathbf{w}$ and different hyperplane
- no, it is only guaranteed to find a solution if one exists. It does not look for the maximum margin separator.