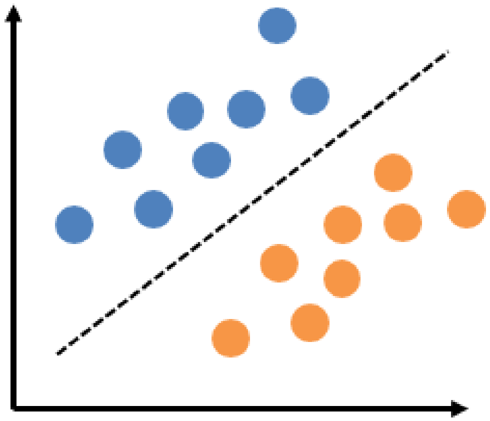
PERCEPTRON ALGORITHM



Ronen Nir & Itay Evron

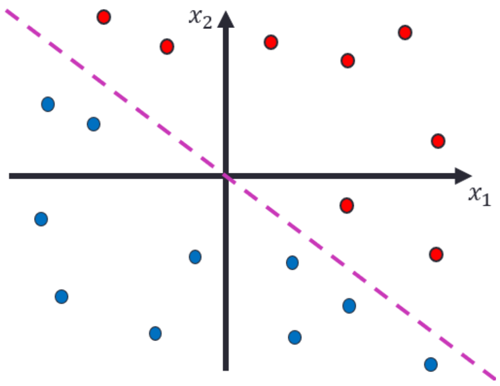
Outline

- The perceptron algorithm
  - Homogeneous vs. non-homogeneous
- Digit recognition demo
- Optimization perspective
  - Subgradients
  - Perceptron as SGD

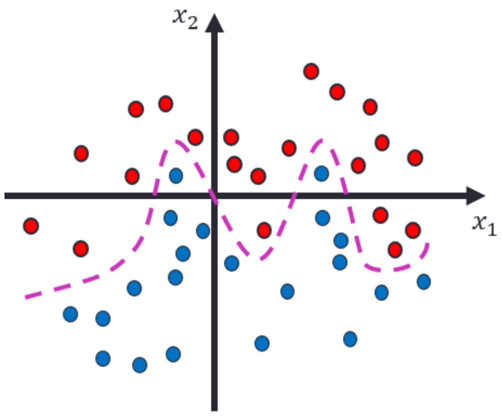
Homogeneous linear models

- Decision rules are $h_{\boldsymbol{w}}(\boldsymbol{x}) = \mathrm{sign}(\boldsymbol{w}^\top \boldsymbol{x})$
- Decision boundaries are linear, indecisive where $\boldsymbol{w}^\top \boldsymbol{x} = 0$

Homogeneously linearly separable



Linearly inseparable



Perceptron – Intuition

- First and simplest linear model
- Trained iteratively:

```
𝒘 = 𝟎_d

while didn't separate trainset
    for i=1 to m
        ŷ_i = sign(𝒘ᵀ𝒙_i)

        if y_i != ŷ_i
            𝒘 = 𝒘 + η y_i 𝒙_i
```

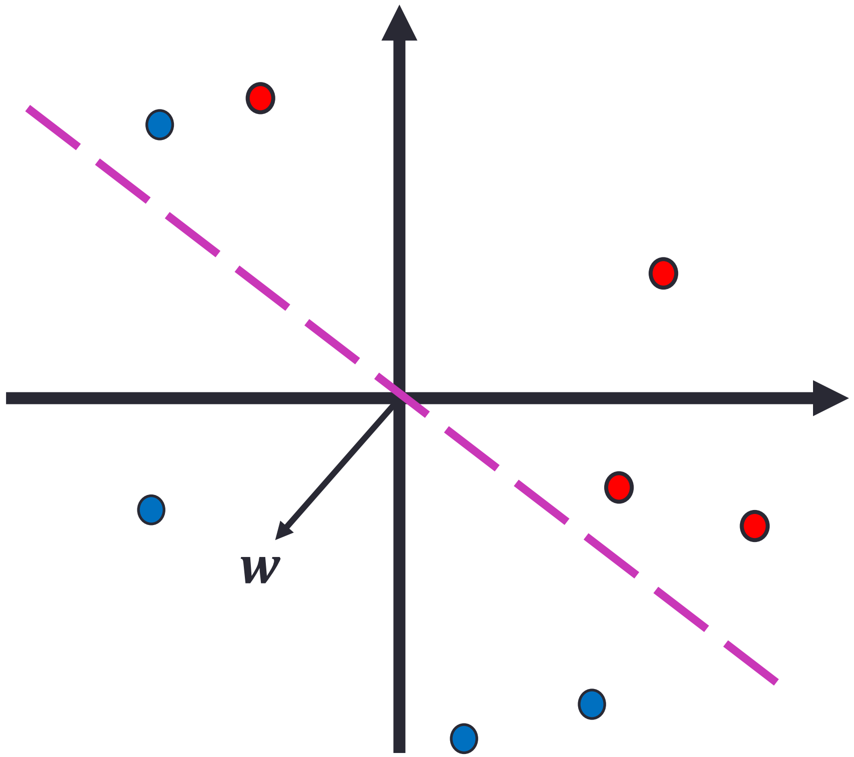
Perceptron – Intuition

- First and simplest linear model
- Trained iteratively:

$\boldsymbol{w} = \boldsymbol{0}_d$

while didn't separate trainset
    for i=1 **to** m
        $\hat{y}_i = \mathrm{sign}(\boldsymbol{w}^\top \boldsymbol{x}_i)$

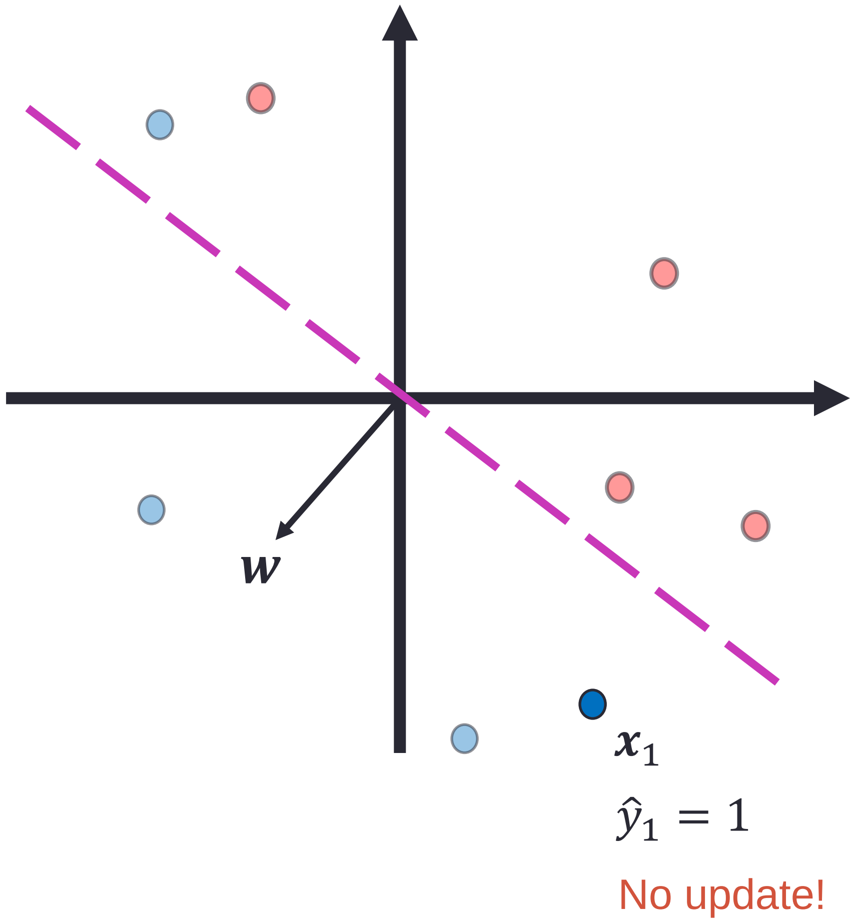
        if $y_i \mathrel{!=} \hat{y}_i$
            $\boldsymbol{w} = \boldsymbol{w} + \eta y_i \boldsymbol{x}_i$



Perceptron – Intuition

- First and simplest linear model
- Trained iteratively:

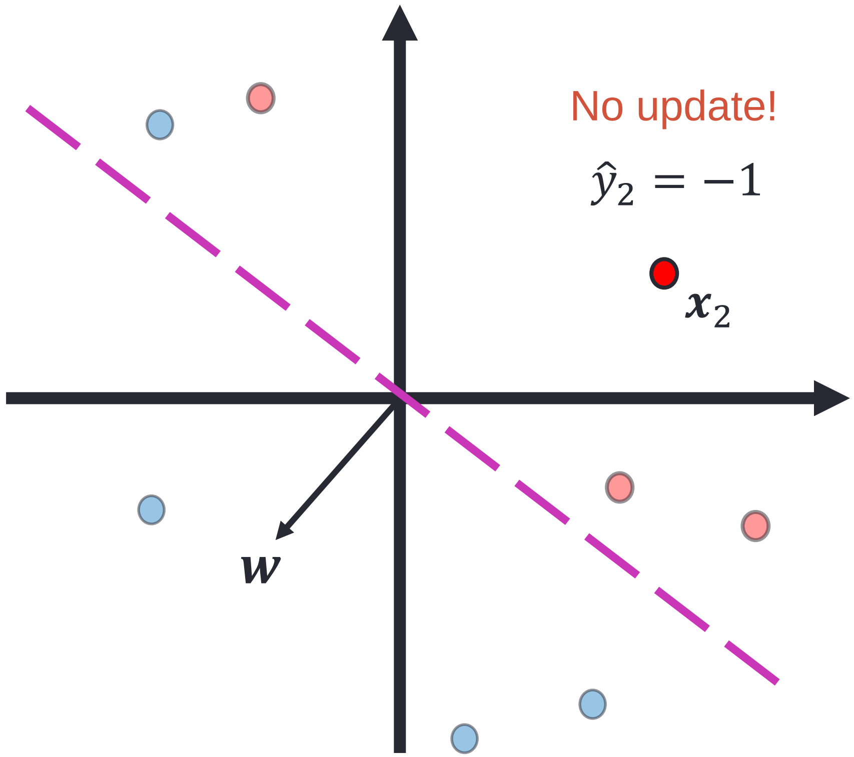
```
w = 0_d

while didn't separate trainset
    for i=1 to m
        ŷ_i = sign(wᵀx_i)

        if y_i != ŷ_i
            w = w + η y_i x_i
```



Perceptron – Intuition

- First and simplest linear model
- Trained iteratively:

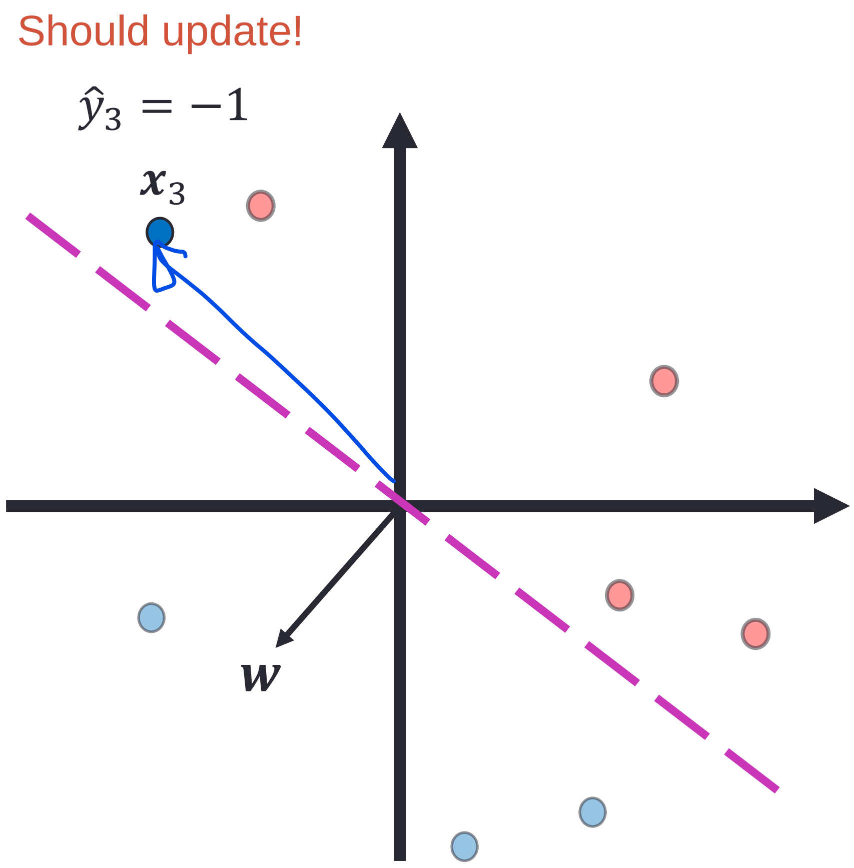
```
w = 0_d

while didn't separate trainset
    for i=1 to m
        ŷ_i = sign(wᵀx_i)

        if y_i != ŷ_i
            w = w + η y_i x_i
```



Perceptron – Intuition

- First and simplest linear model
- Trained iteratively:

```
w = 0_d

while didn't separate trainset
    for i=1 to m
        ŷ_i = sign(wᵀx_i)

        if y_i != ŷ_i
            w = w + ηy_i x_i
```



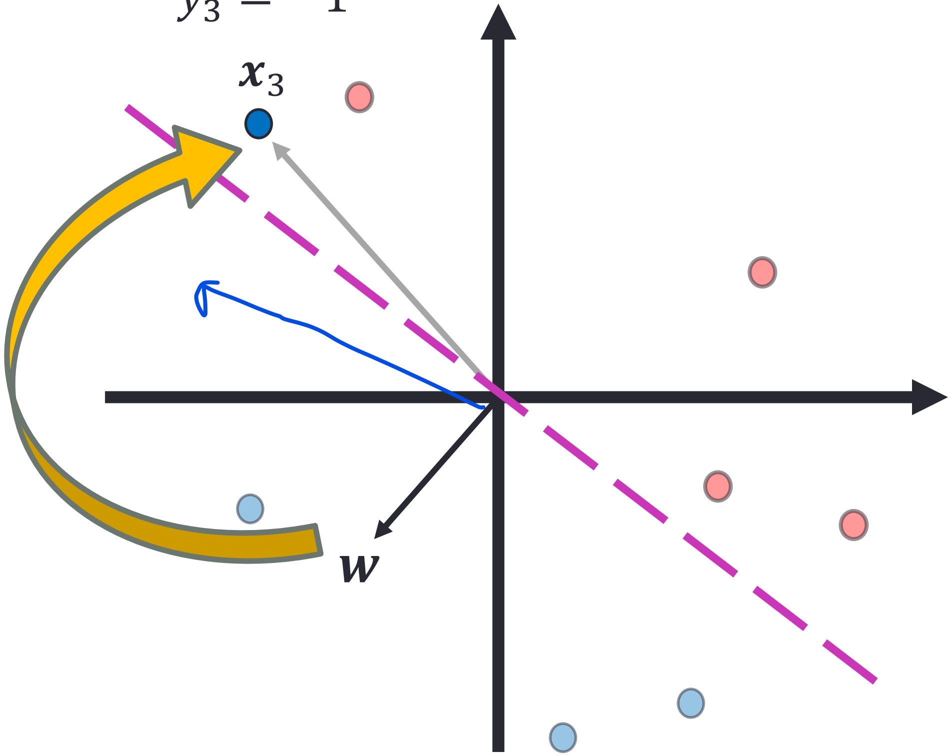
Perceptron – Intuition

- First and simplest linear model
- Trained iteratively:

```
w = 0_d

while didn't separate trainset
    for i=1 to m
        ŷ_i = sign(wᵀx_i)

        if y_i != ŷ_i
            w = w + ηy_ix_i
```

Should update!

$\hat{y}_3 = -1$



We want to move $\boldsymbol{w}$
in the direction of $\boldsymbol{x}_3$

Perceptron – Intuition

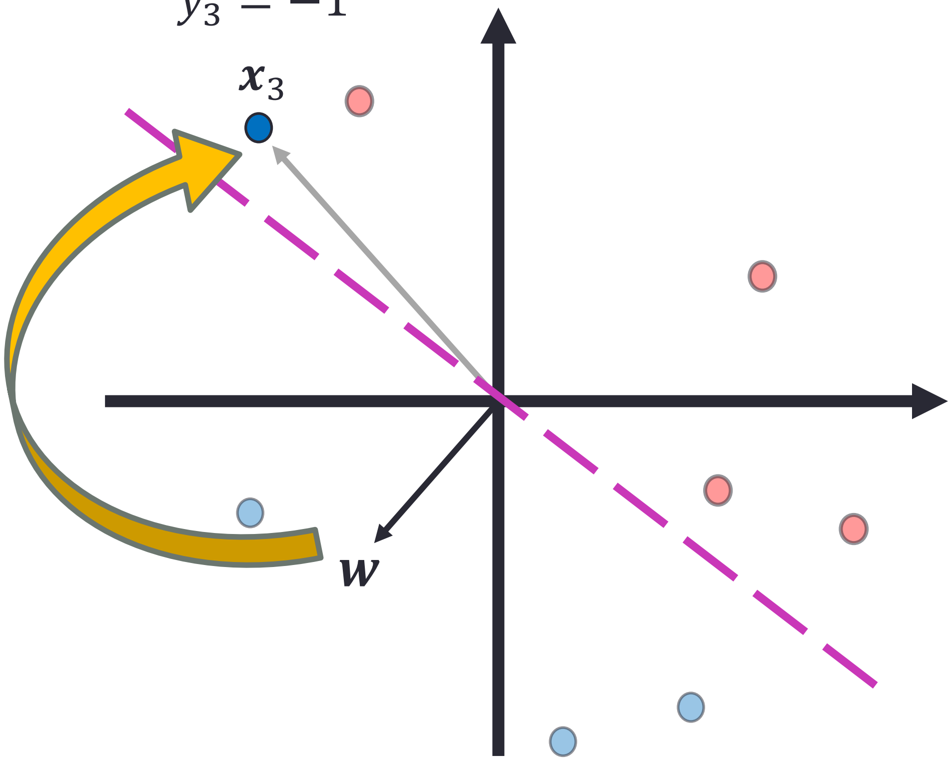
- First and simplest linear model
- Trained iteratively:

```
w = 0_d

while didn't separate trainset
    for i=1 to m
        ŷ_i = sign(wᵀx_i)

        if y_i != ŷ_i
            w = w + ηy_i x_i
```

$\boldsymbol{w} = \boldsymbol{w} + 1 \cdot \boldsymbol{x}_i$

Should update!

$\hat{y}_3 = -1$



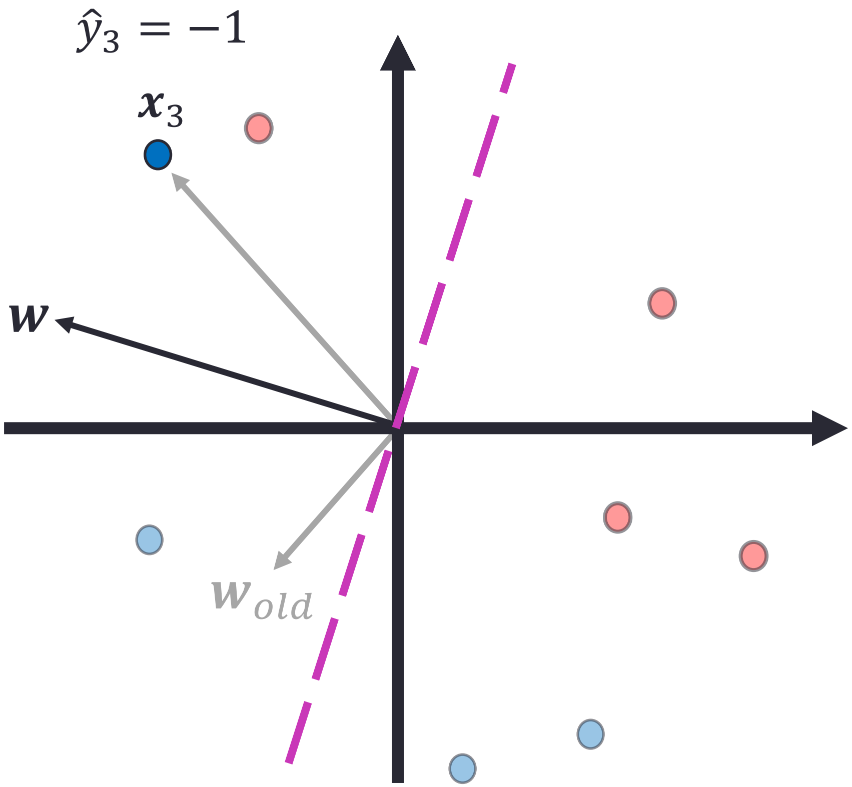
Perceptron – Intuition

- First and simplest linear model
- Trained iteratively:

```
w = 0_d

while didn't separate trainset
    for i=1 to m
        ŷ_i = sign(wᵀx_i)

        if y_i != ŷ_i
            w = w + η y_i x_i
```

$w = w + 1 \cdot x_i$

Should update!



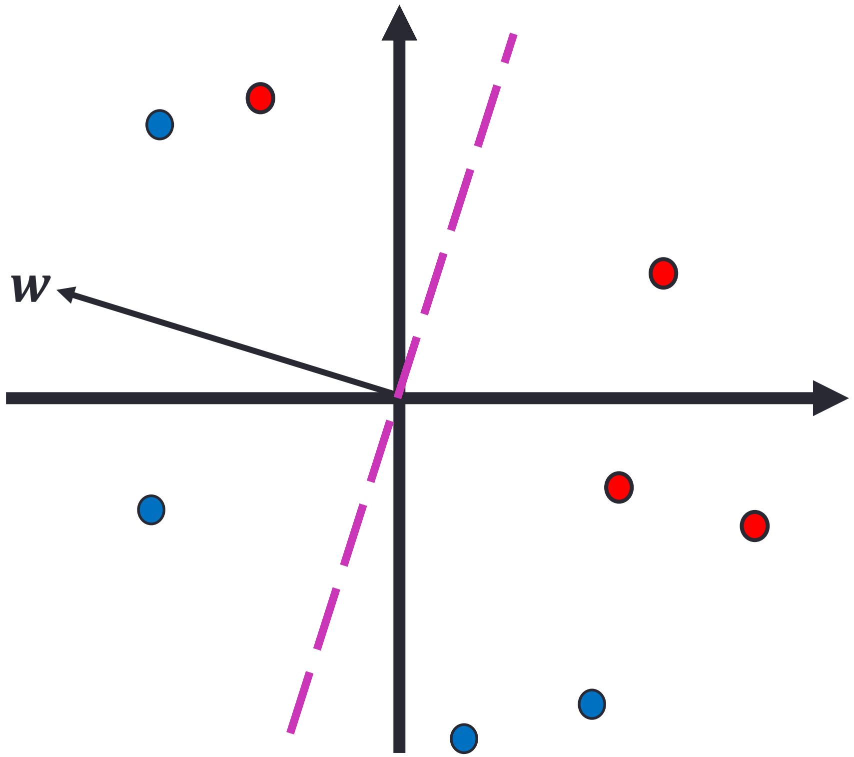
Perceptron – Intuition

- First and simplest linear model
- Trained iteratively:

$\boldsymbol{w} = \mathbf{0}_d$

while didn't separate trainset
    for i=1 **to** m
        $\hat{y}_i = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x}_i)$

        if $y_i \;!\!= \hat{y}_i$
            $\boldsymbol{w} = \boldsymbol{w} + \eta y_i \boldsymbol{x}_i$



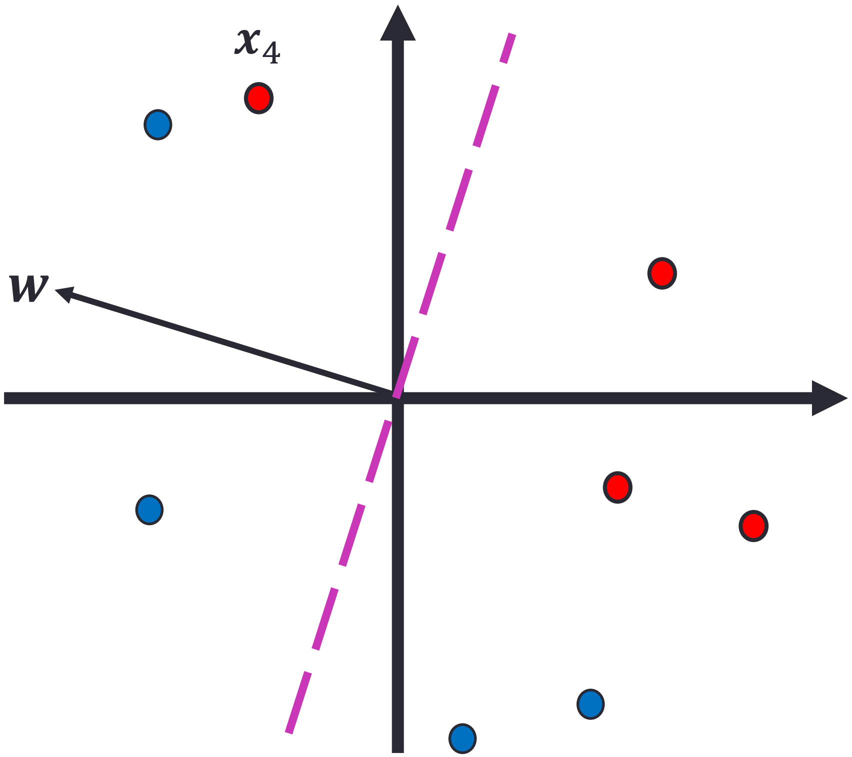
Perceptron – Intuition

- First and simplest linear model
- Trained iteratively:

```
w = 0_d

while didn't separate trainset
    for i=1 to m
        ŷ_i = sign(wᵀx_i)

        if y_i != ŷ_i
            w = w + η y_i x_i
```

Should update!

$\hat{y}_4 = 1$



Perceptron – Intuition

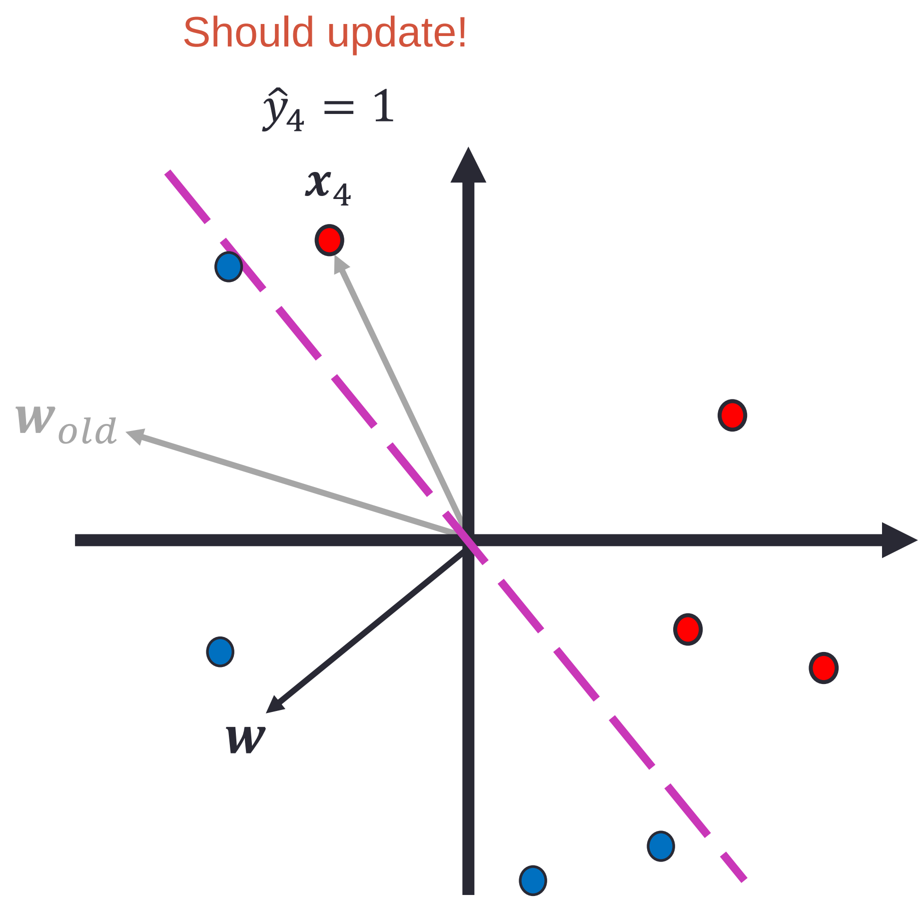
- First and simplest linear model
- Trained iteratively:

$\boldsymbol{w} = \boldsymbol{0}_d$

while didn't separate trainset
    for i=1 **to** m
        $\hat{y}_i$ = sign$(\boldsymbol{w}^\top \boldsymbol{x}_i)$

        if y_i != $\hat{y}_i$
            $\boldsymbol{w} = \boldsymbol{w} + \eta y_i \boldsymbol{x}_i$

$$\boldsymbol{w} = \boldsymbol{w} + (-1) \cdot \boldsymbol{x}_i$$



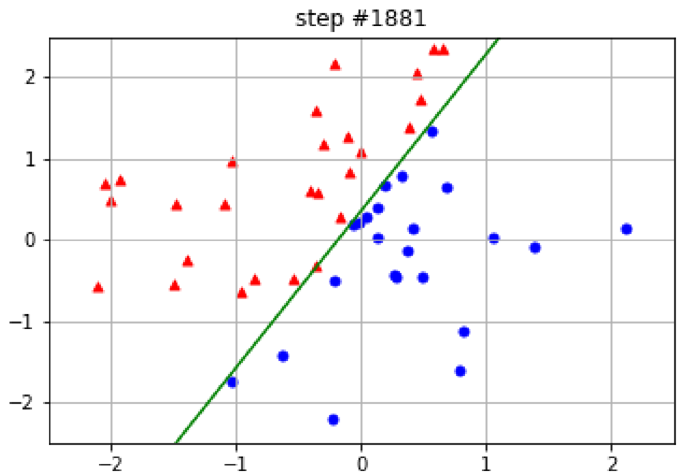
Perceptron

Guarantee: the algorithm will stop on linearly separable data
(under mild conditions; without proof)

- First and simplest linear model
- Trained iteratively:

```
eta = 1
w = np.zeros(d + 1)

errorFound = True

while errorFound:
    errorFound = False

    for x_i, y_i in zip(X,y):
        y_predicted = np.sign(w.dot(x_i))

        if y_predicted != y_i:
            errorFound = True

            w = w + (eta * y_i) * x_i
```

x_i has a d+1 dimension that holds '1'



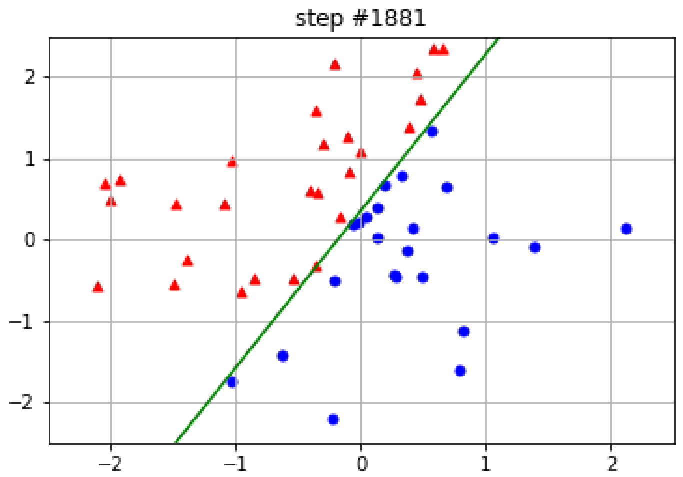
Done!

Perceptron

- First and simplest linear model
- Trained iteratively:

Recall: why +1?

```
eta = 1
w = np.zeros(d + 1)

errorFound = True

while errorFound:
    errorFound = False

    for x_i, y_i in zip(X,y):
        y_predicted = np.sign(w.dot(x_i))

        if y_predicted != y_i:
            errorFound = True

            w = w + (eta * y_i) * x_i
```



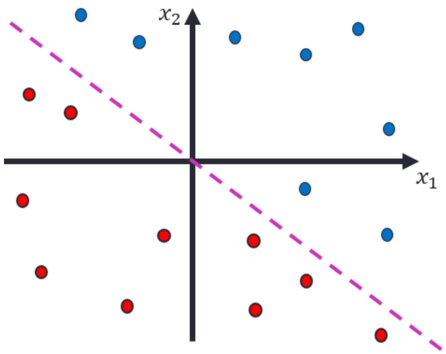
Done!

Recall: Extension to non-homogeneous

- Reduce non-homogeneous case to homogeneous:
  - Add a constant feature to all examples
  - Find a $(d+1)$-dimensional homogeneous separator

$$\mathrm{sign}(\boldsymbol{w}^\top \boldsymbol{x} + b) = \mathrm{sign}\left(\begin{bmatrix}\boldsymbol{w}\\ b\end{bmatrix}^\top \begin{bmatrix}\boldsymbol{x}\\ 1\end{bmatrix}\right)$$

Homogeneous linear separator

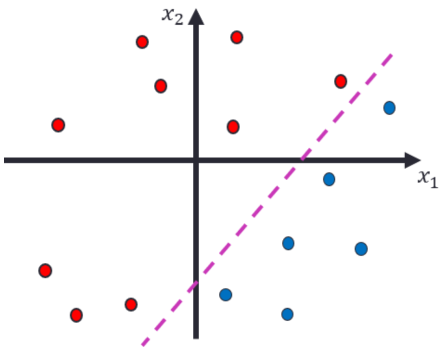


- Can extend every homogeneous linear model!
  - But careful: b does not affect complexity

    and should not be regularized!

    (for instance in SVMs)

Non-homogeneous linear separator



PRACTICAL DEMO

Digit recognition

Demo: MNIST

- Famous computer vision dataset
- Examples are grayscale images in $\{0,1,\ldots,255\}^{28\times 28}$
- 10 classes
- 60,000 train examples, 10,000 test example

Demo: MNIST

- Let's try to solve a binary classification task: 0 or not 0?
- All images are flattened into $\{0,1,\dots,255\}^{784}$ vectors
- Let's train a perceptron!

Demo: MNIST

Load data

```
from keras.datasets import mnist
(train_X, train_y), (test_X, test_y) = mnist.load_data()
train_X = train_X.reshape(-1, 784)     # shape: (60000, 784)
test_X = test_X.reshape(-1, 784)       # shape: (10000, 784)
```

Make binary labels

```
PREDICTED_CLASS = 0
train_binary_y = [1 if y == PREDICTED_CLASS else -1
                  for y in train_y]
test_binary_y = [1 if y == PREDICTED_CLASS else -1
                 for y in test_y]
```

Prepare for non-homogeneous training

```
train_X = np.hstack([train_X, np.ones((train_X.shape[0], 1))])
test_X = np.hstack([test_X, np.ones((test_X.shape[0], 1))])
```

Demo: MNIST

Don’t assume
data is linearly separable

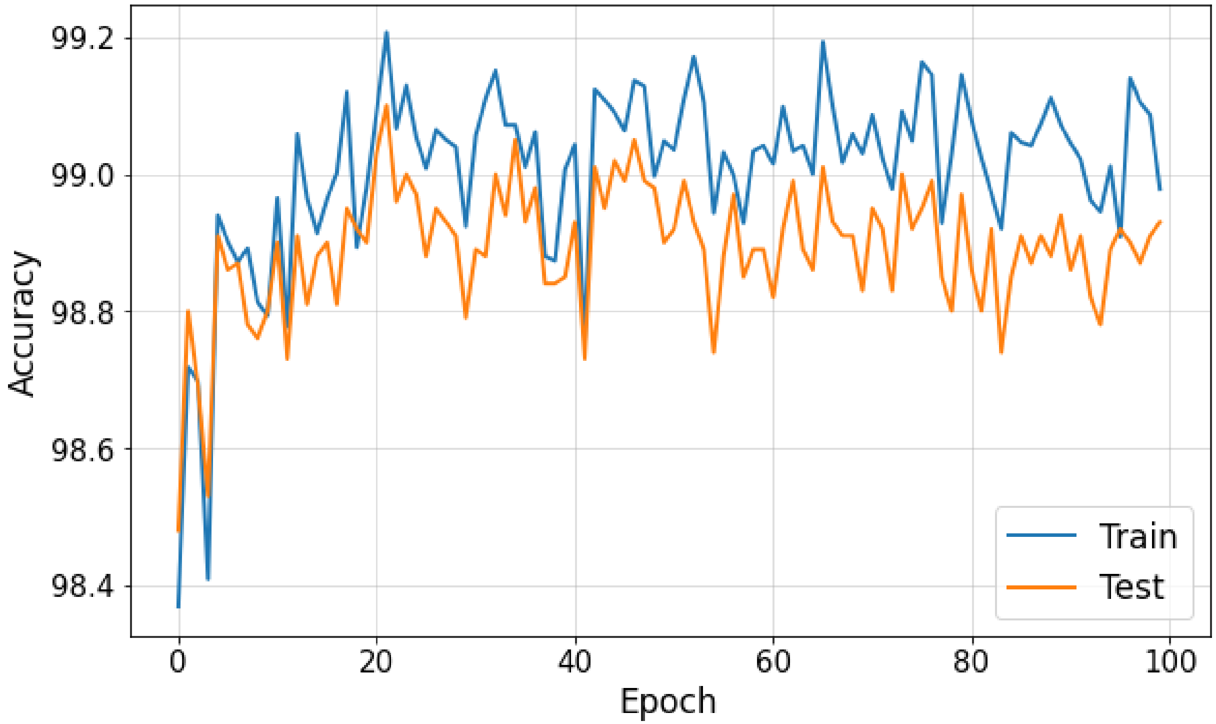
Train the perceptron

```
eta = 1
w = np.zeros(d + 1)

errorFound = True

for epoch in range(100):
    errorFound = False

    for x_i, y_i in zip(X,y):
        y_predicted = np.sign(w.dot(x_i))

        if y_predicted != y_i:
            errorFound = True
            w = w + (eta * y_i) * x_i

    if not errorFound:
      print(“Data is linearly separable!”)
      break
```

Demo: MNIST

- Classes are approximately balanced
- What is the accuracy of a random guess for the "0 or not 0" task?

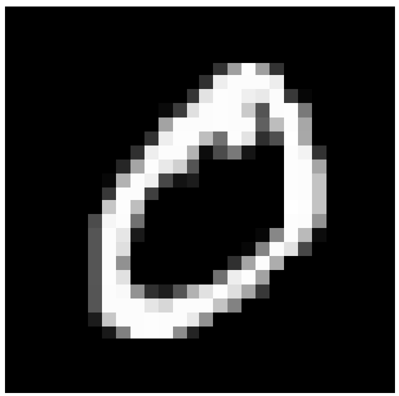
Learning curve



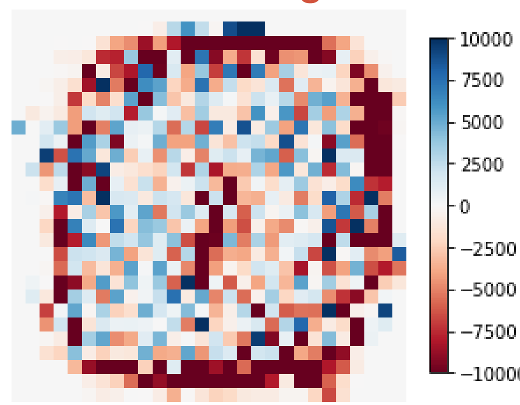
Demo: MNIST

- Easy to get an intuition of the learned linear model:

Example image

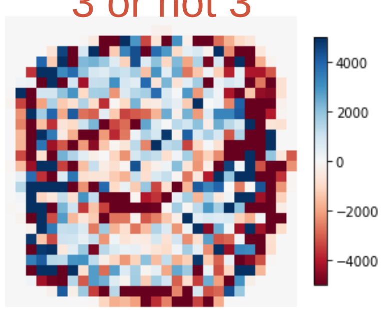


Learned weights

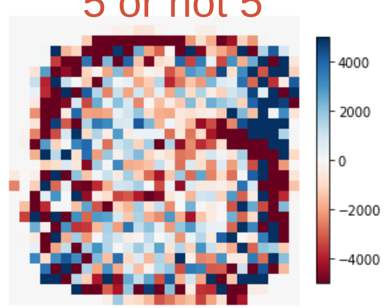


- If instead we classify other digits:

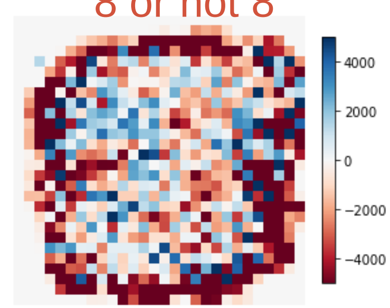
3 or not 3



5 or not 5



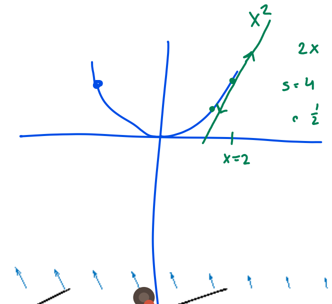
8 or not 8



OPTIMIZATION PERSPECTIVE

Why does any of it work?

Recap: Gradient descent (GD)



Assume a differentiable loss $\mathcal{L}: \mathbb{R}^d \to \mathbb{R}$

- Goal: find $\boldsymbol{w}^* = \underset{\boldsymbol{w} \in \mathbb{R}^d}{\operatorname{argmin}} \mathcal{L}(\boldsymbol{w})$

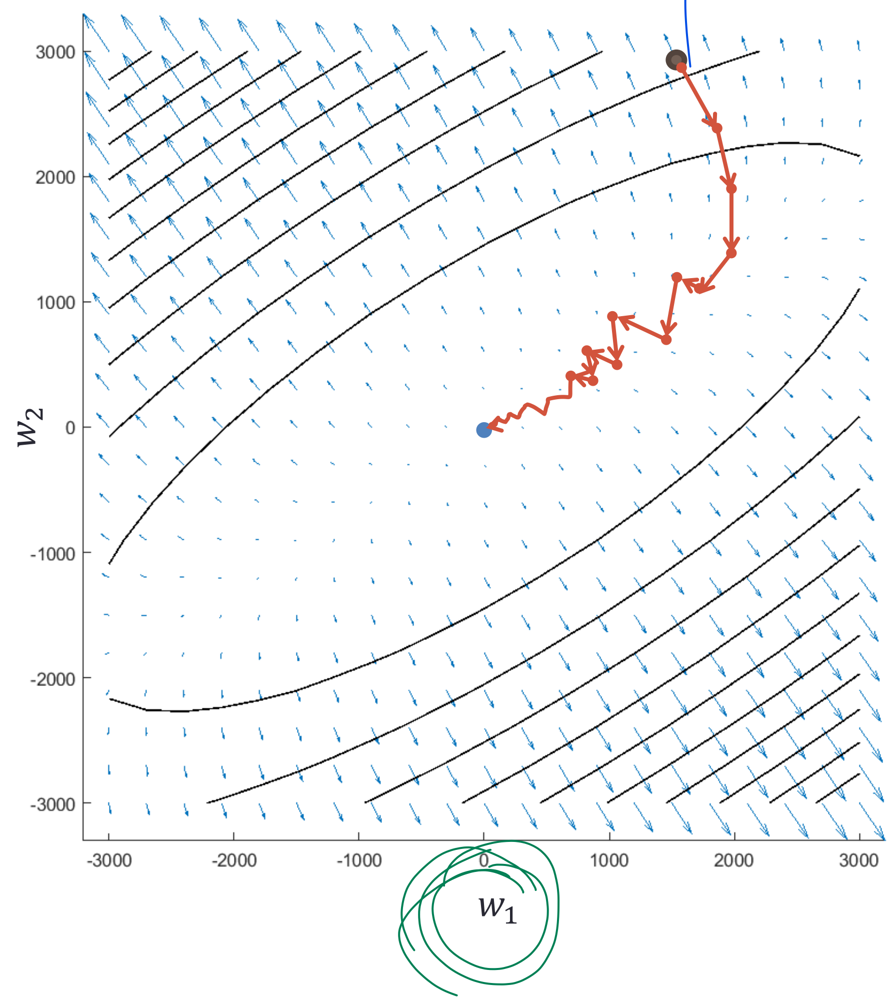
- Idea: gradients point to the steepest ascent direction of the loss landscape

- Descend iteratively:

W=(X, X

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla_{\boldsymbol{w}} \mathcal{L}\left(\boldsymbol{w}^{(k)}\right)$$

- Guarantee: for small enough η, GD converges to a local minimum



Recap: Stochastic GD (SGD)

- Many losses decompose over the trainset $\mathcal{L}(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \ell(\boldsymbol{x}_i, y_i, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$

Example: Soft SVM's formulation

$$\boxed{\underset{\boldsymbol{w}\in\mathbb{R}^d}{\operatorname{argmin}} \frac{1}{m}\sum_{i\in[m]} \ell_{hinge}(\boldsymbol{x}_i, y_i, \boldsymbol{w}) + \lambda\|\boldsymbol{w}\|_2^2}$$

- The gradients also decompose: $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{w}}\ell(\boldsymbol{x}_i, y_i, \boldsymbol{w}) + \lambda\nabla_{\boldsymbol{w}}R(\boldsymbol{w})$

Example: Soft SVM's gradient

$$\boxed{\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = \frac{1}{m}\sum_{i\in[m]} \nabla_{\boldsymbol{w}}\ell_{hinge}(\boldsymbol{x}_i, y_i, \boldsymbol{w}) + 2\lambda\boldsymbol{w}}$$

- GD uses all directions $\quad \boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta\left(\frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{w}}\ell(\boldsymbol{x}_i, y_i, \boldsymbol{w}) + \lambda\nabla_{\boldsymbol{w}}R(\boldsymbol{w})\right)$
- SGD uses a random subset $\quad \boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta\left(\nabla_{\boldsymbol{w}}\ell\left(\underset{\uparrow}{\boldsymbol{x}_i}, y_i, \boldsymbol{w}^{(k)}\right) + \lambda\nabla_{\boldsymbol{w}}R(\boldsymbol{w})\right)$

$(\boldsymbol{x}_i, y_i) \sim S$

Recap: Stochastic GD (SGD)

- Many losses decompose over the trainset $\mathcal{L}(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \ell(\boldsymbol{x}_i, y_i, \boldsymbol{w}) + \lambda R(\boldsymbol{w})$

- The gradients also decompose: $\nabla_{\boldsymbol{w}}\mathcal{L}(\boldsymbol{w}) = \frac{1}{m}\sum_{i=1}^{m} \nabla_{\boldsymbol{w}}\ell(\boldsymbol{x}_i, y_i, \boldsymbol{w}) + \lambda \nabla_{\boldsymbol{w}} R(\boldsymbol{w})$

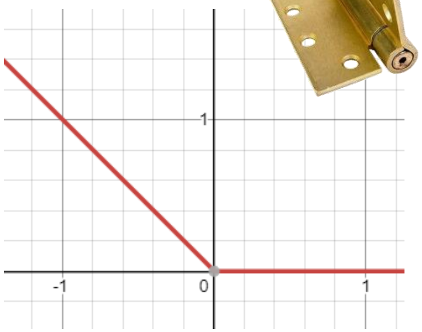
- GD uses all directions $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \nabla_{\boldsymbol{w}}\mathcal{L}\left(\boldsymbol{w}^{(k)}\right)$

- SGD uses a random subset $\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \eta \left(\nabla_{\boldsymbol{w}}\ell\left(\boldsymbol{x}_i, y_i, \boldsymbol{w}^{(k)}\right) + \lambda \nabla_{\boldsymbol{w}} R(\boldsymbol{w})\right)$

- Notice: the directions are equal in expectation!

$$\underbrace{\mathbb{E}_{(x_i,y_i)\sim S}\left[\nabla_{\boldsymbol{w}}\ell\left(\boldsymbol{x}_i, y_i, \boldsymbol{w}^{(k)}\right)\right]}_{\text{SGD}} + \lambda \nabla_{\boldsymbol{w}} R\left(\boldsymbol{w}^{(k)}\right) = \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(\nabla_{\boldsymbol{w}}\ell\left(\boldsymbol{x}_i, y_i, \boldsymbol{w}^{(k)}\right)\right)}_{\text{GD}} + \lambda \nabla_{\boldsymbol{w}} R\left(\boldsymbol{w}^{(k)}\right)$$

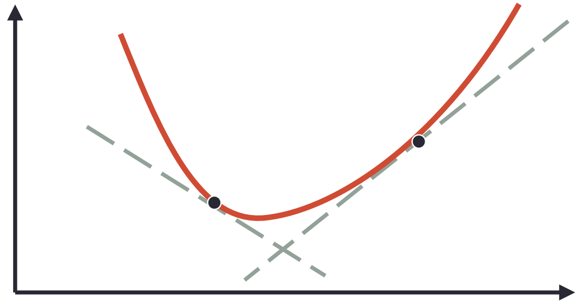
Perceptron as an SGD algorithm



- Define the loss $\mathcal{L}(\boldsymbol{w}) = \sum_i \ell(y_i \, \boldsymbol{w}^\top \boldsymbol{x}_i)$
  - Where ℓ is a hinge-like function: $\ell(z) = \max\{0, -z\}$
- Prove: the perceptron algorithm performs SGD on $\mathcal{L}(\boldsymbol{w})$
- But wait!
  - This loss is not differentiable at every point!
  - How can we use GD?
  - We want to generalize gradients to non-differentiable functions

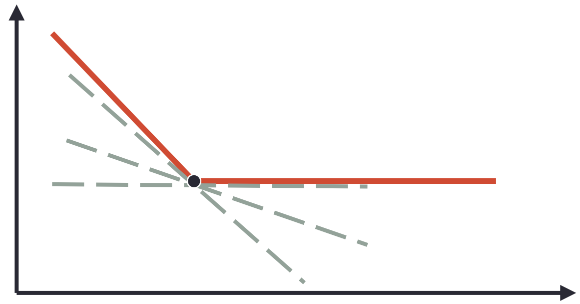
Tangents of convex differentiable functions

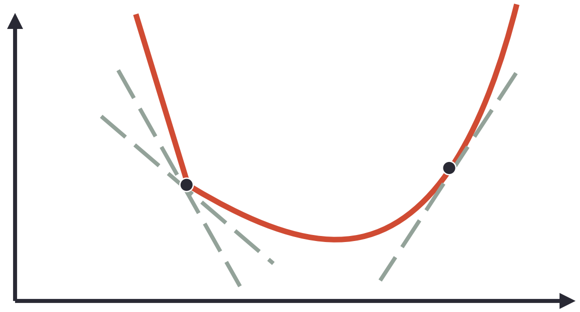
- Property: the tangents of convex functions are below the function



Subgradients of convex functions

- Intuition: a subgradient is the slope of any tangent to the function at a given point.



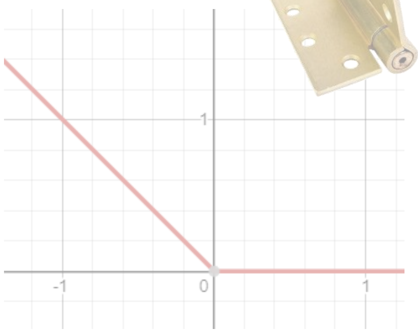


- Formally: Let $f: V \to \mathbb{R}$ be a convex function.

  Denote the set of subgradients of f at point $u \in V$ by $\partial f(u)$.

$$g \in \partial f(u) \text{ if } \forall v \in V: \; f(v) \geq f(u) + \langle g, v - u \rangle$$

Perceptron as an SGD algorithm

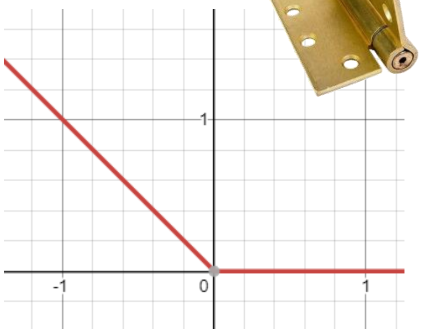


- Define the loss $\mathcal{L}(\boldsymbol{w}) = \sum_i \ell(y_i\, \boldsymbol{w}^\top \boldsymbol{x}_i)$
  - Where ℓ is a hinge-like function: $\ell(z) = \max\{0, -z\}$

- Prove: the perceptron algorithm performs SGD on $\mathcal{L}(\boldsymbol{w})$

- But wait!
  - This loss is not differentiable at every point!
  - Use subgradients instead of gradients!
  - However, fixed "small enough" step sizes no longer guarantee convergence, even for convex functions (specifically for the perceptron they do).
    - Extra: but diminishing step sizes do.

Perceptron as an SGD algorithm



- Define the loss $\mathcal{L}(\boldsymbol{w}) = \sum_i \ell(y_i\, \boldsymbol{w}^\top \boldsymbol{x}_i)$
  - Where ℓ is a hinge-like function: $\ell(z) = \max\{0, -z\}$

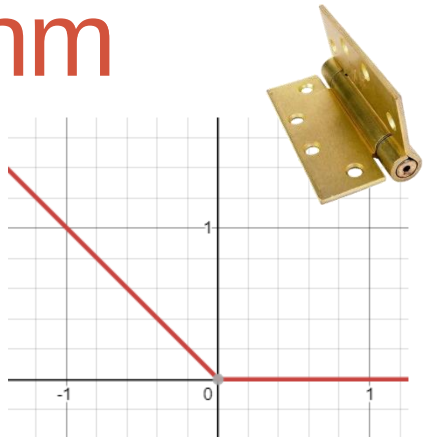
- Prove: the perceptron algorithm performs SGD on $\mathcal{L}(\boldsymbol{w})$

The perceptron update step

$$
\begin{aligned}
&\hat{y}_{\mathrm{i}} = \mathrm{sign}(\boldsymbol{w}^\top \boldsymbol{x}_i) \\
&\textbf{if } y_i \; != \; \hat{y}_{\mathrm{i}} \\
&\quad \boldsymbol{w} = \boldsymbol{w} + \eta y_i \boldsymbol{x}_i
\end{aligned}
$$

- Intermediate steps:
  1. Find $\frac{\partial}{\partial w_j} \ell(y_i\, \boldsymbol{w}^\top \boldsymbol{x}_i) = \frac{\partial}{\partial w_j} \max\{0, -y_i\, \boldsymbol{w}^\top \boldsymbol{x}_i\}$
  2. Find $\nabla_{\boldsymbol{w}} \ell(y_i\, \boldsymbol{w}^\top \boldsymbol{x}_i)$

Perceptron as an SGD algorithm



- Define the loss $\mathcal{L}(\boldsymbol{w}) = \sum_i \ell(y_i\, \boldsymbol{w}^\top \boldsymbol{x}_i)$
  - Where ℓ is a hinge-like function: $\ell(z) = \max\{0, -z\}$
- Prove: the perceptron algorithm performs SGD on $\mathcal{L}(\boldsymbol{w})$
- Extra: is this loss convex w.r.t $\boldsymbol{w}$? (similar exercise in Short HW3)
- Notice: does not encourage margins

Extra: Margin perceptron

- We now wish to use the common hinge function

$$\ell_{hinge}(z) = \max\{0, 1 - z\}$$

1. What changes in the learned separators can we expect?
2. Extra: Update the perceptron update step so it performs SGD on the "shifted" loss: $\mathcal{L}_s(\mathbf{X}, \boldsymbol{y}, \boldsymbol{w}) = \sum_i \ell_{hinge}(y_i\, \boldsymbol{w}^\top \boldsymbol{x}_i)$

The original perceptron update step:

```
y_predicted = np.sign(w.dot(x_i))

if y_predicted != y_i:
    w = w + (mu * y_i) * x_i
```

3. Think: What is the difference between this and Soft SVM?

Summary

- Perceptron is a linear classification algorithm
  - Simple but works well
  - Performs SGD with a hinge-like loss
  - No margin guarantees