CMSC 510 – L04
Regularization Methods for Machine Learning

Instructor:

Dr. Tom Arodz

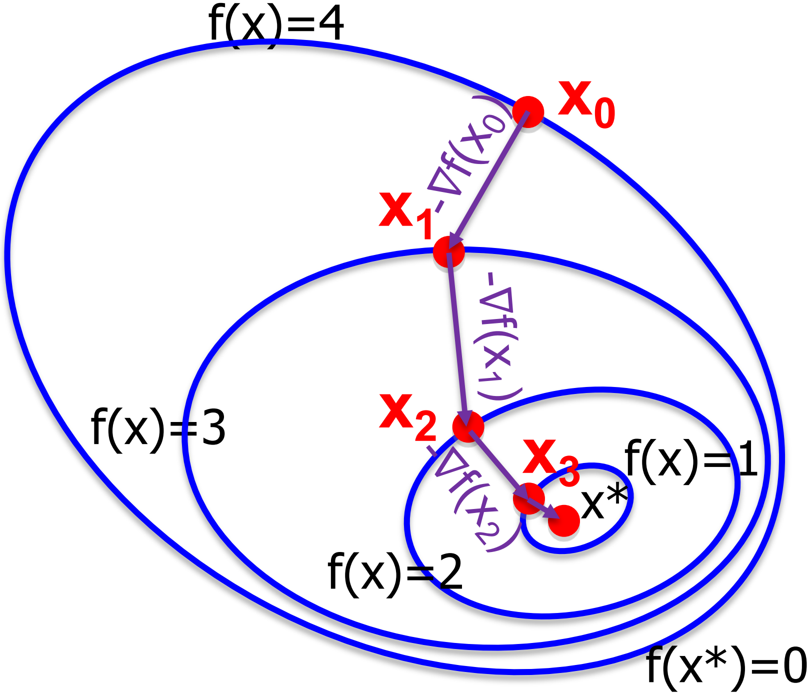
Recap: Gradient descent

$$\nabla f(\mathbf{z}) = \left(\frac{\partial}{\partial x_1} f(\mathbf{x})|_{\mathbf{x}=\mathbf{z}}, \ldots, \frac{\partial}{\partial x_n} f(\mathbf{x})|_{\mathbf{x}=\mathbf{z}} \right)$$

Gradient descent:

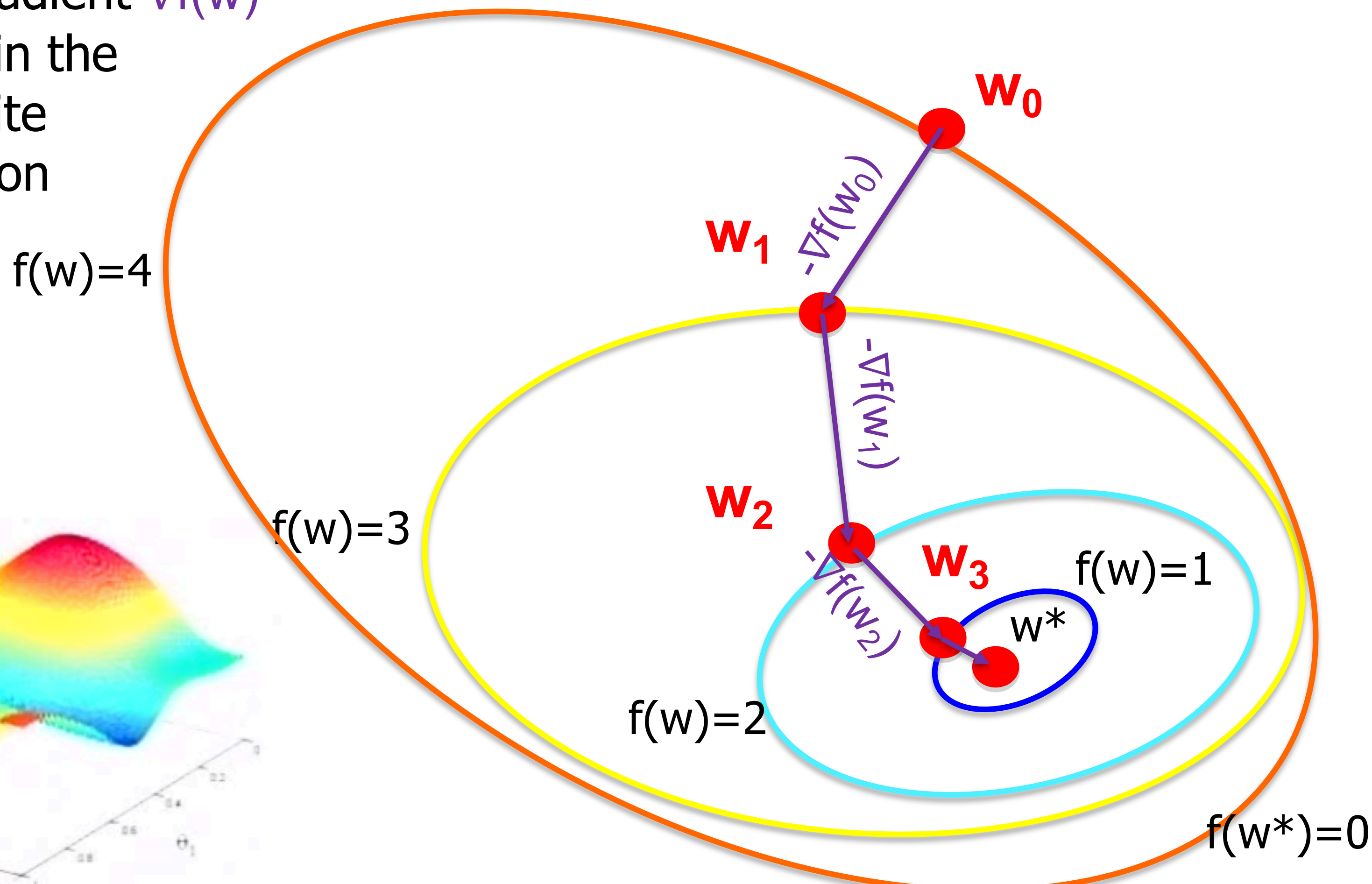
- We start from x_0
- We calculate $x_1 = x_0 - \nabla f(x_0)/L$
- We calculate $x_2 = x_1 - \nabla f(x_1)/L$
- $\mathbf{x_{n+1} = x_n - \nabla f(x_n)/L}$

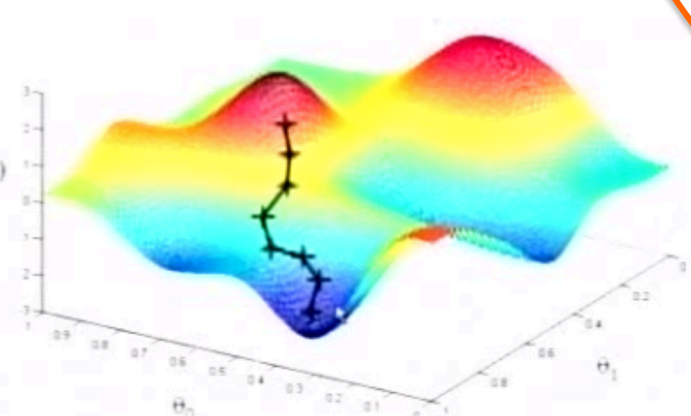
If we choose L large enough
g.d. goes down in each step,
converging towards
global minimum (e.g. for convex f)
or
local minimum



Gradient descent

- Gradient descent for minimizing f(w)
- Loop:
  - At position w, find the direction of steepest increase of f(w)
    - gradient $\nabla f(w)$
  - Move in the opposite direction





Linear Models

- "predicted y"$=\langle w,x\rangle+b = w^T x + b = w_1 x_1 + w_2 x_2 + b$

Linear Models - Regression

- Finding "good" w,b
  - For regression problems, where we are trying to predict a real number "true y", we typically use
    MSE: mean squared error
- ("true y" - "predicted y")^2
- error for model (w,b), for sample (x,y) is:
  $(y - w^T x - b)^2$
- we minimize mean error: average error on the training set

Linear Models - Regression

- 1D Example:
  - three samples z=(x,y):
    - $z_1=(0,1)$, $z_2=(2,3)$, $z_3=(-1,0)$
- $$\text{MSE} = [(y_1 - (wx_1 + b))^2 + (y_2 - (wx_2 + b))^2 + (y_3 - (wx_3 + b))^2] / 3$$
- $$\text{MSE} = [(1 - (0w + b))^2 + (3 - (2w + b))^2 + (0 - (-1w + b))^2] / 3$$
- Gradients of MSE w.r.t. to w?

  w.r.t. to b?

Linear Models - Regression

- 1D Example:
  - three samples z=(x,y):
    - $z_1=(0,1)$, $z_2=(2,3)$, $z_3=(-1,0)$
- MSE = $[(y_1 - (wx_1 + b))^2$
  $+ (y_2 - (wx_2 + b))^2$
  $+(y_3 - (wx_3 + b))^2] / 3$
- MSE = $[(1 - (0w + b))^2$
  $+ (3 - (2w + b))^2$
  $+(0 - (-1w + b))^2] / 3$
- Gradients of MSE w.r.t. to w?
  w.r.t. to b?
- Gradient is additive **($\nabla \Sigma = \Sigma \nabla$),**
  so we just need to be able to find gradient of a formula
  $(y-(ax+b))^2$
  apply it to each training sample (x,y), and add up

Linear Models - Regression

- Apply chain rule of differentiation to:

- $\partial(y-(ax+b))^2 / \partial a$

$$\frac{\partial}{\partial a}\left((y-(a\,x+b))^2\right) = -2\,x\,(-a\,x-b+y)$$

- $\partial(y-(ax+b))^2 / \partial b$

$$\frac{\partial}{\partial b}\left((y-(a\,x+b))^2\right) = -2\,(-a\,x-b+y)$$

- Let:
  - $h(x,a,b)=ax+b$
  - $e(y,x,a,b)=y-h(x,a,b)$
  - $L(y,x,a,b)=e(y,x,a,b)^2$

Designing a classification method

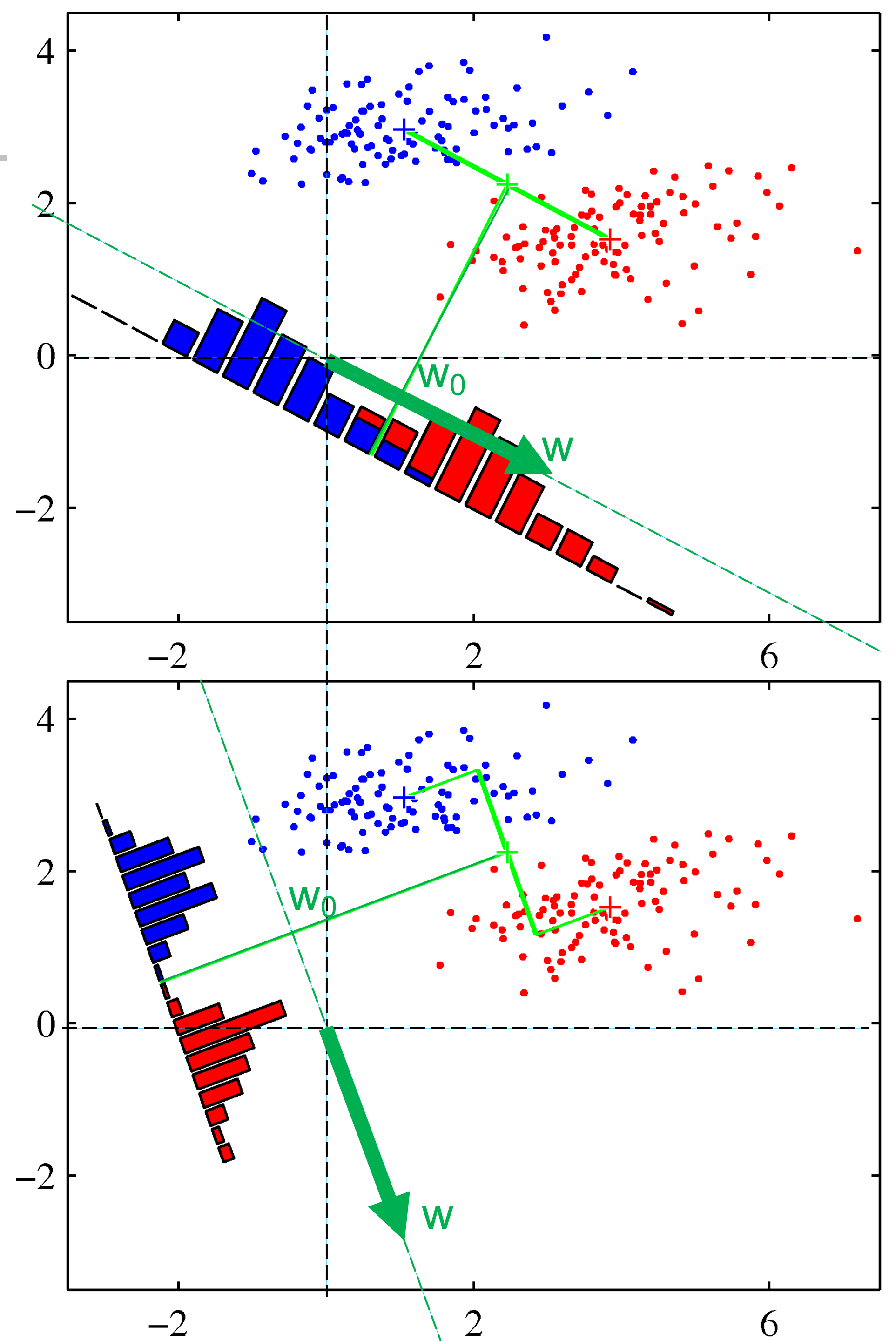
1. Define the space of possible decision boundaries
   - What will be the form of classifiers?
     - E.g. space of all possible lines/planes/hyperplanes

2. Define the loss/risk function
   - How to evaluate the quality of a specific classifier from the space of possible classifiers?
     - E.g. perceptron loss and empirical risk associated with it

3. Define the method for minimizing the risk using data from the training set
   - How to reach a high-quality classifier?
     - E.g. gradient descent over the space of model parameters

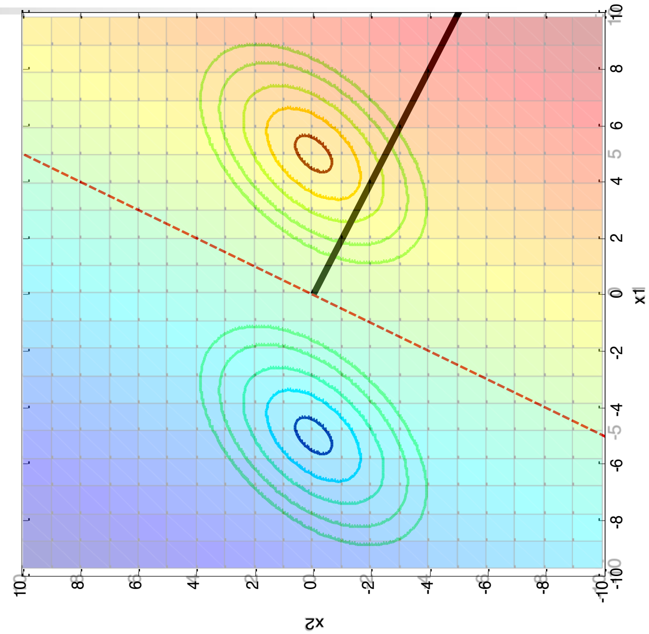
Classifiers

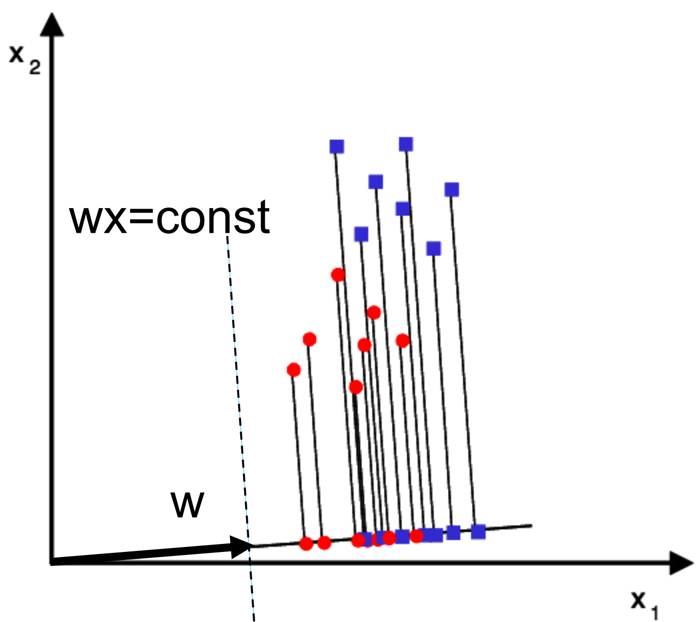
- Linear classifiers:
  - True class: -1, 1
  - We fit: $(w^T x + w_0)$
  - Then use "sign" to get the predicted class: class=sign($w^T x + w_0$)
  - $w^T x$ - linear projection of samples on a line
  - $-w_0$ defines the decision threshold on that line
  - "predicted y"=<w,x>+b
    $= w^T x + b = w_1 x_1 + w_2 x_2 + b$

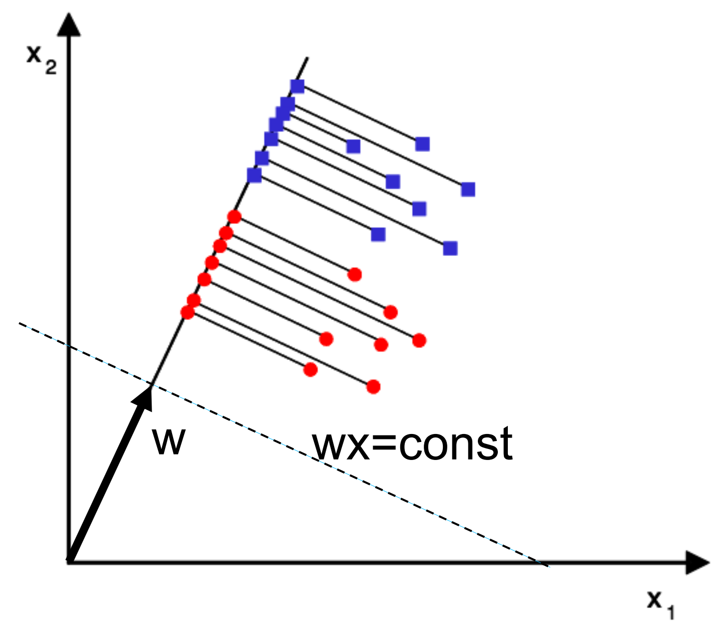


Linear Classifiers

- Linear classifier:
  - class=sign($w^T x+w_0$)
- $w^T x$ - linear projection of samples on a line
- w_0 defines a threshold on that line

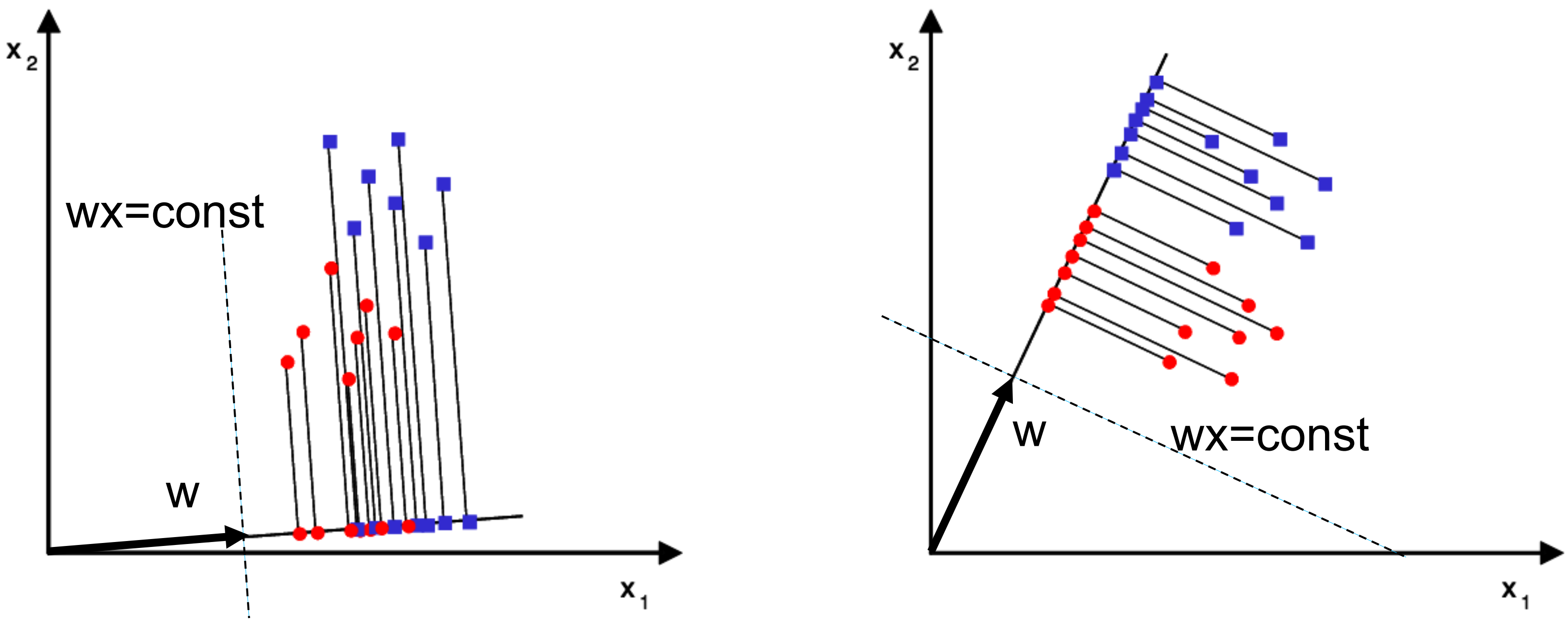






Choosing weights

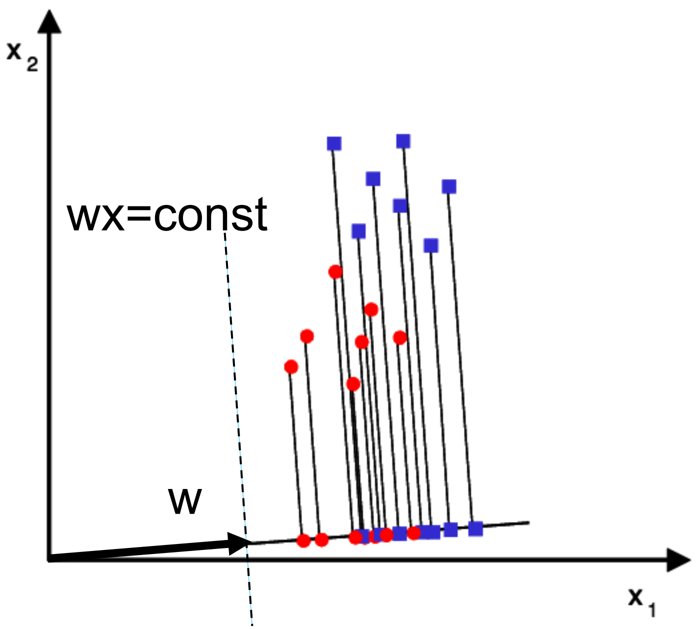
NOT THIS ☹ THIS ☺



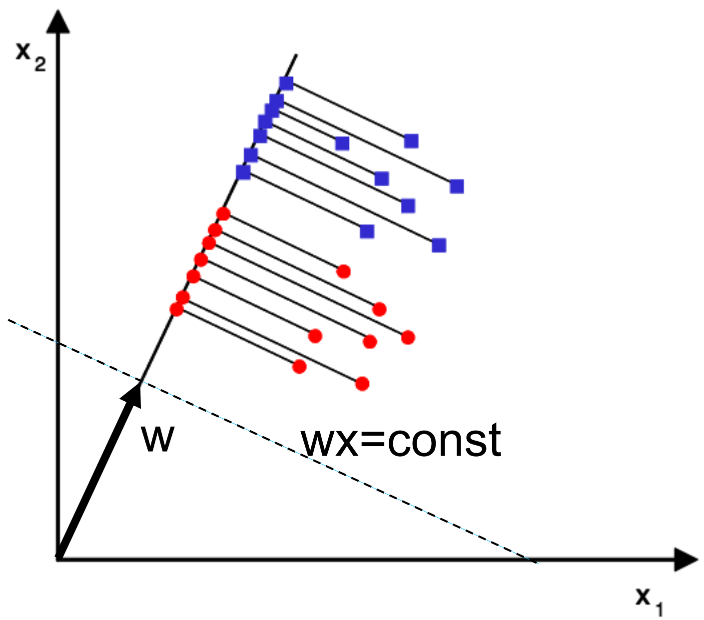
- Our predictions are: $h\left(\mathbf{x}\right) = \mathrm{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$
- How would you know if this classifier h (i.e. w and w_0) is good or bad?

Choosing weights

NOT THIS ☹



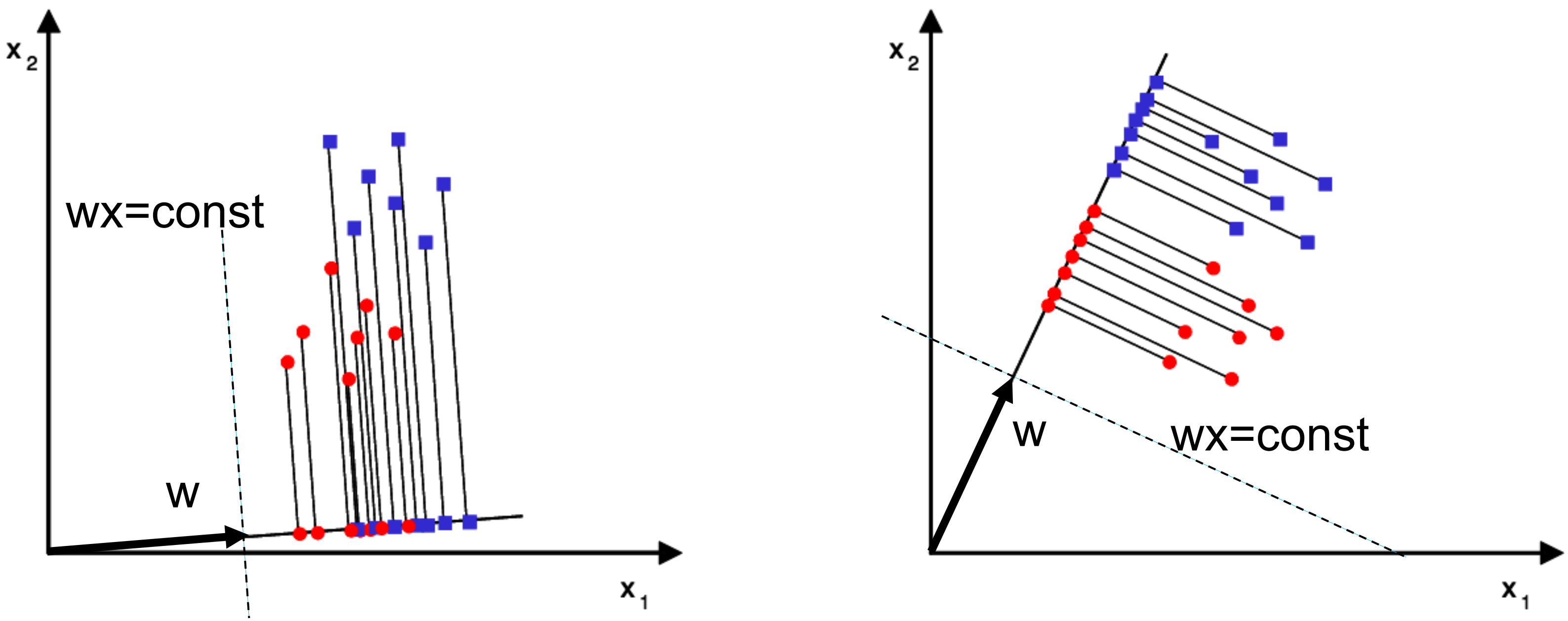
THIS ☺



- How would you know if **w** is good or bad?
- We need some measure of quality of **w**
  - E.g. number of misclassified points
  - Or something related

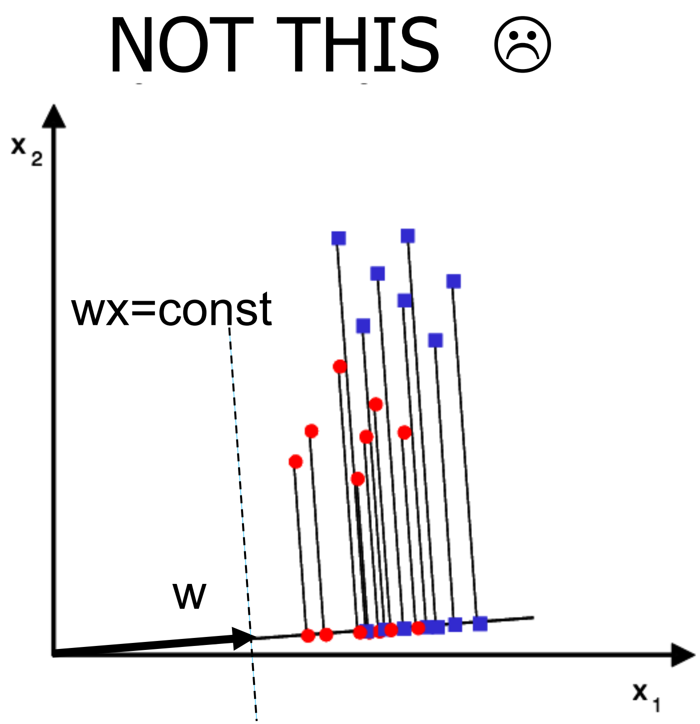
Choosing weights

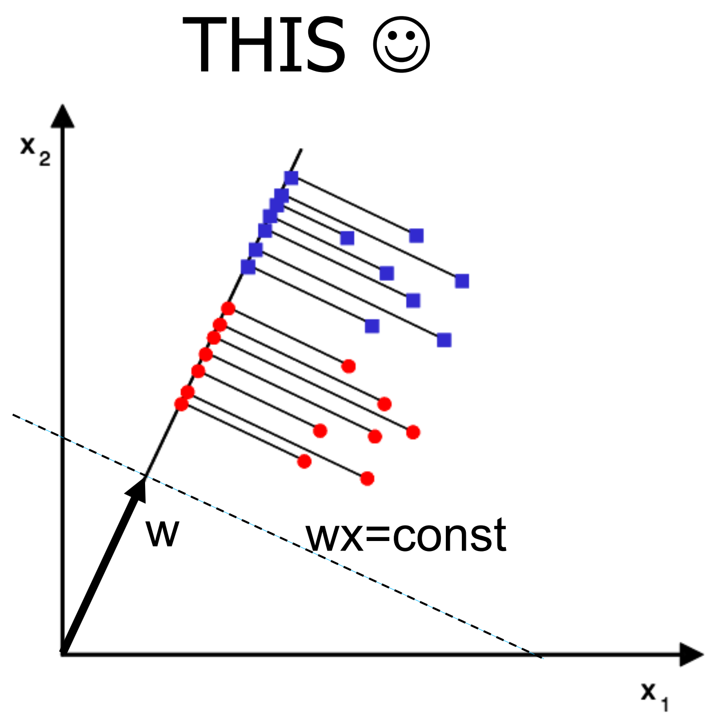
NOT THIS ☹ THIS ☺



- Our predictions are: $h(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$
- We have some measure of quality of *h*
  - E.g. number of errors made
  - Where to measure it? On what samples?

Choosing weights

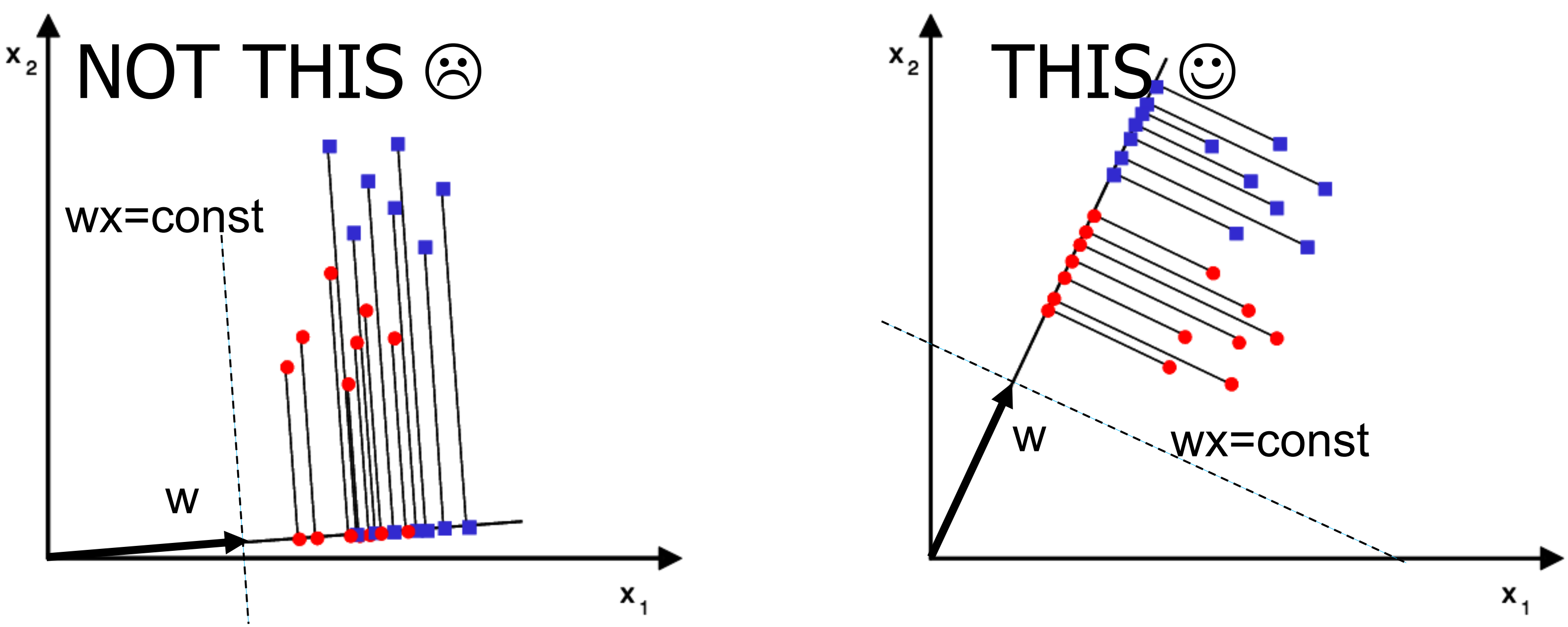




- Our predictions are: $h(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$
- We have some measure of quality of *h*
  - E.g. number of errors made
  - Where to measure it? On what samples?
    - Easiest solution: on the training set!

Linear classifiers

- Come up with a good vector **w**



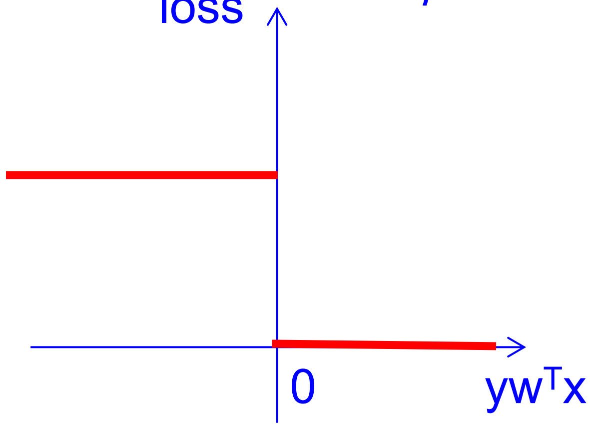
- We need to be able to measure which **w** is “good”
- For a single training sample (x,y), that measure is called “loss”

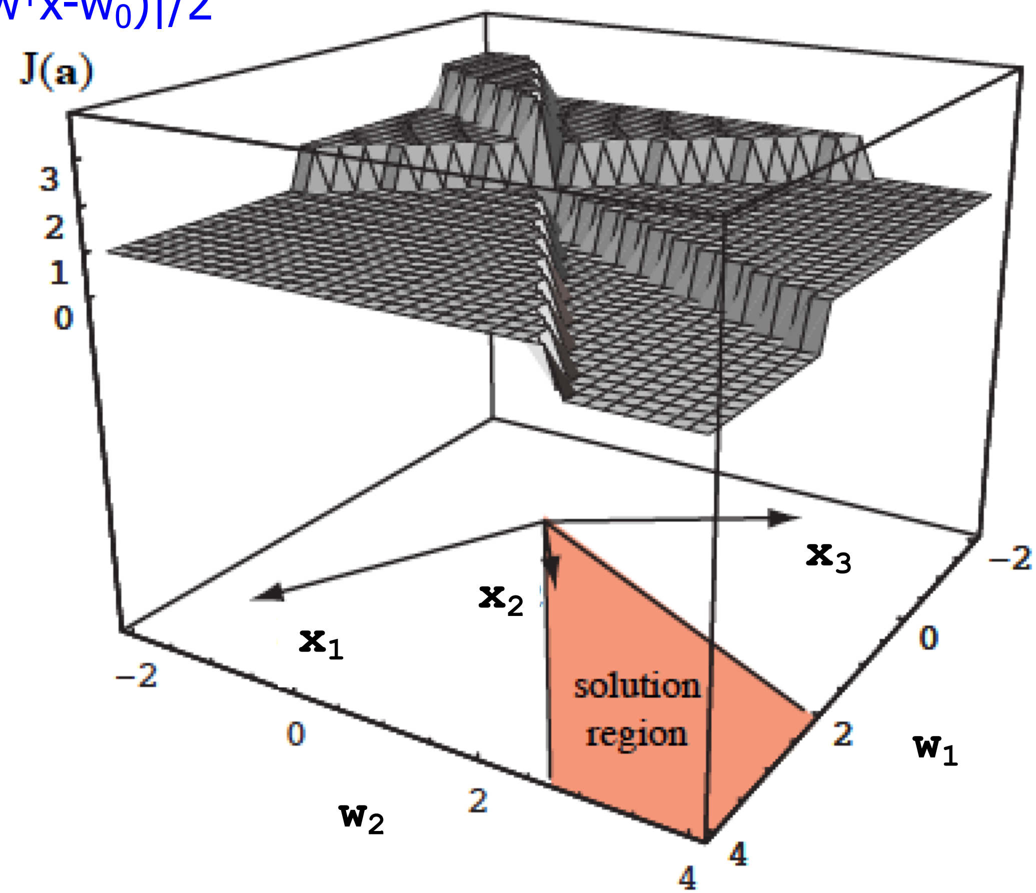
0/1 loss - # of wrong predictions

- **Present a sample x and predict:**

$$h(\mathbf{x}) = \mathrm{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$$

- **0/1 Loss: 0 if h(x)=y (or: yh(x)>0), 1 otherwise**
  - 0/1 loss: $|y - \mathrm{sign}(w^Tx - w_0)|/2$





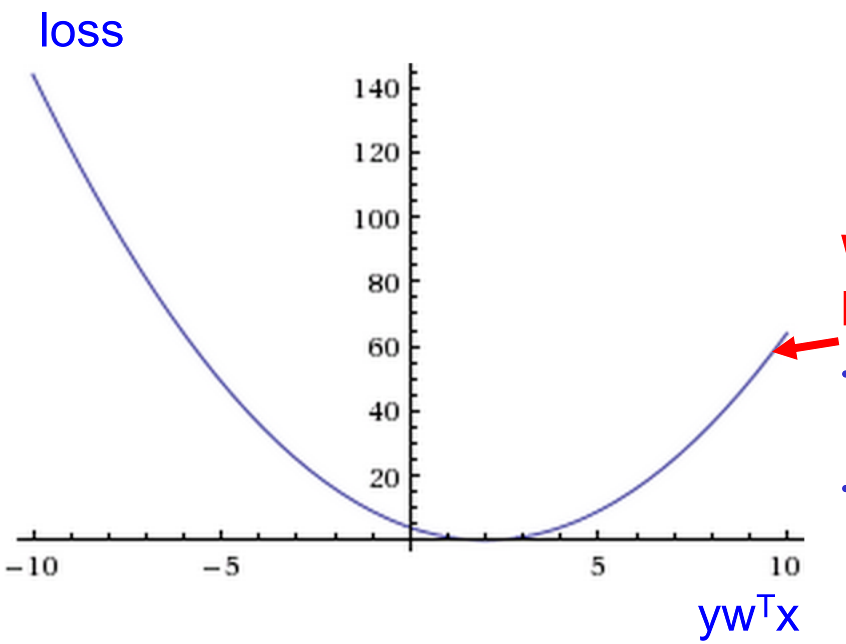
- Why not use 0/1 loss?
- It's flat! We don't know in which direction to move to get a better solution!
- At each possible w, we have # of misclassified points

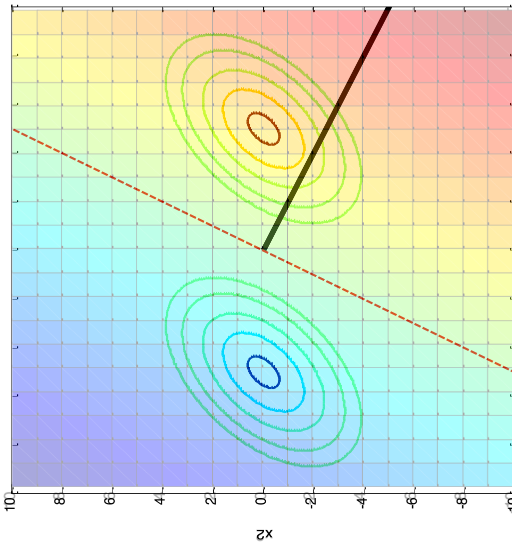
If not 0/1 loss, than maybe MSE?

- MSE for classification: $(y-w^Tx)^2$
  - *Least mean squares (Π_k is probability of class k)*

$$l(h,z) = \left(\frac{1}{\pi_k} - yw^Tx\right)^2$$

  - Hint: multiply by $y^2=1$
- The loss is convex, so we can reach global minimum





We get large loss not only for incorrect, but for correct predictions too!

- If data is Gaussian, very few points should be that far from decision boundary.
- But what if data is not Gaussian? BAD!
  - non-monotonic loss is suspicious for classification (but ok for regression)

Choosing the classifier

- Classification – probabilistic setting
  - We have observations in a fixed F-dimensional feature space
    - Every sample x is a vector (point) in that feature space

$$\mathbf{x} = \left[x^{\langle 1\rangle}, x^{\langle 2\rangle}, ..., x^{\langle F\rangle}\right]^T \qquad \mathbf{x} \in \mathcal{X} \qquad \mathcal{X} \subset \mathbb{R}^F$$

  - Sample x belongs to class y, either {-1, +1} $\mathbf{z} = (\mathbf{x}, y)$
    - So together we have an extended space $\mathcal{Z} = \mathcal{X} \times \{-1, +1\}$
    - We have a training set of m samples: $S_m \in \mathcal{Z}^m$

$$S_m = \{\mathbf{z}_1 = (\mathbf{x}_1, y_1), \mathbf{z}_2 = (\mathbf{x}_2, y_2), ..., \mathbf{z}_m = (\mathbf{x}_m, y_m)\}$$

  - We want to construct a classifier, e.g.:

$$h(\mathbf{x}) = \mathrm{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$$

  - Overall, a classifier is a function that returns real values:

$$h : \mathcal{X} \mapsto \mathbb{R}$$

    - If h(x)>0, we predict 1,
    - If h(x)<0, we predict -1
    - All possible classifiers of a certain type form a space $\mathcal{H}$
  - A learning algorithm A is a function that takes training set and returns a classifier $A : \mathcal{Z}^m \mapsto \mathcal{H}$

Choosing the classifier

- Designing a learning algorithm A $A : \mathcal{Z}^m \mapsto \mathcal{H}$.
- How should A choose classifier h from $\mathcal{H}$?
- We design a measure of quality of *h*
  - E.g. number of errors made
  - In general, we call it a loss $\ell : \mathcal{H} \times \mathcal{Z} \mapsto \mathbb{R}_+$
  - Loss function:
    - Input: a classifier *h*, and a labeled sample *z*
    - Output: a nonnegative real number
      - 0 – no loss – we're happy with the prediction
      - >0 – some loss – quantifies how unhappy we are
    - E.g. 0/1 loss (misclassification loss)

$$\ell(h, \mathbf{z}) = \mathrm{I}(h(\mathbf{x}) \neq y)$$

      - Returns 0 for correct prediction
      - Returns 1 for incorrect prediction

Choosing the classifier

- Designing a learning algorithm A $A : \mathcal{Z}^m \mapsto \mathcal{H}.$
- How should A choose classifier h from $\mathcal{H}$?
- A measure of quality of *h*

  $\mathcal{Z} = \mathcal{X} \times \{-1, +1\}$
  - Loss $\ell : \mathcal{H} \times \mathcal{Z} \mapsto \mathbb{R}_+$
  - Loss is defined for a single sample from *Z*
  - In general, our samples come from distribution D over space of samples *Z = feature space* x {-1,1}
  - The expected loss for distribution D is called *risk*

    $$R(h, D) = \mathbb{E}_{\mathbf{z}\sim D}[\ell(h, \mathbf{z})]$$
  - E.g. for 0/1 loss, $\ell(h, \mathbf{z}) = \mathrm{I}(h(\mathbf{x}) \neq y)$
    risk of classifier h is just the probability of making an error:

    $$R(h, D) = \mathbb{P}_{\mathbf{z}\sim D}[h(\mathbf{x}) \neq y]$$

Choosing the classifier

- Designing a learning algorithm A $A : \mathcal{Z}^m \mapsto \mathcal{H}$.
- A measure of quality of *h*
  - Define a loss $\ell : \mathcal{H} \times \mathcal{Z} \mapsto \mathbb{R}_+$
  - The expected loss for distribution D is called *risk*
$$R(h, D) = \mathbb{E}_{\mathbf{z} \sim D}[\ell(h, \mathbf{z})]$$
  - Distribution is unknown => risk is unknown
  - But we have the training set!
  - We can calculate the average loss on the training set
    - Empirical risk: $\widehat{R}_{S_m}(h) = \frac{1}{m} \sum_{i=1}^{m} \ell(h, \mathbf{z}_i)$
- How should A choose classifier h from $\mathcal{H}$?
- Empirical risk minimization: $A(S_m) = \arg\min_{h \in \mathcal{H}} \widehat{R}_{S_m}(h)$

Designing a classification method

1. Define the space of possible decision boundaries
   - What will be the form of classifiers?
     - Space of all possible lines/planes/hyperplanes
2. Define the loss/risk function
   - How to evaluate the quality of a specific classifier from the space of possible classifiers?
     - E.g. MSE loss and empirical risk associated with it
3. Define the method for minimizing the risk using data from the training set
   - How to reach a high-quality classifier?
     - E.g. gradient descent over the space of feature weights

Perceptron

- A simple analog to a biological neuron
  - Takes many inputs
  - Aggregates the information
  - Produces a response
  - The weights of each input are adjusted throughout time

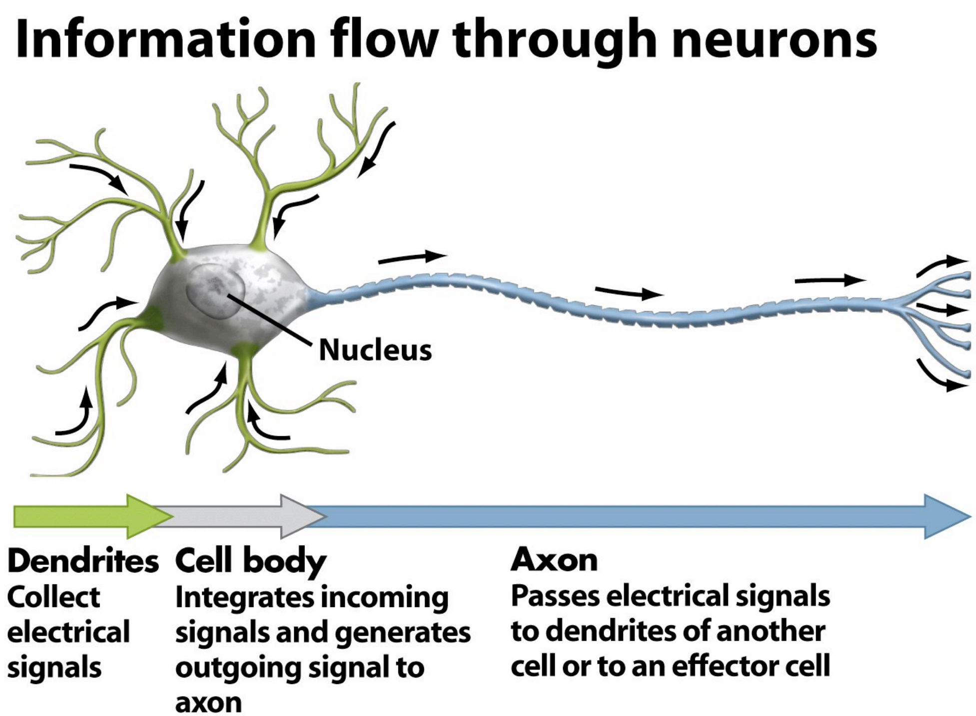
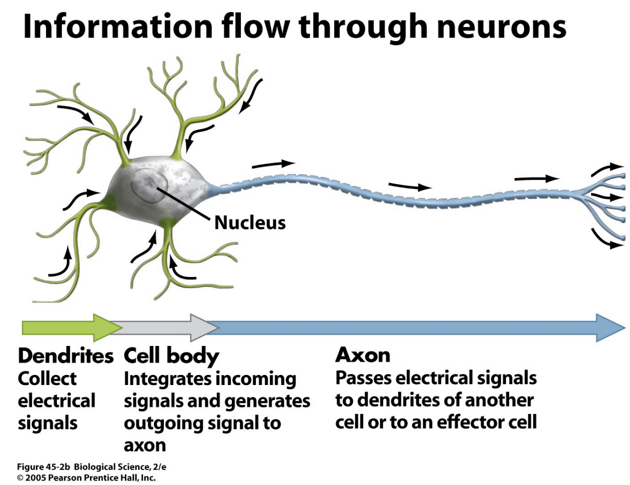


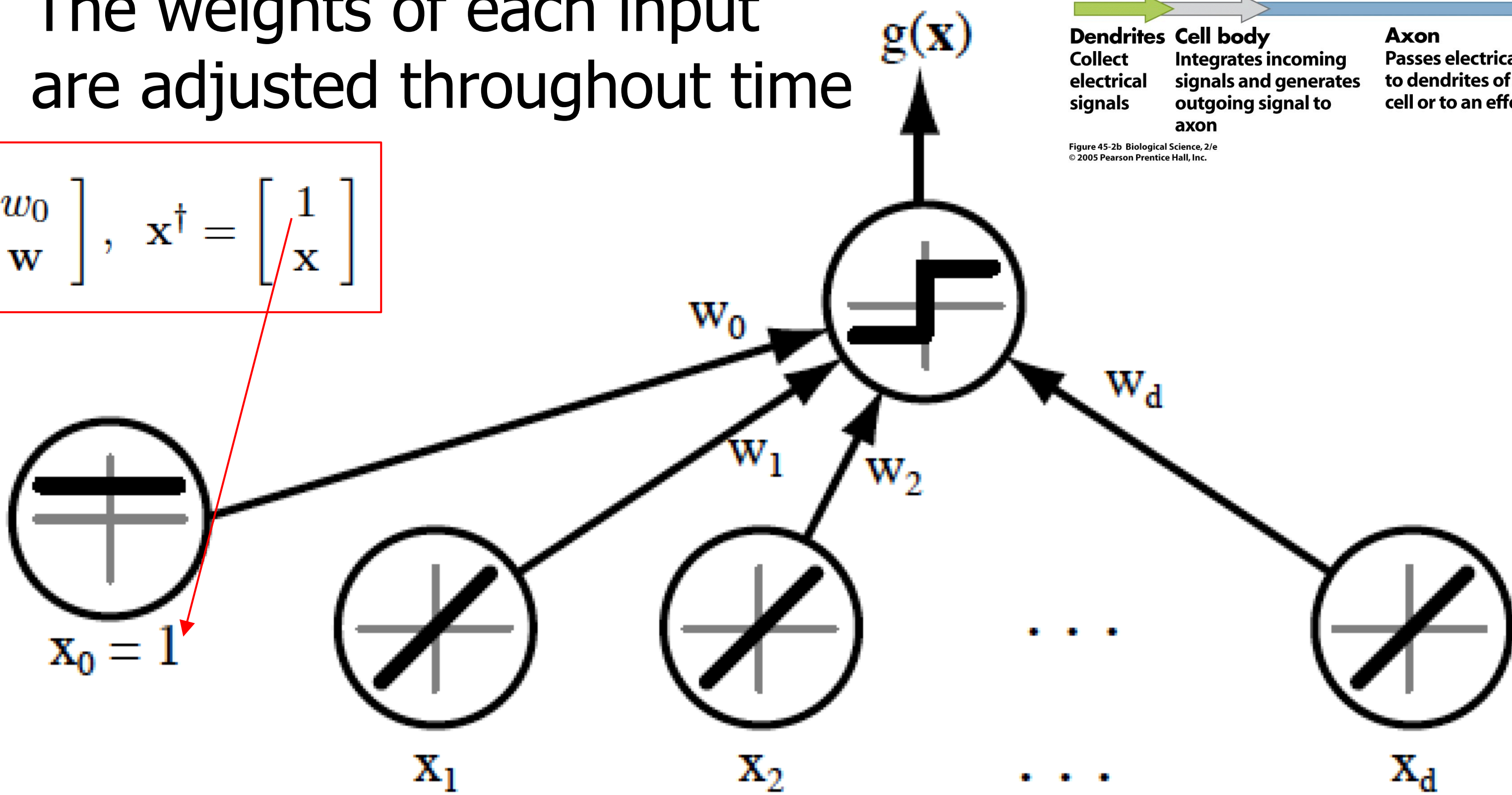
Figure 45-2b Biological Science, 2/e
© 2005 Pearson Prentice Hall, Inc.

Perceptron

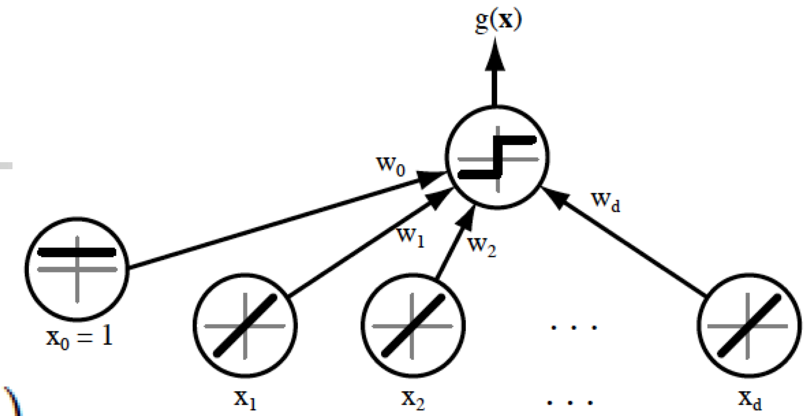
- A simple analog to a biological neuron
  - Takes many inputs
  - Aggregates the information
  - Produces a response
  - The weights of each input are adjusted throughout time



$$\mathbf{w}^{\dagger} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}, \quad \mathbf{x}^{\dagger} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$$



Perceptron



- Prediction will be made by evaluating a function: $h(\mathbf{x}) = \mathrm{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$
  - w and w_0 are unknown, need to be learned from examples
- **Algorithm for learning:**
  - **Set initial values of w, w_0**
  - **Adjust w, w_0 :**
    train the perceptron based on training set
  - **Training loop:**
    - **Present a sample x and predict class:**

$$h(\mathbf{x}) = \mathrm{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$$

    - **Compare true class *y* with predicted class h(x)**
    - **If prediction is right, go to next sample**
    - **If prediction is wrong, update weights**

$$\mathbf{w}^{\dagger}{}_{t+1} = \mathbf{w}^{\dagger}{}_{t} + 2cy\mathbf{x}^{\dagger}$$

$$\mathbf{w}^{\dagger} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}, \quad \mathbf{x}^{\dagger} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$$

- **Why does this algorithm make sense?**

Perceptron

- **Present a sample x and predict:**

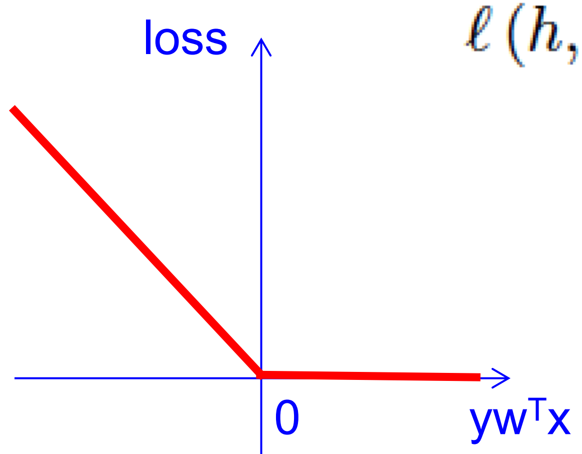
$$h(\mathbf{x}) = \operatorname{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$$

- **Compare true class *y* with predicted class h(x)**
- **If prediction is wrong, update weights**

$$\mathbf{w}^{\dagger}{}_{t+1} = \mathbf{w}^{\dagger}{}_{t} - 2cy\mathbf{x}^{\dagger}$$

- **Why this algorithm?**
- **Minimize empirical risk:** $\widehat{R}_{S_m}(h) = \frac{1}{m}\sum_{i=1}^{m}\ell(h, \mathbf{z}_i)$
- **For what loss?**
  - **Perceptron loss:**

$$\ell(h, \mathbf{z}) = \begin{cases} 0 & \text{for } y\mathbf{w}^{\dagger T}\mathbf{x}^{\dagger} \geq 0 \\ -y\mathbf{w}^{\dagger T}\mathbf{x}^{\dagger} & \text{for } y\mathbf{w}^{\dagger T}\mathbf{x}^{\dagger} < 0 \end{cases}$$



- **What's the behavior of yw^Tx ?**

Perceptron

- Present a sample x and predict:

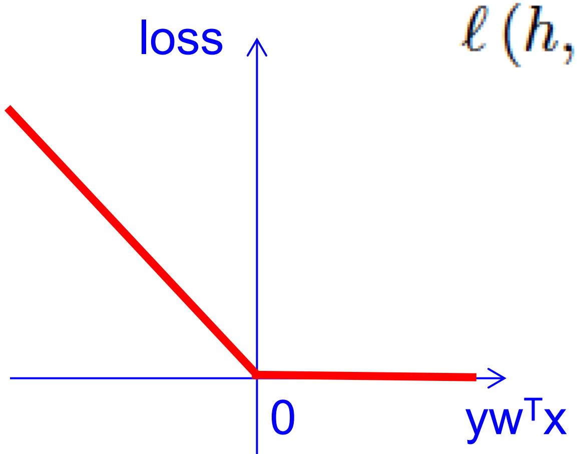
$$h(\mathbf{x}) = \operatorname{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$$

- Compare true class *y* with predicted class **h(x)**
- If prediction is wrong, update weights

$$\mathbf{w}^{\dagger}_{t+1} = \mathbf{w}^{\dagger}_{t} - 2cy\mathbf{x}^{\dagger}$$

- **Why this algorithm?**
- **Minimize empirical risk:** $\widehat{R}_{S_m}(h) = \frac{1}{m}\sum_{i=1}^{m}\ell(h, \mathbf{z}_i)$
- **For what loss?**
  - **Perceptron loss:**

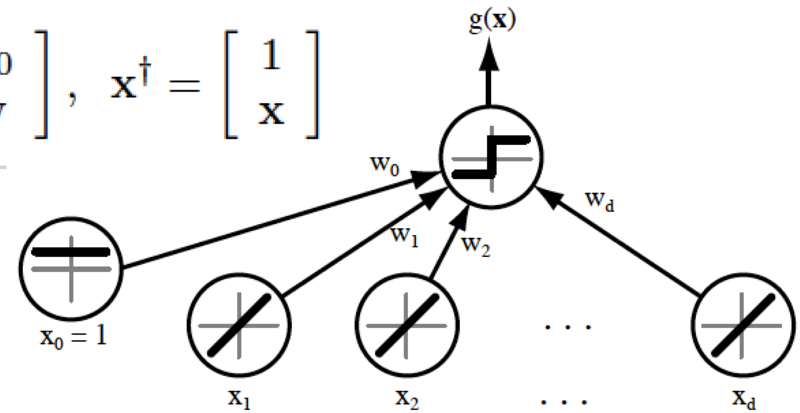
$$\ell(h, \mathbf{z}) = \begin{cases} 0 & \text{for } y\mathbf{w}^{\dagger T}\mathbf{x}^{\dagger} \geq 0 \\ -y\mathbf{w}^{\dagger T}\mathbf{x}^{\dagger} & \text{for } y\mathbf{w}^{\dagger T}\mathbf{x}^{\dagger} < 0 \end{cases}$$



- **$yw^Tx > 0$ for correct prediction**
  - **Same sign of w^Tx and y**
- **$yw^Tx < 0$ for incorrect prediction**
  - **Opposite sign of w^Tx and y**

Perceptron

$$\mathbf{w}^\dagger = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}, \quad \mathbf{x}^\dagger = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$$



- **Present a sample x and predict:**

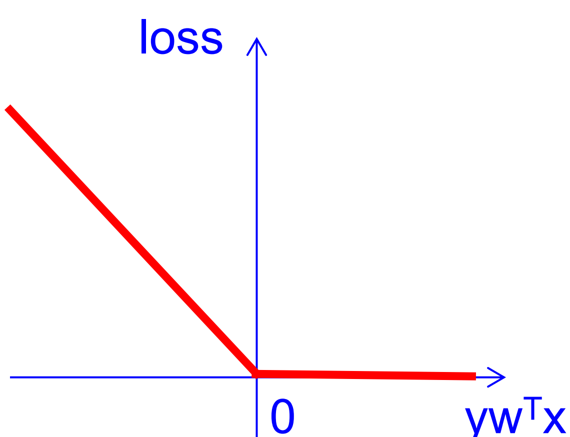
$$h(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T \mathbf{x} - w_0\right)$$

- **Compare true class *y* with predicted class h(x)**
- **If prediction is wrong, update weights**

$$\mathbf{w}^\dagger{}_{t+1} = \mathbf{w}^\dagger{}_t - 2cy\mathbf{x}^\dagger$$

- Minimize empirical risk of perceptron loss:
- By modifying w after seeing each sample z

$$\ell(h, \mathbf{z}) = \begin{cases} 0 & \text{for } y\mathbf{w}^{\dagger T}\mathbf{x}^\dagger \geq 0 \\ -y\mathbf{w}^{\dagger T}\mathbf{x}^\dagger & \text{for } y\mathbf{w}^{\dagger T}\mathbf{x}^\dagger < 0 \end{cases}$$

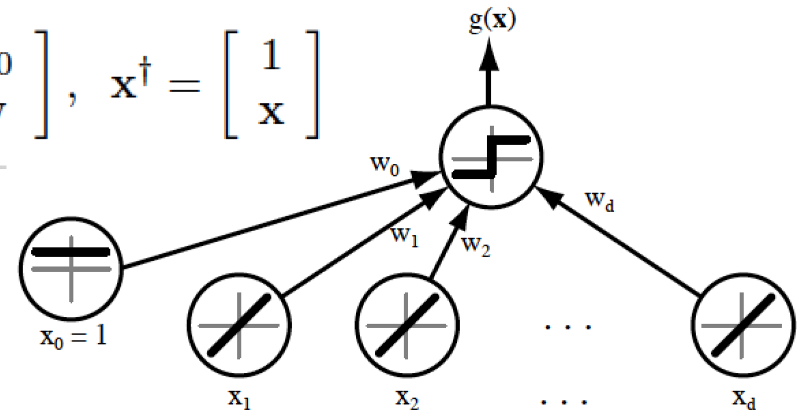


$$\mathbf{w}^\dagger{}_{t+1} = \mathbf{w}^\dagger{}_t - 2c \left.\frac{\partial \ell(h, \mathbf{z})}{\partial \mathbf{w}^\dagger}\right|_{\mathbf{w}^\dagger = \mathbf{w}^\dagger{}_t}$$

$$\mathbf{w}^\dagger{}_{t+1} = \mathbf{w}^\dagger{}_t + 2cy\mathbf{x}^\dagger$$

Perceptron

$$\mathbf{w}^{\dagger} = \begin{bmatrix} w_0 \\ \mathbf{w} \end{bmatrix}, \quad \mathbf{x}^{\dagger} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$$



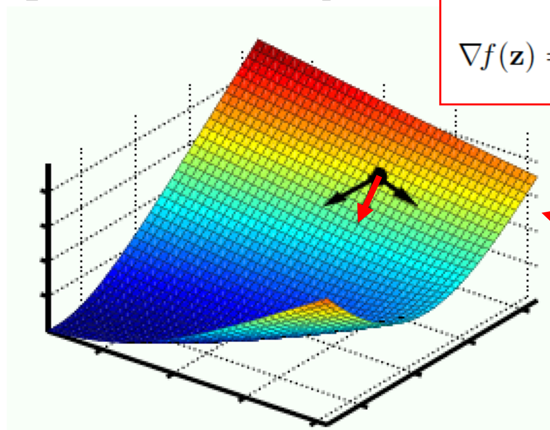
- **Present a sample x and predict:**

$$h(\mathbf{x}) = \mathrm{sign}\left(\mathbf{w}^T\mathbf{x} - w_0\right)$$

- **Compare true class *y* with predicted class h(x)**
- **If prediction is wrong, update weights**

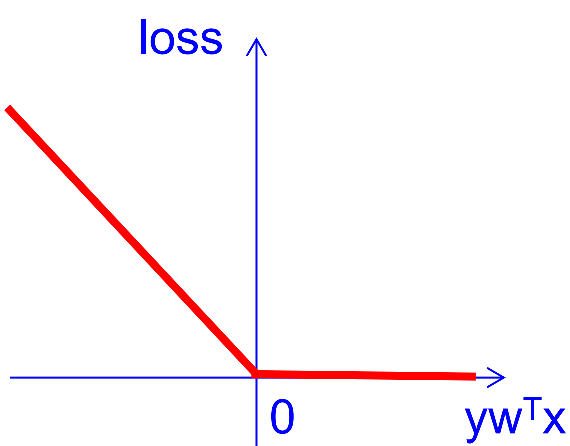
$$\mathbf{w}^{\dagger}{}_{t+1} = \mathbf{w}^{\dagger}{}_{t} + 2cy\mathbf{x}^{\dagger}$$

$$\nabla f(\mathbf{z}) = \left(\frac{\partial}{\partial x_1} f(\mathbf{x})|_{\mathbf{x}=\mathbf{z}}, \ldots, \frac{\partial}{\partial x_n} f(\mathbf{x})|_{\mathbf{x}=\mathbf{z}}\right)$$



- Minimize empirical risk of perceptron loss:
- By modifying w after seeing each sample z

–gradient (stochastic)



$$\mathbf{w}^{\dagger}{}_{t+1} = \mathbf{w}^{\dagger}{}_{t} - 2c \left.\frac{\partial \ell(h, \mathbf{z})}{\partial \mathbf{w}^{\dagger}}\right|_{\mathbf{w}^{\dagger}=\mathbf{w}^{\dagger}{}_{t}}$$

$$\ell(h, \mathbf{z}) = \begin{cases} 0 & \text{for } \ y\mathbf{w}^{\dagger T}\mathbf{x}^{\dagger} \geq 0 \\ -y\mathbf{w}^{\dagger T}\mathbf{x}^{\dagger} & \text{for } \ y\mathbf{w}^{\dagger T}\mathbf{x}^{\dagger} < 0 \end{cases}$$

(stochastic) gradient descent

- Our function to be minimized is (proportional to) empirical risk:
  $f(w)=\sum_i f_i(w) = \sum_i loss(y_i,x_i,w)$

- Gradient descent (batch learning):
  - in a loop, modify w to minimize f(w)
    - that means we need to evaluate the sum over all samples
    - we use gradient of $\sum_i loss(y_i,x_i,w)$ w.r.t. w
  - We present all samples, and only then we update the weights

- Stochastic gradient descent (online learning):
  - in a loop, randomly pick a single i, modify w to minimize $f_i(w)$
    - that means we need to evaluate only one sample at a time
    - we use gradient of $loss(y_i,x_i,w)$ w.r.t. w
  - We present one sample, and immediately update the weights
  - Each step is then a "noisy" gradient, but the noise should average itself out over many iterations
- mini-batch stochastic gradient descent :
  - In between the above: $\sum_i loss(y_i,x_i,w)$ over e.g. randomly chosen 64 samples from the whole training set