WARWICK
THE UNIVERSITY OF WARWICK

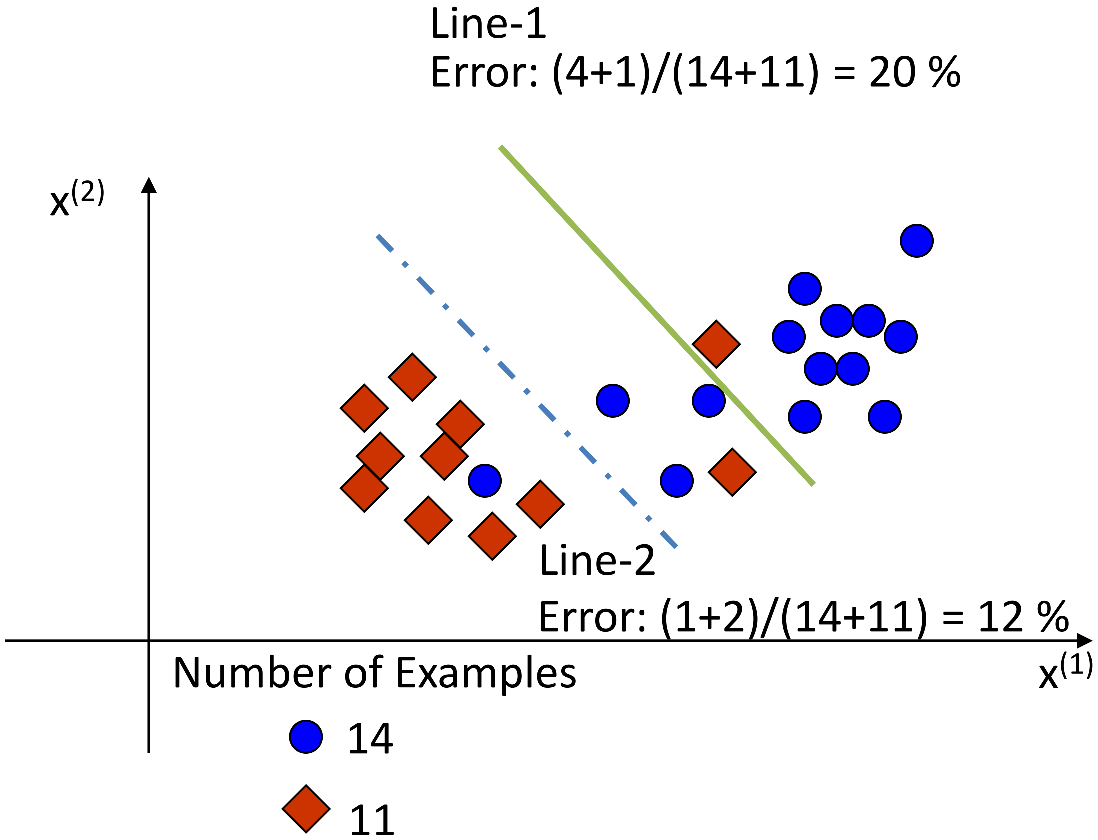
From Lines to Perceptrons

Dr. Fayyaz Minhas

Department of Computer Science
University of Warwick
https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs909/

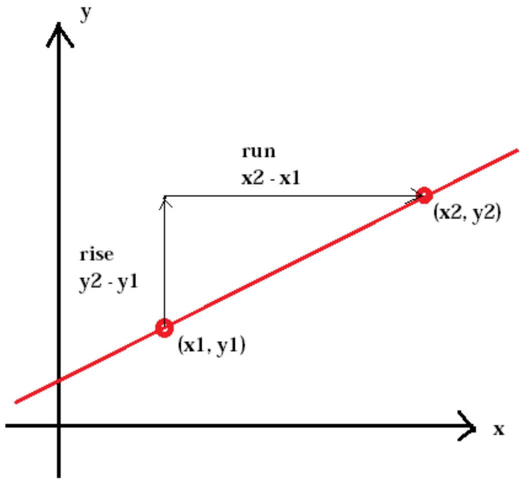
Another way of looking at Classification

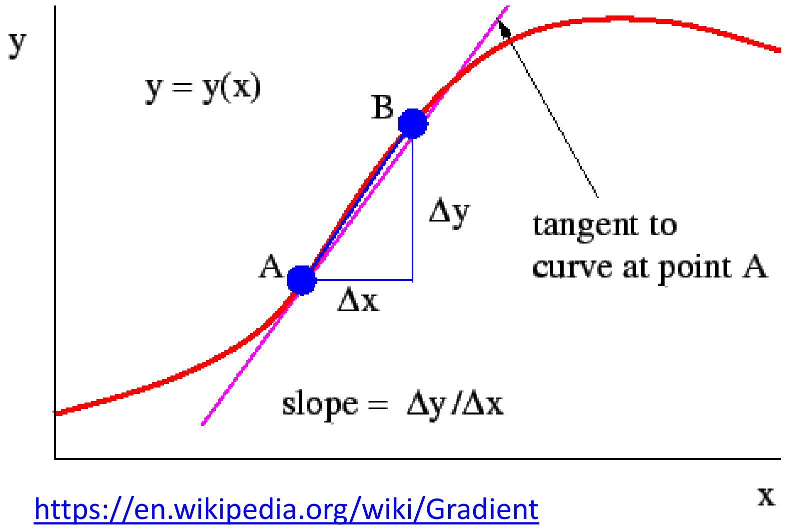
- We would like to minimize the number of errors a discriminant function $f(\boldsymbol{x})$ makes
- **Representation**: Assume we look at only linear functions
  $$f(\boldsymbol{x};\boldsymbol{w}) = w_0 + w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_d x^{(d)} = 0$$
- **Evaluation**: We need to define error that a particular $f(x;w)$ makes
- **Optimization**: We need to minimize the error by tuning $\boldsymbol{w}$



Preliminaries

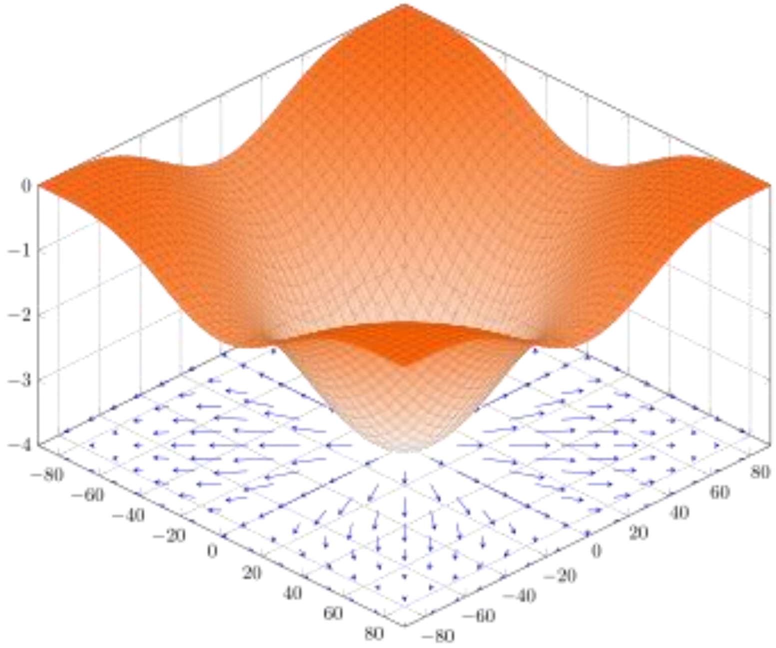
- Gradients





https://en.wikipedia.org/wiki/Gradient

$$\nabla f(\boldsymbol{x}) = \begin{bmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x^{(1)}} \\ \frac{\partial f(\boldsymbol{x})}{\partial x^{(2)}} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x^{(d)}} \end{bmatrix}$$



$f(x,y) = -(\cos^2 w + \cos^2 y)^2$

Finding minima and maxima of functions

- Given a function f(w)
- Take the derivative
- Substitute the derivative to zero
- Solve for x when $\frac{df}{dw} = 0$

$$f(w) = (w - 0.5)^2$$
$$\frac{df}{dw} = 2(w - 0.5) = 0$$
$$w^* = 0.5$$

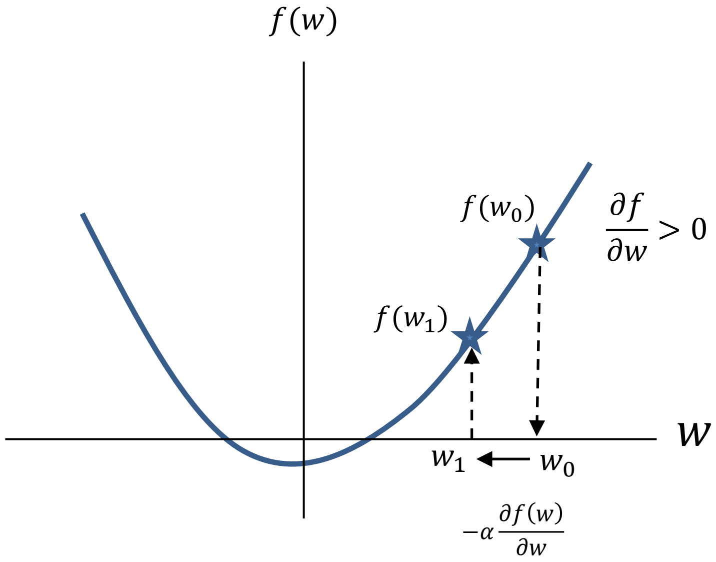
- Works when we can solve for w

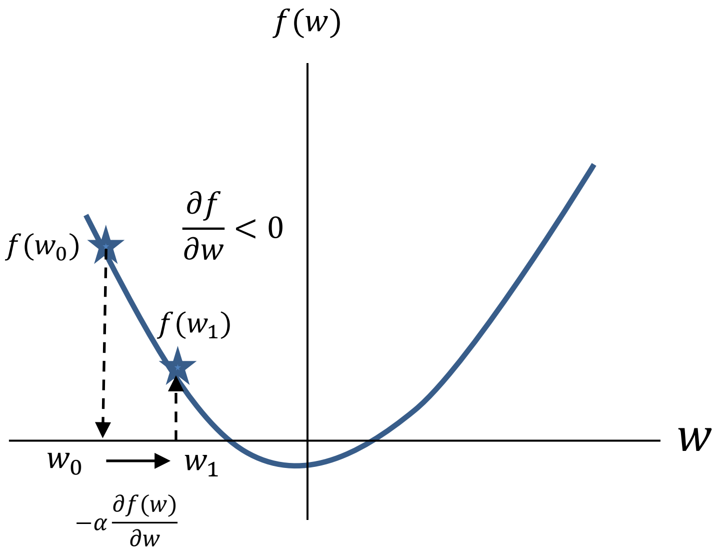
$$f(w) = (w - 0.5)^2 + sin(4w)$$
$$\frac{df}{dw} = 2(w - 0.5) + 4\cos(4w) = 0$$
$$w^* = ?$$

Preliminaries: Gradient Descent

- In order to find the minima of a function, keep taking steps along a direction opposite to the gradient of the function

$$\boldsymbol{w}^{(\boldsymbol{k}+\mathbf{1})} = \boldsymbol{w}^{(\boldsymbol{k})} - \alpha \nabla f(\boldsymbol{w}^{(\boldsymbol{k})})$$





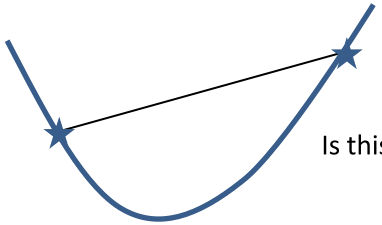
GD Implementation

```python
import numpy as np

def gd(fxn,dfxn,w0=0.0,lr = 0.01,eps=1e-4,nmax=1000, history = True):
    """
    Implementation of a gradient descent solver.
        fxn: function returns value of the target function for a given w
        dfxn: gradient function returns the gradient of fxn at w
        w0: initial position [Default 0.0]
        lr: learning rate [0.001]
        eps: min step size threshold [1e-4]
        nmax: maximum number of iters [1000]
        history: whether to store history of x or not [True]
    Returns:
        w: argmin_x f(w)
        converged: True if the final step size is less than eps else false
        H: history
    """
    H = []
    w = w0
    if history:
        H = [[w,fxn(w)]]
    for i in range(nmax):
        dw = -lr*dfxn(w) #gradient step
        if np.linalg.norm(dw)<eps: # we have converged
            break
        if history:
            H.append([w+dw,fxn(w+dw)])
        w = w+dw #gradient update
    converged = np.linalg.norm(dw)<eps
    return w,converged,np.array(H)
```

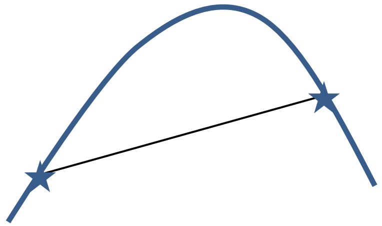
```python
if __name__=='__main__':
    import matplotlib.pyplot as plt
    def myfunction(w):
        z = (w-0.5)**2#+np.sin(4*w)
        return z
    def mygradient(w):
        dz = 2*(w-0.5)#+4*np.cos(4*w)
        return dz


    wrange = np.linspace(-3,3,100)
    #select random initial point in the range
    w0 = np.min(wrange)+(np.max(wrange)-np.min(wrange))*np.random.rand()

    w,c,H = gd(myfunction,mygradient,w0=w0,lr = 0.01,eps=1e-4,nmax=1000, history = True)

    plt.plot(wrange,myfunction(wrange)); plt.plot(wrange,mygradient(wrange));
    plt.legend(['f(w)','df(w)'])
    plt.xlabel('w');plt.ylabel('value')
    s = 'Convergence in '+str(len(H))+' steps'
    if not c:
        s = 'No '+s
    plt.title(s)
    plt.plot(H[0,0],H[0,1],'ko',markersize=10)
    plt.plot(H[:,0],H[:,1],'r.-')
    plt.plot(H[-1,0],H[-1,1],'k*',markersize=10)
    plt.grid(); plt.show()
```

Convex vs. non-convex functions

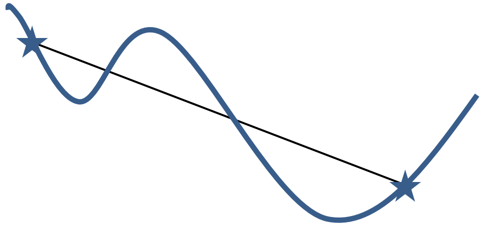
- If you draw a line between "any" two points on a function and the line always remains above or on the function, then that function is called convex function
  - Strict Convexity
- Convex functions will have a single minima



Is this convex?



Is this convex?



Is this convex?

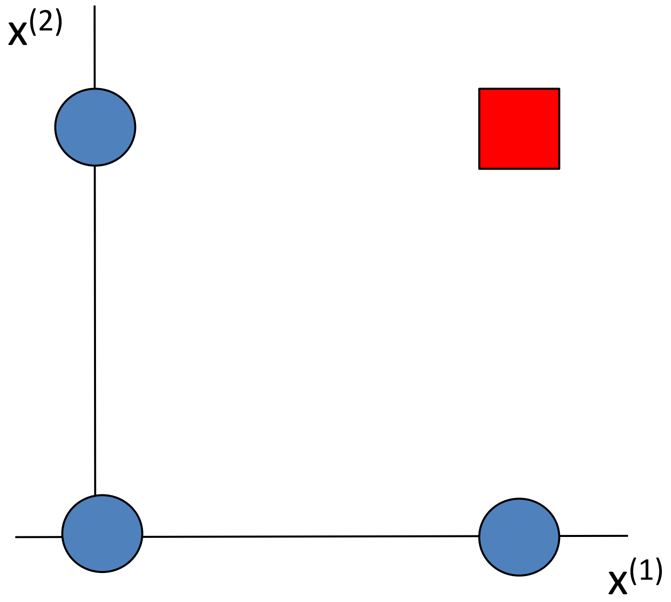
https://en.wikipedia.org/wiki/Convex_function

Building Linear Discriminants

- Representation
  - Features
  - Linear Function

$$f(\mathbf{x};\boldsymbol{w}) = w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_d x^{(d)} + b = 0$$

- Evaluation
  - Misclassification
- Optimization
  - Find a line that minimizes misclassifications
  - How done: Visual reckoning / Constraint Satisfaction
- Why Study Linear Models?



(1,1): $w_1(1.0)+w_2(1.0)+b > 0$

(1,0): $w_1(1.0)+w_2(0.0)+b < 0$

(0,1): $w_1(0.0)+w_2(1.0)+b < 0$

(0,0): $w_1(0.0)+w_2(0.0)+b < 0$

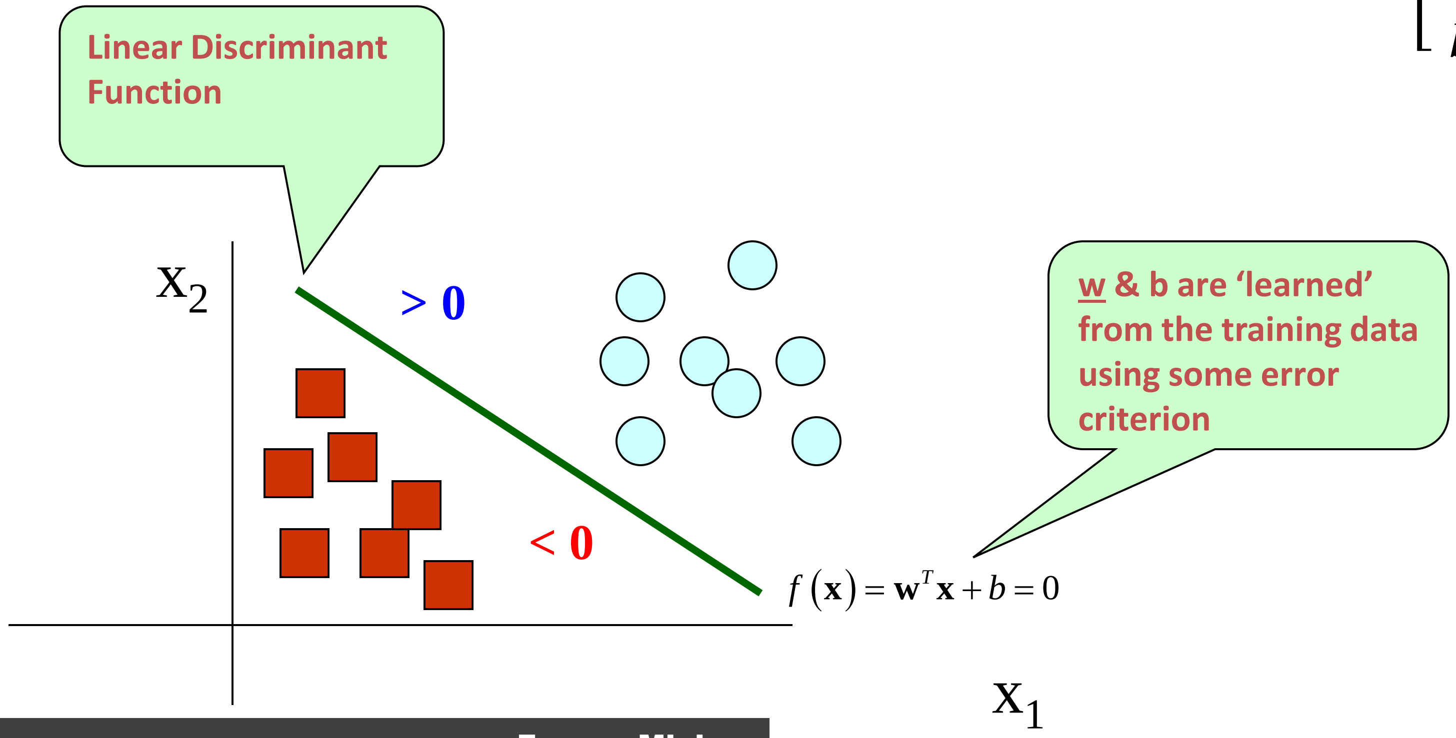
A more mathematical look

- Linear Discriminants
- The linear discriminant function is given by

$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \quad \boldsymbol{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(d)} \end{bmatrix}$$

$$f(\mathbf{x}; \boldsymbol{w}) = w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_d x^{(d)} + b = \mathbf{w}^T \mathbf{x} + b$$
$$f(\mathbf{x}'; \boldsymbol{w}') = w_1 x^{(1)} + w_2 x^{(2)} + \cdots + w_d x^{(d)} + b = \mathbf{w}'^T \mathbf{x}'$$

$$\mathbf{w}' = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \\ b \end{bmatrix} \quad \boldsymbol{x}' = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(d)} \\ 1 \end{bmatrix}$$



Classification Loss Function

- A misclassification is an error
  - If a training example has a label of $y = +1$,then its discriminant function score $f(\boldsymbol{x})$ should be _____
  - If a training example has a label of $y = -1$,then its discriminant function score $f(\boldsymbol{x})$ should be _____
  - Thus, we have an error whenever: ___________

Classification Loss Function

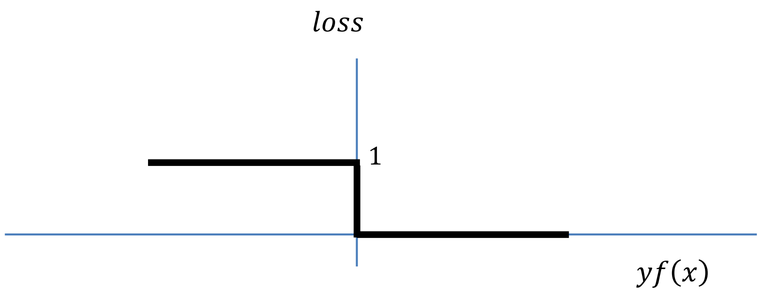
- A misclassification is an error
  - If a training example has a label of $y = +1$,then its discriminant function score $f(\boldsymbol{x})$ should be > 0

  - If a training example has a label of $y = -1$,then its discriminant function score $f(\boldsymbol{x})$ should be < 0

  - Thus, we have an error whenever: $yf(\boldsymbol{x}) < 0$

0-1 Loss/Error

- Consider a single example:
  - Our error function is: $l(f(x), y) = \begin{cases} 0 & yf(x) > 0 \\ 1 & yf(x) \leq 0 \end{cases}$

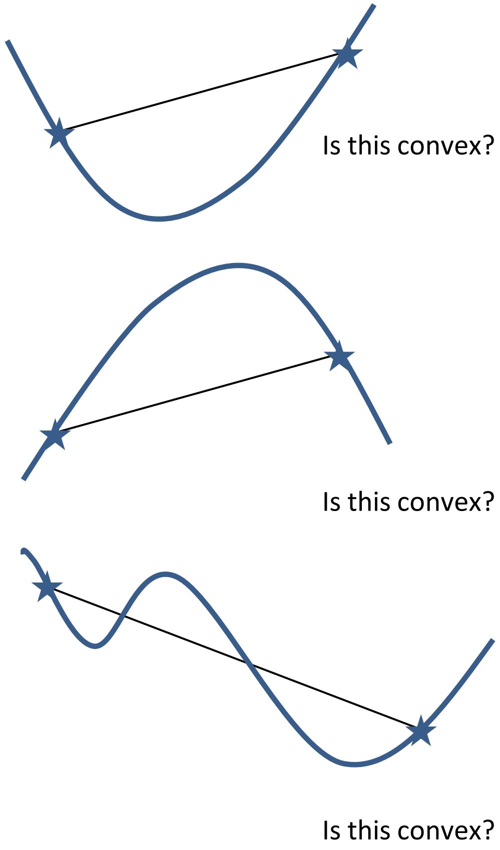


Formally called the zero-one loss function

0-1 Misclassification Error

- We want to find the parameters of the discriminant that minimize the loss for all examples in training
- Issues with 0-1 loss
  - Non Differentiable
  - Leads to poor optimization
- We need a "surrogate" or approximation of the loss
  - Should be continuous
  - Should be an over-approximation of the 0-1 loss
    - Generates at least as much error as the 0-1 loss would
  - Should be convex
    - Convex loss function leads to convex optimization problems which are easier to solve as they have a single minima

Convexity: If a line connecting two points on a curve lies on or above the curves at all times

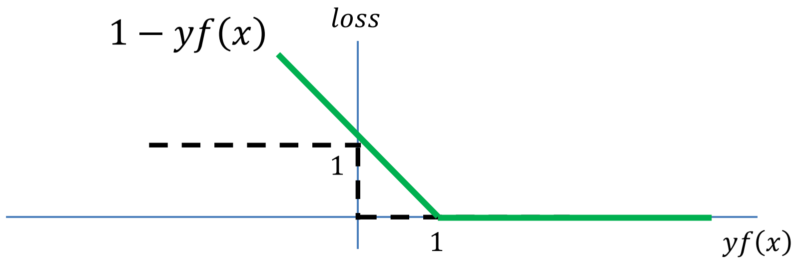


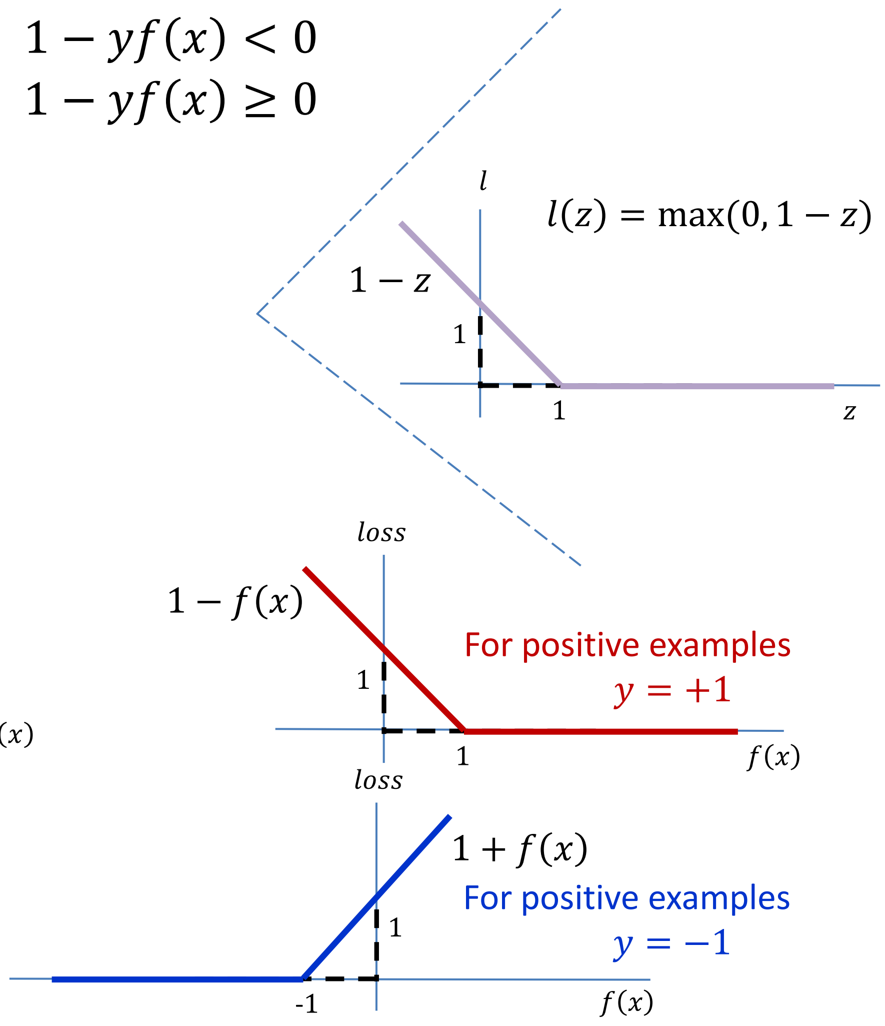
Surrogate Classification Loss

$$l(f(x),y) = \begin{cases} 0 & yf(x) > 1 \\ 1 - yf(x) & yf(x) \le 1 \end{cases} = \begin{cases} 0 & 1 - yf(x) < 0 \\ 1 - yf(x) & 1 - yf(x) \ge 0 \end{cases}$$

OR

$$l(f(x),y) = \max(0,1 - yf(x))$$





- **Hinge Loss Function**
  - A convex over-approximation of the 0-1 loss
  - Adds some “margin” of error to the classification
    - Prediction label can be +1 or -1 depending upon whether $f(x) > 0$ or $f(x) < 0$
    - However, we incur a loss if for positive training examples $f(x) < 1$ or for negative examples $f(x) > -1$

Optimization

$$\min_w L(\boldsymbol{X},\boldsymbol{Y};\boldsymbol{w}) = \sum_{i=1}^{N} \max\{0, 1 - y_i f(\boldsymbol{x}_i;\mathbf{w})\}$$

- How can we solve it?
  - Take the derivative and substitute to zero
  - How else can we solve it?
    - Use gradient descent

Optimization

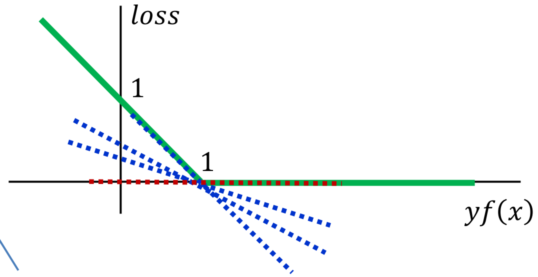
$$\min_w L(\boldsymbol{X},\boldsymbol{Y};\boldsymbol{w}) = \sum_{i=1}^{N} l(f(\boldsymbol{x}_i;\mathbf{w})),y_i) = \sum_{i=1}^{N} \max\{0,1-y_i f(\boldsymbol{x}_i;\mathbf{w})\}$$

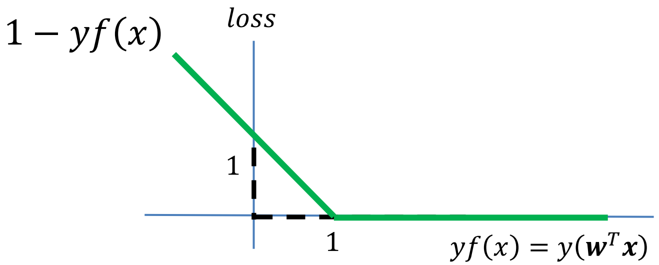
$$\frac{\partial L}{\partial \boldsymbol{w}} = \sum_{i=1}^{N} \frac{\partial l(f(\boldsymbol{x}_i;\mathbf{w})),y_i)}{\partial \boldsymbol{w}}$$

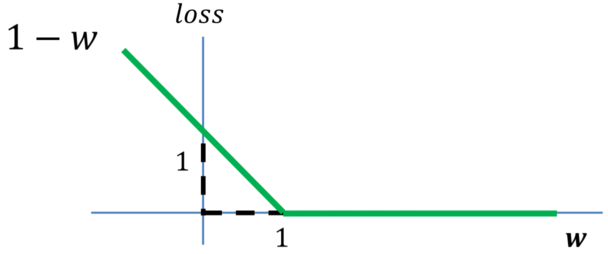
$$\frac{\partial}{\partial \boldsymbol{w}} \max\{0,1-y(\boldsymbol{w}^T\boldsymbol{x})\} = \begin{cases} 0 & 1-yf(\boldsymbol{x};\mathbf{w}) < 0 \\ -y\boldsymbol{x} & else \end{cases} = \begin{cases} -y\boldsymbol{x} & l(f(\boldsymbol{x};\mathbf{w})),y) > 0 \\ 0 & else \end{cases}$$

What happens when $yf(x) = 1$ where the function has a "kink"?

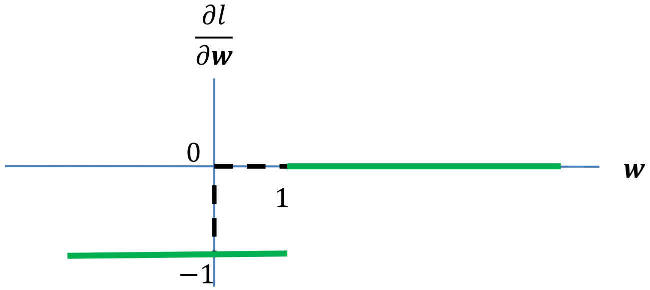
There, we can choose to define the "sub"-gradient to be the slope of any line that lies below or on the loss function itself (see dotted lines below). Consequently defining $\frac{\partial L}{\partial w}|_{yf(x)=1} = \mathbf{0}$ should work (slope of red line).







For a simple example in which $x = 1, y = 1$



For a simple example in which $x = 1, y = 1$

Algorithm

- Given:
  - Training Examples: $\{(\boldsymbol{x}_i, y_i) | i = 1 \dots N\}, \mathrm{y_i} \in \{-1, +1\}$
  - Learning rate (step size): α
- Initialize $w^{(0)}$ at random
- Until Convergence ($k = 1 \dots K$ epochs)
  - For $i = 1 \dots N$
    - Pick example $\boldsymbol{x}_i$ with label y_i
    - Compute $f(\boldsymbol{x}_i) = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})^{\boldsymbol{T}}} \boldsymbol{x}_{\boldsymbol{i}}$
    - If $y_i f(\boldsymbol{x}_i) < 1$ then update weight vector using gradient descent

$$\boldsymbol{w}^{(\boldsymbol{k})} = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})} - \alpha \nabla \boldsymbol{l}\left(\boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})}\right) = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})} - \alpha(-y_i \boldsymbol{x}_i) = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})} + \alpha y_i \boldsymbol{x}_i$$

    - Check for convergence to stop

$$\nabla_{\boldsymbol{w}} \max\{0, 1 - y(\boldsymbol{w}^T \boldsymbol{x})\} = \begin{cases} 0 & 1 - y f(\boldsymbol{x}; \mathbf{w}) < 0 \\ -y\boldsymbol{x} & else \end{cases}$$

REO For Perceptron

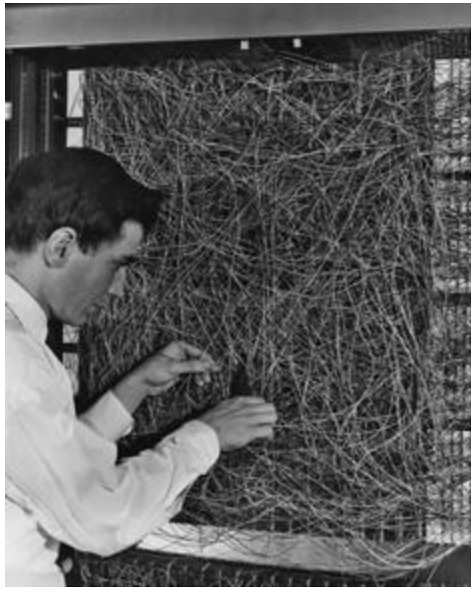
- Representation
  - Features
  - Discriminant
    - Linear: $f(\boldsymbol{x}_i; \boldsymbol{w}) = \boldsymbol{w}^{\boldsymbol{T}} \boldsymbol{x}_{\boldsymbol{i}}$
- Evaluation
  - 0/1 (Step) Loss
  - Hinge Loss
- Optimization
  - Using Gradient Descent

- Given:
  - Training Examples: $\{(\boldsymbol{x}_i, y_i) | i = 1 \ldots N\}, \mathrm{y_i} \in \{-1, +1\}$
- Initialize $w^{(0)}$ at random
- Until Convergence
  - For i = 1…N
    - Pick example $\boldsymbol{x}_i$ with label y_i
    - Compute $f(\boldsymbol{x}_i) = \mathbf{w}^{(k)^T} \mathbf{x} + b$
    - If $y_i f(\boldsymbol{x}_i) < 1$ then update your weight vector using gradient descent

$$\boldsymbol{w}^{(\boldsymbol{k})} = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})} - \alpha \nabla \boldsymbol{l}\left(\boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})}\right) = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})} - \alpha(-y_i \boldsymbol{x}_i) = \boldsymbol{w}^{(\boldsymbol{k}-\mathbf{1})} + \alpha y_i \boldsymbol{x}_i$$

Perceptron

- A simpler version of this algorithm is called: Perceptron
  - It updated weights whenever an example was misclassified $(y_i f(\boldsymbol{x}_i) < 0)$ instead of when $y_i f(\boldsymbol{x}_i) < 1$
  - Rosenblatt (1962)
  - Minsky and Papert (1969, 1988)
  - This algorithm provides theoretical guarantees of convergence to a correct separating boundary
    - If the data is linearly separatable and you allow the pereceptron algorithm to run long enough, you will find the separating line!
    - **Perceptron Learning Rule Convergence Theorem**

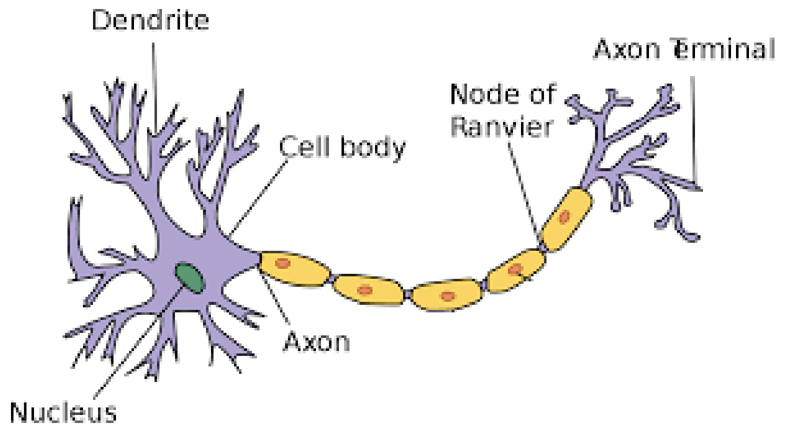


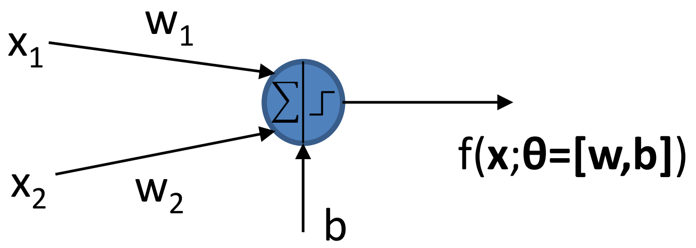
Frank Rosenblatt
July 11, 1928 – July 11, 1971

Marvin Minsky
Aug. 9, 1927 – Jan. 24, 2016

Perceptron

- One of the first “artificial” neural networks





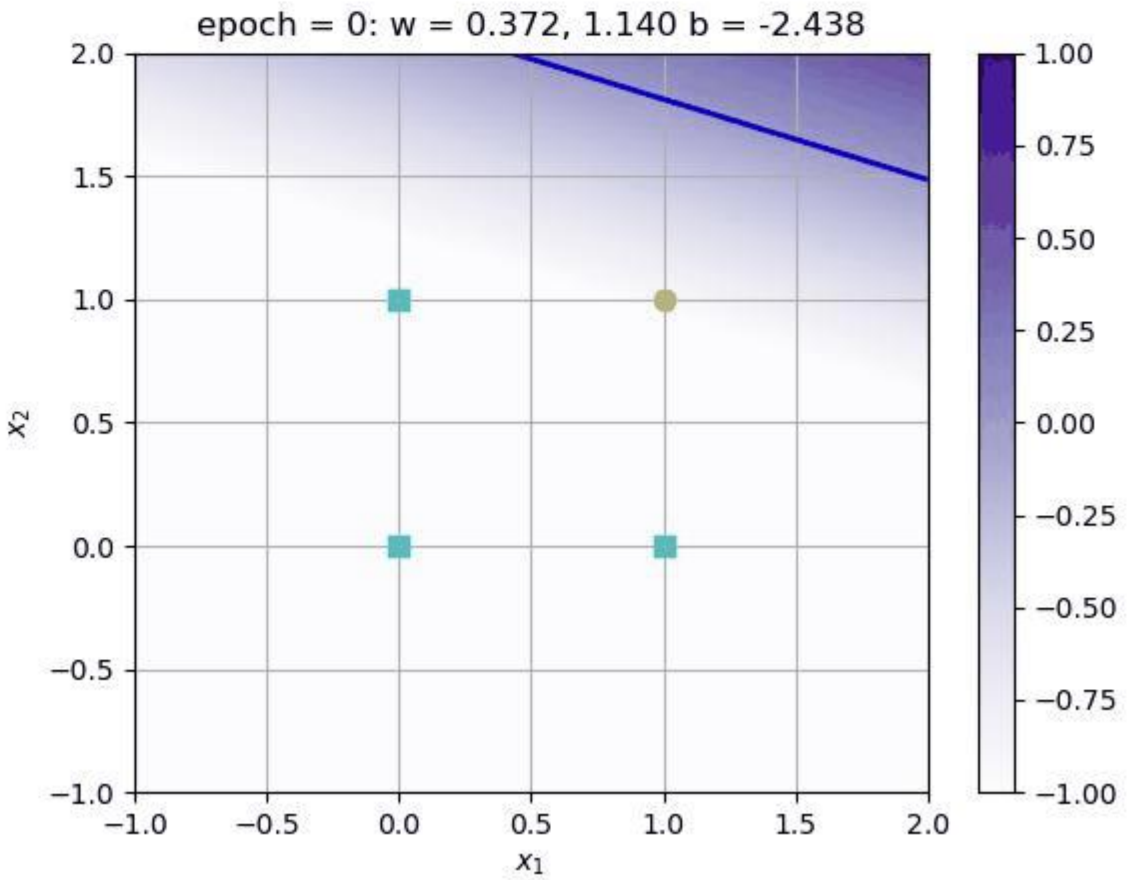
$$\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix} \boldsymbol{x} = \begin{bmatrix} x^{(1)} \\ x^{(2)} \\ \vdots \\ x^{(d)} \end{bmatrix}$$

$$f(\boldsymbol{x};\boldsymbol{\theta}) = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + b$$

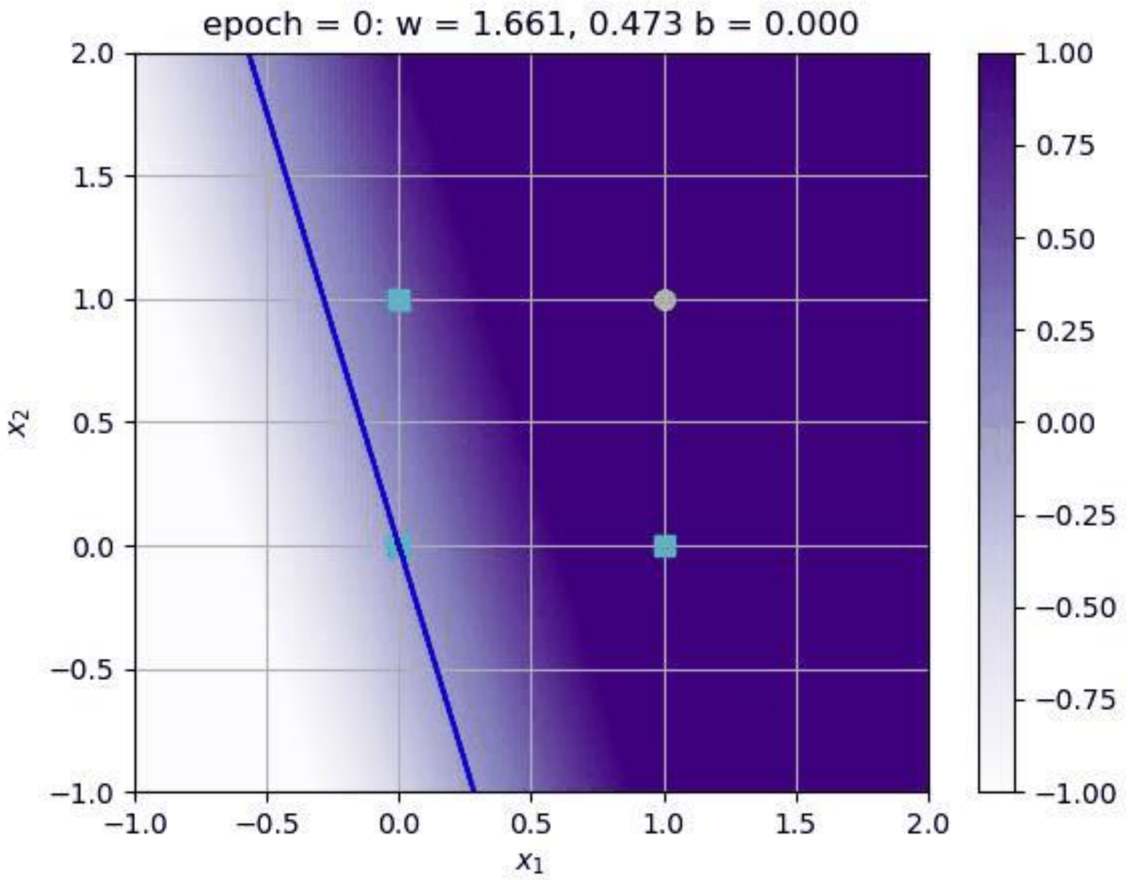
Coding Exercise

```
import numpy as np
import matplotlib.pyplot as plt
import itertools

class Perceptron:

    def __init__(self,alpha = 0.1, epochs = 200):
        self.alpha = alpha
        self.epochs = epochs
        self.W = np.array([0])
        self.bias = np.random.randn()
        self.Lambda = 0.5
    def fit(self,Xtr,Ytr):
        d = Xtr.shape[1]
        self.W = np.random.randn(d)
        for e in range(self.epochs):
            finished = True
            for i,x in enumerate(Xtr):
                if Ytr[i]*self.predict(np.atleast_2d(x))<1:
                    finished = False
                    self.W += self.alpha*Ytr[i]*x
                    self.bias += self.alpha*Ytr[i]
            if finished: break

    def score(self,x):
        return np.dot(x,self.W) + self.bias

    def predict(self,x):
        return np.sign(self.score(x))
```

```
if __name__=='__main__':
    from plotit import plotit
    Xtr = np.array([[-1,0],[0,1],[4,4],[2,3]])
    ytr = np.array([-1,-1,+1,+1])
    clf = Perceptron()
    clf.fit(Xtr,ytr)
    z = clf.score(Xtr)
    print("Prediction Scores:",z)
    y = clf.predict(Xtr)
    print("Prediction Labels:",y)
    plotit(Xtr,ytr,clf=clf.score,conts=[0],
           extent = [-5,+5,-5,+5])
```



https://github.com/foxtrotmike/CS909/blob/master/perceptron_video.py


epoch = 0: w = 1.661, 0.473 b = 0.000
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_1
x_2
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00

End of Lecture

We want to make a machine that will be proud of us.

- Danny Hillis