Deep Learning 1

Jian Li

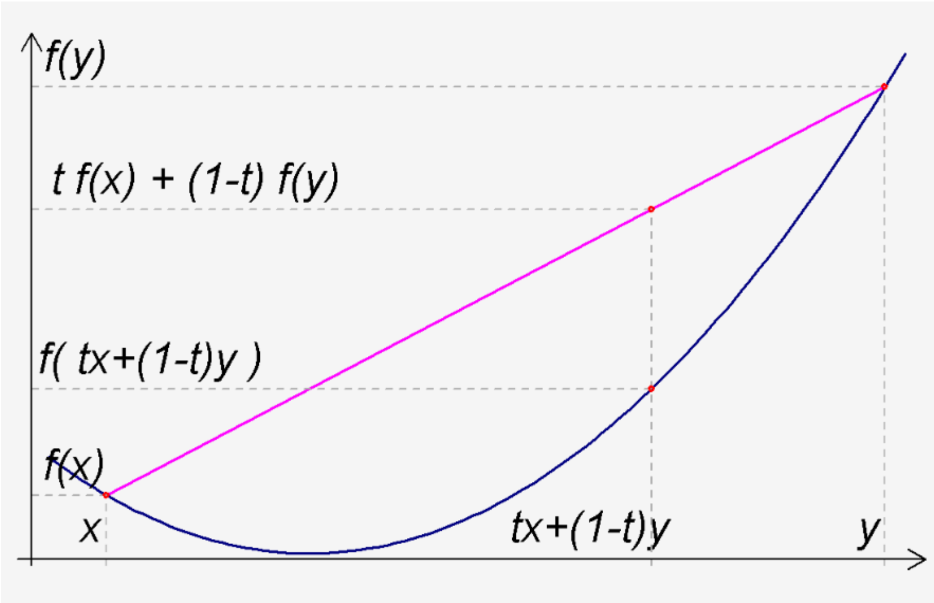
IIIS, Tsinghua University

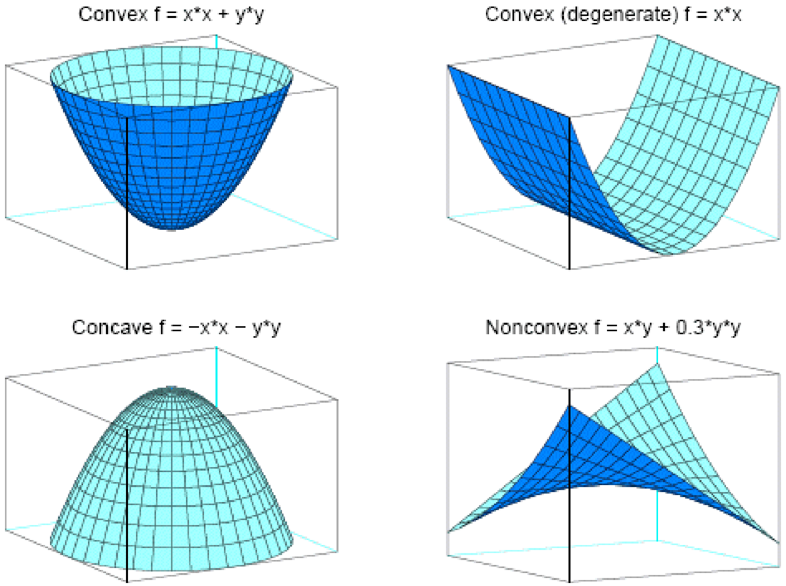
Optimization Basics

Convex Functions

- $f\left(\frac{x+y}{2}\right) \leq \frac{1}{2}(f(x) + f(y))$
- $f(tx + (1-t)y) \leq tf(x) + (1-t)f(y), \quad \forall t \in [0,1]$

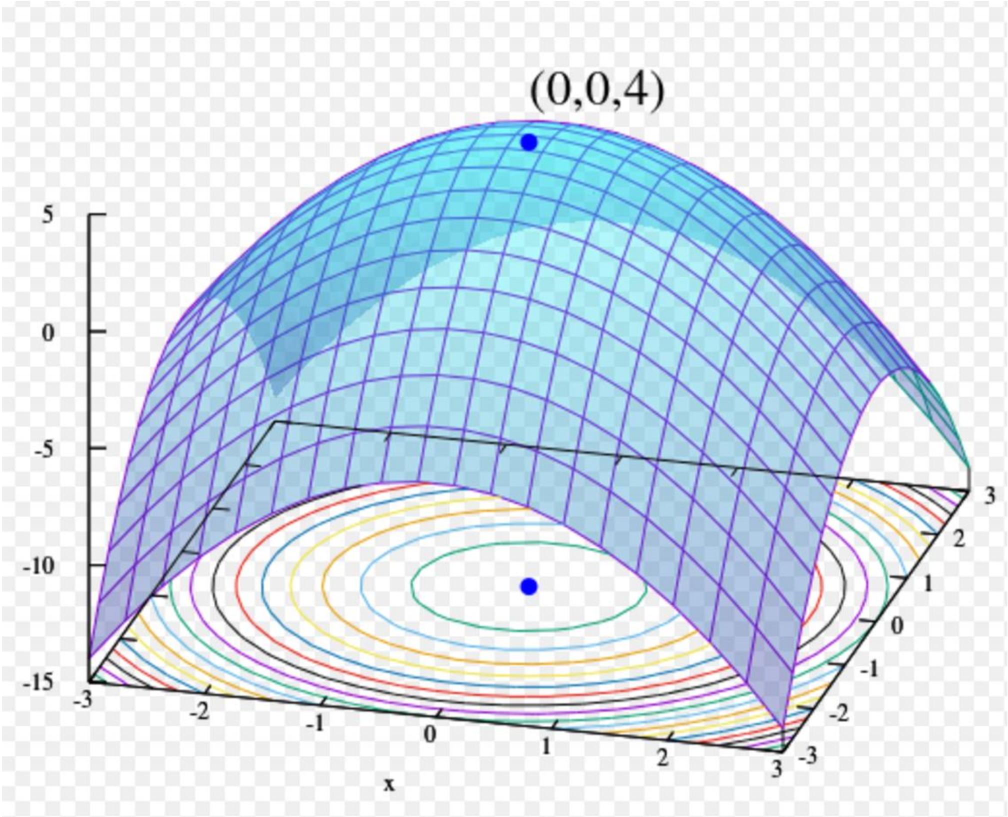
(the above two definitions are equivalent for continuous functions)



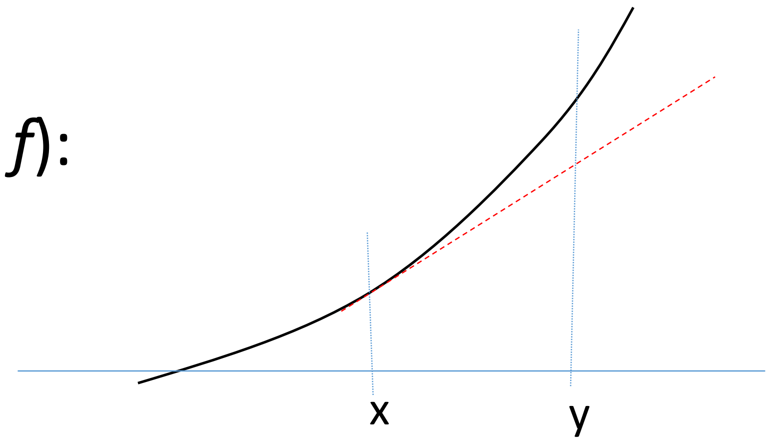


Concave functions

- f is concave if $-f$ is convex



Convex Functions



- First order condition (for differentiable *f*):
  - $f(y) \geq f(x) + \nabla f(x)(y - x)$
  - $\nabla f(x) = \left[\frac{\partial f}{\partial x_i}\right]_i$
- Second order condition (for twice-differentiable *f*):
  - Hessian matrix $\nabla^2 f(x)$ is positive semidefinite (psd)
  - $\nabla^2 f(x) = \left[\frac{\partial^2 f}{\partial x_i \partial x_j}\right]_{i,j}$
  - $\nabla^2 f(x)$ specifies the local 2^{nd} order shape of f (how f matches a quadratic function locally)
  - Recall Tayler expansion：

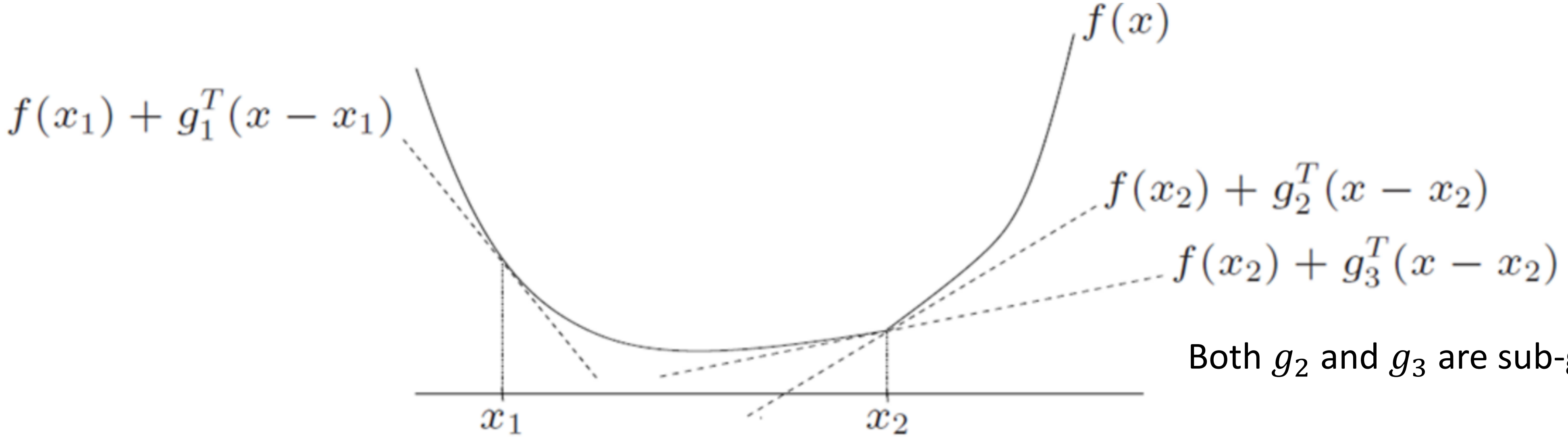
$$f(y) = f(x) + \nabla f(x)(y - x) + \frac{1}{2}(y - x)^T \nabla^2 f(x)(y - x) + \cdots$$

Convex Functions

- What if *f is non-differentiable (but still convex)?*

- Subgradient

  First order condition: g is a subgradient at x if $f(y) \geq f(x) + g^T(y - x) \forall y$



Both g_2 and g_3 are sub-grad

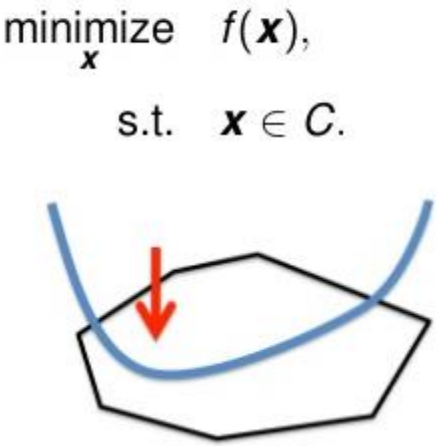
Convex Optimization

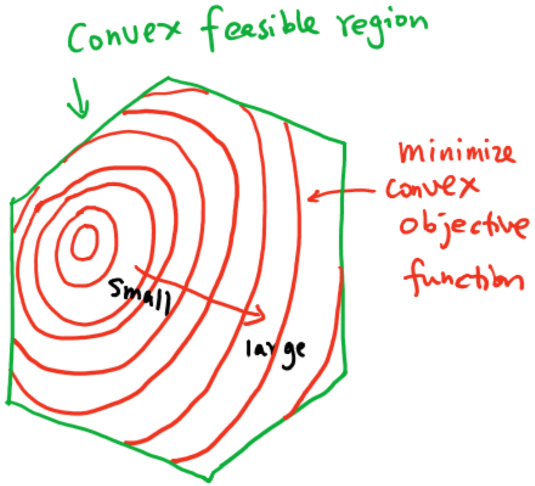
- Convex optimization: $f_0, f_1, \ldots$ are convex functions, h_j are linear functions

$$\begin{aligned} &\mathit{minimize}_x\ f_0(x) \\ &\mathit{subject\ to}\ f_i(x) \le 0\ \forall i \\ &h_j(x) = 0\ \forall j \end{aligned}$$

f_0:objective function

All x satisfying the constraints defines a convex region (feasible region).



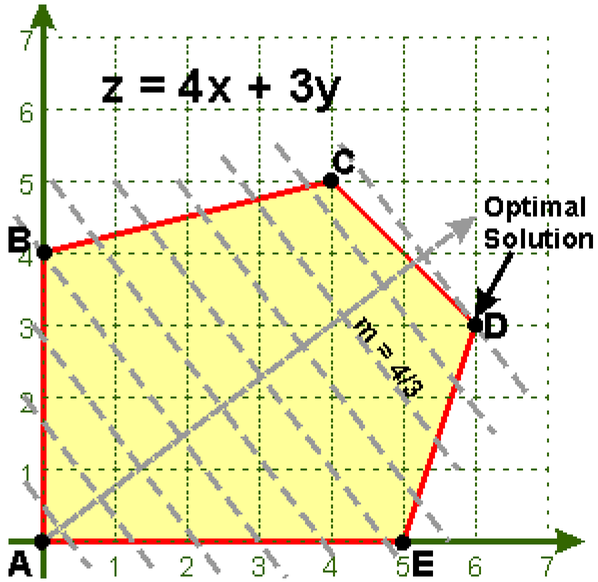


Contour Map

Convex Optimization

- Linear Programming:

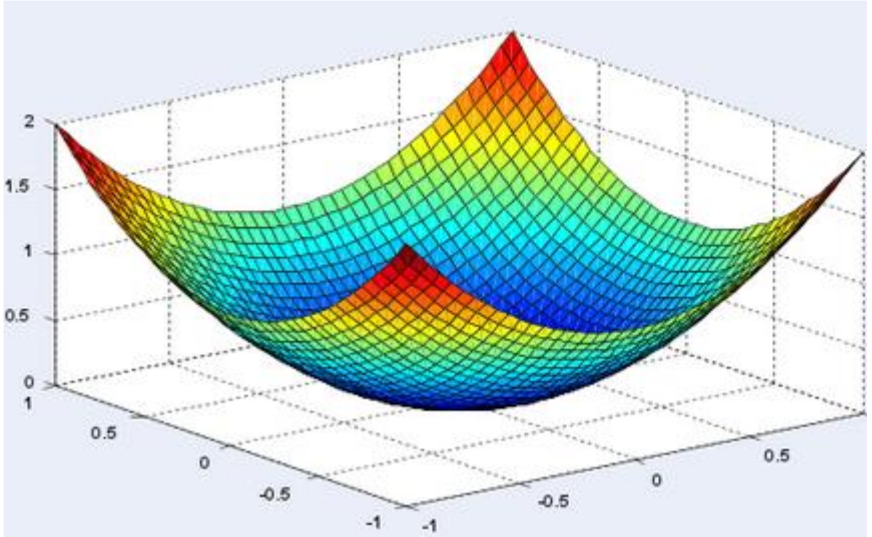
$$minimize\ c^T x \ \ subject\ to\ Ax \geq b, x \geq 0$$



Convex Optimization

- Quadratic Programming: P is positive semi-definite

$$\text{minimize } \frac{1}{2}x^T P x + q^T x + r$$
$$\text{subject to } Ax \geq b, x \geq 0$$



Note that if P is not psd, the objective function is not convex (it can be even concave).

Convex Optimization

- Second Order Cone Program (SOCP)
- Geometric Programming (GP)
- Semidefinite Programming (SDP)

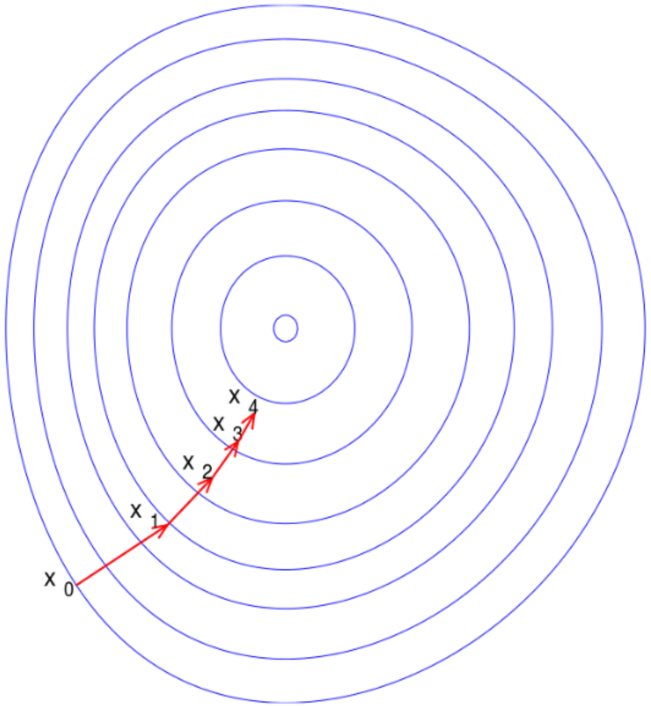
See the classic book [Convex Optimization] by Stephen Boyd and Lieven Vandenberghe

Sub-gradient Descent

Sub-gradient Descent for unconstraint minimization:

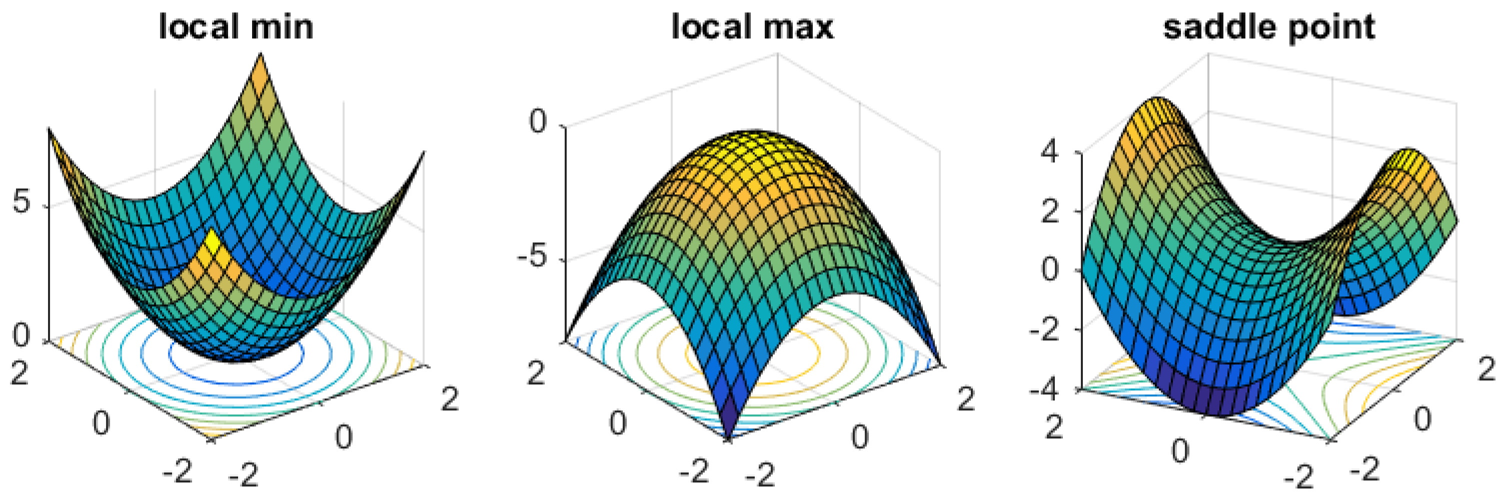
- Iterate until converge: $x^{(k+1)} = x^{(k)} - \alpha_k g_k$
- α_k: step size
  - Constant step size: $\alpha_1 = \alpha_2 = \alpha_3 = \cdots$
  - Decreasing step size: $\alpha_k = O(1/k)$, $\alpha_k = O(1/\sqrt{k})$,....
- Testing convergence
  - $|f(x^{(k+1)}) - f(x^{(k)})|$ is small enough
  - ....

Subgradient at $x^{(k)}$



Sub-gradient Descent

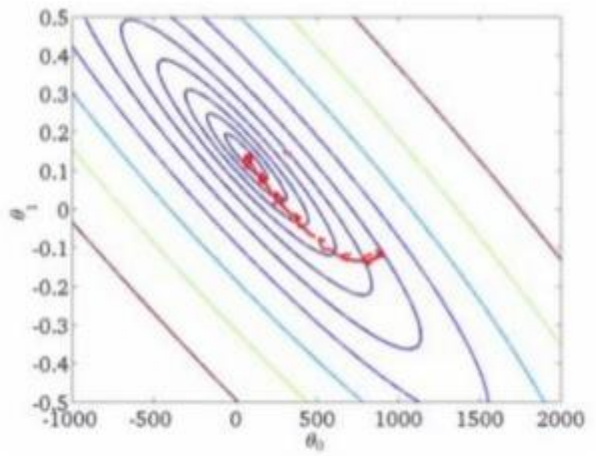
- Guarantee to converge for convex function
  - May converge to local optimal or saddle point for nonconvex functions



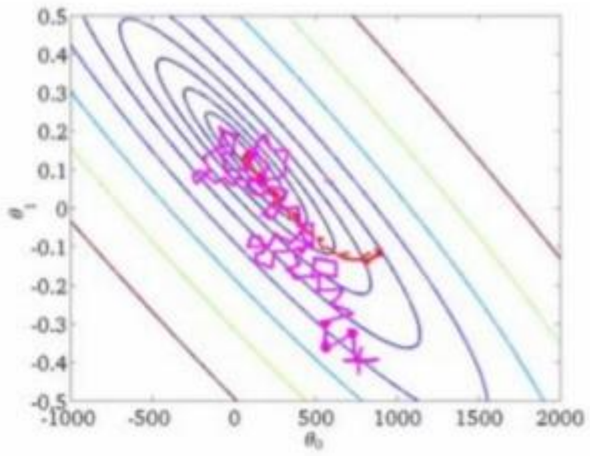
Stochastic Gradient Descent

- Common loss function in ML
  - $loss(w) = (1/n)\sum_{i=1}^{n} \ell_i(w)$ (each ℓ_i corresponds to a data point, *w* is the parameter we want to learn)
    - E.g. $\ell_i(w) = (w^T x_i - y_i)^2$
- SGD: Iterate until converge: $w^{(k+1)} = w^{(k)} - \alpha_k h_k$
  - h_k is a random vector such that $\mathrm{E}[h_k] = g_k$
  - For $loss(w) = (1/n)\sum_{i=1}^{n} \ell_i(w)$, we can choose $h_k = \nabla \ell_i(w^{(k)})$ where i is chosen uniformly at random from [*n*]
    - It is easy to see that $\mathrm{E}[h_k] = \mathrm{E}\left[\nabla \ell_i\left(w^{(k)}\right)\right] = \left(\frac{1}{n}\right)\sum_i \nabla \ell_i\left(w^{(k)}\right) = \nabla loss(w^{(k)})$
    - Hence, in each iteration, we only need one data point

GD vs SGD



Batch: gradient



Stochastic: single-example gradient

SGD

- How to implement SGD (to make it run faster and on larger data sets)
  - Parallel algorithm (Synchronous vs Asynchronous)
    - Analyzing the convergence for asynchronous algorithm can be tricky
      - *Hogwild!: A Lock-Free Approach to Parallelizing Stochastic Gradient Descent F. Niu, B. Recht, C. Ré, and S. J. Wright. NIPS, 2011*
      - *Asynchronous stochastic convex optimization*. John C. Duchi, Sorathan Chaturapruek, and C. Ré. NIPS15.
  - Reduce the variance
    - Rie Johnson and Tong Zhang. Accelerating Stochastic Gradient Descent using Predictive Variance Reduction, NIPS 2013.
  - Mini-batch
    - Instead of computing gradient for each single data point, we do it for a mini-batch (which contains more than 1 points (e.g., 5-20).
  - System level optimization

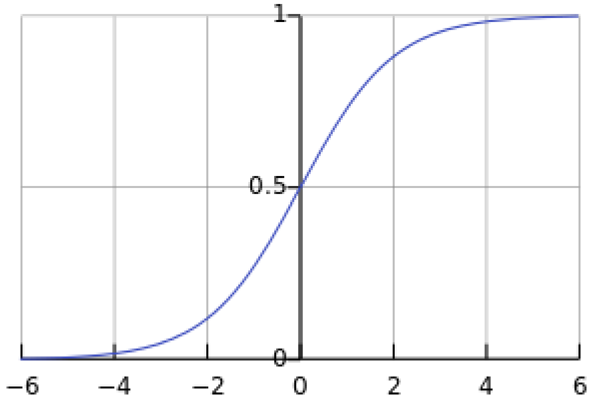
Logistic Regression

Logistic Regression

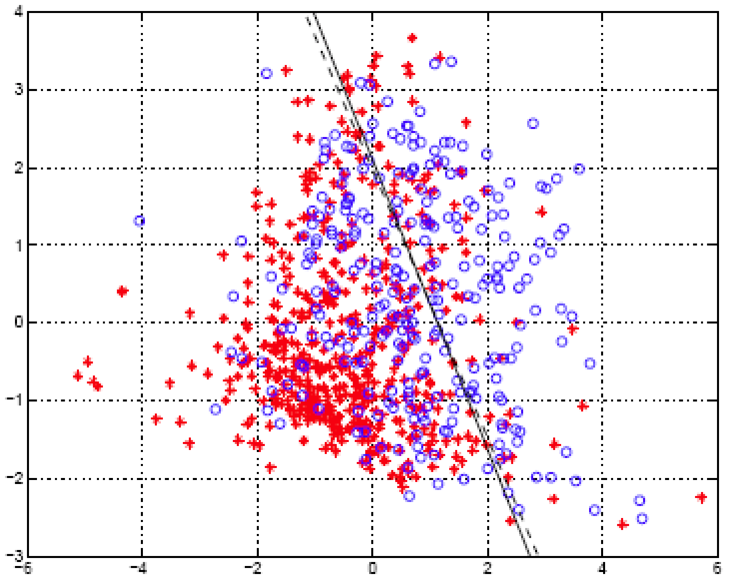
- Two class: p(x)=Pr[G=1|x]

$$\log \frac{p(x)}{1-p(x)} = \beta_0 + x \cdot \beta$$

$$p(x;b,w) = \frac{e^{\beta_0 + x\cdot\beta}}{1+e^{\beta_0 + x\cdot\beta}} = \frac{1}{1+e^{-(\beta_0 + x\cdot\beta)}}$$



Logistic function: $\sigma(t) = \frac{e^t}{e^t+1} = \frac{1}{1+e^{-t}}$



Logistic Regression

- Likelihood: $L(\beta_0, \beta) = \prod_{i=1}^{n} p(x_i)^{y_i}(1 - p(x_i))^{1-y_i}$

- Objective – maximize the log-likelihood

$$
\begin{aligned}
\ell(\beta_0, \beta) &= \sum_{i=1}^{n} y_i \log p(x_i) + (1 - y_i) \log 1 - p(x_i) \\
&= \sum_{i=1}^{n} \log 1 - p(x_i) + \sum_{i=1}^{n} y_i \log \frac{p(x_i)}{1 - p(x_i)} \\
&= \sum_{i=1}^{n} \log 1 - p(x_i) + \sum_{i=1}^{n} y_i (\beta_0 + x_i \cdot \beta) \\
&= \sum_{i=1}^{n} - \log 1 + e^{\beta_0 + x_i \cdot \beta} + \sum_{i=1}^{n} y_i (\beta_0 + x_i \cdot \beta)
\end{aligned}
$$

Logistic Regression

- The gradient:

$$\begin{aligned}\frac{\partial \ell}{\partial \beta_j} &= -\sum_{i=1}^{n} \frac{1}{1+e^{\beta_0 + x_i \cdot \beta}} e^{\beta_0 + x_i \cdot \beta} x_{ij} + \sum_{i=1}^{n} y_i x_{ij} \\ &= \sum_{i=1}^{n} \left(y_i - p(x_i; \beta_0, \beta)\right) x_{ij}\end{aligned}$$

- Cross Entropy (between two distributions p and q):

$$H(p,q) = -\sum_{x} p(x) \log q(x).$$

- The objective of RL is in fact minimizing the cross entropy

Logistic Regression

- Multiclass:

$$\Pr\left(Y = c | \vec{X} = x\right) = \frac{e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}{\sum_c e^{\beta_0^{(c)} + x \cdot \beta^{(c)}}}$$

Softmax function

- Homework:

  (1) Compute the log-likelihood for multiclass LR and its gradient

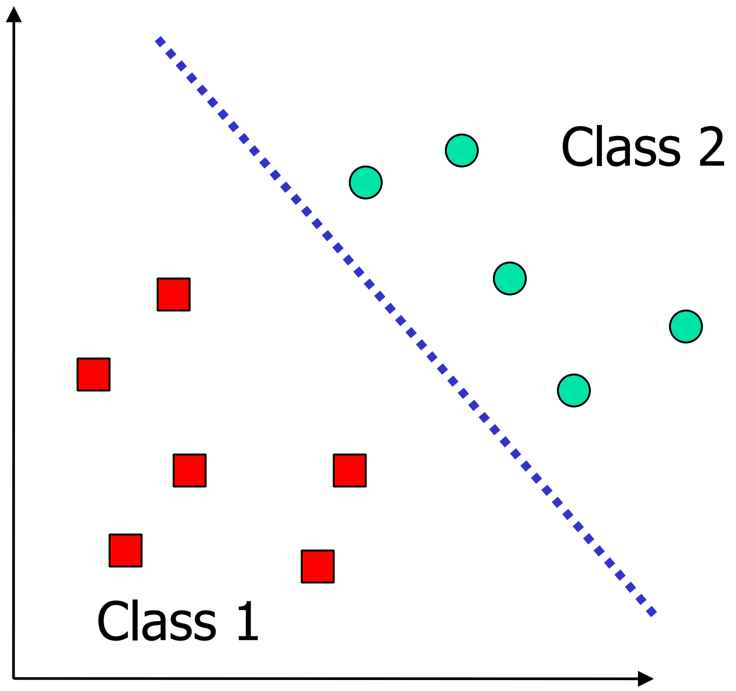
  (2) Express the objective as the cross entropy function

SVM and The Idea of Max Margin

Support Vector Machines (SVM)

- Derived from statistical learning theory by Vapnik and Chervonenkis (COLT-92)
- Base on convex optimization. Solid theoretical foundation.
- Mainstream machine learning method. Very successful for many problems for many years.
- The ideas and algorithms are very important in the development of machine learning.
- The max-margin idea (the hinge loss) extends to many many learning problems
  - Can be incorporated in deep learning

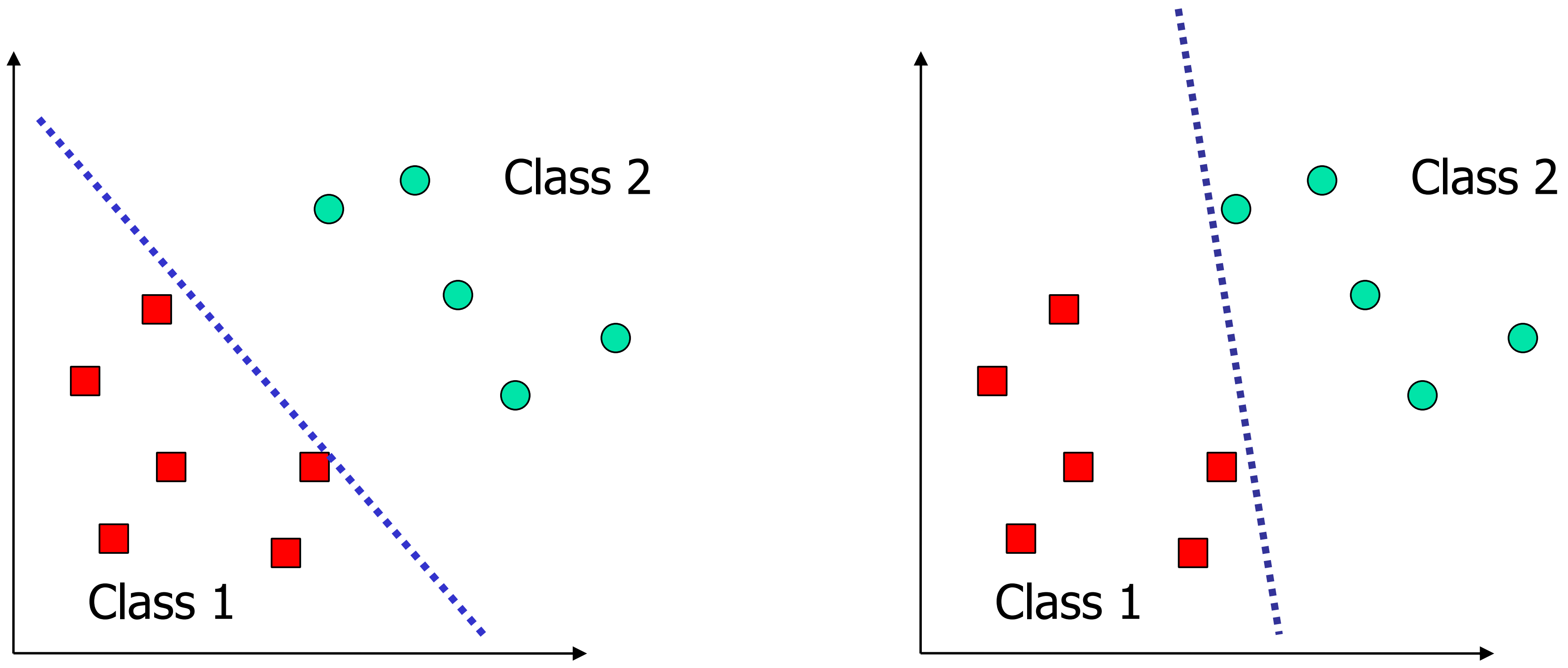
Two Class Problem: Linear Separable Case



- Many decision boundaries can separate these two classes
- Which one should we choose?

Note: Perceptron algorithm can also be used to find a decision boundary between class 1 and class 2

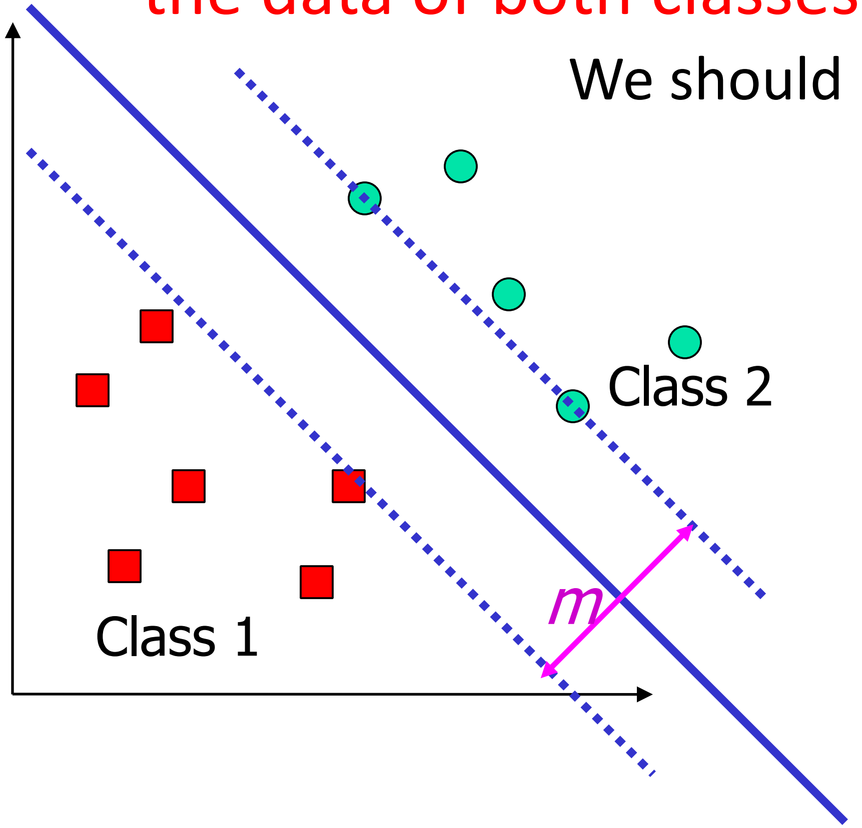
Example of Bad Decision Boundaries



Good Decision Boundary: Maximizing the Margin

- The decision boundary should be as far away from the data of both classes as possible

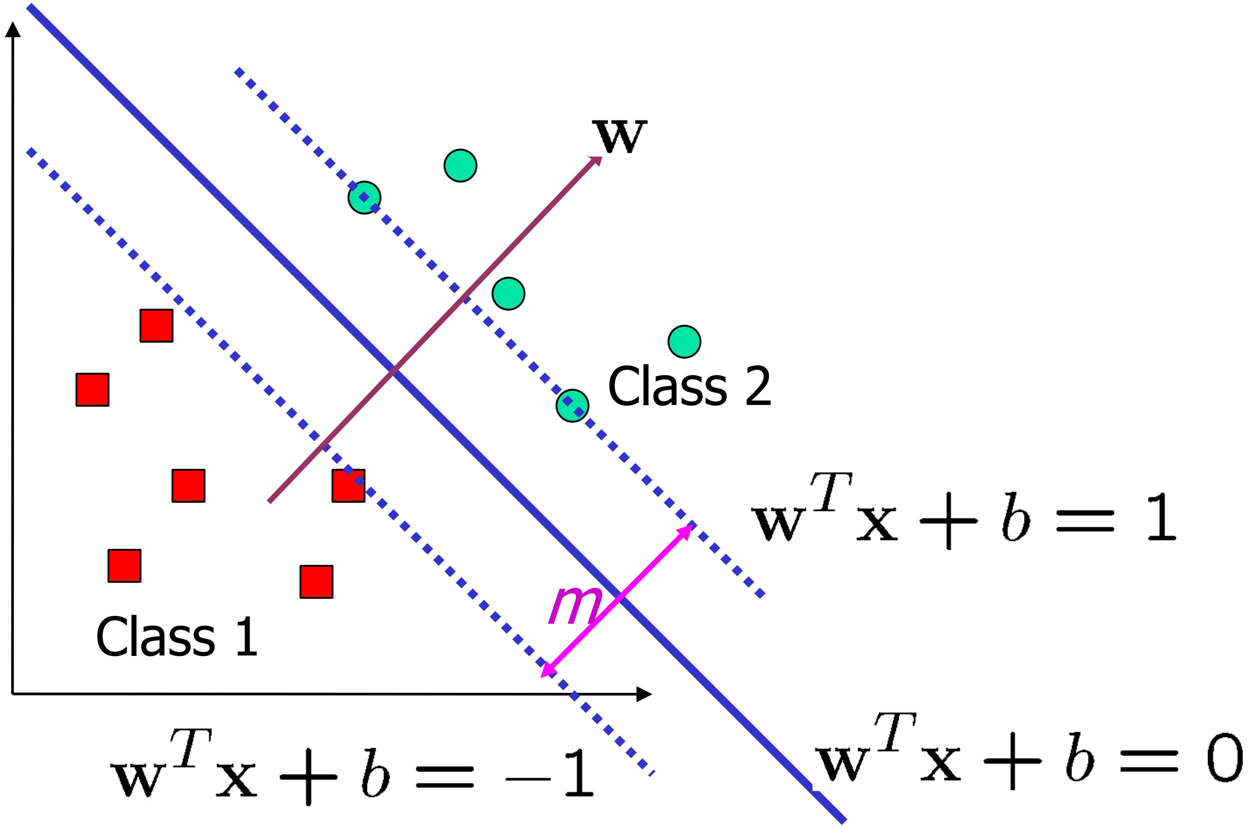
We should maximize the margin, *m*



The maximum margin linear classifier is the linear classifier with the maximum margin. This is the simplest kind of SVM (Called an Linear SVM)

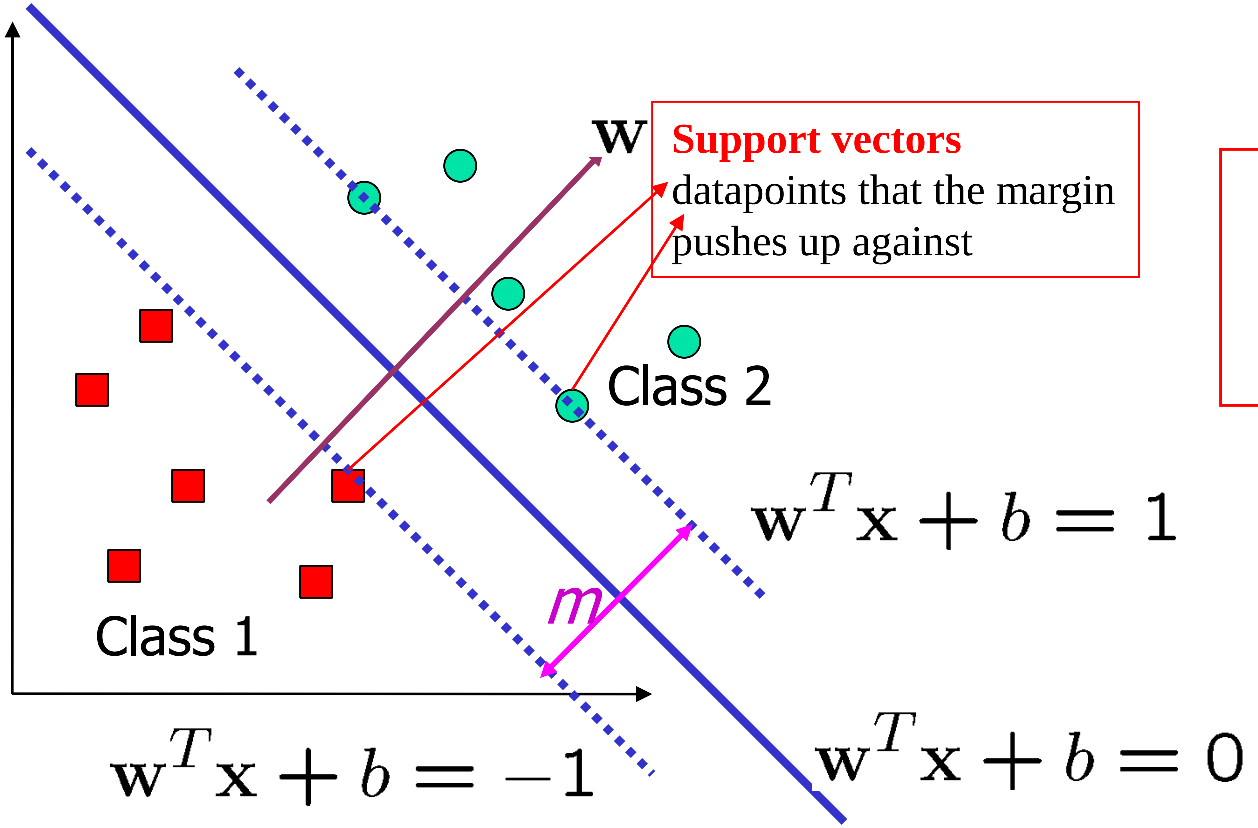
Good Decision Boundary: Maximizing the Margin

- Maximize the margin, *m*



Good Decision Boundary: Maximizing the Margin

- Maximize the margin, *m*



$$m = \frac{2}{||\mathbf{w}||} \qquad \|\mathbf{x}\| := \sqrt{x_1^2 + \cdots + x_n^2}.$$

Why? A HW.
Don't look up the internet. It is simple geometry. Do it by yourself

The Optimization Problem

- Let $\{x_1, \ldots, x_n\}$ be our data set and let $y_i \in \{1,-1\}$ be the class label of x_i
- The decision boundary should classify all points correctly $\Rightarrow$

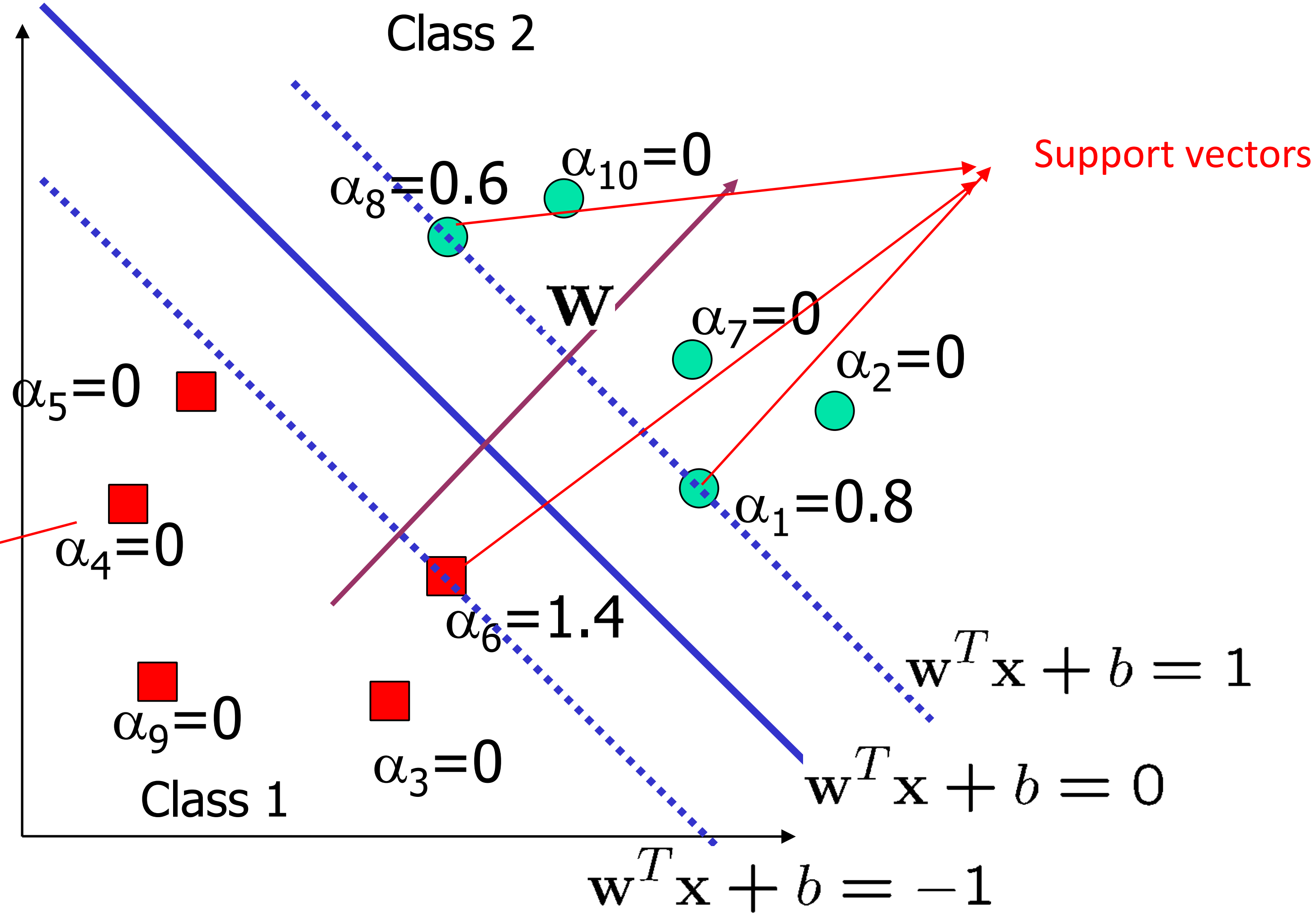
$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$$

- A constrained optimization (Quadratic Programming) problem

$$\text{Minimize } \frac{1}{2}||\mathbf{w}||^2$$

$$\text{subject to } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \qquad \forall i$$

A Geometrical Interpretation



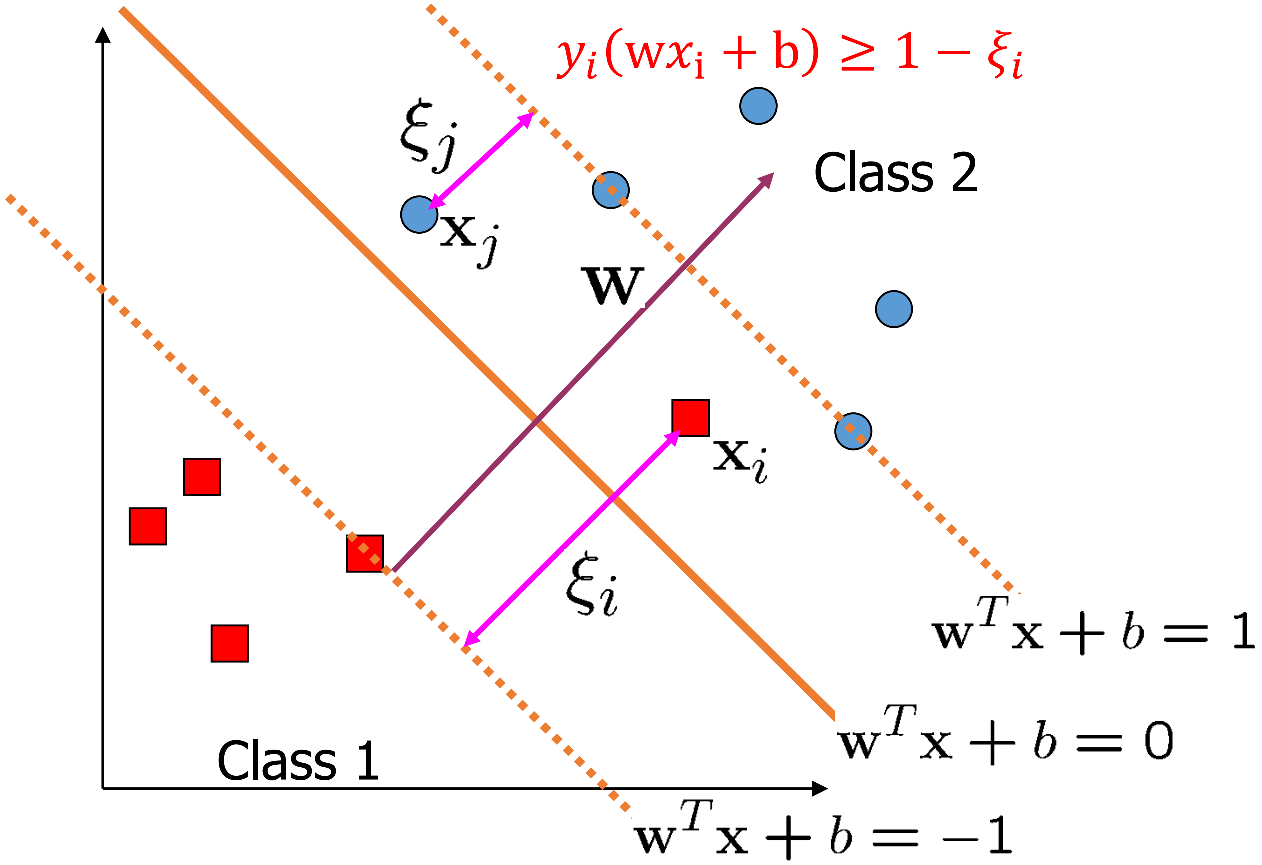
The boundary does not depend on the internal pts

(can be seen from complementary slackness)

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

Non-linearly Separable Problems

- We allow "error" ξ_i in classification; it is based on the output of the discriminant function $\mathbf{w}^T\mathbf{x}$+b
- ξ_i approximates the number of misclassified samples



New QP:

$$\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$

Subject to
$y_i(wx_i + b) \geq 1 - \xi_i$ for all i,
$\xi_i \geq 0$ for all i

C : tradeoff parameter

What happens for very large *C*?
For very small *C*?

Hinge Loss

- Hinge Loss:

$$\ell(t) = \max(0, 1 - t)$$

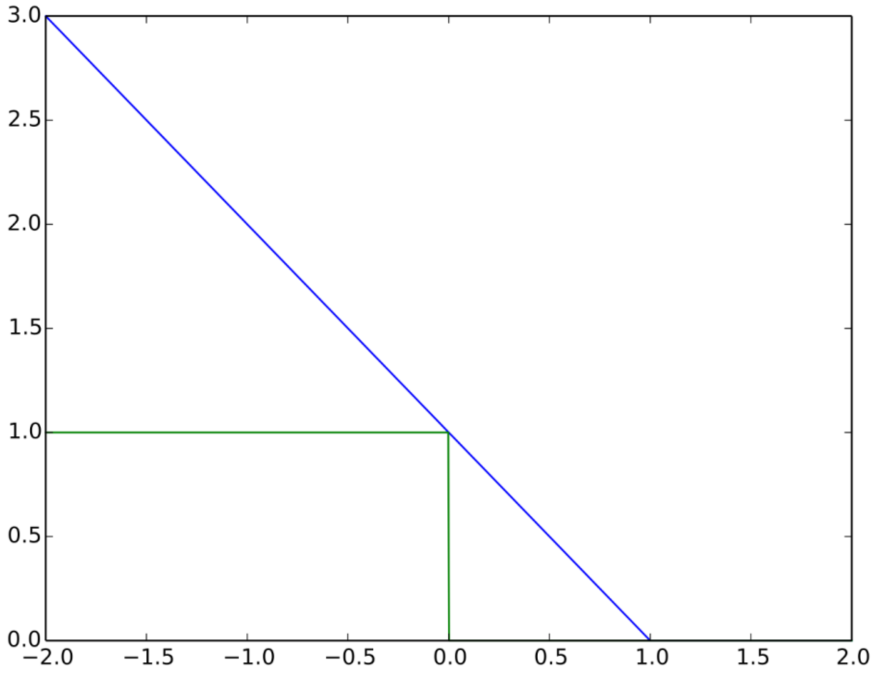
- Think it as a Convexification of 0-1 loss
- A reformulation of non-separable SVM using Hinge Loss:

$$\boxed{min. \quad \frac{1}{2}\|w\|^2 + C\sum_i \ell(y_i(wx_i + b))}$$

$$\Updownarrow$$

$$\frac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i$$

S.t. $y_i(\mathrm{w}x_\mathrm{i} + \mathrm{b}) \geq 1 - \xi_i$ for all i,
$\xi_i \geq 0$ for all i



Hinge Loss

Hinge Loss

- More generally, for classification function, the hinge loss of point x_i:

$$\ell(f(x_i), y_i) = \max(0, 1 - y_i f(x_i))$$

- In linear SVM, $f(x) = wx + b$
- If $f(x_i)$ has the same sign as y_i, and $|f(x_i)| \geq 1$, the loss $\ell(f(x_i), y_i) = 0$

- Hinge loss is a convex loss function (but not differentiable). Hence, we can use standard sub-gradient descent algorithm to solve it.

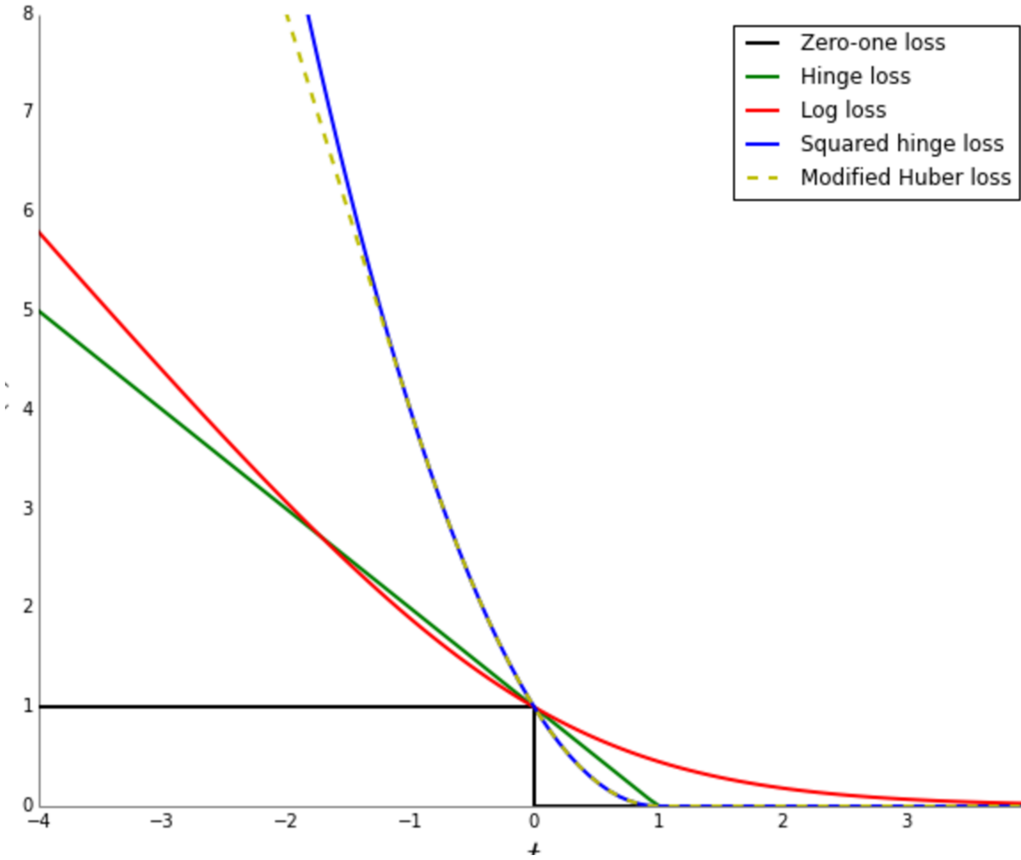
Other (surrogate) Loss Functions

- Log (logistic) loss:

$$\ell(f(x_i), y_i) = \log_2(1 + e^{-y_i f(x_i)})$$

The loss for logistic regression ($y_i = \pm 1$)

(the loss in previous slides was for $y_i = 0,1$)

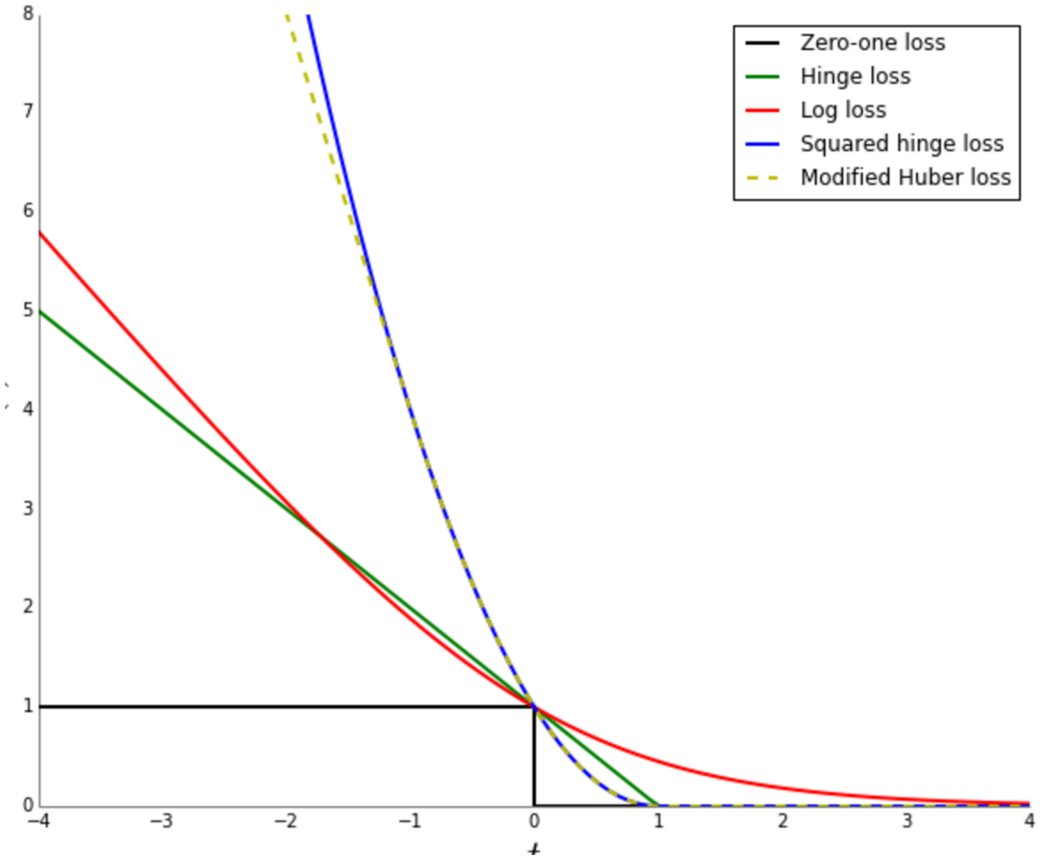


Other (surrogate) Loss Functions

- Modified Huber Loss:

$$\ell(f(x_i), y_i) =$$

$$\begin{cases} -2y_i f(x_i) + 1 & y_i f(x_i) \leq 0 \\ (y_i f(x_i) - 1)^2 & 0 < y_i f(x_i) \leq 1 \\ 0 & \text{otherwise} \end{cases}$$



- Exponential loss (in boosting)
- Sigmoid loss (nonconvex)

Regularization

- SVM: $min. \frac{1}{2}||w||^2 + C\sum_i \ell(f(x_i), y_i)$
- More generally:

$$\boxed{mininimize \quad \underbrace{\text{penalty}(w)} + \underbrace{C\sum_i \ell oss(f(x_i), y)}}$$

Regularization: What do we want about *w*

- ℓ_2: $||w||_2^2$
- $\ell_1\ (LASSO)$: $||1||_1$
  (it encourages the sparsity of w)

Loss: how well the function fits the training data

Why regularization?
One important reason: prevent overfitting.
Better generalization to new data points

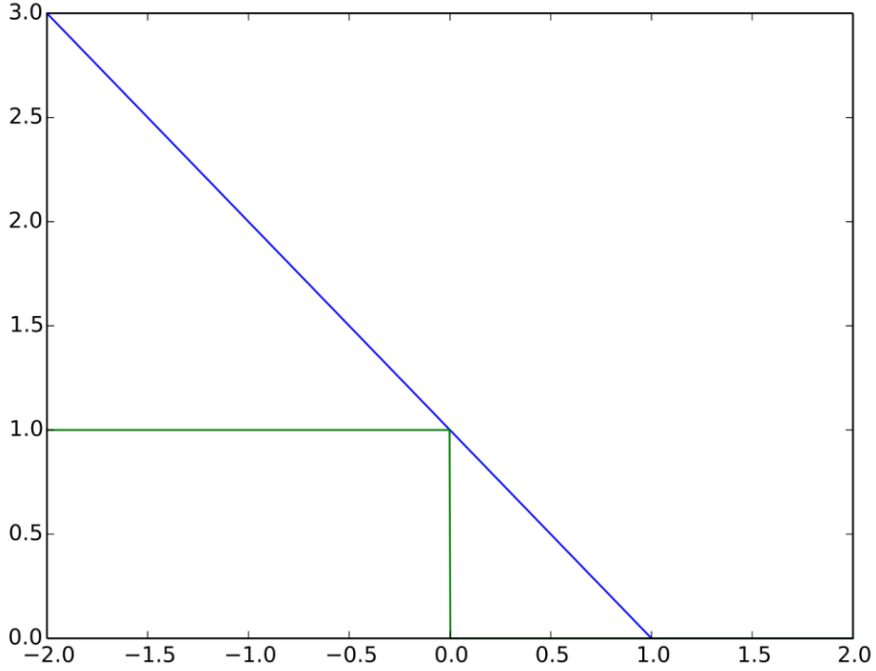
SVM implementations

- Svmlight: http://svmlight.joachims.org/
- LIBSVM and LIBLINEAR
- Implemented in many machine learning libraries:
  - sofia-ml (google)
  - scikit-learn (python)
  - matlab

Sub-gradient Descent

- Subgradient for Hinge loss:

- $\frac{\partial \ell}{\partial w_i} = -y_i x_i$ if $y_i f(x_i) < 1$
- $\frac{\partial \ell}{\partial w_i} = 0$ if $y_i f(x_i) \geq 1$



- Coding HW:

Implement the subgradient descent
algorithm for SVM
(create a simple 2-d example and visualize it)

Multiclass SVM

Idea: There is a different weight vector w_i or each class *i* (label)

matrix W $= [w_1, w_2, \ldots, w_k]$

The correct label should be better than the other labels by a margin

Primal

$$\max_{W,\xi} \frac{1}{2}||W||_2^2 + C\sum_i \xi_i$$

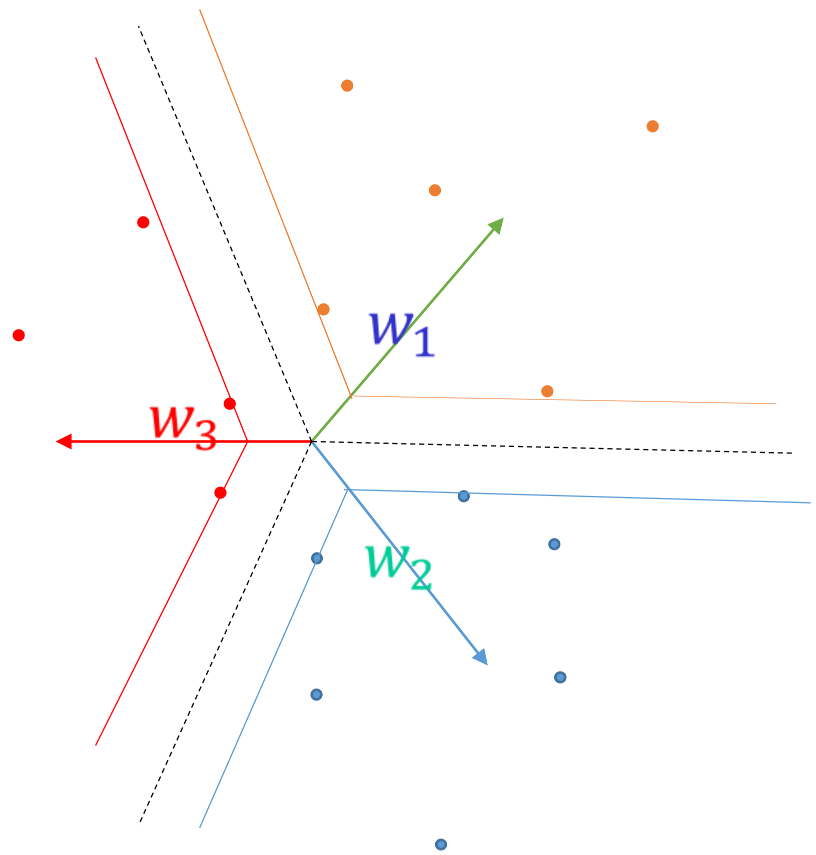
$$s.t. \;\; w_{y_i}^T \Phi(x_i) - w_y^T \Phi(x_i) \geq \Delta(y_i, y) - \xi_i \quad \forall i, y$$

Dissimilarity of y_i and y

E.g., $\Delta(y_i, y) = 1 - I(y_i = y)$

Note that SVM is a special case, while $w_+ = w$ and $w_- = -w$.

Multiclass SVM



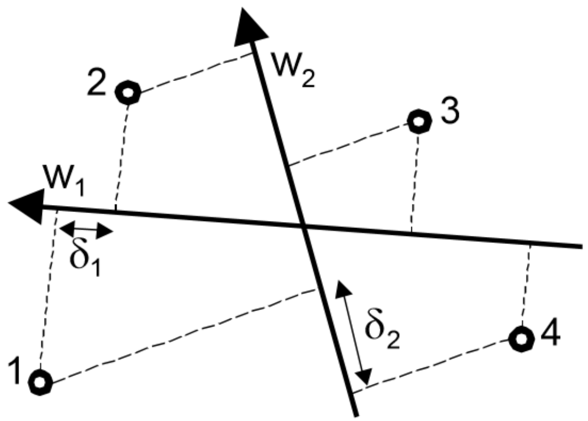
Multiclass SVM

- HW: Write a Hinge loss formulation for multiclass SVM (it should be equivalent to the above formulation)

SVM-Rank

- Imagine that the search engine wants rank a collection of documents for a query
- Training data (query, ranking): $(q_1, r_1^*), (q_2, r_2^*), \ldots, (q_n, r_n^*)$.
- For each doc d and a query q, we can produce a feature $\Phi(q, d)$
- Want to learn a good ranking function
- Assume linear ranking functions: d_i is better than d_j iff $\vec{w}\Phi(q, d_i) > \vec{w}\Phi(q, d_j)$

The weight vector we want to learn

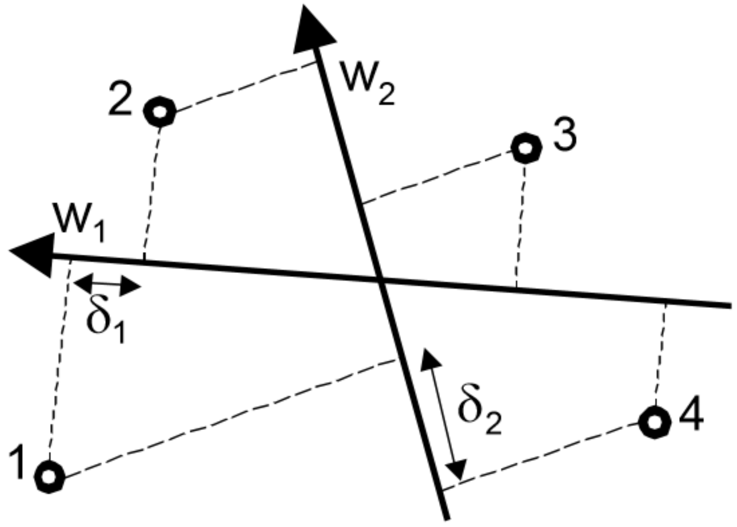


Optimizing Search Engines using Click through Data, Joachims

SVM-Rank

- Training data (query, ranking): $(q_1, r_1^*), (q_2, r_2^*), ..., (q_n, r_n^*)$.
- We want to following set of inequalities hold for the training data

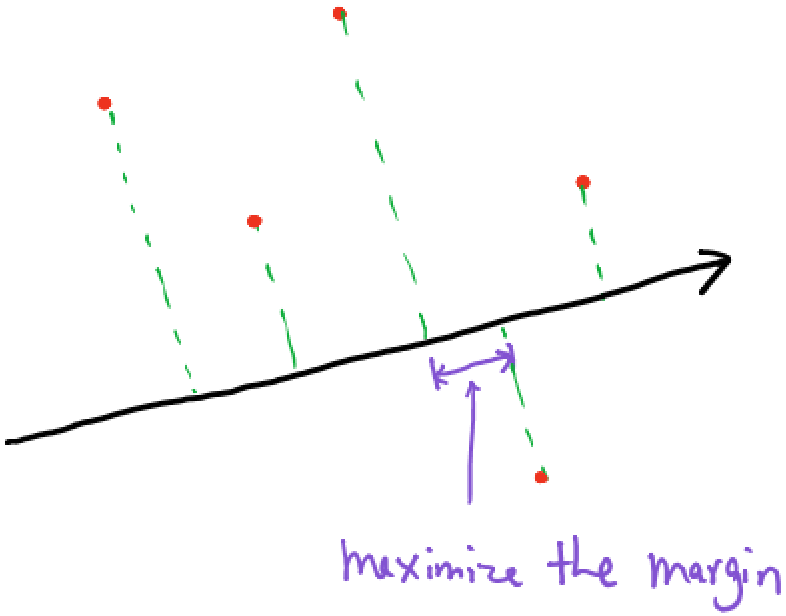
$$\forall (d_i, d_j) \in r_1^* : \quad \vec{w}\Phi(q_1, d_i) > \vec{w}\Phi(q_1, d_j)$$
$$\cdots$$
$$\forall (d_i, d_j) \in r_n^* : \quad \vec{w}\Phi(q_n, d_i) > \vec{w}\Phi(q_n, d_j)$$



Optimizing Search Engines using Clickthrough Data, Joachims

SVM-Rank

- Training data (query, ranking): $(q_1, r_1^*), (q_2, r_2^*), \ldots, (q_n, r_n^*)$.
- Natural Idea: Maximize the margin



OPTIMIZATION PROBLEM 1. (RANKING SVM)

minimize: $$V(\vec{w}, \vec{\xi}) = \frac{1}{2}\, \vec{w} \cdot \vec{w} + C \sum \xi_{i,j,k}$$

subject to:

$$\forall (d_i, d_j) \in r_1^* : \vec{w}\Phi(q_1, d_i) \geq \vec{w}\Phi(q_1, d_j) + 1 - \xi_{i,j,1}$$

$$\ldots$$

$$\forall (d_i, d_j) \in r_n^* : \vec{w}\Phi(q_n, d_i) \geq \vec{w}\Phi(q_n, d_j) + 1 - \xi_{i,j,n}$$

$$\forall i \forall j \forall k : \xi_{i,j,k} \geq 0$$

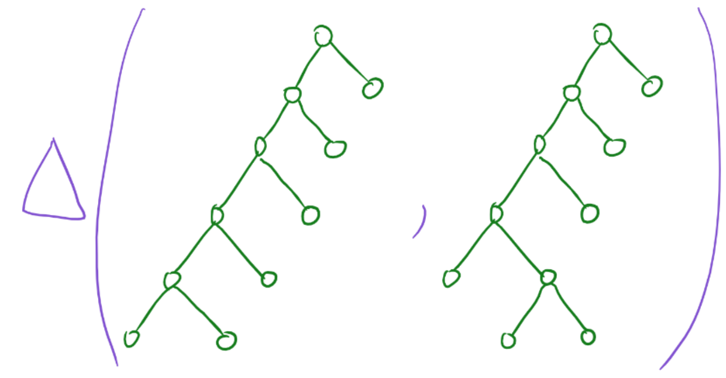
Optimizing Search Engines using Clickthrough Data, Joachims

Another formulation for Multiclass SVM
(more generally, Max-Margin for structured learning)

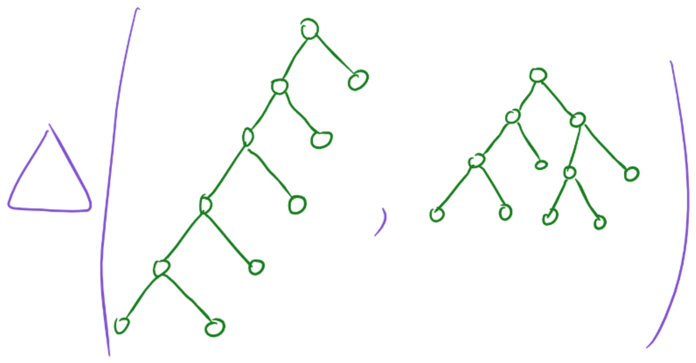
- Structured predictions
  - There can be a huge number of labels due to combinatorics
    - Ex 1: Muti-label prediction $Y = \{+1, -1\}^k$ SVM is a special case of k=1
      - $|Y| = 2^k$
      - $\Delta(Y, Y') = Hamming\ dist(Y, Y')$
    - Ex 2: Taxonomy classification
      - Each label is a tree of size *k*
      - $\Delta(Y, Y') = tree\ distance\ between\ Y\ and\ Y'$
  - We can't afford to have one weight vector for each label.
  - We will be finding a single weight vector *w*

Another formulation for Multiclass SVM
(more generally, Max-Margin for structured learning)

- Taxonomy classification (e.g., tree edit distance)



small



large

Another formulation for Multiclass SVM
(more generally, Max-Margin for structured learning)

- Construct features which depends on both the input *x* and the label *y*

$$\psi = \phi_{\mathcal{X}} \times \phi_{\mathcal{Y}} : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}^{d_x \cdot d_y}, \quad \psi(x,y) := \phi_{\mathcal{X}}(x) \times \phi_{\mathcal{Y}}(y)$$

- The primal quadratic program

$$(w^*, \xi^*) = \underset{w,\xi \geq 0}{\arg\min}\, \mathcal{H}(w,\xi) := \frac{\lambda}{2}\langle w, w\rangle + \frac{1}{n}\|\xi\|_1$$

$$\text{subject to} \quad \langle w, \delta\psi_i(y)\rangle \geq \Delta(y_i, y) - \xi_i \quad (\forall i, \forall y \in \mathcal{Y} - \{y_i\})$$

$$\text{binary: [subject to} \quad \langle w, y_i\phi(x_i)\rangle \geq 1 - \xi_i \quad (\forall i) \quad]$$

$$\text{where } \delta\psi_i(y) := \psi(x_i, y_i) - \psi(x_i, y).$$

The correct label should be better than the other lables by a margin

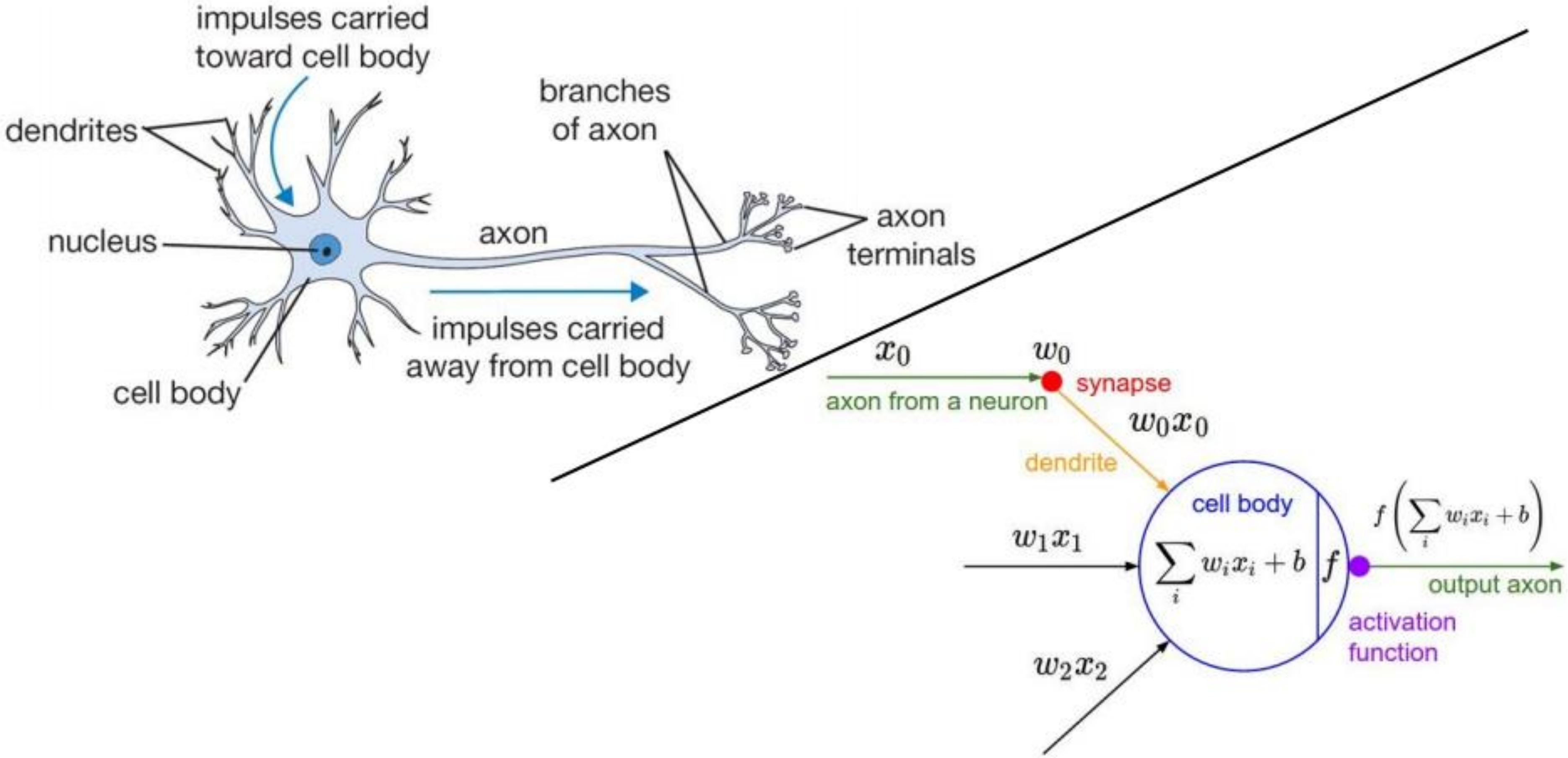
If we predict *y* and $\Delta(y_i, y)$ is large, we need to pay a large loss (ξ_i is large)

- We can also write our the dual.
- It can be quite challenging to solve the corresponding convex programming (a lot of constraints or variables)

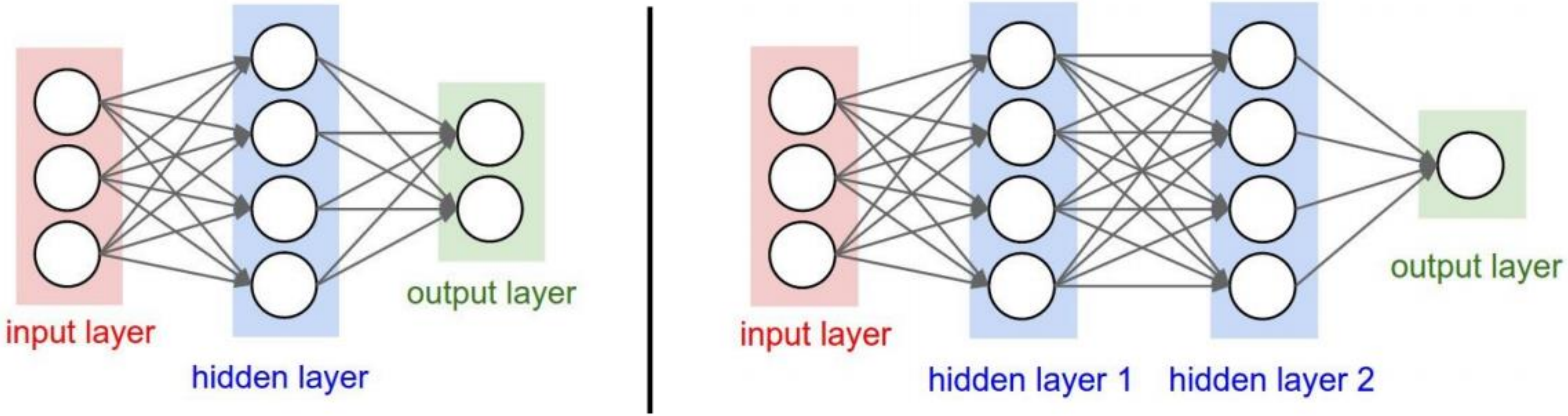
Neural Network Basics

Artificial Neural Networks

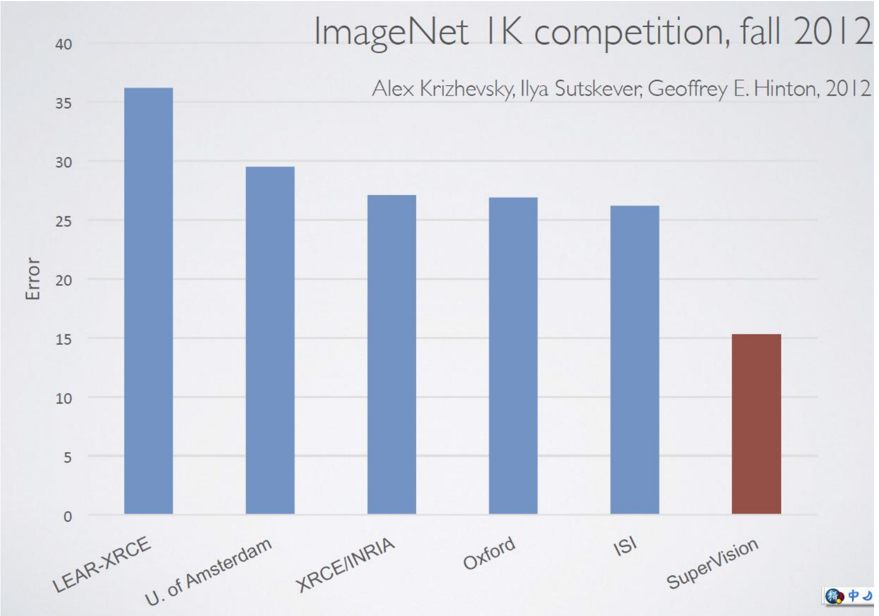
- First proposed by Warren McCulloch and Walter Pitts (in 1940s)



Artificial Neural Networks



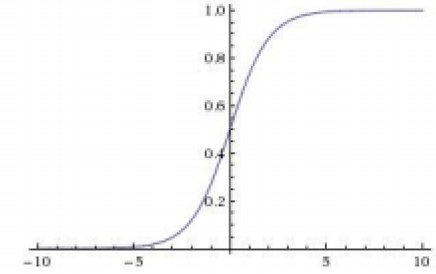
Artificial Neural Networks



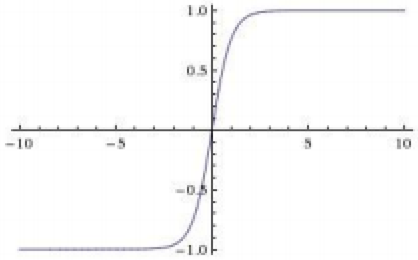
Activation functions

Sigmoid

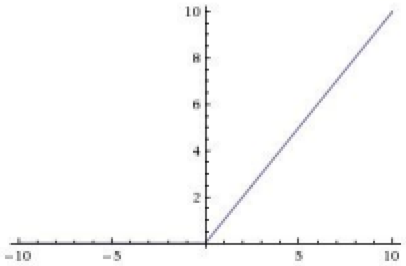
$$\sigma(x) = 1/(1 + e^{-x})$$



tanh tanh(x)

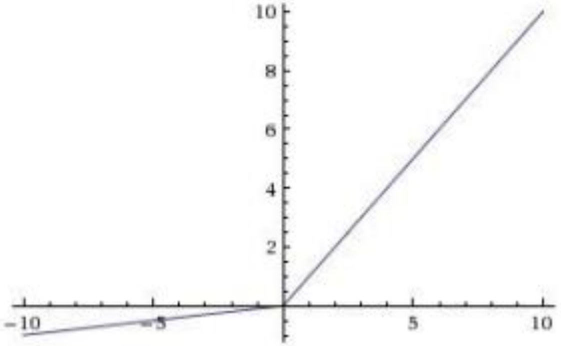


ReLU max(0,x)



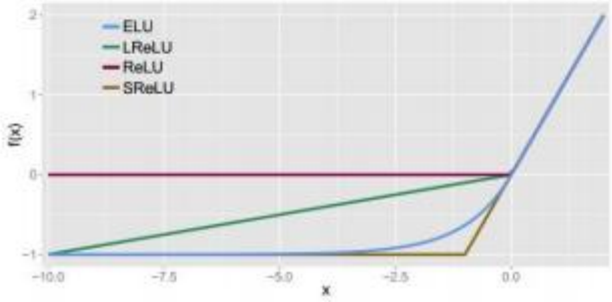
Leaky ReLU

max(0.1x, x)

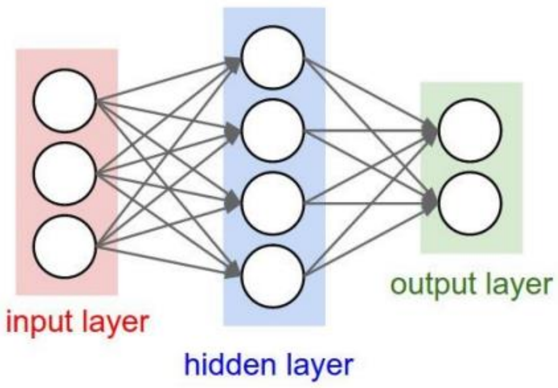


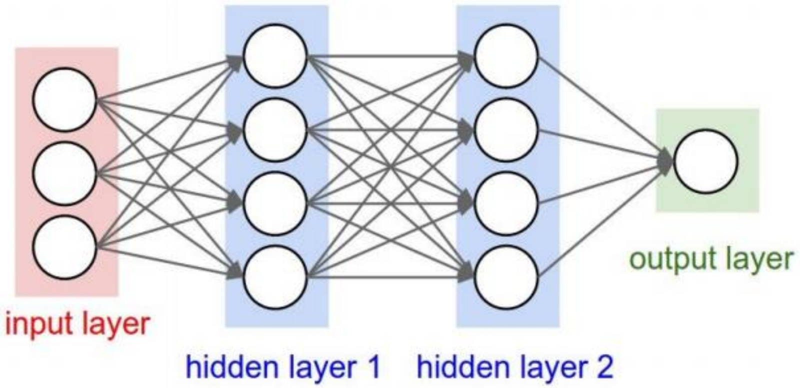
Maxout $\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU $f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha\,(\exp(x) - 1) & \text{if } x \le 0 \end{cases}$



View NN as functions





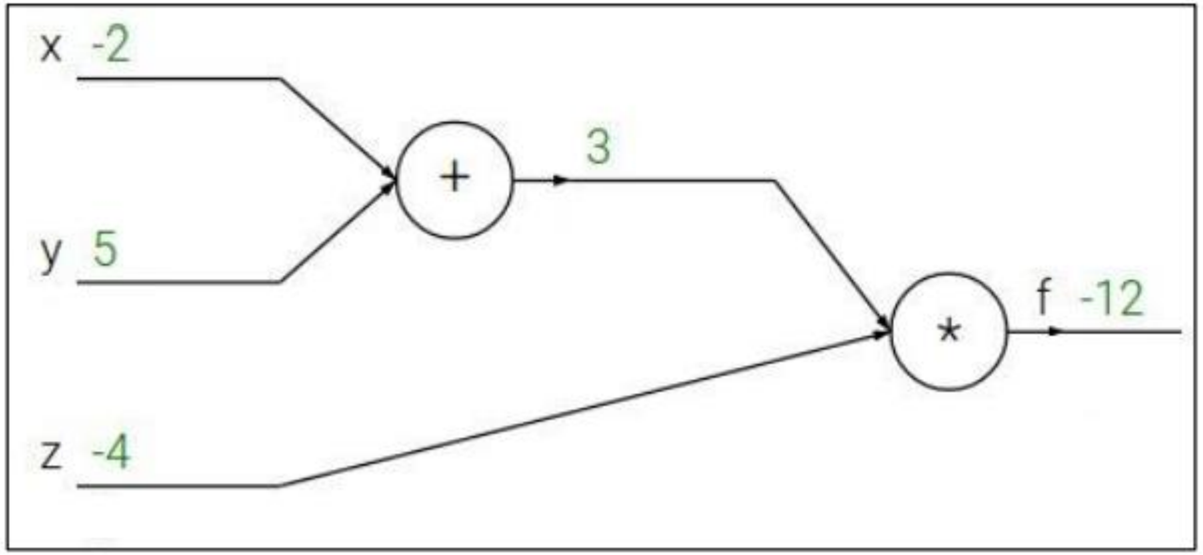
$$\mathcal{M}(\mathbf{w},\mathbf{x}) = \sum_{j=1}^{N_h} w_j^{(2)} \phi\left(\sum_{i=1}^{d} w_{ji}^{(1)} x_i\right)$$

$$\mathcal{M}(\mathbf{w},\mathbf{x}) = \sum_{i^{(\lambda)}} w_{i^{(\lambda)}}^{(\lambda)} \phi\left(\sum_{i^{(\lambda-1)}=1}^{N_{\lambda-1}} w_{i^{(\lambda)},i^{(\lambda-1)}}^{(\lambda-1)} \phi\left(\sum_{i^{(\lambda-2)}=1}^{N_{\lambda-2}} w_{i^{(\lambda-1)},i^{(\lambda-2)}}^{(\lambda-2)} \cdots \right.\right.$$
$$\left.\left.\cdots\ \phi\left(\sum_{i^{(1)}=1}^{N_1=d} w_{i^{(2)},i^{(1)}}^{(1)} x_{i^{(1)}}\right)\right)\right)$$

Forward Computation

$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4



Compute the gradient (Back Propagation)

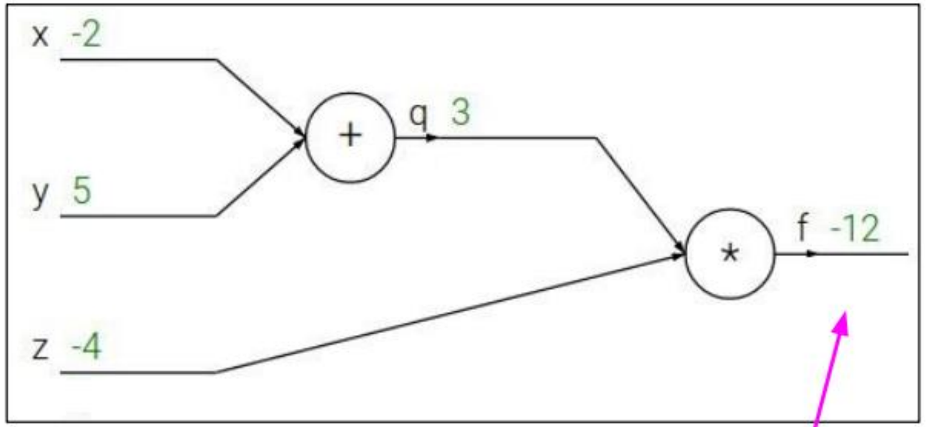
$$f(x,y,z) = (x+y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial f}$$

Compute the gradient (Back Propagation)

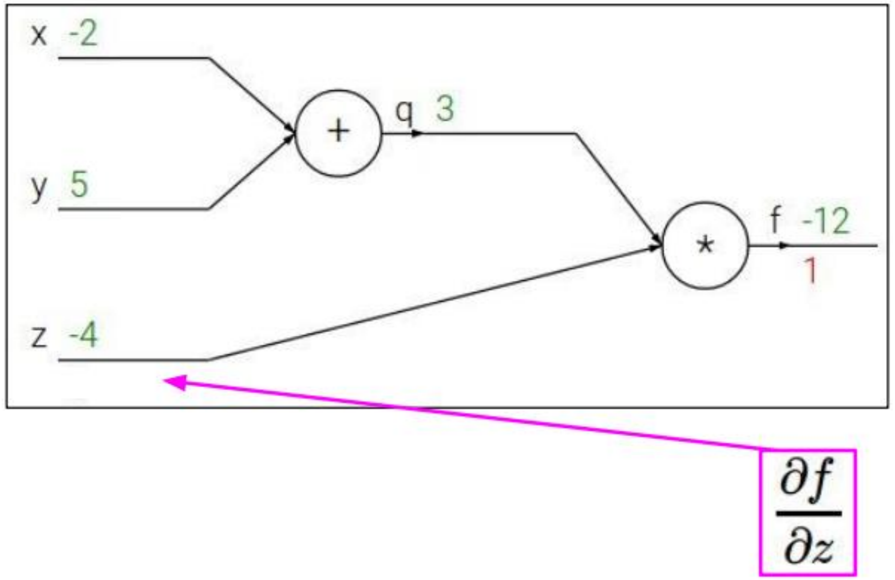
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



Compute the gradient (Back Propagation)

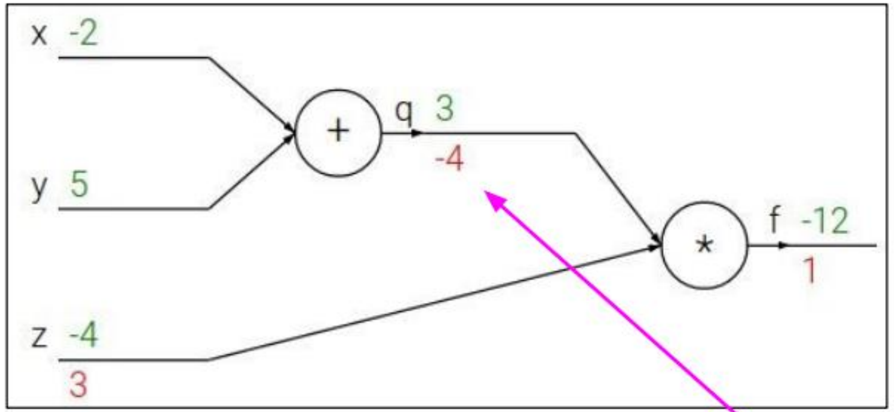
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial q}$$

Compute the gradient (Back Propagation)

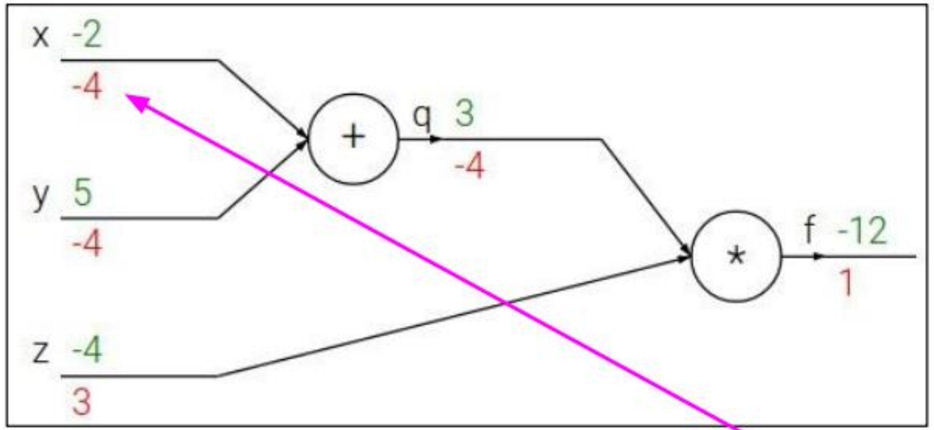
$$f(x, y, z) = (x + y)z$$

e.g. x = -2, y = 5, z = -4

$$q = x + y \qquad \frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$$

$$f = qz \qquad \frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$$

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



$$\frac{\partial f}{\partial x}$$

Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

Compute the gradient (Back Propagation)

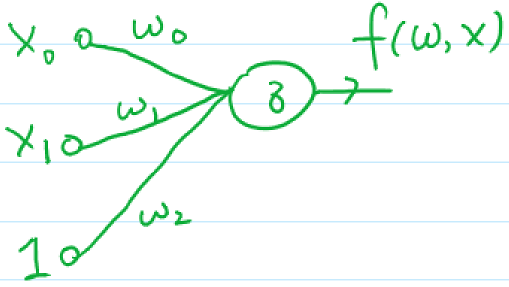
Chain rule: (to compute $\frac{\partial f}{\partial w}$)

$$F = f(g_1(x,y,z),\ g_2(x,y,z))$$

$$\frac{\partial F}{\partial x} = \frac{\partial f}{\partial g_1}\cdot\frac{\partial g_1}{\partial x} + \frac{\partial f}{\partial g_2}\cdot\frac{\partial g_2}{\partial x}$$

e.g. consider $f(w,x) = \dfrac{1}{1+e^{-(w_0x_0+w_1x_1+w_2)}} = \sigma(\underbrace{w_0x_0+w_1x_1+w_2}_{y})$

for sigmoid function $\sigma(x) = \dfrac{1}{1+e^{-x}}$

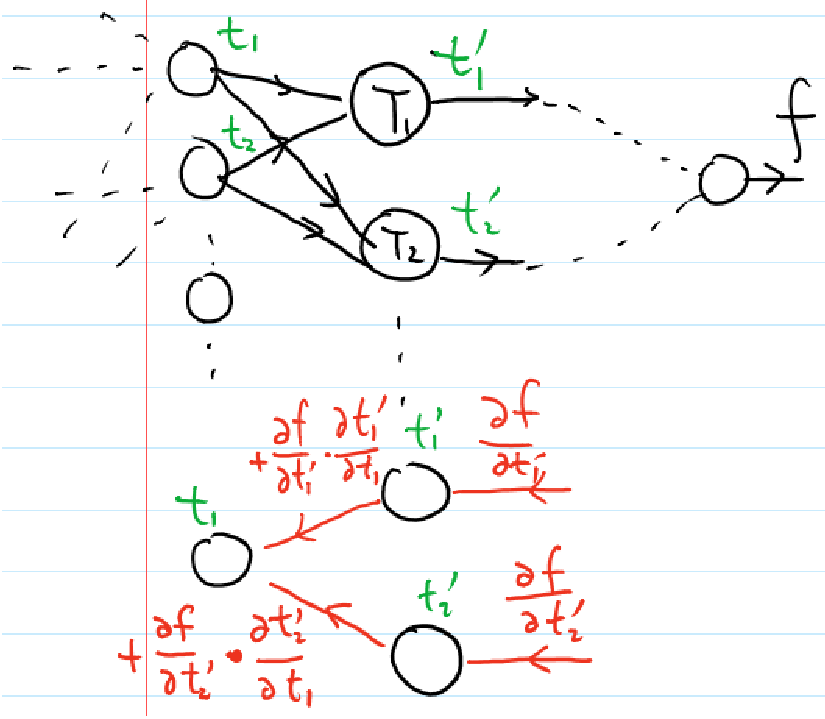


note: $\dfrac{d\sigma(x)}{dx} = \dfrac{e^{-x}}{(1+e^{-x})^2} = (1-\sigma(x))\sigma(x)$

$$\frac{\partial f}{\partial w_1} = \frac{\partial f}{\partial y}\cdot\frac{\partial y}{\partial w_1} = (1-\sigma(y))\cdot\sigma(y)\cdot x_1 \quad (\text{where } y = w_0x_0+w_1x_1+w_2)$$

Compute the gradient (Back Propagation)

Backprop tries to compute $\frac{\partial f}{\partial w}$ backwards



t, t_1, t_2 are the output variables of corresp gates

suppose we have already computed $\frac{\partial f}{\partial t_1'}$, $\frac{\partial f}{\partial t_2'}$ ...

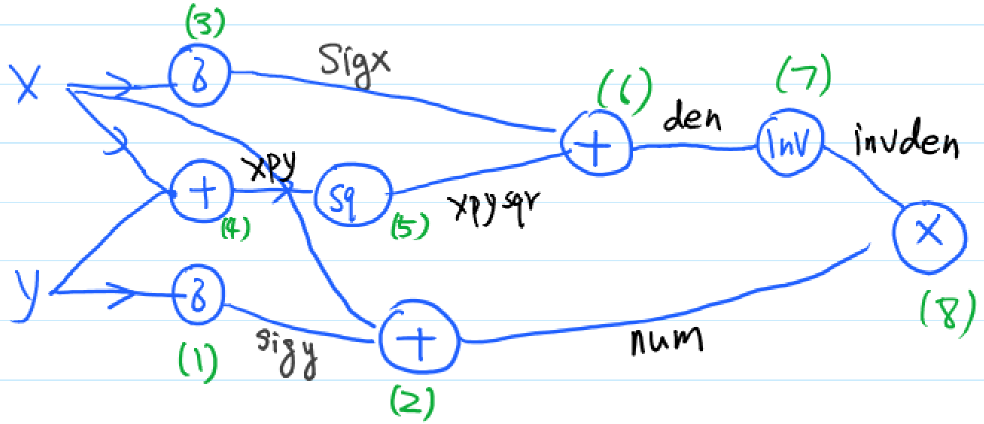
$t_1' = T_1(t_1, t_2, \cdots)$ $\quad t_2' = T_2(t_1, t_2, \cdots)$

$$\frac{\partial f}{\partial t_1} = \frac{\partial f}{\partial t_1'} \cdot \frac{\partial t_1'}{\partial t_1} + \frac{\partial f}{\partial t_2'} \cdot \frac{\partial t_2'}{\partial t_1} + \cdots$$

Back Propagation

$$f(x,y) = \frac{x + \sigma(y)}{\sigma(x) + (x+y)^2}$$

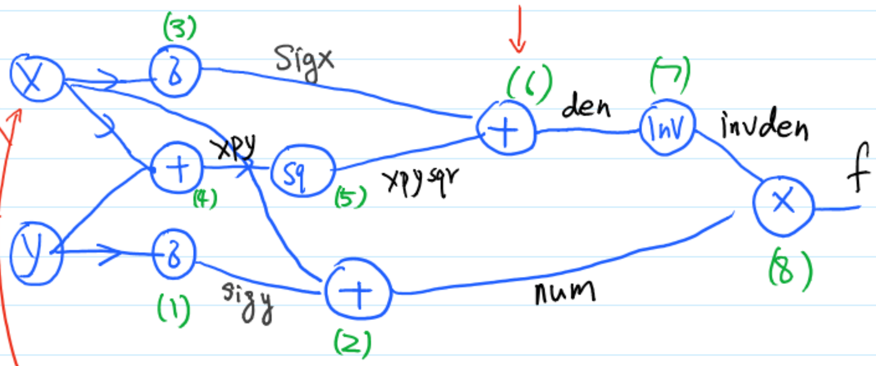
```
x = 3 # example values
y = -4

# forward pass
sigy = 1.0 / (1 + math.exp(-y)) # sigmoid in numerator    #(1)
num = x + sigy # numerator                                #(2)
sigx = 1.0 / (1 + math.exp(-x)) # sigmoid in denominator #(3)
xpy = x + y                                               #(4)
xpysqr = xpy**2                                           #(5)
den = sigx + xpysqr # denominator                         #(6)
invden = 1.0 / den                                        #(7)
f = num * invden # done!                                  #(8)
```



Back Propagation

```python
# backprop f = num * invden
dnum = invden # gradient on numerator                         #(8)
dinvden = num                                                 #(8)
# backprop invden = 1.0 / den
dden = (-1.0 / (den**2)) * dinvden                            #(7)
# backprop den = sigx + xpysqr
dsigx = (1) * dden                                            #(6)
dxpysqr = (1) * dden                                          #(6)
# backprop xpysqr = xpy**2
dxpy = (2 * xpy) * dxpysqr                                    #(5)
# backprop xpy = x + y
dx = (1) * dxpy                                               #(4)
dy = (1) * dxpy                                               #(4)
# backprop sigx = 1.0 / (1 + math.exp(-x))
dx += ((1 - sigx) * sigx) * dsigx # Notice += !! See notes below #(3)
# backprop num = x + sigy
dx += (1) * dnum                                              #(2)
dsigy = (1) * dnum                                            #(2)
# backprop sigy = 1.0 / (1 + math.exp(-y))
dy += ((1 - sigy) * sigy) * dsigy                             #(1)
```

e.g. $\frac{\partial f}{\partial Sigx} = \frac{\partial f}{\partial den}$



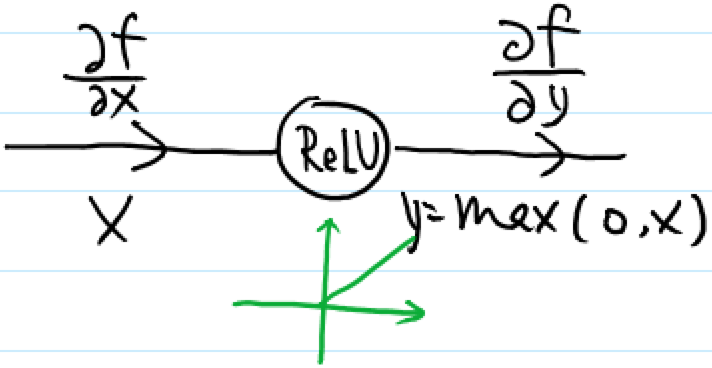
We better cache the forward variables as they are useful in computing gradient as well!

$$\frac{\partial f}{\partial x} = \underbrace{\frac{\partial f}{\partial Sigx}\cdot\frac{\partial Sigx}{\partial x}}_{(3)} + \underbrace{\frac{\partial f}{\partial xpy}\cdot\frac{\partial xpy}{\partial x}}_{(4)} + \underbrace{\frac{\partial f}{\partial num}\cdot\frac{\partial num}{\partial x}}_{(2)}$$

$$= (1 - Sigx)\cdot Sigx \cdot dSigx + dxpy + dnum$$

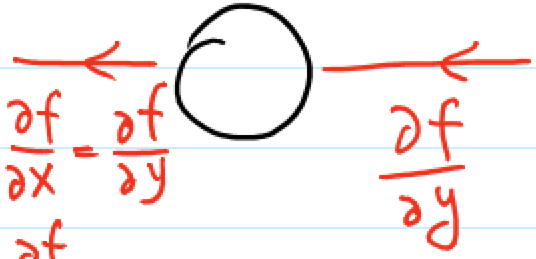
Back Propagation

Relu Gate



$$\frac{\partial f}{\partial y} = \frac{\partial f}{\partial x} \quad \text{if } x \geq 0$$

$$\frac{\partial f}{\partial y} = 0 \quad \text{if } x < 0$$



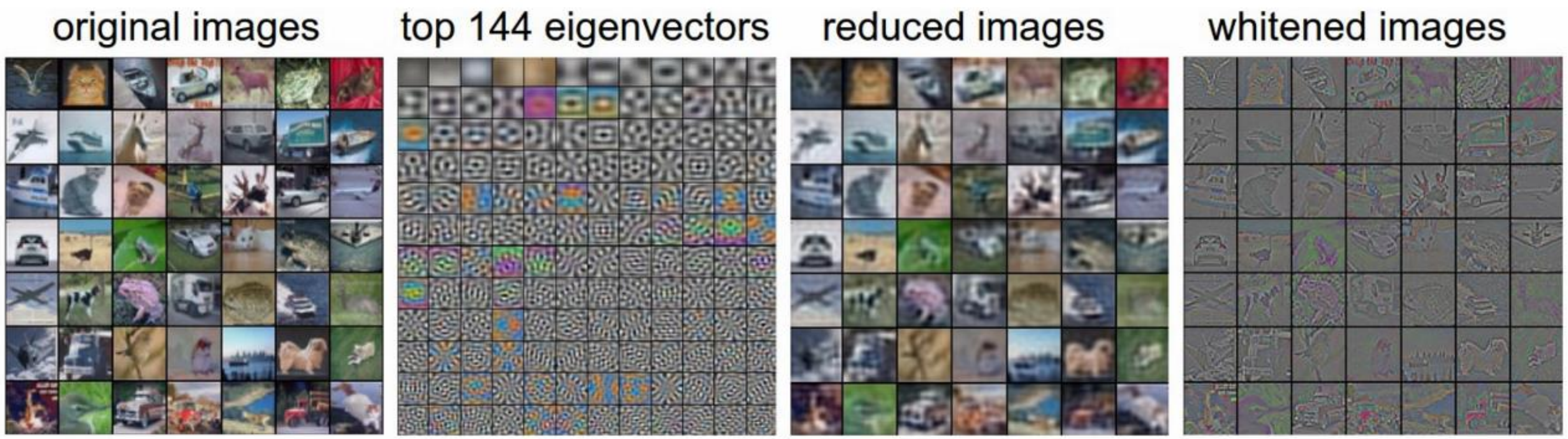
$x \geq 0$: $\frac{\partial f}{\partial x} = \frac{\partial f}{\partial y}$

$x <$: $\frac{\partial f}{\partial x} = 0$

Training NN

Preprocessing ① zero mean

② Whitening : equalize the top-k principle directions



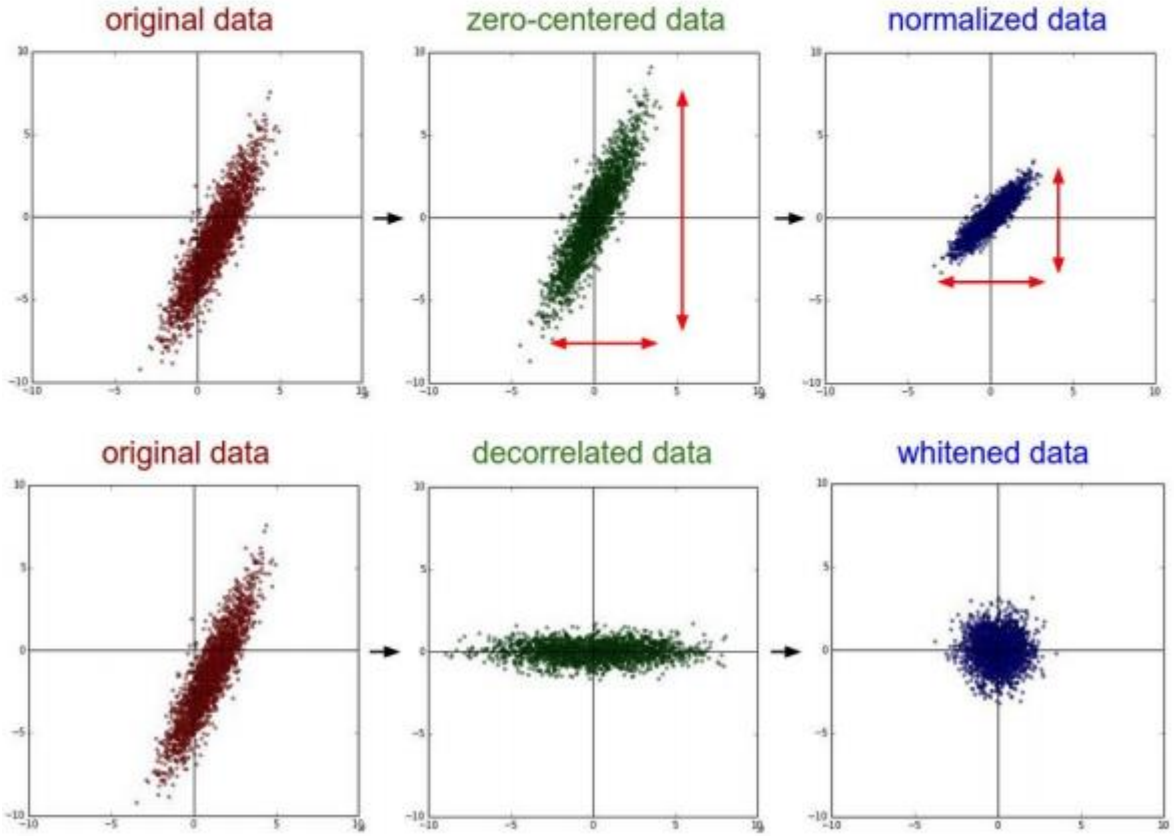
images by top-144 eigenvectors

after equalizing the top-144 directions

Note: not used so much in CNN

(see PCA. next time)

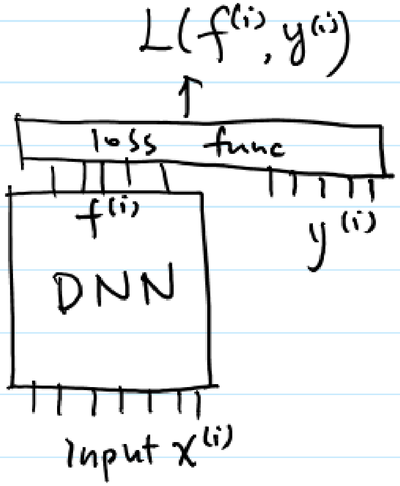
Whitening



original data
zero-centered data
normalized data
original data
decorrelated data
whitened data

Training NN

Training. Mini-batch SGD



Sample a batch of data.

Forward (compute all forward variables)

$$L = \sum_{i \in \text{mini-batch}} L(f^{(i)}, y^{(i)})$$

Back prop to compute the gradient)

$$\nabla_w L = \left(\frac{\partial L}{\partial w_1} \cdots\cdots \right)^T$$

Update the weight

$$w \leftarrow w + \eta \nabla_w L$$

Loss functions

Classification:

from the same f vector

$L = \frac{1}{N}\sum_{i=1}^{N} L_i$ ① $L_i = \sum_{j \neq y_i} \max(0, f_j - f_{y_i} + 1)$ (multiclass SVM loss)

recall $\max(0, x)$ hinge loss

$f = (f_1 \cdots f_j \cdots f_d)$ output of NN

e.g. $f = (3, -5, 2.5)$. the true lable is $y_i = 0$

$L = \max(0, -5-3+1) + \max(0, 2.5-3+1)$

$= 0 + 0.5$

if f[0] is larger than other entries by 1, the loss is 0

② $L_i = -\log\left(\frac{e^{f_{y_i}}}{\sum_j e^{f_j}}\right)$ (cross-entropy loss / soft-max)

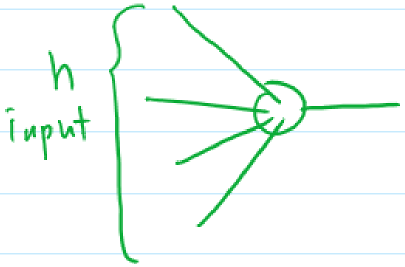
Training NN

Weight initialization

① zeros

② small random numbers $\sim N(0, 0.01)$

③ carlibrating the variance



$w \sim \frac{1}{\sqrt{n}} N(0, 1)$, ensure the outputs have similar distr

$$Var\left(\sum_{i=1}^{n} w_i x_i\right) = \sum_{i=1}^{n} Var(w_i)\, Var(x_i)$$

(assuming $E(X_i) = 0$)

Training NN

- Tricks

① Choose successive examples from different classes

– try to update gradient fast

② Choose the data with large error first

– careful. outliers would be disastrous

③ Momentum

$$\Delta w^{(t+1)} \leftarrow \eta \nabla_w L + \mu \Delta w^{(t)}$$

$$w \leftarrow w + \Delta w^{(t+1)}$$

Useful in the direction with low curvature

Adagrad

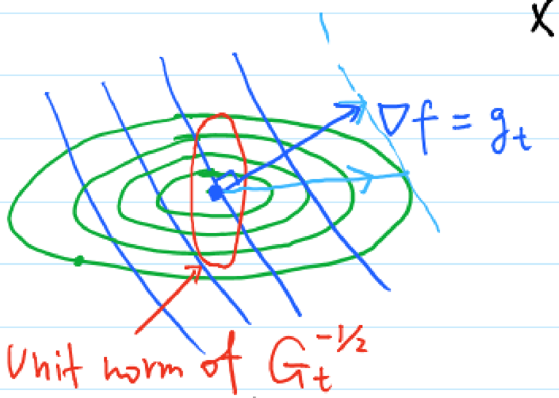
[Duchi, Hazan, Singer JMLR11]

projected grad descent

$$x_{t+1} = \Pi_X(x_t - \eta g_t) = \arg\min_{x \in X} \| x - (x_t - \eta g_t)\|_2^2$$

Steepest descent in Mahalanobis norm $\|x\|_A = \sqrt{x^T A x}$

$$x_{t+1} = \arg\min_{x \in X} \| x - (x_t - \eta G_t^{-1/2} g_t)\|_{G_t^{1/2}} \qquad G_t = \sum_{\tau=1}^{t} g_\tau g_\tau^T$$



Unit norm of $G_t^{1/2}$: $x^T G_t^{1/2} x = 1$

$$x^T \begin{bmatrix} 1 & \\ & 5 \end{bmatrix} x = 1$$

The above is computation expensive.

Adagrad

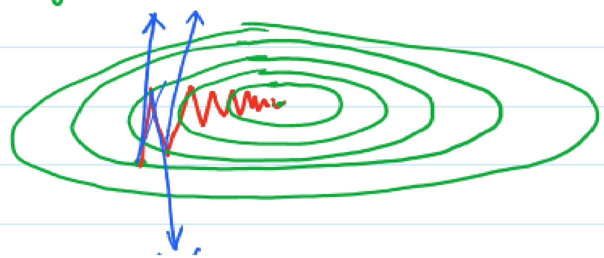
(diagonal adaptation) $$x_{t+1} = \underset{x \in X}{\arg\min} \left\| x - \left(x_t - \eta\, \underbrace{\mathrm{diag}(G_t)}_{\begin{pmatrix} \sum_{\tau=1}^{t} g_{\tau 1}^2 \\ \vdots \\ \sum_{\tau=1}^{t} g_{\tau i}^2 \end{pmatrix}}^{-1/2} g_t\right) \right\|_{\mathrm{diag}(G_t)^{1/2}}$$

try to make the first order method better conditioned.

related to FTRL, dual averaging - proximal method.

e.g. ∇f

GD



in ADAGRAD, after a few iteration the variance in y direction accumulate

- Hw: implement Adagrad for a simple function (in low dim) and compare it with the standard GD (and visualize it)

RMSProp

[Tieleman, Hinto 2012]

$$V_t = \text{decayrate} \times V_{t-1} + (1 - \text{decayrate}) \cdot \begin{pmatrix} g_{t1}^2 \\ \vdots \\ g_{ti}^2 \\ \vdots \end{pmatrix}$$

$$x_{t+1} \longleftarrow x_t - \eta (V_t)^{-1/2} g_t$$

ADAM

ADMA:

$$g_t \leftarrow \nabla f$$

$$m_t \leftarrow \beta_1 \times m_{t-1} + (1-\beta_1) \cdot g_t$$

$$v_t \leftarrow \beta_2 \times v_{t-1} + (1-\beta_2) \cdot \begin{pmatrix} g_{t_1}^2 \\ \vdots \\ g_{t_i}^2 \\ \vdots \end{pmatrix}$$

$$\bar{m}_t \leftarrow m_t / (1-\beta_1^t)$$

$$\bar{v}_t \leftarrow v_t / (1-\beta_2^t)$$

$$x_{t+1} \leftarrow x_t - \eta \left(\bar{v}_t\right)^{-1/2} \bar{m}_t$$

Batch Normalization [Ioffe, Szegedy]

For a layer of input vector x:

Normalize:

$$\boxed{\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}}$$

$$\boxed{y^{(k)} = \gamma^{(k)}\widehat{x}^{(k)} + \beta^{(k)}}$$

Additional parameters to learn (thru BP)

- BP needs to be modified to account for the change
- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout, maybe

Training NN

- Dropout:
  - An effective way to prevent overfitting
  - In each iteration, drop each node with probability *p,* and train the remaining network.
  - In some sense, it has the effect of regularization
  - Make the training faster
  - Can be seen as an ensemble of many network structures (in a loose sense)
- Data Augmentation
  - E.g., images – flip, rotate, shift the images, delete some (rows or col) pixels
- Lots Lots of other tricks [Book: Neural Networks: Tricks of the Trade]

Reference

- Crammer K, Singer Y. On the algorithmic implementation of multiclass kernel-based vector machines[J]. Journal of machine learning research, 2001, 2(Dec): 265-292. (multi-class SVM)
- Joachims, Thorsten. "Optimizing search engines using clickthrough data."*Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2002. (SVM-rank)
- Altun, Yasemin, Mikhail Belkin, and David A. Mcallester. "Maximum margin semi-supervised learning for structured variables." *Advances in neural information processing systems*. 2005.

Acknowledgement:

The slides use materials from (1) Carla P. Gomes's slides for SVM (2) Some slides borrowed from the course slides from cs231n at Stanford

- Hw: page 20, 27,37, 40

- Coding: you can use any programming language you prefer.
- You need to submit your source code, an executable, and figures for the visualization results.
- Recommendation: Anaconda (it is a free Python distribution with most popular packages, very convenient for scientific computing, and producing nice figures).