Deep Learning 1

Jian Li

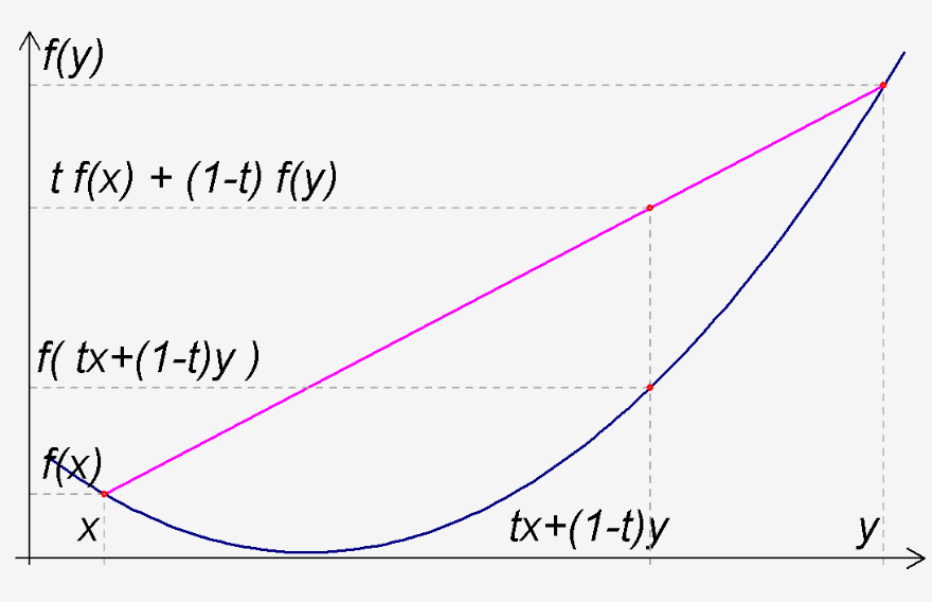
IIIS, Tsinghua University

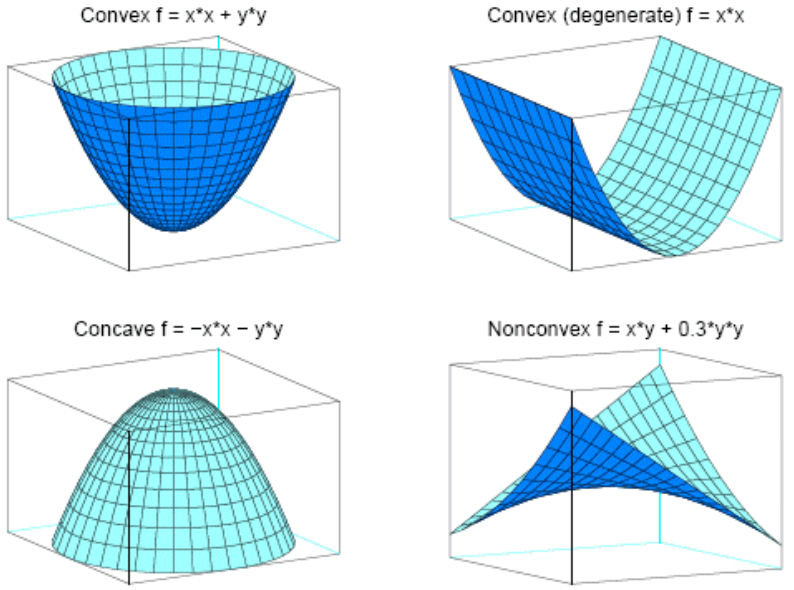
Optimization Basics

Convex Functions

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

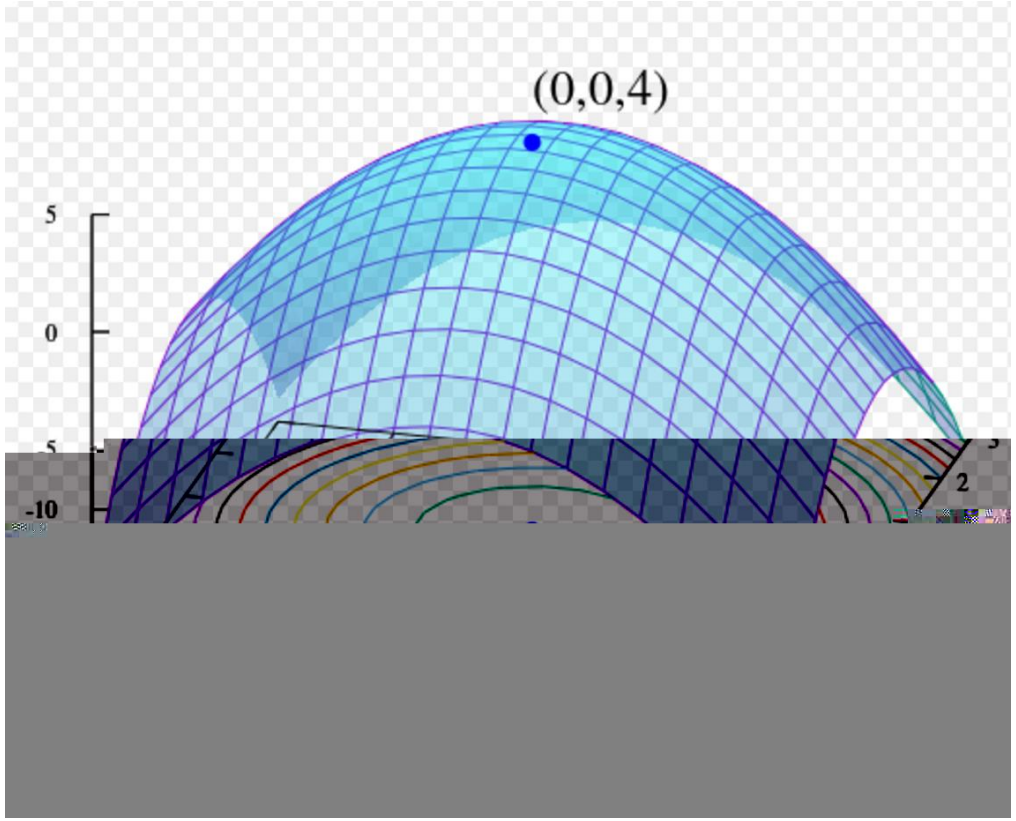
(the above two definitions are equivalent for continuous functions)





Concave functions

- *f* is concave if ς*f* is convex



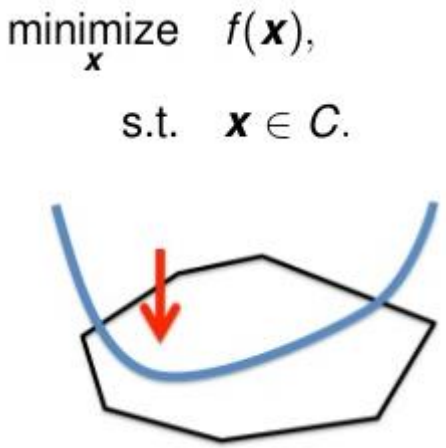
Convex Optimization

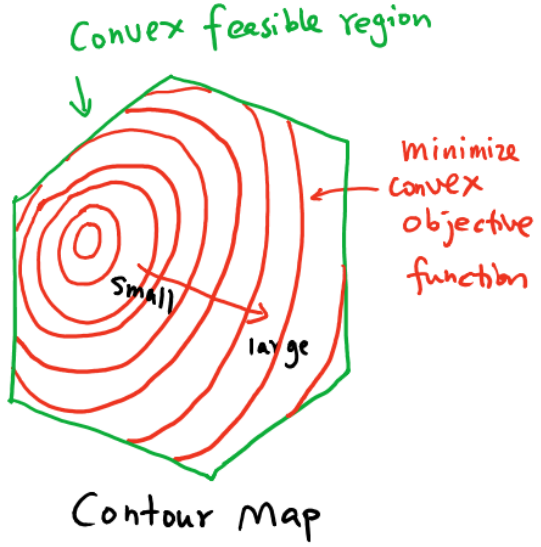
- Convex optimization: $f_0, f_1, \ldots$ are convex functions, h_j are linear functions

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

f_0:objective function

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

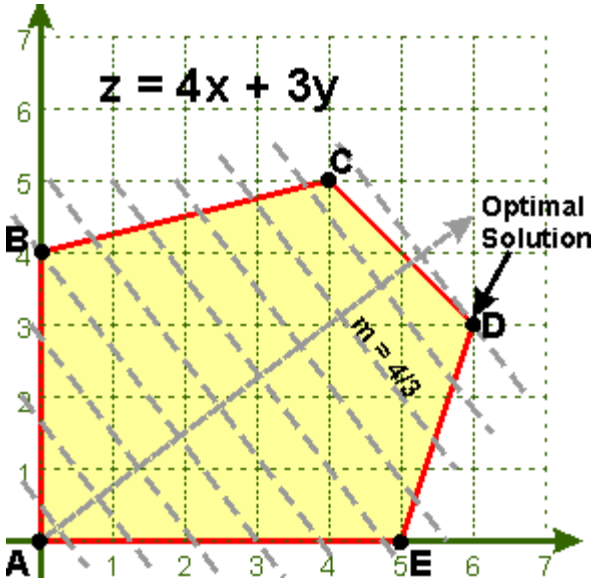




Convex Optimization

- Linear Programming:

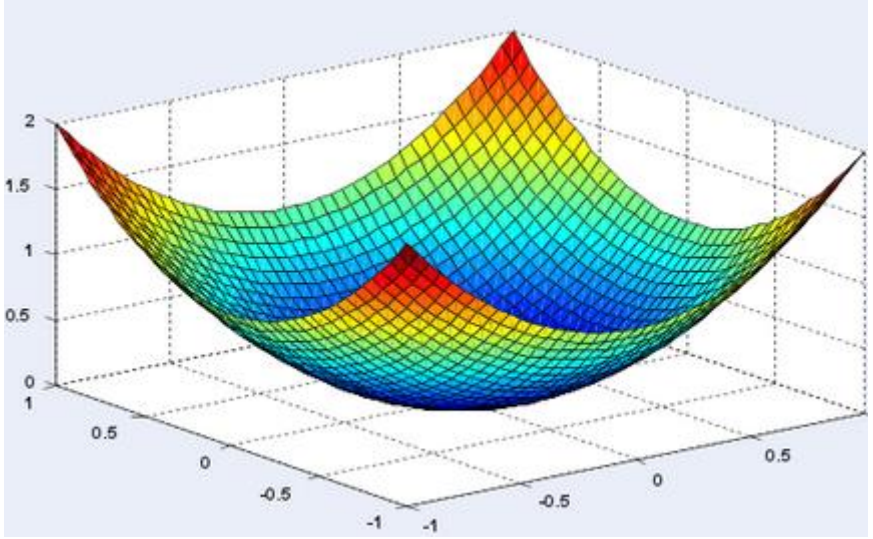
$$min\ c^T x \text{ subject to } Ax \geq b, x \geq 0$$



Convex Optimization

- Quadratic Programming: P is positive semi-definite

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



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

Convex Optimization

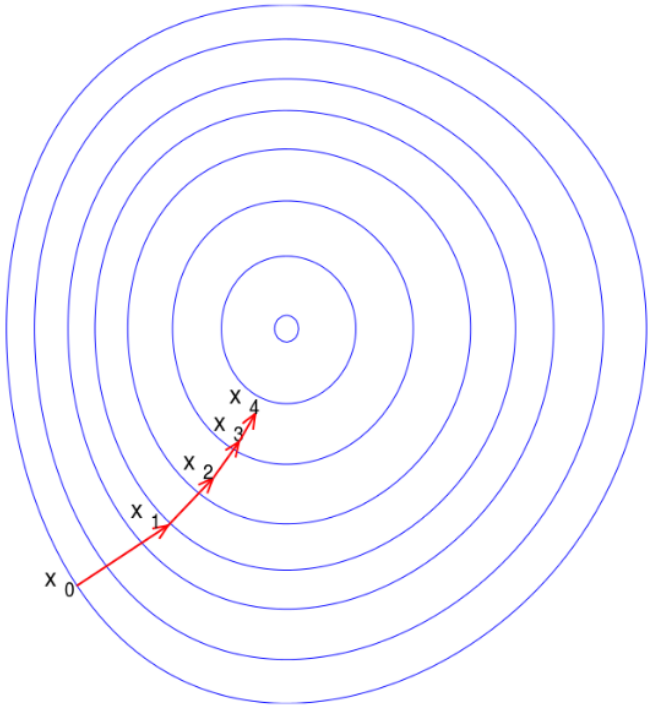
- Second Order Cone Program (SOCP)
- Geometric Programming (GP)
- Semidefinite Programming (SDP)

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

Sub-gradient Descent

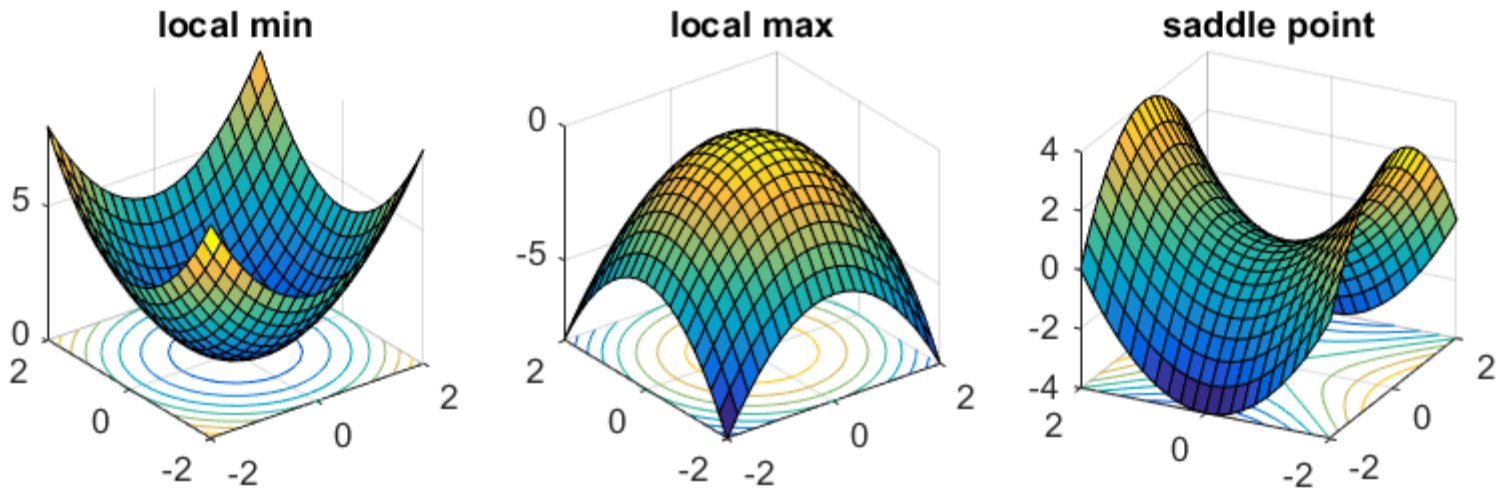
Sub-gradient Descent for unconstraint minimization:

- 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: $\alpha_k = O(1/k)$, $\alpha_k = O(1/\sqrt{k})$ΣXϕ
- Testing convergence
  - $|f\left(x^{(k+1)}\right) - f(x^{(k)})|$ is small enough
  - Xϕ



Sub-gradient Descent

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

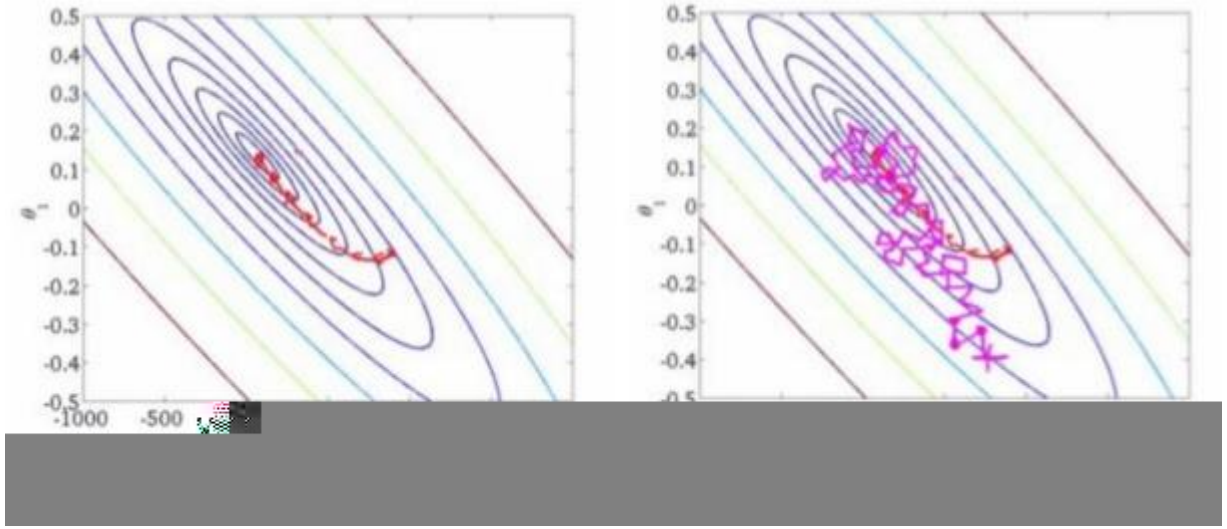


Stochastic Gradient Descent

- 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. $\ell_i(w) = (w^T x_i - y_i)^2$

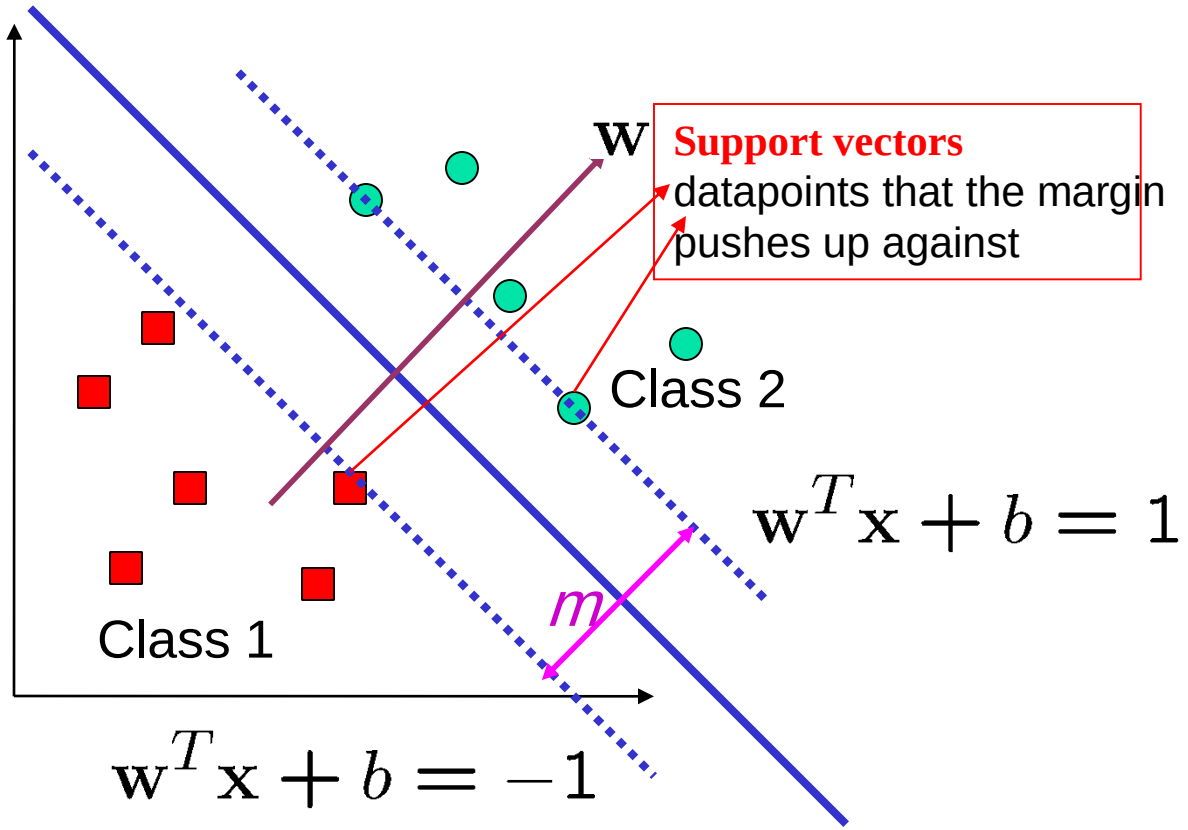
- SGD: Iterate until converge: $w^{(k+1)} = w^{(k)} - \alpha_k h_k$
  - h_k is a random vector such that $\mathrm{E}[h_k] = g_k$
  - For $loss(w) = (1/n)\sum_{i=1}^{n} \ell_i(w)$, we can choose $h_k = \nabla \ell_i(w^{(k)})$ where i is chosen uniformly at random from [n]
    - It is easy to see that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

GD vs SGD



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.
52y⁄ú f22⁄ dzLJ úKŜ ⁄yúŜ⁄yŜúφ It is simple geometry. Do it by yourself

The Optimization Problem

- Let $\{x_1, ..., x_n\}$ 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 $\Rightarrow$

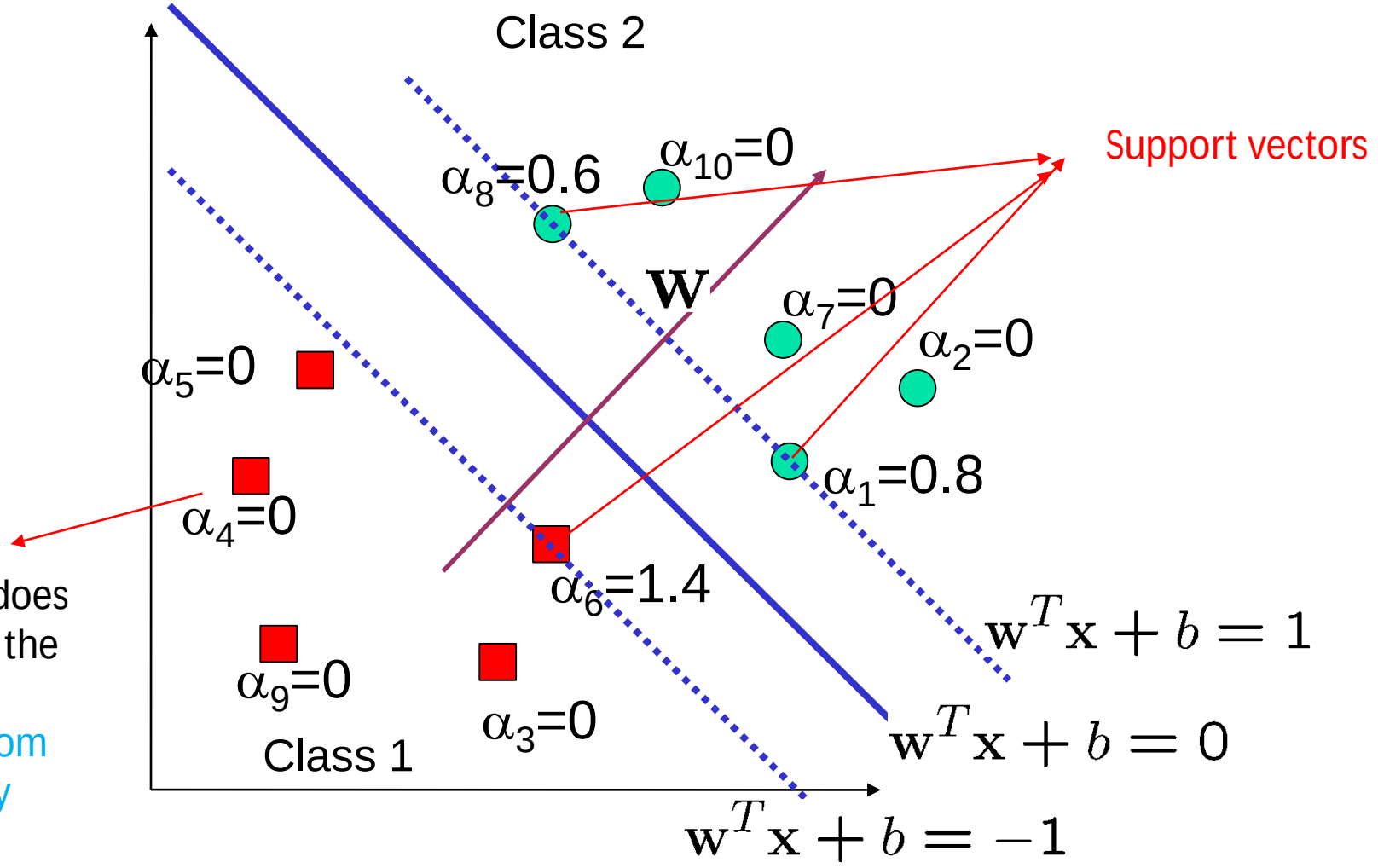
$$y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1, \qquad \forall i$$

- 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

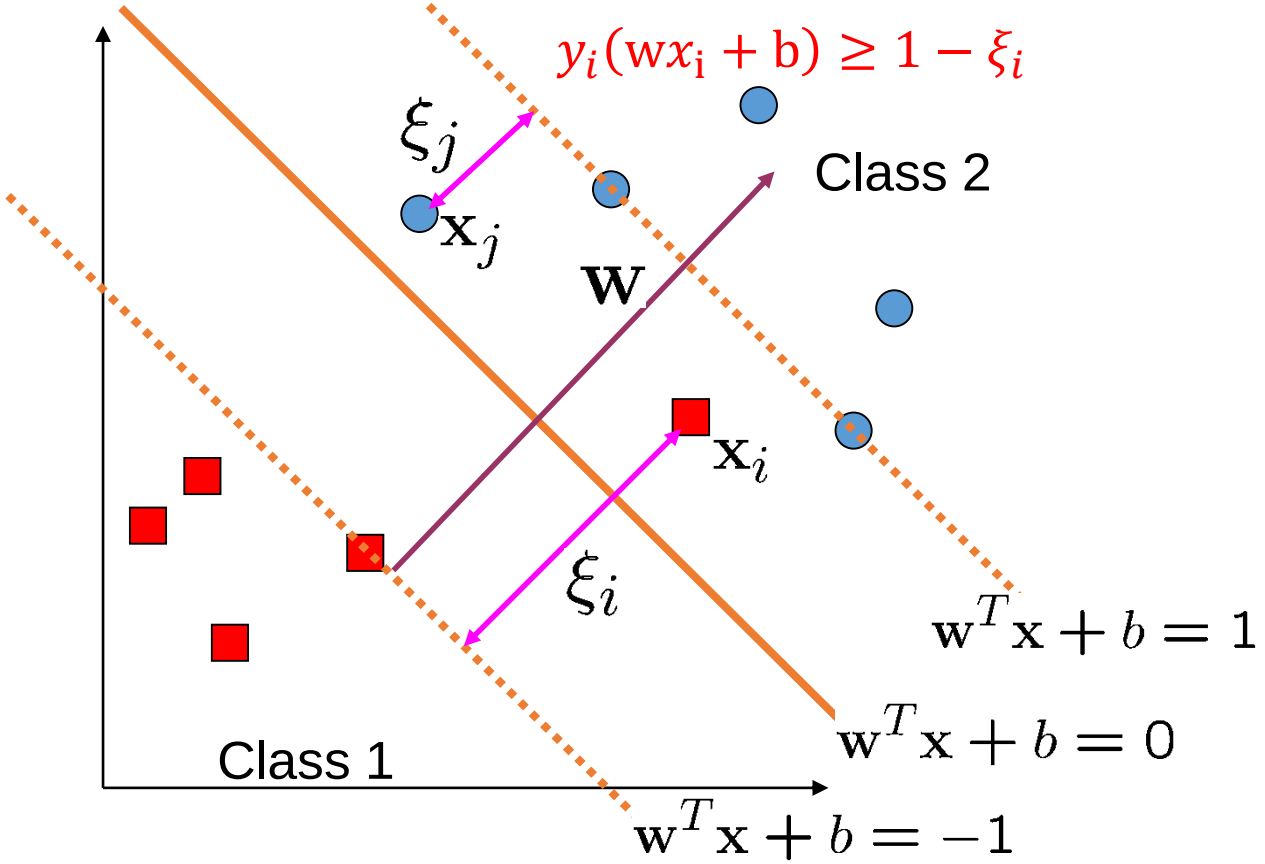


The boundary does not depend on the internal pts (can be seen from complementary slackness)

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

Non-linearly Separable Problems

- ² Ŝ l·ff2ø áŜNJNJ2NJé 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
$y_i(\mathrm{w}x_\mathrm{i} + \mathrm{b}) \geq 1 - \xi_i$ 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:

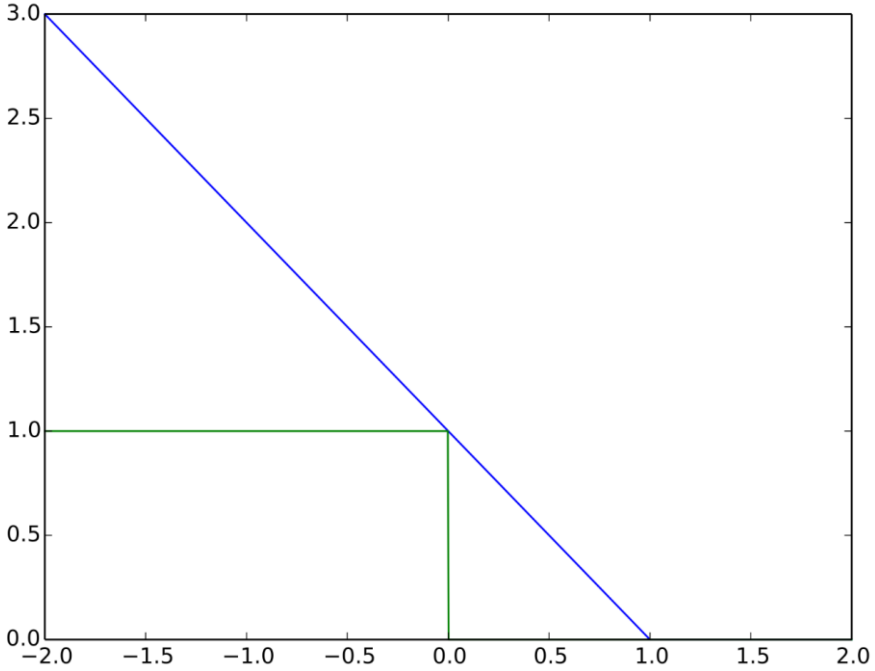
$$\ell(t) = \max(0, 1 - t)$$

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

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

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

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, $f(x) = w\,x + b$
- 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.

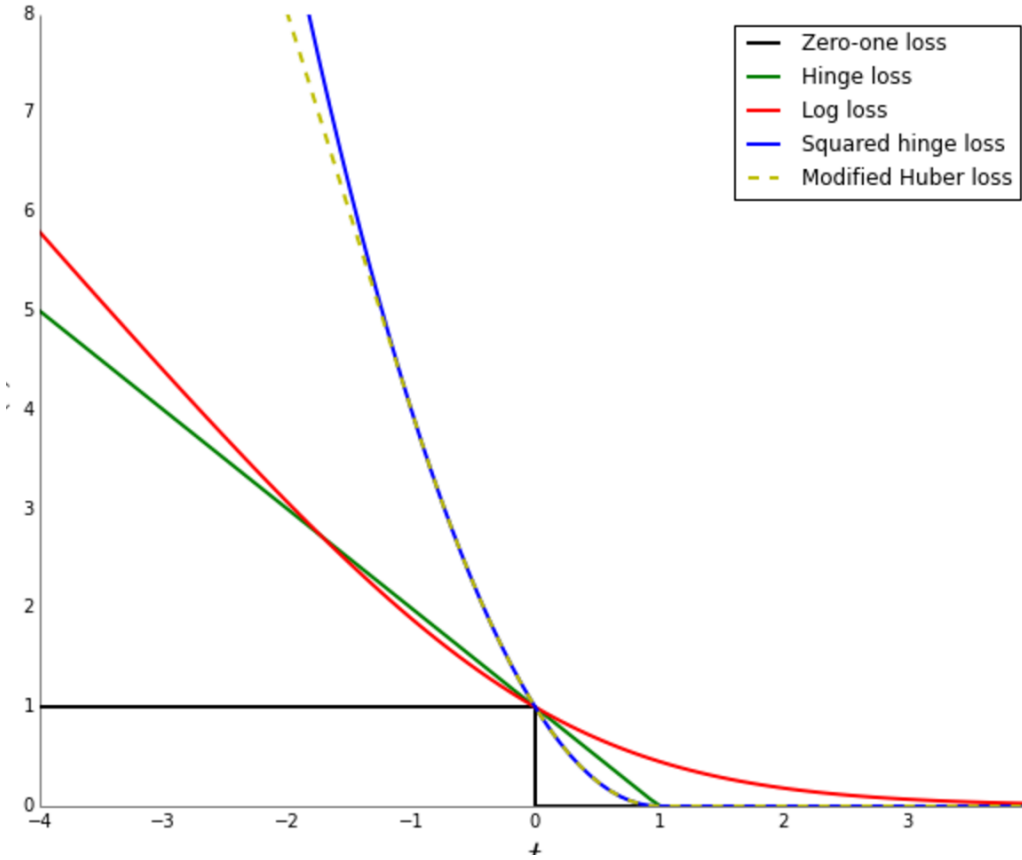
Other (surrogate) Loss Functions

- Log (logistic) loss:

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

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

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

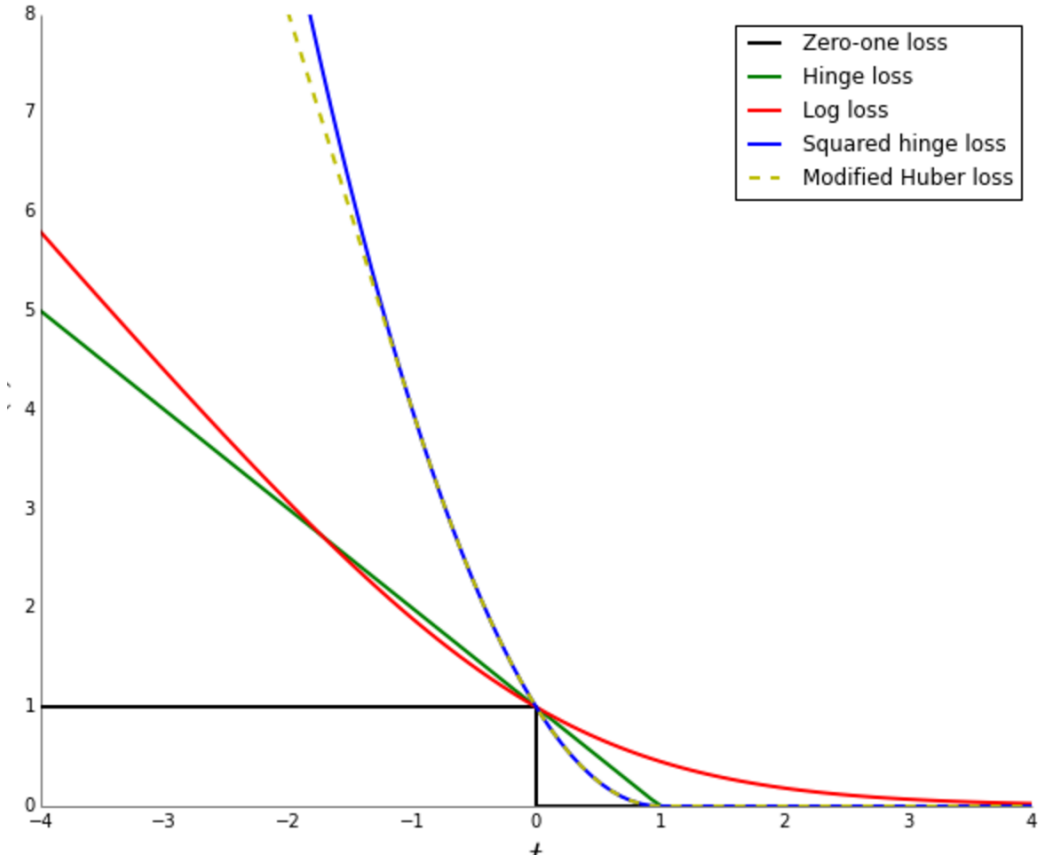


Other (surrogate) Loss Functions

- Modified Huber Loss:

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

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



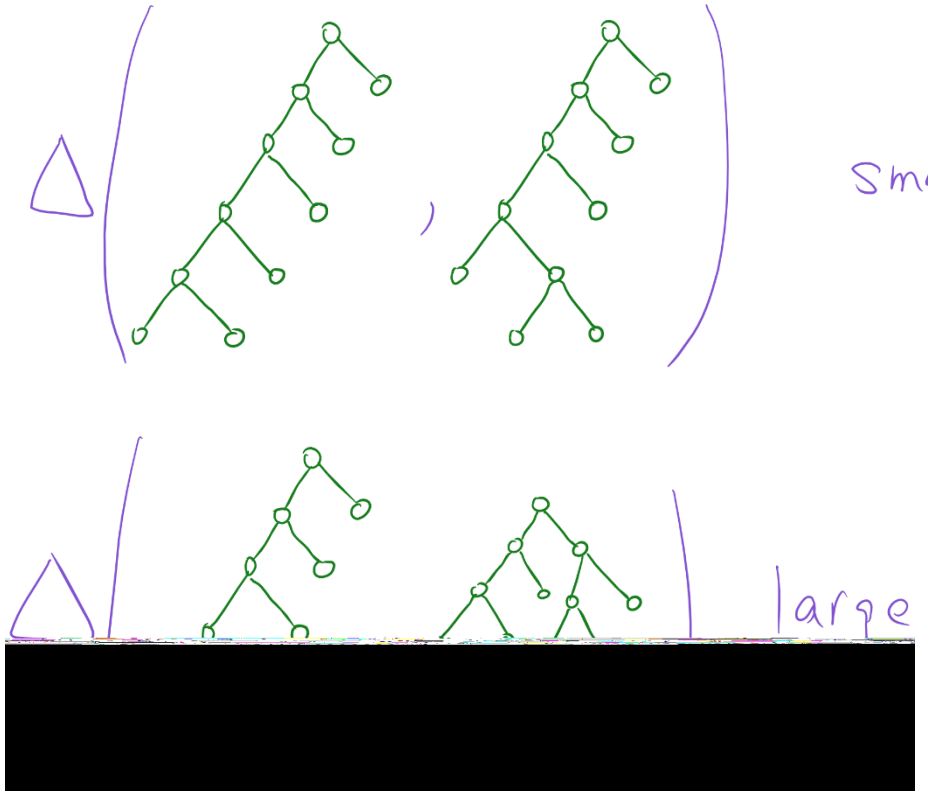
- Exponential loss (in boosting)
- Sigmoid loss (nonconvex)

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
      - $|Y| = 2^k$
      - $\Delta(Y, Y') = Hamming(Y, Y')$
    - Ex 2: Taxonomy classification
      - Each label is a tree of size *k*
      - $\Delta(Y, Y') = tree\_edit\_distance(Y, Y')$
  - ² Ŝ ŏI·yQű I·ŦŦƵNŘ ű2 KI·ØŜ ƶyŜ ØŜλƎKű ØŜŏű2NJ Ŧ2NJ ŜI·ŏK fI·ōŜfΦ
  - 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)



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

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

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

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

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

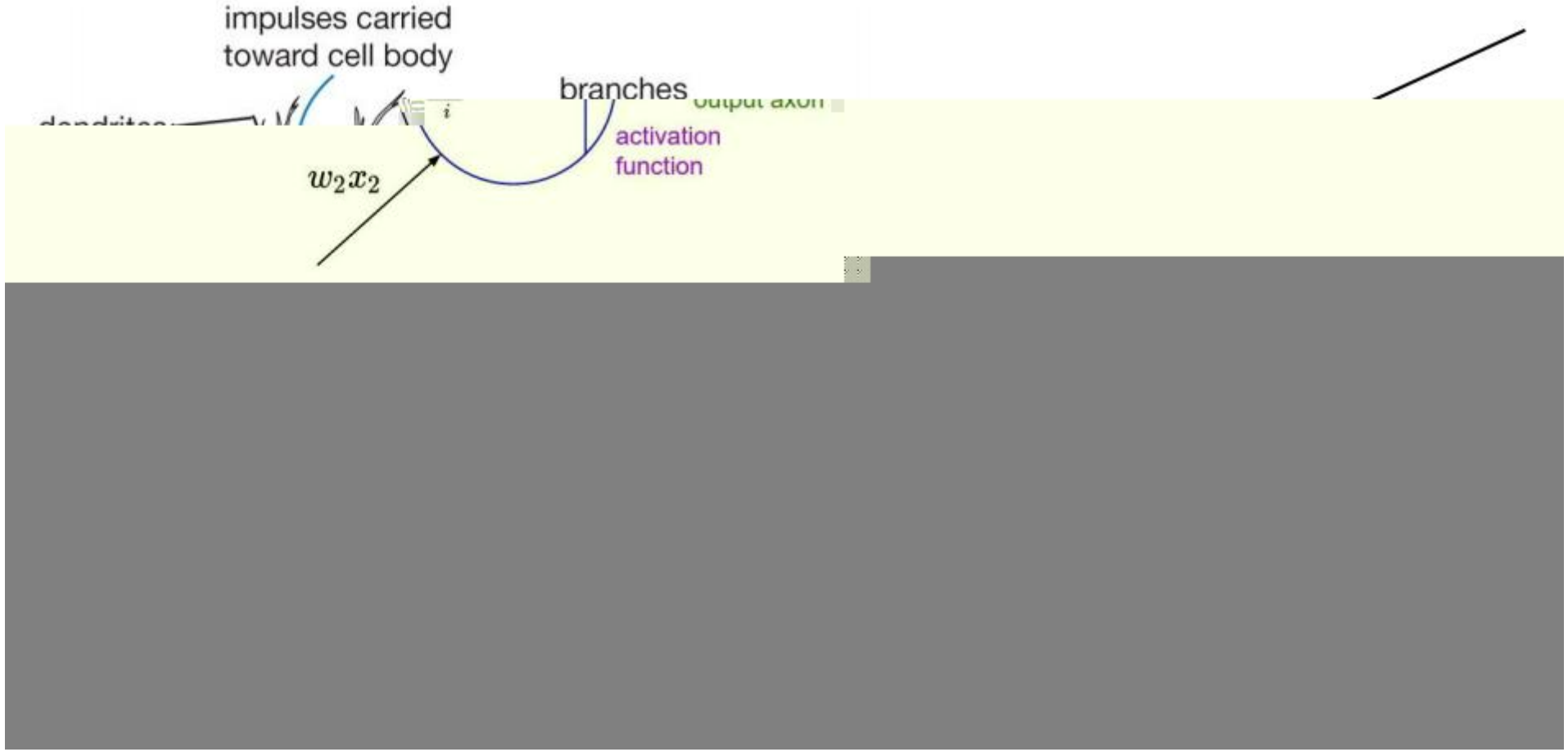
If we predict *y* and [redacted] is large, we need to pay a large loss (ξ_i is large)

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

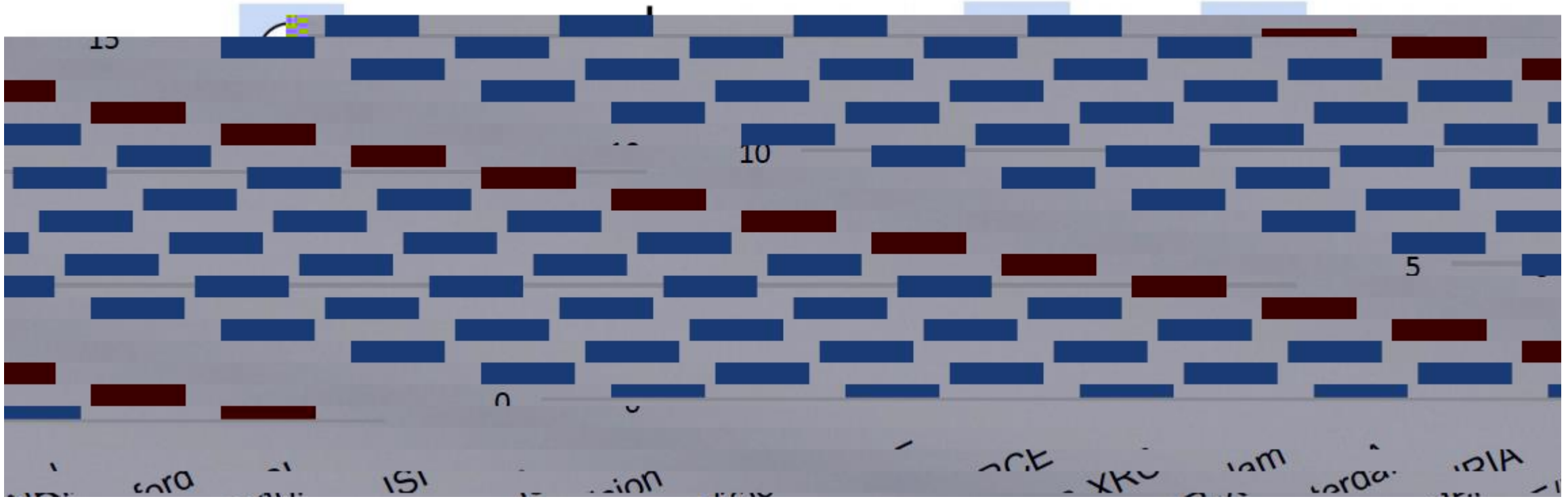
Neural Network Basics

Artificial Neural Networks

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



Artificial Neural Networks

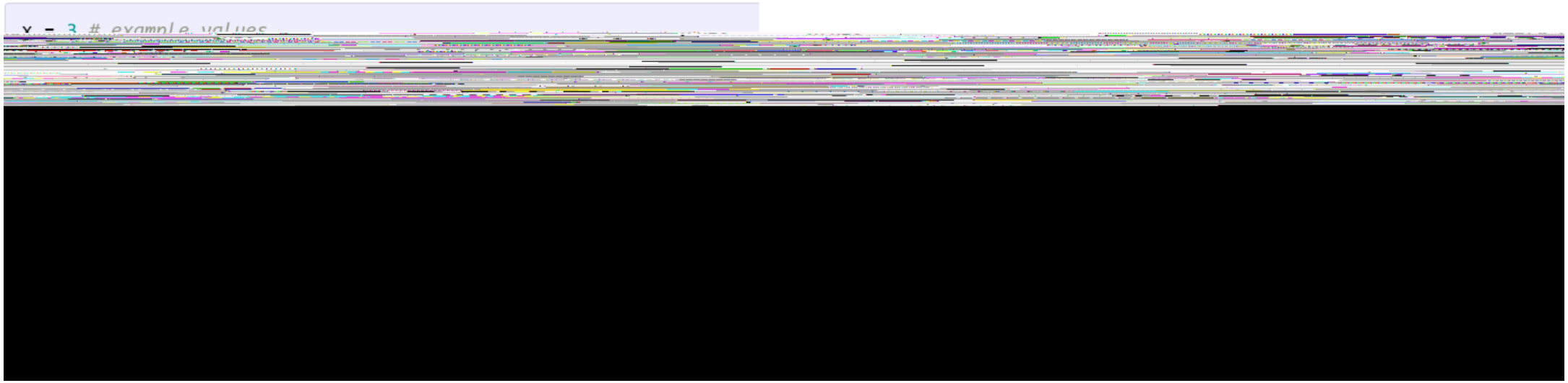


Forward Computation

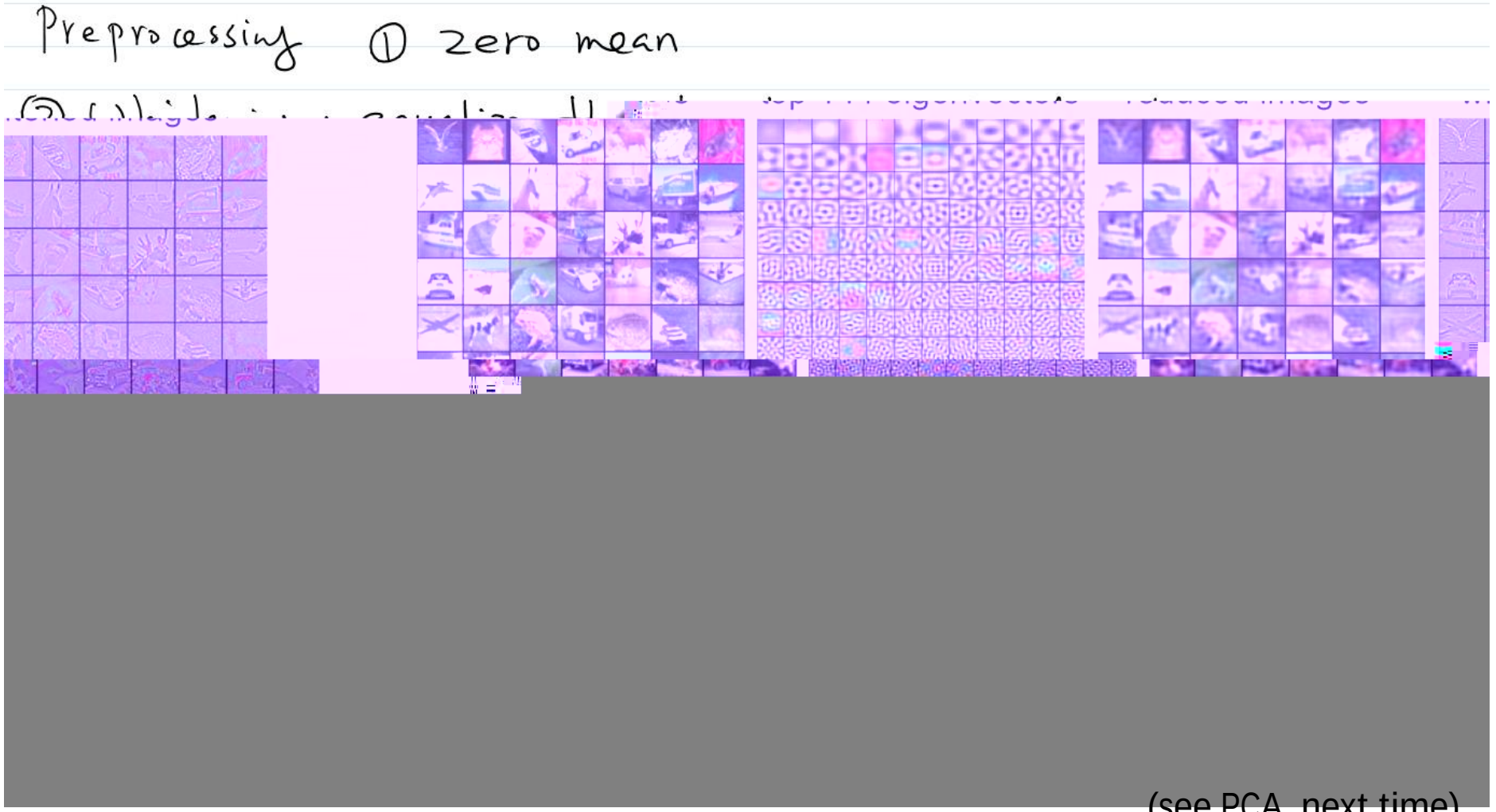
$$f(x, y, z) = (x + y)z$$

Back Propagation

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



Training NN



(see PCA. next time)

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

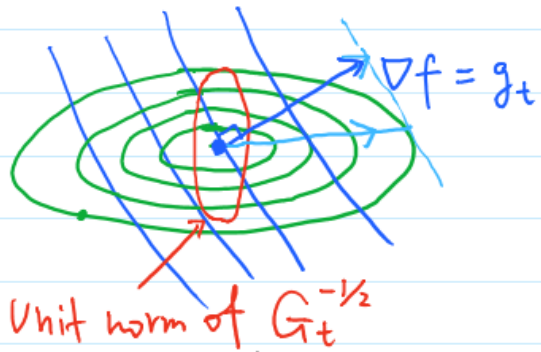
Adagrad [Duchi, Hazan, Singer JMLR11]

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.

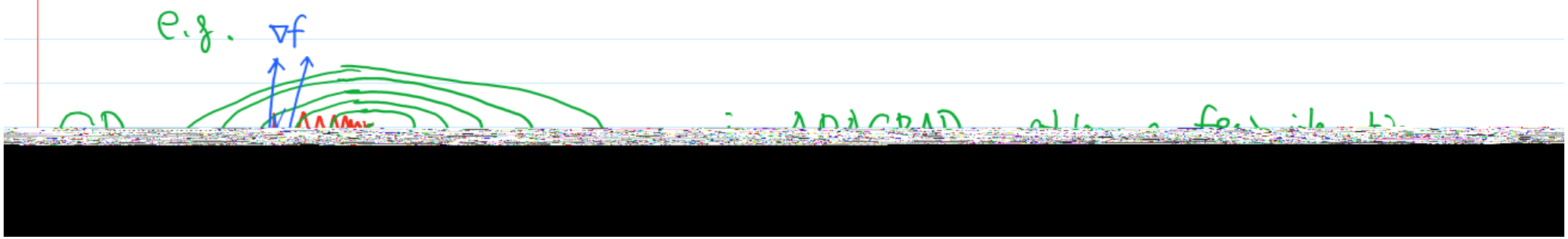
Adagrad

$$(\text{diagnal adaptation}) x_{t+1} = \operatorname*{argmin}_{x \in X} \| x - (x_t - \eta \operatorname{diag}(G_t)^{-1/2} g_t \|_{\operatorname{diag}(G_t)^{1/2}}$$

$$\operatorname{diag}(G_t) = \begin{pmatrix} \sum_{\tau=1}^{t} g_{\tau 1}^2 & & \\ & \ddots & \\ & & \sum_{\tau=1}^{t} g_{\tau i}^2 \end{pmatrix}$$

try to make the first order method better conditioned.

related to FTRL, dual averaging - proximal method.



ADAM

ADMA:

$$g_t \leftarrow \nabla f$$

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

$$v_t \leftarrow \beta_2 \times v_{t-1} + (1-\beta_2) \cdot \begin{pmatrix} g_{t1}^2 \\ \vdots \\ g_{ti}^2 \\ \vdots \end{pmatrix}$$

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

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

$$x_{t+1} \leftarrow x_t - \gamma \, (\overline{v}_t)^{-1/2} \, \overline{m}_t$$

Batch Normalization [Ioffe, Szegedy]

For a layer of input vector x:

Normalize:

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

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

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 ς 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) / l·ℕfl· t℗ D₴ϒŜăΩă ăfϏŘŜă ₮₴ℕ {± a óнú 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).