Linear models

Subhransu Maji

CMPSCI 689: Machine Learning

24 February 2015

26 February 2015

Overview

- Linear models
  - Perceptron: model and learning algorithm combined as one
  - Is there a better way to learn linear models?
- We will separate models and learning algorithms
  - Learning as optimization
  - Surrogate loss function
  - Regularization

} model design

  - Gradient descent
  - Batch and online gradients
  - Subgradient descent
  - Support vector machines

} optimization

Learning as optimization

$$\min_{\mathbf{w}} \sum_{n} \mathbf{1}[y_n \mathbf{w}^T \mathbf{x}_n < 0]$$

fewest mistakes

- The perceptron algorithm will find an optimal **w** if the data is separable
  - efficiency depends on the margin and norm of the data
- However, if the data is not separable, optimizing this is NP-hard
  - i.e., there is no efficient way to minimize this unless **P=NP**

Learning as optimization

$$\min_{\mathbf{w}} \sum_{n} \mathbf{1}[y_n \mathbf{w}^T \mathbf{x}_n < 0] + \lambda R(\mathbf{w})$$

hyperparameter

fewest mistakes

simpler model

- In addition to minimizing training error, we want a simpler model
  - Remember our goal is to minimize generalization error
  - Recall the bias and variance tradeoff for learners
- We can add a regularization term *R(**w**)* that prefers simpler models
  - For example we may prefer decision trees of shallow depth
- Here λ is a hyperparameter of optimization problem

Learning as optimization

$$\min_{\mathbf{w}} \sum_{n} \mathbf{1}[y_n \mathbf{w}^T \mathbf{x}_n < 0] + \lambda R(\mathbf{w})$$

hyperparameter

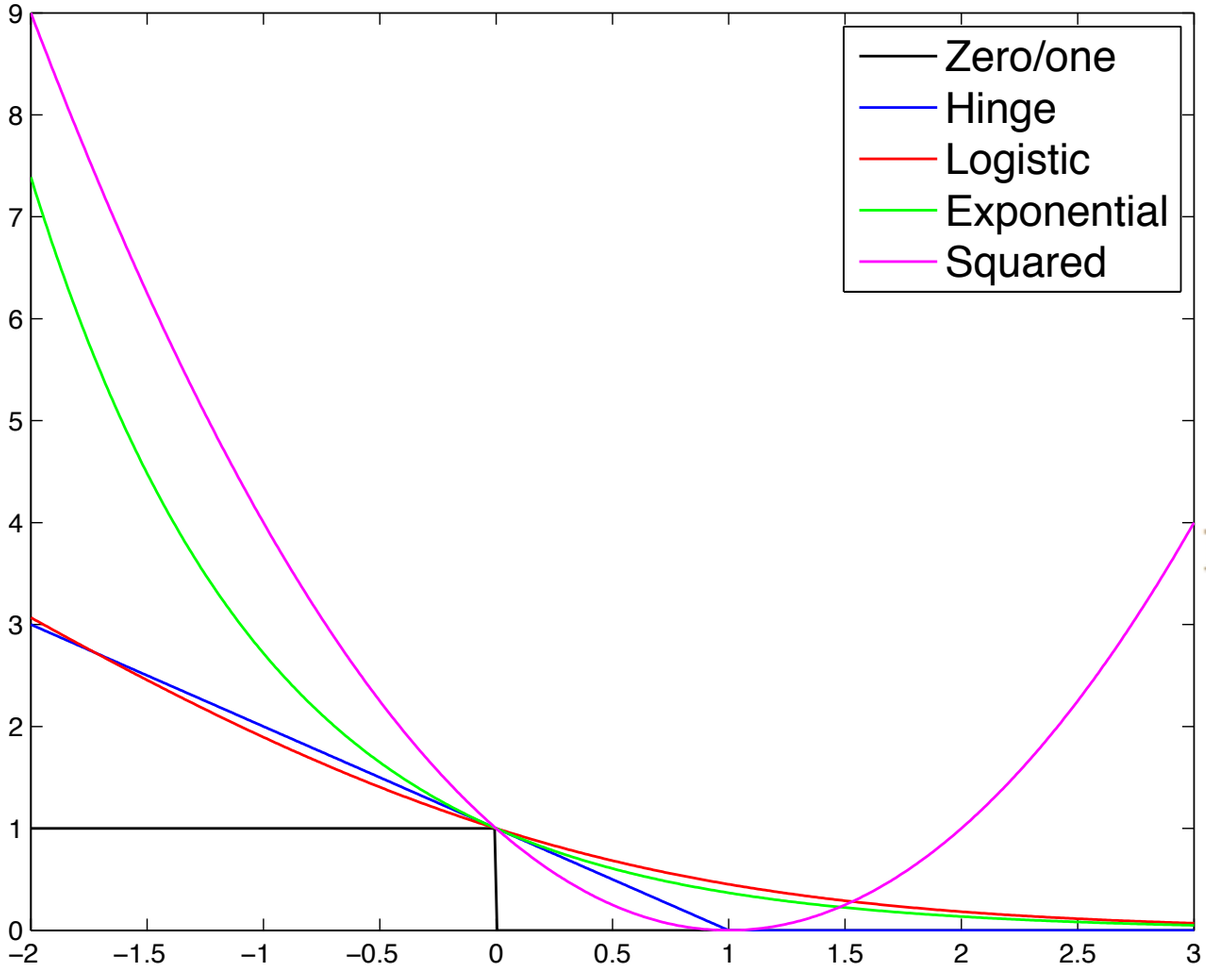
fewest mistakes

simpler model

- The questions that remain are:
  - What are good ways to adjust the optimization problem so that there are efficient algorithms for solving it?
  - What are good regularizations $R(\mathbf{w})$ for hyperplanes?
  - Assuming that the optimization problem can be adjusted appropriately, what algorithms exist for solving the regularized optimization problem?

Convex surrogate loss functions

- Zero/one loss is hard to optimize
  - Small changes in **w** can cause large changes in the loss
- Surrogate loss: replace Zero/one loss by a smooth function
  - Easier to optimize if the surrogate loss is convex
- Examples:



$$y = +1 \qquad \hat{y} \leftarrow \mathbf{w}^T \mathbf{x}$$

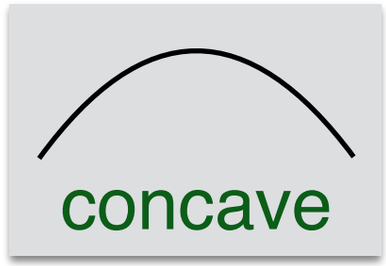
Zero/one: $\ell^{(0/1)}(y, \hat{y}) = \mathbf{1}[y\hat{y} \leq 0]$

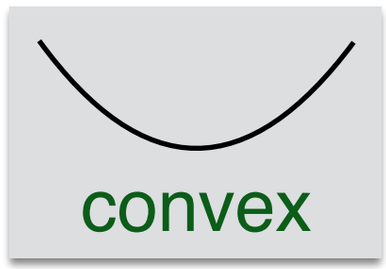
Hinge: $\ell^{(\text{hin})}(y, \hat{y}) = \max\{0, 1 - y\hat{y}\}$

Logistic: $\ell^{(\log)}(y, \hat{y}) = \frac{1}{\log 2} \log\left(1 + \exp[-y\hat{y}]\right)$

Exponential: $\ell^{(\exp)}(y, \hat{y}) = \exp[-y\hat{y}]$

Squared: $\ell^{(\text{sqr})}(y, \hat{y}) = (y - \hat{y})^2$





Weight regularization

- What are good regularization functions $R(\boldsymbol{w})$ for hyperplanes?
- We would like the weights —
  - To be small —
    - Change in the features cause small change to the score
    - Robustness to noise
  - To be sparse —
    - Use as few features as possible
    - Similar to controlling the depth of a decision tree
- This is a form of inductive bias

Weight regularization

- Just like the surrogate loss function, we would like *R(**w**)* to be convex
- Small weights regularization

$$R^{(\text{norm})}(\mathbf{w}) = \sqrt{\sum_d w_d^2} \qquad R^{(\text{sqrd})}(\mathbf{w}) = \sum_d w_d^2$$

- Sparsity regularization

$$R^{(\text{count})}(\mathbf{w}) = \sum_d \mathbf{1}[|w_d| > 0]$$

not convex

- Family of "p-norm" regularization

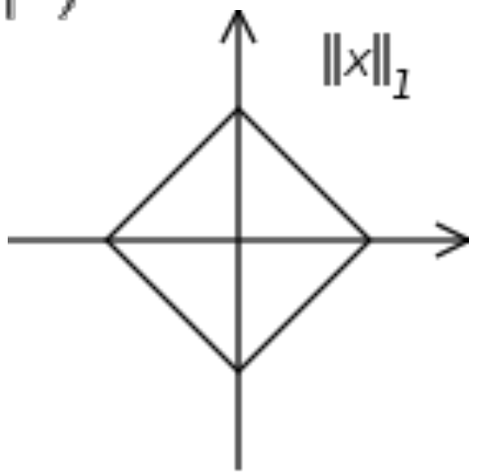
$$R^{(\text{p-norm})}(\mathbf{w}) = \left(\sum_d |w_d|^p \right)^{1/p}$$

Contours of p-norms

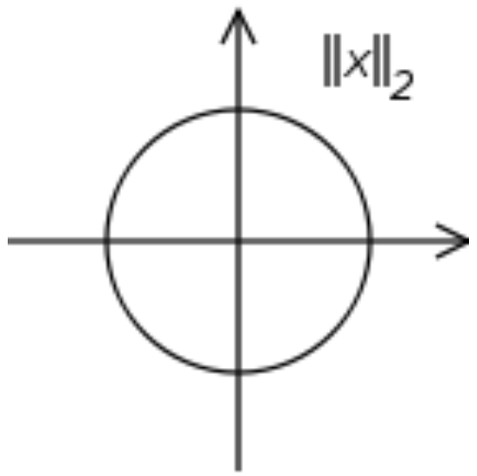
$$\|x\|_p = \left(|x_1|^p + |x_2|^p + \cdots + |x_n|^p\right)^{\frac{1}{p}}$$

convex for $p \geq 1$

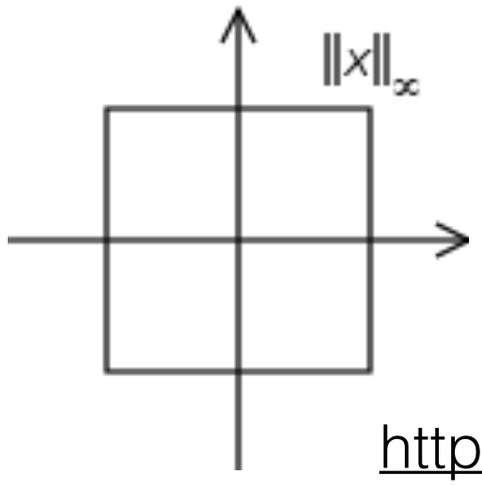
$$\|x\|_1 = \sum_{i=1}^{n} |x_i|$$



$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} |x_i|^2}$$



$$\|x\|_\infty = \max_{i=1,\ldots,n} |x_i|$$



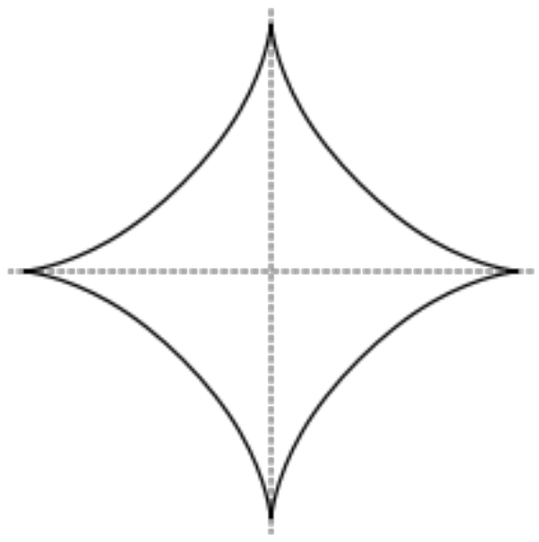
http://en.wikipedia.org/wiki/Lp_space

Contours of p-norms

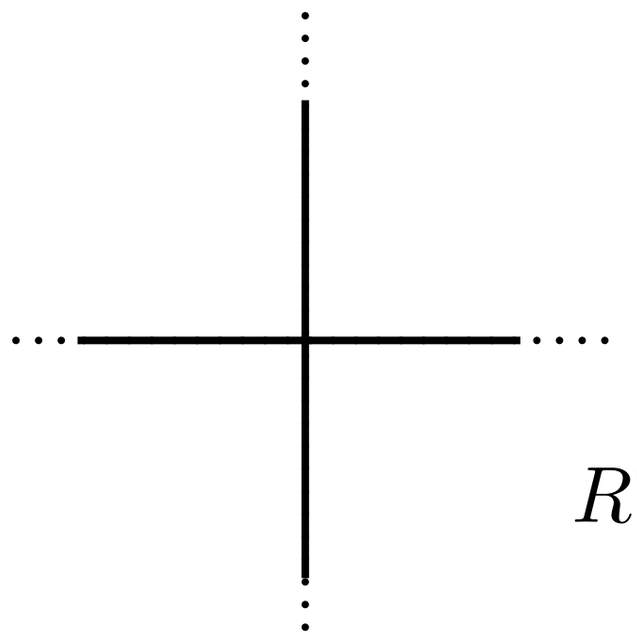
$$\|x\|_p = (|x_1|^p + |x_2|^p + \cdots + |x_n|^p)^{\frac{1}{p}}$$

not convex for $0 \leq p < 1$

$$p = \frac{2}{3}$$



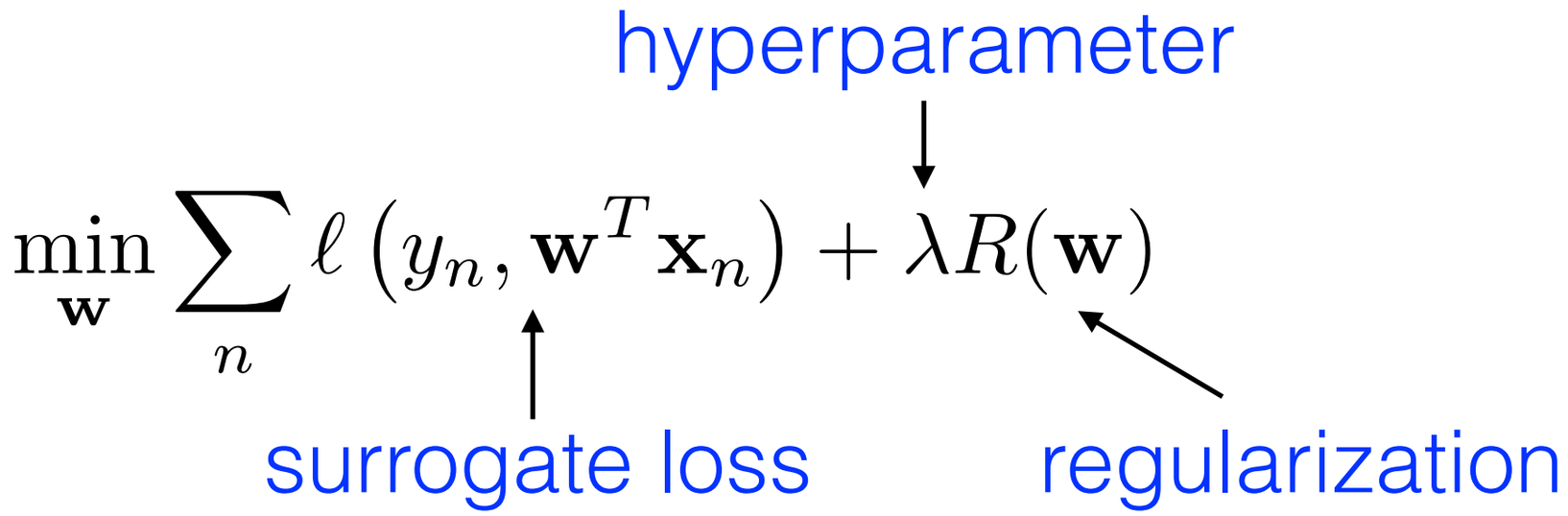
$$p = 0$$



$$R^{(\text{count})}(\mathbf{w}) = \sum_d \mathbf{1}[|w_d| > 0]$$

http://en.wikipedia.org/wiki/Lp_space

General optimization framework

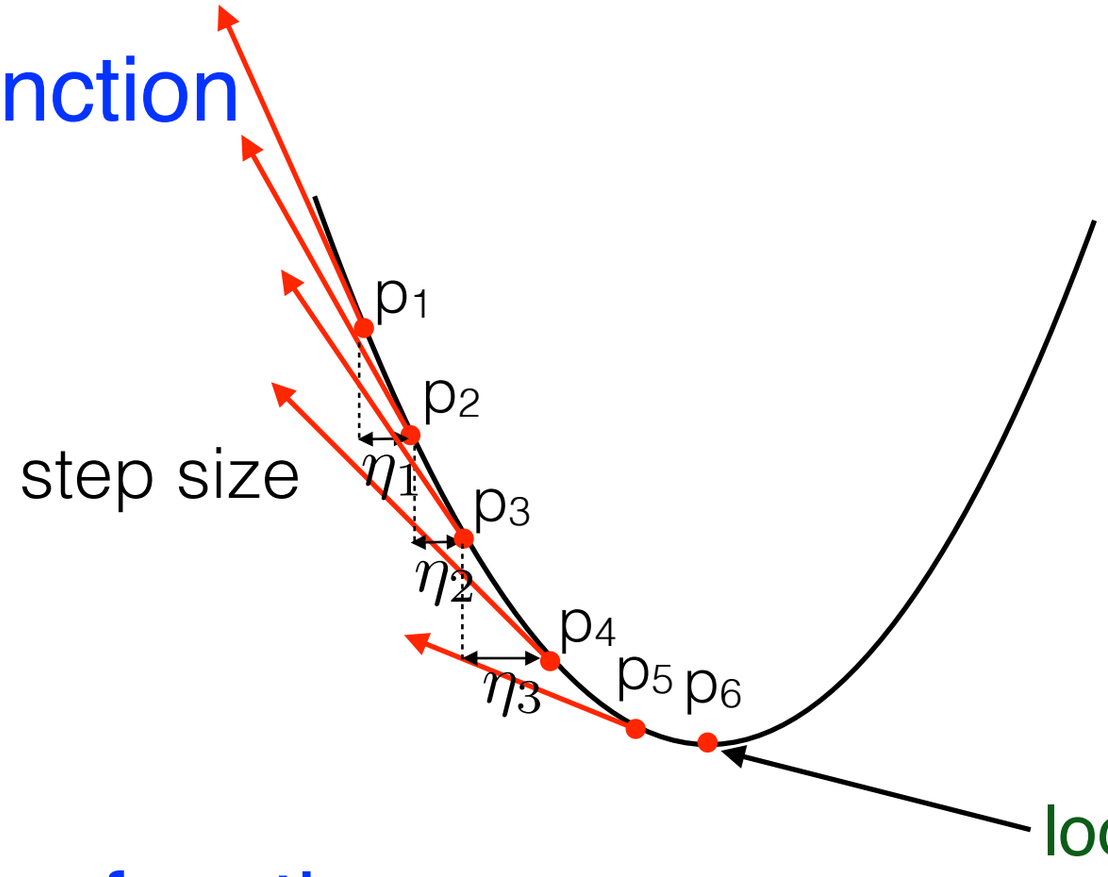


- Select a suitable:
  - convex surrogate loss
  - convex regularization
- Select the hyperparameter λ
- Minimize the regularized objective with respect to **w**
- This framework for optimization is called Tikhonov regularization or generally Structural Risk Minimization (SRM)

http://en.wikipedia.org/wiki/Tikhonov_regularization

Optimization by gradient descent

Convex function



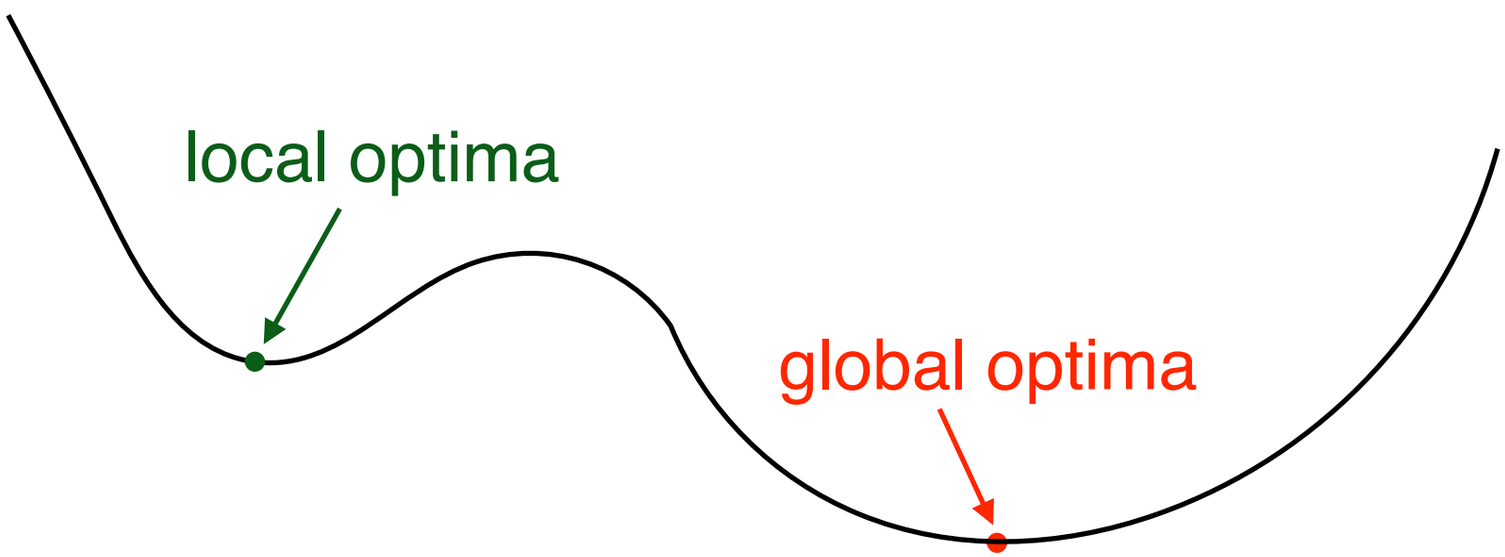
$$g^{(k)} \leftarrow \nabla_p F(p)|_{p_k}$$

compute gradient at the current location

$$p_{k+1} \leftarrow p_k - \eta_k g^{(k)}$$

take a step down the gradient

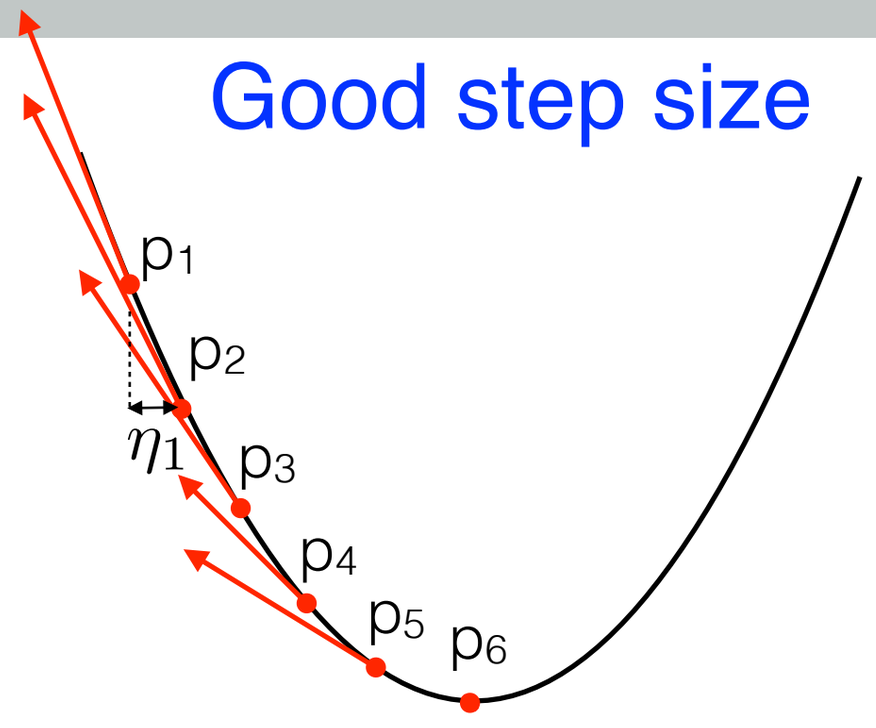
Non-convex function



Choice of step size

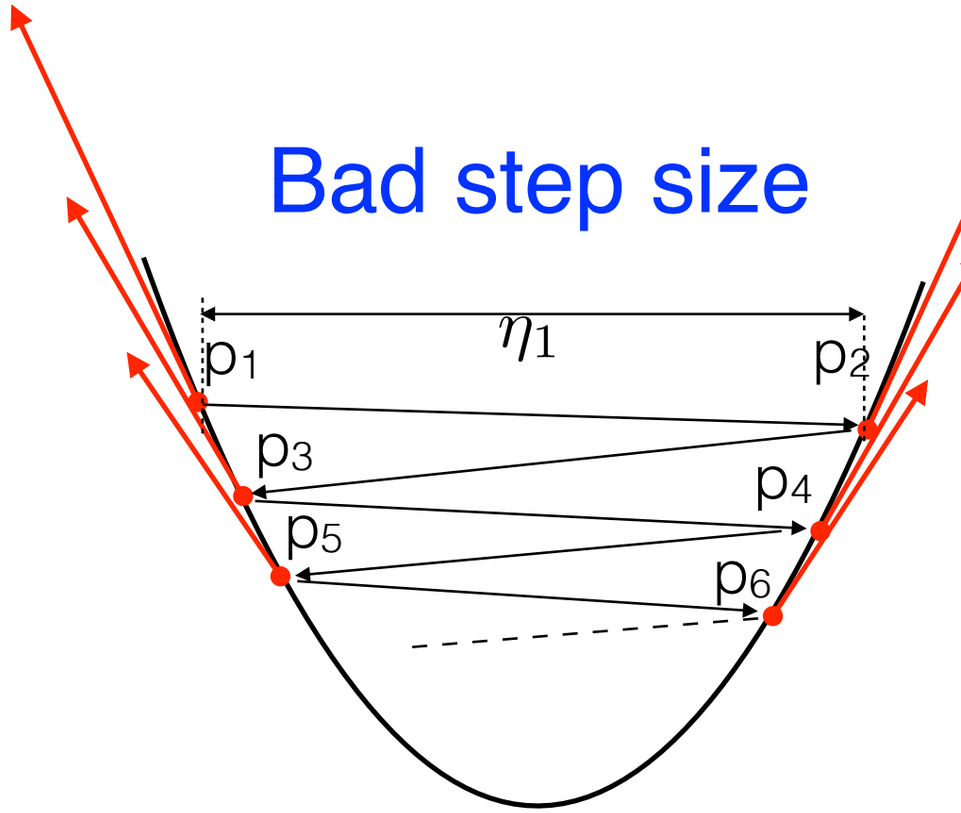
- The step size is important —
  - too small: slow convergence
  - too large: no convergence
- A strategy is to use large step sizes initially and small step sizes later:

$$\eta_t \leftarrow \eta_0/(t_0 + t)$$



- There are methods that converge faster by adapting step size to the curvature of the function
  - Field of convex optimization

http://stanford.edu/~boyd/cvxbook/



Example: Exponential loss

$$\mathcal{L}(\mathbf{w}) = \sum_n \exp(-y_n \mathbf{w}^T \mathbf{x}_n) + \frac{\lambda}{2}||\mathbf{w}||^2 \qquad \text{objective}$$

$$\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_n -y_n \mathbf{x}_n \exp(-y_n \mathbf{w}^T \mathbf{x}_n) + \lambda \mathbf{w} \qquad \text{gradient}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left(\sum_n -y_n \mathbf{x}_n \exp(-y_n \mathbf{w}^T \mathbf{x}_n) + \lambda \mathbf{w} \right) \qquad \text{update}$$

loss term

$$\mathbf{w} \leftarrow \mathbf{w} + c y_n \mathbf{x}_n$$

high for misclassified points

similar to the perceptron update rule!

regularization term

$$\mathbf{w} \leftarrow (1 - \eta\lambda)\mathbf{w}$$

shrinks weights towards zero

Batch and online gradients

$$\mathcal{L}(\mathbf{w}) = \sum_n \mathcal{L}_n(\mathbf{w}) \quad \text{objective}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \frac{d\mathcal{L}}{d\mathbf{w}} \quad \text{gradient descent}$$

batch gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left(\sum_n \frac{d\mathcal{L}_n}{d\mathbf{w}} \right)$$

sum of n gradients

update weight after you see all points

online gradient

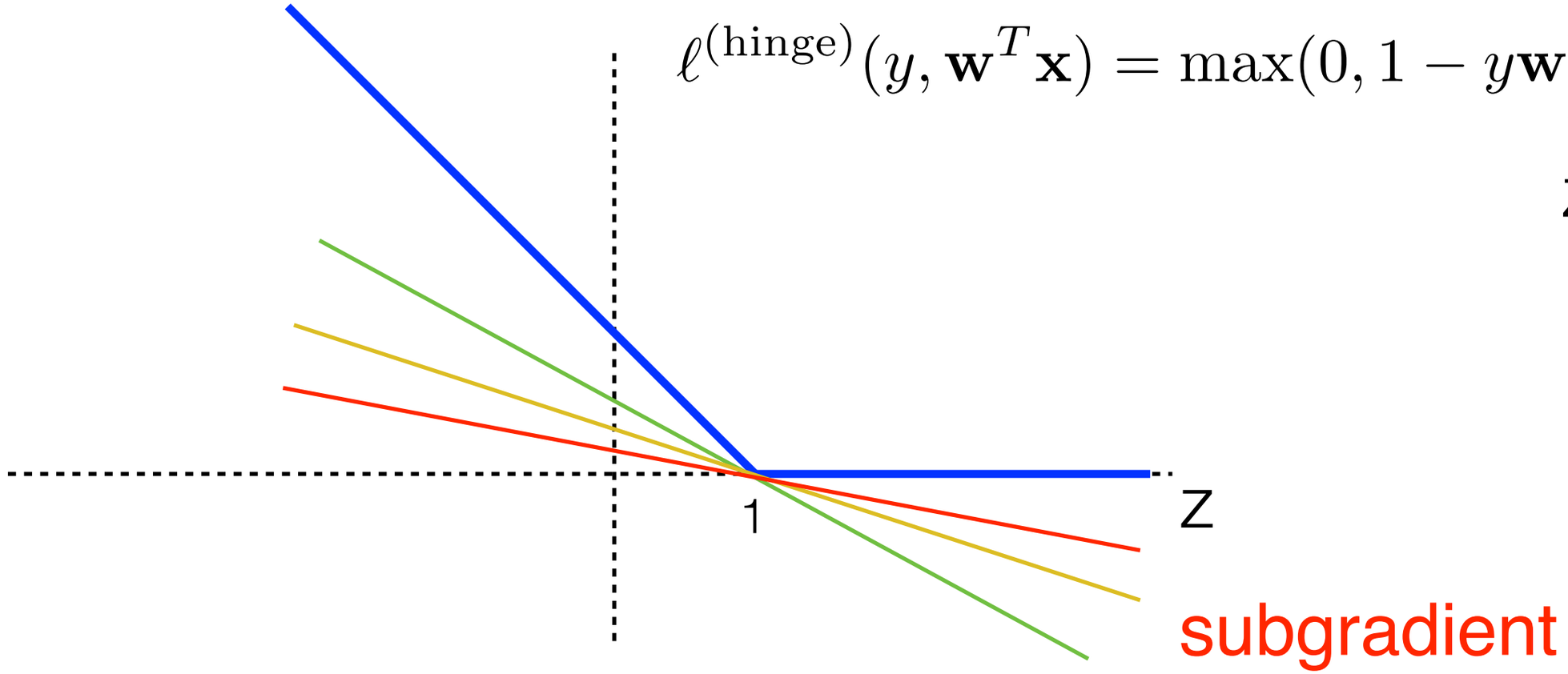
$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left(\frac{d\mathcal{L}_n}{d\mathbf{w}} \right)$$

gradient at n^{th} point

update weights after you see each point

Online gradients are the default method for multi-layer perceptrons

Subgradient



- The hinge loss is not differentiable at z=1
- Subgradient is any direction that is below the function
- For the hinge loss a possible subgradient is:

$$\frac{d\ell^{\text{hinge}}}{d\mathbf{w}} = \begin{cases} 0 & \text{if } y\mathbf{w}^T\mathbf{x} > 1 \\ -y\mathbf{x} & \text{otherwise} \end{cases}$$

Example: Hinge loss

$$\mathcal{L}(\mathbf{w}) = \sum_n \max(0, 1 - y_n \mathbf{w}^T \mathbf{x}_n) + \frac{\lambda}{2}||\mathbf{w}||^2 \quad \text{objective}$$

$$\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_n -\mathbf{1}[y_n \mathbf{w}^T \mathbf{x}_n \leq 1] y_n \mathbf{x}_n + \lambda \mathbf{w} \quad \text{subgradient}$$

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \left(\sum_n -\mathbf{1}[y_n \mathbf{w}^T \mathbf{x}_n \leq 1] y_n \mathbf{x}_n + \lambda \mathbf{w} \right) \quad \text{update}$$

loss term

$$\mathbf{w} \leftarrow \mathbf{w} + \eta y_n \mathbf{x}_n$$

only for points $y_n \mathbf{w}^T \mathbf{x}_n \leq 1$

perceptron update $y_n \mathbf{w}^T \mathbf{x}_n \leq 0$

regularization term

$$\mathbf{w} \leftarrow (1 - \eta\lambda)\mathbf{w}$$

shrinks weights towards zero

Example: Squared loss

$$\mathcal{L}(\mathbf{w}) = \sum_n \left(y_n - \mathbf{w}^T \mathbf{x}_n\right)^2 + \frac{\lambda}{2}||\mathbf{w}||^2$$ objective

↓ matrix notation

$$\underbrace{\begin{bmatrix} x_{1,1} & x_{1,2} & \dots & x_{1,D} \\ x_{2,1} & x_{2,2} & \dots & x_{2,D} \\ \vdots & \vdots & \ddots & \vdots \\ x_{N,1} & x_{N,2} & \dots & x_{N,D} \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{bmatrix}}_{w} = \underbrace{\begin{bmatrix} \sum_d x_{1,d} w_d \\ \sum_d x_{2,d} w_d \\ \vdots \\ \sum_d x_{N,d} w_d \end{bmatrix}}_{\hat{\mathbf{Y}}} \approx \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\hat{\mathbf{Y}}}$$

↓ equivalent loss

$$\min_w \quad \mathcal{L}(w) = \frac{1}{2}||\mathbf{X}w - \mathbf{Y}||^2 + \frac{\lambda}{2}||w||^2$$

Example: Squared loss

$$\min_{w} \quad \mathcal{L}(w) = \frac{1}{2} ||\mathbf{X}w - \mathbf{Y}||^2 + \frac{\lambda}{2} ||w||^2$$

objective

$$\begin{aligned}
\nabla_w \mathcal{L}(w) &= \mathbf{X}^\top (\mathbf{X}w - \mathbf{Y}) + \lambda w \\
&= \mathbf{X}^\top \mathbf{X} w - \mathbf{X}^\top \mathbf{Y} + \lambda w \\
&= \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right) w - \mathbf{X}^\top \mathbf{Y}
\end{aligned}$$

gradient

At optima the gradient=0

$$\begin{aligned}
& \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}\right) w - \mathbf{X}^\top \mathbf{Y} = 0 \\
\iff \quad & \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D\right) w = \mathbf{X}^\top \mathbf{Y} \\
\iff \quad & w = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D\right)^{-1} \mathbf{X}^\top \mathbf{Y}
\end{aligned}$$

exact closed-form solution

Matrix inversion vs. gradient descent

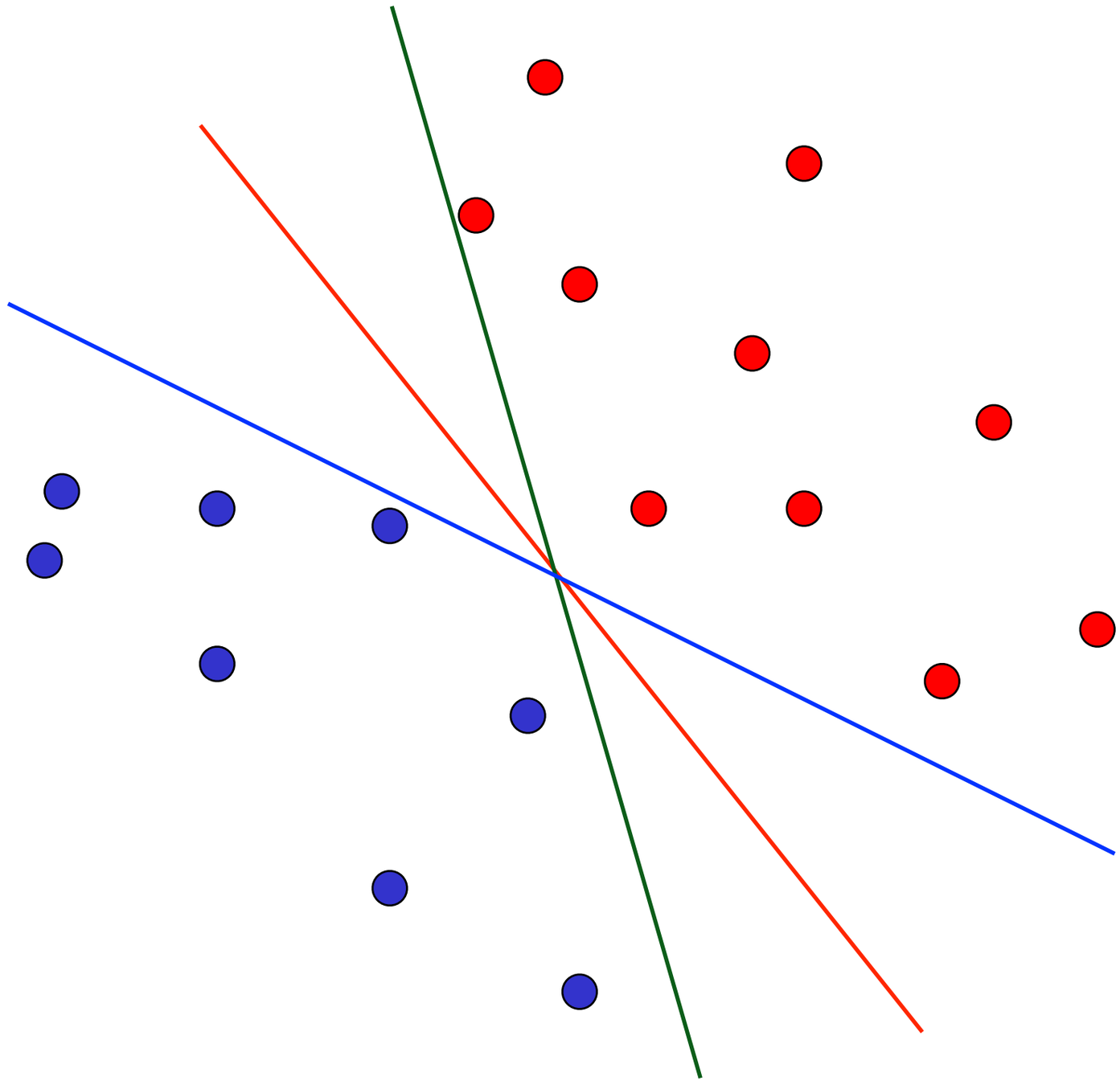
- Assume, we have D features and N points
- Overall time via matrix inversion
  - The closed form solution involves computing:

$$\boldsymbol{w} = \left(\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I}_D\right)^{-1} \mathbf{X}^\top \boldsymbol{\gamma}$$

  - Total time is $O(D^2N + D^3 + DN)$, assuming $O(D^3)$ matrix inversion
  - If N > D, then total time is $O(D^2N)$
- Overall time via gradient descent
  - Gradient: $\frac{d\mathcal{L}}{d\mathbf{w}} = \sum_n -2(y_n - \mathbf{w}^T\mathbf{x}_n)\mathbf{x}_n + \lambda\mathbf{w}$
  - Each iteration: $O(ND)$; T iterations: $O(TND)$
- Which one is faster?
  - Small problems D < 100: probably faster to run matrix inversion
  - Large problems D > 10,000: probably faster to run gradient descent

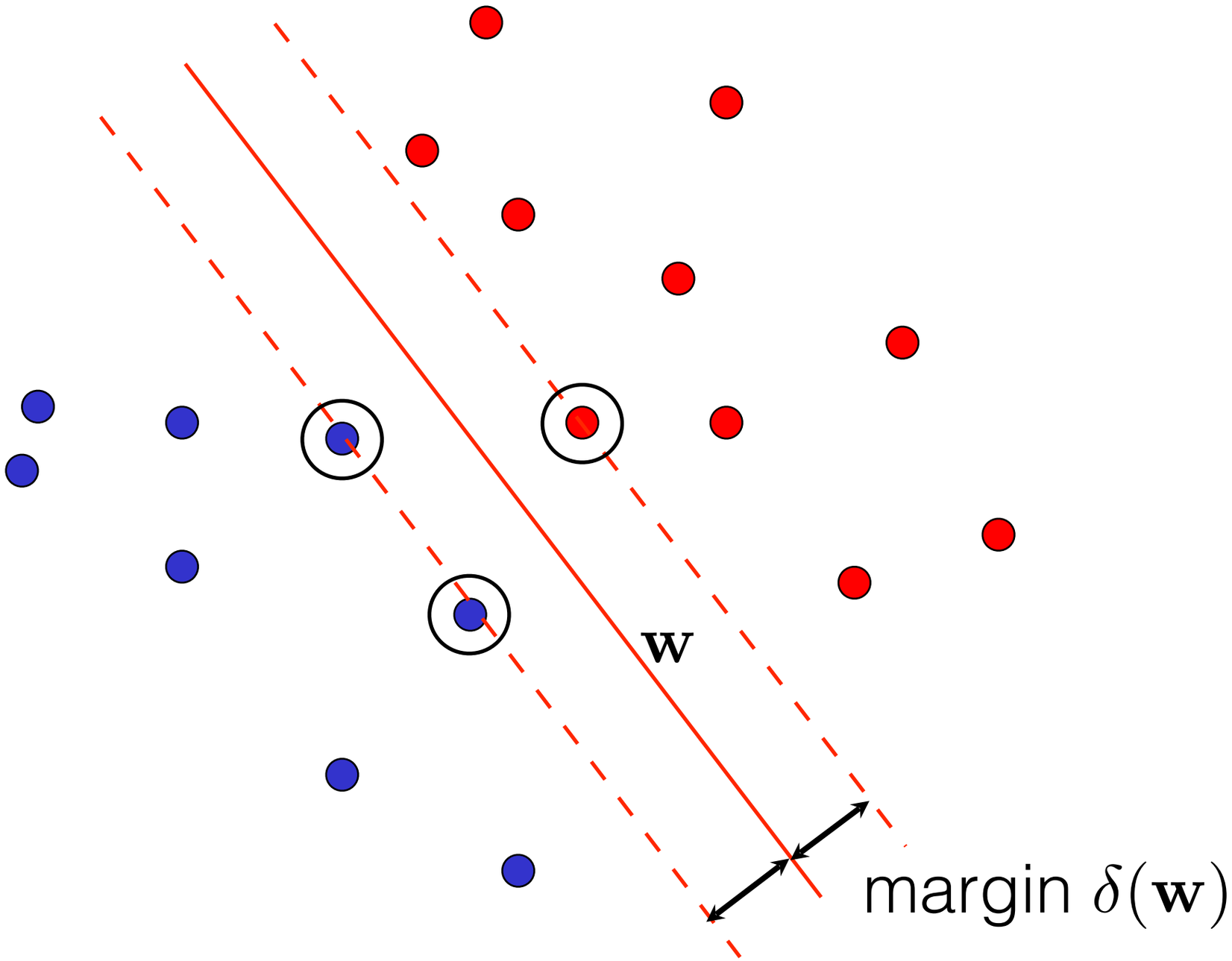
Picking a good hyperplane

- Which hyperplane is the best?



Support Vector Machines (SVMs)

- Maximize the distance to the nearest point (margin), while correctly classifying all the points



Optimization for SVMs

Separable case: hard margin SVM

$$\min_{\mathbf{w}} \frac{1}{\delta(\mathbf{w})}$$

maximize margin

$$\text{subject to: } y_n \mathbf{w}^T \mathbf{x}_n \geq 1, \forall n$$

separate by a non-trivial margin

Non-separable case: soft margin SVM

$$\min_{\mathbf{w}} \frac{1}{\delta(\mathbf{w})} + C \sum_n \xi_n$$

maximize margin minimize slack

$$\text{subject to: } y_n \mathbf{w}^T \mathbf{x}_n \geq 1 - \xi_n, \forall n$$

allow some slack

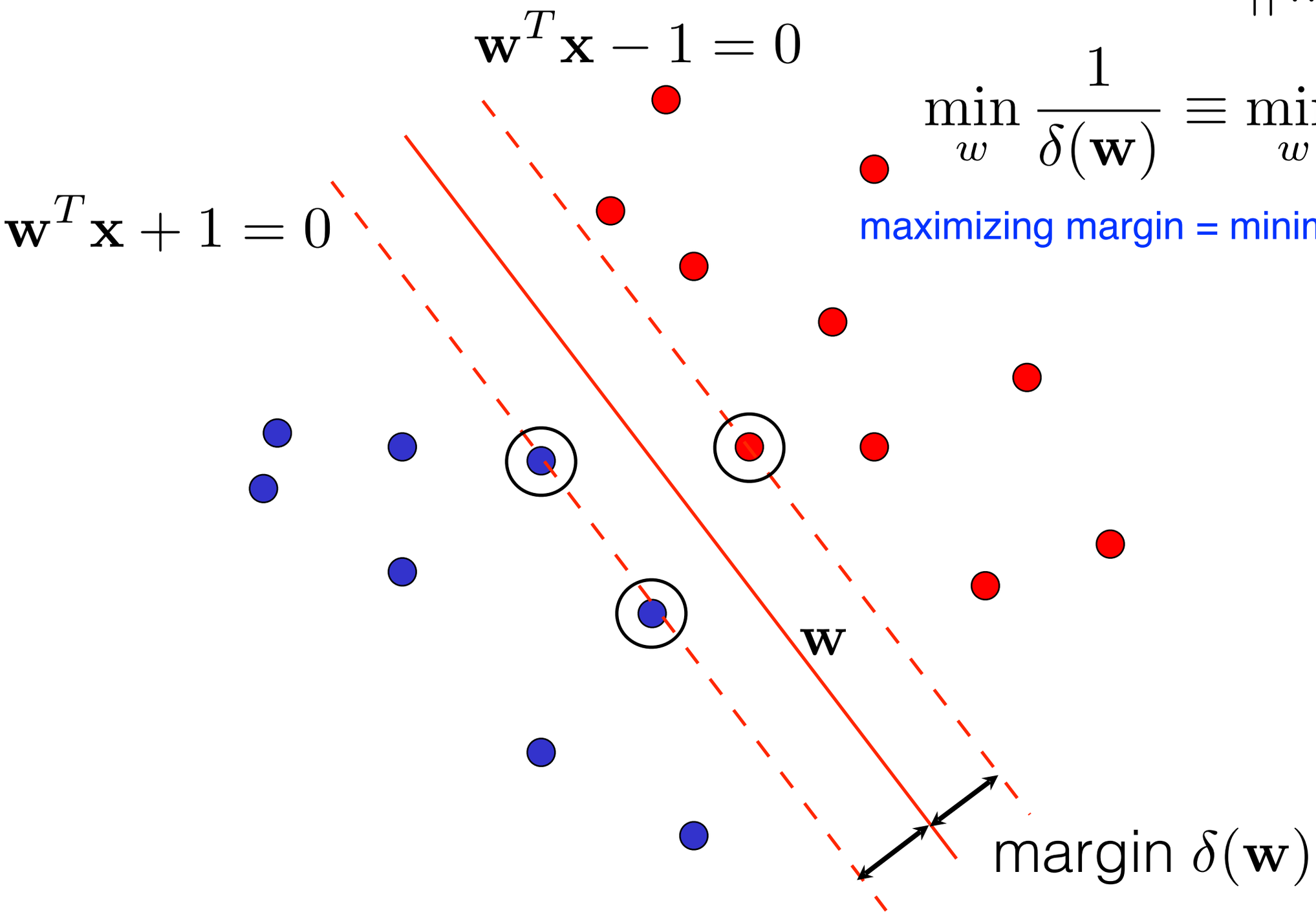
$$\xi_n \geq 0$$

Margin of a classifier

$$\delta(\mathbf{w}) = \frac{1}{||\mathbf{w}||}$$

$$\min_w \frac{1}{\delta(\mathbf{w})} \equiv \min_w ||\mathbf{w}||$$

maximizing margin = minimizing norm



Equivalent optimization for SVMs

Separable case: hard margin SVM

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2$$

maximize margin

squaring and half for convenience

$$\text{subject to: } y_n \mathbf{w}^T \mathbf{x}_n \geq 1, \forall n$$

separate by a non-trivial margin

Non-separable case: soft margin SVM

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C \sum_n \xi_n$$

maximize margin minimize slack

$$\text{subject to: } y_n \mathbf{w}^T \mathbf{x}_n \geq 1 - \xi_n, \forall n$$

allow some slack

$$\xi_n \geq 0$$

Slack variables

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_n \xi_n$$
$$\text{subject to: } y_n \mathbf{w}^T \mathbf{x}_n \geq 1 - \xi_n, \forall n$$
$$\xi_n \geq 0$$

soft margin SVM

- Suppose I tell you what **w** is, but forgot to give you the slack variables
- Can you derive the optimal slack for the nth example?
  - $y_n \mathbf{w}^T \mathbf{x}_n = 0.8$, $\xi_n = ?$ 0.2
  - $y_n \mathbf{w}^T \mathbf{x}_n = -1$, $\xi_n = ?$ 2.0
  - $y_n \mathbf{w}^T \mathbf{x}_n = 2.5$, $\xi_n = ?$ 0

$$\xi_n = \begin{cases} 0 & y_n \mathbf{w}^T \mathbf{x}_n \geq 1 \\ 1 - y_n \mathbf{w}^T \mathbf{x}_n & \text{otherwise} \end{cases}$$

$$\min_{\mathbf{w}} \frac{1}{2}||\mathbf{w}||^2 + C\sum_n \max(0, 1 - y_n \mathbf{w}^T \mathbf{x}_n)$$

Same as hinge loss with squared norm regularization!

Optimization for linear models

- Under suitable conditions*, provided you pick the step sizes appropriately, the convergence rate of gradient descent is O(1/N)
  - i.e., if you want a solution within 0.0001 of the optimal you have to run the gradient descent for N=1000 iterations.
- For linear models (hinge/logistic/exponential loss) and squared-norm regularization there are off-the-shelf solvers that are fast in practice: SVM^{perf} , LIBLINEAR, PEGASOS
  - SVM^{perf} , LIBLINEAR use a different optimization method

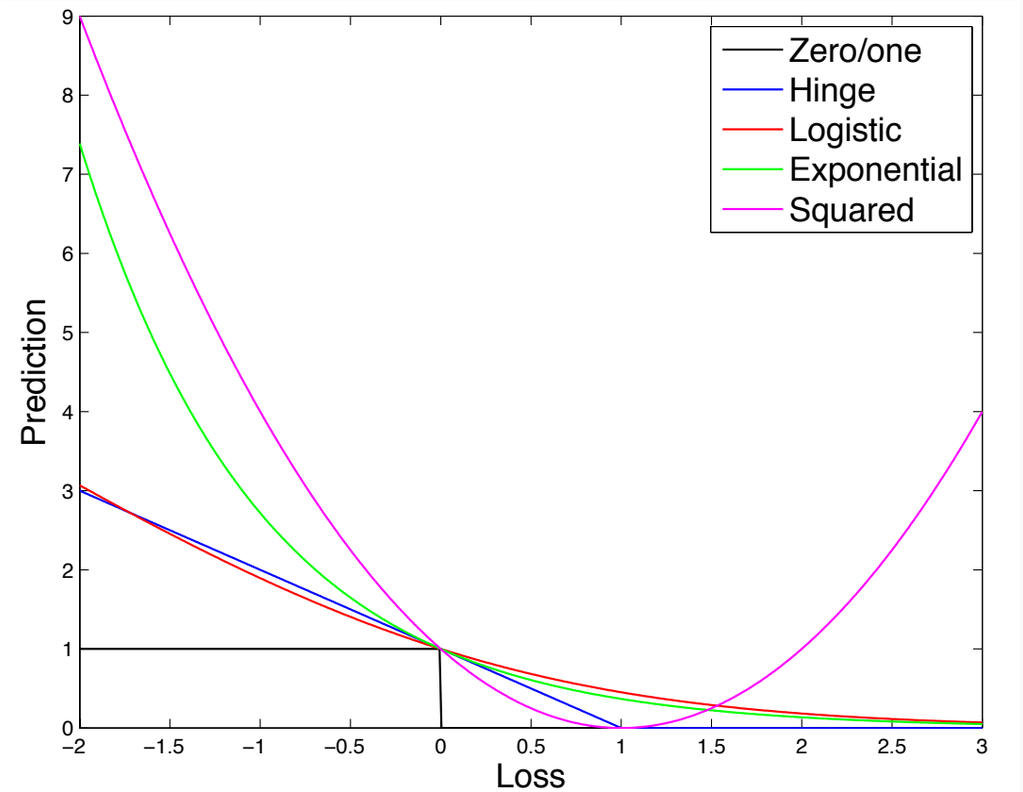
* the function is strongly convex: $f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{m}{2} \|y - x\|_2^2$

Slides credit

- Figures of various “p-norms” are from Wikipedia
  - http://en.wikipedia.org/wiki/Lp_space
- Some of the slides are based on CIML book by Hal Daume III

Appendix: code for surrogateLoss

Output →



Matlab code ←

```
% Code to plot various loss functions
y1=1;
y2=linspace(-2,3,500);
zeroOneLoss = y1*y2 <=0;
hingeLoss = max(0, 1-y1*y2);
logisticLoss = log(1+exp(-y1*y2))/log(2);
expLoss = exp(-y1*y2);
squaredLoss = (y1-y2).^2;

% Plot them
figure(1); clf; hold on;
plot(y2, zeroOneLoss,'k-','LineWidth',1);
plot(y2, hingeLoss,'b-','LineWidth',1);
plot(y2, logisticLoss,'r-','LineWidth',1);
plot(y2, expLoss,'g-','LineWidth',1);
plot(y2, squaredLoss,'m-','LineWidth',1);
ylabel('Prediction','FontSize',16);
xlabel('Loss','FontSize',16);
legend({'Zero/one', 'Hinge', 'Logistic', 'Exponential', 'Squared'}, 'Location',
'NorthEast', 'FontSize',16);
box on;
```