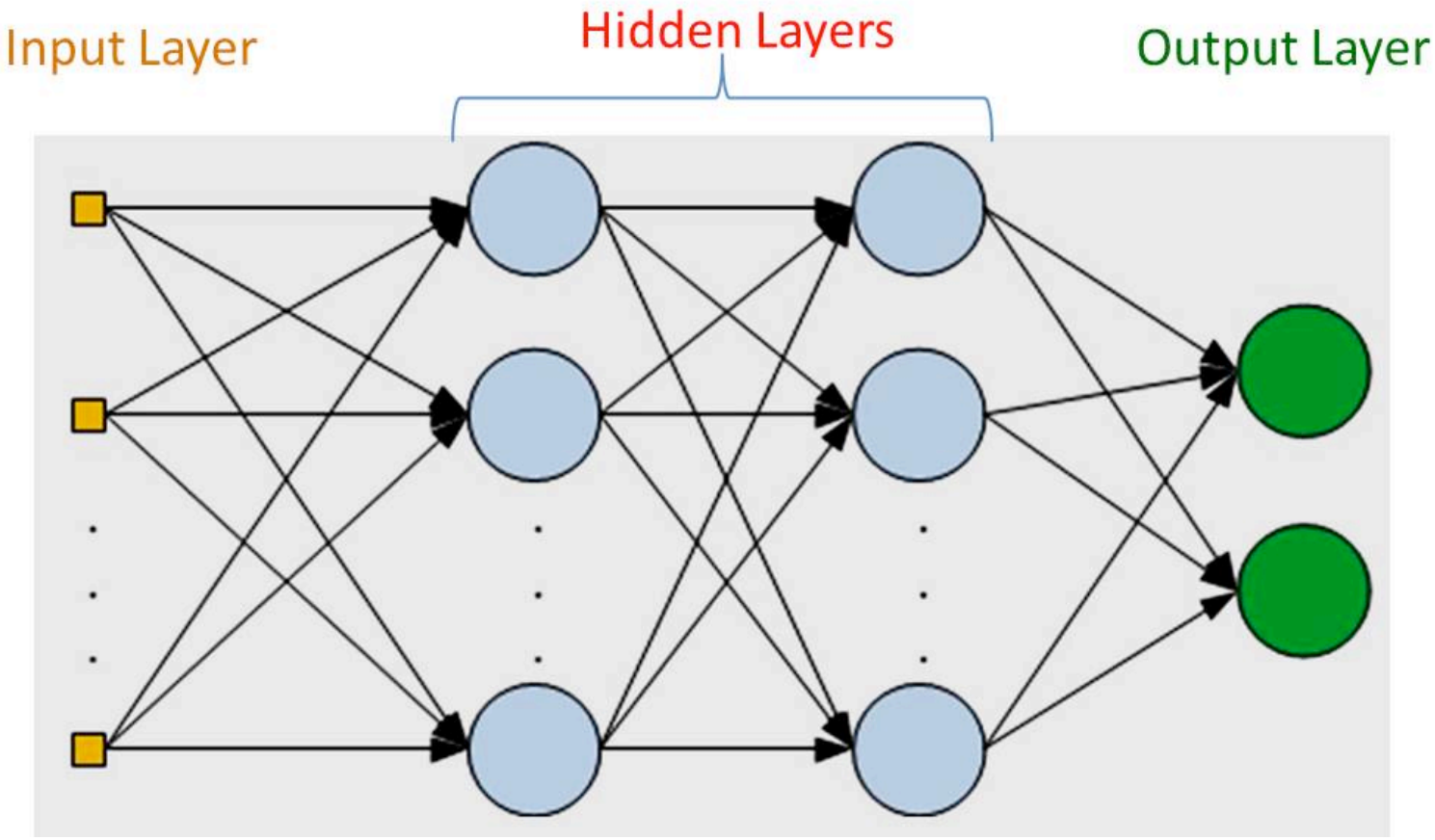
Principles of ML

Key to the Success.

Review

Neural Network: Abstract View

Multi Layer Perceptron (MLP)



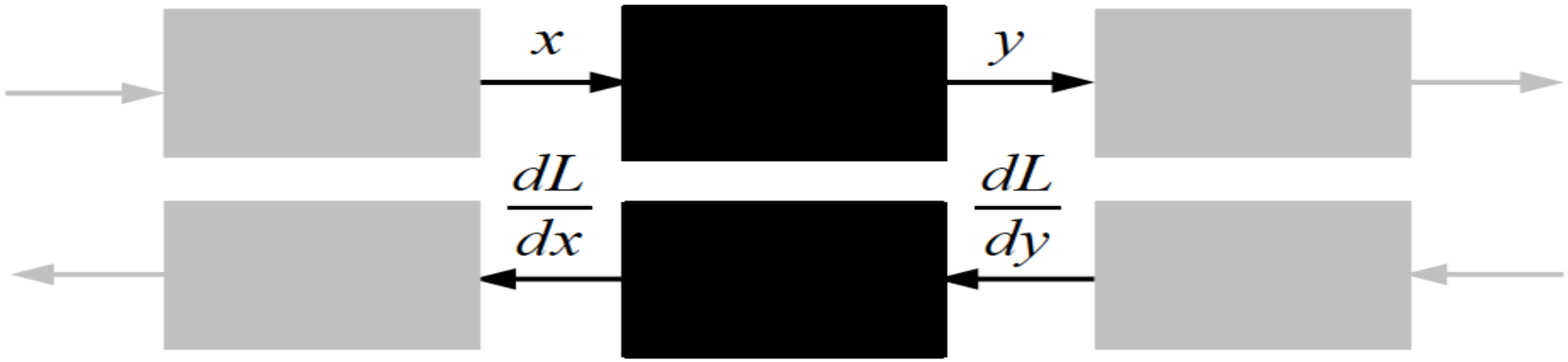
Two Computational Blocks/Steps

$$\mathbf{y} = \mathbf{W}\mathbf{x}$$

$$\mathbf{z} = \phi(\alpha) = \frac{1}{1 + e^{-\alpha}}$$

Sigmoid Nonlinearity

Chain Rule



We know . $\frac{dy}{dx}$ and $\frac{dy}{d\theta}$

Backpropagation

$$\frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx} \quad \textbf{(1)}$$

$$\frac{dL}{d\theta} = \frac{dL}{dy} \cdot \frac{dy}{d\theta} \quad \textbf{(2)}$$

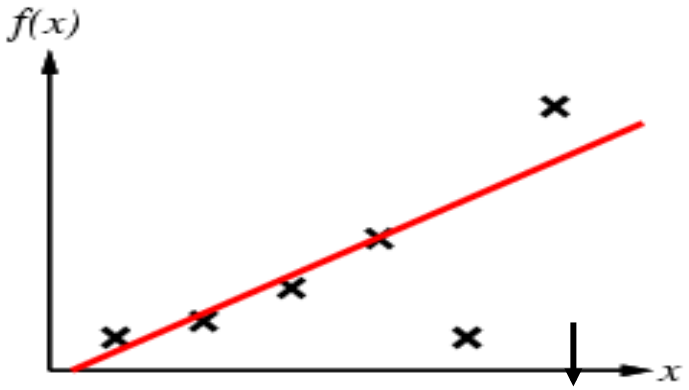
$$\theta^{n+1} = \theta^n - \eta \frac{dL}{d\theta} \quad \textbf{(3)}$$

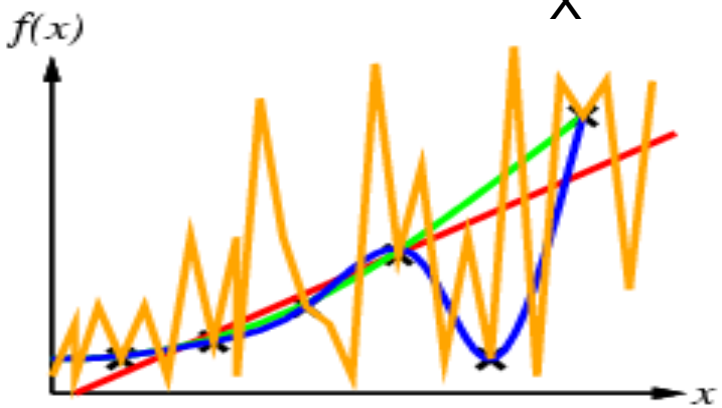
Let there be N stages. For a computational block l,

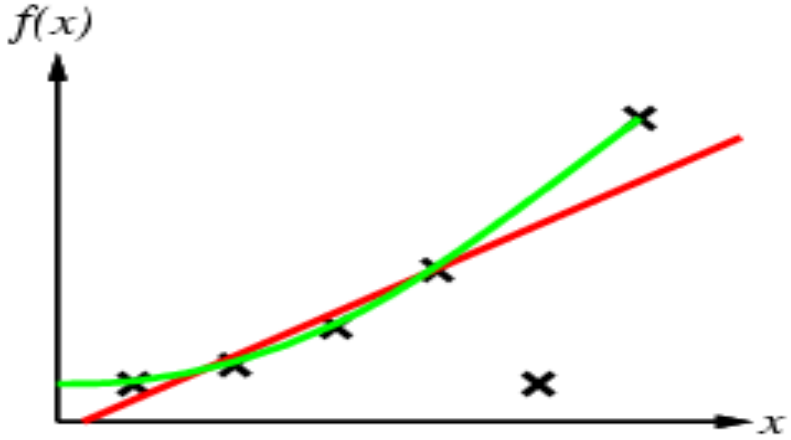
1. Compute $\frac{dL}{dx}$ using equation 1.
2. If the block as a learnable parameter θ, then
   - Compute $\frac{dL}{d\theta}$ using equation 2.
   - Update the parameters using equation 3.
3. Set the $\frac{dL}{dx}$ of stage l as $\frac{dL}{dy}$ of stage $l-1$, and repeat the steps 1-3, until we reach the first block.

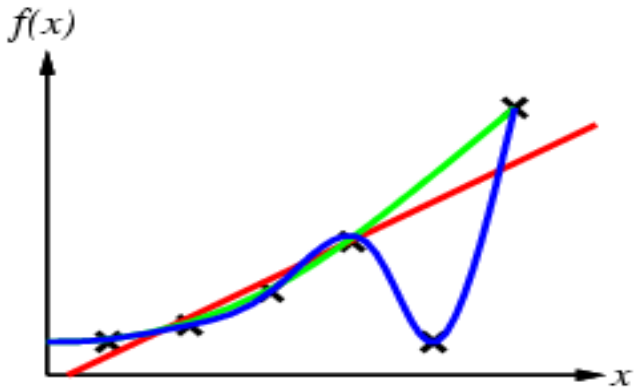
Questions?

What is the best model?





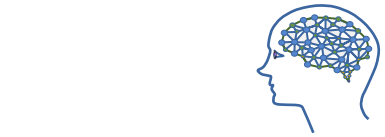


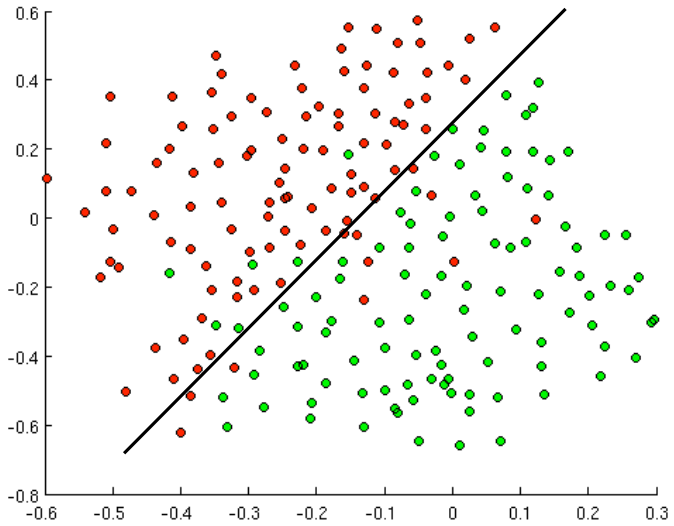


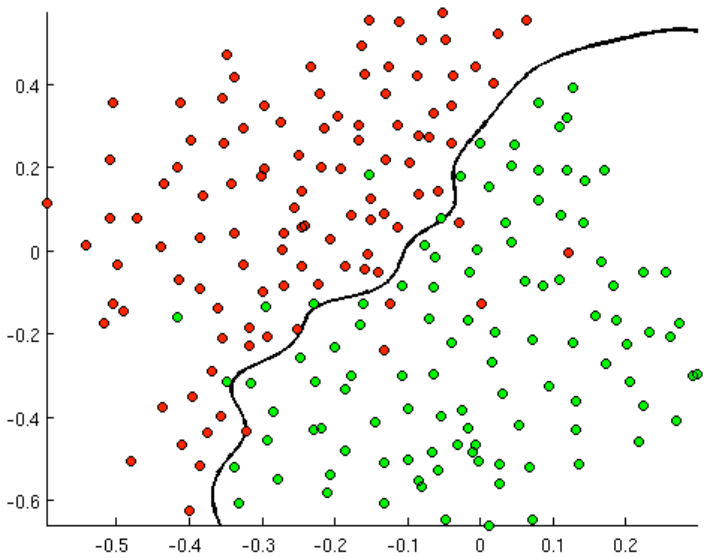
Given six "training" (x_i, y_i) pairs, find the y corresponding to the new "test" x*

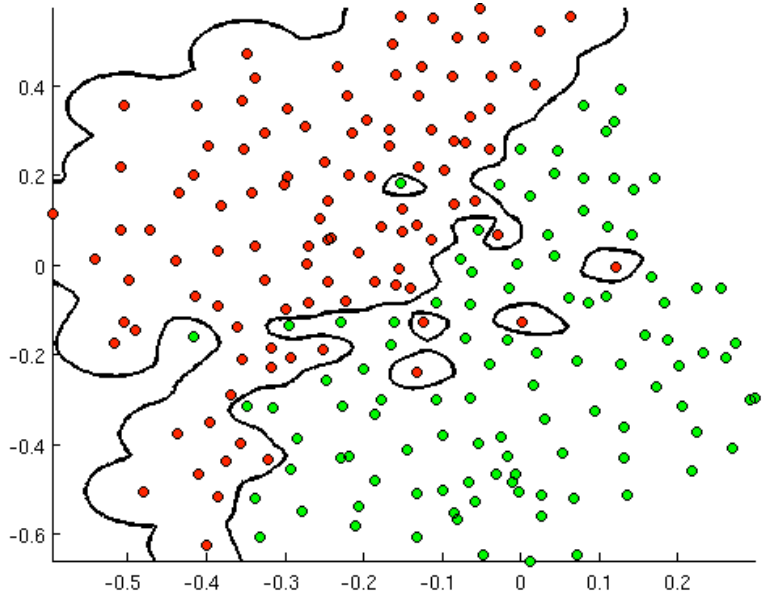
Which curve is the best?

Which is the best classifier?









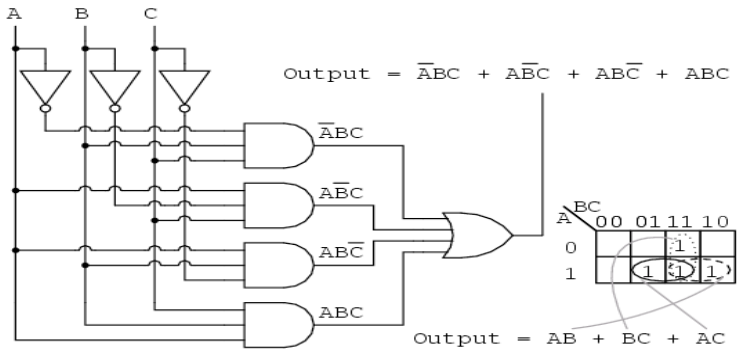
What happens when we simply learn?

- We often minimize an error over the training examples.
- The discrepancy between the actual and predicted. Let x_i is the input, z_i is the true output and y_i is the predicted output. f() is the model, say a Neural Network.
  - Eg.

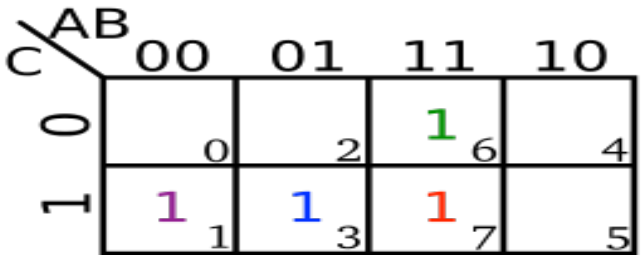
$$\sum_i (z_i - y_i)^2 = \sum_i (z_i - f(x_i))^2$$

- If we want only to minimize this, we can even get zero error. The perfect error.
- But we do not want that.**Then what do we want?**

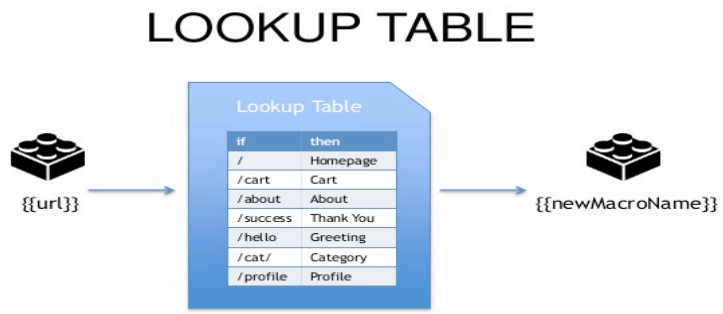
We know about f() that fits samples.

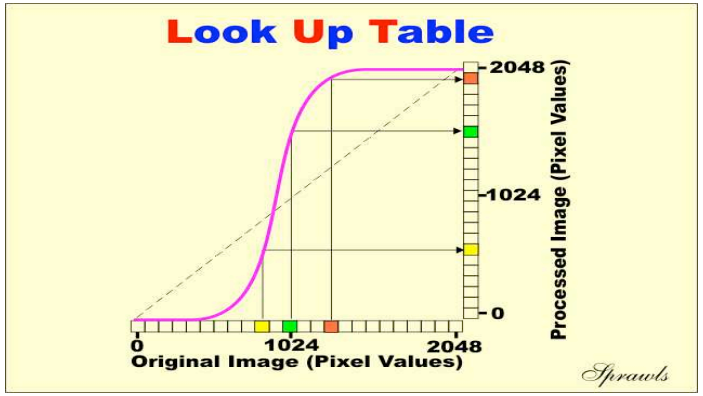


$$f(a,b,c) = \underset{111}{ABC} + \underset{110}{AB\overline{C}} + \underset{011}{\overline{A}BC} + \underset{001}{\overline{A}\overline{B}C}$$



Is K-Map Learning?





Is LUT Learning?

Generalization

Learning is concerned with accurate prediction of future data, *not* accurate prediction of training or available data.

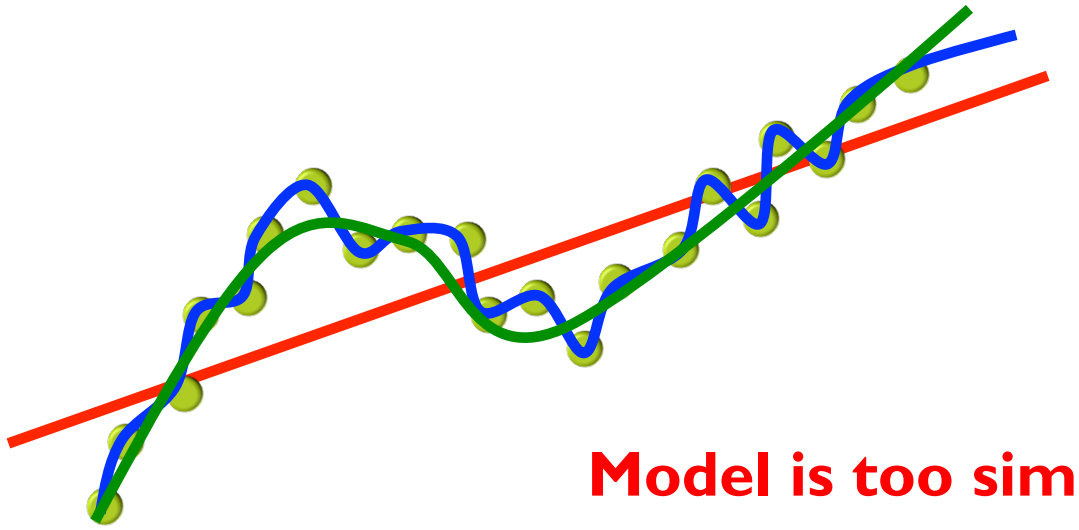
We only have “training data”. Then how do we minimize error on unseen/future data?

Occam's Razor

Select the simplest hypothesis (solution) that suits the data.

Eg. Minimize Sum of "fit error" and "degree of the polynomial"

Model the Signal; Not Noise

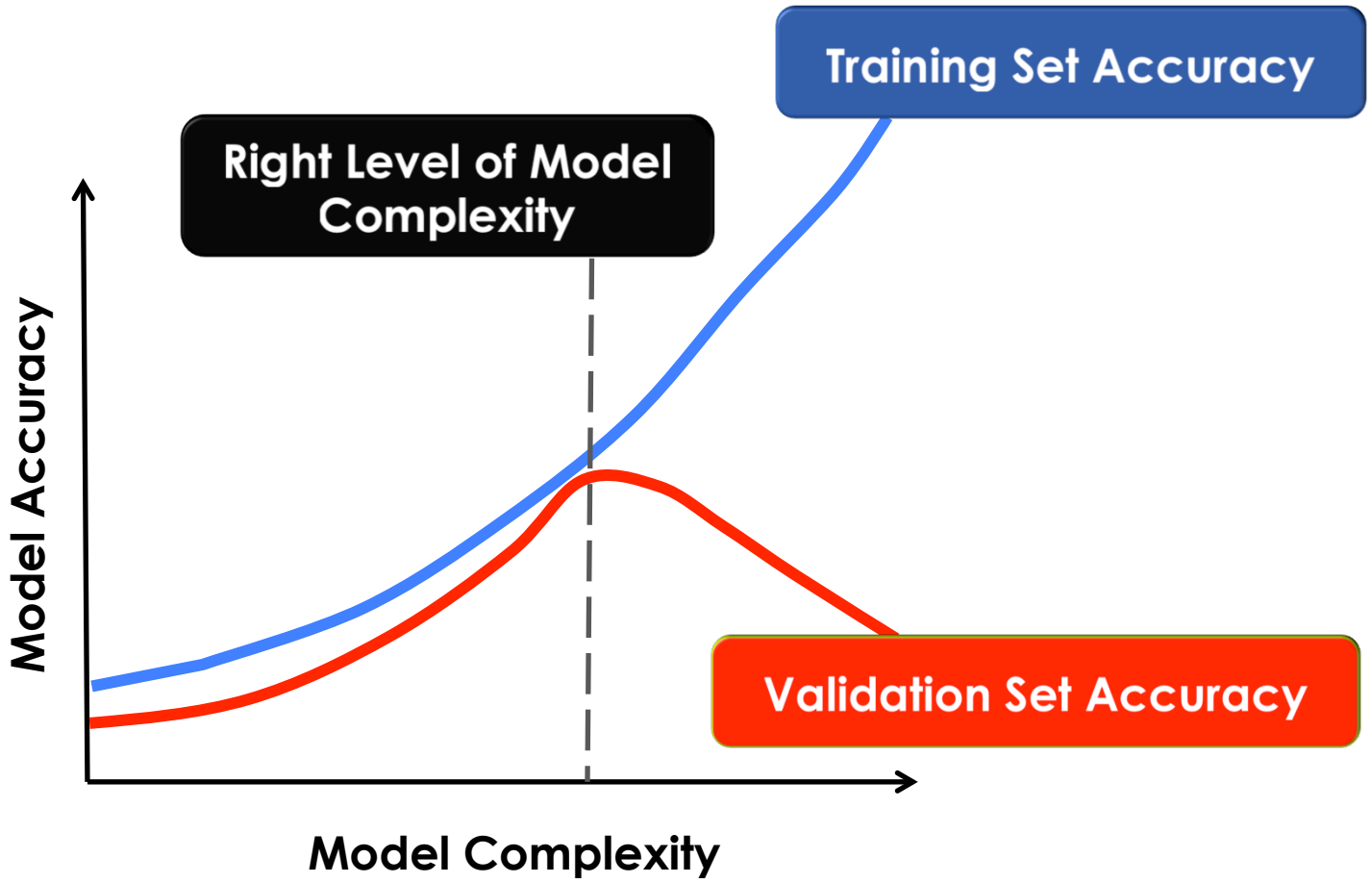


Model is too simple → UNDER LEARN

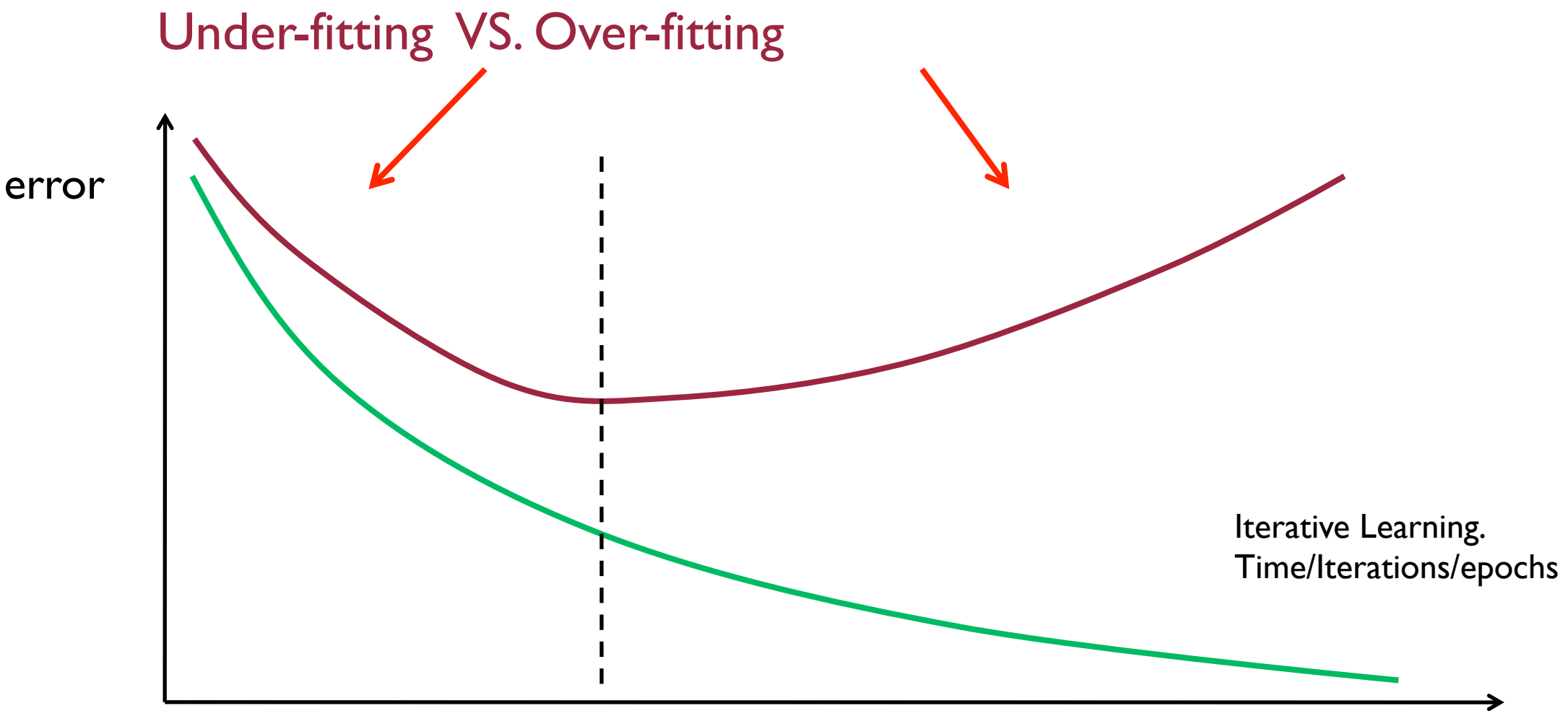
Model is too complex → MEMORIZE

Model is just right → GENERALIZE

Generalize; Don't Memorize!



What we may see?



Questions?

Next: Closer look at Overfitting

Overfitting

- "Overfitting is a modeling error which occurs when a function is too closely fit to a limited set of data points. Overfitting the model generally takes the form of making an overly complex model to explain idiosyncrasies in the data under study. In reality, the data often studied has some degree of error or random noise within it. Thus attempting to make the model conform too closely to slightly inaccurate data can infect the model with substantial errors and reduce its predictive power."

Your favorite question ☺: What is the "best" parameter for my training data?

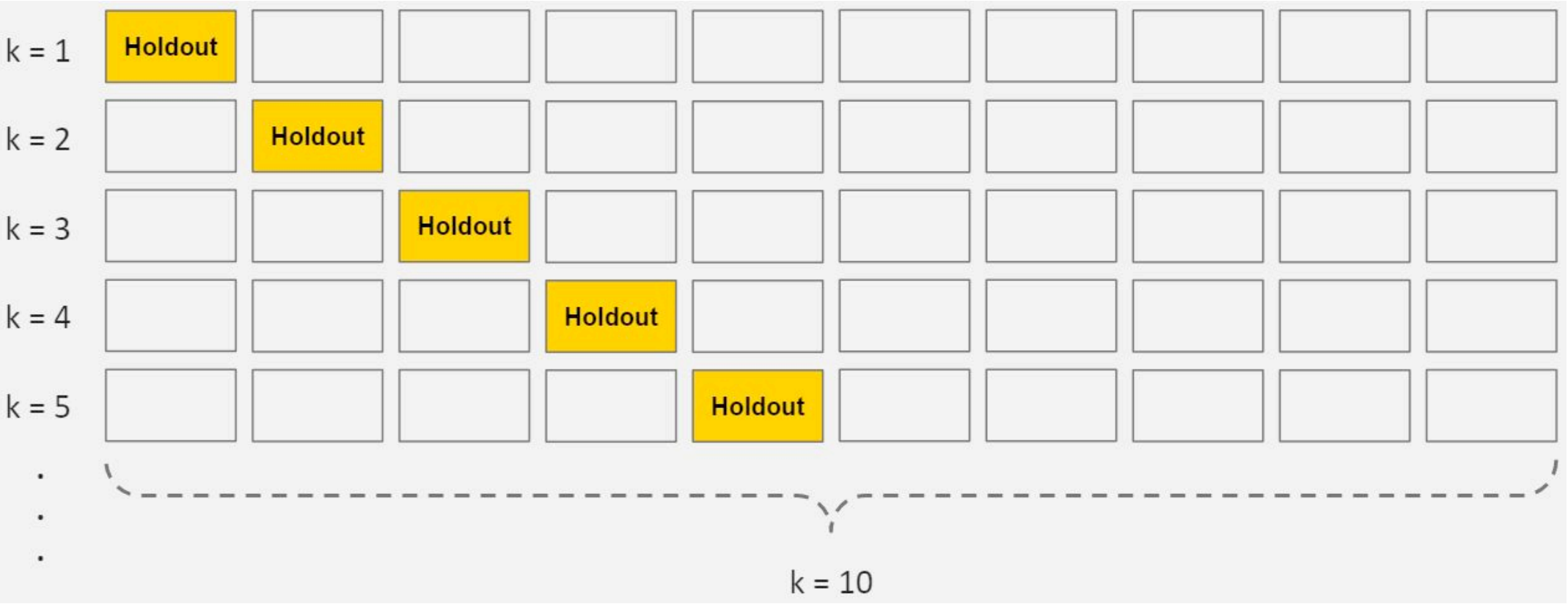
Were you asking for recipe for overfitting? (you know the answer now ☺)

Questions?

How to prevent/reduce overfitting?

- Cross Validation
- **Train with more data**
- **Reduce/Remove features**
- Early Stopping
- Regularization
- **Ensembling (may be in a later lecture)**

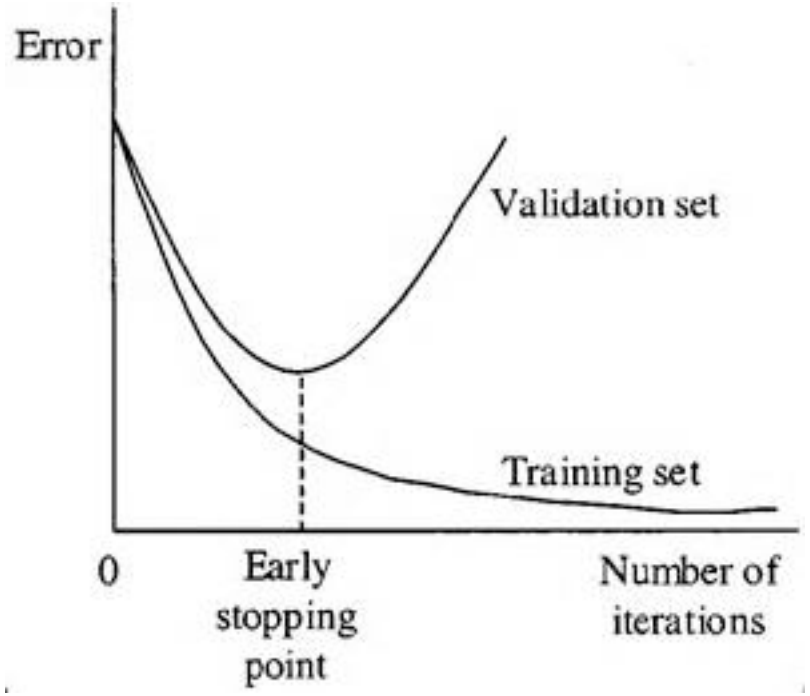
Cross validation



Crossvalidation and LOO

1. Split the training data into k subsets. Use k-1 for training and one for evaluation. Final error is the average of all the k sets.
2. Split the data into x and 100-x percentages for train and evaluation. Repeat the task many times and average the error.
3. If number of samples (N) are small, use N-1 for training and one for testing. Repeat N times and average. Leave One Out (LOO)

Early Stopping



Regularization: **regularization** is a process of introducing additional information in order to solve an ill-posed problem or to prevent overfitting.

What to Minimize? (regularization)

- We were minimizing:

$$\mathcal{E} = \sum_i (z_i - y_i)^2 = \sum_i (z_i - f(x_i))^2$$

- We can minimize
  - Fit error + some measure of complexity of the model

$\mathcal{E} +$ Sum of Weights

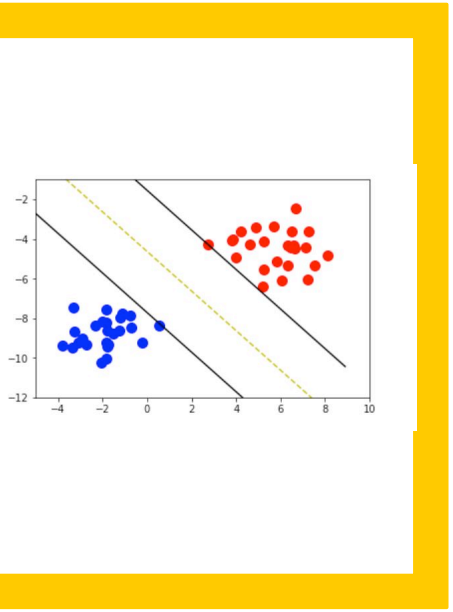
$\mathcal{E} +$ Number of Non-zero Weights

Train, Validation, Test Data Sets

- Train: data used for training
  - Find learnable parameters of the model
- Validation: "internal" test data
  - Estimate the error on one or more subsets (eg. Crossvalidation)
  - Vary hyper-parameters (eg. Architecture of a Neural Network) and retrain
- Test
  - Evaluate the final (best) model on test data.

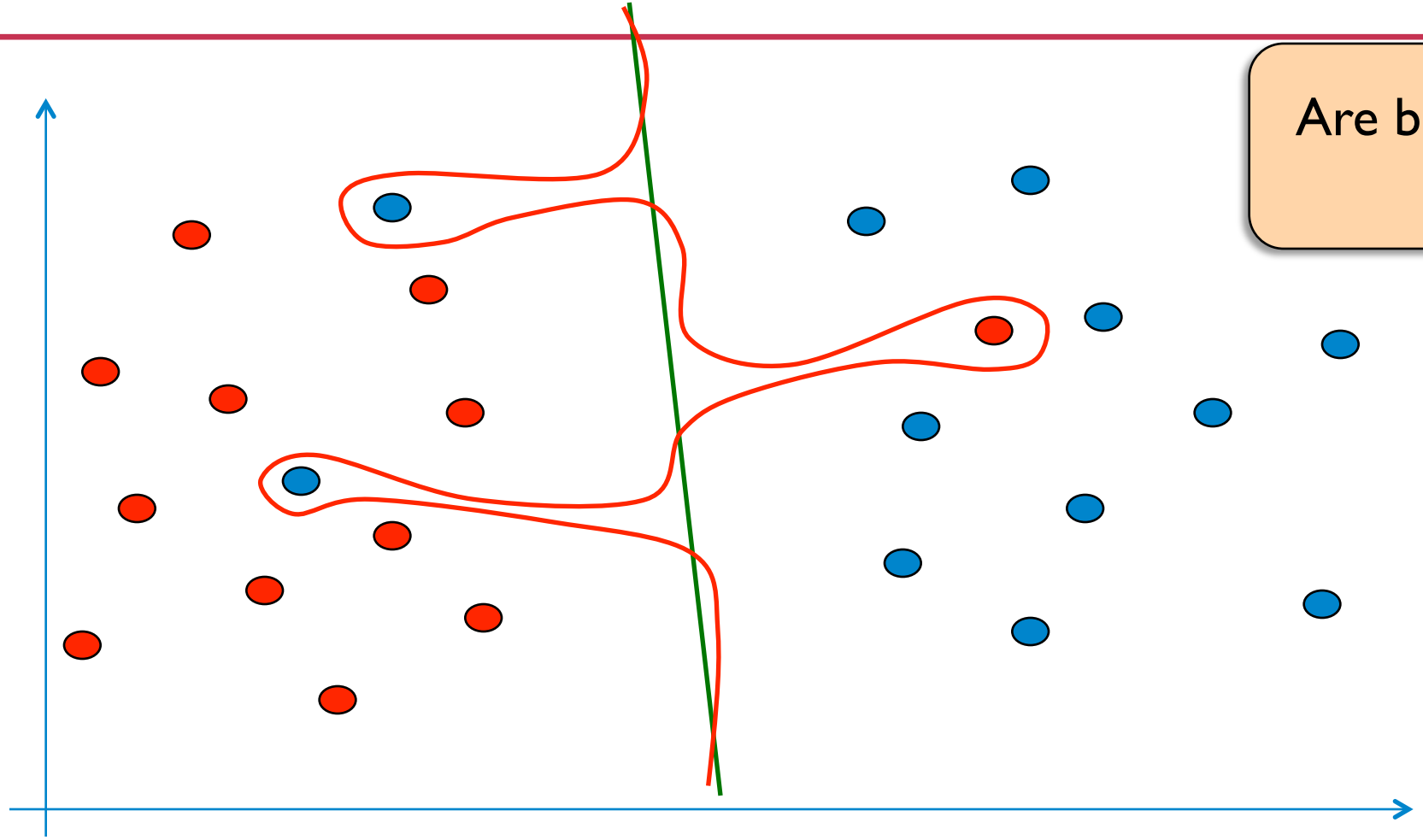
Summary, Questions?

- Find model that is simple and fitting the data.
- Problem of overfitting.
  - How to detect? What to observe during learning? How to avoid.
- Regularize the solution by adding extra term that also minimize the “complexity”.
- Right way to design solution with
  - Train, Val and Test splits.



Support Vector Machines

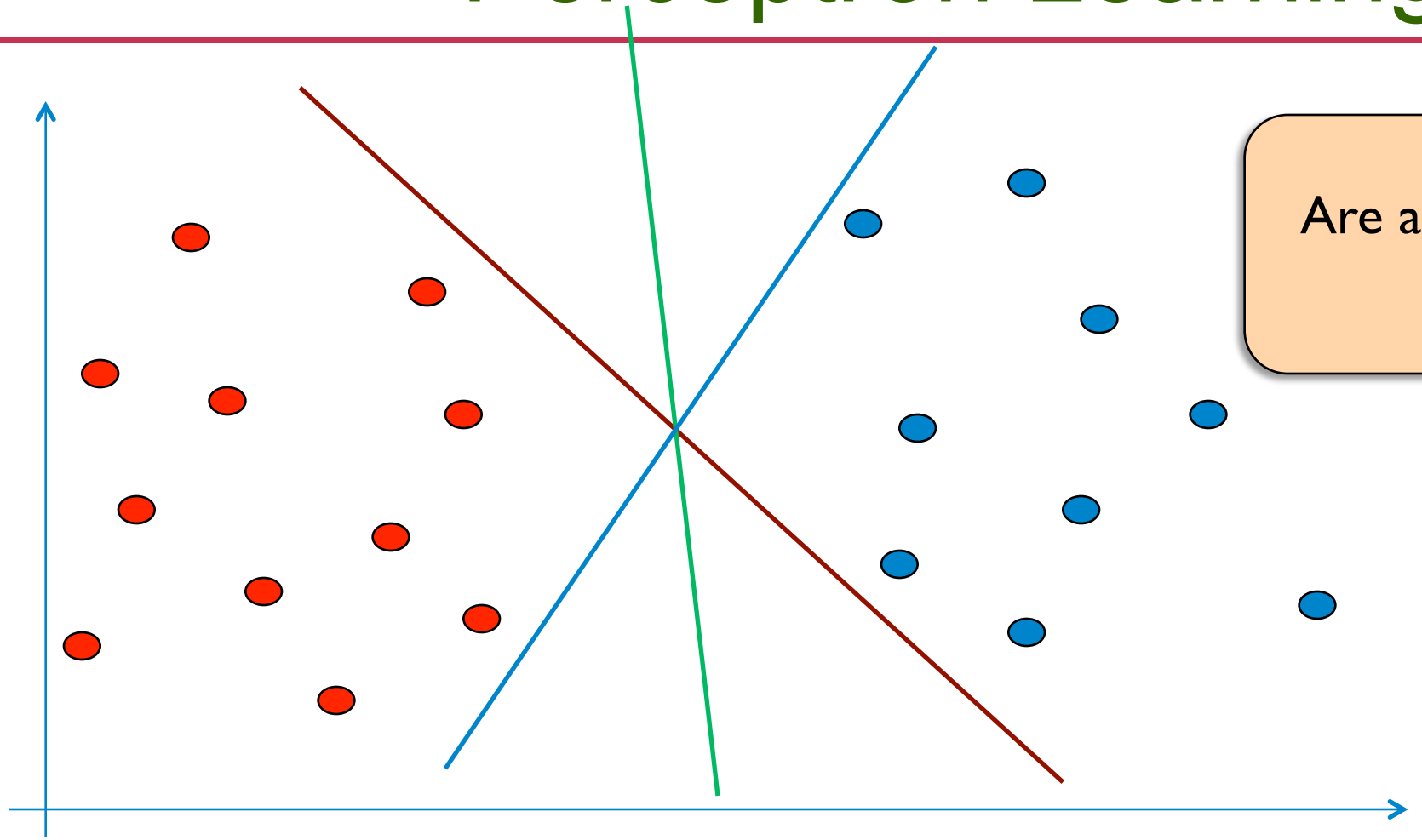
Generalization and Model Complexity



Are both solutions equally good?

- Is it good to use a complex curve to reduce training error?

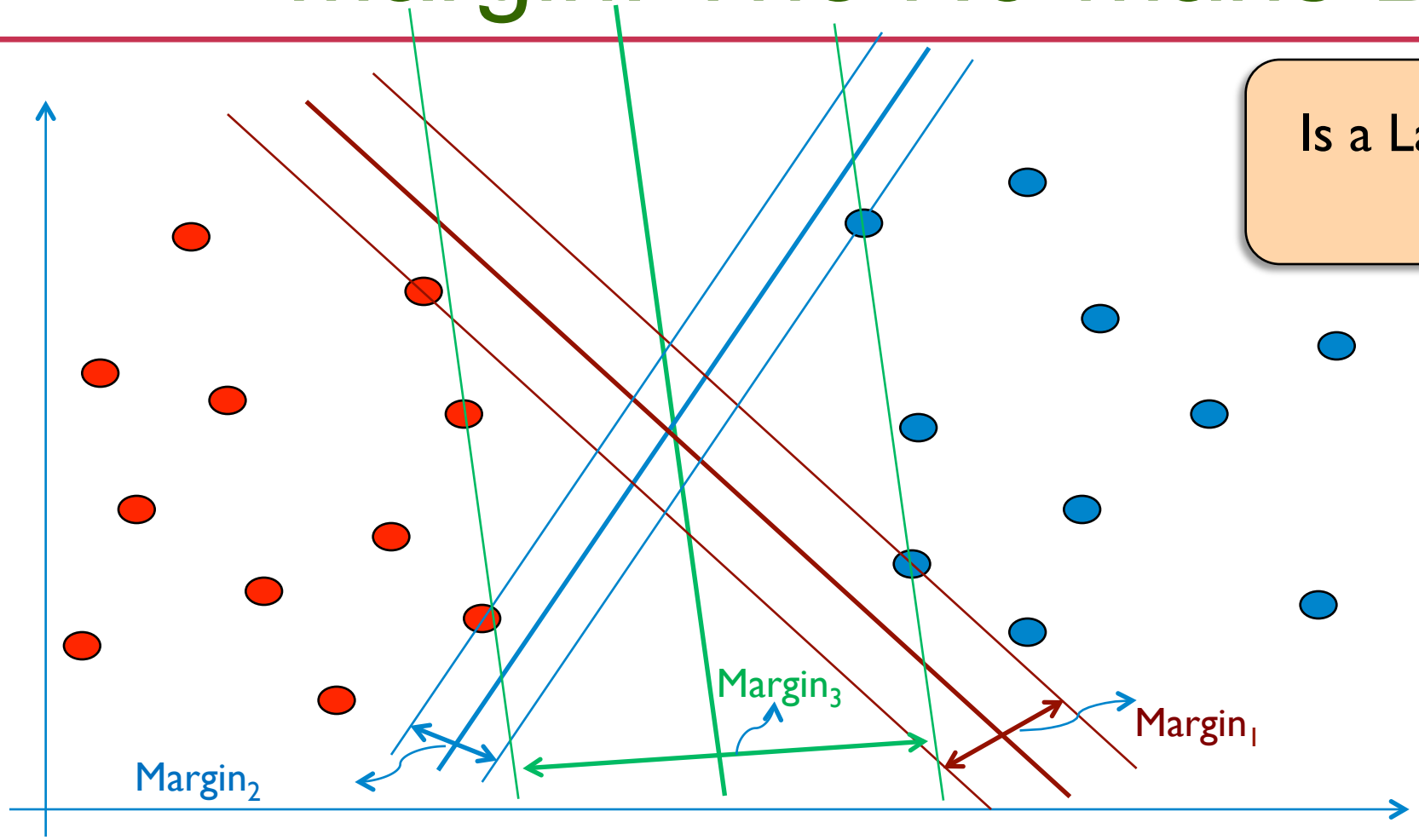
Perceptron Learning



Are all solutions equally good?

- Multiple solutions exist for linearly separable data
- Perceptron learning (any GD) results in a feasible solution

Margin: The No-mans Band

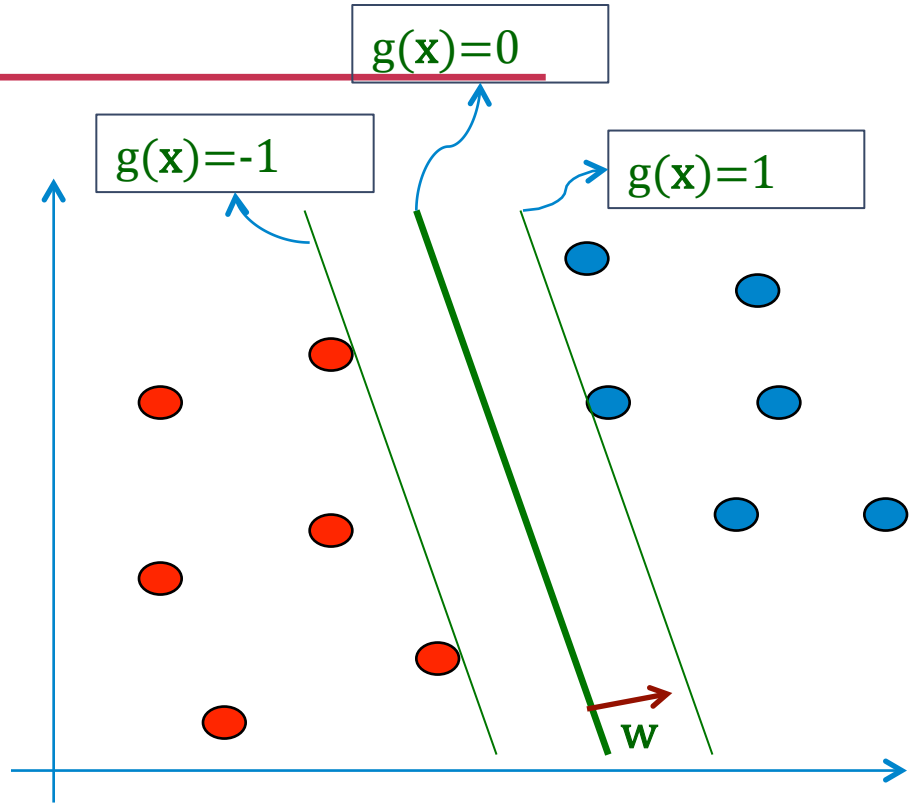


Is a Larger Margin better? Why?

- Margin: Width of a band around decision boundary without any training samples
- Margin varies with the position and orientation of the separating hyperplane

SVM: Formulation

- Let $g(x)=\mathbf{w}^T\mathbf{x}+b$.
- We want to maximize margin such that:
  - $\mathbf{w}^T\mathbf{x}_i+b \geq 1$ for $y_i=1$
  - $\mathbf{w}^T\mathbf{x}_i+b \leq -1$ for $y_i=-1$
  - $y_i(\mathbf{w}^T\mathbf{x}_i+b) \geq 1$ for all i



Eg. Distance from $(0,0)$ to $a \cdot x + b \cdot y + c = 0$ is $\frac{c}{\sqrt{a^2+b^2}}$.

Similarly distance from origin to $w^T x + b = 1$ is $\frac{b-1}{||w||}$. And to $w^T x + b = -1$ is $\frac{b+1}{||w||}$. The distance between the the two lines/planes (margin) is

$$\frac{b+1}{||w||} - \frac{b-1}{||w||} = \frac{2}{||w||}$$

The objective of maximize the margin is same as minimize $\frac{1}{2}||w||$, such that all positive samples and negative samples are beyond the respective half planes.

SVM: Primal and Dual

$$\min \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\textit{subject to} \quad y_i(\mathbf{w}^\mathbf{T}\mathbf{x_i} + b) - 1 \geq 0 \forall i$$

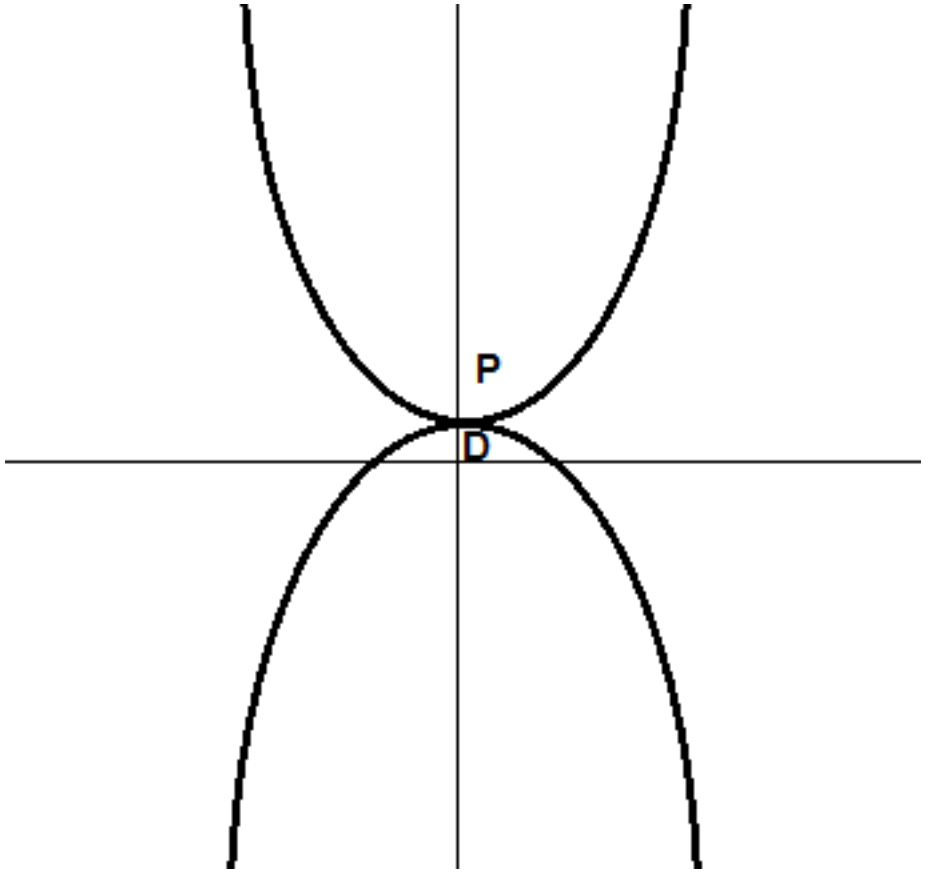
$$y_i \in \{-1, 1\}$$

This results in maximization of

$$J_d(\alpha) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i=1}^{N} \alpha_i\alpha_j y_i y_j \mathbf{x_i^T x_j}$$

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

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

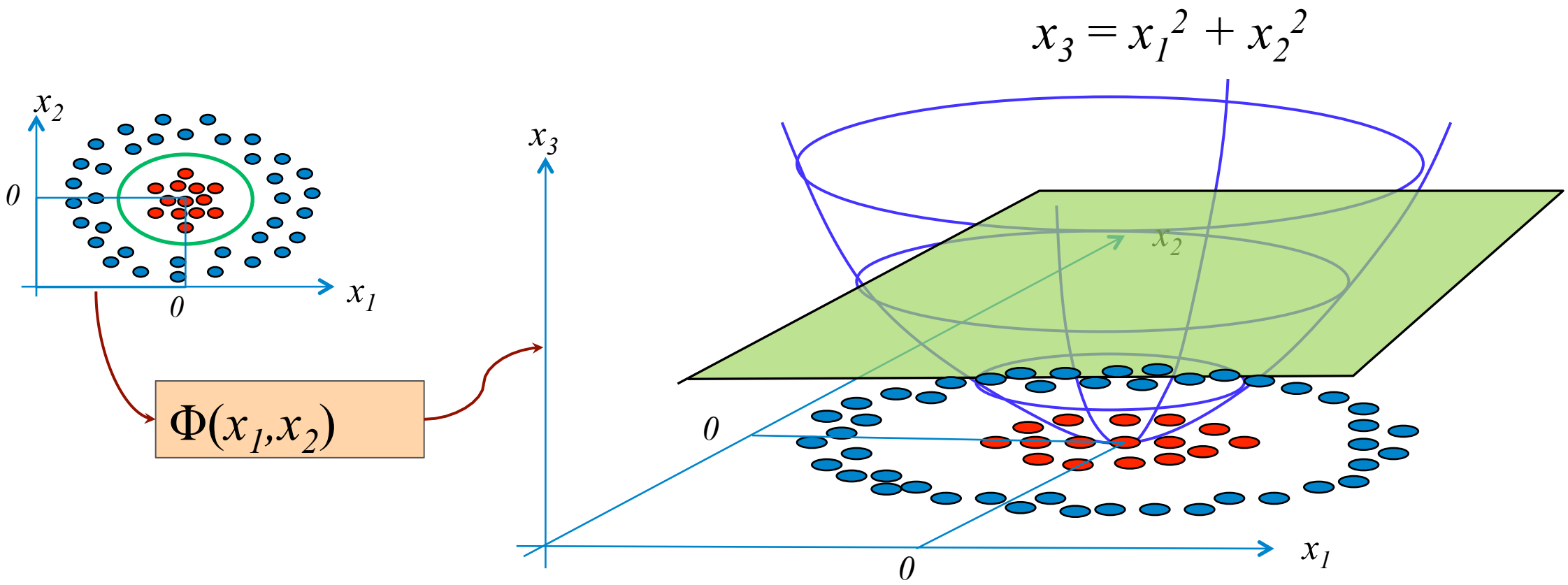


Kernels

Interestingly, the vectors appear as only dot product in the formulation. This allows us to solve the problem in a very high dimension (where the data set will well behave) without explicitly bothering about the mapping which converts into higher dimension.
We need only a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$

$$K(\mathbf{s}_i, \mathbf{x}_i) = \Phi(\mathbf{s}_i) \cdot \Phi(\mathbf{x}_i)$$

Kernels



Φ is a non-linear mapping into a possibly high-dimensional space

A Simple Quadratic Kernel

$$\text{Let } \Phi(\mathbf{X}) = \Phi\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1 x_1 \\ x_1 x_2 \\ x_2 x_1 \\ x_2 x_2 \end{bmatrix}$$

We can compute $K(X,Y)=(X.Y)^2$ instead of mapping with Φ explicitly and then computing dot product.

$$\text{Let } K(\mathbf{X},\mathbf{Y}) = \Phi(\mathbf{X}) \bullet \Phi(\mathbf{Y}) = \begin{bmatrix} x_1^2 \\ x_1 x_2 \\ x_2 x_1 \\ x_2^2 \end{bmatrix} \bullet \begin{bmatrix} y_1^2 \\ y_1 y_2 \\ y_2 y_1 \\ y_2^2 \end{bmatrix}$$

$$= x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2 = (x_1 y_1 + x_2 y_2)^2 = (\mathbf{X} \bullet \mathbf{Y})^2$$

Notes

- Linear SVMs are very good. Fast. Possibly one of the best linear classifier.
- Kernel SVMs (nonlinear SVMs) were the best until the popularity of Deep Learning.
- Popular implementations use Quadratic Programming for training. Gradient descent based solvers (pegasos) also exists.
- Limitation:
  - Primarily binary (two class). Multi class extensions exists.

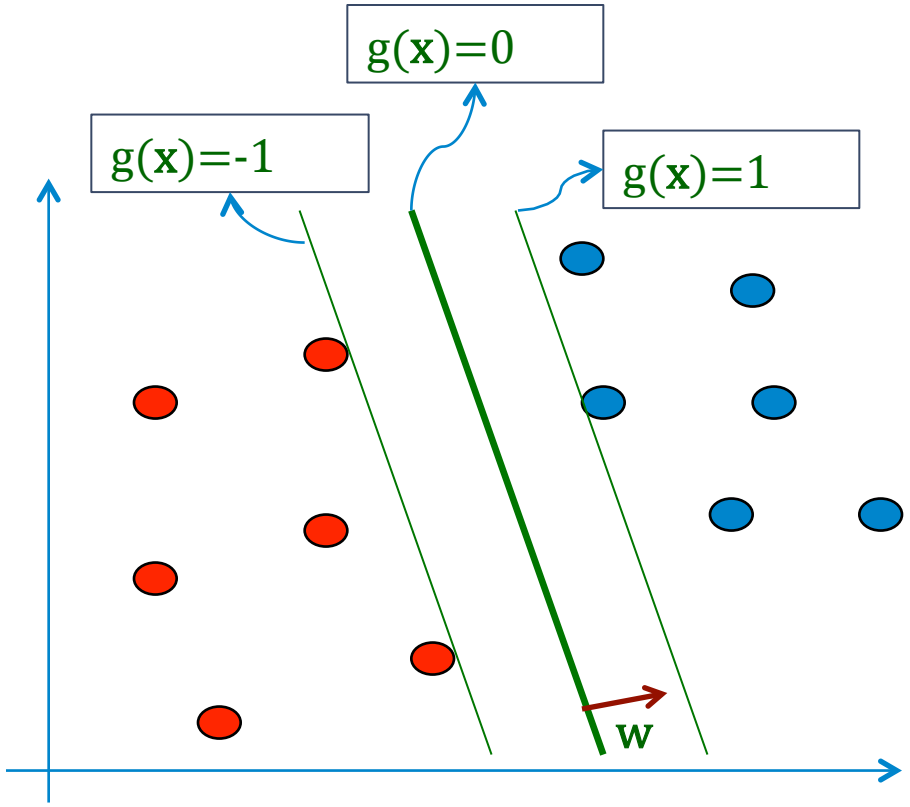
Summary

- SVMs are very good.
- Convex optimization. No worries about local minima.
- Many excellent solvers. (Often we never code ourselves.)
- Linear SVMs are efficient for training and testing (both memory and flops).
- Kernels (nonlinear) SVMs are accurate. Need not be efficient/compact.

Thanks. Questions?

Formulation

- Let $g(x)=\mathbf{w}^T\mathbf{x}+b$.
- We want to maximize k such that:
  - $\mathbf{w}^T\mathbf{x}_i+b \geq 1$ for $y_i=1$
  - $\mathbf{w}^T\mathbf{x}_i+b \leq -1$ for $y_i=-1$
- Value of $g(\mathbf{x})$ dependents on $\|\mathbf{w}\|$:
  1. Keep $\|\mathbf{w}\|=1$, and maximize $g(\mathbf{x})$, or
  2. Let $g(\mathbf{x})\geq 1$, and minimize $\|\mathbf{w}\|$.
- We use approach (2) and formulate the problem as:
  - Minmize: $\frac{1}{2}\mathbf{w}^T\mathbf{w}$
  - Subject to: $d_i(\mathbf{w}^T\mathbf{x}_i+b) \geq 1$, for $i=1..N$



Solving the Dual Form

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \mathbf{x}_i^T \mathbf{x}_j$$

$$\text{Subject to } \alpha_i \geq 0 \quad \forall_i \quad \text{and} \quad \sum_{i=1}^{N} \alpha_i d_i = 0$$

QP Solver

α_i

$$\mathbf{w}_o = \sum_{i=1}^{N} \alpha_i d_i \mathbf{x}_i$$

- The only unknowns (variables) are α_is.
- The constraints are also on α_is only.
- Data vectors appear only as dot products
- Objective is convex, subject to linear constraints
- Can be solved using standard convex quadratic program solvers

KKT Conditions

$$\alpha_i [d_i (\mathbf{w}_o^T \mathbf{x}_i + b_o) - 1] = 0$$

$$b_o = 1 - \mathbf{w}_o^T \mathbf{x}_{s+}$$

Solving the SVM with a Mapping

- Data vectors occur only as dot products in SVM-learning and testing

Testing Phase

- to

$$\text{Label} = sign(\mathbf{w}_o \bullet \Phi(\mathbf{x}_{test}) + b_o)$$

$$\mathbf{w}_o = \sum_{i=1}^{N} \alpha_i d_i \Phi(\mathbf{x}_i)$$

$$\therefore \text{L}$$

$$Q(\boldsymbol{\alpha}) = \sum_{i=1}^{N} \alpha_i - \tfrac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j d_i d_j \, \text{K}(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{Label} = sign\left(\sum_{i=1}^{N} (\alpha_i d_i \, \text{K}(\mathbf{x}_i, \mathbf{x}_{test})) + b_o \right)$$