Class is 170.

Announcements

Matlab Grader homework,
1 and 2 (of less than 9) homeworks Due 22 April **tonight,** Binary graded.
For HW1, please get word count <100
167, 165,164 has done the homework. (**If you have not done it talk to me/TA!**)
Homework 3 (released ~tomorrow) due **~5 May**

Jupiter "GPU" home work released Wednesday. Due 10 May

Projects: 27 Groups formed. Look at Piazza for help.
Guidelines is on Piazza
May 5 proposal due. TAs and Peter can approve.

Today:
- Stanford CNN 9, Kernel methods (Bishop 6),
- Linear models for classification, Backpropagation

Monday
- Stanford CNN 10, Kernel methods (Bishop 6), SVM,
- Play with Tensorflow playground before class http://playground.tensorflow.org

Projects

- **3-4** person groups preferred
- Deliverables: Poster & Report & main code (plus proposal, midterm slide)

- **Topics** your own or chose form suggested topics. Some **physics inspired**.

- **April 26 groups** due to TA (if you don't have a group, ask in piaza we can help). TAs will construct group after that.

- **May 5** proposal due. TAs and Peter can approve.
- Proposal: One page: Title, A large paragraph, data, weblinks, references.
- Something **physical**

DataSet

- 80 % preparation, 20 % ML
- Kaggle:
  https://inclass.kaggle.com/datasets
  https://www.kaggle.com
- UCI datasets: http://archive.ics.uci.edu/ml/index.php

- Past projects…

- Ocean acoustics data

In 2017 Many choose the source localization

- two CNN projects,

Many thanks for the fun projects! Below are the final projects from the class. Only the report is posted, the corresponding code is just as important.

1. Source localization in an ocean waveguide using supervised machine learning, Group3, Group6, Group8, Group10, Group11, Group15
2. Indoor positioning framework for most Wi-Fi-enabled devices, Group1
3. MyShake Seismic Data Classification, Group2
4. Multi Label Image Classification, Group4
5. Face Recognition using Machine Learning, Group7
6. Deep Learning for Star-Galaxy Classification, Group9
7. Modeling Neural Dynamics using Hidden Markov Models, Group12
8. Star Prediction Based on Yelp Business Data And Application in Physics, Group13
9. Si K edge X-ray spectrum absorption interpretation using Neural Network, Group14
10. Plankton Classification Using VGG16 Network, Group16
11. A Survey of Convolutional Neural Networks: Motivation, Modern Architectures, and Current Applications in the Earth and Ocean Sciences, Group17
12. Use satellite data to track the human footprint in the amazon rainforest, Group18
13. Automatic speaker diarization using machine learning techniques, Group19
14. Predicting Coral Colony Fate with Random Forest, Group20

Syllabus

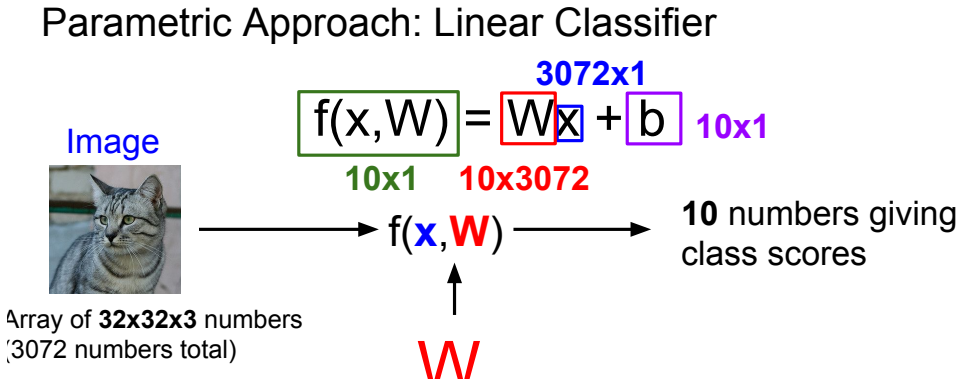
2018: Best reports 6,10,12 15; interesting 19, 47 poor 17; alone is hard 20.

Group	Topic	Authors	Poster	Report
1	**Reimplementation of source localization in an ocean waveguide using supervised learning**	Jinzhao Feng, Zhuoxi Zeng, Yu Zhang	Poster	Paper
2	**Machine learning methods for ship detection in satelite images**	Yifan Li, Huadong Zhang, Xiaoshi Li, Quianfeng Guo	Poster	Paper
3	**Transparent Conductor Prediction**	Yan Sun, Yiyuan Xing, Xufan Xiong, Tianduo Hao	Poster	Paper
4	**Ship identification in sateklite Images**	Weilun Zhang, Zhaoliang Zheng, Mingchen Mao,	Poster	Paper
5	**Fruit Recognition**	Eskil Jarslkog, Richard Wang, Joel Andersson	Poster	Paper
6	**RSNA Bone Age Prediction**	Juan Camilo Castillo, Yitian Tong, Jiyang Zhao, Fengcan Zhu	Poster	Paper
7	**Facial Expression Classification into Emotions**	David Orozco, Christopher Lee, Yevgeniy Arabadzhi, Deval Gupta	Poster	Paper
8	**Urban Scene Segmentation for Autonomous Vehicles**	Hsiao-Chen Huang, Eddie Tseng, Ping-Chun Chiang, Chih-Yen Lin	Poster	Paper
9	**Face Detection Using Deep Learning**	Yu Shen, Kuan-Wei Chen, Yizhou Hao, Min Hsuan Wu	Poster	Paper
10	**Understanding the Amazon Rainforest using Neural Networks**	Naveen Dharshana Ketagoda, Christian Jonathan Koguchi, Niral Lalit Pathak, Samuel Sunarjo	Poster	Paper
11	**Mercedes-Benz Bench Test Time Estimation**	Lanjihong Ma, Kexiong Wu, Bo Xiao, Zihang Yu	Poster	Paper
12	**Vegetation Classification in Hyperspectral Image**	Osman Cihan Kilinc, Kazim Ergun, Yuming Qiao, Fengjunyan Li	Poster	Paper
13	**Threat Detection Using AlexNet on TSA scans**	Amartya Bhattacharyya, Christine H Lind, Rahul Shirpurkar	Poster	Paper
14	**Flagellates Classification via Transfer Learning**	Eric Ho, Brian Henriquez, Jeffrey Yeung	Poster	Paper
15	**Biomedical Image Segmentation**	Lucas Tindall, Amir Persekian, Max Jiao	Poster	Paper
16	**"Deep Fakes" using Generative Adversarial Networks (GAN)**	Tianxiang Shen, Ruixian Liu, Ju Bai, Zheng Li	Poster	Paper
17	**Dog Breed Classification via Convolutional Neural Network**	Yizhou Chen; Xiaotong Chen; Xuanzhen Xu	Poster	Paper
18	**Dog Breed Identification**	Wenting Shi, Jiaquan Chen, Fangyu Liu, Muyun Liu	Poster	Paper
19	**Impact of Skewed Distributions on an Automated Plankton Classifier**	Will Chapman, Emal Fatima, William Jenkins, Steven Tien, Shawheen Tosifian	Poster	Paper
20	**Blood Cell Detection using Single shot MultiBox Detector**	Inyoung Huh	Poster	Paper

Bayes and Softmax (Bishop p. 198)

- Bayes:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)} = \frac{p(y|x)p(x)}{\sum_{y \in Y} p(x,y)}$$



- Classification of N classes:

$$p(\mathcal{C}_n|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_n)p(\mathcal{C}_n)}{\sum_{k=1}^{N} p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}$$

$$= \frac{\exp(a_n)}{\sum_{k=1}^{N} \exp(a_k)}$$

with

$$a_n = \ln\left(p(\mathbf{x}|\mathcal{C}_n)p(\mathcal{C}_n)\right)$$

Softmax to Logistic Regression (Bishop p. 198)

$$p(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{\sum_{k=1}^{2} p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}$$

$$= \frac{\exp(a_1)}{\sum_{k=1}^{2} \exp(a_k)} = \frac{1}{1+\exp(-a)}$$

with

$$a = \ln \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)}$$

- $a_1 = \ln[p(\boldsymbol{x}|C_1)p(C_1)]$
- $a = a_1 - a_2$
- $p(C_1|x) = \frac{1}{1+\exp(a_2-a_1)}$

The Kullback-Leibler Divergence

P true distribution, q is approximating distribution

$$\begin{aligned}\mathrm{KL}(p\|q) &= -\int p(\mathbf{x})\ln q(\mathbf{x})\,\mathrm{d}\mathbf{x} - \left(-\int p(\mathbf{x})\ln p(\mathbf{x})\,\mathrm{d}\mathbf{x}\right) \\ &= -\int p(\mathbf{x})\ln\left\{\frac{q(\mathbf{x})}{p(\mathbf{x})}\right\}\,\mathrm{d}\mathbf{x}\end{aligned}$$

$$\mathrm{KL}(p\|q) \simeq \frac{1}{N}\sum_{n=1}^{N}\{-\ln q(\mathbf{x}_n|\boldsymbol{\theta}) + \ln p(\mathbf{x}_n)\}$$

$$\mathrm{KL}(p\|q) \geqslant 0 \qquad\qquad \mathrm{KL}(p\|q) \not\equiv \mathrm{KL}(q\|p)$$

Cross entropy

- KL divergence (p true q approximating)

$$D_{\{KL\}}(p||q) = \sum_n^N p_n \ln(p_n) - \sum_n^N p_n \ln(q_n)$$
$$= -H(p) + H(p,q)$$

- Cross entropy

$$H(p,q) = H(q) + D_{\{KL\}}(p||q) = -\sum_n^N p_n \ln(q_n)$$

- *Implementations*

tf.keras.losses.CategoricalCrossentropy()

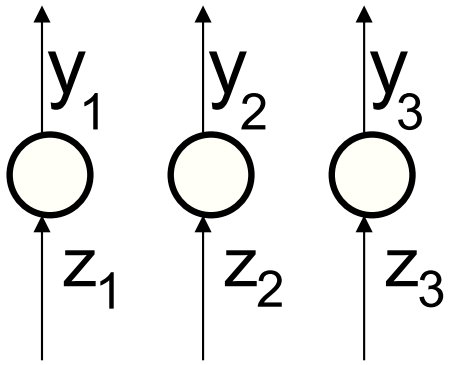
tf.losses.sparse_softmax_cross_entropy

torch.nn.CrossEntropyLoss()

Cross-entropy or "**softmax**" function for multi-class classification

The output units use a non-local non-linearity:

$$y_i = \frac{e^{z_i}}{\sum_j e^{z_j}}$$



output units

$$\frac{\partial y_i}{\partial z_i} = y_i\,(1 - y_i)$$

The natural cost function is the negative log prob of the right answer

target value

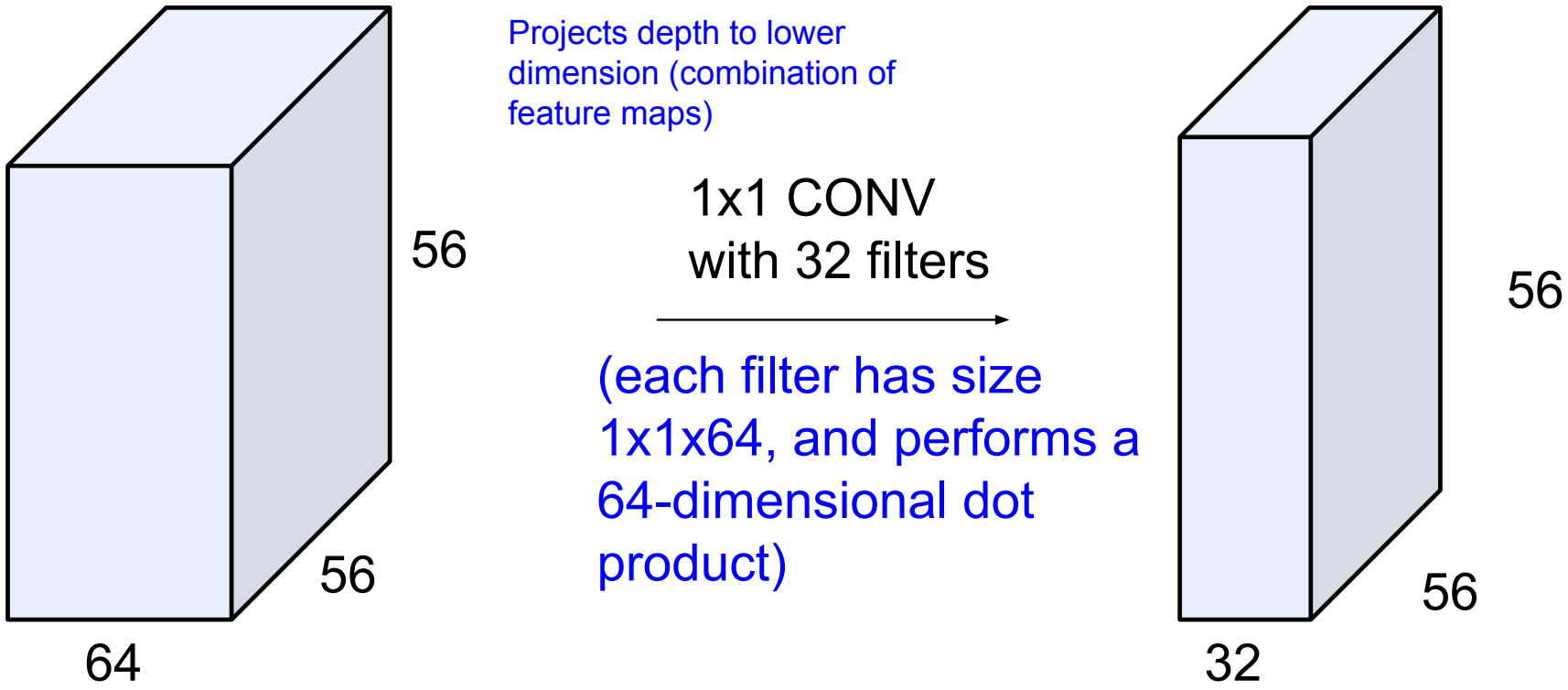
$$E = -\sum_j t_j \ln y_j$$

$$\frac{\partial E}{\partial z_i} = \sum_j \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial z_i} = y_i - t_i$$

Reminder: 1x1 convolutions

preserves spatial dimensions, reduces depth!

Projects depth to lower dimension (combination of feature maps)



Summary: CNN Architectures

Case Studies

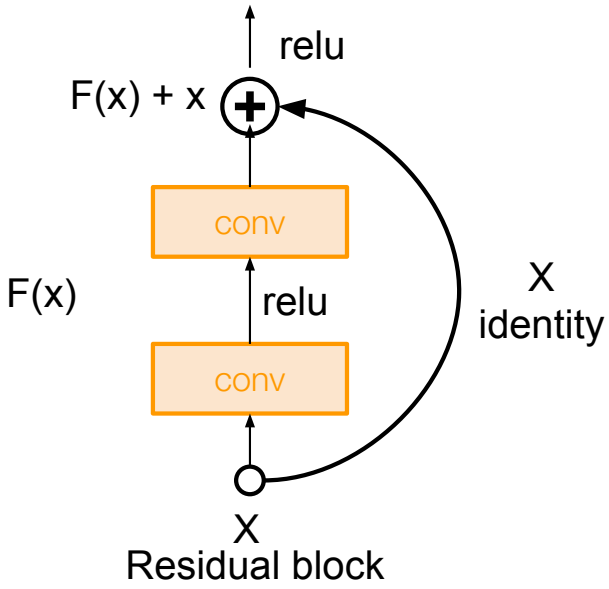
- AlexNet
- VGG
- GoogLeNet
- ResNet

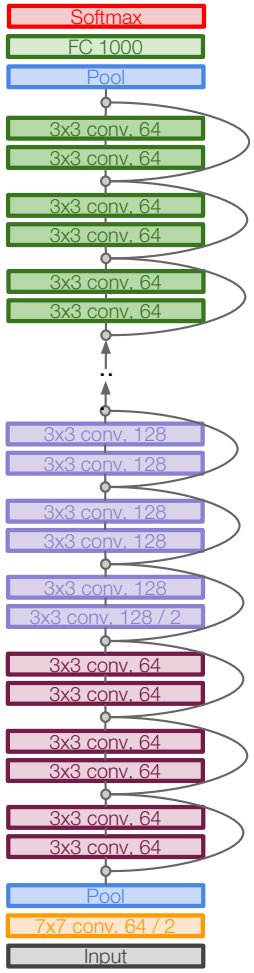
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC'15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC'15 and COCO'15!

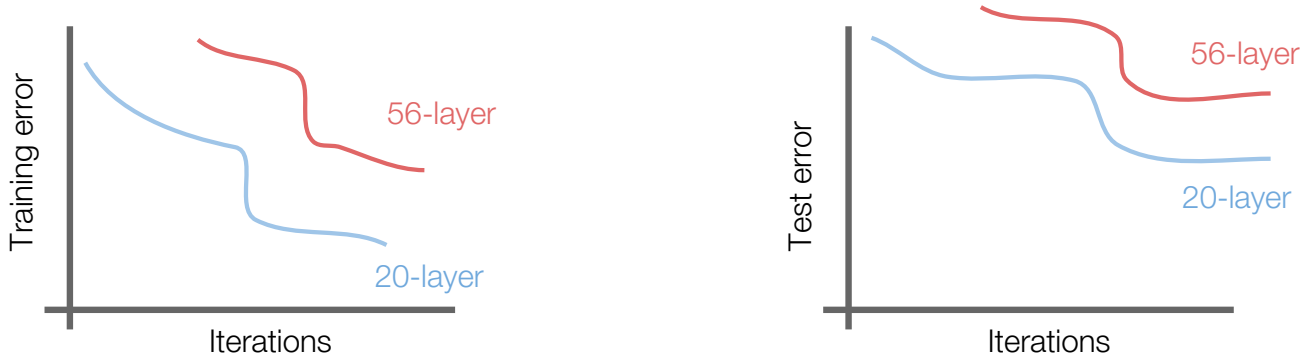




Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?



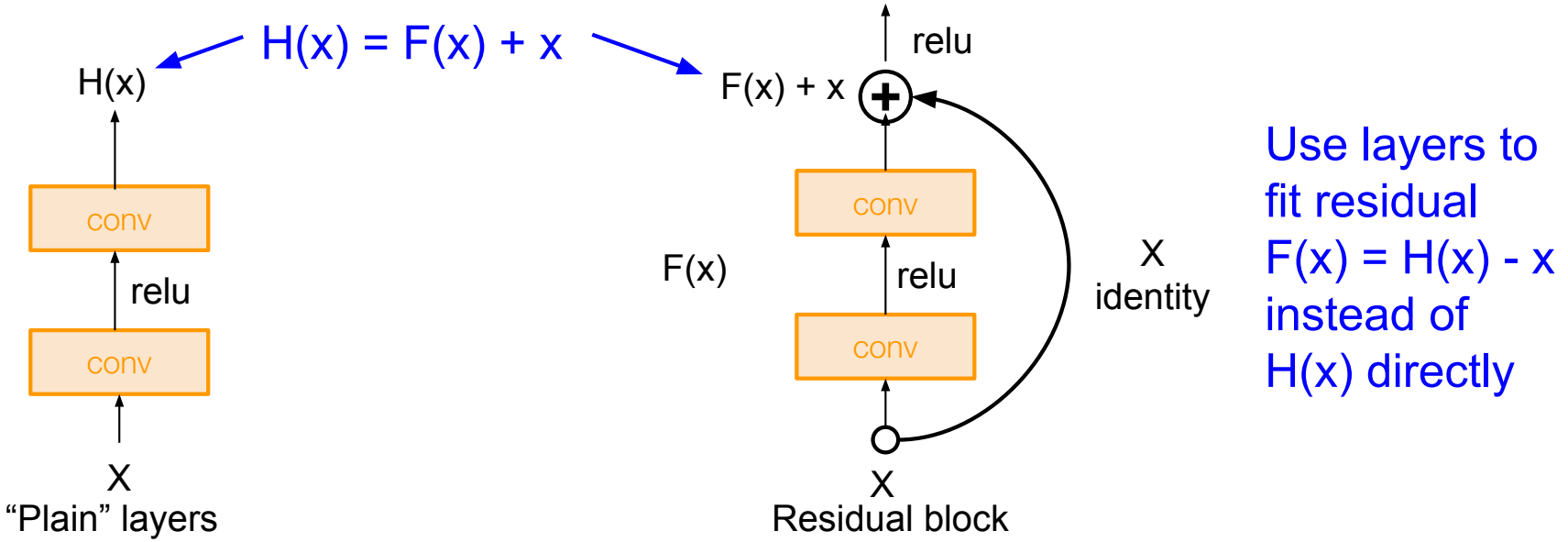
56-layer model performs worse on both training and test error
-> The deeper model performs worse, but it’s not caused by overfitting!

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping



Use layers to fit residual F(x) = H(x) - x instead of H(x) directly

Kernels

- Kernel function

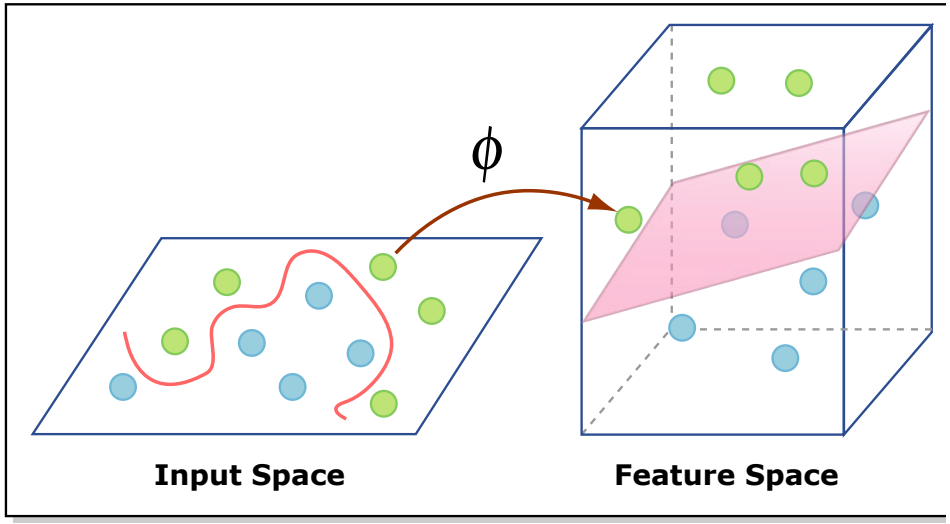
$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}'). \tag{6.1}$$

- Kernel trick: substitute the inner product of freatures with

Kernels

We might want to consider something more complicated than a linear model:

Example 1: $[x^{(1)}, x^{(2)}] \rightarrow \mathbf{\Phi}\left([x^{(1)}, x^{(2)}]\right) = \left[x^{(1)2}, x^{(2)2}, x^{(1)}x^{(2)}\right]$



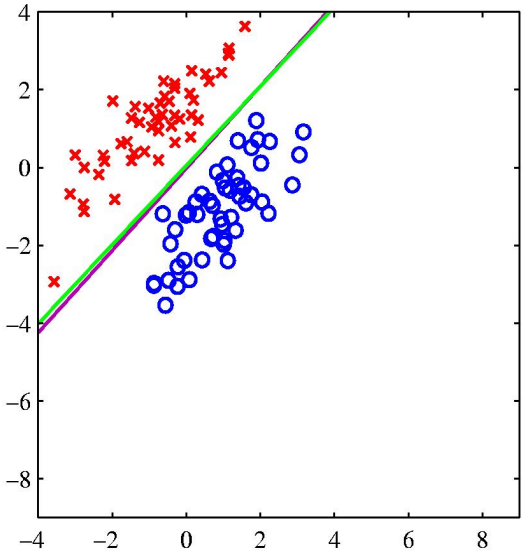
Information unchanged, but now we have a **linear** classifier on the transformed points.

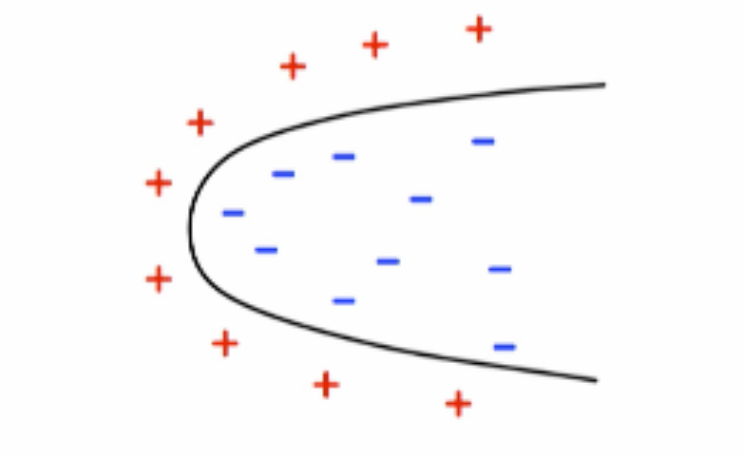
With the kernel trick, we just need kernel

$k(\boldsymbol{a}, \boldsymbol{b}) = \boldsymbol{\Phi}(\boldsymbol{a})^T \, \boldsymbol{\Phi}(\boldsymbol{b})$

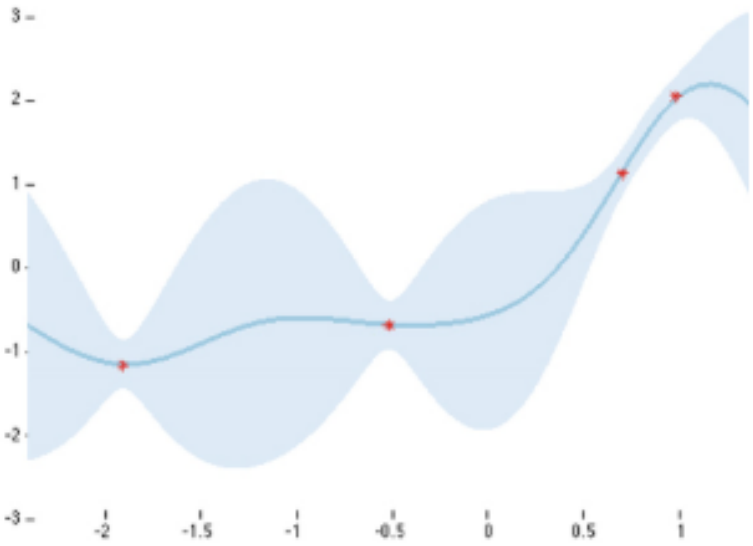
$$k(\mathbf{x}, \mathbf{x}') = \boldsymbol{\phi}(\mathbf{x})^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}'). \qquad (6.1)$$

Basis expansion





Gaussian Process (Bishop 6.4, Murphy15)



This is what a Gaussian process *posterior* looks like with 4 data points and a squared exponential covariance function. The bold blue line is the predictive mean, while the light blue shade is the predictive uncertainty (2 standard deviations). The model uncertainty is small near the data, and increases as we move away from the data points.

$$t_n = y_n + \epsilon_n$$

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), \kappa(\mathbf{x}, \mathbf{x}'))$$

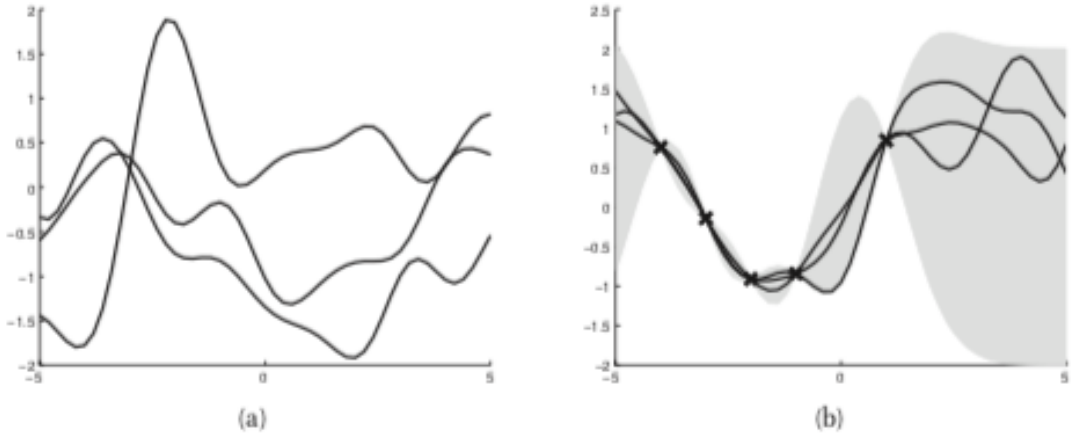


Figure 15.2 Left: some functions sampled from a GP prior with SE kernel. Right: some samples from a GP posterior, after conditioning on 5 noise-free observations. The shaded area represents $\mathbb{E}[f(\mathbf{x})] \pm 2\text{std}(f(\mathbf{x}))$. Based on Figure 2.2 of (Rasmussen and Williams 2006). Figure generated by `gprDemoNoiseFree`.

Dual representation, Sec 6.2

Primal problem: $\min_{\boldsymbol{w}} E(\boldsymbol{w})$

$$E = \frac{1}{2}\sum_n^N \{\boldsymbol{w}^T\boldsymbol{x}_n - t_n\}^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2$$

Solution $\boldsymbol{w} = \boldsymbol{X}^+\boldsymbol{t} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_M)^{-1}\boldsymbol{X}^T\boldsymbol{t}$

$= \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda\boldsymbol{I}_N)^{-1}\boldsymbol{t} = \boldsymbol{X}^T(\boldsymbol{K} + \lambda\boldsymbol{I}_N)^{-1}\boldsymbol{t} = \boldsymbol{X}^T\boldsymbol{a}$

The kernel is $\mathbf{K} = \boldsymbol{X}\boldsymbol{X}^T$

Dual representation is : $\min_{\boldsymbol{a}} E(\boldsymbol{a})$

$$E = \frac{1}{2}\sum_n^N \{\boldsymbol{w}^T\boldsymbol{x}_n - t_n\}^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \|\boldsymbol{K}\boldsymbol{a} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$

a is found inverting NxN matrix

w is found inverting MxM matrix

Only kernels, no feature vectors

Dual representation, Sec 6.2

Dual representation is : $\min_{\boldsymbol{a}} E(\boldsymbol{a})$

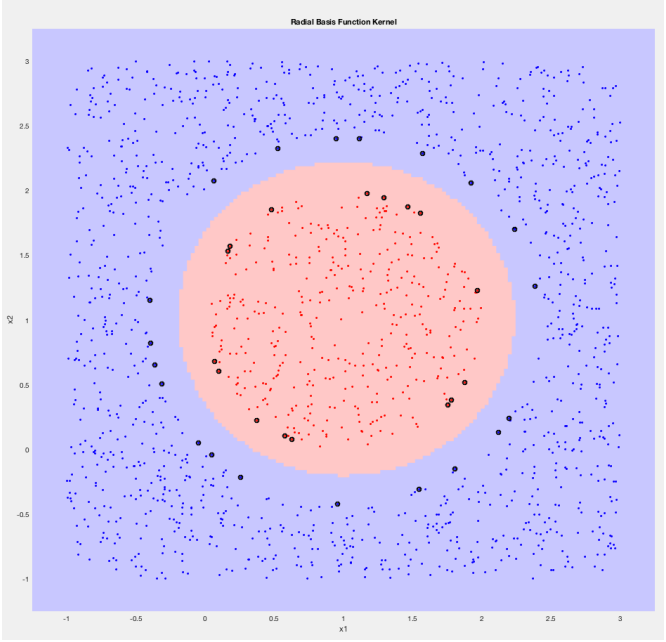
$$E = \frac{1}{2}\sum_n^N \{\boldsymbol{w}^T \boldsymbol{x}_n - t_n\}^2 + \frac{\lambda}{2}\|\boldsymbol{w}\|^2 = \|\boldsymbol{K}\boldsymbol{a} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$

Prediction

$$y = \boldsymbol{w}^T\boldsymbol{x} = \boldsymbol{a}^T\boldsymbol{X}\boldsymbol{x} = \sum_n^N a_n \boldsymbol{x}_n^T\boldsymbol{x} = \sum_n^N a_n k(\boldsymbol{x}_n, \boldsymbol{x})$$

- Often **a is sparse (…** Support vector machines**)**
- We don't need to know **x or** $\boldsymbol{\varphi(x)}$**.** $\boldsymbol{Just\ the\ Kernel}$

$$E(\boldsymbol{a}) = \|\boldsymbol{K}\boldsymbol{a} - \boldsymbol{t}\|_2^2 + \frac{\lambda}{2}\boldsymbol{a}^T\boldsymbol{K}\boldsymbol{a}$$



Gaussian Kernels

- Gaussian Kernel

$$k(x, x') = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{x}')\right)$$

Diagonal Σ: (this gives ARD)

$$k(x, x') = \exp\left(-\frac{1}{2}\sum_i^N \frac{(x_i - x_i')^2}{\sigma_i^2}\right)$$

Isotropic σ_i^2 gives an RBF

$$k(x, x') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right)$$

Gaussian Kernels

- Gaussian Kernel

$$k(x, x') = \exp\left(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x}')^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{x}')\right)$$

Diagonal Σ: (this gives ARD)

$$k(x, x') = \exp\left(-\frac{1}{2}\sum_i^N \frac{(x_i - x_i')^2}{\sigma_i^2}\right)$$

Isotropic σ_i^2 gives an RBF

$$k(x, x') = \exp\left(-\frac{\|\boldsymbol{x} - \boldsymbol{x}'\|_2^2}{2\sigma^2}\right)$$

Commonly used kernels

Polynomial: $K(\mathbf{x},\mathbf{y}) = (\mathbf{x}.\mathbf{y}+1)^p$

Gaussian radial basis function: $K(\mathbf{x},\mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2}$

Neural net: $K(\mathbf{x},\mathbf{y}) = \tanh(k\,\mathbf{x}.\mathbf{y} - \delta)$

Parameters that the user must choose

For the neural network kernel, there is one "hidden unit" per support vector, so the process of fitting the maximum margin hyperplane decides how many hidden units to use. Also, it may violate Mercer's condition.

Example 4:

$$\begin{aligned} k(\mathbf{x},\mathbf{z}) &= (\mathbf{x}^T\mathbf{z}+c)^2 = \left(\sum_{j=1}^{n} x^{(j)}z^{(j)} + c\right)\left(\sum_{\ell=1}^{n} x^{(\ell)}z^{(\ell)} + c\right) \\ &= \sum_{j=1}^{n}\sum_{\ell=1}^{n} x^{(j)}x^{(\ell)}z^{(j)}z^{(\ell)} + 2c\sum_{j=1}^{n} x^{(j)}z^{(j)} + c^2 \\ &= \sum_{j,\ell=1}^{n} (x^{(j)}x^{(\ell)})(z^{(j)}z^{(\ell)}) + \sum_{j=1}^{n} (\sqrt{2c}x^{(j)})(\sqrt{2c}z^{(j)}) + c^2, \end{aligned}$$

and in $n = 3$ dimensions, one possible feature map is:

$$\mathbf{\Phi}(\mathbf{x}) = [x^{(1)2}, x^{(1)}x^{(2)}, ..., x^{(3)2}, \sqrt{2c}x^{(1)}, \sqrt{2c}x^{(2)}, \sqrt{2c}x^{(3)}, c]$$

and c controls the relative weight of the linear and quadratic terms in the inner product.

Even more generally, if you wanted to, you could choose the kernel to be any higher power of the regular inner product.

Can be inner product in infinite dimensional space

Assume $x \in R^1$ and $\gamma > 0$.

$$e^{-\gamma\|x_i - x_j\|^2} = e^{-\gamma(x_i - x_j)^2} = e^{-\gamma x_i^2 + 2\gamma x_i x_j - \gamma x_j^2}$$

$$= e^{-\gamma x_i^2 - \gamma x_j^2}\left(1 + \frac{2\gamma x_i x_j}{1!} + \frac{(2\gamma x_i x_j)^2}{2!} + \frac{(2\gamma x_i x_j)^3}{3!} + \cdots\right)$$

$$= e^{-\gamma x_i^2 - \gamma x_j^2}\left(1 \cdot 1 + \sqrt{\frac{2\gamma}{1!}} x_i \cdot \sqrt{\frac{2\gamma}{1!}} x_j + \sqrt{\frac{(2\gamma)^2}{2!}} x_i^2 \cdot \sqrt{\frac{(2\gamma)^2}{2!}} x_j^2 \right.$$

$$\left. + \sqrt{\frac{(2\gamma)^3}{3!}} x_i^3 \cdot \sqrt{\frac{(2\gamma)^3}{3!}} x_j^3 + \cdots\right) = \phi(x_i)^T \phi(x_j),$$

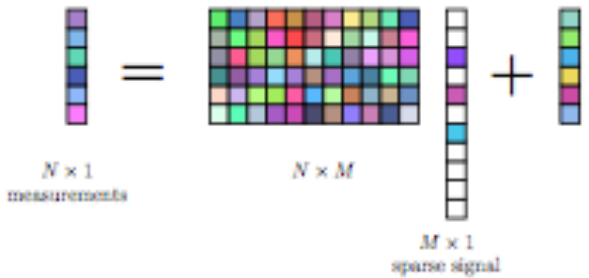
where

$$\phi(x) = e^{-\gamma x^2}\left[1, \sqrt{\frac{2\gamma}{1!}} x, \sqrt{\frac{(2\gamma)^2}{2!}} x^2, \sqrt{\frac{(2\gamma)^3}{3!}} x^3, \cdots\right]^T.$$

- FINISHED HERE 30 April 2018
- Showed also http://playground.tensorflow.org/ in the last 10 min.

Sparse Bayesian Learning (SBL)

$$\text{Model}: \mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{n}$$
$$\text{Prior}: \mathbf{x} \sim \mathcal{N}(\mathbf{x}; 0, \mathbf{\Gamma})$$
$$\mathbf{\Gamma} = \text{diag}(\gamma_1, \ldots, \gamma_M)$$
$$\text{Likelihood}: p(\mathbf{y}|\mathbf{x}) = \mathcal{N}(\mathbf{y}; \mathbf{A}\mathbf{x}, \sigma^2 \mathbf{I}_N)$$



$$\text{Evidence}: p(\mathbf{y}) = \int_{\mathbf{x}} p(\mathbf{y}|\mathbf{x}) p(\mathbf{x}) d\mathbf{x} = \mathcal{N}(\mathbf{y}; 0, \mathbf{\Sigma}_\mathbf{y})$$
$$\mathbf{\Sigma}_\mathbf{y} = \sigma^2 \mathbf{I}_N + \mathbf{A}\mathbf{\Gamma}\mathbf{A}^H$$

$$\text{SBL solution}: \hat{\mathbf{\Gamma}} = \arg\max_{\mathbf{\Gamma}} p(\mathbf{y})$$
$$= \arg\min_{\mathbf{\Gamma}} \left\{ \log|\mathbf{\Sigma}_\mathbf{y}| + \mathbf{y}^H \mathbf{\Sigma}_\mathbf{y}^{-1} \mathbf{y} \right\}$$

M.E.Tipping, "Sparse Bayesian learning and the relevance vector machine," Journal of Machine Learning Research, June 2001.

Solving a Rank-Deficient System

If A is *m*-by-*n* with *m* > *n* and full rank *n*, each of the three statements

```
x = A\b
x = pinv(A)*b
x = inv(A'*A)*A'*b
```

theoretically computes the same least-squares solution x, although the backslash operator does it faster.

Nice slide, But why?

However, if A does not have full rank, the solution to the least-squares problem is not unique. There are many vectors x that minimize

```
norm(A*x -b)
```

The solution computed by x = A\b is a basic solution; it has at most *r* nonzero components, where *r* is the rank of A. The solution computed by x = pinv(A)*b is the minimal norm solution because it minimizes norm(x). An attempt to compute a solution with x = inv(A'*A)*A'*b fails because A'*A is singular.

Lecture 10
Support Vector Machines

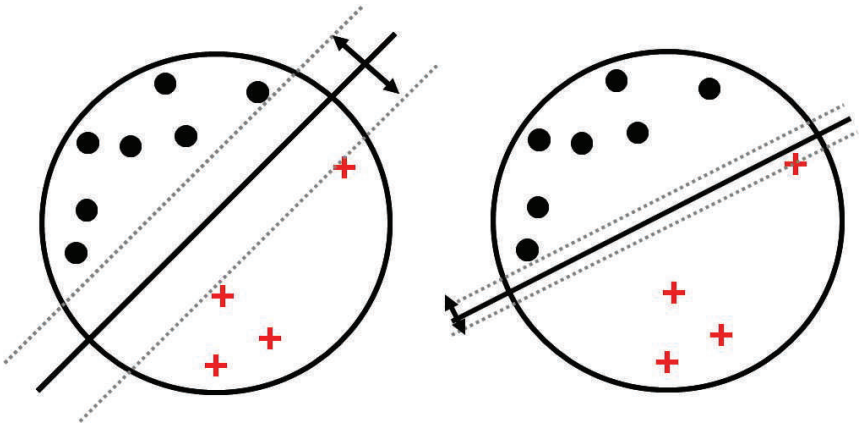
Non Bayesian!

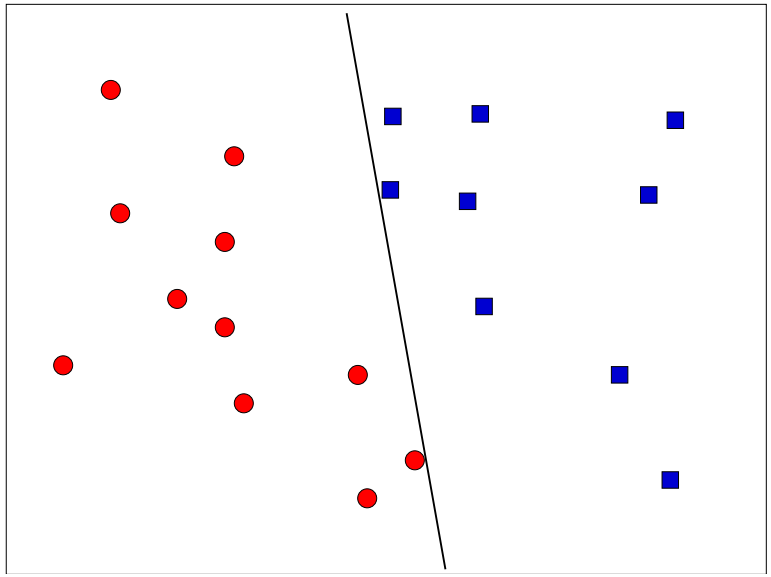
Features:

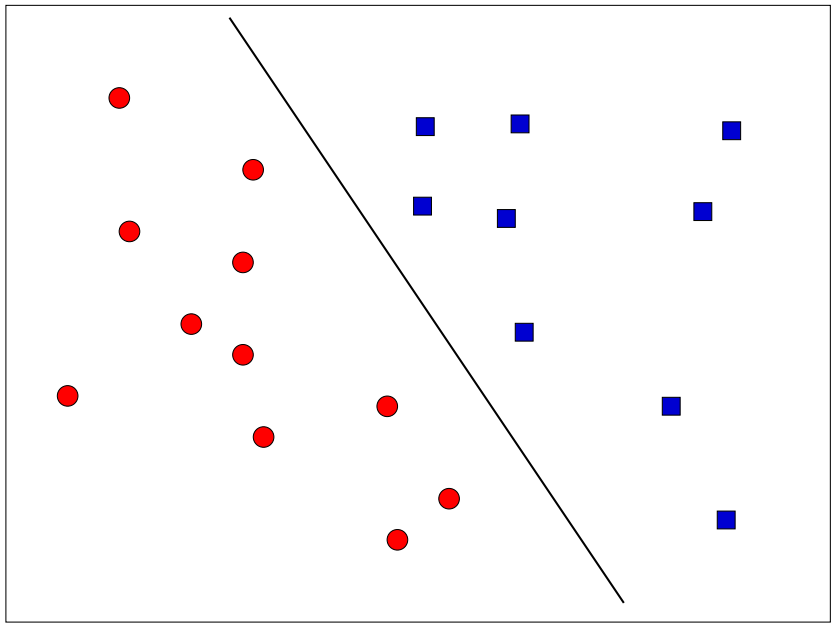
- Kernel
- Sparse representations
- Large margins

Regularize for plausibility

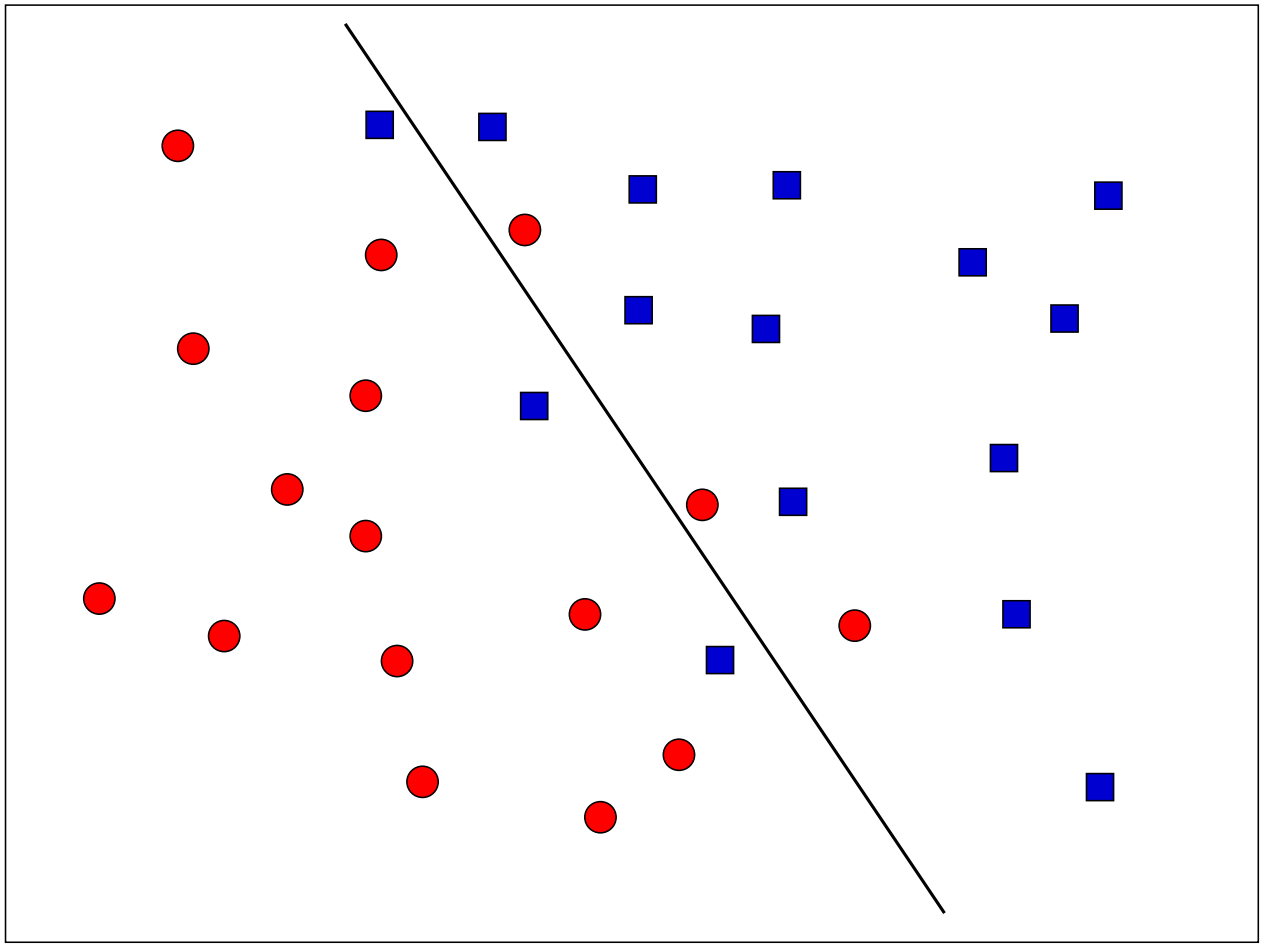
- Which one is best?
- We maximize the margin





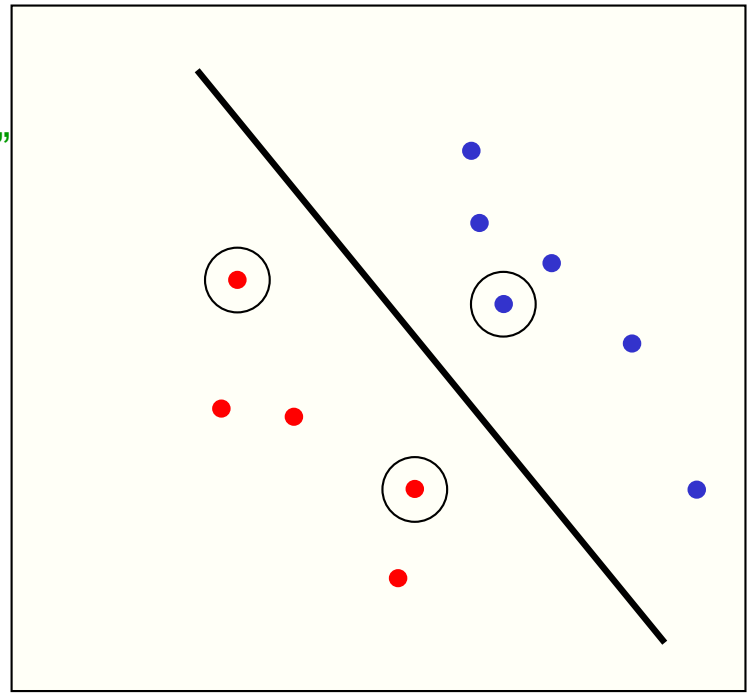


Regularize for plausibility



Support Vector Machines

- The line that maximizes the minimum margin is a good bet.
  - The model class of "hyper-planes with a margin *m*" has a low VC dimension if *m* is big.
- This maximum-margin separator is determined by a subset of the datapoints.
  - Datapoints in this subset are called "support vectors".
  - It is useful computationally if only few datapoints are support vectors, because the support vectors decide which side of the separator a test case is on.



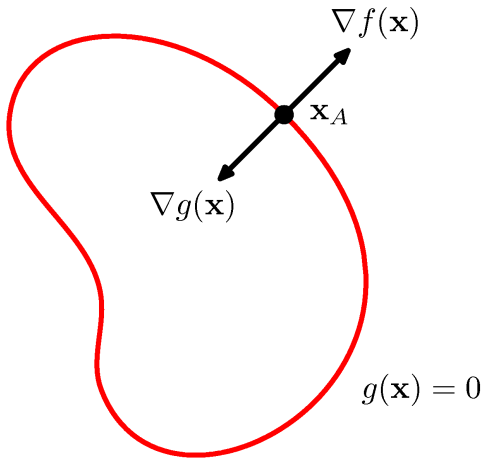
The support vectors are indicated by the circles around them.

Lagrange multiplier (Bishop App E)

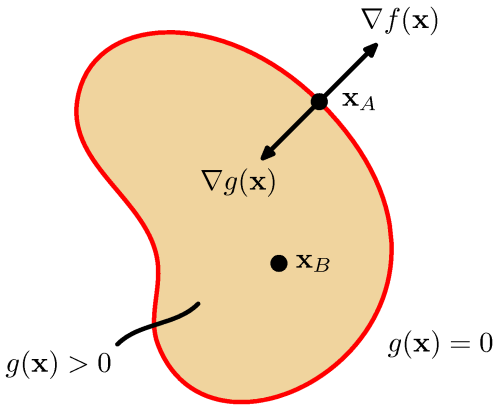
$$\max\big(f(x)\big) \text{ subject to } g(x) = 0$$

Taylor expansion

$$g(\boldsymbol{x} + \boldsymbol{\varepsilon}) = g(\boldsymbol{x}) + \boldsymbol{\epsilon}^T \nabla\, g(\boldsymbol{x})$$



$$L(x, \lambda) = f(x) + \lambda g(x)$$

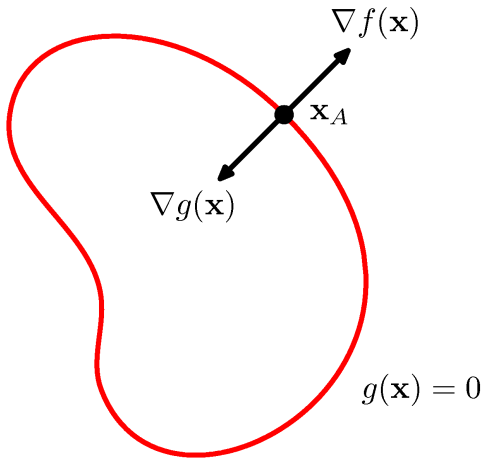


Lagrange multiplier (Bishop App E)

$$\max\big(f(\boldsymbol{x})\big) \text{ subject to } g(\boldsymbol{x}) > 0$$
$$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$$

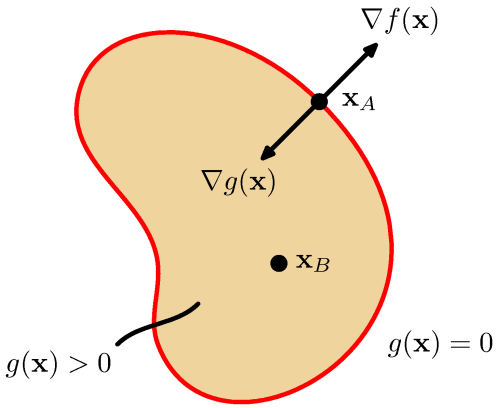
Either $\nabla \mathrm{f}(\boldsymbol{x}) = \mathbf{0}$
Then $g(\boldsymbol{x})$ is inactive, λ=0

$Or\ g(\boldsymbol{x}) = 0\ but\ \lambda > 0$



Thus optimizing $L(\boldsymbol{x}, \lambda)$ with the Karesh-Kuhn-Trucker (KKT) equations

$$g(\boldsymbol{x}) \geq 0$$
$$\lambda \geq 0$$
$$\lambda g(\boldsymbol{x}) = 0$$



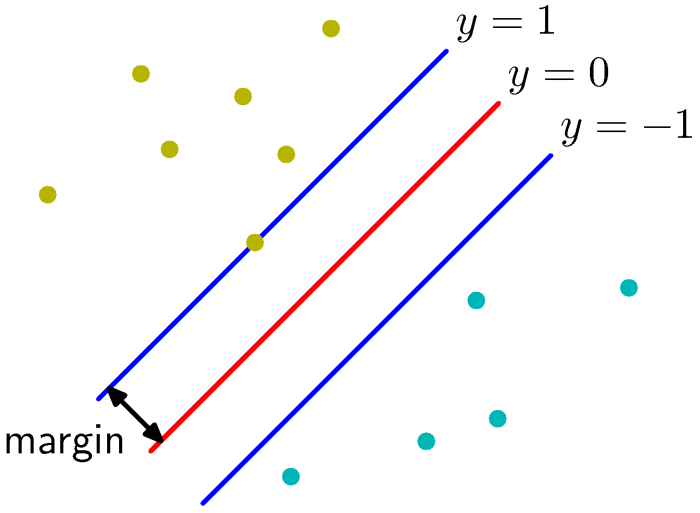
Testing a linear SVM

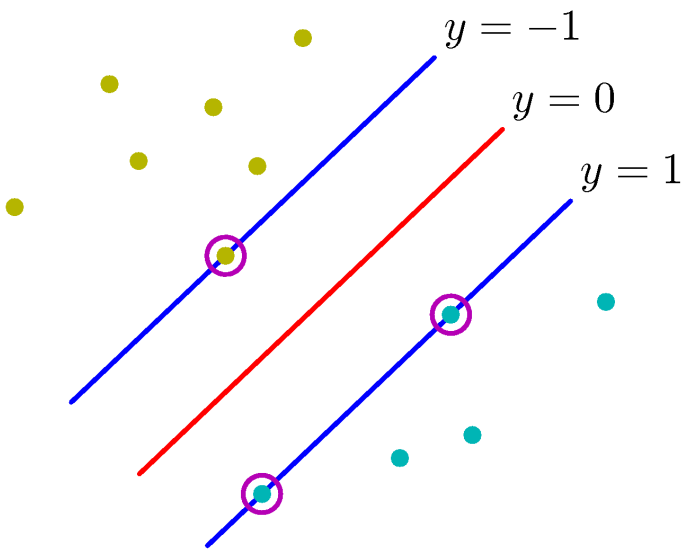
- The separator is defined as the set of points for which:

$$\mathbf{w}.\mathbf{x} + b = 0$$

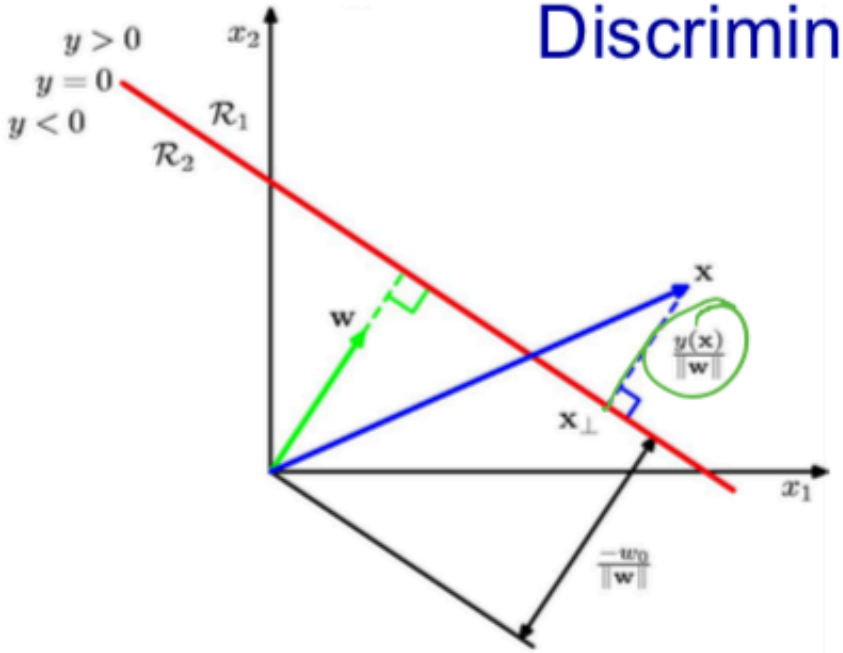
$$so\ if\ \ \mathbf{w}.\mathbf{x}^c + b > 0\ \ say\ its\ a\ positive\ case$$

$$and\ if\ \ \mathbf{w}.\mathbf{x}^c + b < 0\ \ say\ its\ a\ negative\ case$$





Discriminant functions



The planar decision surface in data-space for the simple linear discriminant function:

$$\mathbf{w}^T\mathbf{x} + w_0 \geq 0$$

$y = w^T x + w_0$.

$y_\perp = w^T x_\perp + w_0 = 0$ ✓

$\|w\|_2^2 = w^T w$

X on plane => y=0 =>

Distance from plane

$r = \frac{y}{\|w\|_2}$

$x = x_\perp + r\frac{w}{\|w\|_2}$

$w^T x = w^T x_\perp + r\frac{w^T w}{\|w\|_2}$

$= -w_0 + r\|w\|_2$

$x = x_\perp + \frac{y}{\|w\|_2}$

Large margin

$$y = \boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x}) + b$$

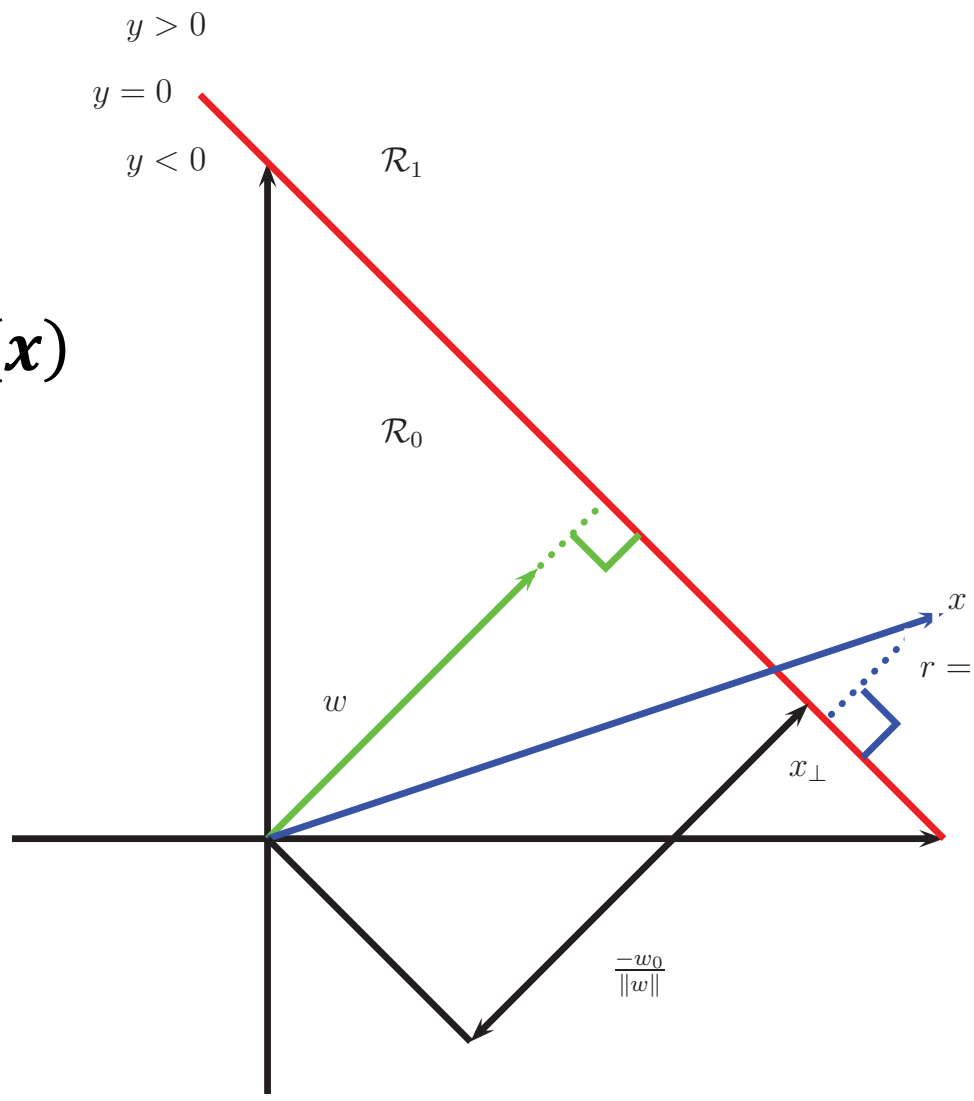
$$\boldsymbol{x}_n = \boldsymbol{x}_\perp + r_n \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

x on plane => y=0 =>$b = -\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x})$

$$r_n = \frac{\boldsymbol{w}^T \boldsymbol{\phi}(\boldsymbol{x_n}) + b}{\|\boldsymbol{w}\|} = \frac{y_n}{\|\boldsymbol{w}\|}$$

$$t_n y_n \geq 1$$

$$\max_{\boldsymbol{w}} \frac{1}{\|\boldsymbol{w}\|} \min_n t_n y_n$$



Maximum margin (Bishop 7.1)

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

Subject to

$$t_n\left(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)+b\right) \geqslant 1, \qquad n=1,\ldots,N. \tag{7.5}$$

Lagrange function

$$L(\mathbf{w},b,\mathbf{a}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{n=1}^{N} a_n\left\{t_n(\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)+b)-1\right\} \tag{7.7}$$

Differentiation

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \tag{7.8}$$

$$0 = \sum_{n=1}^{N} a_n t_n. \tag{7.9}$$

Dual representation

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n,\mathbf{x}_m) \tag{7.10}$$

with respect to $\mathbf{a}$ subject to the constraints

$$a_n \geqslant 0, \qquad n=1,\ldots,N, \tag{7.11}$$

$$\sum_{n=1}^{N} a_n t_n = 0. \tag{7.12}$$

This can be solved with quadratic programming

Maximum margin (Bishop 7.1)

- KKT conditions

$$a_n \geqslant 0 \tag{7.14}$$
$$t_n y(\mathbf{x}_n) - 1 \geqslant 0 \tag{7.15}$$
$$a_n \{t_n y(\mathbf{x}_n) - 1\} = 0. \tag{7.16}$$

either $a_n = 0$ or $t_n y(\mathbf{x}_n) = 1$.

- Solving for a_n

$$\mathbf{w} = \sum_{n=1}^{N} a_n t_n \boldsymbol{\phi}(\mathbf{x}_n) \tag{7.8}$$

- Prediction

$$y(\mathbf{x}) = \sum_{n=1}^{N} a_n t_n k(\mathbf{x}, \mathbf{x}_n) + b. \tag{7.13}$$

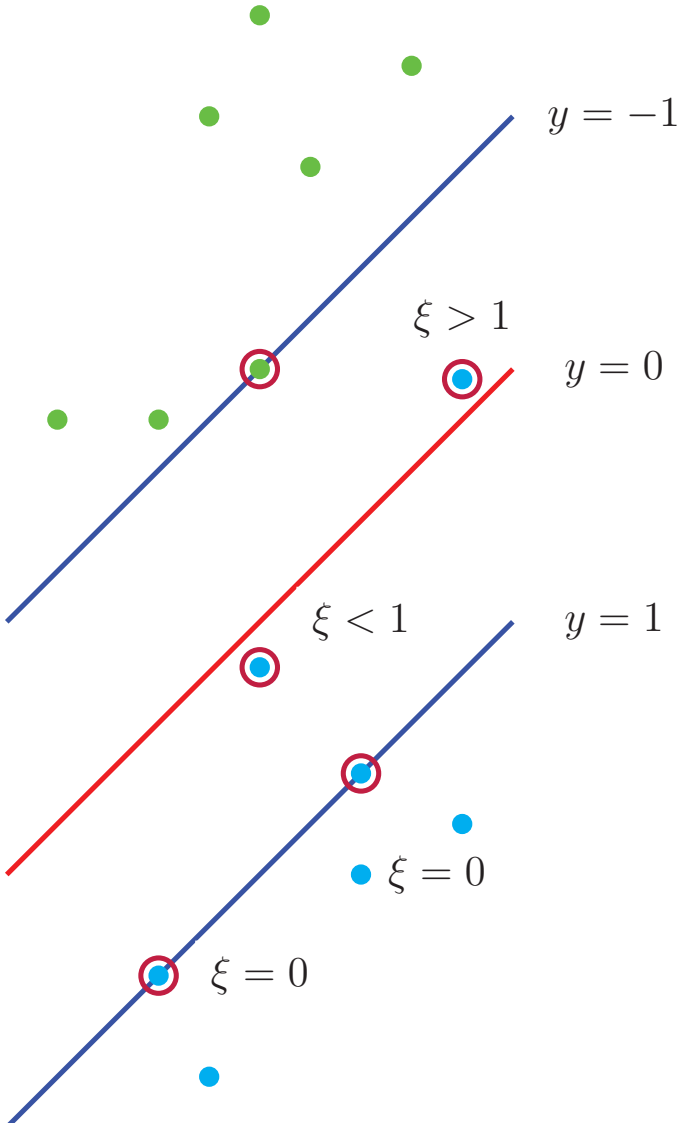
If there is no separating plane…

- Use a bigger set of features.
  - Makes the computation slow? "Kernel" trick makes the computation fast with many features.
- Extend definition of maximum margin to allow non-separating planes.
  - Use "slack" variables $\xi = |t_n - y(\boldsymbol{x}_n)|$

$$t_n y(\mathbf{x}_n) \geqslant 1 - \xi_n, \qquad n = 1, \ldots, N \qquad (7.20)$$

Objective function

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}\|\mathbf{w}\|^2 \qquad (7.21)$$



SVM classification summarized--- Only kernels

- Minimize with respect to $\boldsymbol{w}, \mathrm{w}_0$

$$C \sum_n^N \zeta_n + \frac{1}{2} \|\boldsymbol{w}\|^2 \qquad \text{(Bishop 7.21)}$$

- Solution found in dual domain with Lagrange multipliers
  - $a_n, n = 1 \cdots N$ and
- This gives the support vectors S

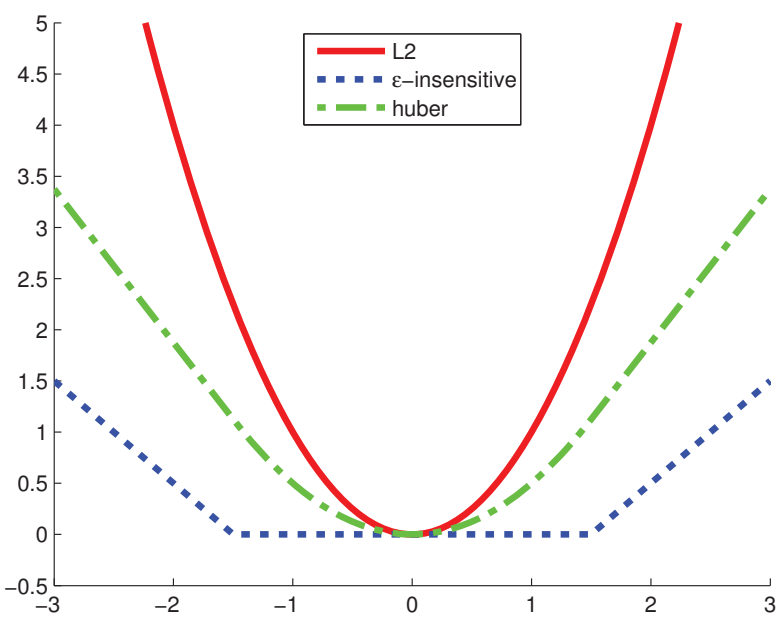
$$\hat{\boldsymbol{w}} = \sum_{n \in S} a_n \, t_n \boldsymbol{\varphi}(xn) \qquad \text{(Bishop 7.8)}$$

- Used for predictions

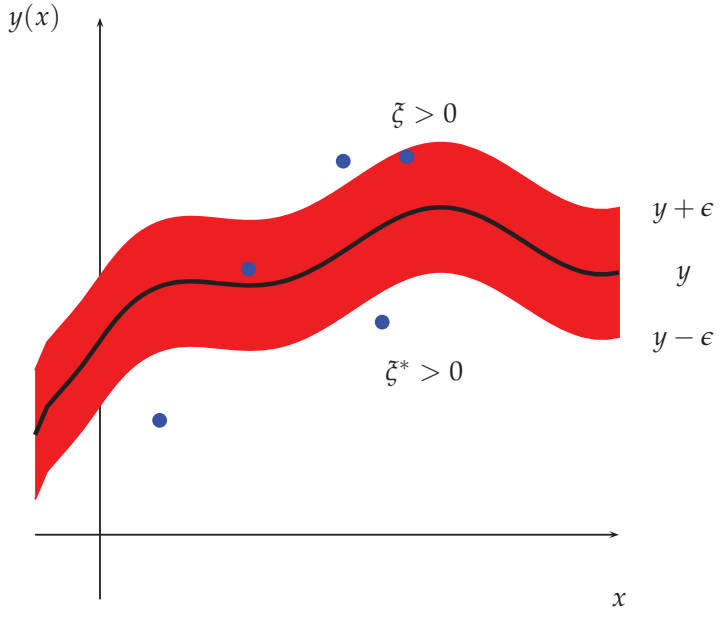
$$\hat{y} = \mathrm{w}_0 + \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\varphi}(x) = \mathrm{w}_0 + \sum_{n \in S} a_n \, t_n \boldsymbol{\varphi}(x_n)^{\mathrm{T}} \boldsymbol{\varphi}(x)$$

$$= \mathrm{w}_0 + \sum_{n \in S} a_n \, t_n k(x_n, x) \qquad \text{(Bishop 7.13)}$$

SVM for regression



(a)



(b)

Figure 14.10 (a) Illustration of ℓ_2, Huber and ϵ-insensitive loss functions, where $\epsilon = 1.5$. Figure generated by `huberLossDemo`. (b) Illustration of the ϵ-tube used in SVM regression. Points above the tube have $\xi_i > 0$ and $\xi_i^* = 0$. Points below the tube have $\xi_i = 0$ and $\xi_i^* > 0$. Points inside the tube have $\xi_i = \xi_i^* = 0$. Based on Figure 7.7 of (Bishop 2006a).

SVMs are Perceptrons!

- SVM's use each training case, x, to define a feature K(x, .) where K is user chosen.
  - So the user designs the features.
- SVM do "feature selection" by picking support vectors, and learn feature weighting from a big optimization problem.
- =>SVM is a clever way to train a standard perceptron.
  - What a perceptron cannot do, SVM cannot do.

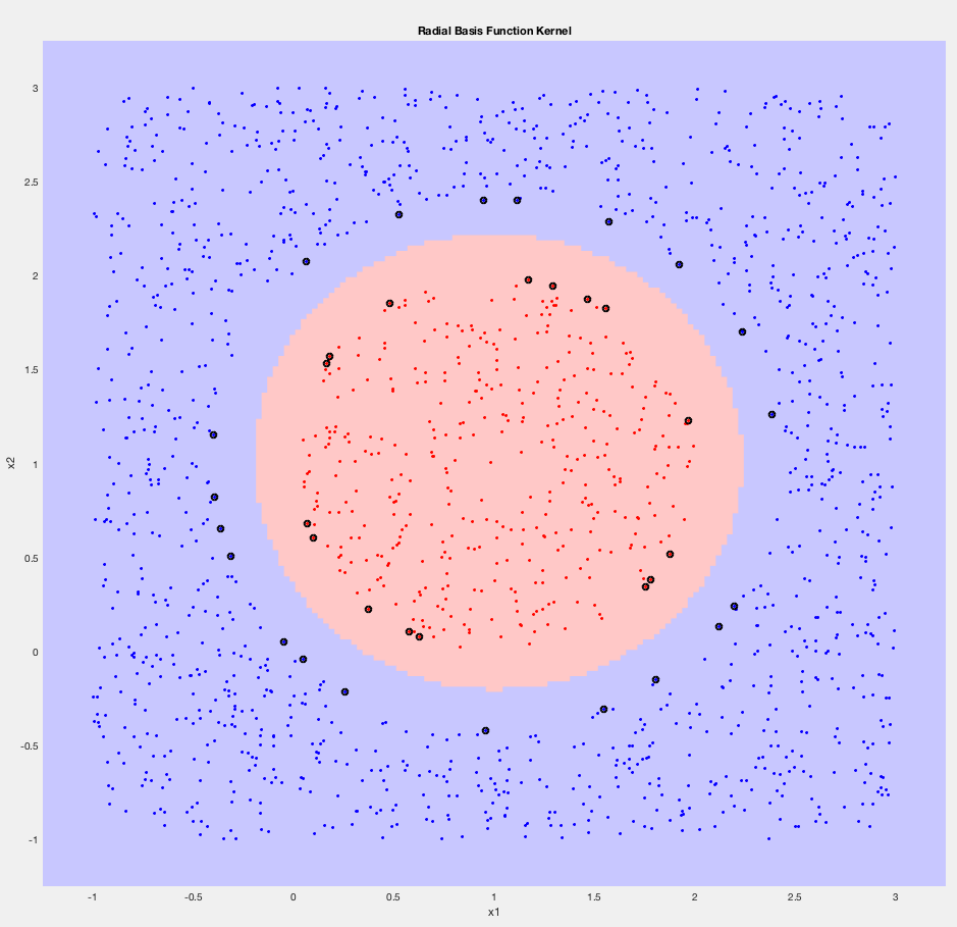
- SVM DOES:
  - Margin maximization
  - Kernel trick
  - Sparse

SVM Code for classification (libsvm)

Part of ocean acoustic data set http://noiselab.ucsd.edu/ECE285/SIO209Final.zip

case 'Classify'

```
% train
    model = svmtrain(Y, X,['-c 7.46 -g ' gamma ' -q ' kernel]);
% predict
    [predict_label,~, ~] = svmpredict(rand([length(Y),1]), X, model,'-q');
```




```
>> modelmodel =    struct with fields:
Parameters: [5×1 double]
nr_class: 2
totalSV: 36
rho: 8.3220
Label: [2×1 double]
sv_indices: [36×1 double]
ProbA: []              ProbB: []
nSV: [2×1 double]
sv_coef: [36×1 double]
SVs: [36×2 double]
```

Finding the Decision Function

libsvm

- $\mathbf{w}$: maybe infinite variables
- The dual problem

$$\begin{aligned} \min_{\boldsymbol{\alpha}} \quad & \frac{1}{2}\boldsymbol{\alpha}^T Q \boldsymbol{\alpha} - \mathbf{e}^T \boldsymbol{\alpha} \\ \text{subject to} \quad & 0 \le \alpha_i \le C, i = 1, \ldots, l \\ & \mathbf{y}^T \boldsymbol{\alpha} = 0, \end{aligned}$$

where $Q_{ij} = y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ and $\mathbf{e} = [1, \ldots, 1]^T$

Corresponds to (Bishop 7.32) With y=t

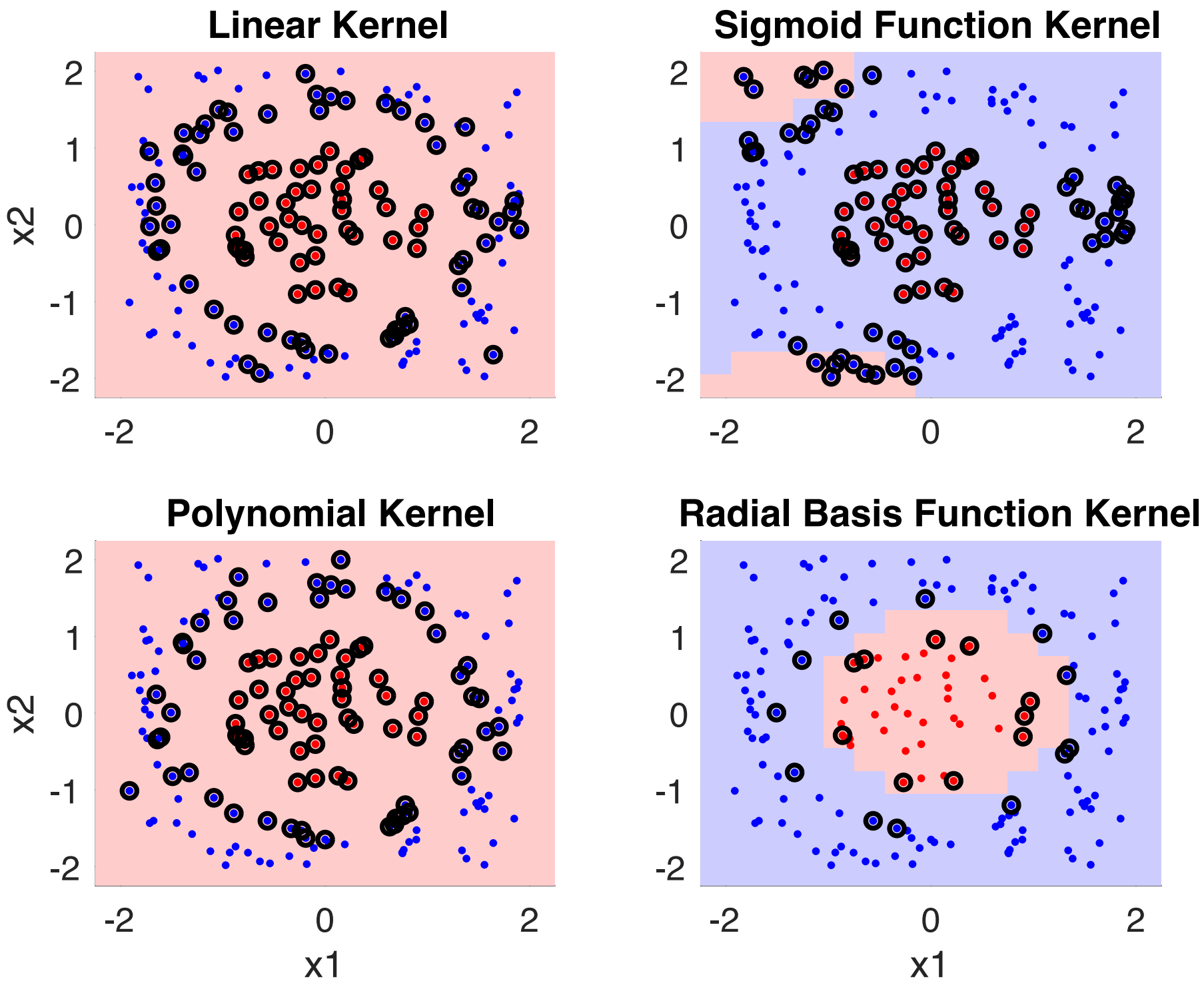
- At optimum

$$\mathbf{w} = \sum_{i=1}^{l} \alpha_i y_i \phi(\mathbf{x}_i)$$

- A finite problem: #variables = #training data

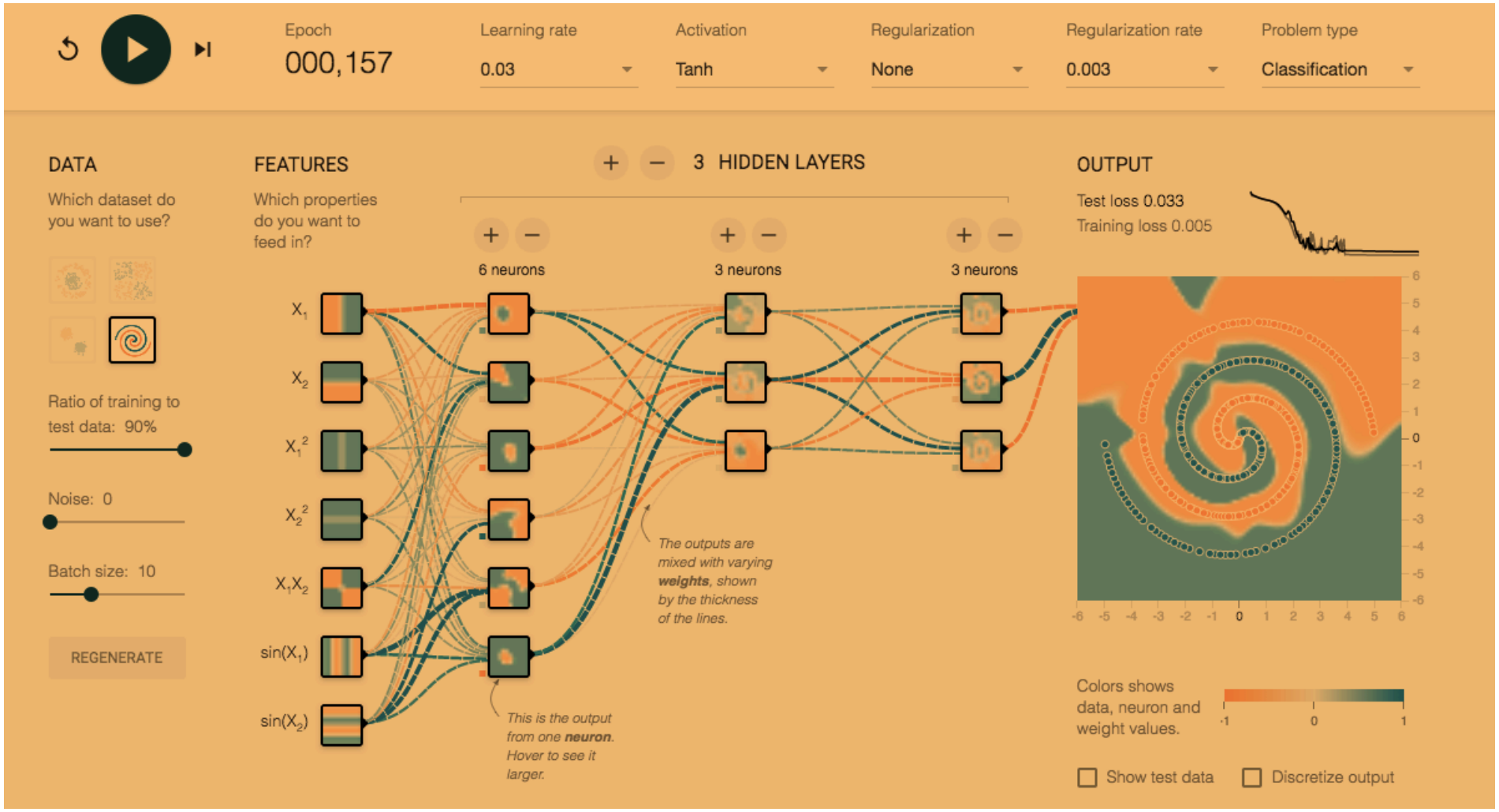
Using these results to eliminate $\mathbf{w}$, b, and $\{\xi_n\}$ from the Lagrangian, we obtain the dual Lagrangian in the form

$$\widetilde{L}(\mathbf{a}) = \sum_{n=1}^{N} a_n - \frac{1}{2} \sum_{n=1}^{N} \sum_{m=1}^{N} a_n a_m t_n t_m k(\mathbf{x}_n, \mathbf{x}_m) \qquad (7.32)$$


Linear Kernel
Sigmoid Function Kernel
Polynomial Kernel
Radial Basis Function Kernel
x2
x1
2
1
0
-1
-2

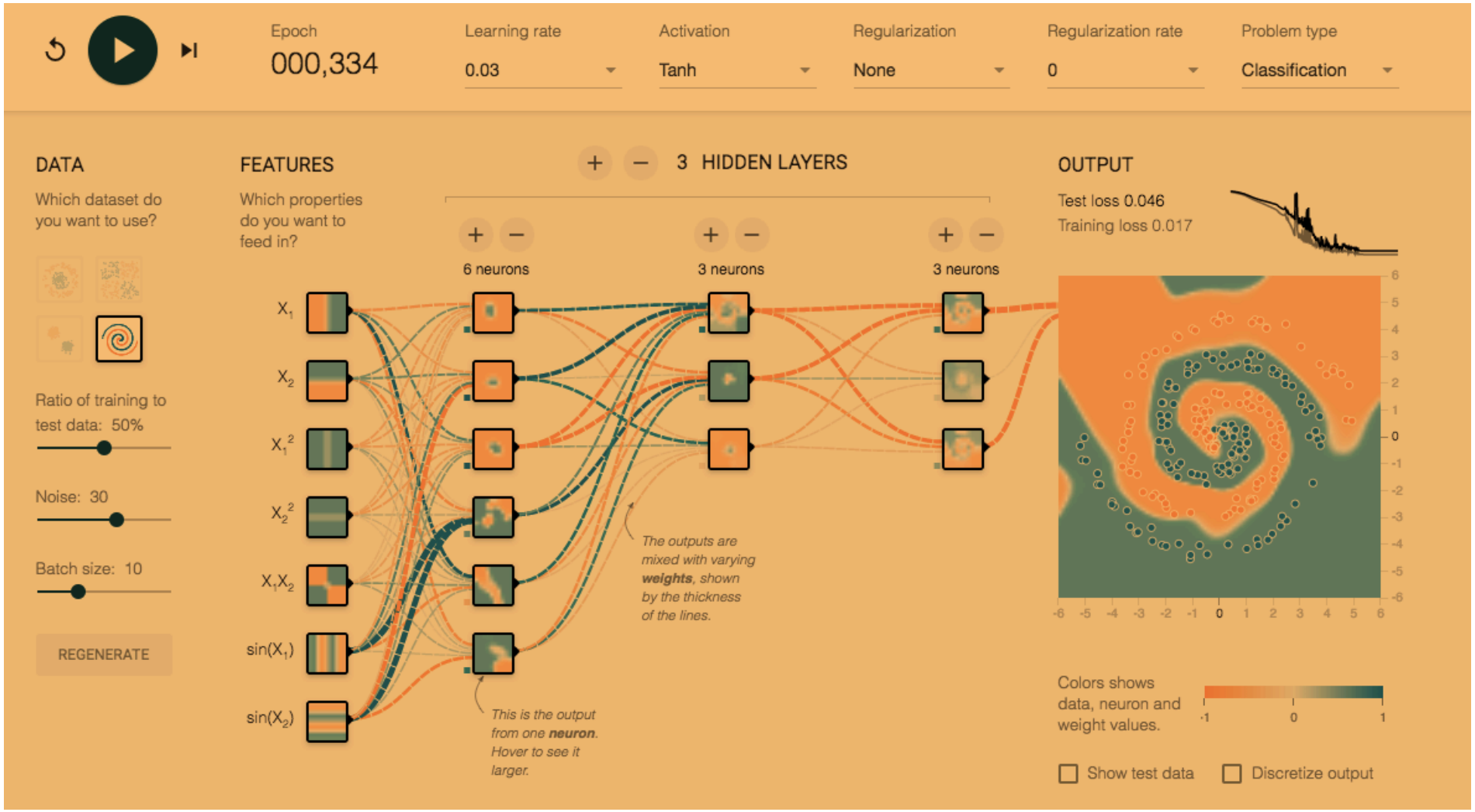
Tensorflow Playground

1. Fitting the spiral with default settings fail due to the small training set. The NN will fit to the training data which is not representative of the true pattern and the network will **generalize** poorly. Increasing the ratio of training to test data to 90% the NN finds the correct shape (1st image).



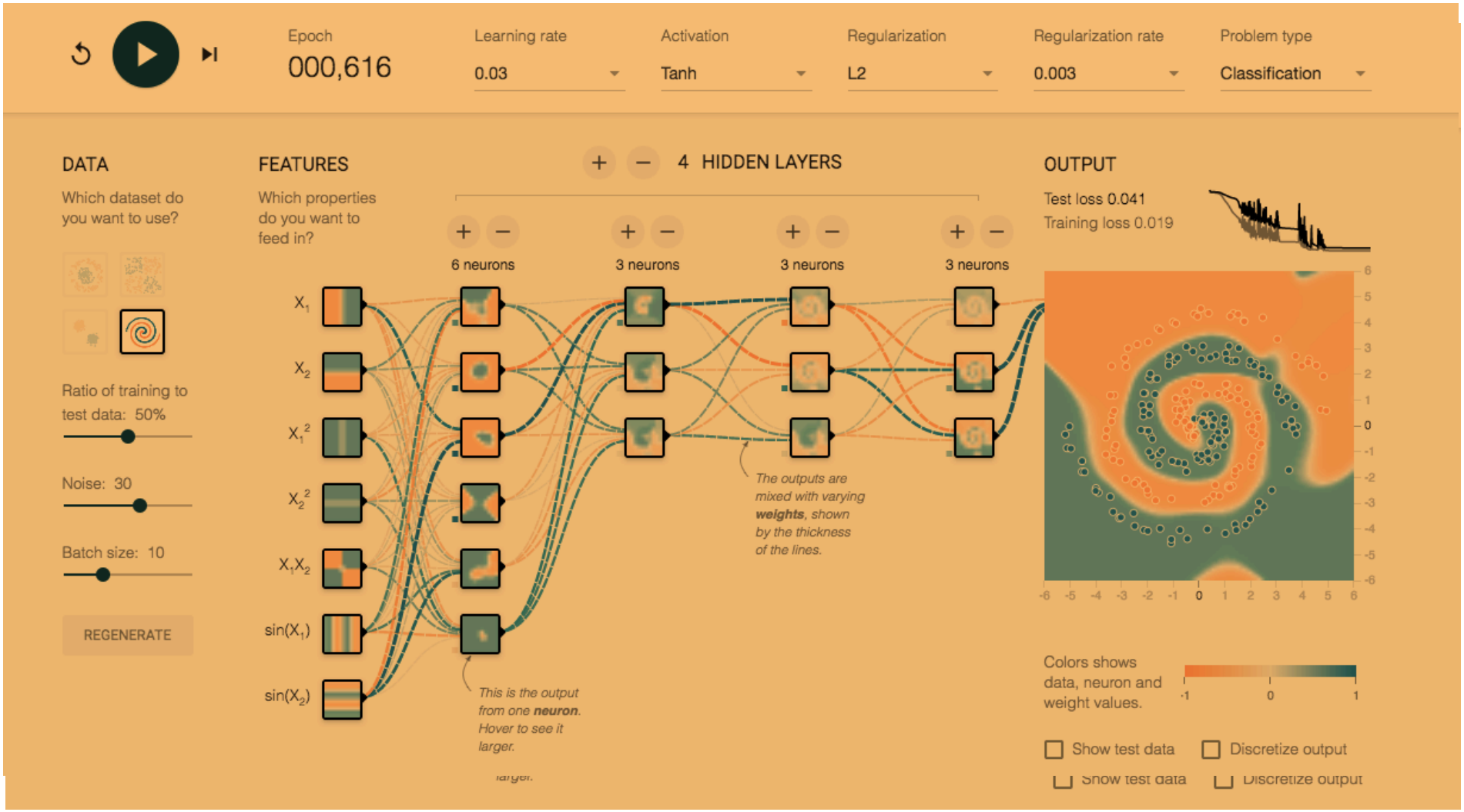
Tensorflow Playground

You can fix the generalization problem by adding **noise** to the data. This allows the small training set to generalize better as it reduce **overfitting** of the training data (2nd image).



Tensorflow Playground

Adding an additional hidden layer the NN fails to classify the shape properly. **Overfitting** once again becomes a problem even after you've added noise. This can be fixed by adding appropriate **L2 regularization** (third image).



- **NOT USED**

Introducing slack variables

- Slack variables are non-negative. When greater than zero they "cheat" by putting the plane closer to the datapoint than the margin. We minimize the amount of cheating by picking a value for lamba.

$$\mathbf{w}.\mathbf{x}^c + b \geq +1 - \xi^c \quad for\ positive\ cases$$

$$\mathbf{w}.\mathbf{x}^c + b \leq -1 + \xi^c \quad for\ negative\ cases$$

$$with\ \ \xi^c \geq 0 \quad for\ all\ c$$

$$and\ \ \frac{\|\mathbf{w}\|^2}{2} + \lambda \sum_c \xi^c \quad as\ small\ as\ possible$$

The classification rule

- The classification rule is simple:

$$bias + \sum_{s \varepsilon SV} w_s K(x^{test}, x^s) \quad > \quad 0$$

The set of support vectors

- The cleverness is in selecting the support vectors maximizing the margin and computing the weight for each support vector.
- Need choosing a good kernel function and maybe choosing a lambda for non-separable cases.

Training a linear SVM

- To find the maximum margin separator, solve the optimization problem:

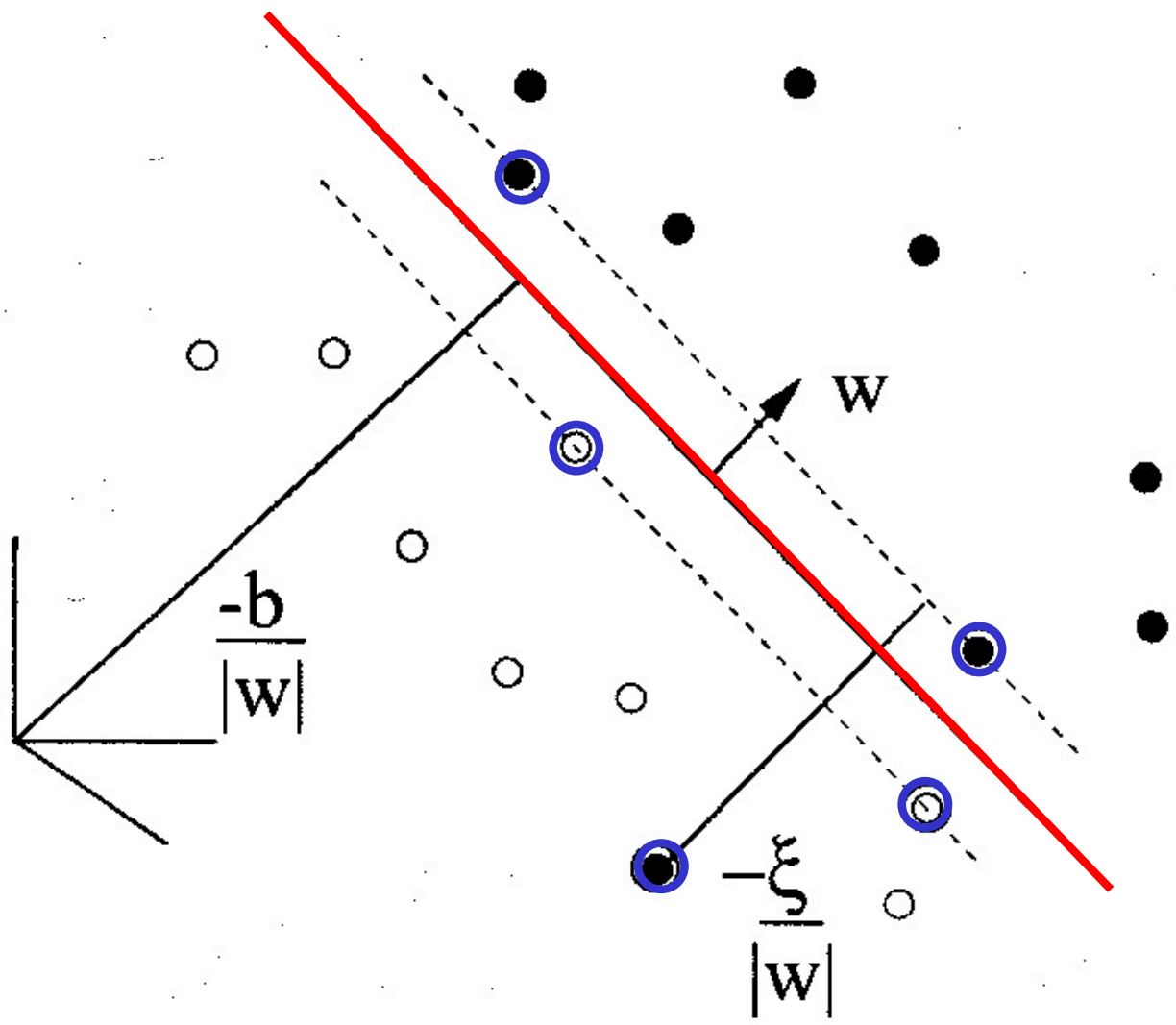
$$\mathbf{w}.\mathbf{x}^c + b > +1 \quad for\ positive\ cases$$

$$\mathbf{w}.\mathbf{x}^c + b < -1 \quad for\ negative\ cases$$

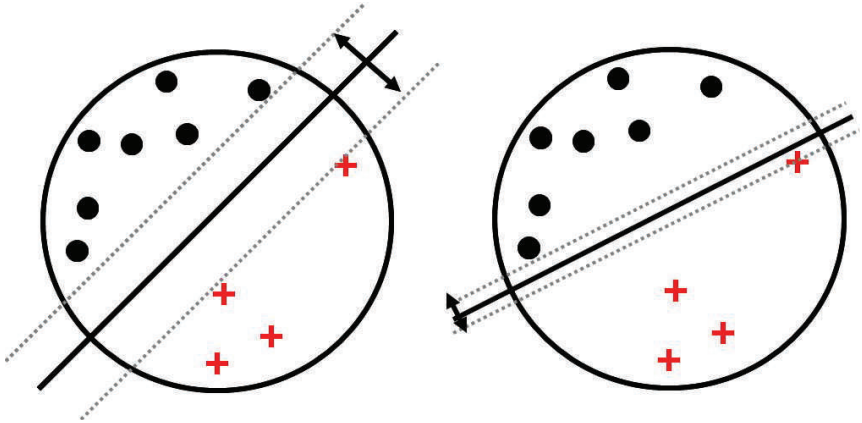
$$and \quad \| \mathbf{w} \|^2 \ is\ as\ small\ as\ possible$$

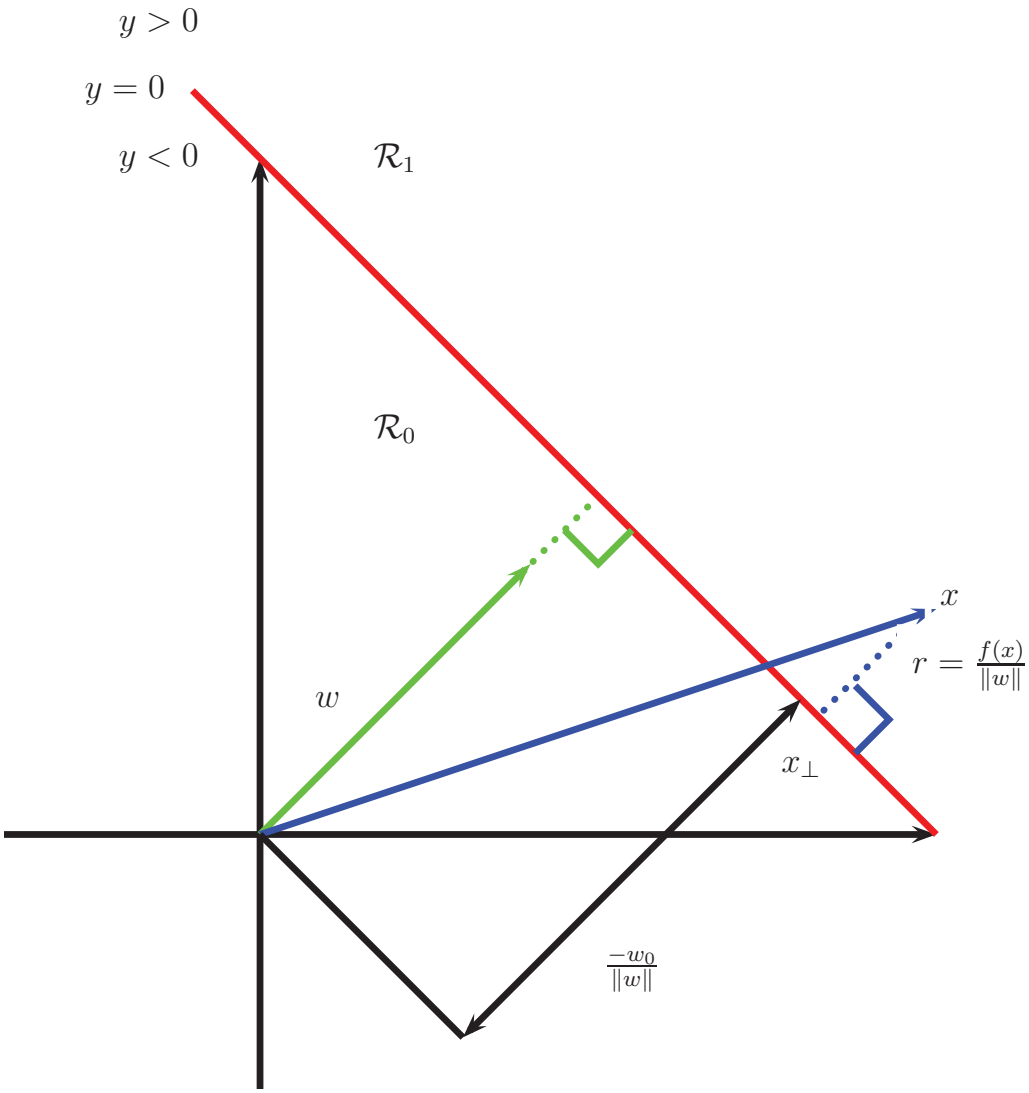
- It's a convex problem. There is one optimum and we can find it without fiddling with *learning rates or weight decay or early stopping.*
  - Don't worry about the optimization problem. It has been solved. Its called quadratic programming.

A picture of the best plane with a slack variable



Large margin



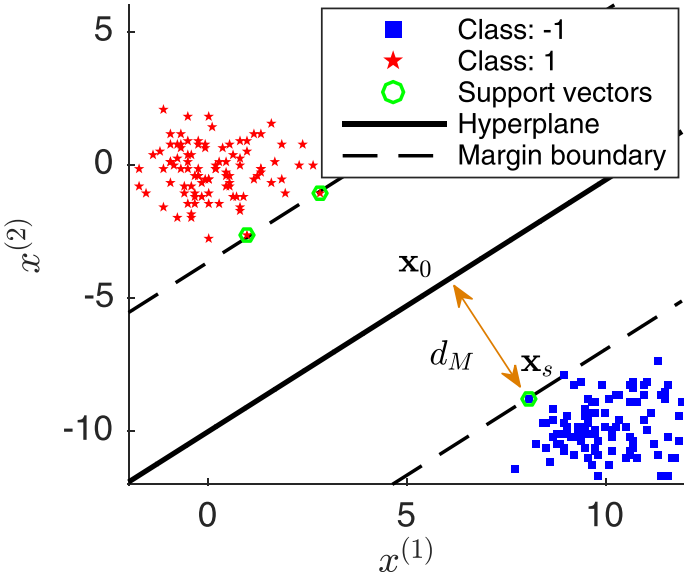


Support Vector machines (SVM)

For points $\mathbf{x}_n$ and $\mathbf{x}_o$ on separating line

$$\mathbf{x}_n = \mathbf{x}_0 + d\frac{\mathbf{w}}{||\mathbf{w}||},$$

$$\mathbf{w}^T\mathbf{x}_0 + b = 0,$$



Thus distance is given by $d(\mathbf{x}_n) = s_n\frac{\mathbf{w}^T\mathbf{x}_n + b}{\|\mathbf{w}\|},$

For all point we maximize the margin

$$\underset{\mathbf{w},b}{\text{argmax}} \quad d_M,$$

$$\text{subject to } \frac{s_n(\mathbf{w}^T\mathbf{x}_n + b)}{\|\mathbf{w}\|} \geq d_M, \quad n = 1, \ldots, N.$$

For non-linear relations $\mathrm{y}_n = \mathbf{w}^T\phi(\mathbf{x}_n) + b,$

We can formulate it in terms of kernel functions, say Gaussian

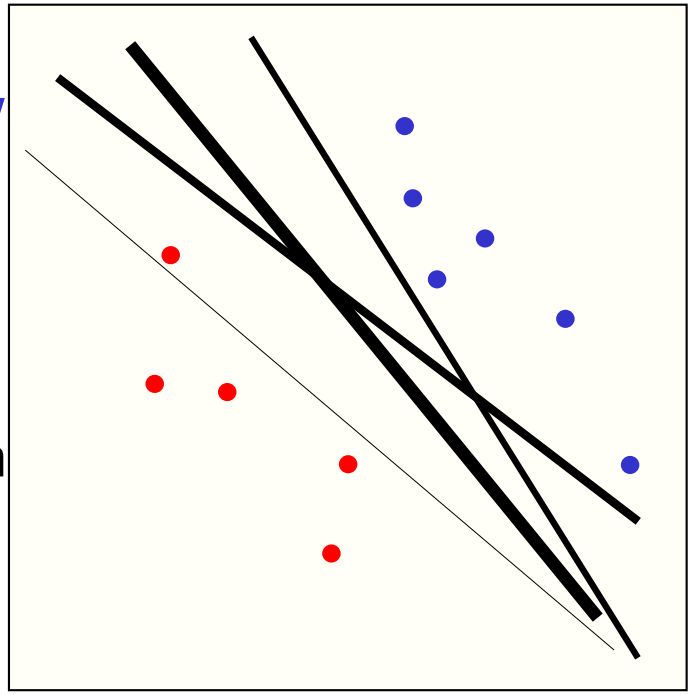
$$k_\phi(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^T\phi(\mathbf{x}_m) \qquad k_\phi(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$$

Preventing overfitting when using big sets of features

- Suppose we use a big set of features to ensure that two classes are linearly separable. What is the best separating line?

- The Bayesian answer is using them all (including ones that do not separate the data.)
- Weight each line by its posterior probability (how well it fits data and prior).
- Is there an efficient way to approximate the Bayesian answer?

- **A Bayesian Interpretation**: Using the maximum margin separator often gives a pretty good approximation to using all separators weighted by their posterior probabilities.



A potential problem and a magic solution

- Mapping input vectors into a very high-D feature space, surely finding the maximum-margin separator is computationally intractable?
  - The mathematics is all **linear**, but the vectors have a huge number of components.
  - Taking the scalar product of two vectors is expensive.
- The way to keep things tractable is "the kernel trick"
- The kernel trick makes your brain hurt when you first learn about it, but it is actually simple.

Preprocessing the input vectors

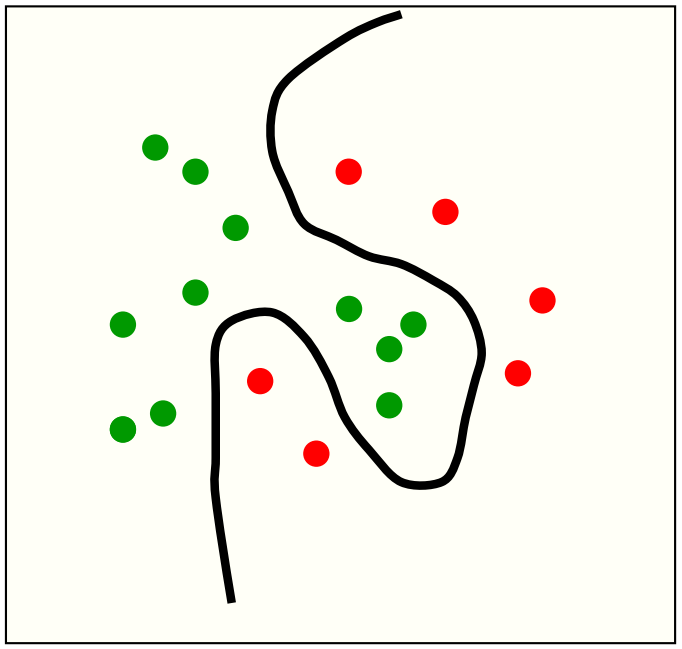
- Instead predicting the answer directly from the raw inputs we could start by extracting a layer of "features".
  - Sensible if certain combinations of input values would be useful (e.g. edges or corners in an image).
- Instead of learning the features we could design them by hand.
  - The hand-coded features are equivalent to a layer of non-linear neurons with no need to be learned.
  - Using a big set of features for a two-class problem, the classes will almost certainly be linearly separable.
    - But surely the linear separator gives poor generalization.

What the kernel trick achieves

- Finding the maximum-margin separator is expressed as scalar products between pairs of datapoints (in high-D feature space).
- These scalar products are the only part of the computation that depends on the dimensionality of the high-D space.
  - We need a fast way to do the scalar products to solve the learning problem in the high-D space.
- The kernel trick is a magic way of doing scalar products.
  - It relies on mapping to the high-D feature space that allows fast scalar products.

How to make a plane curved

- Fitting hyperplanes as separators is mathematically easy.
  - The mathematics is **linear.**
- Replacing the raw input variables with a much larger set of features we get a nice property:
  - A planar separator in high-D feature space is a curved separator in the low-D input space.

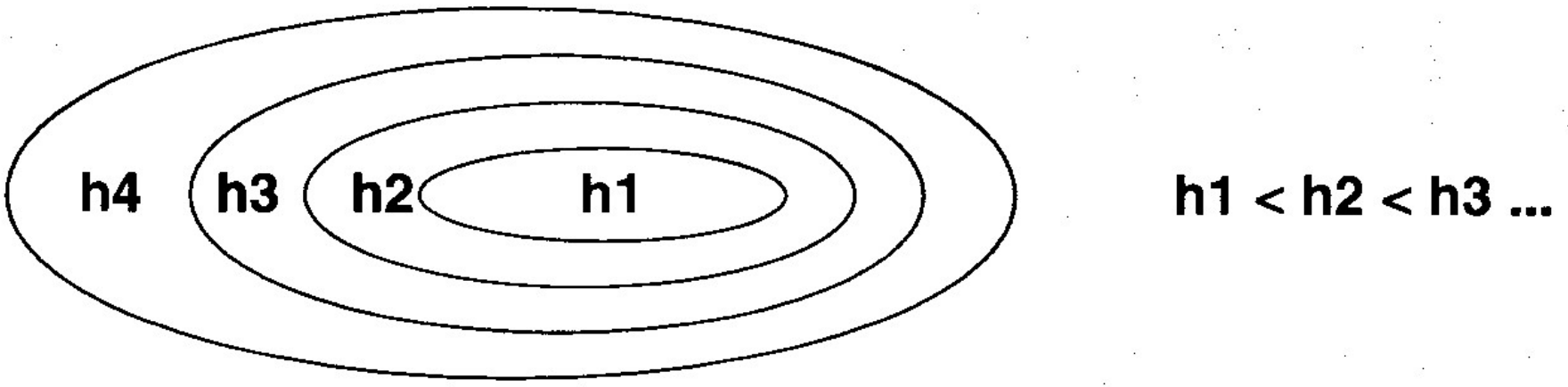


A planar separator in a 20-D feature space projected back to the original 2-D space

Is preprocessing cheating?

- Its cheating if using carefully designed set of task-specific, hand-coded features and claim that the learning algorithm solved the whole problem.
  - The really hard bit is designing the features.
- Its not cheating if we learn the non-linear preprocessing.
  - This makes learning more difficult and more interesting (e.g. backpropagation after pre-training)
- Its not cheating if we use a very big set of non-linear features that is task-independent.
  - Support Vector Machines do this.
  - They prevent overfitting (first half of lecture)
  - They use a huge number of features without requiring as much computation as seems to be necessary (second half).

A hierarchy of model classes



- Some model classes can be arranged in a hierarchy of increasing complexity.
- How to pick the best level in the hierarchy for modeling a given dataset?

A way to choose a model class

- A low error rate on unseen data.
  - This is called “structural risk minimization”
- A guarantee of the following form is helpful:

  Test error rate =< train error rate + f(N, h, p)

  Where N = size of training set,

  h = measure of the model complexity,

  p = the probability that this bound fails

  We need p to allow for really unlucky test sets.
- Then we choose the model complexity that minimizes the bound on the test error rate.

The story so far

- Using a large set of non-adaptive features, we might make the two classes linearly separable.
  - But just fitting any separating plane, it will not generalize well to new cases.
- Fitting the separating plane maximizing the margin (minimum distance to any data points), gives better generalization.
  - Intuitively, maximizing the margin squeezes the surplus capacity that came from using a high-dimensional feature space.
- This is justified by a lot of clever mathematics which shows that
  - large margin separators have lower VC dimension.
  - models with lower VC dimension have a smaller gap between training and test error rates.

Dealing with the test data

- Choosing a high-D mapping for which the kernel trick works, we do not use much CPU time for the high-D when finding the best hyper-plane.
  - We cannot express the hyperplane by using its normal vector in the high-D space because this vector is huge.
  - Luckily, we express it in terms of the support vectors.
- What about the test data. We cannot compute the scalar product $\mathbf{w} \,.\, \phi(\mathbf{x})$ because its in the high-D space.

- Deciding which side of the separating hyperplane a test point lies on, requires a scalar product $\mathbf{w} \,.\, \phi(\mathbf{x})$
- We express this scalar product as a weighted average of scalar products using stored support vectors
  - Could be slow many support vectors.

Performance

- SVM work very well in practice.
  - The user must choose the kernel function and its parameters, but the rest is automatic.
  - The test performance is very good.
- They can be expensive in time and space for big datasets
  - The computation of the maximum-margin hyper-plane depends on the square of number of training cases.
  - Need storing all the support vectors.
- SVM's are good if you have no idea about what structure to impose.
- The kernel trick can also be used for PCA in a high-D space, thus giving a non-linear PCA in the original space.