Kernels

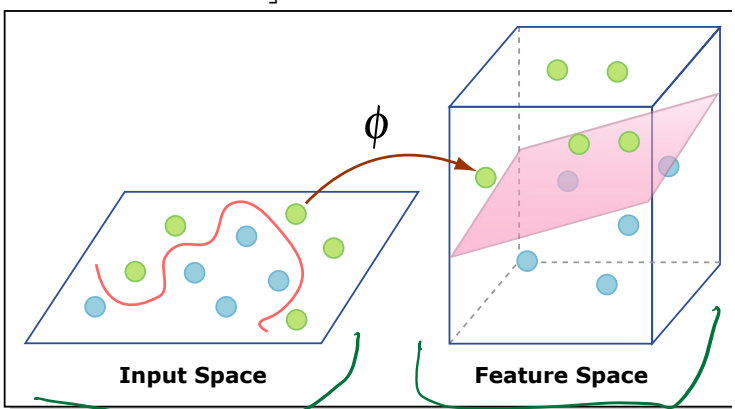
We might want to consider something more complicated than a linear model:

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

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



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.

Dual representation, Sec 6.2

Primal problem: $\min_{\boldsymbol{w}} E(\boldsymbol{w})$ $\quad \boldsymbol{w} \in R^M$

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

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

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

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

Dual representation is : $\min_{\boldsymbol{a}} E(\boldsymbol{a})$ $\quad \boldsymbol{a} \in R^N$

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

Prediction

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

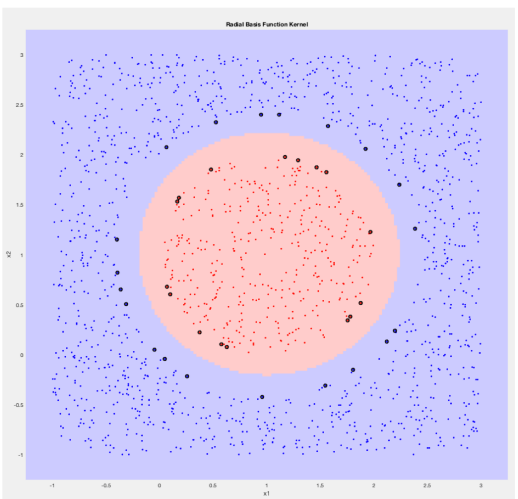
Dual representation, Sec 6.2

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

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

Can be inner product in infinite dimensional space

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

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

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

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

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

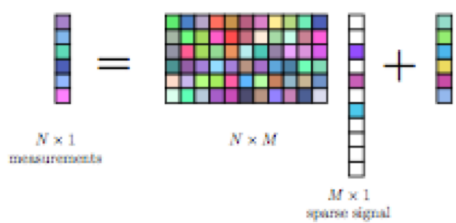
where

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

AND RVM

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.

Lecture 10
Support Vector Machines

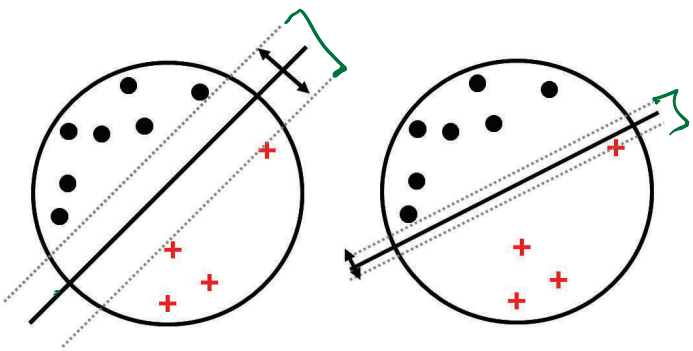
Non Bayesian!

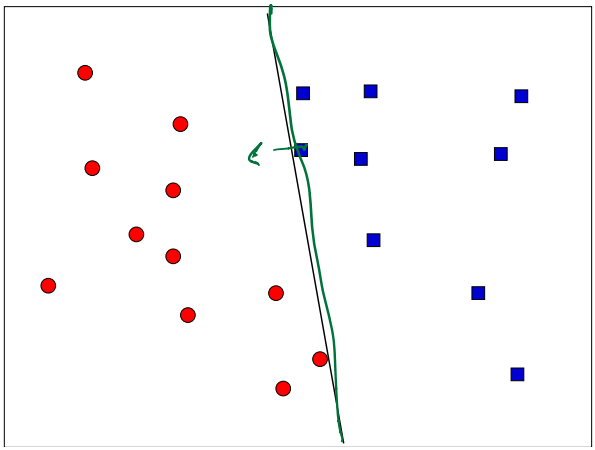
Features:

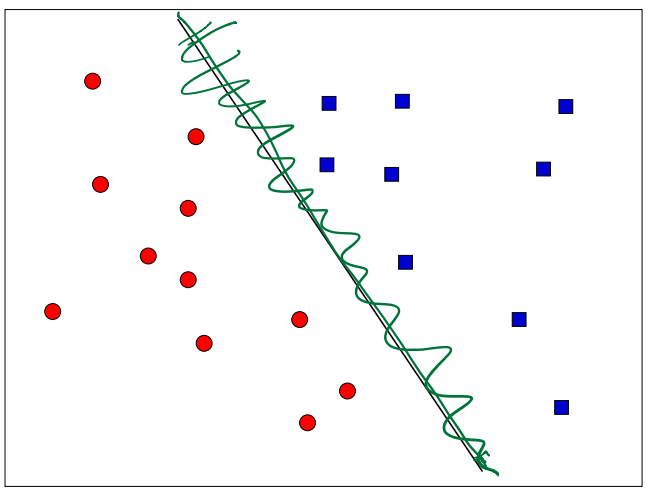
- Kernel
- Sparse representations
- Large margins

Regularize for plausibility

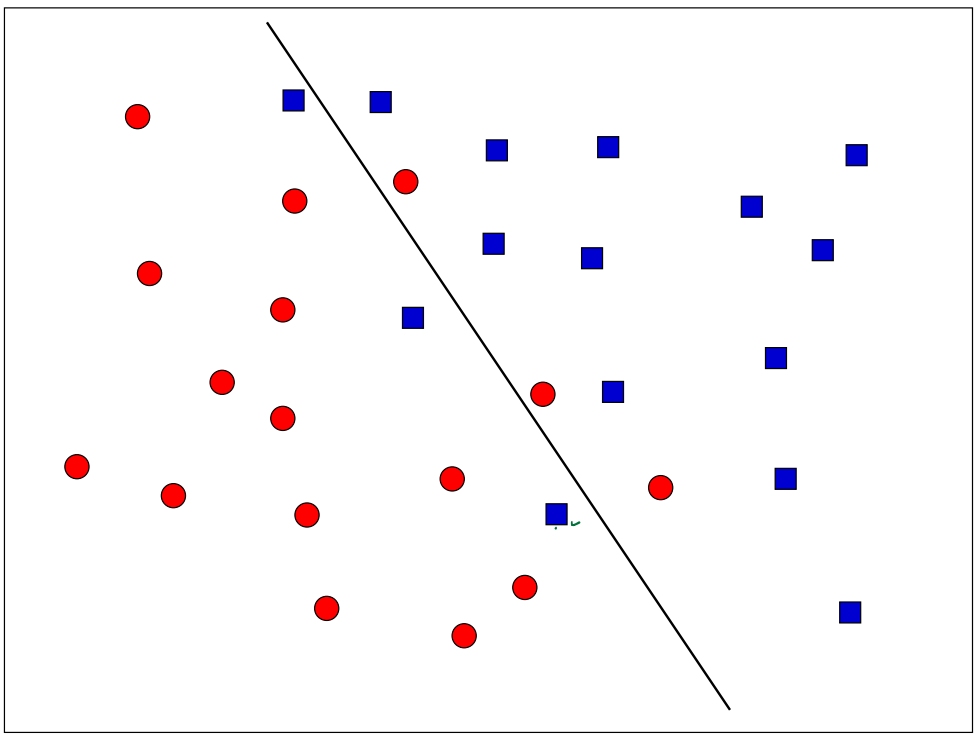
- Which one is best?
- We maximize the margin





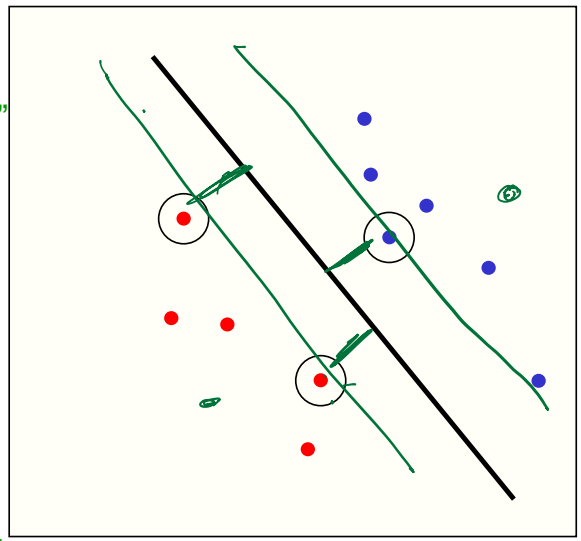


Regularize for plausibility



Support Vector Machines

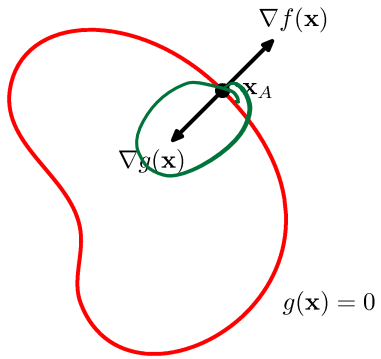
- The line that maximizes the minimum margin is a good bet.
  - The model class of "hyper-planes with a margin *m*" has a low VC dimension if *m* is big.
- This maximum-margin separator is determined by a subset of the datapoints.
  - Datapoints in this subset are called "support vectors".
  - It is 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)

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



Taylor expansion

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

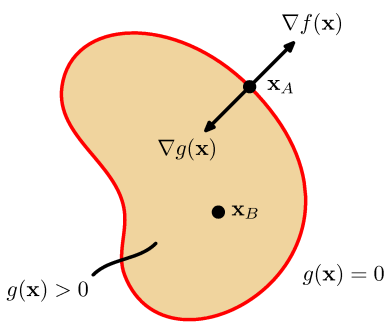
$= 0 + \varepsilon^T \nabla g(x) = 0$

$\nabla f(x) + \lambda \nabla g(x) = 0$

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

$\nabla_x L(x, \lambda) = 0$

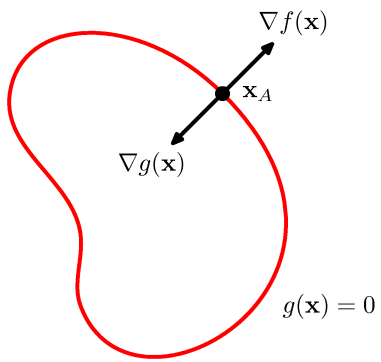
$\frac{dL}{d\lambda} = 0 \Rightarrow g(x) = 0$



Lagrange multiplier (Bishop App E)

$\max\big(f(\boldsymbol{x})\big)$ subject to $g(\boldsymbol{x}) \geq 0$

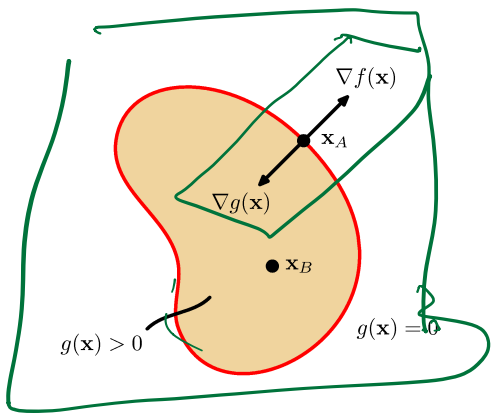
$L(\boldsymbol{x}, \lambda) = f(\boldsymbol{x}) + \lambda g(\boldsymbol{x})$



Either $\nabla f(\boldsymbol{x}) = \mathbf{0}$

Then $g(\boldsymbol{x})$ is inactive, $\lambda = 0$

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



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

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

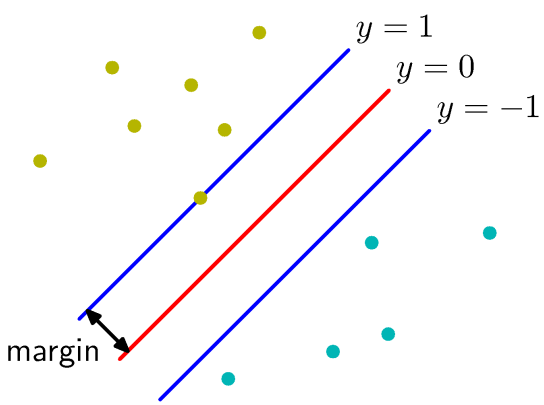
Testing a linear SVM

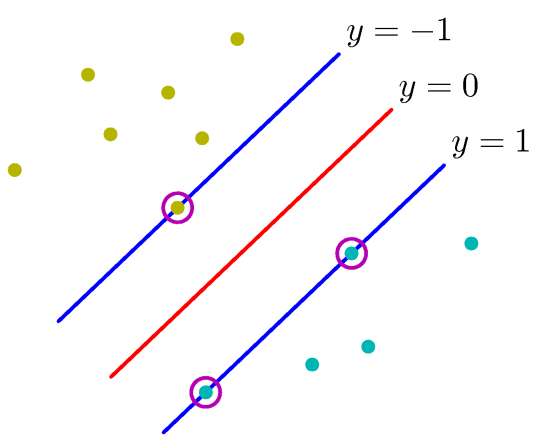
- The separator is defined as the set of points for which:

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

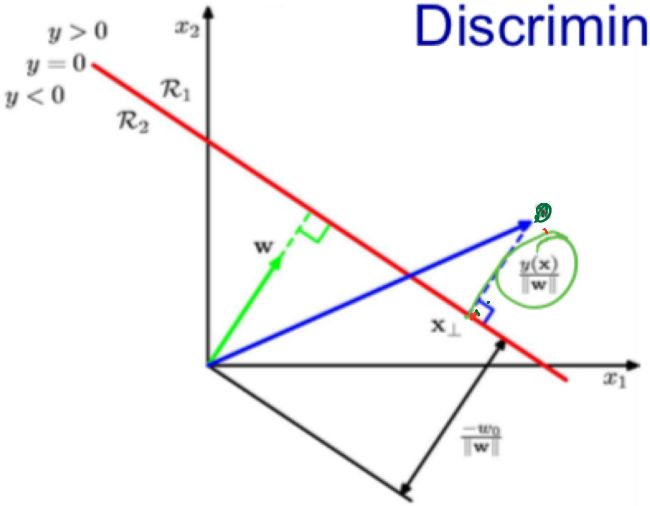
$$so\ if\ \ \mathbf{w}.\mathbf{x}^c + b > 0\ \ say\ its\ a\ positive\ case$$

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





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

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

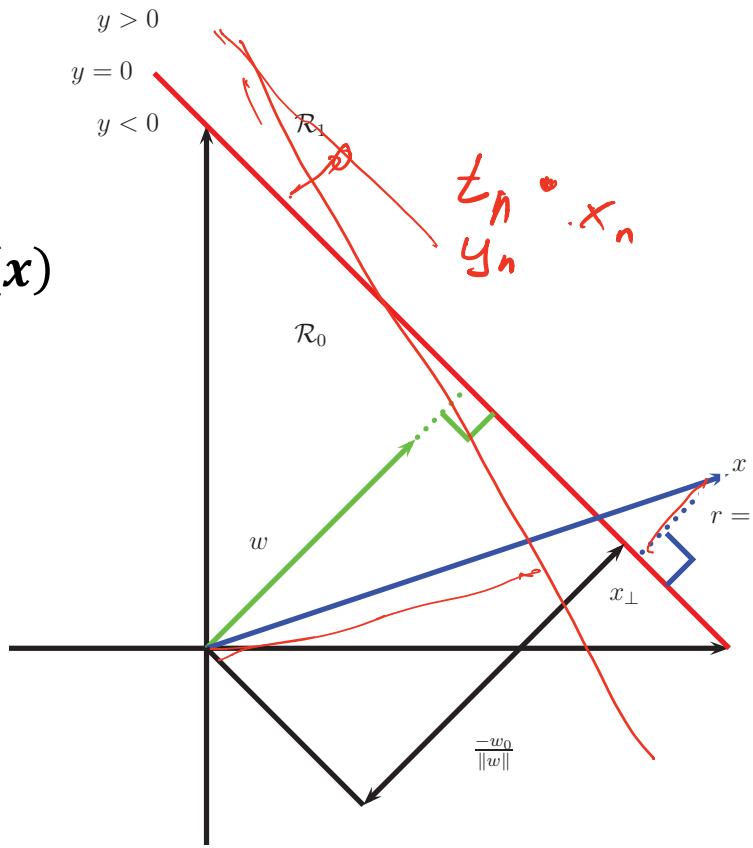
$$\boldsymbol{x}_n = \boldsymbol{x}_\perp + r_n \frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$$

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

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

$$t_n y_n \geq 1 \quad \forall n$$

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



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

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

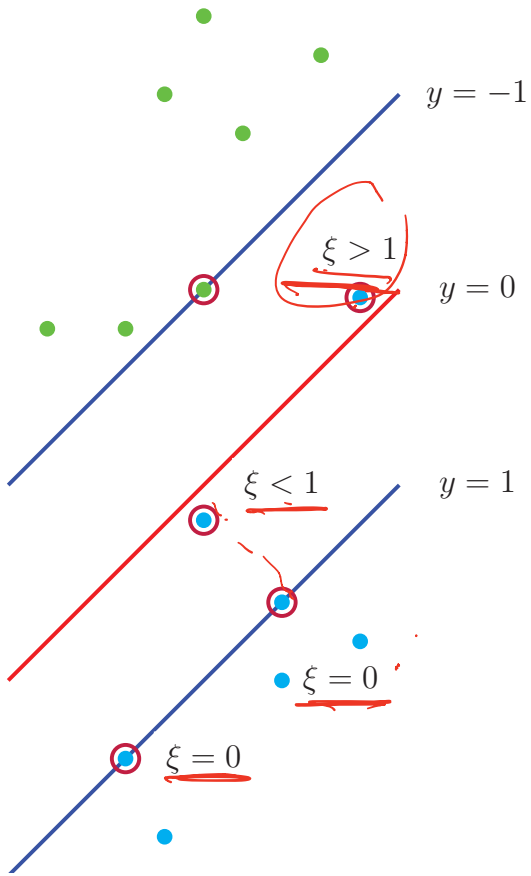
If there is no separating plane…

- 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

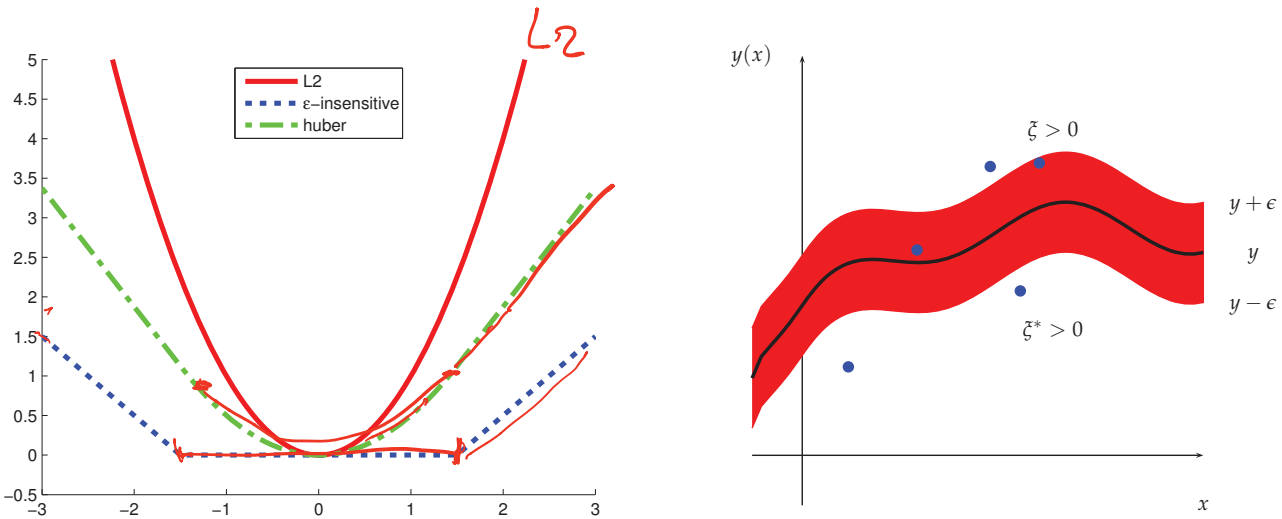
$$\hat{\boldsymbol{w}} = \sum_{n \in S} a_n \, t_n \boldsymbol{\varphi}(xn) \qquad \text{(Bishop 7.8)}$$

- 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.

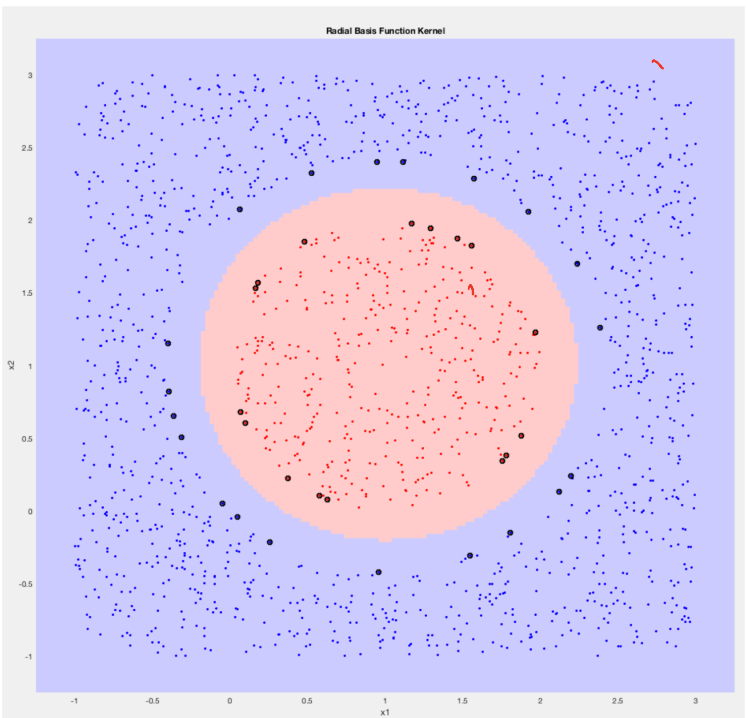
- SVM DOES:
  - Margin maximization
  - Kernel trick
  - Sparse

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

- **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

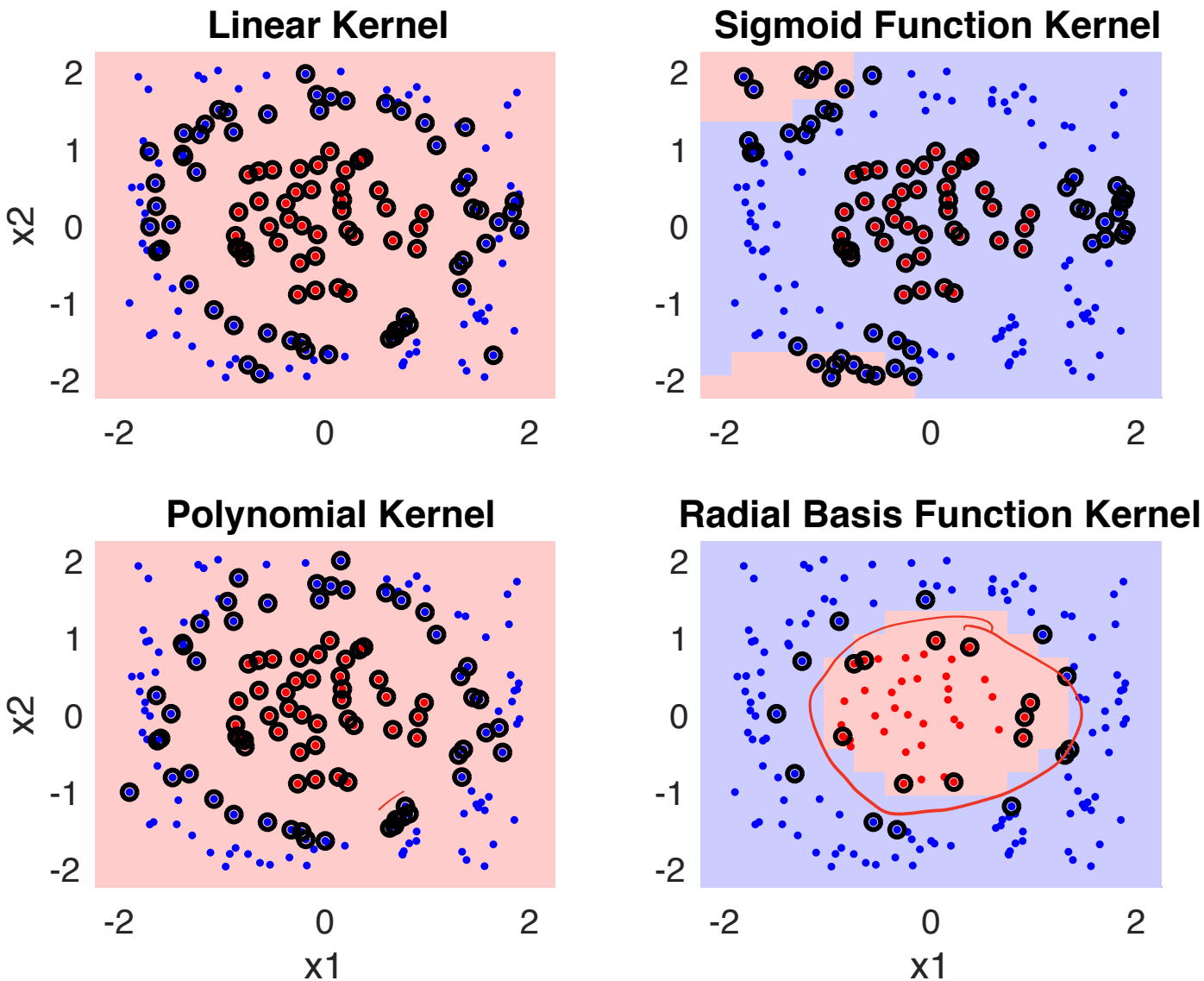
- At optimum

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

- A finite problem: #variables = #training data

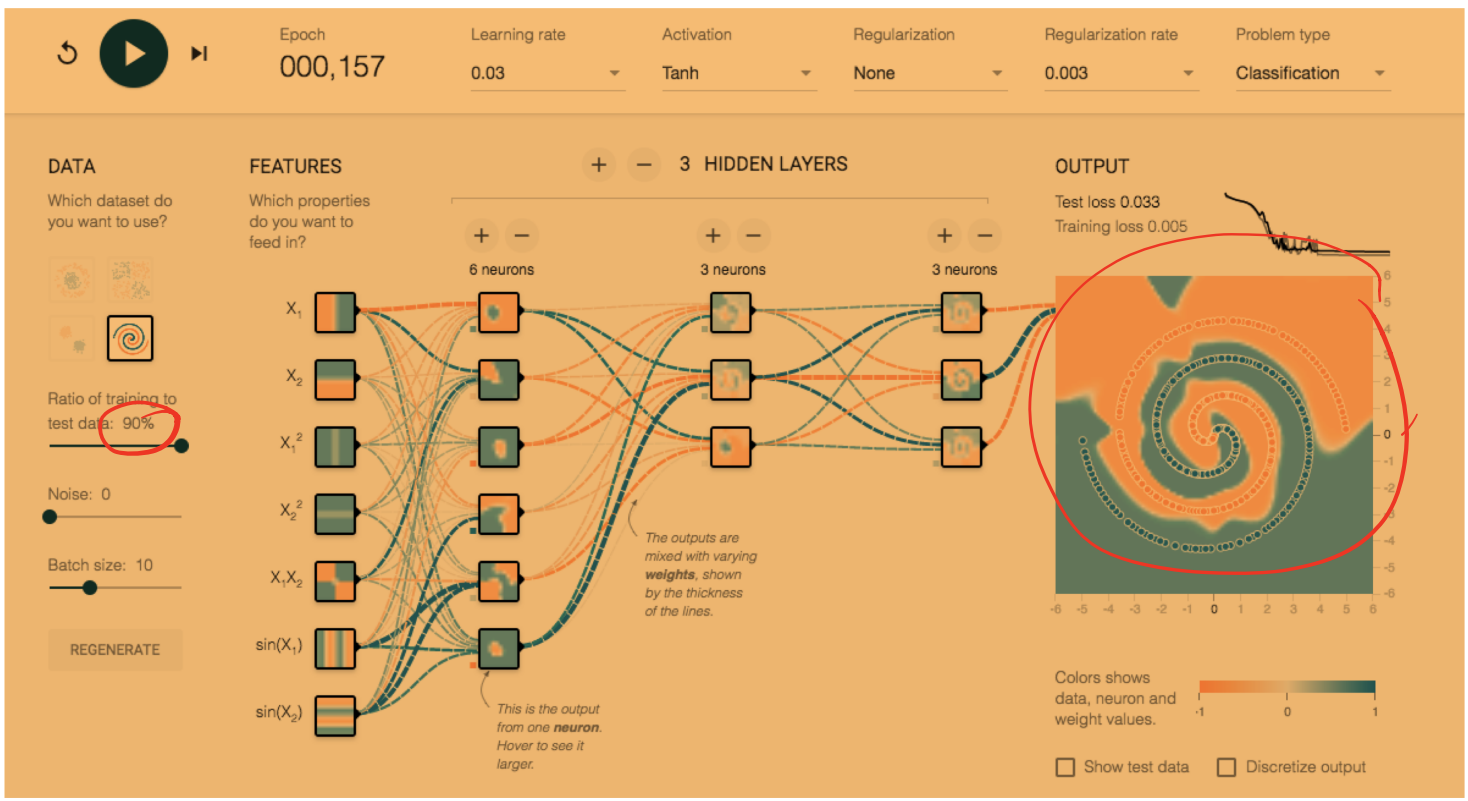
Using these results to eliminate $\mathbf{w}$, b, and $\{\xi_n\}$ from the Lagrangian, we obtain the dual Lagrangian in the form

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


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

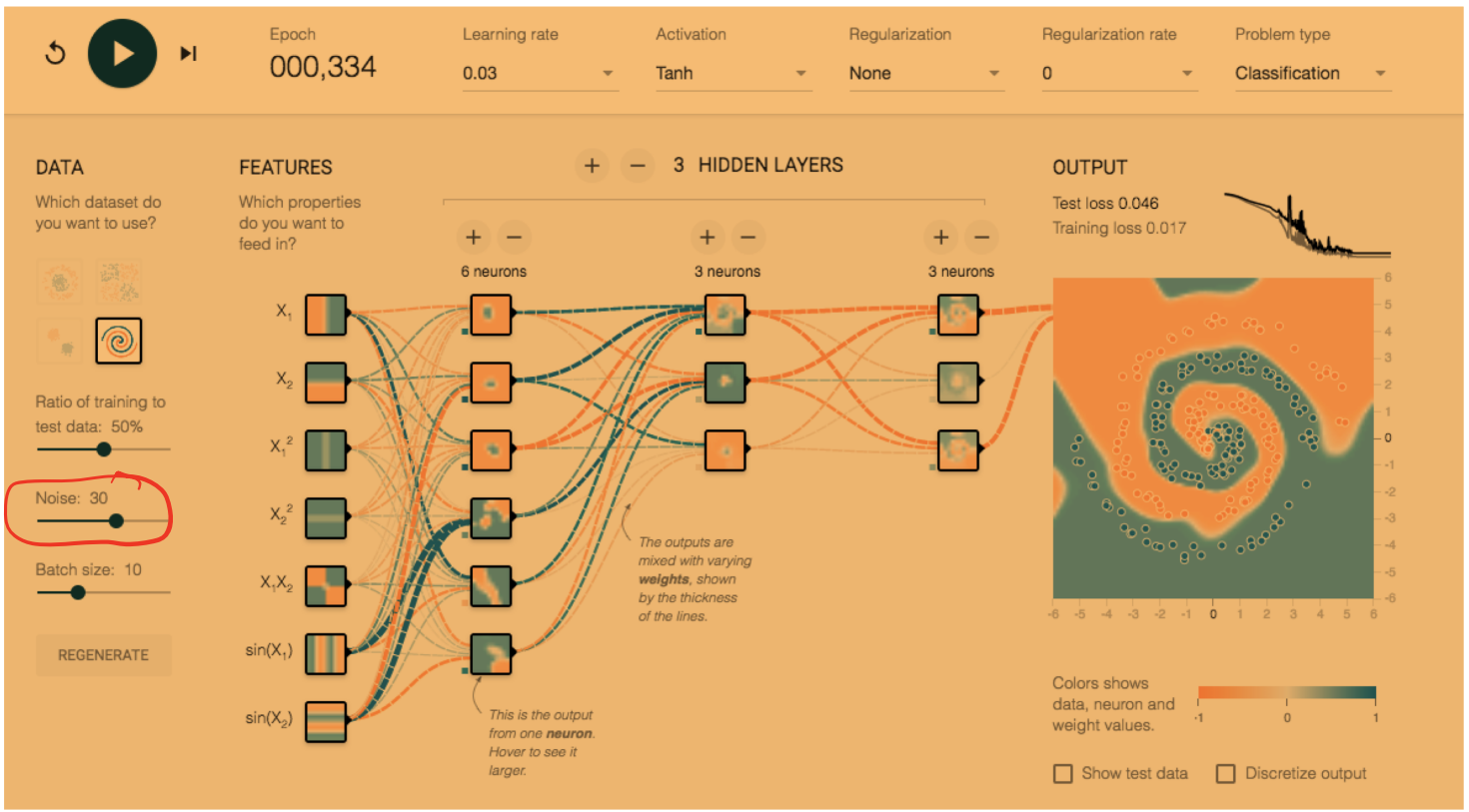
Tensorflow Playground

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).