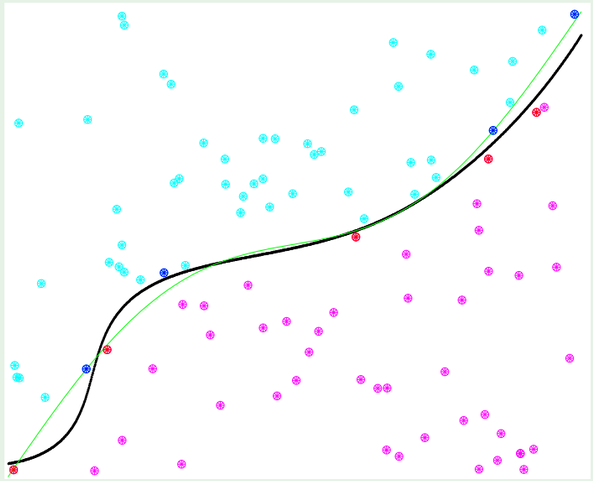
Review of Lecture 15

- Kernel methods

$K(\mathbf{x}, \mathbf{x}') = \mathbf{z}^\mathsf{T}\mathbf{z}'$ for some $\mathcal{Z}$ space

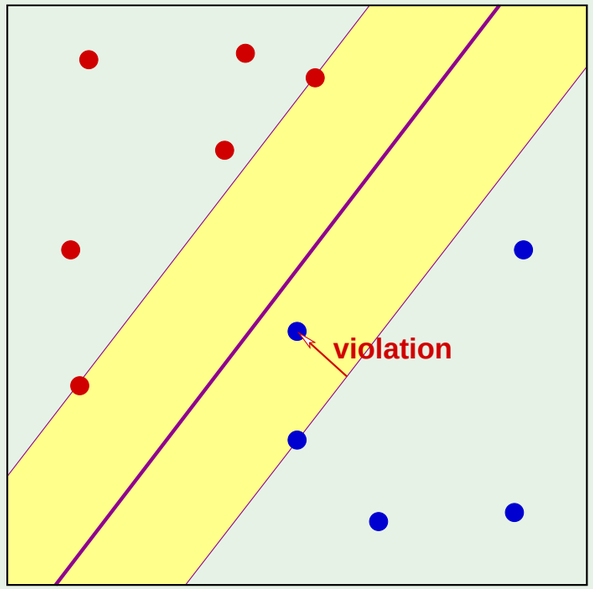


$\uparrow$

$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \left\|\mathbf{x} - \mathbf{x}'\right\|^2\right)$$

- Soft-margin SVM

$$\text{Minimize} \quad \frac{1}{2}\mathbf{w}^\mathsf{T}\mathbf{w} + C\sum_{n=1}^{N}\xi_n$$



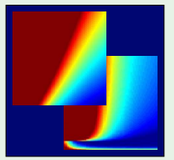
Same as hard margin, but $0 \leq \alpha_n \leq C$

Learning From Data

Yaser S. Abu-Mostafa
California Institute of Technology

Lecture 16: Radial Basis Functions

Sponsored by Caltech's Provost Office, E&AS Division, and IST • Thursday, May 24, 2012



Outline

- RBF and nearest neighbors
- RBF and neural networks
- RBF and kernel methods
- RBF and regularization

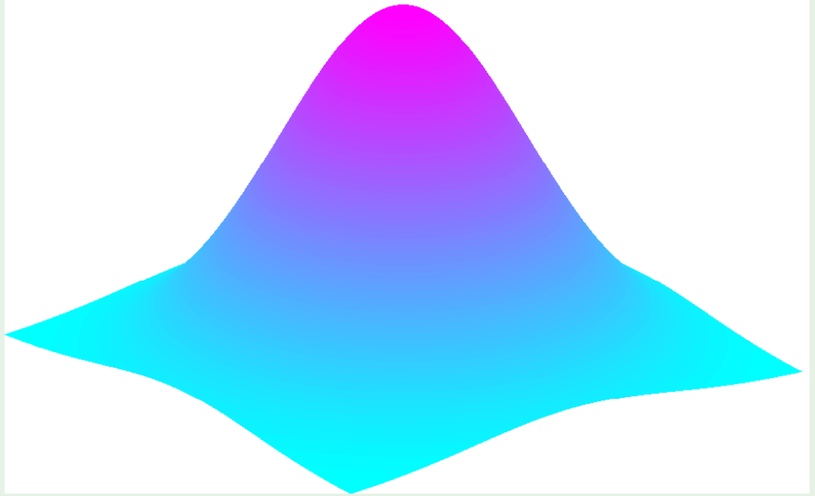
Basic RBF model

Each $(\mathbf{x}_n, y_n) \in \mathcal{D}$ influences $h(\mathbf{x})$ based on $\underbrace{\|\mathbf{x} - \mathbf{x}_n\|}_{\text{radial}}$

Standard form:

$$h(\mathbf{x}) = \sum_{n=1}^{N} w_n \underbrace{\exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2\right)}_{\text{basis function}}$$

The influence of each xn decays as a Gaussian



Each weight wn characterizes the magnitude of the influence of the point xn on the value of x. Note real valued output and real valued target output from training data, so the basic RBF is a regression model.

The learning algorithm

Finding $w_1, \cdots, w_N$:

$$h(\mathbf{x}) = \sum_{n=1}^{N} w_n \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2\right)$$

Given h consists of N terms with N parameters, where N is the number of training points, we should expect that we can achieve Ein=0

based on $\mathcal{D} = (\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$

$E_{\text{in}} = 0$: $h(\mathbf{x}_n) = y_n$ for $n = 1, \cdots, N$:

$$\sum_{m=1}^{N} w_m \exp\left(-\gamma \|\mathbf{x}_n - \mathbf{x}_m\|^2\right) = y_n$$

The solution

$$\sum_{m=1}^{N} w_m \exp\left(-\gamma \left\|\mathbf{x}_n - \mathbf{x}_m\right\|^2\right) = y_n \qquad N \text{ equations in } N \text{ unknowns}$$

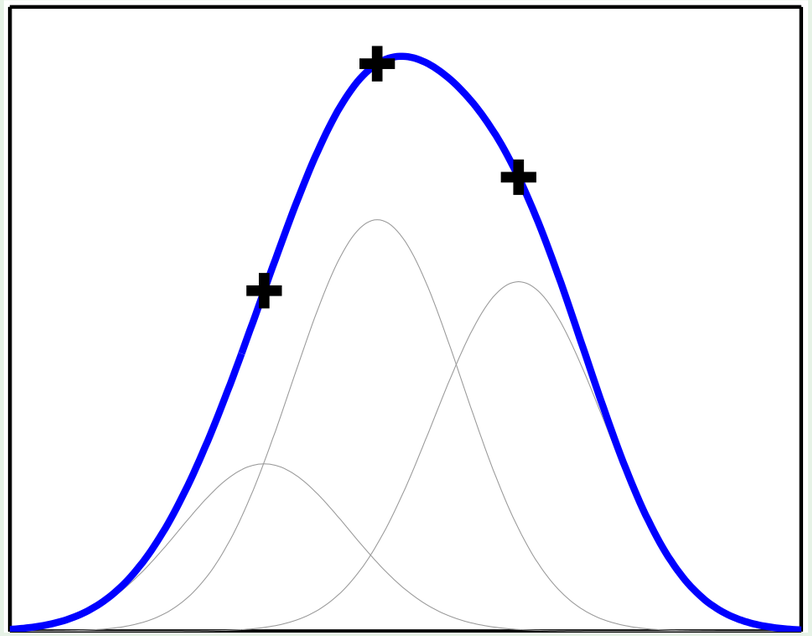
$$\underbrace{\begin{bmatrix} \exp(-\gamma \left\|\mathbf{x}_1 - \mathbf{x}_1\right\|^2) & \dots & \exp(-\gamma \left\|\mathbf{x}_1 - \mathbf{x}_N\right\|^2) \\ \exp(-\gamma \left\|\mathbf{x}_2 - \mathbf{x}_1\right\|^2) & \dots & \exp(-\gamma \left\|\mathbf{x}_2 - \mathbf{x}_N\right\|^2) \\ \vdots & \vdots & \vdots \\ \exp(-\gamma \left\|\mathbf{x}_N - \mathbf{x}_1\right\|^2) & \dots & \exp(-\gamma \left\|\mathbf{x}_N - \mathbf{x}_N\right\|^2) \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}}_{\mathbf{w}} = \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\mathbf{y}}$$

If Φ is invertible, $\boxed{\mathbf{w} = \Phi^{-1}\mathbf{y}}$ "exact interpolation"

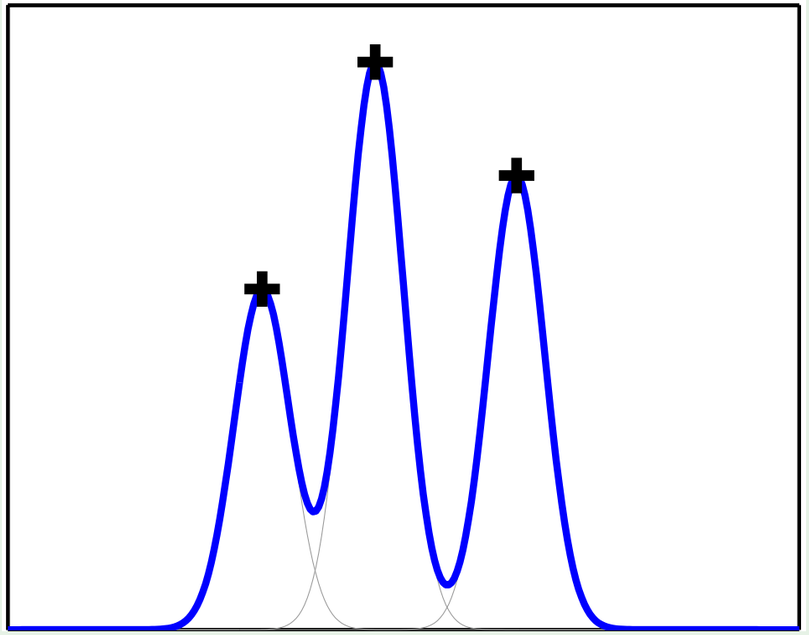
On the points we know the value yn (the training points), we get the value exactly. The kernel (here the Gaussian) interpolates between points to find the value for other x's.

The effect of γ

$$h(\mathbf{x}) = \sum_{n=1}^{N} w_n \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2\right)$$



small γ



large γ

For small gamma, the variance is large so the curve is wide. Depending on how sparse the points are, the interpolation will depend on how fast the drop off is for the Gaussians for each point. The actual value of a 'small' gamma is relative, depending on the distance between the points in the space. In general, it is a good idea to have the width of the Gaussian comparable to the distance between the points so there is a genuine interpolation. As is seen in slide 17 when choosing gamma, the width of the Gaussian has to be sufficient for the influence of the center mu to cover the points in the surrounding cluster. So when we have a gamma for each mu_k, the clusters covering larger areas will tend to have a larger gamma.

RBF for classification

$$h(\mathbf{x}) = \text{sign}\left(\sum_{n=1}^{N} w_n \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2\right)\right)$$

Learning: $\sim$ linear regression for classification

Like when we used linear regression for classification, we want the signal s for each point to equal the corresponding +1 or -1 target - as such even if there is an error, making s close to +1 or -1 means that when we take the sign, it will agree with the target.

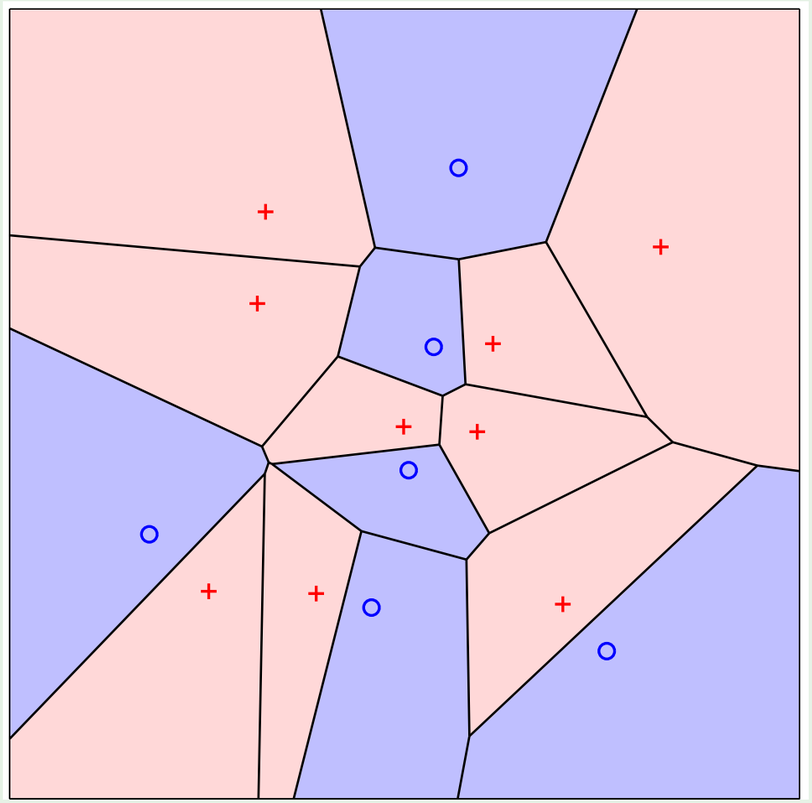
$$s = \sum_{n=1}^{N} w_n \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_n\|^2\right)$$

Minimize $(s - y)^2$ on $\mathcal{D}$ $\quad y = \pm 1$

$$h(\mathbf{x}) = \text{sign}(s)$$

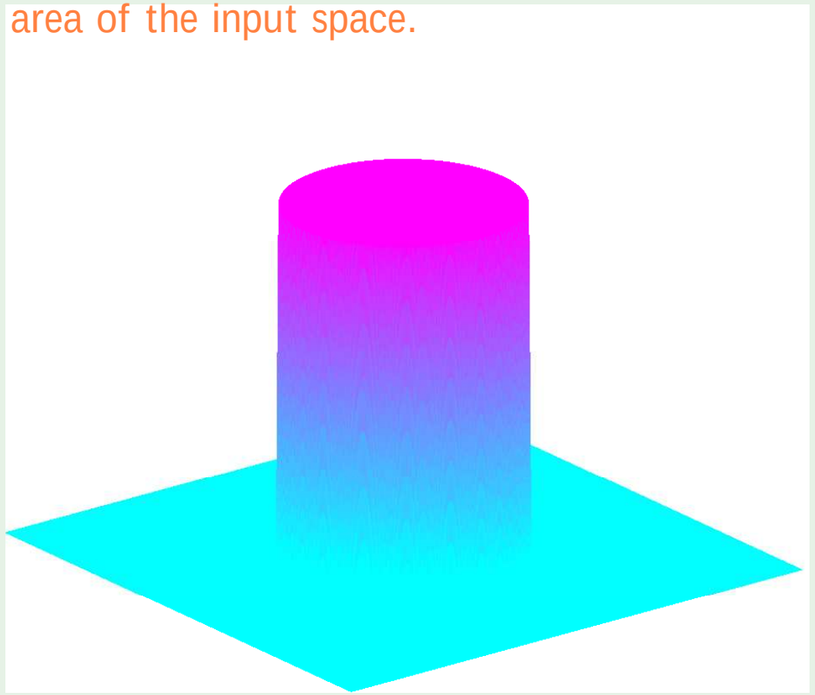
Relationship to nearest-neighbor method

Adopt the y value of a nearby point:



similar effect by a basis function:

The training point xn has total influence its nearby area of the input space.



K-NN: look at the K nearest points and take a vote - if most are +1, consider x as +1. This smooths the surface and although it will still be very abrupt the number of fluctuations will go down. This is similar to using a Gaussian basis function in RBF (instead of a cylinder in strict NN) - each xn has a gradually decaying influence over the classification of nearby points in the input space dependent on their distance from xn. Note that NN/K-NN and radial basis function with different basis functions may be considered to be similarity-based methods, where we classify points according to how similar they are to points in the training set. The particular form of applying the similarity is what defines the algorithm.

RBF with K centers

modification to the exact interpolation model so we do not have as many parameters as we have data points. Given that the generalization ability may be characterized by the ratio of training data to VC dimension, this is an issue.

N parameters $w_1, \cdots, w_N$ based on N data points

(issue for generalization!)

Use $K \ll N$ centers: $\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_K$ instead of $\mathbf{x}_1, \cdots, \mathbf{x}_N$

(now each center influences their neighborhood, rather than each xn)

$$h(\mathbf{x}) = \sum_{k=1}^{K} w_k \exp\left(-\gamma \|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)$$

There are K of the new parameters mu_k, each of which is a d-dimensional vector, where d is the dimensionality of the training data. This combination of K and d could create many new parameters we have to determine, which is against the point of this whole modification. However, there will be an algorithm of determining each mu_k without touching the outputs of the training set/contaminating the data.

1. How to choose the centers $\boldsymbol{\mu}_k$
2. How to choose the weights w_k

Choosing the centers

We need to choose centers which are representative of the training data - it would be nice for every group of points that are nearby to have a center near to them so that it captures this cluster.

Minimize the distance between $\mathbf{x}_n$ and the **closest** center $\boldsymbol{\mu}_k$: K-means clustering

Split $\mathbf{x}_1, \cdots, \mathbf{x}_N$ into clusters $S_1, \cdots, S_K$

We want to find the mu_k and the separation of the points into each cluster Sk such that this value assumes its minimum

$$\text{Minimize} \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$$

Unsupervised learning ☺ since no reference to the label yn, we take the inputs and produce some organization of them

NP-hard ☹ intractable in general to get the absolute minimum, so like in neural networks we find a heuristic (e.g. gradient descent and leading to back-propagation), we start with a random configuration and descend to hopefully a decent local minimum

An iterative algorithm

Lloyd's algorithm: Iteratively minimize $\sum_{k=1}^{K} \sum_{\mathbf{x}_n \in S_k} \|\mathbf{x}_n - \boldsymbol{\mu}_k\|^2$ w.r.t. $\boldsymbol{\mu}_k, S_k$

The algorithm fixes one of the two parameters and tries to minimizes the other

So if Sk were the real cluster, mu_k will be a good representative

$$\boldsymbol{\mu}_k \leftarrow \frac{1}{|S_k|} \sum_{\mathbf{x}_n \in S_k} \mathbf{x}_n$$

Take the mean of the cluster - seems like a good way to minimize the MSE since the SE to the mean is the smallest of the SE to any point - it is the closest to the points collectively in terms of mean squared value

Now freeze mu_k, take every point & measure the distance to this mu_k, compare to its distance to all other mu and if it happens to be smaller, then we say that xn belongs to Sk

$$S_k \leftarrow \{\mathbf{x}_n : \|\mathbf{x}_n - \boldsymbol{\mu}_k\| \leq \text{all } \|\mathbf{x}_n - \boldsymbol{\mu}_\ell\|\}$$

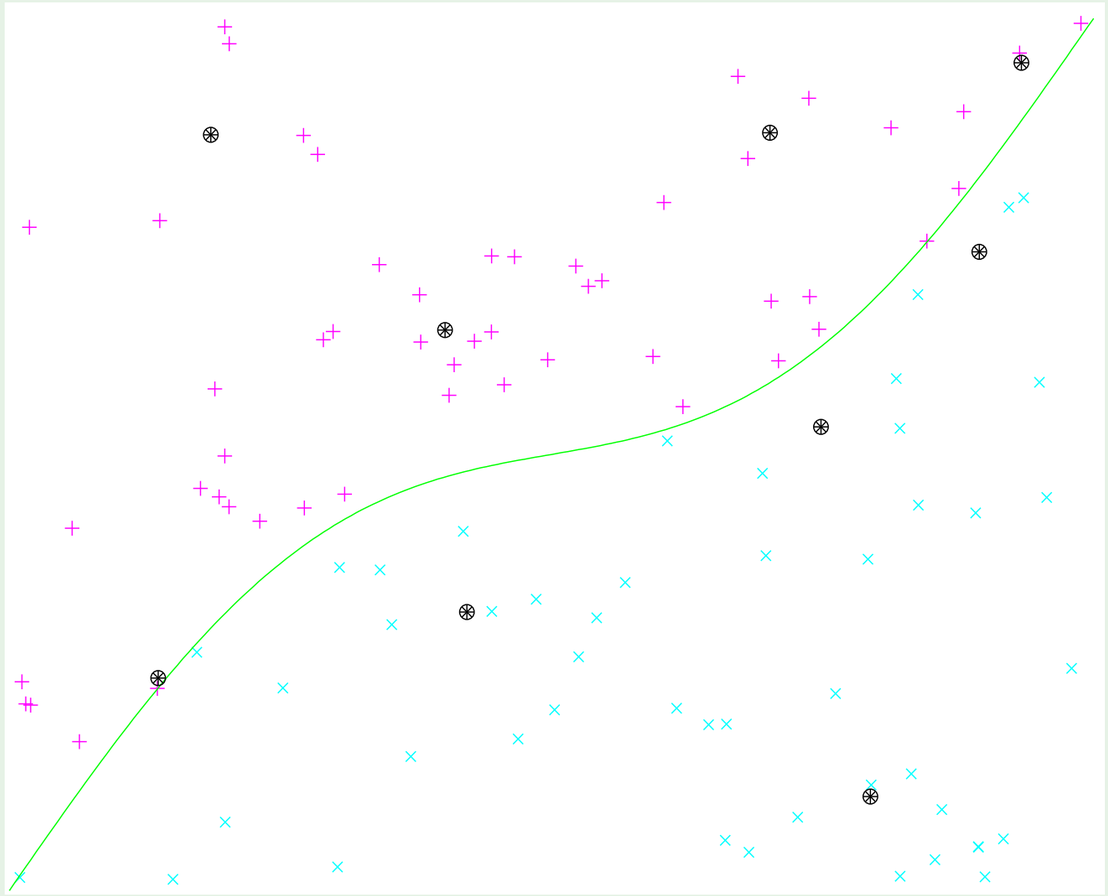
Convergence $\longrightarrow$ **local minimum**

Hence the distance |xn - mu_k| from the above summation will decrease in each iteration. We will converge because we are only dealing with a finite number of points and there are only a finite number of possible mu_k since we take the mean of some if these points. The local minimum we reach will depend on the initial centers or the initial clusters (whichever way we want to begin). Trying many different initial starting points, we can compare the minimum values we get for the above summation and evaluate which is better - picking the best out of these runs will give us a decent clustering and representative mu's.

Lloyd's algorithm in action

1. Get the data points
2. Only the inputs!
3. Initialize the centers
4. Iterate
5. These are your $\boldsymbol{\mu}_k$'s

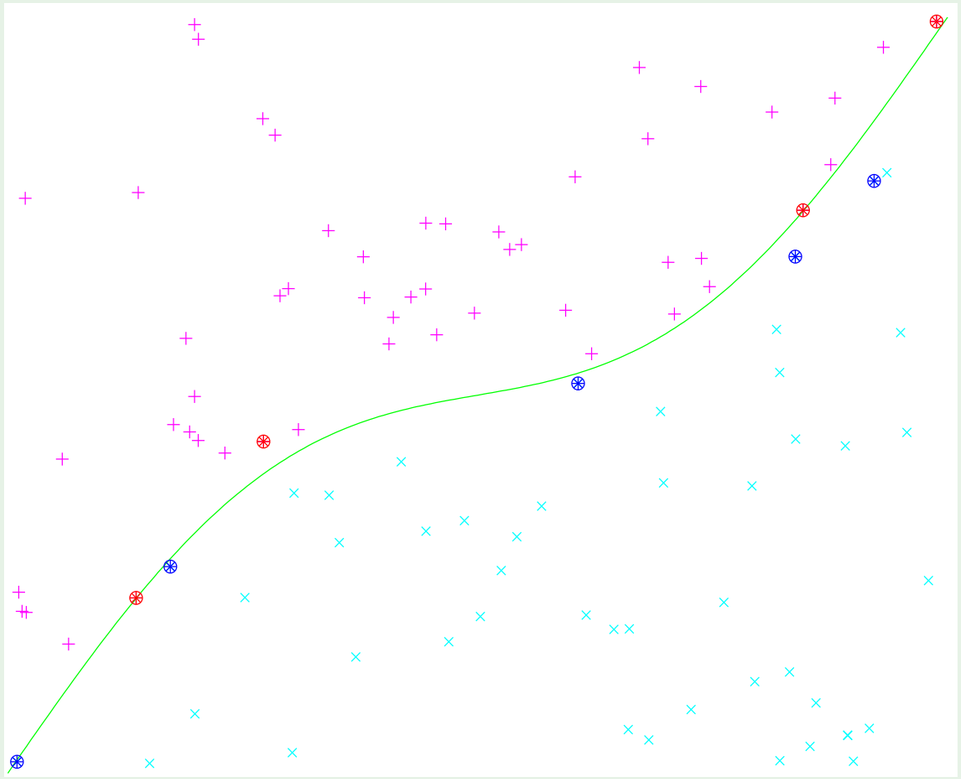
here we use K=9 to compare to the support vectors from last lecture



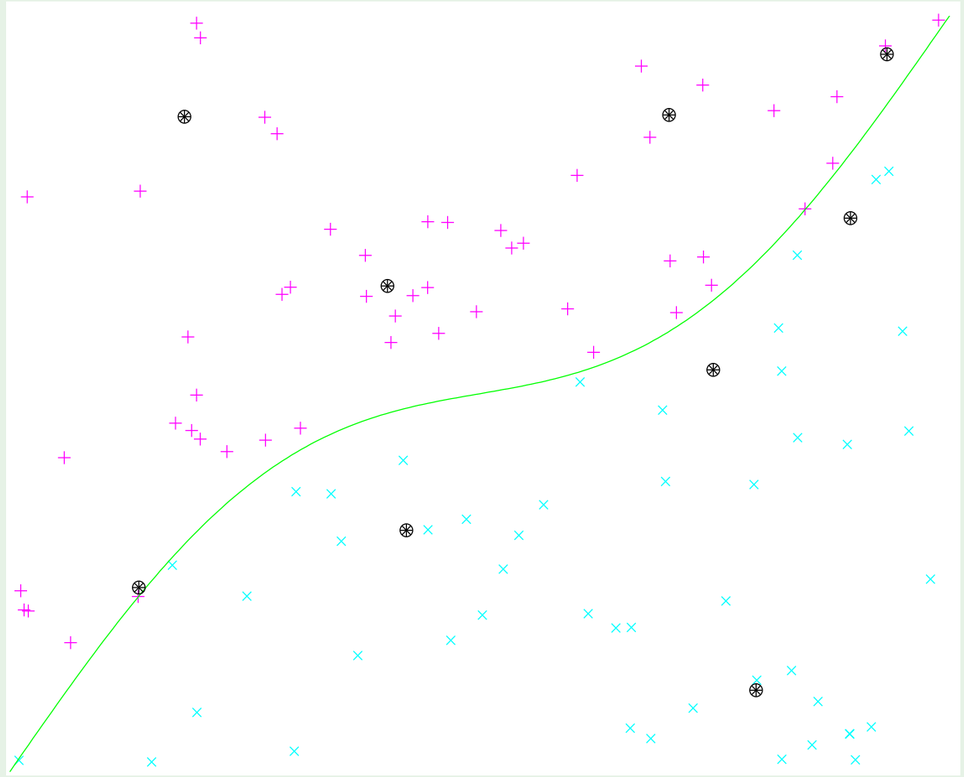
We could have clusters which involve inputs with different labels - this is the price we pay doing unsupervised learning: we are looking for similarity but the similarity is as far as the input is concerned, not as far as the target function is concerned. Note in the above example the data is not naturally clustered - they were not generated from 9 centers so the clustering is incidental, but nonetheless it seems to make sense.

Centers versus support vectors

support vectors



RBF centers



Here we see two different solutions using the same kernel (RBF kernel) using a very different set of approaches - one where we choose the 'important points' in a supervised way and an unsupervised way. Also, note the SVs are also points from the training data, while mu_k are just the centers/ averages of their corresponding cluster.

Choosing the weights

using the labels of the training data

$$\sum_{k=1}^{K} w_k \exp\left(-\gamma \left\|\mathbf{x}_n - \boldsymbol{\mu}_k\right\|^2\right) \approx y_n$$

N equations in $K < N$ unknowns

We have more equations than unknowns, so can only get an approximation of yn (in a mean squared sense)

$$\underbrace{\begin{bmatrix} \exp(-\gamma \left\|\mathbf{x}_1 - \boldsymbol{\mu}_1\right\|^2) & \dots & \exp(-\gamma \left\|\mathbf{x}_1 - \boldsymbol{\mu}_K\right\|^2) \\ \exp(-\gamma \left\|\mathbf{x}_2 - \boldsymbol{\mu}_1\right\|^2) & \dots & \exp(-\gamma \left\|\mathbf{x}_2 - \boldsymbol{\mu}_K\right\|^2) \\ \vdots & \vdots & \vdots \\ \exp(-\gamma \left\|\mathbf{x}_N - \boldsymbol{\mu}_1\right\|^2) & \dots & \exp(-\gamma \left\|\mathbf{x}_N - \boldsymbol{\mu}_K\right\|^2) \end{bmatrix}}_{\Phi} \underbrace{\begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_K \end{bmatrix}}_{\mathbf{w}} \approx \underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{\mathbf{y}}$$

phi is an NxK matrix, so a tall matrix

If $\Phi^{\mathsf{T}}\Phi$ is invertible, $\boxed{\mathbf{w} = (\Phi^{\mathsf{T}}\Phi)^{-1}\Phi^{\mathsf{T}}\mathbf{y}}$ **pseudo-inverse**

(as in linear regression)

instead of exact interpolation, so we are not guarenteed that we will get the correct yn at every data point, so we will have non-zero Ein. However, we are only calculating K weights, so the chances of generalization are likely to be good.

RBF network

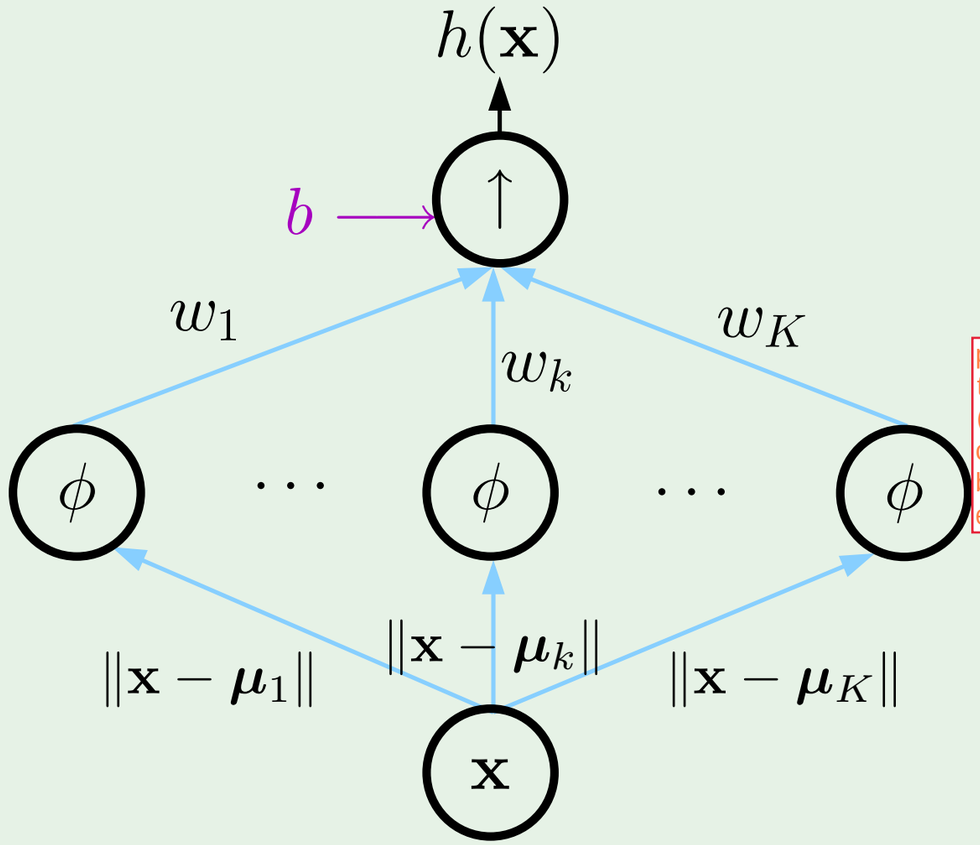
The "features" are $\exp\left(-\gamma \left\|\mathbf{x} - \boldsymbol{\mu}_k\right\|^2\right)$

Nonlinear transform depends on $\mathcal{D}$

since the determination of mu_k depended on D

$\Longrightarrow$ No longer a linear model

as the value of the feature depends on the dataset D (like a neural network, transforming the 1st layer and extracting the features). However, since they only depended on D (rather than the output) it is almost linear. We got the benefit of the pseudo inverse to find the w's, and w's act as multiplicative factors, so linear in w.

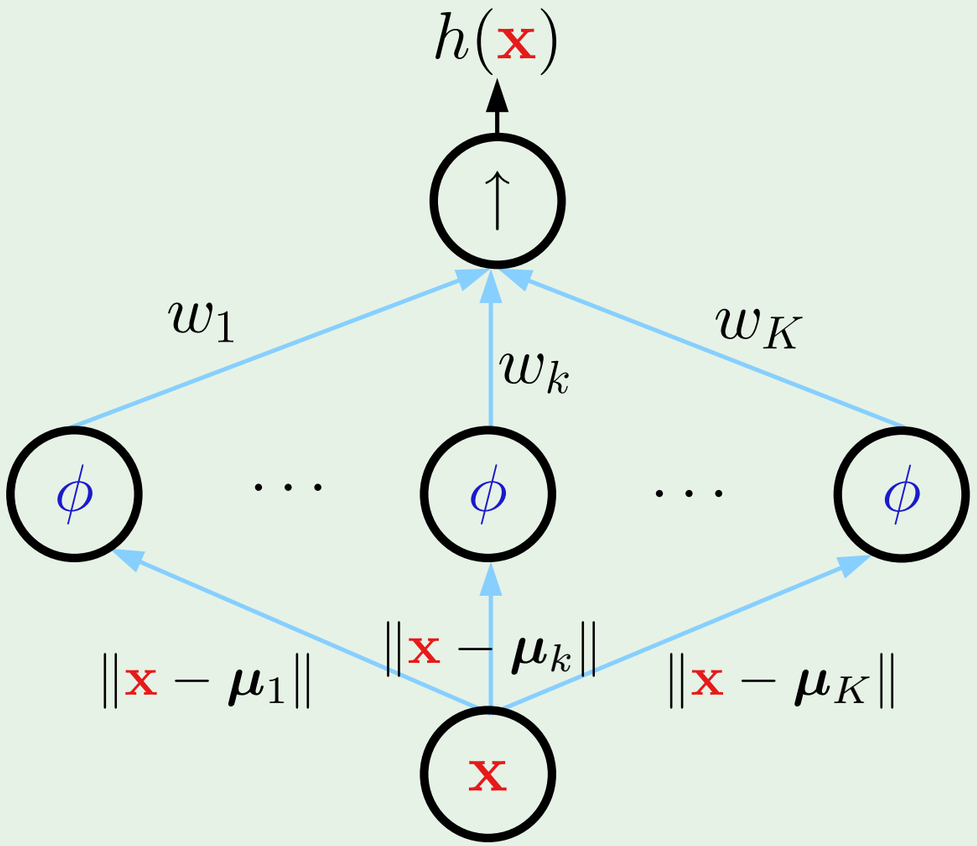


phi represents the non-linearity (Gaussian, hard cylinder, other basis functions etc.)

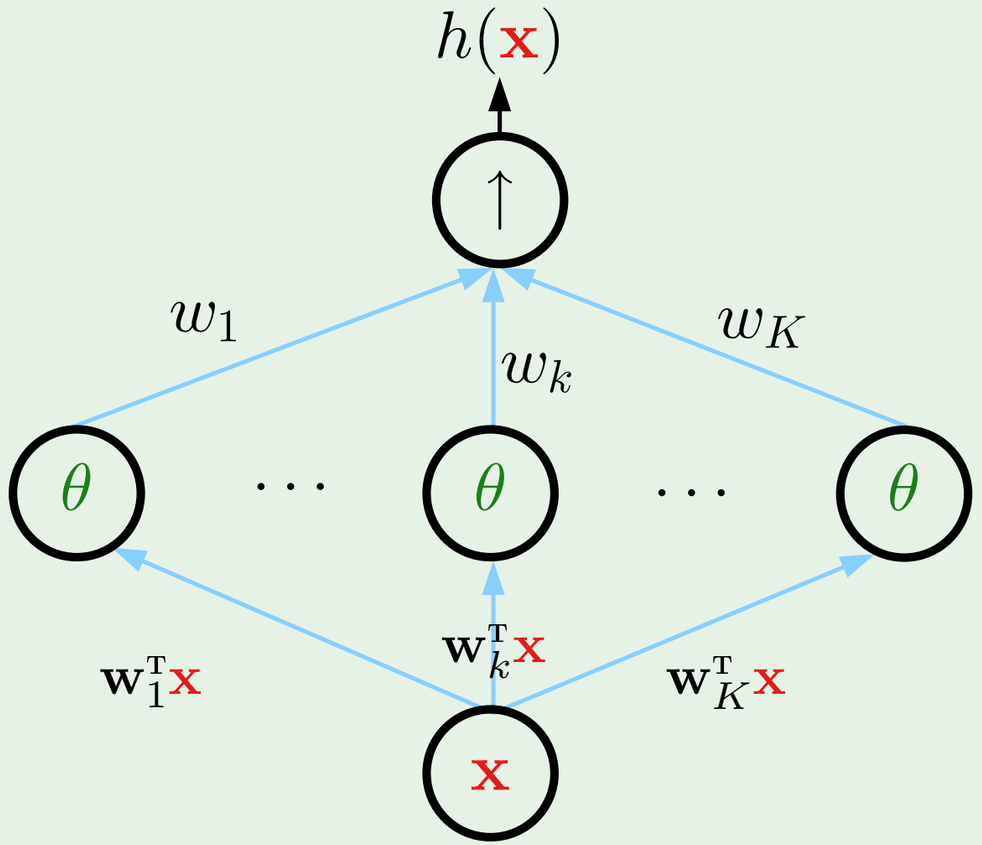
A bias term (b or w_0) is often added

added in the final layer

Compare to neural networks



RBF network



neural network

(1) RBF look at local regions in the space without worrying about the distant regions - using a basis function to capture a small portion of a function will not interfere with a more distant part of the function. This is because if x is far from mu_k, the value passed to the non-linearity layer will be small. Meanwhile, the term wT_k*x in the neural network goes through a sigmoid, so each x_n will contribute. Hence in neural networks, the approximation of the target function in each local region affects the approximation globally, and the way we get something interesting in neural networks is to make sure that the combinations of each neuron gives us the total hypothesis. (2) The two models seem similar in that the non-linearity phi or theta is simply multiplied by the corresponding weight and summed to get the output. However the way we extract features in the RBF is different. In the neural network, w is a fully fledged parameter which depends on the labels (use back propagation to find them, so they are learned features and so it is not a linear model). In RBF, mu_k are not learned based on their affect on the output, so it is almost linear. (3) Any two layer network of this sort of structure lends itself to being an SVM - the 1st layers takes care of the kernel and the 2nd is the linear combination that is built into SVMs. It is possible to implement a two -ayer neural networks using SVMs if, depending on the choice of parameters, the kernel corresponds to the inner product of a legitimate Z-space.

Choosing γ

Treating γ as a parameter to be learned $\quad h(\mathbf{x}) = \sum_{k=1}^{K} w_k \exp\left(-\gamma \|\mathbf{x} - \boldsymbol{\mu}_k\|^2\right)$

(variation on the expectation maximization algorithm)

Iterative approach ($\sim$ **EM algorithm** in mixture of Gaussians):

Since we can easily find the weights using psuedoinverse, there is no need to use a general non-linear optimization like gradient descent to find both parameters at once. Instead we fix one of w and gamma, solve for the other and iterate.

1. Fix γ, solve for $w_1, \cdots, w_K$
2. Fix $w_1, \cdots, w_K$, minimize error w.r.t. γ

(gradient descent)

Given how simple this iterative algorithm is — We can have a different γ_k for each center $\boldsymbol{\mu}_k$

(If we have a dataset where one point is very far away from the others which are close together, the center of the close points should logically reach out further (decay more slowly) while the other center does not have to reach out far, so we can use different gammas for each center)
We adjust the width of the Gaussian according to the region of space we are in.

Outline

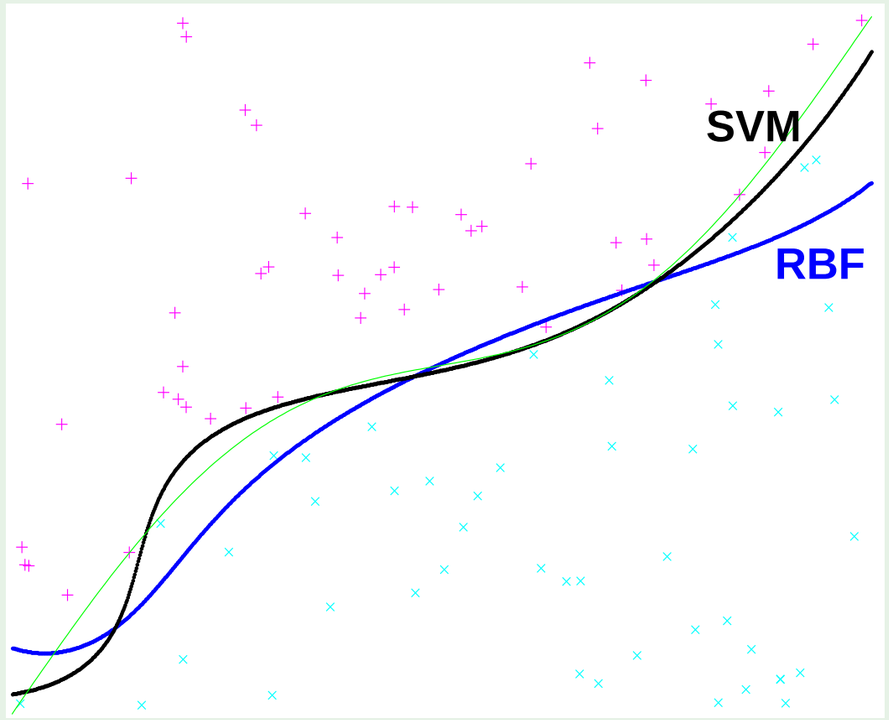
- RBF and nearest neighbors
- RBF and neural networks
- RBF and kernel methods
- RBF and regularization

RBF versus its SVM kernel

SVM kernel implements:

$$\text{sign}\left(\sum_{\alpha_n>0} \alpha_n y_n \exp\left(-\gamma \|\mathbf{x}-\mathbf{x}_n\|^2\right) + b\right)$$

involved maximizing the margin, equate with a kernel and pass to QP



Straight RBF implements:

$$\text{sign}\left(\sum_{k=1}^{K} w_k \exp\left(-\gamma \|\mathbf{x}-\boldsymbol{\mu}_k\|^2\right) + b\right)$$

involved unsupervised learning of centers, then pseudoinverse to find w,
and linear regression for classification (hence the addition of the blue sign(...))
Note that the data does not cluster naturally, also K=9 was only chosen to compare to SVM (not necessarily optimal)

We see SVM is better, even though both methods eventually use the same data: Ein=0, seems to approximate target function better than RBF. We see the ramifications of doing unsupervised learning and what we miss out on by choosing the centers without knowing the label, versus the advantages of SVM (however, SVM can only be used for limited sizes of datasets...

RBF and regularization

formulation from function approximation

RBF can be derived based purely on regularization:

smoothness constraint (constrain the magnitude of the derivatives to be small)

Augmented error:

$$\sum_{n=1}^{N} \left(h(x_n) - y_n\right)^2 + \lambda \sum_{k=0}^{\infty} a_k \int_{-\infty}^{\infty} \left(\frac{d^k h}{dx^k}\right)^2 dx$$

"smoothest interpolation"

This minimization of the above error gives us radial basis functions - the best interpolation (as smooth an interpolation as possible in the sense of the sum of the squares of the derivatives with these coefficients) happens to be Gaussian. This interpretation is the reason for RBF having a certain credibility of being inherently self-regularized and principled (among other properties).

Note there are different underlying assumptions which motivate the use of radial basis functions. As with the final slide, the requirement that the target function is smooth is one of these. Another motivation results from the following: say we have a dataset (x1,y1)...(xn,yn) where each xn has an associated (assumed Gaussian) noise, i.e. we cannot measure the input exactly. We assume that the labels yn are noiseless. We have to make it such that the value of our hypothesis does not change much by changing in x. Doing this means we arrive with an interpolation which is Gaussian.