Regression
(Linear Models)

David Barber

Regression

- This is a form of supervised learning.
- We have a dataset of input-output examples $(\mathbf{x}^n, y^n), n = 1, \ldots, N$
- We want to learn a mapping from inputs $\mathbf{x}$ to output y such that when we get a new input $\mathbf{x}$, we will output the correct output.

Example

- Given a set of vectors with features (attributes)

$$\begin{pmatrix} \text{client age} \\ \text{client postcode indicator} \\ \text{client monthly tesco spend} \\ \text{client purchase types} \end{pmatrix}$$

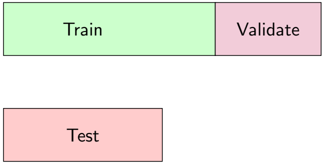
  predict their annual income.
- We may have a very large set of such features, only some of which may be relevant for the prediction.

Generalisation

Basic Assumptions

- The data we have (all inputs and corresponding outputs) is generated from the same unvarying mechanism.
- We will typically split our available data into three distinct parts: train data, validation data and test data.
- We want to find a model that will have good generalisation performance.

Validation

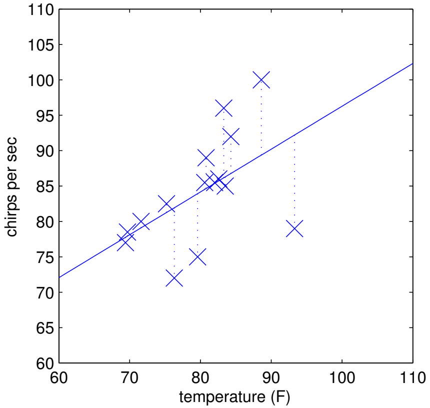


Different models can be trained using the train data. The optimal model is determined by the empirical performance on the validation data. An independent measure of the generalisation performance of this optimal model is obtained by using a separate test set.

Fitting a Line

Given training data $\{(x^n, y^n), n = 1, \dots, N\}$, a linear regression fit is

$$y(x) = a + bx$$



Data from singbats – the number of chirps per second, versus the temperature in Fahrenheit. Straight line regression fit to the data.

Least Squares fitting

$$E(a,b) = \sum_{n=1}^{N} \left[y^n - y(x^n)\right]^2 = \sum_{n=1}^{N} (y^n - a - bx^n)^2$$

To minimise $E(a,b)$ we differentiate with respect to a and b we obtain

$$\frac{\partial}{\partial a} E(a,b) = -2 \sum_{n=1}^{N} (y^n - a - bx^n), \quad \frac{\partial}{\partial b} E(a,b) = -2 \sum_{n=1}^{N} (y^n - a - bx^n) x^n$$

- We then equate these two equations to zero to find the optimum.
- The resulting two linear equations can be solved to find a and b.

Linear Parameter Models for Regression

We can generalise the above to fitting linear functions of vector inputs $\mathbf{x}$. For a dataset $\{(\mathbf{x}^n, y^n), n = 1, \dots, N\}$, a linear parameter model is

$$y(\mathbf{x}) = \mathbf{w}^\mathsf{T}\boldsymbol{\phi}(\mathbf{x})$$

where $\boldsymbol{\phi}(\mathbf{x})$ is a vector valued function of the input vector $\mathbf{x}$. For example, in the case of a straight line fit, with scalar input and output, we have

$$\boldsymbol{\phi}(x) = (1, x)^\mathsf{T}, \qquad \mathbf{w} = (a, b)^\mathsf{T},$$

We define the error as the sum of squared differences between the observed outputs and the predictions under the linear model:

$$E(\mathbf{w}) = \sum_{n=1}^{N} (y^n - \mathbf{w}^\mathsf{T}\boldsymbol{\phi}^n)^2, \qquad \text{where } \ \boldsymbol{\phi}^n \equiv \boldsymbol{\phi}(\mathbf{x}^n)$$

Normal Equations

Differentiating $E(\mathbf{w})$ with respect to w_k, and equating to zero gives

$$\sum_{n=1}^{N} y^n \phi_k^n = \sum_i w_i \sum_{n=1}^{N} \phi_i^n \phi_k^n$$

or, in matrix notation,

$$\sum_{n=1}^{N} y^n \boldsymbol{\phi}^n = \sum_{n=1}^{N} \boldsymbol{\phi}^n (\boldsymbol{\phi}^n)^{\mathsf{T}} \mathbf{w}$$

These are called the normal equations, for which the solution is

$$\mathbf{w} = \left(\sum_{n=1}^{N} \boldsymbol{\phi}^n (\boldsymbol{\phi}^n)^{\mathsf{T}} \right)^{-1} \sum_{n=1}^{N} y^n \boldsymbol{\phi}^n$$

Although we write the solution using matrix inversion, in practice one finds the numerical solution using Gaussian elimination since this is faster and numerically more stable.

Polynomial Fitting

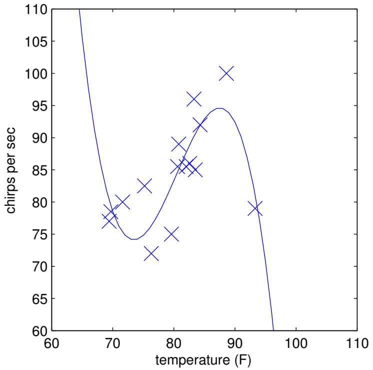


Figure : Cubic polynomial fit to the singbat data.

A cubic polynomial is given by

$$y(x) = w_1 + w_2 x + w_3 x^2 + w_4 x^3$$

As a LPM, this can be expressed using

$$\boldsymbol{\phi}(x) = \left(1, x, x^2, x^3\right)^\mathsf{T}$$

Regularisation

- Sensible to first scale each input dimension x_i so that is has unit variance:

$$x_i^n \to \frac{x_i^n}{\sigma_i}$$

where the variance in the i^{th} dimension is given by

$$\sigma_i^2 = \frac{1}{N}\sum_{n=1}^{N}\left(x_i^n - \mu_i\right)^2, \qquad \mu_i = \frac{1}{N}\sum_{n=1}^{N} x_i^n$$

- Add an extra regularising term $R(\mathbf{w})$ to penalise rapid changes in the output

$$E'(\mathbf{w}) = E(\mathbf{w}) + \lambda R(\mathbf{w})$$

where λ is a regularisation constant.

- We can determine λ using validation data (so that we find a λ that promotes good generalisation performance).

L_2 regularisation or ‘Ridge Regression’

- Add a quadratic penalty to the objective. This prevents large weight values:

$$E'(\mathbf{w}) = \sum_{n=1}^{N} (y^n - \mathbf{w}^{\mathsf{T}} \boldsymbol{\phi}^n)^2 + \lambda \mathbf{w}^{\mathsf{T}} \mathbf{w}$$

  The optimal $\mathbf{w}$ is given by

$$\mathbf{w} = \left(\sum_n \boldsymbol{\phi}^n (\boldsymbol{\phi}^n)^{\mathsf{T}} + \lambda \mathbf{I} \right)^{-1} \sum_{n=1}^{N} y^n \boldsymbol{\phi}^n$$

- λ may be determined using a validation set.
- Historically L_2 regularisation is popular since there is an easy analytic (and computational) solution to finding the optimum.

Issues with Ridge Regression

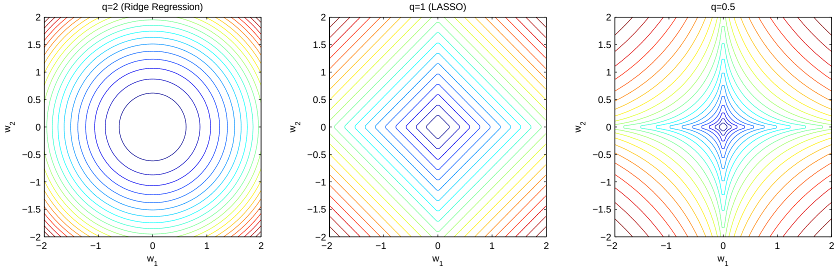
- Consider linear regression with

$$\frac{1}{N}\sum_{n=1}^{N}(y^n - w_1 x_1^n - w_2 x_2^n)^2 + w_1^2 + w_2^2$$

- Let $x_1^n = x_2^n$ ('co-linear' case). Then either x_1 or x_2 is redundant.
- For $w_1 = 1$,$w_2 = 0$ the ridge penalty is $1^2 + 0^2 = 1$
- For $w_1 = 1/2$,$w_2 = 1/2$ the ridge penalty is $(1/2)^2 + (1/2)^2 = 1/2$
- Both cases above have the same squared loss, but the second one in which both weights are equally active is preferred by ridge regression.

Different Penalty terms

Regularisation penalty $\sum_i |w_i|^q$.



- As we decrease q towards 0 we get the ‘ideal’ regulariser that selects weights which are 0 if they do not contribute.
- The objective function is complicated (non convex) for $q < 1$ 🔴
- For $q = 1$ the objective function for squared loss linear regression remains convex and easy to optimise 🟢

L_1 verus L_2 regularisation

$L_2(\mathbf{w}) = \sum_i w_i^2$

- Can also be written as

$$\mathbf{w}^\mathsf{T}\mathbf{w} = \sum_i w_i^2$$

- This heavily penalises large w_i but only penalises by a small amount small w_i.
- The penalty is differentiable.
- Weights which are small will still persist.

$L_1(\mathbf{w}) = \sum_i |w_i|$

- This is the sum of the absolute values of the elements of $\mathbf{w}$.
- This heavily penalises large w_i and amount small w_i by a proportional amount.
- The penalty is non-differentiable.
- Need to use specialised optimisation routines.
- Only significant weights will remain – redundant parameters will be ‘pruned’ to zero.

Understanding the L_1 penalty

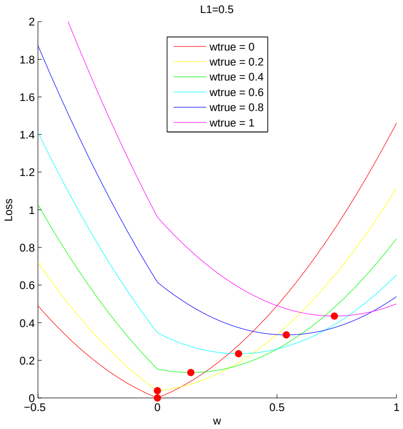
- Consider a Loss function

$$L(w) = \frac{1}{N}\sum_{n=1}^{N}(y^n - wx^n)^2 + \lambda|w|$$

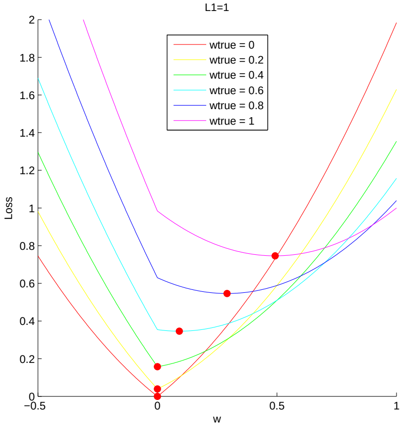
  where the data y^n is generated by

$$y^n = w_0 x^n$$

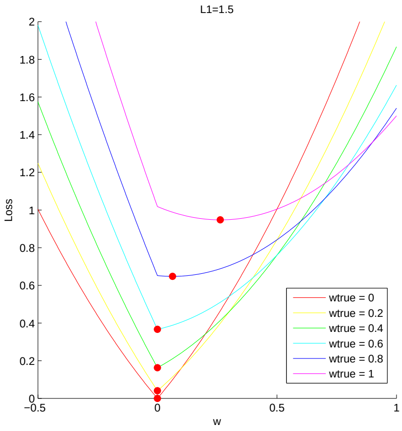
  and the inputs x^n are randomly selected from the Gaussian.
- Ideally we would like the minimum of $L(w)$ to be at the point $w = w_0$ since then we would recover the true data generating process.
- There is a complex interplay between minimising the squared loss term and the penalty.
- Only when the true w_0 is large enough does the estimated w become non-zero.



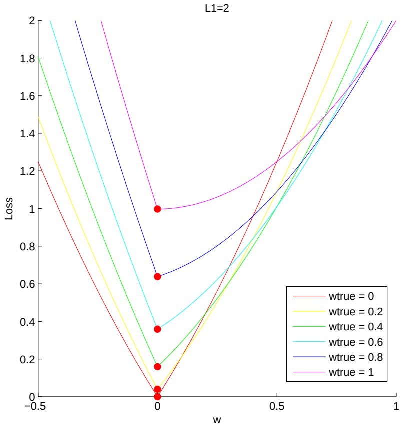
Here the amount of regularisation is small and, except for very small w_0 we estimate non-zero w. The true values of w though are still underestimated.



As we increase the regularisation only for significantly non zero values of w_0 is the estimated w non zero. The estimated w underestimates the true w_0 more.



The regularisation penalty is so high that w is estimated to be zero, even when the true value is significantly non zero.



Here the amount of regularisation is small and, except for very small w_0 we estimate non-zero w. The true values of w though are still underestimated.

L_1 Pruning Effect

Consider a problem in which the train data is generated by $y^n = w_0 x_1^n$, and we try to fit a model

$$w_1 x_1 + w_2 x_2$$

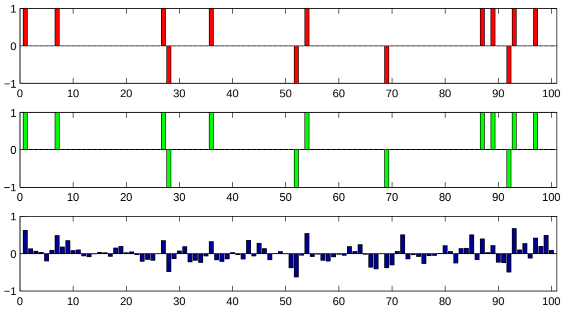
with $x_2^n = x_1^n + \epsilon^n$, $\epsilon^n \ll 1$. In this case we don't need w_2 and we'd like to see this set to 0 by the algorithm.

$$\begin{aligned} E(w_1, w_2) \equiv & \sum_n \left(y^n - w_1 x_1^n - w_2 x_2^n\right)^2 + \lambda(|w_1| + |w_2|) \\ & = \sum_n \left(w_0 x_1^n - w_1 x_1^n - w_2 x_2^n\right)^2 + \lambda(|w_1| + |w_2|) \\ & = \sum_n \left((w_0 - w_1 - w_2) x_1^n - w_2 \epsilon^n\right)^2 + \lambda(|w_1| + |w_2|) \end{aligned}$$

$$E(w_0, 0) = \lambda |w_1|, \qquad E(w_0/2, w_0/2) = w_2^2 \sum_n \epsilon_n^2 + \lambda |w_1|$$

Since $E(w_0, 0) < E(w_0/2, w_0/2)$ the method prefers the sparse solution.

L_1 versus L_2



- Linear regression based on 50 data points and 100 dimensional parameter vector (should be impossible?!)
- Top: true weights
- Middle: L_1 regularised weights.
- Bottom: L_2 regularised weights.
- Provided the true weight vector is sparse, even if we have fewer datapoints than parameters we can still recover the correct solution.

Comments on L_1 regularisation

- L_1 is often used with linear parameter models since it results in simple objective functions that are easy to optimise.
- L_1 penalisation does not deal with the co-linear x case. This is not usually an issue in practice.
- To deal with co-linearity we need a non-convex objective function that is difficult to optimise.

Expert feature selection versus L_1 penalisation

- The historical approach is to ask experts to decide which features (components of x) should be included.
- The modern approach is to throw in all features that one might consider relevant and use L_1 penalisation to determine which are relevant, with the non-relevant features being regularised to zero.

Comments on L_1 regularisation

- For a non-linear regression model, the corresponding L_1 regularised objective is more complex, typically non-convex.
- In this case need special optimisation methods to formally deal with strict L_1 penalty.
- In practice (*e.g.* in deep learning), people often don't worry too much about the non-differentiability at the origin of the L_1 penalty, and just proceed with gradient based training as normal.
- If the non-differentiability is a concern, simple differentiable approximations are popular, such as

  $$|x| \leq \sqrt{x^2 + \epsilon^2}$$

  and use a small ϵ.

Auto-Regressive Models

We can predict the future value v_t based on past values $v_{t-1}, \ldots, v_{t-L}$ using

$$v_t \approx \sum_{l=1}^{L} a_l v_{t-l}$$

where $\mathbf{a} = (a_1, \ldots, a_L)^\mathsf{T}$ are the ‘AR coefficients’.

- The model predicts the future based on a linear combination of the previous L observations.
- This is exactly of the same form as our linear model

  $$v_t \approx \mathbf{a}^\mathsf{T} \boldsymbol{\phi}^t$$

  where $\boldsymbol{\phi}^t = (v_{t-1}, v_{t-2}, \ldots, v_{t-L})^\mathsf{T}$.
- We can train this model as usual using the least squares objective with L_1 or L_2 regularisation.

Auto-Regressive Models (Probabilistic Viewpoint)

We can predict the future value v_t based on past values $v_{t-1}, \ldots, v_{t-L}$ using

$$v_t = \sum_{l=1}^{L} a_l v_{t-l} + \eta_t, \qquad \eta_t \sim \mathcal{N}\left(\eta_t | \mu, \sigma^2\right)$$

where $\mathbf{a} = (a_1, \ldots, a_L)^\mathsf{T}$ are the ‘AR coefficients’ and σ^2 is the ‘innovation’ level. This is an L^{th} order Markov model:

$$p(v_{1:T}) = \prod_{t=1}^{T} p(v_t | v_{t-1}, \ldots, v_{t-L}), \qquad \text{with } v_i = \emptyset \text{ for } i \leq 0$$

with

$$p(v_t | v_{t-1}, \ldots, v_{t-L}) = \mathcal{N}\left(v_t \middle| \sum_{l=1}^{L} a_l v_{t-l}, \sigma^2\right) = \mathcal{N}\left(v_t | \mathbf{a}^\mathsf{T} \hat{\mathbf{v}}_{t-1}, \sigma^2\right)$$

where

$$\hat{\mathbf{v}}_{t-1} \equiv \left[v_{t-1}, v_{t-2}, \ldots, v_{t-L}\right]^\mathsf{T}$$

Training an AR model (Probabilistic Viewpoint)

Maximum Likelihood training of the AR coefficients requires maximising

$$\log p(v_{1:T}) = \sum_{t=1}^{T} \log p(v_t|\hat{\mathbf{v}}_{t-1}) = -\frac{1}{2\sigma^2}\sum_{t=1}^{T}\left(v_t - \hat{\mathbf{v}}_{t-1}^{\mathsf{T}}\mathbf{a}\right)^2 - \frac{T}{2}\log(2\pi\sigma^2)$$

Differentiating *w.r.t.* $\mathbf{a}$ and equating to zero we arrive at

$$\sum_t \left(v_t - \hat{\mathbf{v}}_{t-1}^{\mathsf{T}}\mathbf{a}\right)\hat{\mathbf{v}}_{t-1} = 0$$

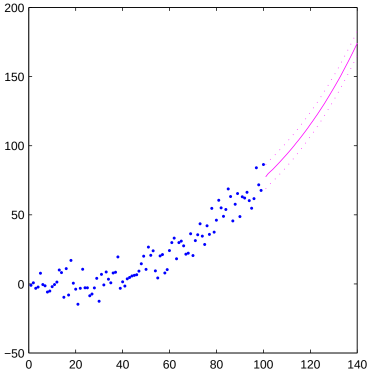
so that optimally

$$\mathbf{a} = \left(\sum_t \hat{\mathbf{v}}_{t-1}\hat{\mathbf{v}}_{t-1}^{\mathsf{T}}\right)^{-1}\sum_t v_t \hat{\mathbf{v}}_{t-1}$$

These equations can be solved by Gaussian elimination. Similarly, optimally,

$$\sigma^2 = \frac{1}{T}\sum_{t=1}^{T}\left(v_t - \hat{\mathbf{v}}_{t-1}^{\mathsf{T}}\mathbf{a}\right)^2$$

AR model: Fitting a trend



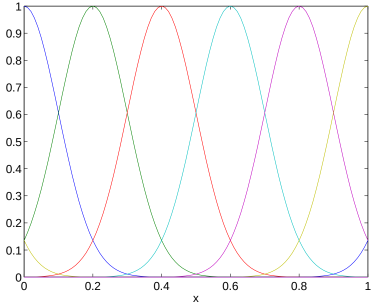
- Fitting an order 3 AR model to the training points.
- The x axis represents time, and the y axis the value of the timeseries.
- The solid line is the mean prediction and the dashed lines $\pm$ one standard deviation.

Radial basis functions

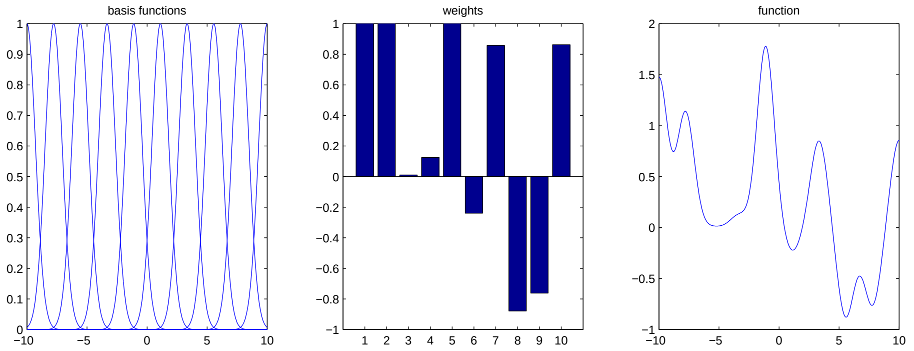
A popular LPM is given by the non-linear function

$$\phi_i(\mathbf{x}) = \exp\left(-\frac{1}{2\alpha^2}\left(\mathbf{x} - \mathbf{m}^i\right)^2\right)$$

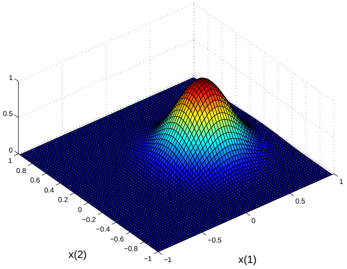
These basis functions are bump shaped, with the centre being given by $\mathbf{m}^i$ and the width by α.



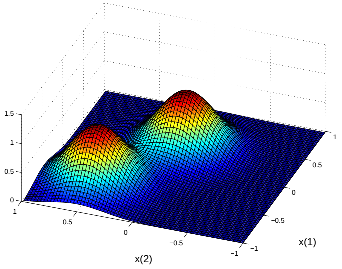
Radial basis functions



RBFs with higher dimensional input



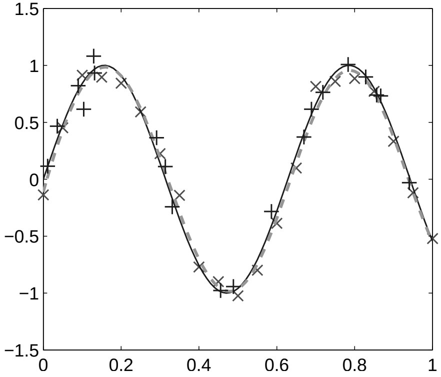
(a)



(b)

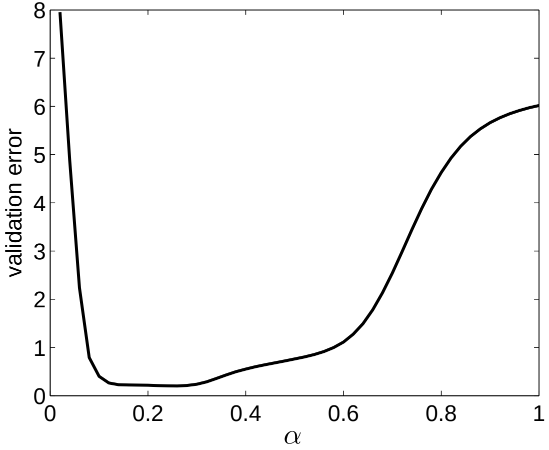
Figure : **(a)**: The output of an RBF function $\exp(-\frac{1}{2}\left(\mathbf{x}-\mathbf{m}^1\right)^2/\alpha^2)$. Here $\mathbf{m}^1 = (0, 0.3)^\mathsf{T}$ and $\alpha = 0.25$. **(b)**: The combined output for two RBFs with $\mathbf{m}^1$ as above and $\mathbf{m}^2 = (0.5, -0.5)^\mathsf{T}$.

Fitting the RBF to data



The $\times$ are the training points, and the $+$ are the validation points. The solid line is the correct underlying function $\sin(10x)$ which is corrupted with a small amount of additive noise to form the train data. The dashed line is the best predictor based on the validation set.

Setting the RBF width α



The validation error as a function of the basis function width for the validation data. Based on the validation error, the optimal setting of the basis function width parameter is $\alpha = 0.25$.

A curse of dimensionality

- If the data has non-trivial behaviour over some input region, then we need to cover this region input space fairly densely with bump type functions.
- In the above case, we used 16 basis functions for a one dimensional input space.
- In 2 dimensions if we wish to cover each dimension to the same discretisation level, we would need $16^2 = 256$ basis functions. Similarly, for 10 dimensions we would need $16^{10} \approx 10^{12}$ functions. To fit such an LPM would require solving a linear system in more than 10^{12} variables.
- This explosion in the number of basis functions with the input dimension is a 'curse of dimensionality'.
- One way around this is to enable the centres of the RBF to move – this is similar to what happens in a neural network.
- Gaussian Processes are an alternative approach – see the BRML book.

Summary

- Regression is about modelling inputs to (continuous) outputs.
- Linear Regression and linear parameter models are the most common and are very easy to train 🟢
- RBFs are also very common non-linear input-output mappings but suffer from the curse of dimensionality 🔴
- It is generally a better idea to use L_1 (Lasso) regularisation than the more classical L_2 (ridge regression) regularisation.
- Even for cases with less datapoints than numbers of parameters, provided the true underlying parameter vector is sparse, we can potentially correctly recover this using L_1 regularisation 🟢
- Modern viewpoint is that we can do feature selection automatically – just throw in all features and use L_1 regularisation to find those that are useful 🟢
- Most regression models can be used in time-series prediction 🟢