4.4.4 Instance-Based Algorithms

Instance-based algorithms simply store their training data. When asked to make a prediction, they use the stored points close (according to some distance metric) to the query point to generate a suitable value. For example, the simplest instance-based algorithm, known as 1-Nearest Neighbor [29, 32], returns the value of the stored point closest to the query as a prediction. The computational load of these algorithms is mostly borne during prediction, not during learning. These methods are often called *lazy learning* algorithms since they postpone their work until it must be done.

Instance-based algorithms typically learn aggressively and are able to generate plausible predictions after only a few training samples. They can also use their stored data to estimate the reliability of a new prediction, and do not suffer from destructive interference. However, they do suffer from the computational complexity of making predictions. Since they typically compare the query point to all of the previously seen training points, a naive implementation can have a prediction complexity of $O(n)$ for n training points. Added to this is the complexity of actually making the prediction based on the stored points.

Despite their computational drawbacks, we have chosen an instance-based learning algorithm as the basis of our value-function approximation scheme. In the next section we describe this algorithm and then go on to show how its computational complexity can be tamed.

4.5 Locally Weighted Regression

Locally weighted regression (LWR) [8, 9] is a variation of the standard linear regression technique in which training points close to the query point have more influence over the fitted regression surface. Given a set of training points, linear regression fits the linear model that minimizes squared prediction error over the whole training set. This implicitly assumes that we know the global form of the underlying function that generated the data. LWR, on the other hand, only fits a function locally, without imposing any requirements on the global form.

The LWR learning procedure is simply to store every training point. Algorithm 1 shows how predictions are made for a given query point. The training and query

Algorithm 1 LWR prediction

```
Input:
    Set of training examples, $\{(\vec{x}_1, \vec{y}_1), (\vec{x}_2, \vec{y}_2) \ldots (\vec{x}_n, \vec{y}_n)\}$
    Query point, $\vec{x}_q$
    Bandwidth, $h$
Output:
    Prediction, $\vec{y}_q$
1: Construct a matrix, $A$, whose lines correspond to the $\vec{x}_i$s
2: Construct a matrix, $b$, whose lines correspond to the $\vec{y}_i$s
3: for each point $(\vec{x}_i, \vec{y}_i)$ do
4:     $d_i \leftarrow \text{distance}(\vec{x}_i, \vec{x}_q)$
5:     $k_i \leftarrow \text{kernel}(d_i, h)$
6:     Multiply the $i$th lines of $A$ and $b$ by $k_i$
7: Perform a linear regression using $A$ and $b$ to get a local model, $f(\vec{x})$
8: return $\vec{y}_q = f(\vec{x}_q)$
```

points are assumed to be pairs of real-valued vectors $(\vec{x}, \vec{y})$. There is one real-valued parameter for the algorithm, h, known as the bandwidth. The algorithm constructs a matrix, A, just as in standard linear regression, where

$$A = \begin{bmatrix} x_1^1 & x_1^2 & \ldots & x_1^p & 1 \\ x_2^1 & x_2^2 & \ldots & x_2^p & 1 \\ x_3^1 & x_3^2 & \ldots & x_3^p & 1 \\ \vdots & \vdots & \ddots & & \vdots \\ x_n^1 & x_n^2 & \ldots & x_n^p & 1 \end{bmatrix}.$$

The rows of A correspond to the $\vec{x}_i$s. Another matrix, b is also constructed, corresponding to the $\vec{y}_i$s,

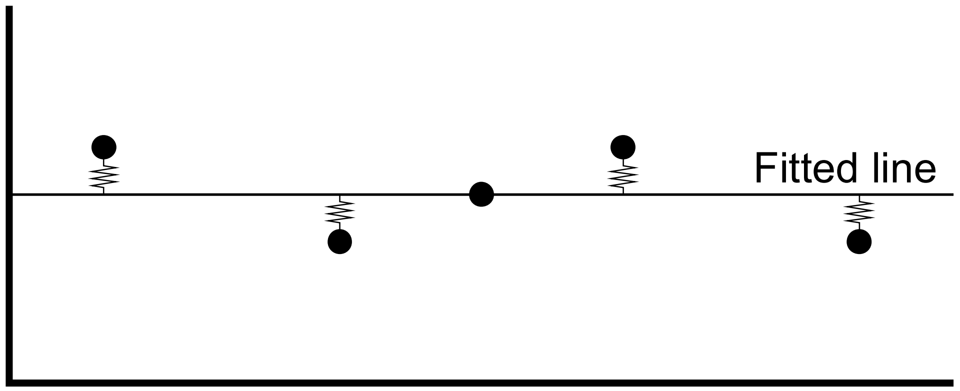
$$b = \begin{bmatrix} y_1^1 & y_1^2 & \ldots & y_1^q \\ y_2^1 & y_1^2 & \ldots & y_2^q \\ \vdots & \vdots & \ddots & \vdots \\ y_n^1 & y_n^2 & \ldots & y_n^q \end{bmatrix}.$$

A standard linear regression would then solve $Ax = b$. However, LWR now goes on to weight the points, so that points close to the query have more influence over the regression. For each data point in the training set, the algorithm calculates that point's distance from the query point (line 4). A kernel function is then applied to this distance, along with the supplied bandwidth, h, to give a weighting for the training point (line 5). This weighting is then applied to the lines of the A and b matrices. A standard linear regression is then performed using the modified A and b to produce a local model at the query point (line 7). This model is then evaluated that the query point, and the value is returned.

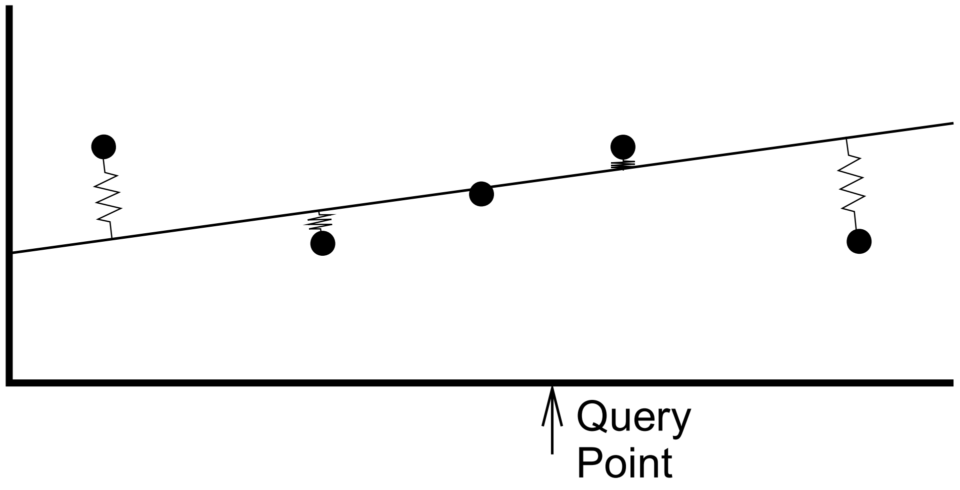
In general, for an m-dimensional space, the bandwidth would actually be an m-dimensional vector, $\vec{h}$. This admits the possibility of a different bandwidth value for each of the dimensions. In this dissertation, however, we assume that the bandwidths for all dimensions are the same, $\vec{h} = \{h, h \ldots h\}$, and simply use the scalar h to represent the bandwidth.

LWR calculates a new model for every new query point. This differs from standard regression techniques, which calculate one global model for all possible queries. Although this allows LWR to model more complex functions, it also means that it is significantly more computationally expensive than a single global regression. One way to think about how LWR works is shown in figure 4.2. A standard linear regression attaches springs between the data points and the fitted line. All of the springs have the same strength, and the line shown in figure 4.2(a) is the result. This line is used to answer all queries. Locally weighted regression, on the other hand, fits a new line for each query. It does this by measuring the distance between the query point and each data point, and making the strength of the springs vary with distance. Springs closer to the query point are stronger, those further away are weaker, according to the kernel function. This means that points close to the query have more influence on how the fitted line is placed. This can be seen in figure 4.2(b), where the same data points generate a different fitted line. This line is *only* used for the particular query point shown. For another query, the spring strengths will be different and a different line will be fit to the data.

Algorithm 1 uses two functions, one to calculate the distance from training points to query points, and one to turn this distance into a weighting for the points. In the following sections we look at these functions more closely, and then go on to discuss why LWR actually works.



(a) Standard linear regression



(b) Locally Weighted regression

Figure 4.2: Fitting a line with linear regression and locally weighted regression.

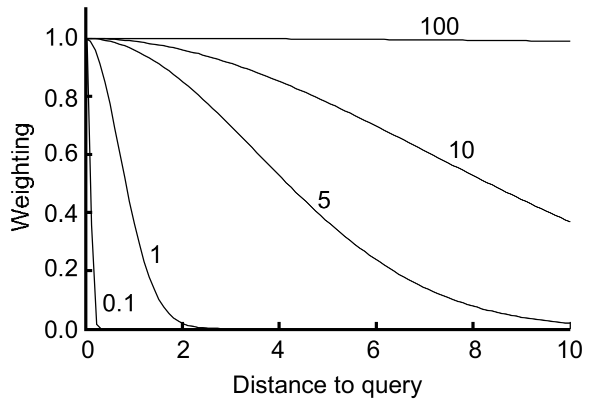


Figure 4.3: Gaussian kernel functions with different bandwidths.

4.5.1 LWR Kernel Functions

The kernel function, used on line 3 of algorithm 1, is used to turn a distance measure into a weighting for each training point. It is parameterized by a bandwidth, h, which controls how quickly the influence of a point drops off with distance from $\vec{x}_q$. A Gaussian,

$$\text{kernel}\,(d, h) = e^{-\left(\frac{d}{h}\right)^2},$$

is typically used, where d is the distance to the query point and h is the bandwidth. Other kernel functions, such as step functions, are also possible. The only restrictions on the form of kernel functions are

$$\begin{aligned} &\text{kernel}\,(0, h) = 1 \\ &\text{kernel}\,(d_1, h) \geq \text{kernel}\,(d_2, h)\,, && d_1 \leq d_2 \\ &\text{kernel}\,(d, h) \rightarrow 0, && \text{as } d \rightarrow \infty \end{aligned} \quad .$$

The actual shape of the kernel has some effect on the results of the local regression, but these effects are typically small for a wide class of kernel functions [109]. We use a Gaussian kernel because it has infinite extent, and will never assign a zero weight to any data point. For kernel functions which go to zero, it is possible for all training points to be far enough away from the query that they are all weighted to zero.

Figure 4.3 shows some example Gaussian kernels and their bandwidths. The bandwidth is a measure of how smooth we think the function to be modeled is. Its value has a significant effect on the outcome of the regression. Large bandwidths produce very smoothed functions, where all of the training points have a similar effect

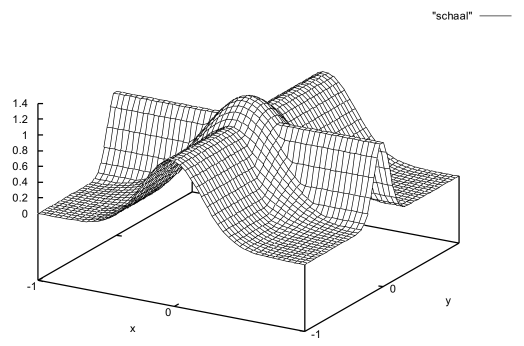


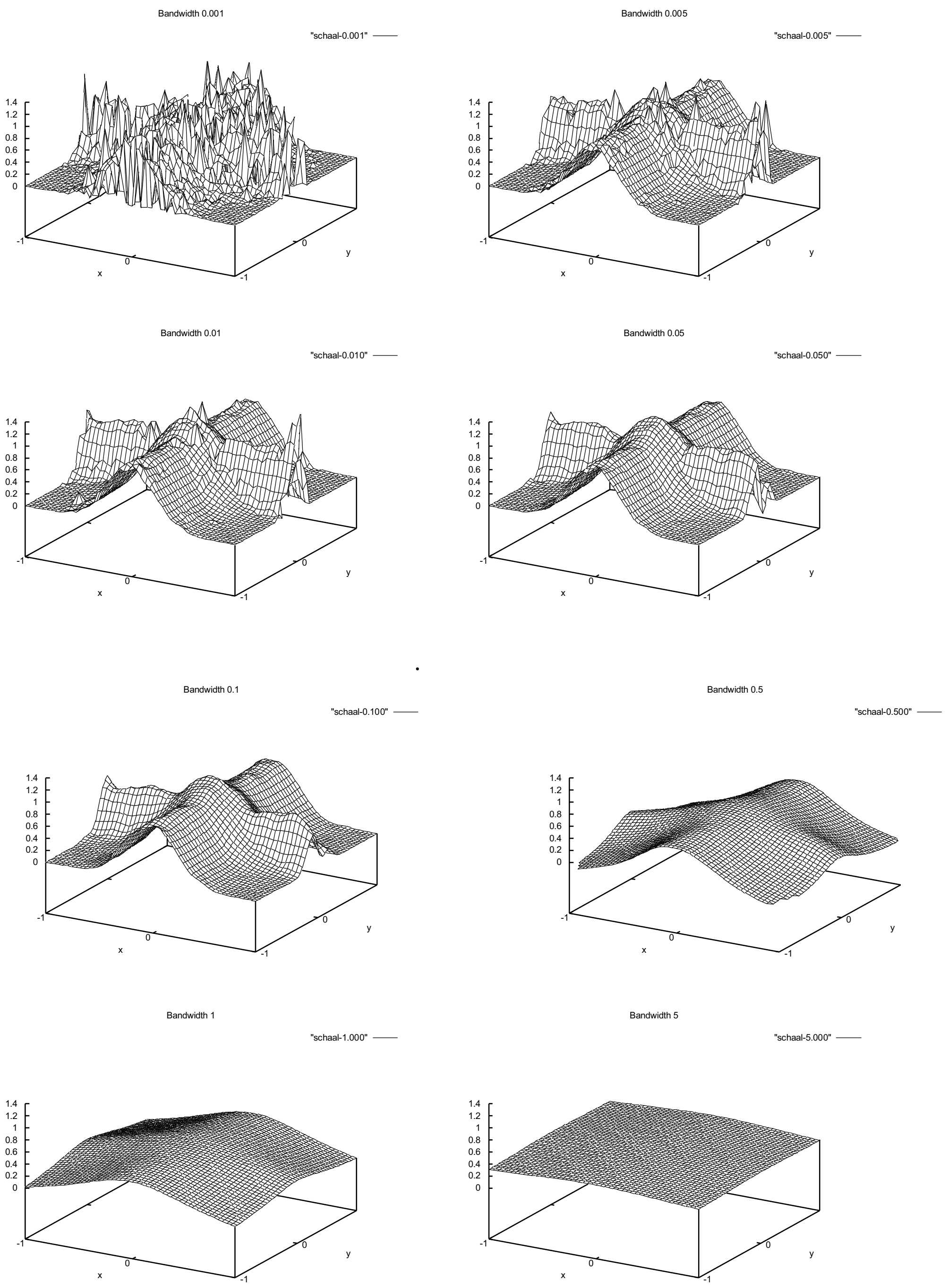
Figure 4.4: Function used by Schaal and Atkeson.

on the regression. Small bandwidths tend towards nearest neighbor predictions. As $h \rightarrow \infty$ the local regression tends towards a global regression, since each training point will have a weight of 1.

We now give an example to illustrate the effects of varying the bandwidth in LWR. The function that we are attempting to learn is one used previously in work by Schaal and Atkeson [92]. The function is shown in figure 4.4 and is given by

$$f(x,y) = \max\left\{\exp\left(-10x^2\right), \exp\left(-50y^2\right), 1.25\exp\left(-5\left(x^2+y^2\right)\right)\right\}.$$

The function consists of two ridges, one narrow and one wider, and a Gaussian bump at the origin. In all of the following examples, we use 500 training points drawn from the input space according to a uniform random distribution. Figure 4.5 shows the models learned with varying bandwidth settings. With very small bandwidths, all points are assigned weights close to zero. This leads to instabilities in the local regressions, and results in a poor fit. With large bandwidths the local regressions take on the character of global regressions since every point is weighted close to 1. Somewhere between these two extremes lies a good choice of bandwidth, where the function is well modeled. Note that in figure 4.5 the function is never particularly well modeled, partly because of the relatively small number of training points and partly because the optimal bandwidth is not shown. Getting a good model of the function hinges on finding this “best” bandwidth, and we will discuss this further in section 4.7.2.



eps

Figure 4.5: LWR approximations using different bandwidth values.

4.5.2 LWR Distance Metrics

LWR uses the distance from the query point to each of the training points as the basis for assigning weights to those points. This means that the choice of distance metric has a large effect on the predictions made. Typically, Euclidean distance is used

$$\text{distance}\left(\vec{p}, \vec{q}\right) = \sqrt{\sum_{i=0}^{n} w_i^2 \left(p_i - q_i\right)^2}$$

where w_i is a weight applied to each dimension. If this weight is 1, the standard Euclidean distance is returned. This implicitly assumes that all elements of the vectors being compared have ranges of approximately equal width. If this is not the case, different weights can be used for each dimension. This is equivalent to normalizing all of the ranges before computing the distance. The weights can also be used to emphasize dimensions that are more important than others, and cause them to have more effect on the distance calculation. It should be noted that this distance metric is designed for continuous quantities does not deal well with nominal values.

An alternative distance metric is the Minkowski metric [15, pp 71–72], given by

$$\text{distance}\left(\vec{p}, \vec{q}\right) = \left(\sum_{i=0}^{n} \left|x_i - y_i\right|^r\right)^{\frac{1}{r}}.$$

Euclidean distance is simply the Minkowski distance with $r = 2$. The Manhattan, or city-block, distance is Minkowski with $r = 1$. As r increases, points that are far from each other (in the Euclidean sense) get further away. When viewed in terms of kernel functions, as r increases, far away points are weighted closer and closer to zero. Other, more complex, distance functions are also possible; Wilson [115, chapter 5] gives a summary in terms of instance-based learning algorithms.

For the work reported in this dissertation, we will use standard Euclidean distance as our metric. Although it might not be the best metric in every case, we have found empirically that it gives reasonable results. It also has the advantage of being easily interpretable, which will proved to be useful when debugging and testing the algorithms described in this section.

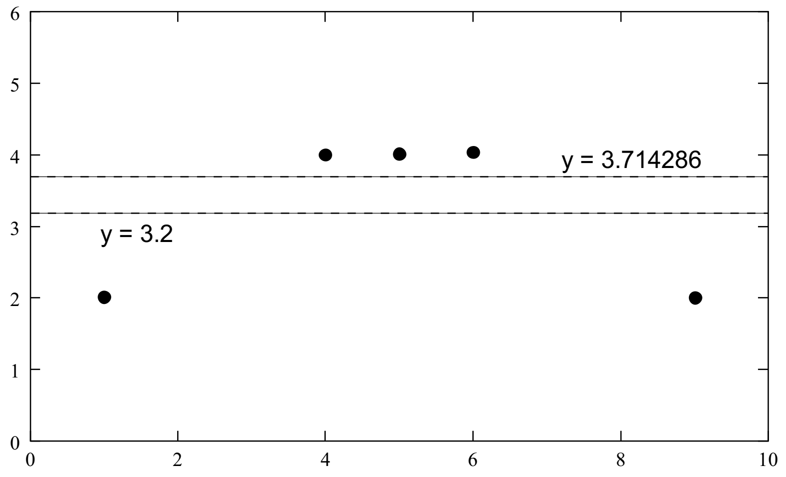


Figure 4.6: Fitted lines using LWR and two different weights for extreme points.

4.5.3 How Does LWR Work?

How does LWR manage to focus attention around the query point and disregard points that are far from it? A standard linear regression fits a line to a set of data points by minimizing the squared prediction error for each data point,

$$\sum_{i=1}^{n} (y_i - \hat{y}_i)^2 ,$$

where (x_i, y_i) is a training point, and $\hat{y}_i$ is the prediction for that point. LWR applies a weight, k_i, to each of the training points based on their distance from the query point. Points far away have smaller weights, points closer in have larger weights. This means that we are now minimizing a weighted sum

$$\sum_{i=1}^{n} (w_i y_i - w_i \hat{y}_i)^2 = w_1^2 (y_1 - \hat{y}_1) + w_2^2 (y_2 - \hat{y}_2) + \ldots + w_n^2 (y_n - \hat{y}_n) .$$

Points that have small weights will not play less of a part in the sum of errors, and will thus have less effect on the final fitted line.

Figure 4.6 illustrates this with a simple example. There are five data points, $(1, 2)$, $(4, 4)$, $(5, 4)$, $(6, 4)$, and $(9, 2)$. The result of a standard linear regression, where all points have the same weight, is the line $y = 3.2$. If weight the first and last points half as much as the other three, the line changes to $y = 3.714286$. This reflects the greater influence of the three middle points. Figure 4.7 shows how the intercept value changes with the relative weights of the two extreme data points. With a weight of