CHAPTER 12

Fitting Lines

Again and again, one needs to choose the "best" parametric model to explain or represent some data. A simple model case is choosing a line to represent some tokens. These tokens could be edge points, or they could be the location of some interest points, or they could be other things entirely, but they are represented by their location. A more interesting case is choosing some lines to represent a set of tokens – there may be more than one line, and you may need to determine how many lines there are. Another very important case, dealt with in some detail in Chapter **??**, might look different to you but isn't really. You are given a set of points $\mathbf{x}_i$ in one image and a set of points $\mathbf{y}_i$ in another, and must find a transformation $\mathcal{T}$ so that $\mathcal{T}(\mathbf{x}_i)$ is close to $\mathbf{y}_i$. Each of these problems is usually referred to as *fitting*.

12.1 FITTING JUST ONE LINE

Fitting just one line to a set of tokens is a good model problem. Assume that all the points that belong to a particular line are known, and the parameters of the line must be found.

12.1.1 Least Squares Fitting

There is a simple strategy for fitting lines, known as *least squares*. *Do not use this* - I am showing it only because it is traditional, and because the form of the derivation is repeated for other problems. This procedure has a long tradition and a substantial bias. You have likely seen this idea, but may not know why you should not use it. Represent a line as $y = ax + b$. There are N tokens, and the i'th token is at $\mathbf{x}_i = (x_{i,1}, y_{i,2})$. Now choose the line that best predicts the measured y coordinate for each measured x coordinate, and so minimizes

$$(1/2)\sum_i (x_{i,2} - ax_{i,1} - b)^2.$$

The term $(1/2)$ very often appears in quadratic problems like this to absorb the 2 that appears when you differentiate. A certain looseness in notation is traditional (you might not see either), because the 2 appears in front of an expression that is equal to zero. Adopting the notation

$$\overline{u} = \frac{\sum u_i}{N}$$

the line is given by the solution to the problem

$$\begin{pmatrix} \overline{x_2^2} \\ \overline{x_2} \end{pmatrix} = \begin{pmatrix} \overline{x_1^2} & \overline{x_1} \\ \overline{x_1} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \end{pmatrix}.$$

Although this is a standard linear solution to a classical problem, it's not much help in vision applications, because the model is an extremely poor one. The difficulty

is that the measurement error is dependent on coordinate frame. Vertical offsets from the line count as errors, which means that near vertical lines lead to quite large values of the error and quite funny fits (Figure 12.1). In fact, the process is so dependent on coordinate frame that it doesn't represent truly vertical lines at all.

If the errors are weighted with weight w_i corresponding to the i'th term, the cost becomes

$$(1/2)\sum_i w_i(x_{2,i} - ax_{1,i} - b)^2.$$

and, if you interpret

$$\overline{u} = \frac{\sum w_i u_i}{\sum w_i}$$

the expression above still works.

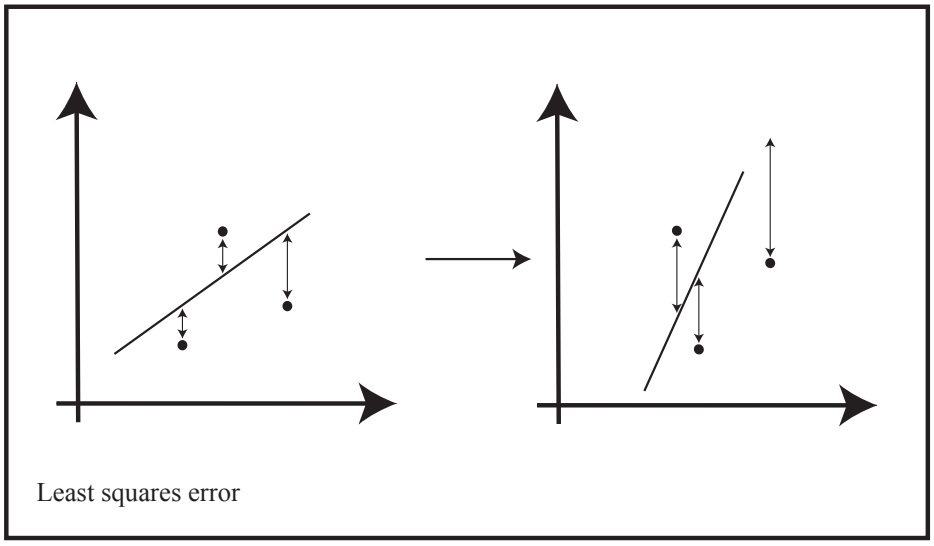

FIGURE 12.1: *Least squares finds the line that minimizes the sum of squared vertical distances between the line and the tokens (because it assumes that the error appears only in the y coordinate). This yields a (very slightly) simpler mathematical problem at the cost of a poor fit. Importantly, if the coordinate system rotates (say, the camera is rotated), the error between the rotated points and the rotated lines changes.*

12.1.2 Total Least Squares Fitting

Working with the actual distance between the token and the line (rather than the vertical distance) leads to a problem known as *total least squares*. Write $\mathbf{a} = (a_1, a_2)^T$ and represent a line as the collection of points where $\mathbf{a}^T\mathbf{x} + c = 0$. Require

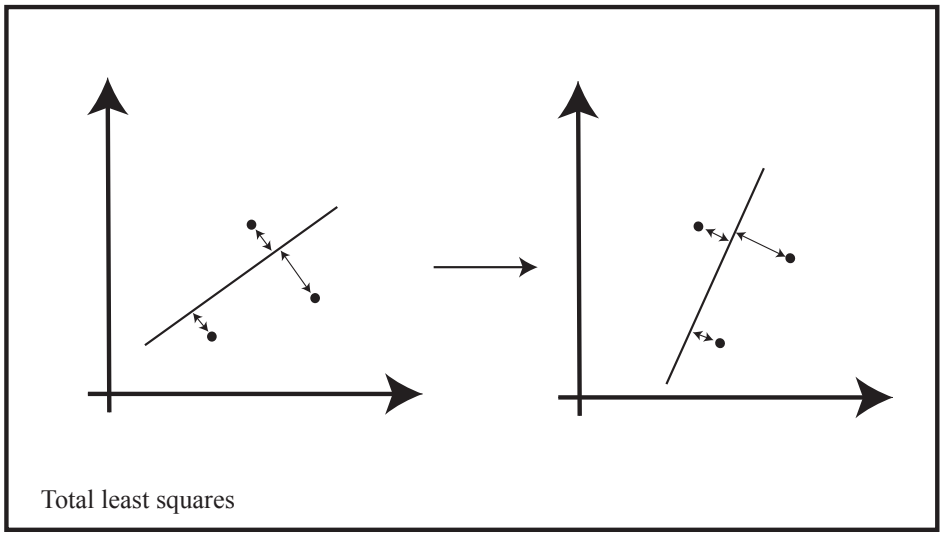

FIGURE 12.2: *Total least-squares finds the line that minimizes the sum of squared perpendicular distances between tokens and the line so near-vertical lines can be fitted without difficulty. If the coordinate system rotates (say, the camera is rotated), the error between the rotated points and the rotated lines does not change.*

that $\mathbf{a}^T\mathbf{a} + c^2 \neq 0$, otherwise the equation is true for all points on the plane. Every line can be represented in this way.

Notice that for $s \neq 0$, the line given by $s(\mathbf{a}, c)$ is the same as the line represented by $(\mathbf{a}, c)$. In the exercises, you are asked to prove the simple, but extremely useful, result that the perpendicular distance from a point $\mathbf{x}$ to a line $(\mathbf{a}, c)$ is given by

$$\text{abs}(\mathbf{a}^T\mathbf{x} + c) \qquad \text{if} \qquad \mathbf{a}^T\mathbf{a} = 1.$$

In my experience, this fact is useful enough to be worth memorizing.

It is convenient to assume that each point has an associated weight w_i. In the case where you have no idea what the weights are, use $w_i = 1$. Write Recall

$$\overline{u} = \frac{\sum_i w_i u_i}{\sum_i w_i}$$

Minimizing the sum of perpendicular distances between points and lines requires minimizing

$$(1/2) \sum_i w_i (\mathbf{a}^T\mathbf{x}_i + c)^2,$$

subject to $\mathbf{a}^T\mathbf{a} = 1$. Introduce a Lagrange multiplier λ; at a solution

$$\begin{pmatrix} \overline{x_1^2} & \overline{x_1 x_2} & \overline{x_1} \\ \overline{x_1 x_2} & \overline{x_2^2} & \overline{x_2} \\ \overline{x_1} & \overline{x_2} & 1 \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \\ c \end{pmatrix} = \lambda \begin{pmatrix} 2a_1 \\ 2a_2 \\ 0 \end{pmatrix}.$$

This means that

$$c = -a_1\overline{x_1} - a_2\overline{x_2}.$$

Substitute this back to get the eigenvalue problem

$$\begin{pmatrix} \overline{x_1^2} - \overline{x_1}\,\overline{x_1} & \overline{x_1x_2} - \overline{x_1}\,\overline{x_2} \\ \overline{x_1x_2} - \overline{x_1}\,\overline{x_2} & \overline{x_2^2} - \overline{x_2}\,\overline{x_2} \end{pmatrix} \begin{pmatrix} a_1 \\ a_2 \end{pmatrix} = \mu \begin{pmatrix} a_1 \\ a_2 \end{pmatrix}.$$

Because this is a 2D eigenvalue problem, two solutions up to scale can be obtained in closed form (for those who care, it's usually done numerically!). The scale is obtained from the constraint that $\mathbf{a}^T\mathbf{a} = 1$. The two solutions to this problem are lines at right angles; one maximizes the sum of squared distances and the other minimizes it (which is which is explored in exercises **exercises**).

Procedure: 12.1 *Fitting a Line using Weighted Total Least Squares*

Given N points $\mathbf{x}_i = (x_{i,1}, x_{i,2})$, and weights w_i associated with each point, find a line that minimizes the weighted total least squares error by forming

$$\begin{aligned} \overline{x_1} &= \frac{\sum_i w_i x_{i,1}}{\sum_i w_i} \\ \overline{x_2} &= \frac{\sum_i w_i x_{i,1}}{\sum_i w_i} \\ \overline{x_1^2} &= \frac{\sum_i w_i x_{i,1}^2}{\sum_i w_i} \\ \overline{x_1x_2} &= \frac{\sum_i w_i x_{i,1}x_{i,2}}{\sum_i w_i} \\ \overline{x_2^2} &= \frac{\sum_i w_i x_{i,2}^2}{\sum_i w_i} \end{aligned}$$

and

$$\mathcal{M} = \begin{pmatrix} \overline{x_1^2} - \overline{x_1}\,\overline{x_1} & \overline{x_1x_2} - \overline{x_1}\,\overline{x_2} \\ \overline{x_1x_2} - \overline{x_1}\,\overline{x_2} & \overline{x_2^2} - \overline{x_2}\,\overline{x_2} \end{pmatrix}.$$

Write $\mathbf{e} = (e_1, e_2)^T$ for the unit eigenvector of $\mathcal{M}$ corresponding to the smallest eigenvalue. The line is

$$e_1x + e_2y - e_1\overline{x_1} - e_2\overline{x_2} = 0$$

12.1.3 Application: Surface Normals in Point Clouds

A *point cloud* is a set of points with no other structure. The idea is developed in greater detail in Chapter **??**. Usually, point clouds are obtained by sampling surfaces in some way – the points lie on objects in scenes, and are close enough

that most points are on the same object as their closest neighbors. This means that you could associate a normal with each point fairly accurately by: finding a few (at least two) nearest neighbors; fitting a plane to the point and its neighbors; and using the normal of that plane.

You can find the nearest neighbors either by blank search (check the distance to every other point, and take the points with the smallest; which is slow) or by using the approximate nearest neighbors method of Section 10.2.1. Fitting a plane to points in 3D is very similar to fitting a line to points in 2D. As for lines, it is not a good idea to represent the plane as $z = ux + vy + w$ and apply least squares. This will be biased and will not represent vertical planes well. Total least squares is a better strategy, just as in line fitting. Represent the plane as $\mathbf{a}^T\mathbf{x} + d = 0$; then the squared distance from a point $\mathbf{x}_i = (x_{i,1}, x_{i,2}, x_{i,3})$ to the plane will be $(\mathbf{a}^T\mathbf{x} + d)^2$ if $\mathbf{a}^T\mathbf{a} = 1$. Now use the analysis above with small changes (the closed form expression for the three solutions is more complicated)..

Procedure: 12.2 *Fitting a Plane using Weighted Total Least Squares*

Given N points $\mathbf{x}_i = (x_{i,1}, x_{i,2})$, and weights w_i associated with each point, find a plane that minimizes the weighted total least squares error by forming

$$\begin{aligned}
\overline{x_1} &= \frac{\sum_i w_i x_{i,1}}{\sum_i w_i} \\
\overline{x_2} &= \frac{\sum_i w_i x_{i,1}}{\sum_i w_i} \\
\overline{x_3} &= \frac{\sum_i w_i x_{i,1}}{\sum_i w_i} \\
\overline{x_1^2} &= \frac{\sum_i w_i x_{i,1}^2}{\sum_i w_i} \\
\overline{x_1 x_2} &= \frac{\sum_i w_i x_{i,1} x_{i,2}}{\sum_i w_i} \\
\overline{x_2^2} &= \frac{\sum_i w_i x_{i,2}^2}{\sum_i w_i} \\
\overline{x_2 x_3} &= \frac{\sum_i w_i x_{i,2} x_{i,3}}{\sum_i w_i} \\
\overline{x_3^2} &= \frac{\sum_i w_i x_{i,3}^2}{\sum_i w_i}
\end{aligned}$$

and

$$\mathcal{M} = \begin{pmatrix}
\overline{x_1^2} - \overline{x_1}\,\overline{x_1} & \overline{x_1 x_2} - \overline{x_1}\,\overline{x_2} & \overline{x_1 x_3} - \overline{x_1}\,\overline{x_3} \\
\overline{x_2 x_1} - \overline{x_2}\,\overline{x_1} & \overline{x_2^2} - \overline{x_2}\,\overline{x_2} & \overline{x_2 x_3} - \overline{x_2}\,\overline{x_3} \\
\overline{x_3 x_1} - \overline{x_3}\,\overline{x_1} & \overline{x_3 x_2} - \overline{x_3}\,\overline{x_2} & \overline{x_3^2} - \overline{x_3}\,\overline{x_3}
\end{pmatrix}.$$

Write $\mathbf{e} = (e_1, e_2, e_3)^T$ for the unit eigenvector of $\mathcal{M}$ corresponding to the smallest eigenvalue. The plane is

$$e_1 x + e_2 y + e_3 z - e_1\overline{x_1} - e_2\overline{x_2} - e_3\overline{x_3} = 0$$

12.1.4 Fitting Curved Shapes

Curves in 2D are different from lines in 2D. For every token on the plane, there is a unique, single point on a line that is closest to it. This is not true for a curve. Because curves curve, there might be more than one point on the curve that looks locally as though it is closest to the token (Figure 12.3). This means it can be very hard to find the smallest distance between a point and a curve. Similar effects occur for surfaces in 3D. If one ignores this difficulty, fitting curves is similar to fitting lines: minimize the sum of squared distances between the points and the

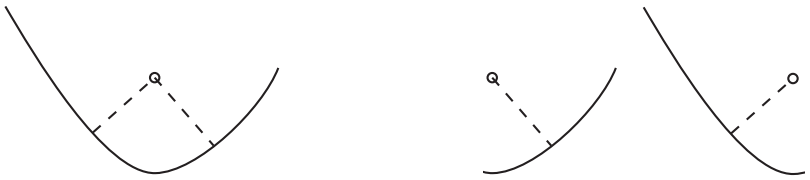

FIGURE 12.3: *There can be more than one point on a curve that looks as if it is closest to a token. This makes fitting curves to points very difficult. On the* **left**, *a curve and a token; dashed lines connect the token to the two points on the curve that could be closest. The dashed line from token to curve and the tangent to the curve are at right angles for each of the two points on the curve.* **Center** *and* **right** *show copies of part of the curve; for each, the closest point on the segment to the token is different, because part of the curve is missing. As a result, no local test can guarantee a point is closest – all candidates must be checked.*

curve as a function of the choice of curve. The exercises explore how to find the closest point on a curve (**exercises**), but generally these fitting problems are very badly behaved.

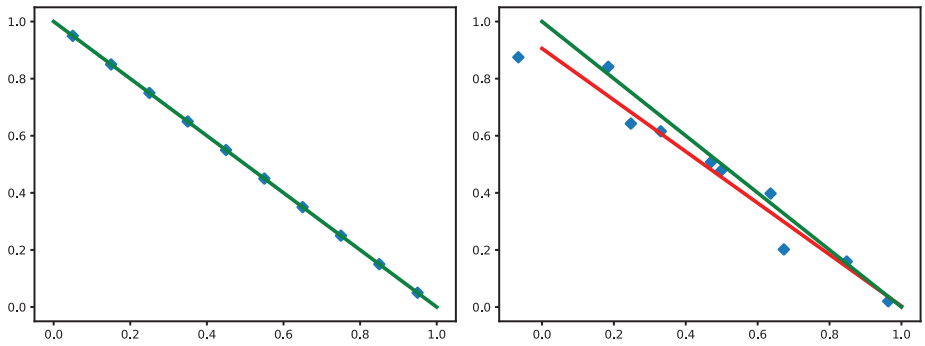

FIGURE 12.4: *Total least squares behaves well when all points lie fairly close to the line.* **Left***: A line fitted with total least squares to a set of points that actually lie on the line.* **Right***: The points have been perturbed by adding a small random term to both x and y coordinate, but the fitted line (red) is not that different from the original (green).*

12.2 FITTING ONE LINE IN THE PRESENCE OF OUTLIERS

All of the line fitting methods described involve squared error terms. This is fine when no point is far from the line (Figure 12.4), but a single very bad data point can give errors that dominate those due to many good data points (Figure 12.5). This is because the square of a large number is very much larger than the square

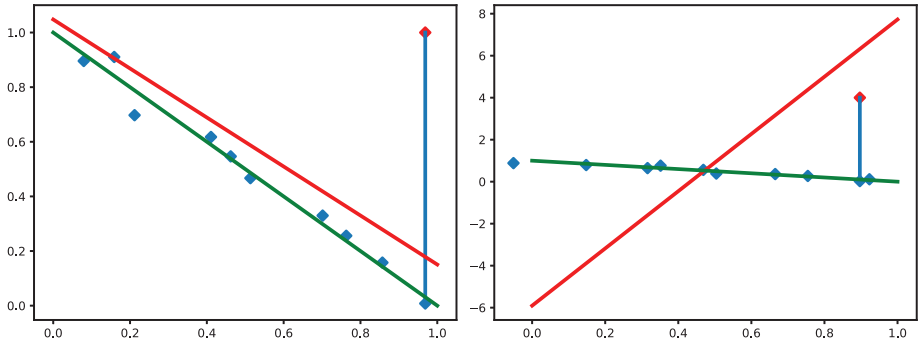

FIGURE 12.5: *Total least squares can misbehave when some points are significantly perturbed. Here the points are those of Figure 12.4, but one point has had its y coordinate replaced with a constant (new point in red, joined to its original position with a line).* **Left:** *The constant is 1, and the line produced by total least squares has shifted significantly.* **Right:** *The constant is 4, and now the fitted line is completely wrong. In each case, the original line is in green (and passes close to the points), and the new line is red.*

of a small number.

Outliers – data points that are far from the model, in this case, the line – are common. If the set of tokens form several lines – imagine a set of tokens on the edges of a square – a line that fits some points (say, those on one side of the square) will have tokens far away from it (those on the other sides). Another source of outliers is errors in collecting or transcribing data points . Practical vision problems usually involve outliers.

Outliers can be handled by reducing the influence of distant points on the estimate. This is where the weights of the previous section come in useful. A natural strategy fits a line using a set of weights, then recomputes the weights using a large weight for points that are "trustworthy" and a small weight for points that are "suspicious". This is the basis of IRLS (iteratively reweighted least squares), a procedure that uses a robust cost function to organize the reweighting of points. Alternatively you could identify good points and ignoring them. A small set of good points will identify the thing we wish to fit; other good points will agree, and the points that disagree are then bad points. This is the basis of RANSAC, an extremely important algorithm described in Section 44.2.2.

12.2.1 Robust Cost Functions

The weights are not known, but can be recovered using a *robust loss*, which reduces the cost of large errors. Typically, these cost functions "look like" squared error for small values, but grow more slowly for distant points.

Write θ for the parameters of the line and $r(\mathbf{x}_i, \theta)$ for a *residual*, which measures consistency between the point and the line. In this case, $r(\mathbf{x}_i, \theta)$ will always

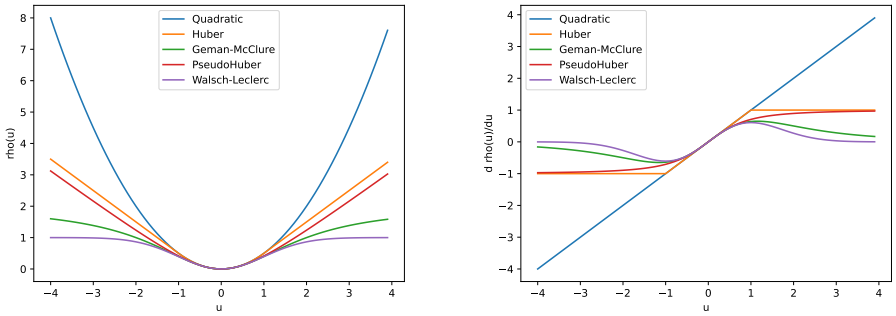

FIGURE 12.6: *Robust losses reduce the effect of large errors.* **Left***: the robust loss functions of the text, compared to a squared error loss. In each case, the scale variable σ is 1. Note how for each of the robust losses, the cost of large errors is reduced compared to quadratic losses.* **Right:** *the derivative for each of these loss functions; this derivative appears in the weight term in IRLS, and is sometimes referred to as the* influence function *of the loss. For losses like Huber and pseudo-Huber loss, large errors still have an influence (the derivative heads toward a non-zero constant); but for other losses, large errors can cause a data point to be completely discounted (the derivative is zero, so the weight in IRLS will be zero).*

be $(\mathbf{a}^T\mathbf{x} + c)$, which is signed, so positive on one side of the line and negative on the other. Rather than minimizing the total least squares criterion, which is

$$(1/2)\sum_i (r(\mathbf{x}_i, \theta))^2$$

as a function of θ, choose the line by minimizing an expression of the form

$$\sum_i \rho(r(\mathbf{x}_i, \theta); \sigma),$$

for some appropriately chosen function ρ with a parameter σ. The total least squares error fits this recipe, using $\rho(u; \sigma) = u^2/2$. The trick is to make $\rho(u; \sigma)$ look like $u^2/2$ for smaller values of u, but ensure that it grows more slowly than $u^2/2$ for larger values of u.

The *Huber loss* uses

$$\rho(u; \sigma) = \begin{cases} \frac{u^2}{2} & |u| < \sigma \\ \sigma|u| - \frac{\sigma^2}{2} & \end{cases}$$

which is the same as $u^2/2$ for $-\sigma \leq u \leq \sigma$, switches to $|u|$ for larger (or smaller) σ, and has continuous derivative at the switch. The Huber loss is convex (meaning that there will be a unique minimum for our models) and differentiable, but is not smooth. The choice of the parameter σ (which is known as *scale*) has an effect on the estimate. You should interpret this parameter as the distance that a point can

lie from the fitted function while still being seen as an *inlier* (anything that isn't even partially an outlier).

The *Pseudo Huber loss* uses

$$\rho(u;\sigma) = \sigma^2\left(\sqrt{1+\left(\frac{u}{\sigma}\right)^2}-1\right).$$

A little fiddling with Taylor series reveals this is approximately u^2 for $|u|/\sigma$ small, and linear for $|u|/\sigma$ big. This has the advantage of being differentiable. The *Geman-McClure loss* uses

$$\rho(u;\sigma) = \frac{2\left(\frac{u}{\sigma}\right)^2}{\left(\frac{u}{\sigma}\right)^2+4} = \frac{2u^2}{u^2+4\sigma^2}$$

which is approximately $(1/2)u^2$ for $|u|$ much smaller than σ, and close to $(1/2)\sigma^2$ for $|u|$ much larger than σ.

The *Geman-McClure loss* uses

$$\rho(u;\sigma) = \frac{2\left(\frac{u}{\sigma}\right)^2}{\left(\frac{u}{\sigma}\right)^2+4} = \frac{2u^2}{u^2+4\sigma^2}$$

which is approximately $(1/2)u^2$ for $|u|$ much smaller than σ, and close to $(1/2)\sigma^2$ for $|u|$ much larger than σ.

The *Welsch loss* (or *LeClerc loss*) uses

$$\rho(u;\sigma) = 1 - e^{\left[-\frac{1}{2}\left(\frac{u}{\sigma}\right)^2\right]}$$

notice this looks like $(1/2)u^2$ for $|u|$ much smaller than σ, and close to 1 for larger u. All of these losses increases monotonically in $|u|$ (the absolute value is important here!), so it is always better to reduce the residual.

12.2.2 IRLS: Weighting Down Outliers

Now the line is chosen by minimizing

$$(1/2)\sum_i \rho(r(\mathbf{x},\theta);\sigma)$$

with respect to $\theta = (a_1, a_2, c)$, subject to $a_1^2 + a_2^2 = 1$. The minimum occurs when

$$\begin{aligned}\nabla_\theta\left(\sum_i \rho(r(\mathbf{x}_i,\theta);\sigma)\right) &= \sum_i\left[\frac{\partial\rho}{\partial u}\right]\nabla_\theta r(\mathbf{x}_i,\theta)\\ &= \lambda\begin{pmatrix}a_1\\a_2\\0\end{pmatrix}.\end{aligned}$$

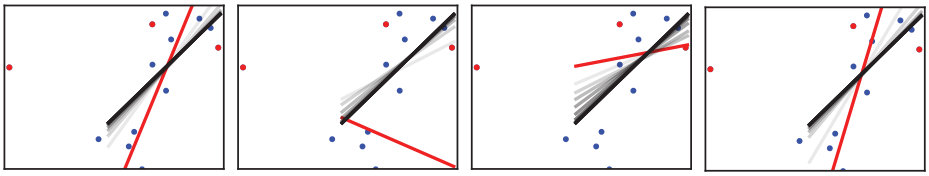

FIGURE 12.7: *Robust losses can control the influence of outliers.* **Blue** *points lie on a line, and have been perturbed by noise;* **red** *points are outliers. The* **red** *line shows a starting line, obtained by drawing a small random sample from the dataset, then fitting a line; the* **gray** *lines show iterates of IRLS applied to a Huber loss (later iterates are more opaque; scales are estimated as in the text). The procedure converges from a range of start points, some quite far from the "true" line. Notice how each start point results in the same line.*

Here λ is a Lagrange multiplier and the derivative $\frac{\partial \rho}{\partial u}$ is evaluated at $r(\mathbf{x}_i, \theta)$, so it is a function of θ. Now notice that

$$\begin{aligned}\sum_i \left[\frac{\partial \rho}{\partial u}\right] \nabla_\theta r(\mathbf{x}_i, \theta) &= \sum_i \left[\left(\frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \theta)}\right)\right] r(\mathbf{x}_i, \theta) \nabla_\theta r(\mathbf{x}_i, \theta) \\ &= \sum_i \left[\left(\frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \theta)}\right)\right] \nabla_\theta \left[(1/2) r(\mathbf{x}_i, \theta)\right]^2\end{aligned}$$

Now $[r(\mathbf{x}_i, \theta)]^2$ is the squared error. At the true minimum $\hat{\theta}$, writing

$$w_i = \left(\frac{\frac{\partial \rho}{\partial u}}{r(\mathbf{x}_i, \hat{\theta})}\right)$$

(where the derivatives are evaluated at that $\hat{\theta}$), then

$$\sum_i w_i \nabla_\theta \left[r(\mathbf{x}_i, \theta)\right]^2 = \lambda \begin{pmatrix} 2a_1 \\ 2a_2 \\ 0 \end{pmatrix}.$$

at $\theta = \hat{\theta}$. The weights w_i are unknown; if they were known, this could be solved (Section 7.2.1). However, for any given line, it is straightforward to estimate the weights, which you should think of as a discount that is applied to the squared error – so weights are large for points that are close to the line, and small for points that are far.

The parameter σ is often referred to as *scale*. In some cases, σ can be chosen from application logic. For example, if you are fitting lines to edges in pictures of buildings and you know that the pictures are fairly sharp and have little pixel noise, you can expect σ to be small. Alternatively, you can estimate σ from data. If the true line was known, then the values of $r(\mathbf{x}_i, \theta)$ would be "small" for inliers and "large" for outliers. Assuming there aren't too many outliers, then a fair and

widely used estimate of scale at the n'th iteration is

$$\sigma^{(n)} = 1.4826 \text{ median}_i \, |r(x_i; \theta^{(n-1)})| \, .$$

A natural strategy to adopt is to start with an estimate of the line, recover weights from that, then repeat:

- **Estimate the line** using the weights.
- **Estimate** σ using the new line and the points.
- **Re-estimate weights** using the new line and the new σ.

This procedure is an instance of a general recipe known as *iteratively reweighted least squares*. More instances of this recipe appear in Section **??**.

Procedure: 12.3 *Fitting a Line using Iteratively Reweighted Least Squares*

This procedure takes a set of N points $(x_{i,1}, x_{i,2})$ putatively lying on a line and obtains an estimate of that line.
Starting: Obtain an initial line $(a^{(1)}, b^{(1)}, c^{(1)})$ with $(a^{(1)})^2+(b^{(1)})^2 = 1$ and an initial scale $\sigma^{(1)}$. One strategy to obtain an initial line is to draw a small set of points uniformly at random from the data, and apply total least squares. The scale may need to come from application considerations. Choose a robust cost function $\rho(u;\sigma)$ from Section 12.2.1 or somewhere else. Form an initial estimate of weights using

$$\begin{aligned} r_i^{(1)} &= a^{(1)}x_{i,1} + b^{(1)}x_{i,2} + c^{(1)} \\ w_i^{(1)} &= \left(\frac{d\rho}{du}\right) / \left(r_i^{(i)}\right) \end{aligned}$$

where the derivative is evaluated at $u = r_i^{(i)}$ and $\sigma^{(1)}$. Now use iterate three steps:

Estimate the line using weighted total least squares (Procedure 12.1) to obtain $(a^{(r+1)}, b^{(r+1)}, c^{(r+1)})$ and $r_i^{(r+1)}$ from $w_i^{(r)}$.

Estimate the scale, possibly using

$$\sigma^{(n)} = 1.4826 \text{ median}_i \left| r_i^{(r+1)} \right| .$$

Alternatively, use a fixed scale obtained using application considerations.

Re-estimate weights using

$$w_i^{(r+1)} = \left(\frac{d\rho}{du}\right) / \left(r_i^{(i)}\right)$$

where the derivative is evaluated at $u = r_i^{(r+1)}$ and $\sigma^{(r+1)}$.

Terminate iterations when either the change in the line is below a threshold or there have been too many.

12.2.3 Problems with Robust Losses

Each of these robust losses has a long record of being useful and helpful, but none is ideal. One important difficulty with IRLS one must start somewhere, and a bad start will result in a bad line. This can be controlled by using a number of randomly chosen lines to start – perhaps obtained by selecting some points at random, and fitting a line to them – then choosing the best line that results. As Figure 12.8

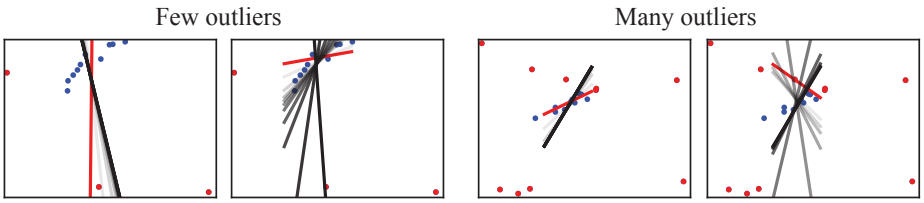

FIGURE 12.8: *Robust losses can fail, particularly when distant points still have some weight or if there are many outliers.* **Left***: a bad start point leads to a bad line;* **center left***: on the same data set, quite a good start point still converges to a bad line. Here there are few outliers, but they are far from the data and they contribute a significant weight to the loss. When there are many outliers, this effect worsens. Because each outlier still contributes a signficant weight to the loss, even a good start fails (***center right***). A poor start (***right***) also fails, and produces the same line as the good start – in fact, most starts end up close to this line. Again,* **blue** *points lie on a line, and have been perturbed by noise;* **red** *points are outliers; the* **red** *line shows a starting line, obtained by drawing a small random sample from the dataset, then fitting a line; the* **gray** *lines show iterates of IRLS (later iterates are more opaque).*

indicates, this isn't always a reliable approach. If the loss places some weight on distant points and the outliers are far away or there are many of them, then the iteration can settle on a bad line *even if* the initial line is good. The alternative – using a loss that wholly deemphasizes distant points, like the Welsch loss – means you may need to use an awful lot of start points. Robust losses are at their best when there are many data points and few outliers, and you have a reliable way of starting the fitting procedure.

12.3 RANSAC: SEARCHING FOR GOOD POINTS

Imagine you must fit a line to a dataset that consists of about 50% outliers. If you draw pairs of points uniformly and at random, then about a quarter of these pairs will consist entirely of good data points. If the dataset is large, then a draw of k pairs will have a probability of 0.75^k of containing only bad pairs, so you can compute a k big enough to guarantee that you see at least one good pair with probability $1 - \epsilon$. Any pair of points yields a line. A good pair will have about half of the other points lying close to that line, so you can tell if you have seen a good pair. You now have an algorithm: compute k so that you have a satisfactory probability of seeing at least one good pair; repeatedly choose a small subset of points at random; fit to that subset; compute the "goodness" of that fit, and if it is better than previous fits, keep it. Doing this enough times will result in the right fit, with a high probability.

It should be clear that this approach applies to fitting other structures, though you may need to draw more points (for example, you need to draw three points in 3D to fit a plane; three points in 2D to fit a circle; and so on). **?** formalized this approach into an algorithm—search for a random sample that leads to a fit on

which many of the data points agree. The algorithm is usually called *RANSAC*, for RANdom SAmple Consensus. and is displayed in Procedure **??**.

Procedure: 12.4 *Fitting a Line Using RANSAC*

This procedure takes a set of N points $(x_{i,1}, x_{i,2})$ putatively lying on a line and obtains an estimate of that line.
Start by choosing: the number of iterations required, k; the threshold used to identify a point that fits well, t; the number of nearby points required to assert a model fits well, d. Set up a collection of good fits, currently empty.
Iterate until k iterations have occurred:

> Draw a sample of two distinct points from the data uniformly and at random, and determine the line through those two points.
>
> For each data point outside the sample, if the distance from the point to the structure is less than t, the point is close
>
> If there are d or more points close to the line then there is a good fit. Refit the line using all these points and a robust loss. Add the result to a collection of good fits.

Use the best fit from the collection of good fits, using the fitting error as a criterion.

The Number of Samples Required

For lines, samples consist of pairs of distinct points drawn uniformly and at random from the dataset. The algorithm applies to much more general fitting, where sample contains the minimum number of points required to fit whatever you want to fit. For example, to fit planes, draw triples of points; to fit circles, draw triples of points, and so on. Assume you need to draw n data points, and that w is the fraction of these points that are good. You need only a reasonable estimate of this number. Now the expected value of the number of draws k required to get one good sample is given by

$$\begin{aligned} \mathrm{E}[k] &= 1P(\text{one good sample in one draw}) + \\ &\quad 2P(\text{one good sample in two draws}) + \ldots \\ &= w^n + 2(1-w^n)w^n + 3(1-w^n)^2 w^n + \ldots \\ &= w^{-n} \end{aligned}$$

(where the last step takes a little manipulation of algebraic series). To be fairly confident that of seeing a good sample, you need to draw rather more than w^{-n} samples; a natural thing to do is to add a few – say f – standard deviations to this number. The standard deviation of k is

$$SD(k) = \frac{\sqrt{1-w^n}}{w^n}.$$

so you could draw

$$w^{-n}(1 + f\sqrt{1 - w^n})$$

samples. An alternative approach to this problem is to look at a number of samples that guarantees a low probability z of seeing only bad samples. In this case

$$z = (1 - w^n)^k,$$

which means that

$$k = \frac{\log(z)}{\log(1 - w^n)}.$$

Notice that either calculation implies you need of the order of w^{-n} samples, which becomes a problem if n is big or w is small. These are regimes in which RANSAC is quite difficult to use successfully.

These calculations assume that samples drawn are independent of one another, which is plausible when the fraction of inliers is not too close to zero. If there are very few inliers (this isn't the same as a small fraction of inliers), then good samples will tend to be correlated because they are likely to share some inliers. In this case a more delicate calculation is required (**exercises**).

If w is unknown, it can be estimated from the fitting attempts. If you observe a sequence of A fitting attempts, of which g appear successful, then you can estimate $w = (g/A)^{1/n}$. You could start with a relatively low estimate of w, generate a sequence of attempted fits, and then improve the estimate of w. If you have more fitting attempts than the new estimate of w predicts, the process can stop. The problem of updating the estimate of w reduces to estimating the probability that a coin comes up heads or tails given a sequence of flips **exercises** .

Telling Whether a Point Is Close

You can tell whether a point lies close to the line fitted to a sample by testing the distance from point to line against a threshold d. In general, specifying this parameter is part of the modeling process. Obtaining a value for this parameter is relatively simple. You generally need only an order of magnitude estimate, and the same value applies to many different experiments. The parameter is often determined by trying a few values and seeing what happens; another approach is to look at a few characteristic datasets, fitting a line by eye, and estimating the average size of the deviations.

The Number of Points That Must Agree

You have fitted a line to some random sample of two data points, and need to know whether that line is good. To do so, count the number of points that lie within some distance of the line (the distance was determined in the previous section). Write y for the probability that an outlier lies in this collection of points. Now choose some number of points t such that the probability that all points near the line are outliers, y^t, is small (say, less than 0.05). While y isn't known, notice that $y \leq (1 - w)$ (because some outliers should be far from the line), so choose t such that $(1 - w)^t$ is small.

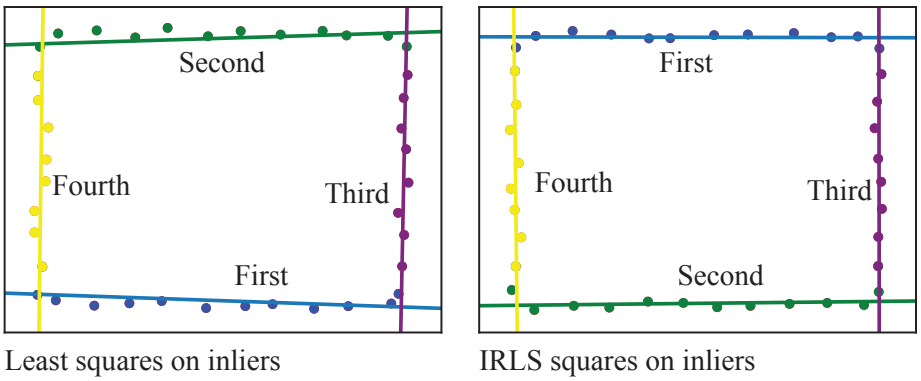

FIGURE 12.9: *Incremental RANSAC can successfully fit multiple lines to a noisy dataset. Points lie on the outline of a square, and have been perturbed by noise. I applied the strategy of Section 12.3.1 using a RANSAC line fitter. Colors show inliers and line for each round (in the order* **blue**, **green**, **purple**, **yellow***). On the* **left**, *the final fit to the inlying points is by total least squares. Because some inliers can be quite far from the line at the corners, the lines tend not to run close to the data points. On the* **right**, *the final fit uses 10 iterations of IRLS, with the Huber loss. The lines are now very close to the data points.*

12.3.1 Fitting Many Lines with RANSAC

Here is a natural strategy for fitting k lines to a set of tokens. Start with a working set of all tokens. For k rounds, find the best line fitting the working set of tokens, then keep this line and take the tokens that are close to it out of the working set. It is difficult to make this strategy work using IRLS. Because there are many lines, there must be many outliers for each line. IRLS can respond quite badly to large numbers of outliers (Figure 12.8). RANSAC is a natural choice. As Figure 12.9 shows, it is quite important to use a robust loss in the line fit step.