COM4509/6509
Machine Learning and Adaptive Intelligence
Lecture 4: Linear Regression

Mike Smith* and Matt Ellis

*m.t.smith@sheffield.ac.uk

Detour: Inner and Outer Products

Inner Product

Let $\underline{a}$ and $\underline{b}$ be column vectors of shape $N \times 1$,

$$a \cdot b = a^T b = a_1 b_1 + a_2 b_2 + \dots + a_N b_N$$

Outer Product

Let $\underline{a}$ and $\underline{b}$ be column vectors of shapes $N \times 1$ and $M \times 1$ respectively

$$a \otimes b = ab^T = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \dots & a_1 b_M \\ a_2 b_1 & & & \vdots \\ \vdots & & \ddots & \vdots \\ a_N b_1 & \dots & & a_N b_N \end{bmatrix}$$

Detour: Inner and Outer Products

Inner Product

Let $\underline{a}$ and $\underline{b}$ be column vectors of shape $N \times 1$,

$a \cdot b = a^T b = a_1 b_1 + a_2 b_2 + \dots + a_N b_N$

Outer Product

Let a and b be column vectors of shapes $N \times 1$ and $M \times 1$ respectively

$$a \otimes b = ab^T = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \dots & a_1 b_M \\ a_2 b_1 & & & \vdots \\ \vdots & & \ddots & \vdots \\ a_N b_1 & \dots & & a_N b_N \end{bmatrix}$$

If $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$ what is the inner product of a and b?

If $a = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $b = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix}$ what is the outer product of a and b?

ACTIVITY: Compute these!

Detour: Inner and Outer Products

Inner Product

Let $\underline{a}$ and $\underline{b}$ be column vectors of shape $N \times 1$,

$a \cdot b = a^T b = a_1 b_1 + a_2 b_2 + \dots + a_N b_N$

Outer Product

Let $\underline{a}$ and $\underline{b}$ be column vectors of shapes $N \times 1$ and $M \times 1$ respectively

$$a \otimes b = ab^T = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \dots & a_1 b_M \\ a_2 b_1 & & & \vdots \\ \vdots & & \ddots & \\ a_N b_1 & \dots & & a_N b_N \end{bmatrix}$$

If $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$ what is the inner product of a and b?

$$a^T b = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = 1 + 2 - 3 = 0$$

If $a = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $b = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix}$ what is the outer product of a and b?

Detour: Inner and Outer Products

Inner Product

Let $\underline{a}$ and $\underline{b}$ be column vectors of shape $N \times 1$,

$a \cdot b = a^T b = a_1 b_1 + a_2 b_2 + \dots + a_N b_N$

Outer Product

Let $\underline{a}$ and $\underline{b}$ be column vectors of shapes $N \times 1$ and $M \times 1$ respectively

$$a \otimes b = ab^T = \begin{bmatrix} a_1 b_1 & a_1 b_2 & \dots & a_1 b_M \\ a_2 b_1 & & & \vdots \\ \vdots & & \ddots & \vdots \\ a_N b_1 & \dots & & a_N b_N \end{bmatrix}$$

If $a = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$, $b = \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix}$ what is the inner product of a and b?

$$a^T b = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ -1 \end{bmatrix} = 1 + 2 - 3 = 0$$

If $a = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$, $b = \begin{bmatrix} -2 \\ -1 \\ 0 \\ 1 \end{bmatrix}$ what is the outer product of a and b?

$$\begin{bmatrix} 1 \\ 2 \end{bmatrix} \begin{bmatrix} -2 & -1 & 0 & 1 \end{bmatrix} = \begin{bmatrix} -2 & -1 & 0 & 1 \\ -4 & -2 & 0 & 2 \end{bmatrix}$$

Linear Regression

We have some training data:

$x_1 = 3 \quad y_1 = 2$

$x_2 = 5 \quad y_2 = 3$

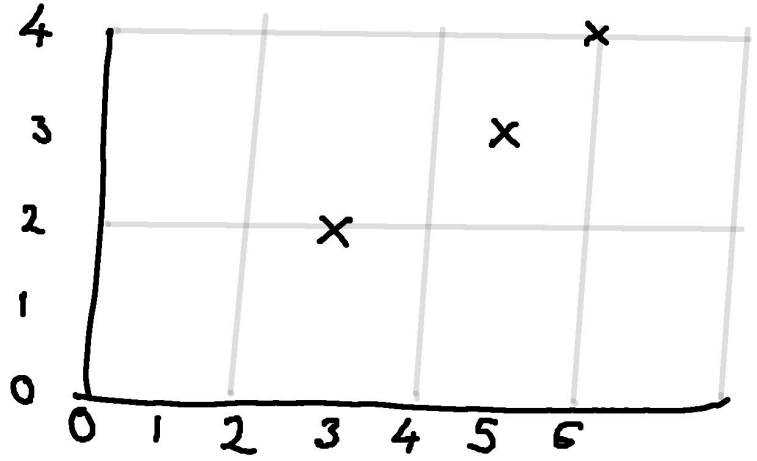
$x_3 = 6 \quad y_4 = 4$

We have some training data:

$x_1 = 3 \quad y_1 = 2$

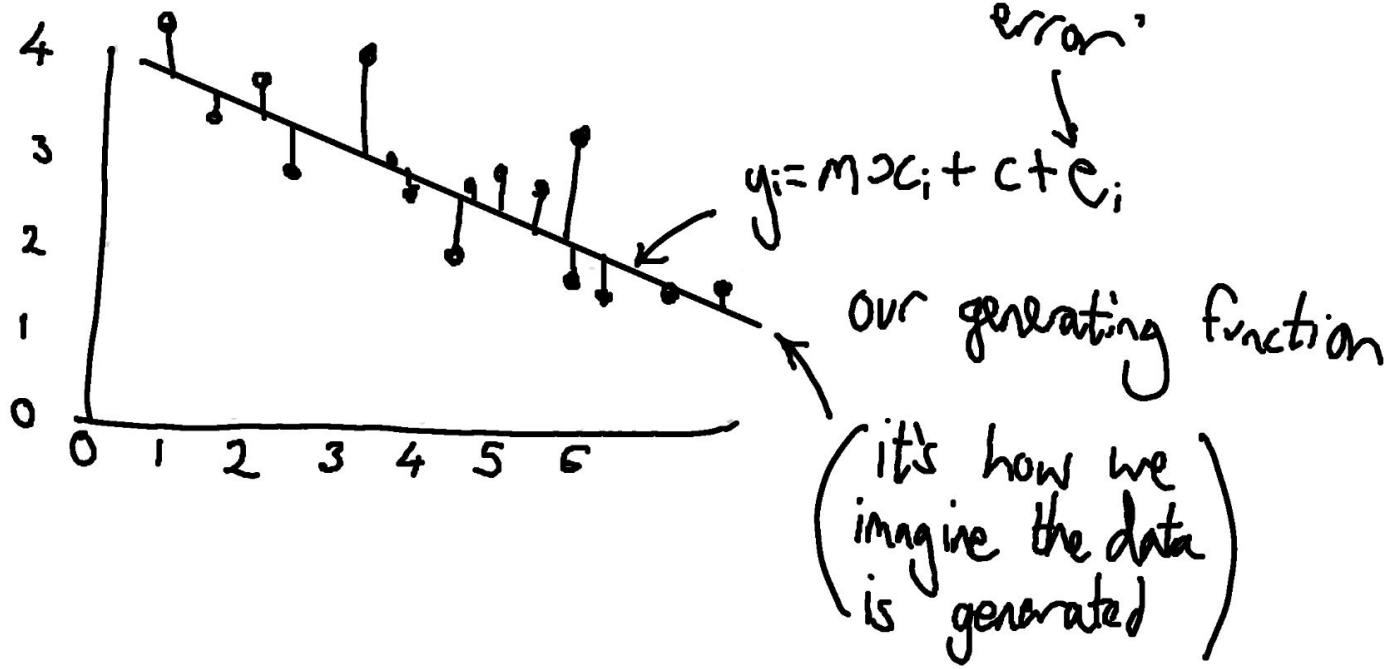
$x_2 = 5 \quad y_2 = 3$

$x_3 = 6 \quad y_4 = 4$

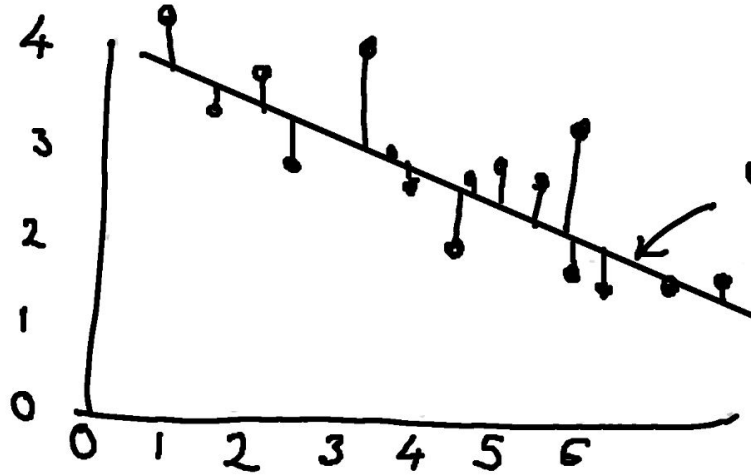


We have a model: Our data is from a linear, 1st-order polynomial with (Gaussian) noise added:

We have a model: Our data is from a linear, 1st-order polynomial with (Gaussian) noise added:



We have a model: Our data is from a linear, 1st-order polynomial with (Gaussian) noise added:



'error'

$y_i = mx_i + c + e_i$

our generating function

(it's how we imagine the data is generated)

We need to find m and c...

We need a way to say how good (or bad) a choice of m and c is.

We need a way to say how good (or bad) a choice of m and c is.

We will use the sum squared error.

We need a way to say how good (or bad) a choice of m and c is.

We will use the sum squared error. Later we will see why this is a good choice.

We need a way to say how good (or bad) a choice of m and c is.

We will use the sum squared error. Later we will see why this is a good choice.

$$E = \sum_{i=1}^{N} \left[(mx_i + c) - y_i \right]^2$$

We need a way to say how good (or bad) a choice of m and c is.

We will use the sum squared error. Later we will see why this is a good choice.

$$E = \sum_{i=1}^{N} \left[\underbrace{(mx_i + c)}_{\text{our prediction}} - \underbrace{y_i}_{\text{true label}} \right]^2$$

So this is:

$$E = \left[(mx_1 + c) - y_1\right]^2 + \left[(mx_2 + c) - y_2\right]^2 + \ldots$$

$$E = \sum_{i=1}^{N} \left[(mx_i + c) - y_i\right]^2$$

our prediction — true label

$$E = \sum_{i=1}^{N} \left[\underbrace{(mx_i + c)}_{\text{our prediction}} - \underbrace{y_i}_{\text{true label}} \right]^2$$

So this is:

$$E = \left[(mx_1 + c) - y_1\right]^2 + \left[(mx_2 + c) - y_2\right]^2 + \ldots$$

This is all quite awkward so it's better to write with vector notation,

$$E = \sum_{i=1}^{N} \left[\underbrace{(mx_i + c)}_{\text{our prediction}} - \underbrace{y_i}_{\text{true label}} \right]^2$$

So this is:

$$E = \left[(mx_1 + c) - y_1\right]^2 + \left[(mx_2 + c) - y_2\right]^2 + \ldots$$

This is all quite awkward so it's better to write with vector notation,

First, we look at how to write a vector of:

$$\begin{pmatrix} mx_1 + c \\ mx_2 + c \\ mx_3 + c \\ \vdots \end{pmatrix}$$

If we first make a matrix that looks like this $\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \end{bmatrix}$

If we first make a matrix that looks like this $\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \end{bmatrix}$

then multiply it by a vector $\begin{pmatrix} m \\ c \end{pmatrix}$

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} =$$

ACTIVITY: What does this equal?

If we first make a matrix that looks like this $\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \end{bmatrix}$

then multiply it by a vector $\begin{pmatrix} m \\ c \end{pmatrix}$

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} mx_1 + c \\ mx_2 + c \\ mx_3 + c \end{bmatrix}$$

If we first make a matrix that looks like this $\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \end{bmatrix}$

then multiply it by a vector $\begin{pmatrix} m \\ c \end{pmatrix}$

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} mx_1 + c \\ mx_2 + c \\ mx_3 + c \end{bmatrix}$$

parameter vector $\underline{w}$

If we first make a matrix that looks like this $\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \\ \vdots & \vdots \end{bmatrix}$

then multiply it by a vector $\begin{pmatrix} m \\ c \end{pmatrix}$

We call this X,

The columns are 'regressors' or 'explanatory variables' often called the design matrix

$$\begin{bmatrix} x_1 & 1 \\ x_2 & 1 \\ x_3 & 1 \end{bmatrix} \begin{bmatrix} m \\ c \end{bmatrix} = \begin{bmatrix} mx_1 + c \\ mx_2 + c \\ mx_3 + c \end{bmatrix}$$

parameter vector w

We write our labels as another vector:

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

We write our labels as another vector:

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

We want to square each row of

$$\underline{y} - X\underline{w}$$

We write our labels as another vector:

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

We want to square each row of

$$\underline{y} - X\underline{w}$$

We can do this with an inner product:

$$(\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w})$$

We write our labels as another vector:

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

We want to square each row of

$$\underline{y} - X\underline{w}$$

We can do this with an inner product:

$$(\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w})$$

ACTIVITY: Check this is equivalent to the sum of squares error.

$$E = \sum_{i=1}^{N} \left[\underbrace{(mx_i + c)}_{\text{our prediction}} - \underbrace{y_i}_{\text{true label}} \right]^2$$

We write our labels as another vector:

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

We want to square each row of

$$\underline{y} - X\underline{w}$$

We can do this with an inner product:

$$(\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w})$$

$$\begin{bmatrix} y_1 - (mx_1 + c) & y_2 - mx_2 + c & \dots \end{bmatrix} \begin{pmatrix} y_1 - (mx_1 + c) \\ y_2 - (mx_2 + c) \\ \vdots \end{pmatrix}$$

We write our labels as another vector:

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

We want to square each row of

$$\underline{y} - X\underline{w}$$

We can do this with an inner product:

$$(\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w})$$

$$\begin{bmatrix} y_1 - (mx_1 + c) & y_2 - mx_2 + c & \dots \end{bmatrix} \begin{pmatrix} y_1 - (mx_1 + c) \\ y_2 - (mx_2 + c) \\ \vdots \end{pmatrix}$$

You can see this leads to the sum of square error we had at the start.

We write our labels as another vector:

$$\underline{y} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

We want to square each row of

$$\underline{y} - X\underline{w}$$

We can do this with an inner product:

$$(\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w})$$

$$\begin{bmatrix} y_1 - (mx_1 + c) & y_2 - mx_2 + c & \dots \end{bmatrix} \begin{pmatrix} y_1 - (mx_1 + c) \\ y_2 - (mx_2 + c) \\ \vdots \end{pmatrix}$$

You can see this leads to the sum of square error we had at the start.

$$E = \sum_{i=1}^{N} \left[\underbrace{(mx_i + c)}_{\text{our prediction}} - \underbrace{y_i}_{\text{true label}} \right]^2$$

We want to minimise our error function,

$$E = (y - X\underline{w})^T (y - X\underline{w})$$

We want to minimise our error function,

$$E = (y - X\underline{w})^T (y - X\underline{w})$$

with respect to $\underline{w}$ (our m & c).

We want to minimise our error function,

$$E = (y - X\underline{w})^T (y - X\underline{w})$$

with respect to $\underline{w}$ (our m & c).

$$\frac{\partial E}{\partial \underline{w}} = ? = 0$$

We want to minimise our error function,

$$E = (y - X\underline{w})^T (y - X\underline{w})$$

with respect to $\underline{w}$ (our m & c).

$$\frac{\partial E}{\partial \underline{w}} = ? = 0$$

Check the matrix cookbook...

We want to minimise our error function,

$$E = (\underline{y} - X\underline{\omega})^T (\underline{y} - X\underline{\omega})$$

with respect to $\underline{\omega}$ (our m & c).

$$\frac{\partial E}{\partial \underline{\omega}} = ? = 0$$

Check the matrix cookbook...

Assume $\mathbf{W}$ is symmetric, then

This identity looks good →

$$\frac{\partial}{\partial \mathbf{s}}(\mathbf{x} - \mathbf{A}\mathbf{s})^T \mathbf{W}(\mathbf{x} - \mathbf{A}\mathbf{s}) = -2\mathbf{A}^T \mathbf{W}(\mathbf{x} - \mathbf{A}\mathbf{s}) \quad (84)$$

We want to minimise our error function,

$$E = (y - X\underline{\omega})^T (y - X\underline{\omega})$$

with respect to $\underline{\omega}$ (our m & c).

$$\frac{\partial E}{\partial \underline{\omega}} = ? = 0$$

Check the matrix cookbook...

Assume $\mathbf{W}$ is symmetric, then

This identity looks good →

$$\frac{\partial}{\partial \mathbf{s}}(\mathbf{x} - \mathbf{A}\mathbf{s})^T \mathbf{W}(\mathbf{x} - \mathbf{A}\mathbf{s}) = -2\mathbf{A}^T \mathbf{W}(\mathbf{x} - \mathbf{A}\mathbf{s}) \quad (84)$$

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top \mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top \mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top \mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top\mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top\mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top\mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top\mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

(and other rules we have learnt about transposes etc)

$$\frac{\partial}{\partial \underline{w}} (\underline{y}-X\underline{w})^T(\underline{y}-X\underline{w})$$

ACTIVITY: Have a go at finding the value of w that minimises this error function, E! We need to differentiate E wrt w, then set to zero.

Here are some hints:

1. Move the transpose inside the bracket, e.g. by noting that $(Xw)^T=w^TX^T$.
2. Multiply out the brackets.
3. Differentiate each term wrt w.
4. Set equal to zero.
5. Multiply both sides by $(XX^T)^{-1}$ and cancel any constants on both sides etc.

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top\mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top\mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top\mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top\mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

$$\frac{\partial}{\partial \underline{w}} (\underline{y}-X\underline{w})^T(\underline{y}-X\underline{w}) = (\underline{y}^T-\underline{w}^TX^T)(\underline{y}-X\underline{w})$$

(moving the transpose inside the bracket).

Move the transpose inside the bracket, e.g. by noting that $(Xw)^T=w^TX^T$.
Multiply out the brackets.
Differentiate each term wrt w.
Set equal to zero.
Multiply both sides by $(XX^T)^{-1}$ and cancel any constants on both sides etc.

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top \mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top \mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top \mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

$$\frac{\partial}{\partial \underline{w}} (\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w}) = (\underline{y}^T - \underline{w}^T X^T)(\underline{y} - X\underline{w})$$

$$= \frac{\partial}{\partial \underline{w}} (\underline{y}^T \underline{y} - \underline{y}^T X\underline{w} - \underline{w}^T X^T \underline{y} + \underline{w}^T X^T X\underline{w})$$

(multiplying out the brackets)

Multiply out the brackets.
Differentiate each term wrt w.
Set equal to zero.
Multiply both sides by $(XX^T)^{-1}$ and cancel any constants on both sides etc.

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top \mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top \mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top \mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

$$\frac{\partial}{\partial \underline{w}} (\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w}) = (\underline{y}^T - \underline{w}^T X^T)(\underline{y} - X\underline{w})$$

$$= \frac{\partial}{\partial \underline{w}} (\underline{y}^T\underline{y} - \underline{y}^T X\underline{w} - \underline{w}^T X^T \underline{y} + \underline{w}^T X^T X\underline{w})$$

$\underline{y}^T X\underline{w}$ is a scalar, so $= (X^T\underline{y})^T\underline{w}$; $\frac{\partial}{\partial \underline{w}} = -X^T\underline{y}$

$\underline{w}^T X^T \underline{y} = -X^T\underline{y}$

$\underline{w}^T X^T X\underline{w} = 2X^T X\underline{w}$

Differentiate each term wrt w.
Set equal to zero.
Multiply both sides by $(XX^T)^{-1}$ and cancel any constants on both sides etc.

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top \mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top \mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top \mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

$$\frac{\partial}{\partial \underline{w}} (\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w}) = (\underline{y}^T - \underline{w}^T X^T)(\underline{y} - X\underline{w})$$

$$= \frac{\partial}{\partial \underline{w}} \left(\underline{y}^T\underline{y} - \underline{y}^T X\underline{w} - \underline{w}^T X^T \underline{y} + \underline{w}^T X^T X \underline{w} \right)$$

$\underline{y}^T X\underline{w}$: 0; $-(X^T\underline{y})^T\underline{w}$, $-X^T\underline{y}$; $\underline{w}^T X^T\underline{y}$: $-X^T\underline{y}$; $\underline{w}^T X^T X\underline{w}$: $2X^T X\underline{w}$

$$= -2X^T\underline{y} + 2X^T X\underline{w} = 0$$

Set equal to zero to find minimum

Set equal to zero.
Multiply both sides by $(XX^T)^{-1}$ and cancel any constants on both sides etc.

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top \mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top \mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top \mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

$$\frac{\partial}{\partial \underline{w}} (\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w}) = (\underline{y}^T - \underline{w}^T X^T)(\underline{y} - X\underline{w})$$

$$= \frac{\partial}{\partial \underline{w}} \left(\underline{y}^T\underline{y} - \underline{y}^T X\underline{w} - \underline{w}^T X^T \underline{y} + \underline{w}^T X^T X \underline{w} \right)$$

0 ; $-(X^T\underline{y})^T\underline{w}$; $-X^T\underline{y}$; $-X^T\underline{y}$; $2X^TX\underline{w}$

$$= -2X^T\underline{y} + 2X^TX\underline{w} = 0$$

$$\Rightarrow 2X^T\underline{y} = 2X^TX\underline{w}$$

Multiply both sides by $(XX^T)^{-1}$ and cancel any constants on both sides etc.

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top \mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top \mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top \mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

$$\frac{\partial}{\partial \underline{w}} (\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w}) = (\underline{y}^T - \underline{w}^T X^T)(\underline{y} - X\underline{w})$$

$$= \frac{\partial}{\partial \underline{w}} \left(\underline{y}^T\underline{y} - \underline{y}^T X\underline{w} - \underline{w}^T X^T \underline{y} + \underline{w}^T X^T X \underline{w} \right)$$

0 ; $-(X^T\underline{y})^T\underline{w}$; $-X^T\underline{y}$; $-X^T\underline{y}$; $2X^TX\underline{w}$

$$= -2X^T\underline{y} + 2X^TX\underline{w} = 0$$

$$\Rightarrow 2X^T\underline{y} = 2X^TX\underline{w}$$

$$(X^TX)^{-1}X^T\underline{y} = (X^TX)^{-1}X^TX\underline{w}$$

cancel

Multiply both sides by $(XX^T)^{-1}$ and cancel any constants on both sides etc.

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top \mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top \mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top \mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

$$\frac{\partial}{\partial \underline{w}} (\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w}) = (\underline{y}^T - \underline{w}^T X^T)(\underline{y} - X\underline{w})$$

$$= \frac{\partial}{\partial \underline{w}} \left(\underline{y}^T\underline{y} - \underline{y}^T X\underline{w} - \underline{w}^T X^T \underline{y} + \underline{w}^T X^T X \underline{w} \right)$$

$\underline{y}^T X\underline{w}$: 0, $-(X^T\underline{y})^T\underline{w}$, $-X^T\underline{y}$; $\underline{w}^T X^T\underline{y}$: $-X^T\underline{y}$; $\underline{w}^T X^T X\underline{w}$: $2X^T X\underline{w}$

$$= -2X^T\underline{y} + 2X^T X\underline{w} = 0$$

$$\Rightarrow 2X^T\underline{y} = 2X^T X\underline{w}$$

$$(X^T X)^{-1} X^T \underline{y} = (X^T X)^{-1} X^T X \underline{w}$$

(cancel)

$$(X^T X)^{-1} X^T \underline{y} = \underline{w}$$

$f(\mathbf{w})$	$\frac{df(\mathbf{w})}{d\mathbf{w}}$
$\mathbf{w}^\top \mathbf{x}$	$\mathbf{x}$
$\mathbf{x}^\top \mathbf{w}$	$\mathbf{x}$
$\mathbf{w}^\top \mathbf{w}$	$2\mathbf{w}$
$\mathbf{w}^\top \mathbf{C}\mathbf{w}$	$2\mathbf{C}\mathbf{w}$.

Could also work out from these rules

$$\frac{\partial}{\partial \underline{w}} (\underline{y} - X\underline{w})^T (\underline{y} - X\underline{w}) = (\underline{y}^T - \underline{w}^T X^T)(\underline{y} - X\underline{w})$$

$$= \frac{\partial}{\partial \underline{w}} \left(\underline{y}^T\underline{y} - \underline{y}^T X\underline{w} - \underline{w}^T X^T \underline{y} + \underline{w}^T X^T X \underline{w} \right)$$

$\underline{y}^T X\underline{w}$: 0; $(X^T\underline{y})^T\underline{w}$: $-X^T\underline{y}$; $\underline{w}^T X^T\underline{y}$: $-X^T\underline{y}$; $\underline{w}^T X^T X\underline{w}$: $2X^T X\underline{w}$

$$= -2X^T\underline{y} + 2X^T X\underline{w} = 0$$

$$\Rightarrow 2X^T\underline{y} = 2X^T X\underline{w}$$

$$(X^T X)^{-1} X^T \underline{y} = (X^T X)^{-1} X^T X \underline{w}$$

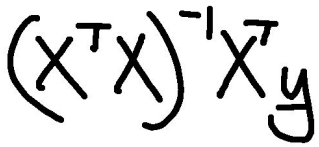
(cancel)

$$(X^T X)^{-1} X^T \underline{y} = \underline{w}$$

known as the normal equation

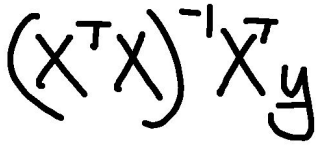
Can be solved if $X^T X$ is invertable.

Iterative Optimisation



For the case of linear regression, we've just seen it's got a **closed form solution**. I.e. we can set the gradient to zero, rearrange, and get the answer.

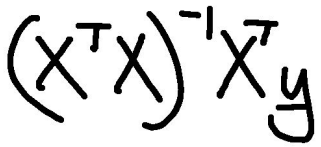
Iterative Optimisation



For the case of linear regression, we've just seen it's got a **closed form solution**. I.e. we can set the gradient to zero, rearrange, and get the answer.

For most problems we can't do this. But we might be able to still compute the **gradient** (and use a **gradient descent** algorithm)

Iterative Optimisation



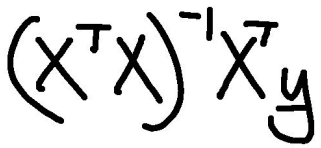
For the case of linear regression, we've just seen it's got a **closed form solution**. I.e. we can set the gradient to zero, rearrange, and get the answer.

For most problems we can't do this. But we might be able to still compute the **gradient** (and use a **gradient descent** algorithm)

Side note:

> Even though linear regression has a closed form solution, it might be too tricky to compute. ACTIVITY: What size is (X^TX)?

Iterative Optimisation



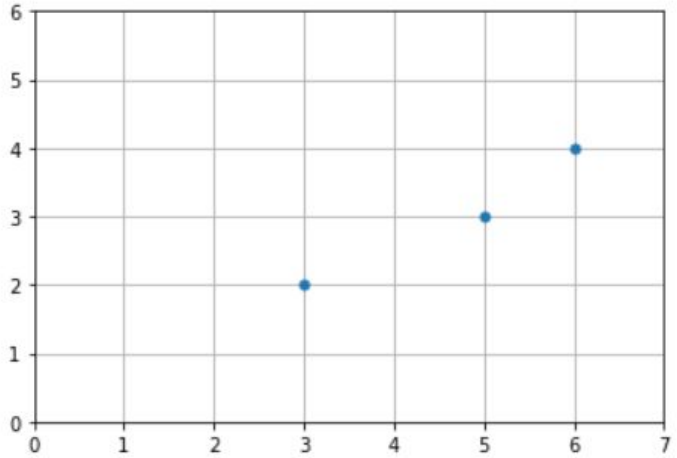
For the case of linear regression, we've just seen it's got a **closed form solution**. I.e. we can set the gradient to zero, rearrange, and get the answer.

For most problems we can't do this. But we might be able to still compute the **gradient** (and use a **gradient descent** algorithm)

Side note:

> Even though linear regression has a closed form solution, it might be too tricky to compute. ACTIVITY: What size is (X^TX)?
>
> X is (N x D), so X^TX is (D x D).
>
> It takes roughly $O(D^3)$ time to compute the inverse, and uses approximately $O(D^2)$ memory. If our data is high-dimensional (i.e. D is big) then both of these complexities can be a problem.
>
> So we might still need to use gradient descent…

```
x = np.array([3,5,6])
y = np.array([2,3,4])
plt.plot(x,y,'.',markersize=10)
plt.xlim([0,7])
plt.ylim([0,6])
plt.grid()
```



Let’s code an example!

```python
def getprediction(w,x):
    X = np.c_[x,np.ones_like(x)]
    return X@w
```

This is the method that computes a **prediction** for a given value of **w** and x.

```python
def getprediction(w,x):
    X = np.c_[x,np.ones_like(x)]
    return X@w
```

Build the design matrix X from our vector **x**. This has two columns (two regressors). The first is full of xs, the second full of 1s.

This is the method that computes a **prediction** for a given value of **w** and x.

```
1 def getprediction(w,x):
2     X = np.c_[x,np.ones_like(x)]
3     return X@w
```

Our prediction is X**w**. In numpy matrix multiplication is with the @ symbol.

Build the design matrix X from our vector **x**. This has two columns (two regressors). The first is full of xs, the second full of 1s.

This is the method that computes a **prediction** for a given value of **w** and x.

```
def getprediction(w,x):
    X = np.c_[x,np.ones_like(x)]
    return X@w
```

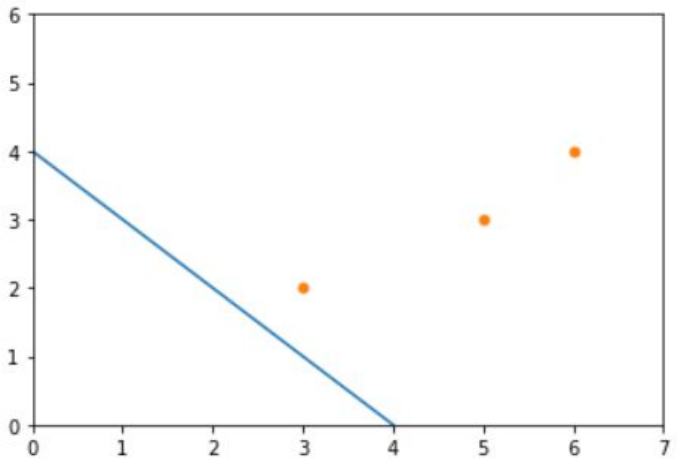
```
w = [-1,4]
xtest = np.linspace(0,7,100)
ytest = getprediction(w,xtest)
plt.plot(xtest,ytest,'-')
plt.plot(x,y,'.',markersize=10)
plt.xlim([0,7])
plt.ylim([0,6])
```

Let's plot the predictions for different values of x, with **w** equal to $[-1,4]^T$.

```python
def getprediction(w,x):
    X = np.c_[x,np.ones_like(x)]
    return X@w
```

```python
w = [-1,4]
xtest = np.linspace(0,7,100)
ytest = getprediction(w,xtest)
plt.plot(xtest,ytest,'-')
plt.plot(x,y,'.',markersize=10)
plt.xlim([0,7])
plt.ylim([0,6])
```

```
(0.0, 6.0)
```



Let's plot the predictions for different values of x, with **w** equal to $[-1,4]^T$.

Not a very good fit!

```
1 def costfn(w,xtest,ytest):
2     return np.sum((getprediction(w,xtest)-ytest)**2)
```

Let’s write the cost function. This returns the sum squared error.

```
1 def costfn(w,xtest,ytest):
2     return np.sum((getprediction(w,xtest)-ytest)**2)
```

```
1 costfn([-1,4],x,y)
```

```
53
```

For $\mathbf{w} = [-1,4]^T$, E = 53.

```
1 def costfn(w,xtest,ytest):
2     return np.sum((getprediction(w,xtest)-ytest)**2)
```

```
1 costfn([-1,4],x,y)
```

```
53
```

```
1 costfn([0.5,4],x,y)
```

```
33.5
```

For $\mathbf{w} = [0.5,4]^T$, E = 33.5.

```
1 def costfn(w,xtest,ytest):
2     return np.sum((getprediction(w,xtest)-ytest)**2)
```

```
1 costfn([-1,4],x,y)
```

```
53
```

```
1 costfn([0.5,4],x,y)
```

```
33.5
```

For $\mathbf{w} = [0.5,4]^T$, E = 33.5.

Ooh, the error is less, it seemed like increasing the first term (the gradient) from -1 to +0.5 helped improved our predictions.

If only there was a way of finding out how much E changes depending on **w**…

```
def costfn(w,xtest,ytest):
    return np.sum((getprediction(w,xtest)-ytest)**2)
```

```
costfn([-1,4],x,y)
```

```
53
```

```
costfn([0.5,4],x,y)
```

```
33.5
```

```
def costfn_grad(w,x,y):
    X = np.c_[x,np.ones_like(x)]
    return -2*X.T@y + 2*X.T@X@w
```

Ah! Here's the gradient of E wrt **w**! We worked this out in the last section!

This says $-2X^T\mathbf{y} + 2X^TX\mathbf{w}$
Which is what we found the gradient (dE/dw) to be:

$$= -2X^T y + 2X^T X w$$

```python
def costfn(w,xtest,ytest):
    return np.sum((getprediction(w,xtest)-ytest)**2)
```

```python
costfn([-1,4],x,y)
```

```
53
```

```python
costfn([0.5,4],x,y)
```

```
33.5
```

```python
def costfn_grad(w,x,y):
    X = np.c_[x,np.ones_like(x)]
    return -2*X.T@y + 2*X.T@X@w
```

This says $-2X^T\mathbf{y} + 2X^TX\mathbf{w}$
Which is what we found the gradient (dE/dw) to be:

$$= -2X^T y + 2X^T X w$$

Ah! Here's the gradient of E wrt **w**! We worked this out in the last section!

For complicated functions we use **autodiff** frameworks (TensorFlow, PyTorch, Keras, Theano, JAX, etc). E.g. if the getprediction implemented a deep neural network.

```
def costfn(w,xtest,ytest):
    return np.sum((getprediction(w,xtest)-ytest)**2)
```

```
costfn([-1,4],x,y)
```

```
53
```

```
costfn([0.5,4],x,y)
```

```
33.5
```

Off topic: To check I've coded my gradient correctly, I'm computing it numerically too…

```
def costfn_grad(w,x,y):
    X = np.c_[x,np.ones_like(x)]
    return -2*X.T@y + 2*X.T@X@w
```

```
def costfn_numerical_grad(w,xtest,ytest):
    grad = np.array([0,0])
    grad[0] = (costfn(w+np.array([0.001,0]),xtest,ytest)-costfn(w-np.array([0.001,0]),xtest,ytest))/0.002
    grad[1] = (costfn(w+np.array([0,0.001]),xtest,ytest)-costfn(w-np.array([0,0.001]),xtest,ytest))/0.002
    return grad
```

```
def costfn(w,xtest,ytest):
    return np.sum((getprediction(w,xtest)-ytest)**2)
```

```
costfn([-1,4],x,y)
```

```
53
```

```
costfn([0.5,4],x,y)
```

```
33.5
```

```
def costfn_grad(w,x,y):
    X = np.c_[x,np.ones_like(x)]
    return -2*X.T@y + 2*X.T@X@w
```

```
def costfn_numerical_grad(w,xtest,ytest):
    grad = np.array([0,0])
    grad[0] = (costfn(w+np.array([0.001,0]),xtest,ytest)-costfn(w-np.array([0.001,0]),xtest,ytest))/0.002
    grad[1] = (costfn(w+np.array([0,0.001]),xtest,ytest)-costfn(w-np.array([0,0.001]),xtest,ytest))/0.002
    return grad
```

```
costfn_grad([0,4],x,y)
```

```
array([22,  6])
```

```
def costfn(w,xtest,ytest):
    return np.sum((getprediction(w,xtest)-ytest)**2)
```

```
costfn([-1,4],x,y)
```

```
53
```

```
costfn([0.5,4],x,y)
```

```
33.5
```

```
def costfn_grad(w,x,y):
    X = np.c_[x,np.ones_like(x)]
    return -2*X.T@y + 2*X.T@X@w
```

```
def costfn_numerical_grad(w,xtest,ytest):
    grad = np.array([0,0])
    grad[0] = (costfn(w+np.array([0.001,0]),xtest,ytest)-costfn(w-np.array([0.001,0]),xtest,ytest))/0.002
    grad[1] = (costfn(w+np.array([0,0.001]),xtest,ytest)-costfn(w-np.array([0,0.001]),xtest,ytest))/0.002
    return grad
```

```
costfn_grad([0,4],x,y)
```

```
array([22,  6])
```

Ok, so we've got our gradient, how do we use it to optimise **w**?

```
costfn_numerical_grad([0,4],x,y)
```

```
array([22,  6])
```

Gradient Descent

The simplest/most obvious approach is **gradient (steepest) descent**.

```
w = [-1,4] #initial value
for it in range(10000):
    grad = costfn_grad(w,x,y)
    w = w - grad*0.01
```

Gradient Descent

The simplest/most obvious approach is **gradient (steepest) descent**.

```
w = [-1,4] #initial value
for it in range(10000):
    grad = costfn_grad(w,x,y)
    w = w - grad*0.01
```

Here we simply subtract (a proportion of) the gradient from the current value of **w**. We need to do this in small increments defined by the **learning rate** (aka step size), η [eta]. In this case I've chosen η=0.01.

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \frac{d}{d\mathbf{w}} E(\mathbf{w}) \Big|_{\mathbf{w}=\mathbf{w}_k}$$

"The new value of **w** (at iteration k+1) equals the old value of **w** (at iteration k) minus eta (the learning rate) times the gradient of the error E wrt **w**, at the old value of **w**."

Gradient Descent

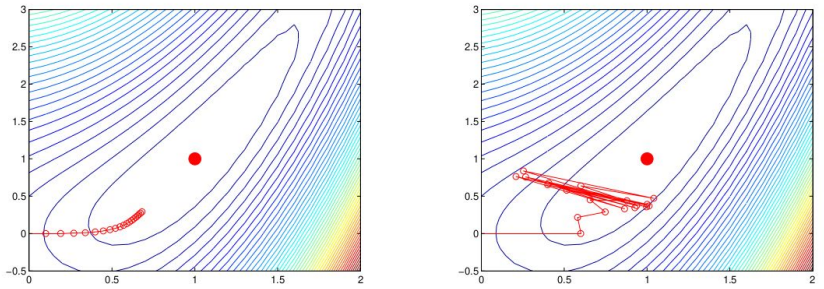
```
w = [-1,4] #initial value
for it in range(10000):
    grad = costfn_grad(w,x,y)
    w = w - grad*0.01
```

Here we simply subtract (a proportion of) the gradient from the current value of **w**. We need to do this in small increments defined by the **learning rate** (aka step size), η [eta]. In this case I've chosen η=0.01.

From Mauricio's slide

Step size

- The main issue in gradient descent is how to set the step size.
- If it is too small, convergence will be very slow. If it is too large, the method can fail to converge at all.



(a) (b)

Figure: The function to optimise is $h(w_1, w_2) = 0.5(w_1^2 - w_2)^2 + 0.5(w_1 - 1)^2$. The minimum is at $(1, 1)$. In (a) $\eta = 0.1$. In (b) $\eta = 0.6$.

Gradient Descent

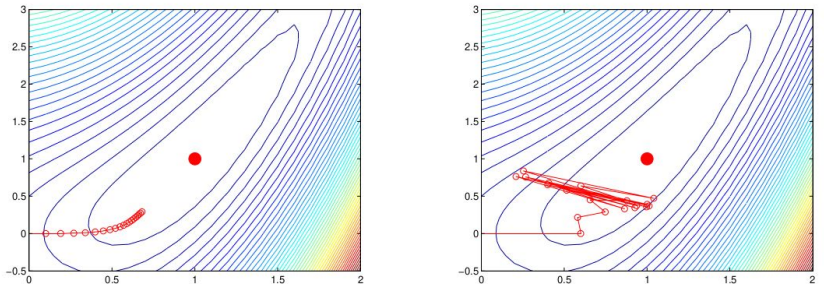
Line Search:

One solution is to pick a direction, and find a point along it where we've reduced the function the most…

From Mauricio's slide

Step size

- The main issue in gradient descent is how to set the step size.
- If it is too small, convergence will be very slow. If it is too large, the method can fail to converge at all.



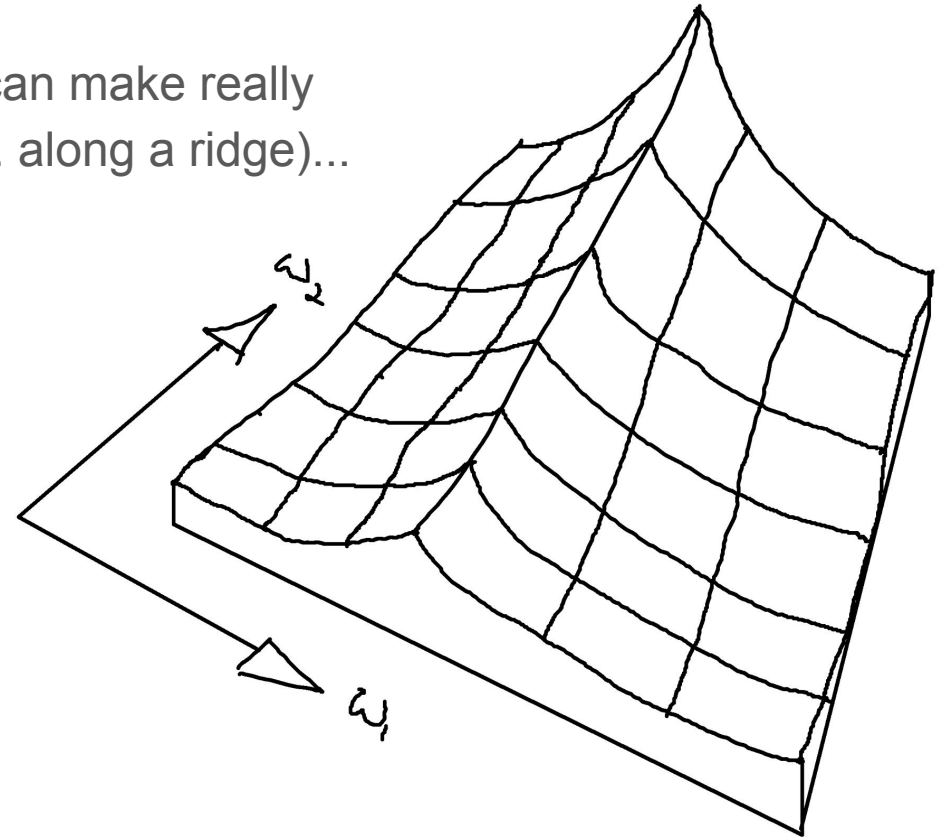
(a) (b)

Figure: The function to optimise is $h(w_1, w_2) = 0.5(w_1^2 - w_2)^2 + 0.5(w_1 - 1)^2$. The minimum is at $(1, 1)$. In (a) $\eta = 0.1$. In (b) $\eta = 0.6$.

Alternative approaches to Gradient Descent

Gradient descent can make really slow progress (e.g. along a ridge)...

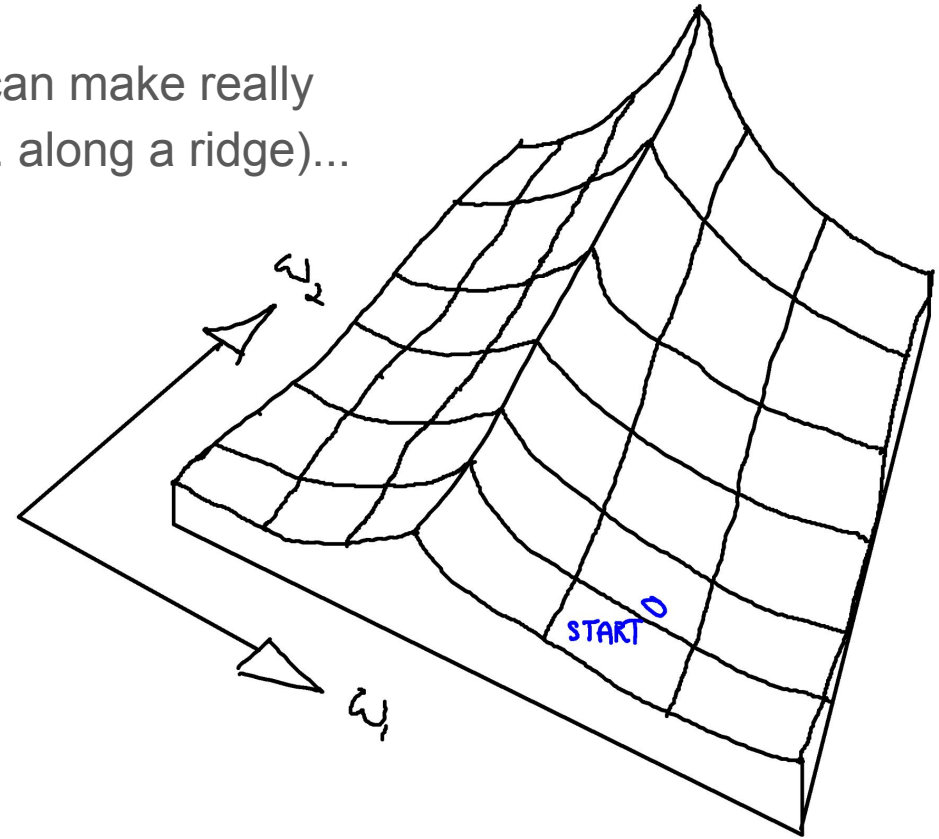
(gradient ascent in this example).



Alternative approaches to Gradient Descent

Gradient descent can make really slow progress (e.g. along a ridge)...

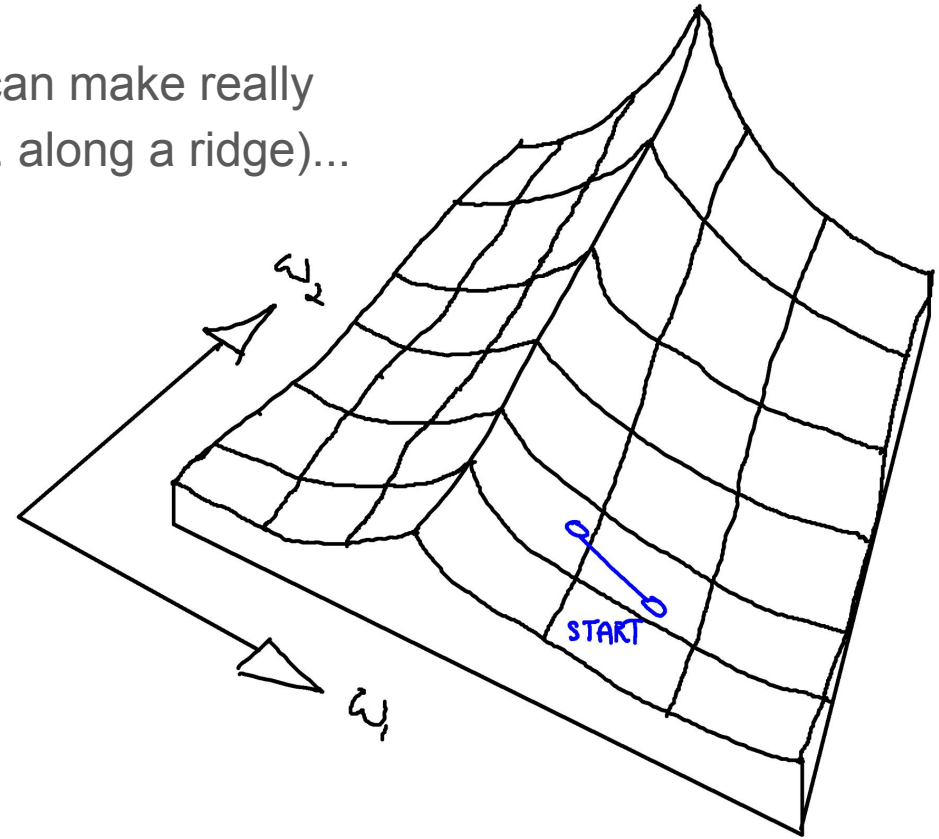
(gradient ascent in this example).



Alternative approaches to Gradient Descent

Gradient descent can make really slow progress (e.g. along a ridge)...

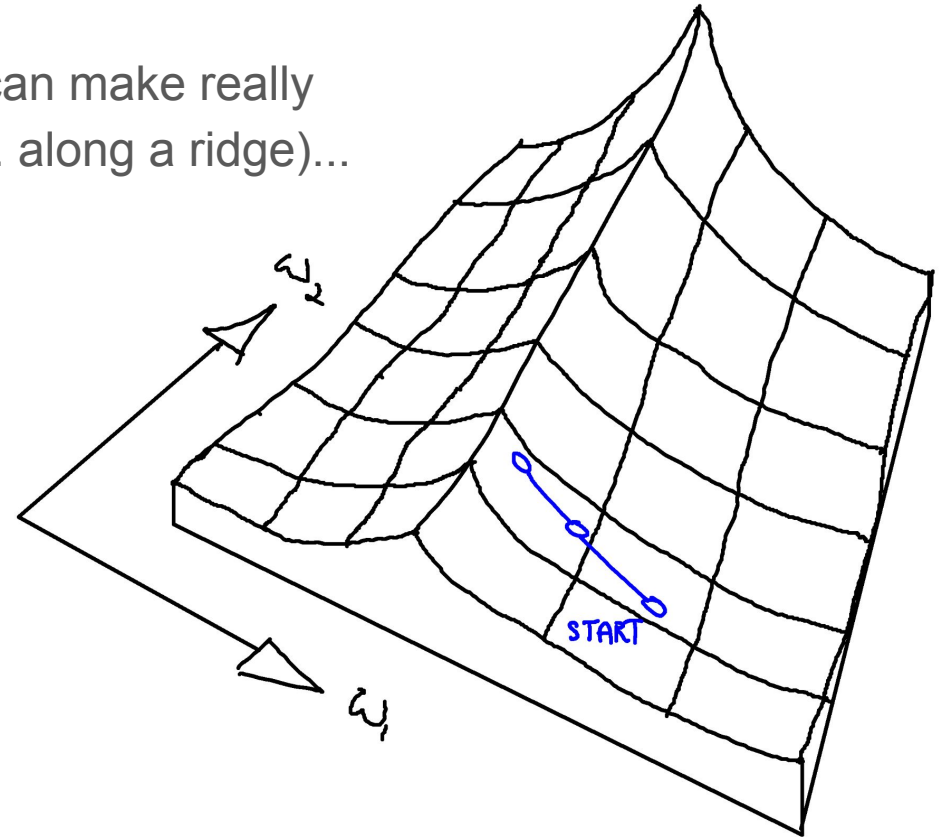
(gradient ascent in this example).



Alternative approaches to Gradient Descent

Gradient descent can make really slow progress (e.g. along a ridge)...

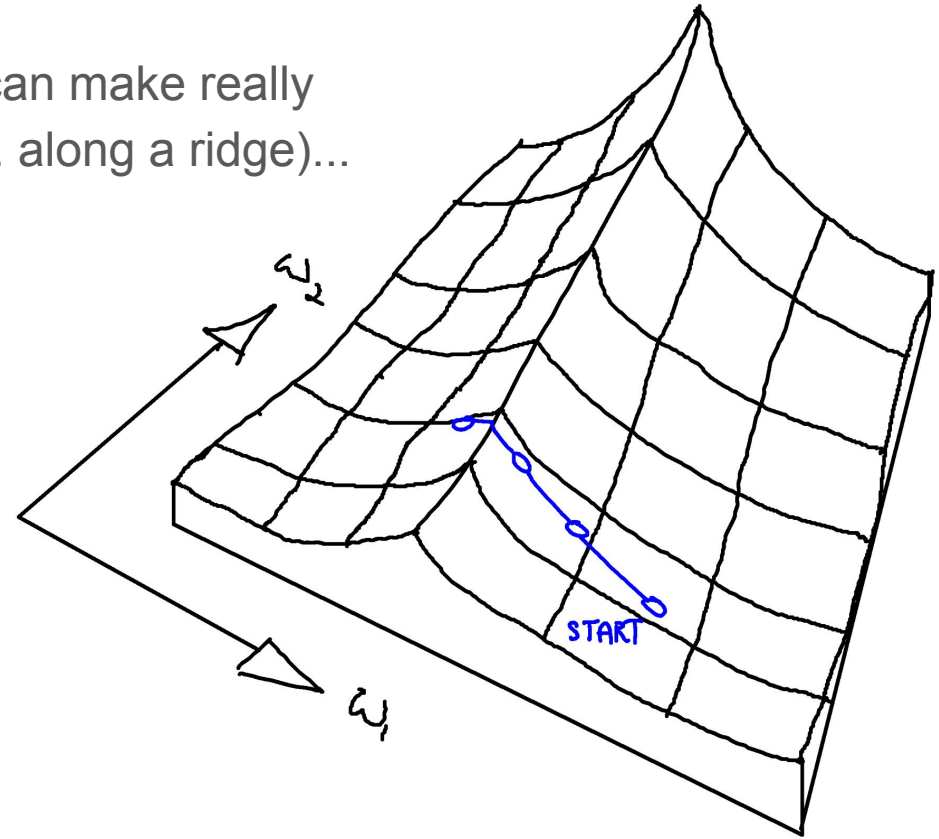
(gradient ascent in this example).



Alternative approaches to Gradient Descent

Gradient descent can make really slow progress (e.g. along a ridge)...

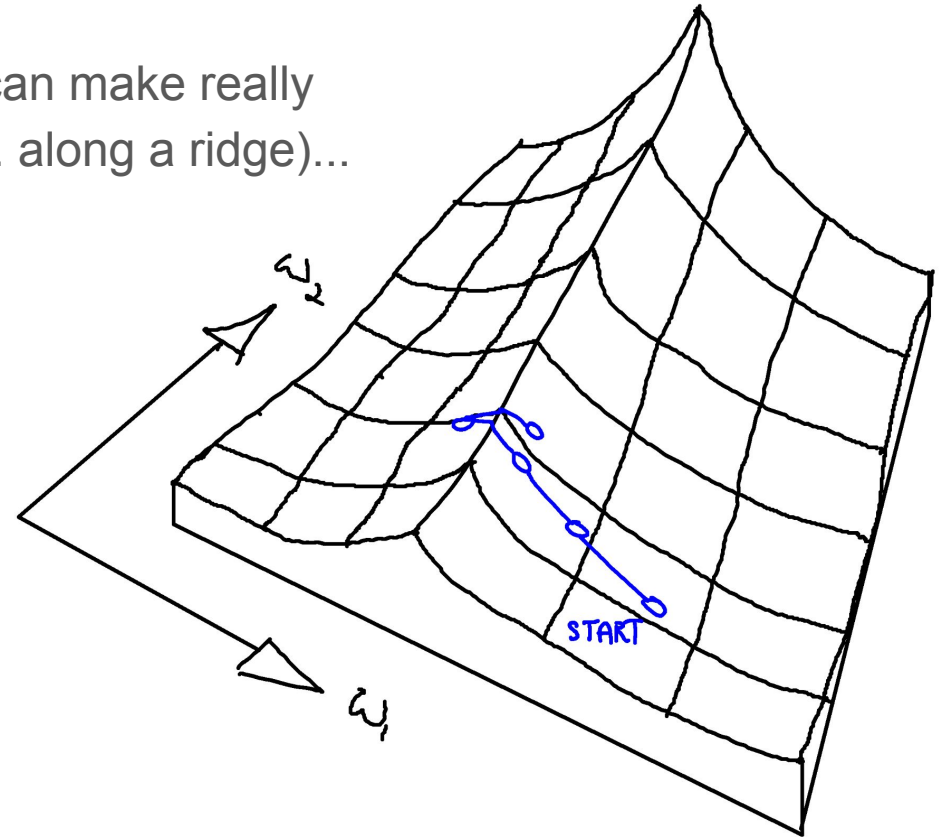
(gradient ascent in this example).



Alternative approaches to Gradient Descent

Gradient descent can make really slow progress (e.g. along a ridge)...

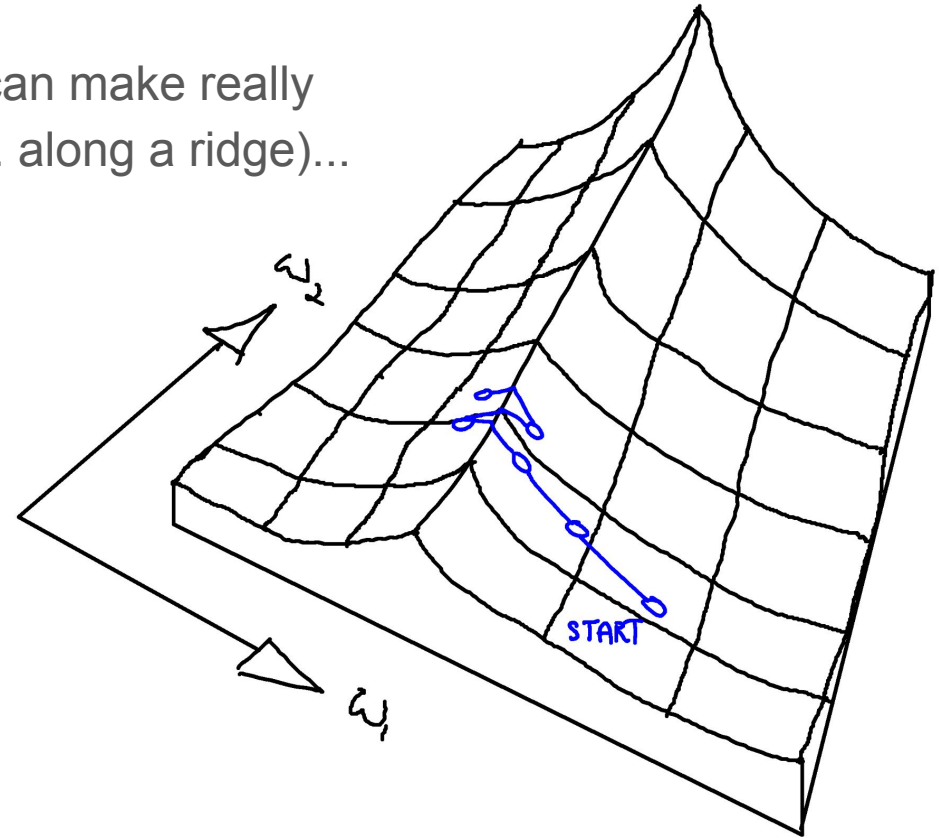
(gradient ascent in this example).



Alternative approaches to Gradient Descent

Gradient descent can make really slow progress (e.g. along a ridge)...

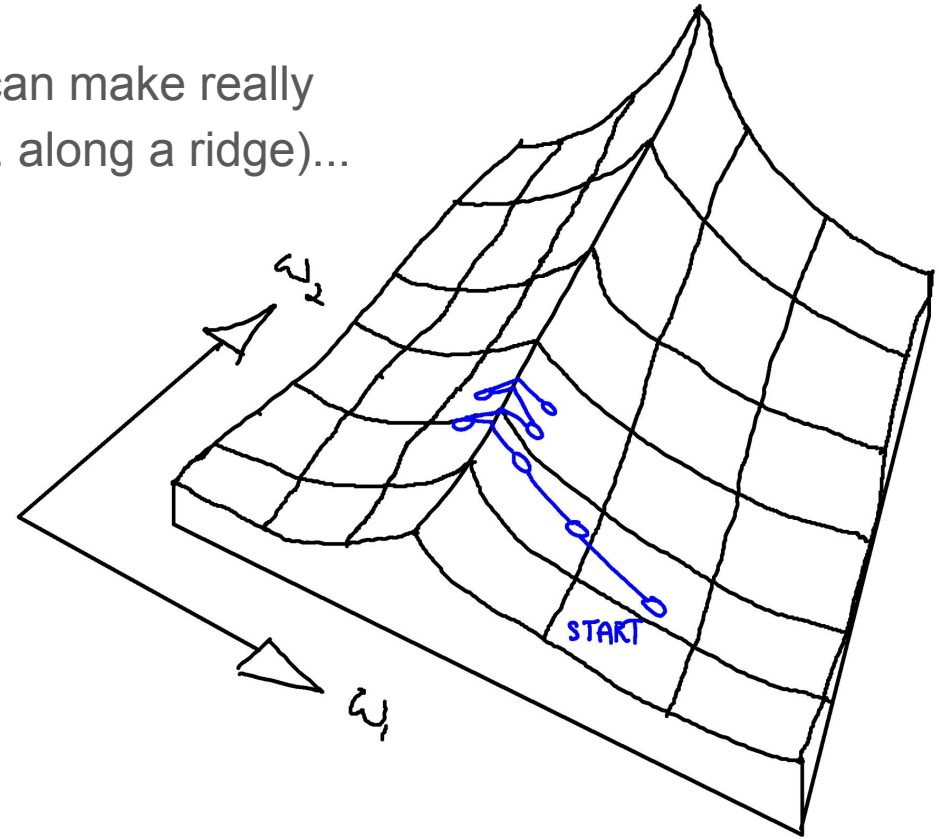
(gradient ascent in this example).



Alternative approaches to Gradient Descent

Gradient descent can make really slow progress (e.g. along a ridge)...

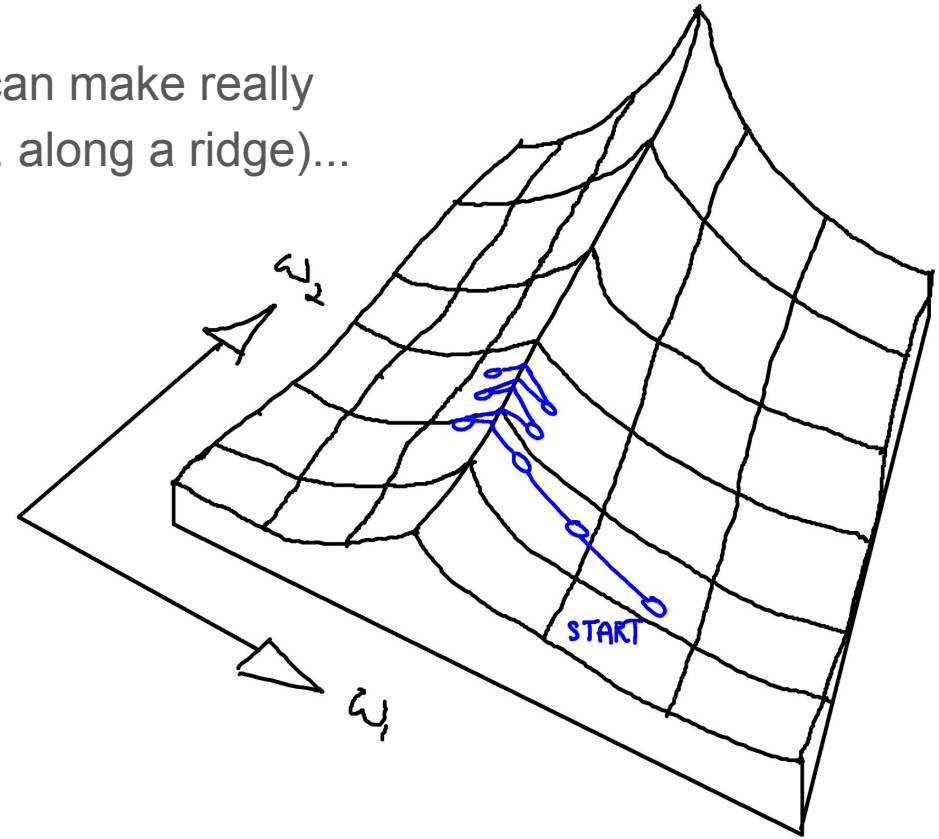
(gradient ascent in this example).



Alternative approaches to Gradient Descent

Gradient descent can make really slow progress (e.g. along a ridge)...

(gradient ascent in this example).



Alternative approaches to Gradient Descent

$$x_{k+1} = x_k + t = x_k - \frac{f'(x_k)}{f''(x_k)}$$

Newton's method [**not examinable!**] will get to a minima quite quickly as it takes into account the 2nd order derivatives:

> At each iteration, it amounts to the fitting of a parabola to the graph at the trial value, and then proceeding to the maximum or minimum of that parabola (in higher dimensions, this may also be a saddle point).
>
> – wikipedia

The problems is that this requires finding the Hessian (matrix of 2nd order partial derivatives), which is really expensive.

But approximations to the true Hessian exist. For example the BFGS algorithm.

Alternative approaches to Gradient Descent

- Newton’s approach takes too much computation per step
- Steepest descent ends up with many many small steps

An intermediate approach is **conjugate gradient descent**.

Other tricks

- The example gradient descent we computed found the gradient using **all** the data.
  - This is called batch gradient descent.

- If you have a large dataset, or the function you are trying to minimise maybe has a high time/space complexity: consider **mini-batch gradient descent.**
  - This leads to **stochastic gradient descent** (SGD) - as the gradient will depend on the sample chosen at each iteration.

Other tricks

- Choosing the learning rate (in SGD) is now more difficult. Typically some decaying sequence is chosen. If it fulfills the **Robbins-Monro conditions** (and the function we're optimising is differentiable and convex) then, in the limit, it is guaranteed to converge.

$$\eta_k = \frac{1}{k}, \quad \eta_k = \frac{1}{(\tau_0 + k)^\kappa}$$

Example sequences that fulfill the conditions.

$$\sum_{k=1}^{\infty} \eta_k = \infty, \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty.$$

Two key conditions of Robbins-Monro.

Typically though, we use a library optimisation algorithm, such as ADAM, which will adjust the learning rate to help improve convergence.

```
        Ztemp = np.c_[np.tile(Z,[S,1]),np.repeat(np.arange(S),len(Z))]
    if Z.shape[1]==2:
        Ztemp = Z
    self.Z = np.c_[np.tile(Ztemp,[C,1]),np.repeat(np.arange(self.C),len(Ztemp))]

    if likemodel=='distribution' or likemodel=='process':
        assert gpflowkernellike is not None, "You need to specify the kernel to use a distribution or process"
        self.klike = Kernel(gpflowkernellike)

        self.Zlike = np.c_[Ztemp,np.repeat(0,len(Ztemp))]
        if likelr is None: likelr = lr * 4 #probably can optimise this a little quicker?
        self.likeoptimizer = tf.keras.optimizers.Adam(learning_rate=likelr,amsgrad=False)

    self.fullX = X
```

```
                if (self.mulike is None) or (it%50<25): #optimise latent fns
                    gradients = tape.gradient(elbo_loss, [self.mu,self.scale])
                    self.optimizer.apply_gradients(zip(gradients, [self.mu, self.scale]))
                else: #this optimises the likelihood...
                    if self.likemodel=='distribution':
                        gradients = tape.gradient(elbo_loss, [self.mulike])
                        self.likeoptimizer.apply_gradients(zip(gradients, [self.mulike]))
                    if self.likemodel=='process':
                        gradients = tape.gradient(elbo_loss, [self.mulike,self.scalelike])
                        self.likeoptimizer.apply_gradients(zip(gradients, [self.mulike,self.scalelike]))
```

Snippet of a module I wrote for calibrating air pollution sensors.

Typically though, we use a library optimisation algorithm, such as ADAM, which will adjust the learning rate to help improve convergence.

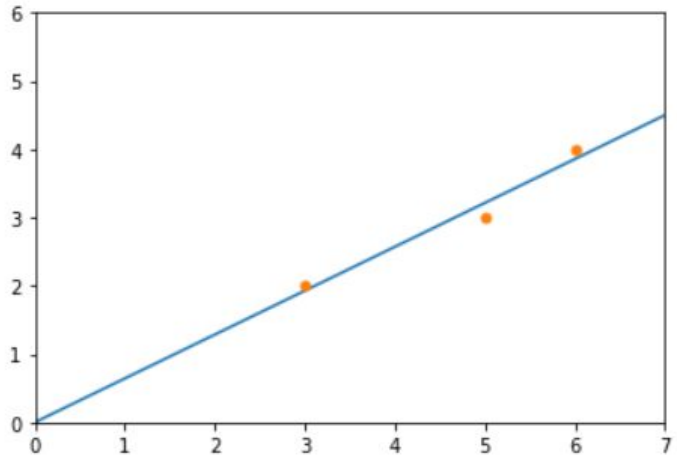
Other tricks

Finally note: Most optimisers will work better if the inputs (features) are normalised.

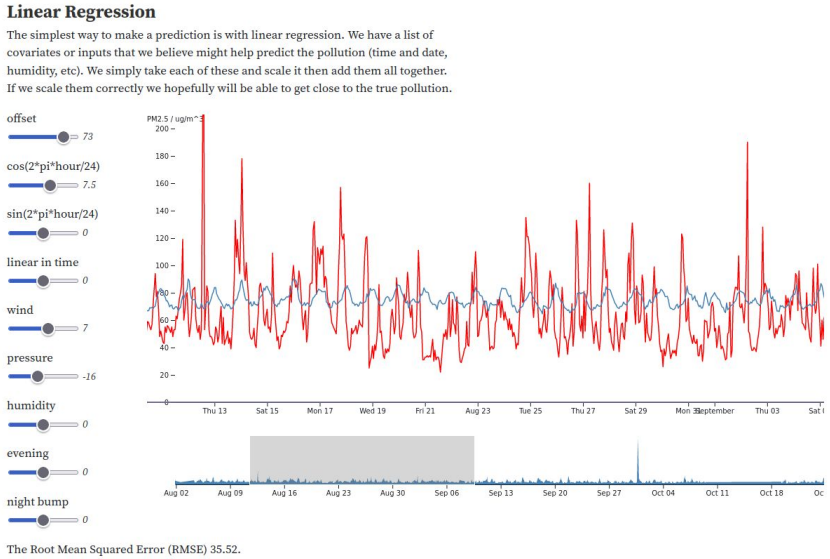
```
w = [-1,4] #initial value
for it in range(10000):
    grad = costfn_grad(w,x,y)
    w = w - grad*0.01
```

```
xtest = np.linspace(0,7,10)
ytest = getprediction(w,xtest)
plt.plot(xtest,ytest,'-')
plt.plot(x,y,'.',markersize=10)
plt.xlim([0,7])
plt.ylim([0,6])
```

```
(0.0, 6.0)
```



break



Linear regression: try predicting the air pollution in this example by manually adjusting the parameters (weights) that say how much to scale each feature.

Maximum Likelihood Estimation

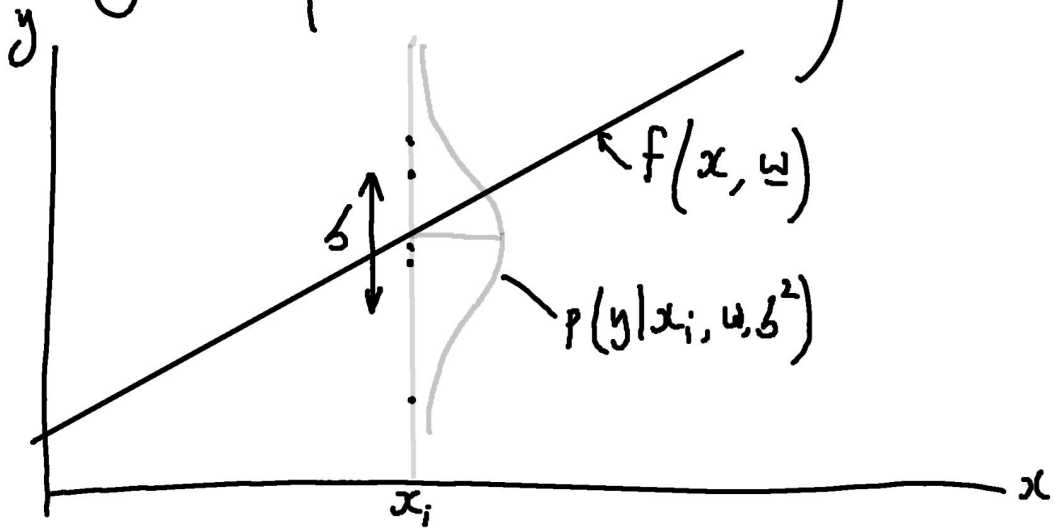
We said that our noise (error) term was assumed to be Gaussian distributed.

If we assume this, then we can come at the problem from a different angle: Trying to maximise the probability that the model generated the data, by adjusting model parameters.

We said our noise (error) term
was assumed to be Gaussian distributed,

We said our noise (error) term was assumed to be Gaussian distributed,

$$y_i = N\left(f(\underline{x}_i, \underline{w}),\ b^2\right)$$



So, for a given x_i and $\underline{w}$, we can say how probable a given value of y_i is:

So, for a given x_i and $\underline{w}$, we can say how probable a given value of y_i is:

$$p(y_i \mid x_i, \underline{w}, \sigma^2) = \mathcal{N}(y \mid f(x_i, \underline{w}), \sigma^2)$$

So, for a given x_i and $\underline{w}$, we can say how probable a given value of y_i is:

$$p(y_i \mid x_i, \underline{w}, \sigma^2) = N(y \mid f(x_i, \underline{w}), \sigma^2)$$

If we assume our data points are <u>independent</u>, then

$$p(y_1 \ldots y_n \mid x_1 \ldots x_n, \underline{w}, \sigma^2) = \prod_{i=1}^{n} p(y_i \mid x_i, \underline{w}, \sigma^2)$$

So, for a given x_i and $\underline{w}$, we can say how probable a given value of y_i is:

$$p(y_i \mid x_i, \underline{w}, \sigma^2) = N(y \mid f(x_i, \underline{w}), \sigma^2)$$

If we assume our data points are <u>independent</u>, then

$$p(y_1 \ldots y_n \mid x_1 \ldots x_n, \underline{w}, \sigma^2) = \prod_{i=1}^{n} p(y_i \mid x_i, \underline{w}, \sigma^2)$$

We assume our errors follow the <u>same distribution</u>:

i.i.d. → independent & identically distributed

So, for a given x_i and $\underline{w}$, we can say how probable a given value of y_i is:

$$p(y_i \mid x_i, \underline{w}, \sigma^2) = N(y \mid f(x_i, \underline{w}), \sigma^2)$$

If we assume our data points are <u>independent</u>, then

$$p(y_1 \ldots y_n \mid x_1 \ldots x_n, \underline{w}, \sigma^2) = \prod_{i=1}^{n} p(y_i \mid x_i, \underline{w}, \sigma^2)$$

We assume our errors follow the <u>same distribution</u>:

i.i.d. → independent & identically distributed

In this case a Normal distribution with variance σ^2

So, for a given x_i and $\underline{w}$, we can say how probable a given value of y_i is:

$$p(y_i \mid x_i, \underline{w}, \sigma^2) = N(y \mid f(x_i, \underline{w}), \sigma^2)$$

If we assume our data points are <u>independent</u>, then

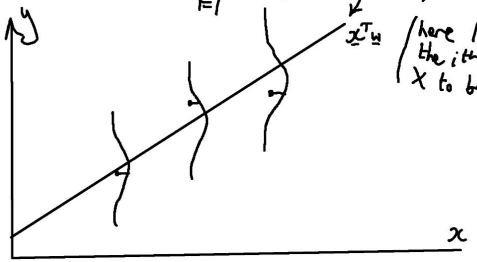
$$p(y_1 \ldots y_n \mid x_1 \ldots x_n, \underline{w}, \sigma^2) = \prod_{i=1}^{N} p(y_i \mid x_i, \underline{w}, \sigma^2)$$

We assume our errors follow the <u>same distribution</u>:

i.i.d. → independent & identically distributed

In this case a Normal distribution with variance σ^2

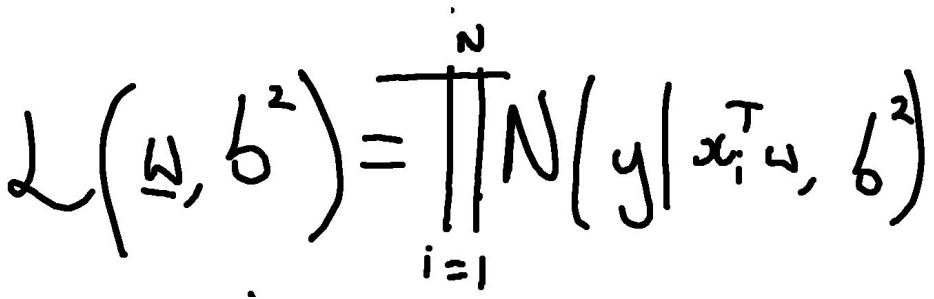
$$p(\underline{y} \mid X, \underline{w}, \sigma^2) = \prod_{i=1}^{N} N(y_i \mid x_i^T \underline{w}, \sigma^2)$$



(here I've taken the i^{th} row of X to be x_i)

Likelihood

Before, we wanted to minimise a cost function. Here we want to maximise the probability that the data came from our model (by adjusting our model). Because we are interested in this probability as a function of **w** and σ^2, it won't integrate to one over these parameters. So it's not a probability distribution. We instead call it the **likelihood function,**



Maximum likelihood estimation selects parameters (in this case, **w** and σ^2) to maximise the likelihood function.

PDF for a Normal

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

The Normal Distribution

Looks tricky, but let's step through how you might work this out from scratch…

PDF for a Normal

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Looks tricky, but let's step through how you might work this out from scratch…

PDF for a Normal

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

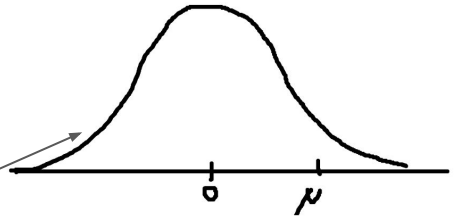
Gaussian shape → e^{-x^2}

Looks tricky, but let's step through how you might work this out from scratch…

PDF for a Normal

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Gaussian shape → e^{-x^2}



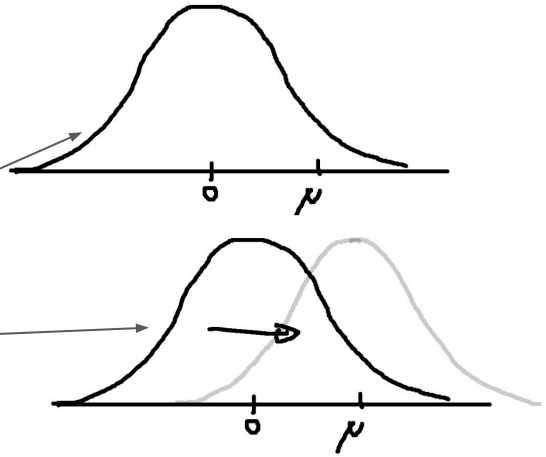
Looks tricky, but let's step through how you might work this out from scratch…

PDF for a Normal

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Gaussian shape → e^{-x^2}

shift it by mean → $e^{-(x-\mu)^2}$

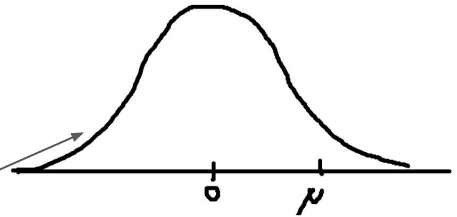


Looks tricky, but let's step through how you might work this out from scratch…

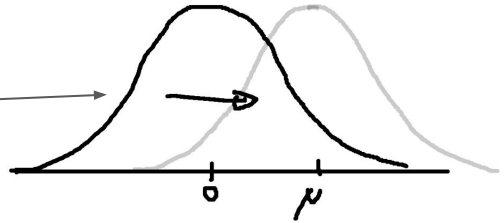
PDF for a Normal

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

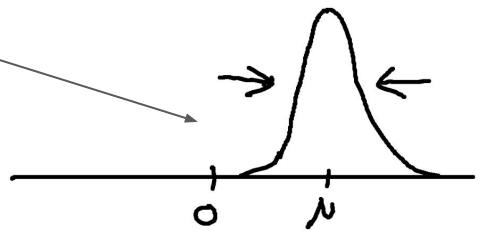
Gaussian shape → e^{-x^2}



shift it by mean → $e^{-(x-\mu)^2}$



Make it narrow or wide → $e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$



Looks tricky, but let's step through how you might work this out from scratch…

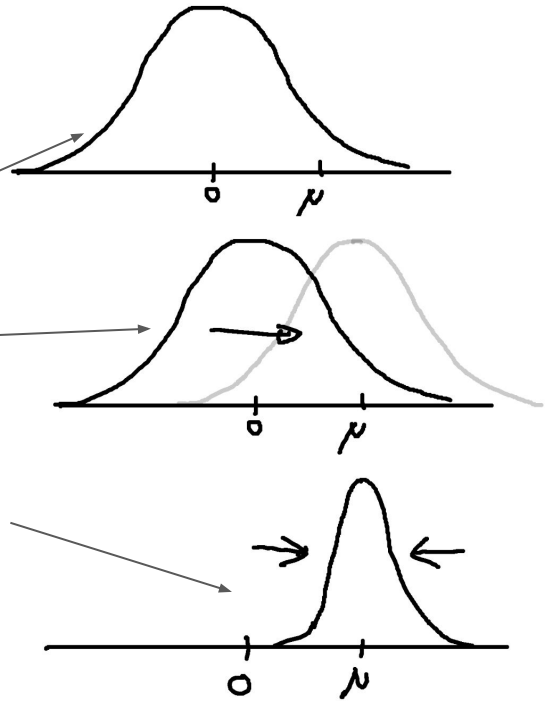
PDF for a Normal

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$

Gaussian shape → e^{-x^2}

shift it by mean → $e^{-(x-\mu)^2}$

$$\frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2}$$



Need a normalising term:
E.g. If σ is big, then our Gaussian is wide, so we need to make it less tall.

Let’s go back now to our expression for the likelihood

$$L(\underline{w}, \sigma^2) = \prod_{i=1}^{N} N(y | x_i^T w, \sigma^2)$$

Remember we had assumed iid for our noise, so we can say that the joint probability of all the observations is the product of the individual probabilities.

Let's go back now to our expression for the likelihood

$$\mathcal{L}(\underline{w}, \sigma^2) = \prod_{i=1}^{N} \mathcal{N}(y | x_i^T w, \sigma^2)$$

Maximum Likelihood Estimation selects $\underline{w}$ & σ^2 to maximise $\mathcal{L}$:

Let's go back now to our expression for the likelihood

$$L(\underline{w}, \sigma^2) = \prod_{i=1}^{N} N(y_i | x_i^T w, \sigma^2)$$

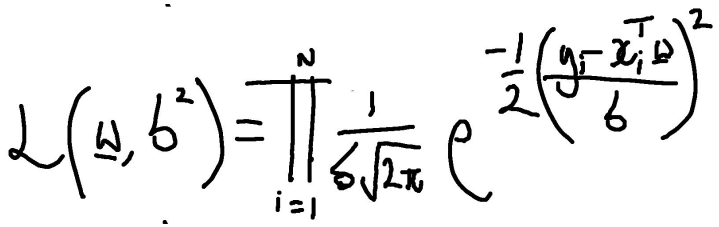
Maximum Likelihood Estimation selects $\underline{w}$ & σ^2 to maximise L:

Let's substitute in the expression for the Normal distribution...

$$L(\underline{w}, \sigma^2) = \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T w}{\sigma}\right)^2}$$

Log Likelihood

We want to differentiate the likelihood with respect to **w** and σ^2, but in its current form that is quite tricky.



Log Likelihood

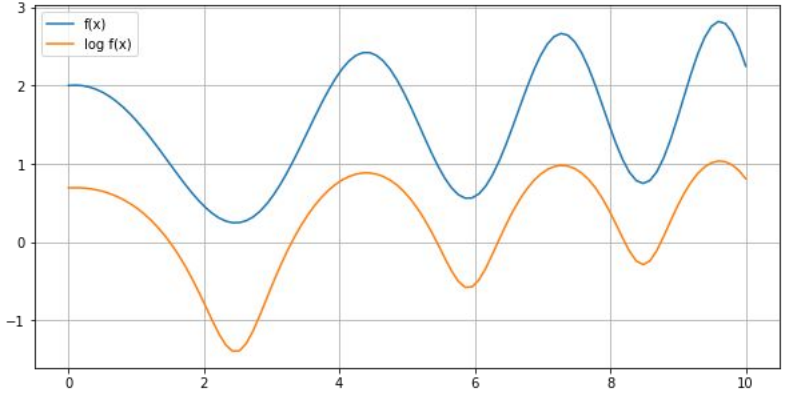
We want to differentiate the likelihood with respect to **w** and σ^2, but in its current form that is quite tricky.

The location of the maximum of L is in the same place as the maximum of log L (as log is monotonic).

So we first, find the log of the likelihood, then differentiate *that*…

ACTIVITY: Compute the log likelihood. Hint: Start by thinking about how the log product of things is the sum of the log of things.

$$L(\underline{w}, b^2) = \prod_{i=1}^{N} \frac{1}{b\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - \underline{x}_i^T \underline{b}}{b}\right)^2}$$



An example function, and its log, to demonstrate that the maxima and minima are in the same places!

The log likelihood (LL)

$$\log L\left(\underline{w}, \sigma^2\right) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{-\frac{1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum \left(y_i - x_i^T \underline{w}\right)^2$$

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum \left(y_i - x_i^T \underline{w}\right)^2$$

Let's write using vector notation...

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum \left(y_i - x_i^T \underline{w}\right)^2$$

Let's write using vector notation...

$$LL = -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} (y - X_w)^T (y - X_w)$$

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum (y_i - x_i^T \underline{w})^2$$

Let's write using vector notation...

$$LL = -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} (y - X_w)^T (y - X_w)$$

$$y^T y - y^T X_w - w^T X^T y + w^T X^T X_w$$

differentiate…

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum (y_i - x_i^T \underline{w})^2$$

Let's write using vector notation...

$$LL = -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} (y - X_w)^T (y - X_w)$$

$$y^T y - y^T X_w - w^T X^T y + w^T X^T X_w$$

differentiate…

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum (y_i - x_i^T \underline{w})^2$$

Let's write using vector notation...

$$LL = -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} (y - X\underline{w})^T (y - X\underline{w})$$

differentiate…

$$y^T y - y^T X\underline{w} - \underline{w}^T X^T y + \underline{w}^T X^T X\underline{w}$$

$$\frac{\partial}{\partial \underline{w}} = -X^T y - X^T y + 2X^T X\underline{w}$$

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum (y_i - x_i^T \underline{w})^2$$

Let's write using vector notation...

$$LL = -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} (y - X\underline{w})^T (y - X\underline{w})$$

differentiate…

$$y^T y - y^T X\underline{w} - \underline{w}^T X^T y + \underline{w}^T X^T X\underline{w}$$

$$\frac{\partial}{\partial \underline{w}} = -X^T y - X^T y + 2X^T X\underline{w}$$

$$= -2X^T (y - X\underline{w})$$

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum (y_i - x_i^T \underline{w})^2$$

Let's write using vector notation...

$$LL = -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} (y - X\underline{w})^T (y - X\underline{w})$$

differentiate…

$$y^T y - y^T X\underline{w} - \underline{w}^T X^T y + \underline{w}^T X^T X\underline{w}$$

$$\frac{\partial}{\partial \underline{w}} = -X^T y - X^T y + 2X^T X\underline{w}$$

$$= -2X^T (y - X\underline{w})$$

$$X^T y = X^T X\underline{w}$$

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum (y_i - x_i^T \underline{w})^2$$

Let's write using vector notation...

$$LL = -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} (y - X_w)^T (y - X_w)$$

differentiate…

$$y^T y - y^T X w - w^T X^T y + w^T X^T X w$$

$$\frac{\partial}{\partial w} = -X^T y - X^T y + 2X^T X w$$

$$= -2X^T (y - Xw)$$

$$X^T y = X^T X w$$

$$(X^T X)^{-1} X^T y = w$$

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{\frac{-1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum \left(y_i - x_i^T \underline{w}\right)^2$$

Let's write using vector notation...

$$LL = -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} (y - X_w)^T (y - X_w)$$

$$y^T y - y^T X_w - w^T X^T y + w^T X^T X_w$$

$$\frac{\partial}{\partial w} = -X^T y - X^T y + 2X^T X_w$$

$$= -2X^T (y - X_w)$$

$$X^T y = X^T X_w$$

Same as before!

$$(X^T X)^{-1} X^T y = w$$

The log likelihood (LL)

$$\log L(\underline{w}, \sigma^2) = \sum_{i=1}^{N} \log \prod_{i=1}^{N} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= \sum_{i=1}^{N} -\log \sigma\sqrt{2\pi} + \log e^{-\frac{1}{2}\left(\frac{y_i - x_i^T \underline{w}}{\sigma}\right)^2}$$

$$= -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} \sum \left(y_i - x_i^T \underline{w}\right)^2$$

Let's write using vector notation...

$$LL = -N \log \sigma\sqrt{2\pi} - \frac{1}{2\sigma^2} (y - Xw)^T (y - Xw)$$

$y^T y - y^T X w - w^T X^T y + w^T X^T X w$

Same as before!

$$\frac{\partial}{\partial w} = -X^T y - X^T y + 2X^T X w$$

$$= -2X^T (y - Xw)$$

$$X^T y = X^T X w$$

$$(X^T X)^{-1} X^T y = w$$

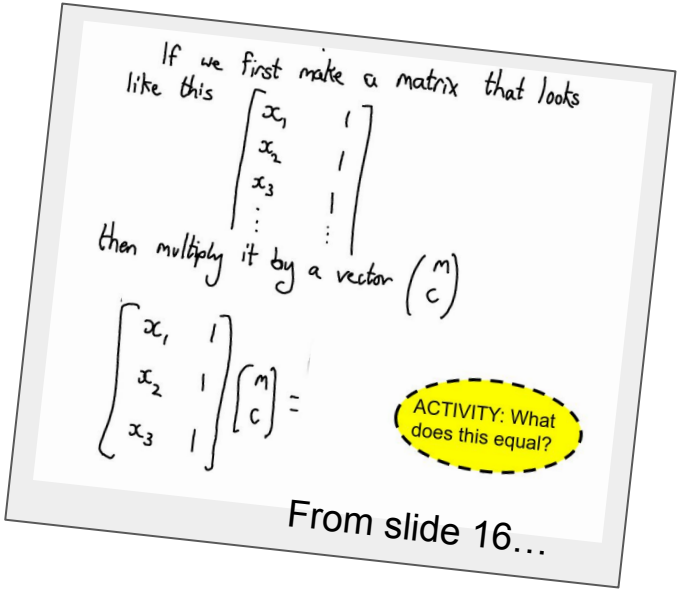
We can differentiate and solve for σ^2 in a similar way.

Summary: Maximum Likelihood Estimation and Ordinary Least Squares

If we have Gaussian-distributed i.i.d. noise, minimising the sum squared error (i.e. “ordinary least squares”) is the maximum likelihood estimator (for linear regression).

Basis Functions

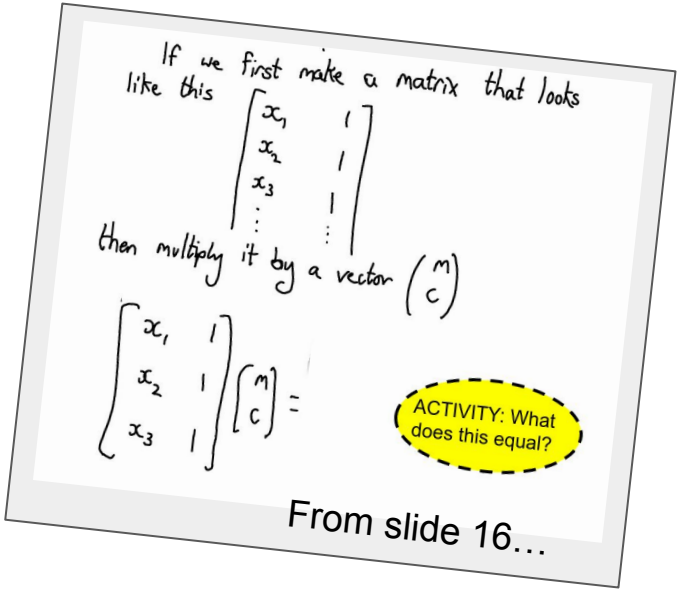
We've already used a basis, the 1st-order polynomial had two bases: x^1 and x^0.



Basis Functions

We've already used a basis, the 1st-order polynomial had two bases: x^1 and x^0.

We could add another column, with the values of x^2.
This will lead to a 2nd-order polynomial.



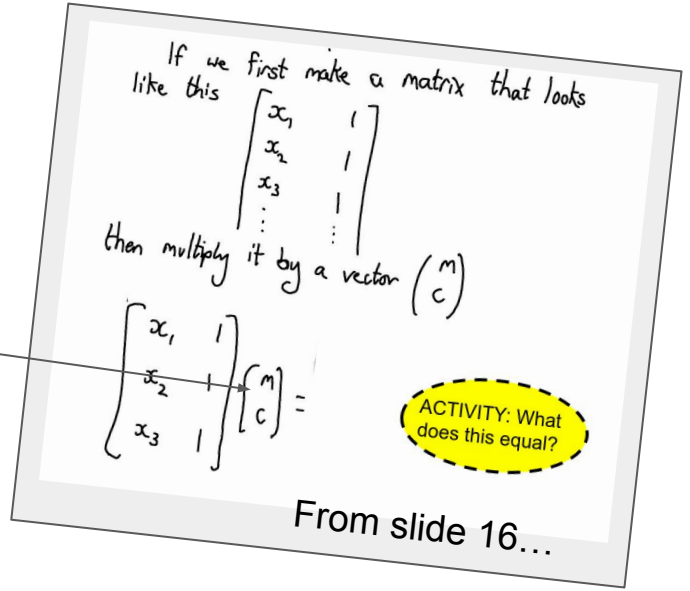
Basis Functions

We've already used a basis, the 1st-order polynomial had two bases: x^1 and x^0.

We could add another column, with the values of x^2. This will lead to a 2nd-order polynomial.

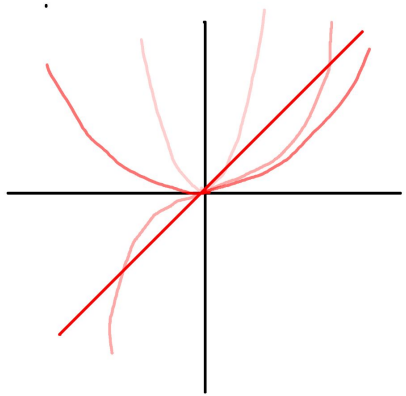
The solution is the same expression as before.

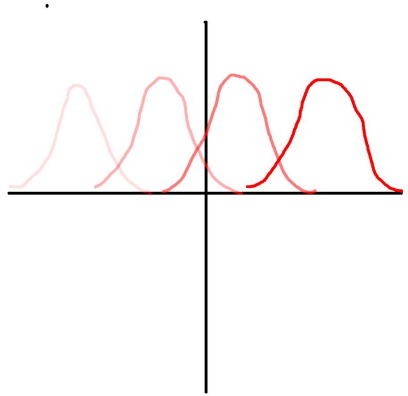
Our parameter vector would just need another number (to describe the contribution of the quadratic term).

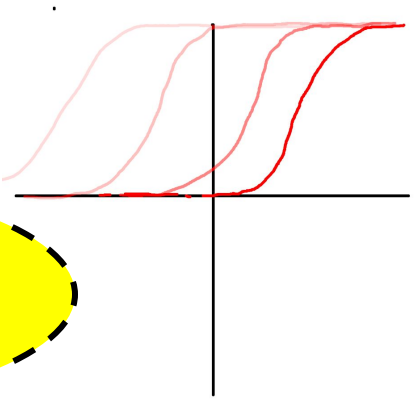


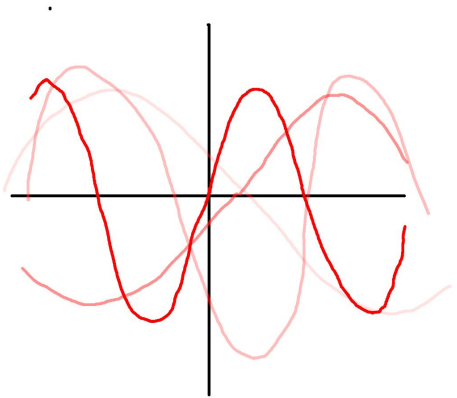
Example Basis Functions

- Polynomial
- Gaussian
- Fourier
- Sigmoid





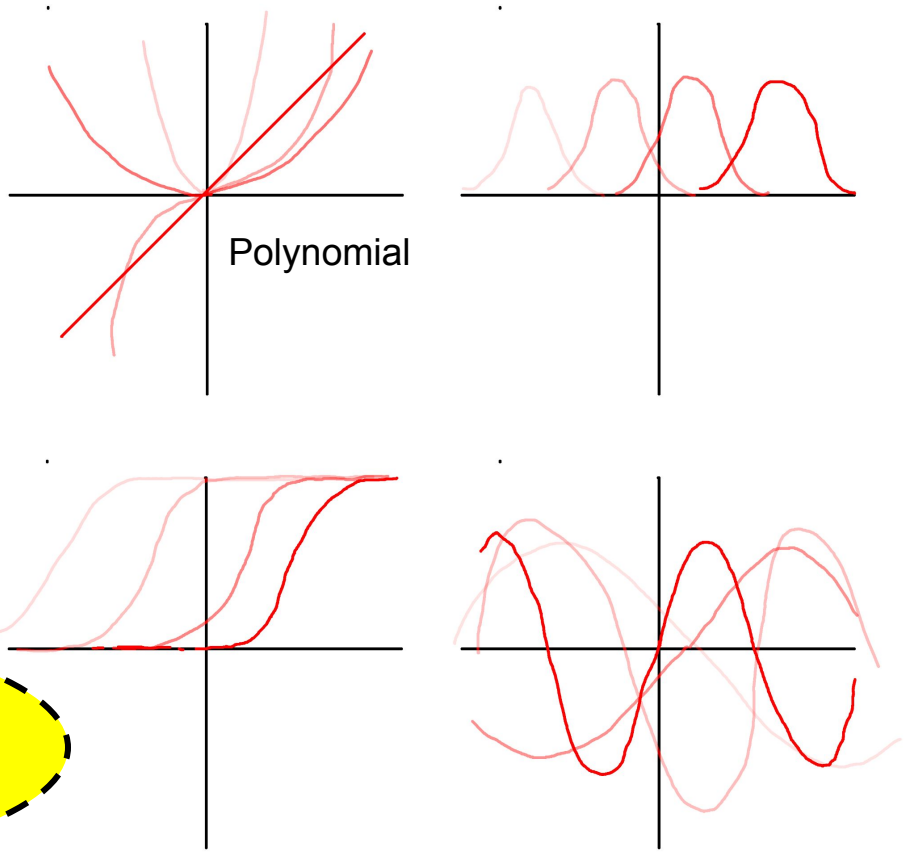




ACTIVITY: Match the words to the bases. One of them is a ‘random’ basis (so not picked from a grid)

Example Basis Functions

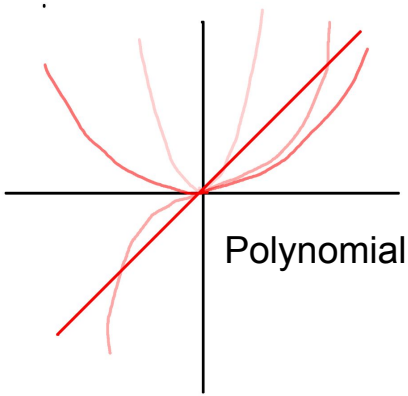
- Polynomial
- Gaussian
- Fourier
- Sigmoid

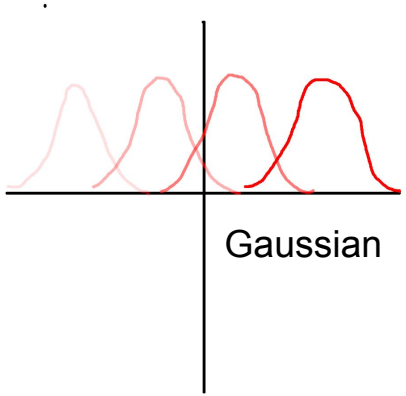


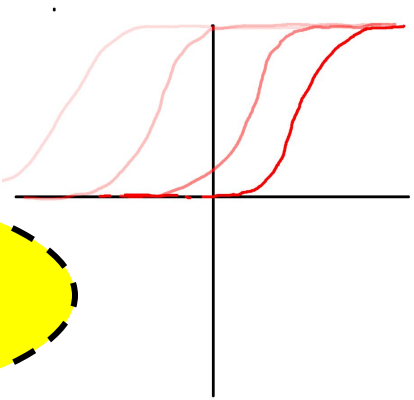
ACTIVITY: Match the words to the bases. One of them is a ‘random’ basis (so not picked from a grid)

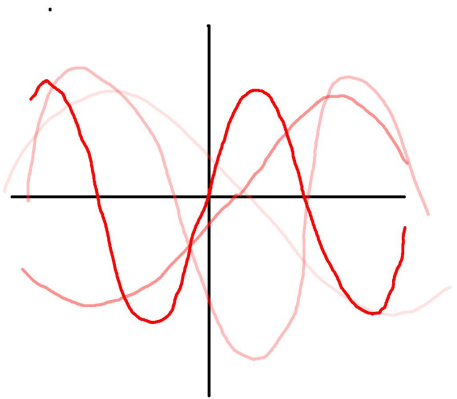
Example Basis Functions

- Polynomial
- Gaussian
- Fourier
- Sigmoid





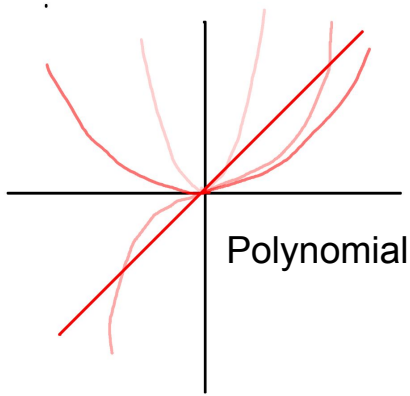


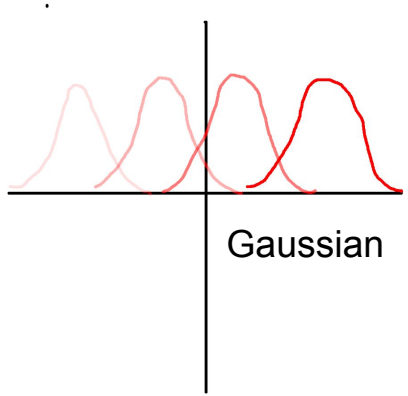


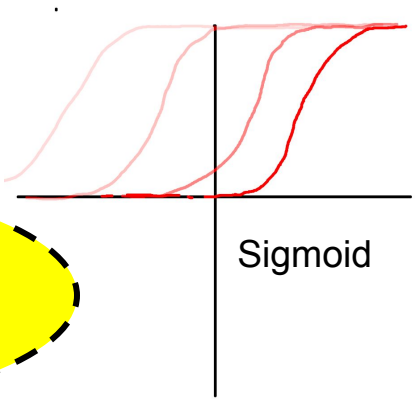
ACTIVITY: Match the words to the bases. One of them is a ‘random’ basis (so not picked from a grid)

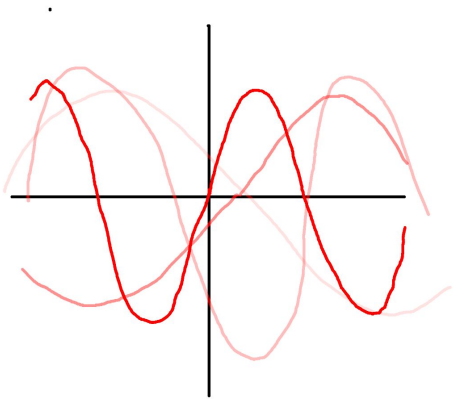
Example Basis Functions

- Polynomial
- Gaussian
- Fourier
- Sigmoid





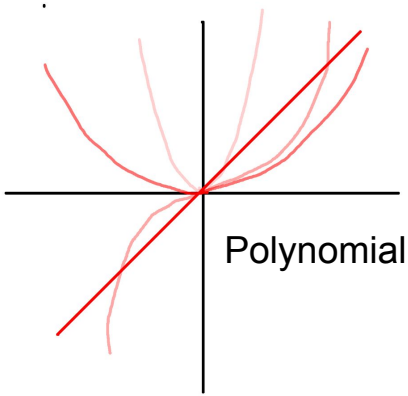


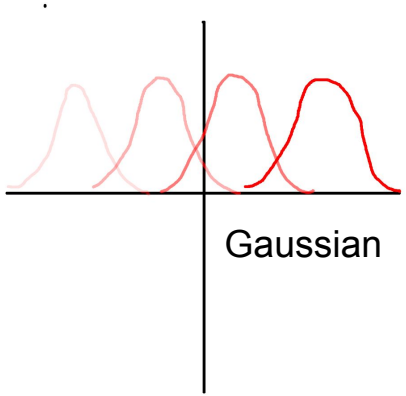


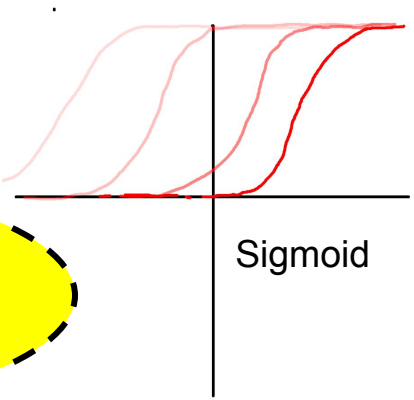
ACTIVITY: Match the words to the bases. One of them is a ‘random’ basis (so not picked from a grid)

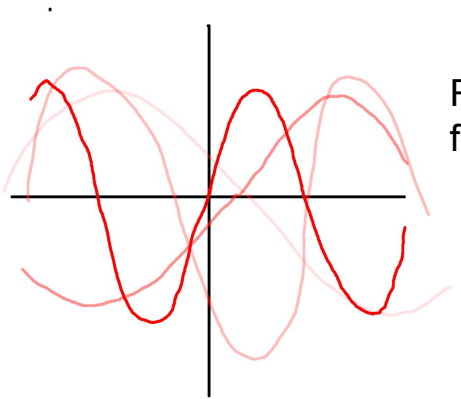
Example Basis Functions

- Polynomial
- Gaussian
- Fourier
- Sigmoid









Random fourier.

ACTIVITY: Match the words to the bases. One of them is a ‘random’ basis (so not picked from a grid)

Regularisation

ACTIVITY: Explore the effect of the number of parameters: consider the score on the training and validation sets.

A common problem is ‘overfitting’. You can get parameter values that are very large to explain data.

See the example.

To help avoid this we can add an additional term to the cost function that penalises lots of large parameters.

$$h(w) = E(w) + \lambda R(w)$$

λ: Regulariser parameter (maybe set using validation set?).

R: regulariser

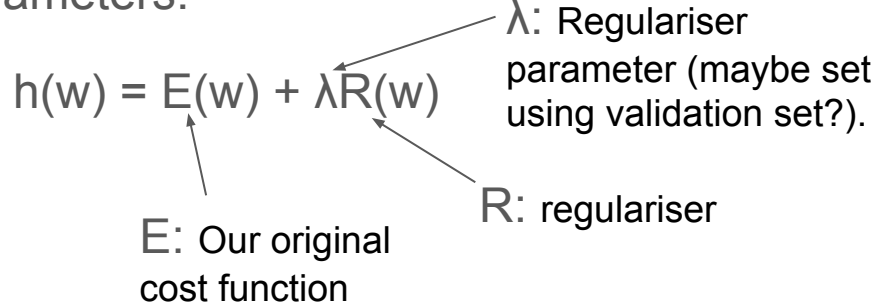
E: Our original cost function

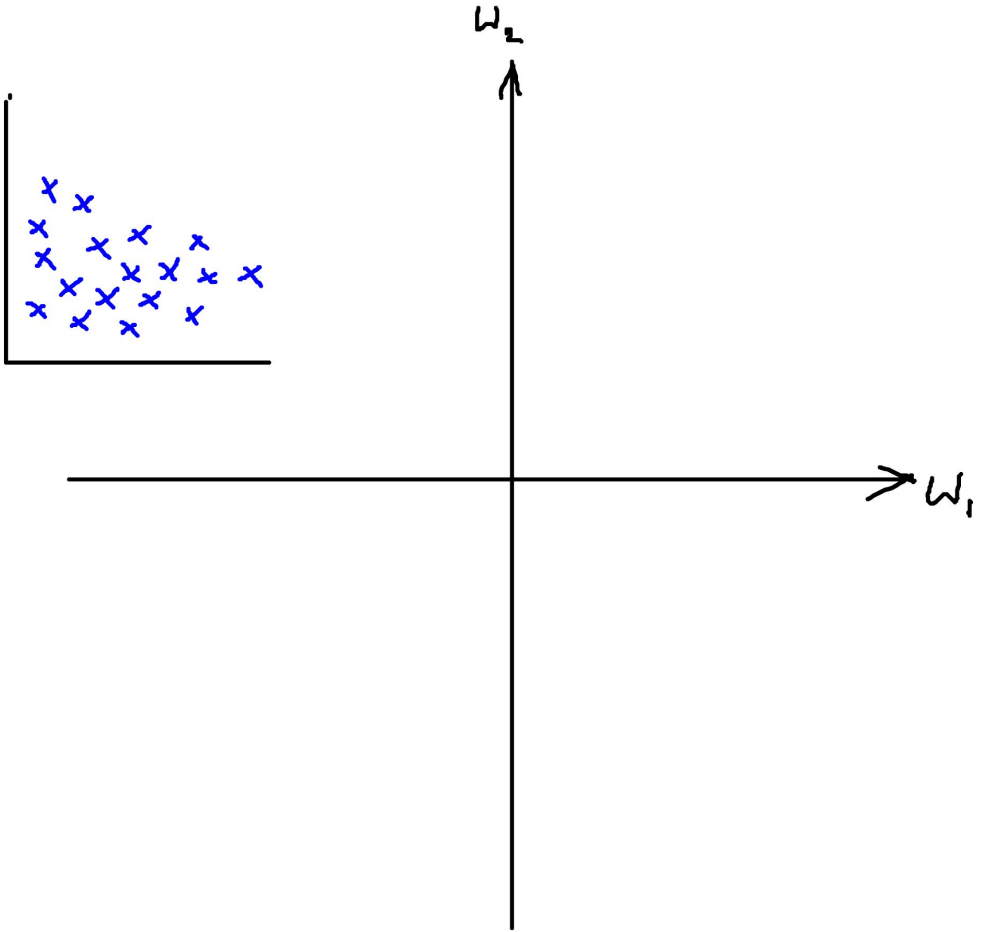
Regularisation

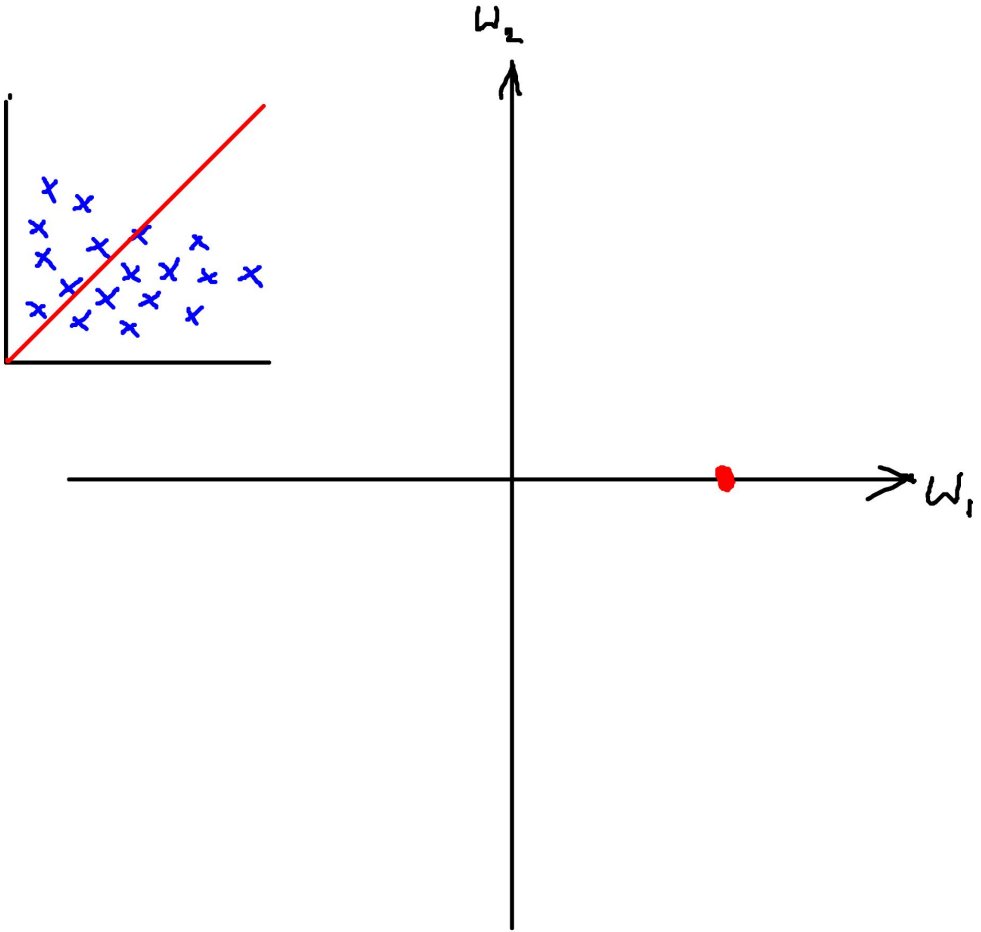
A common problem is ‘overfitting’. You can get parameter values that are very large to explain data.

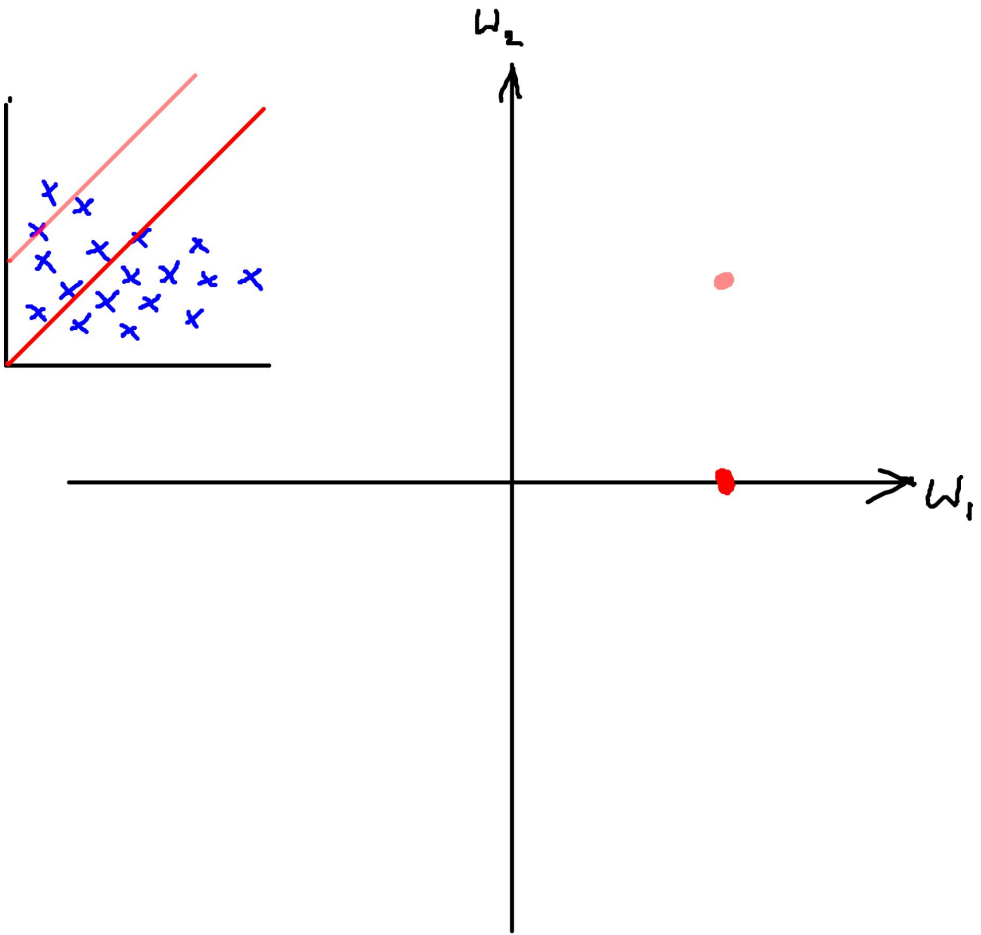
See the example.

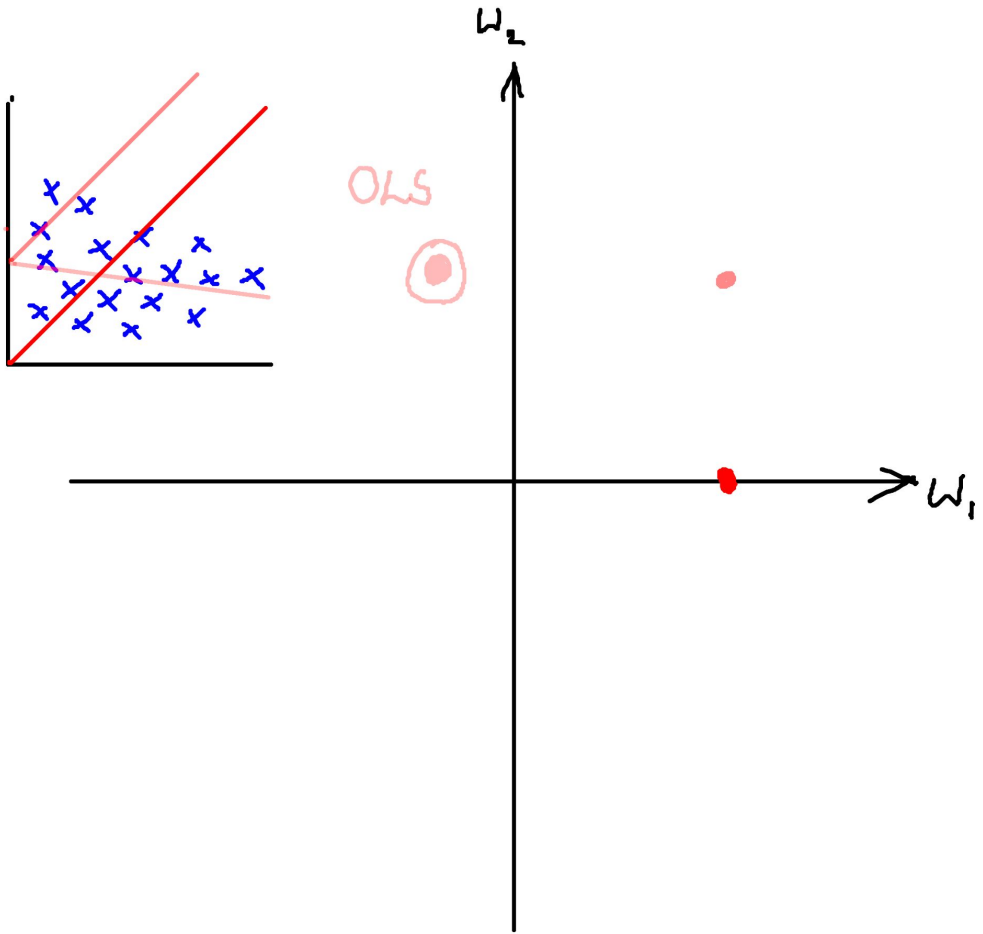
To help avoid this we can add an additional term to the cost function that penalises lots of large parameters.

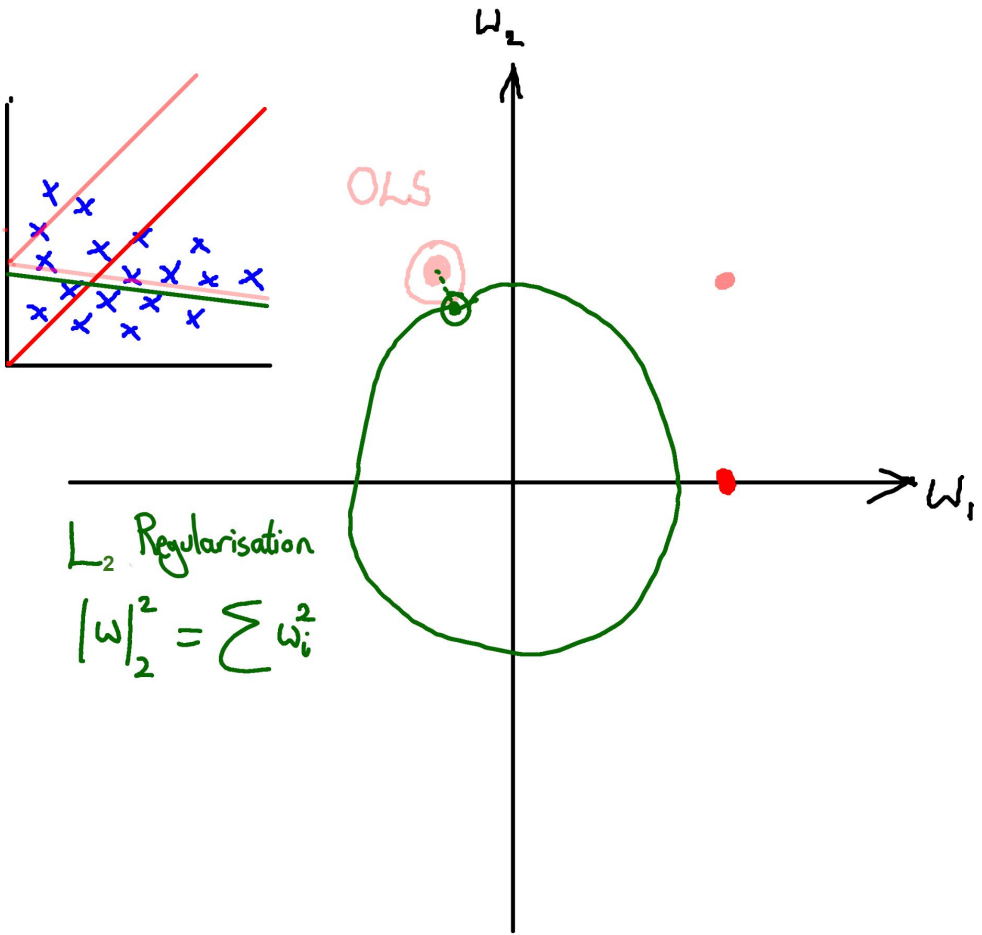



w_2
w_1


w_2
w_1

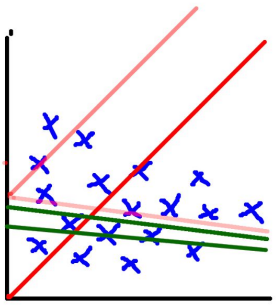

w_2
w_1


w_2
OLS
w_1



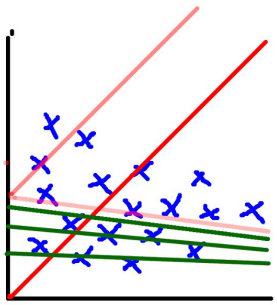
l_2 regularisation - also called ridge regression.

In this toy example we can't see much benefit. But with many bases (or parameters) this often is absolutely necessary to make the regression work.



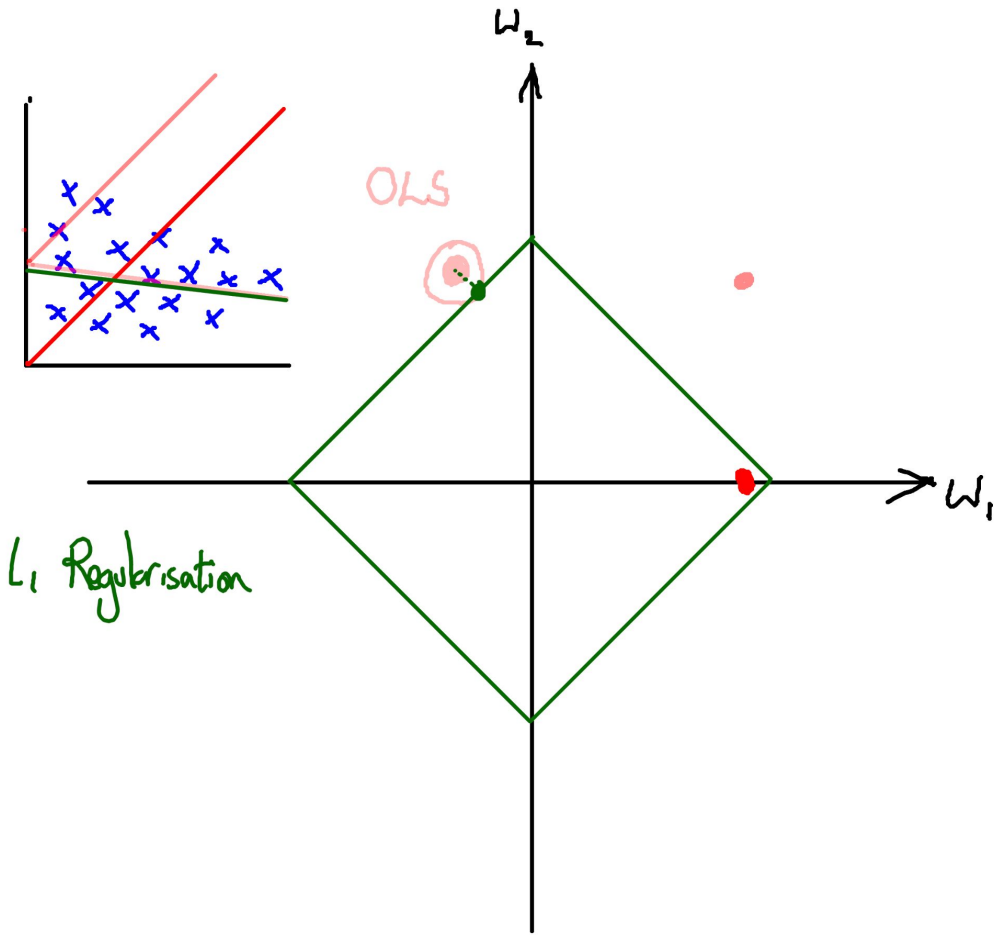
L_2 Regularisation

$$|w|_2^2 = \sum w_i^2$$

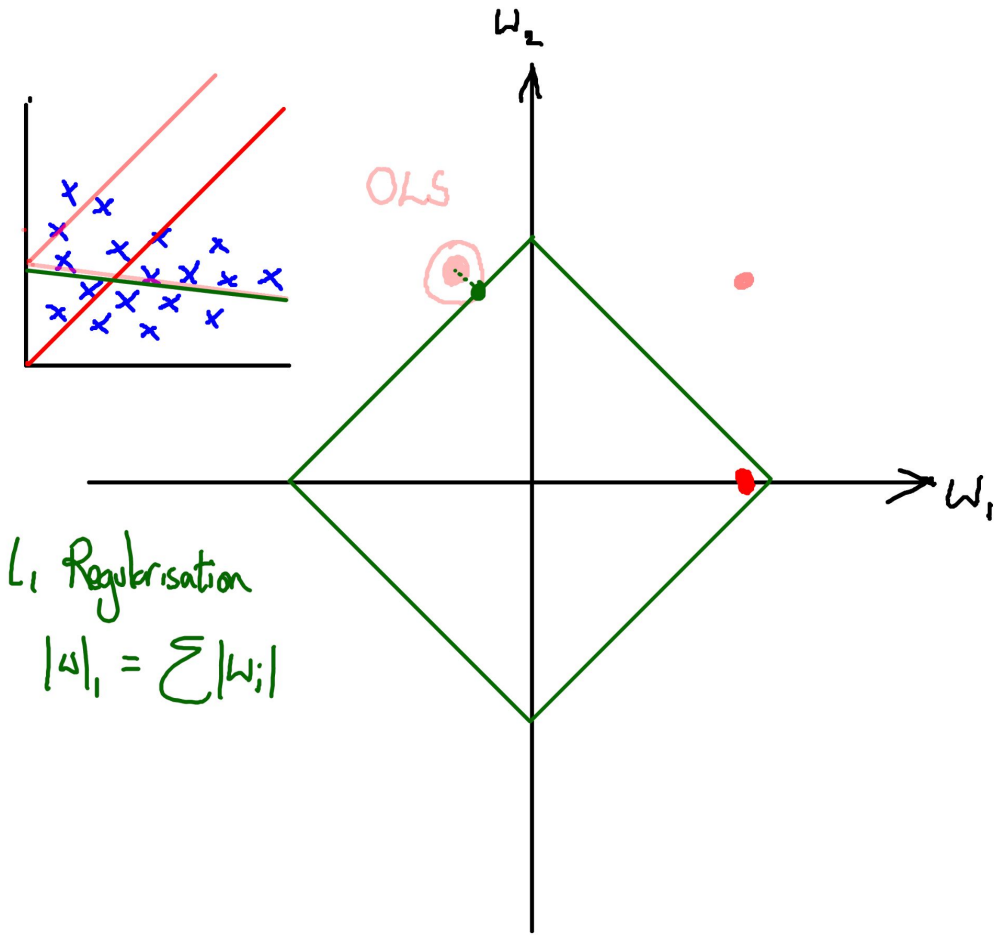


L_2 Regularisation

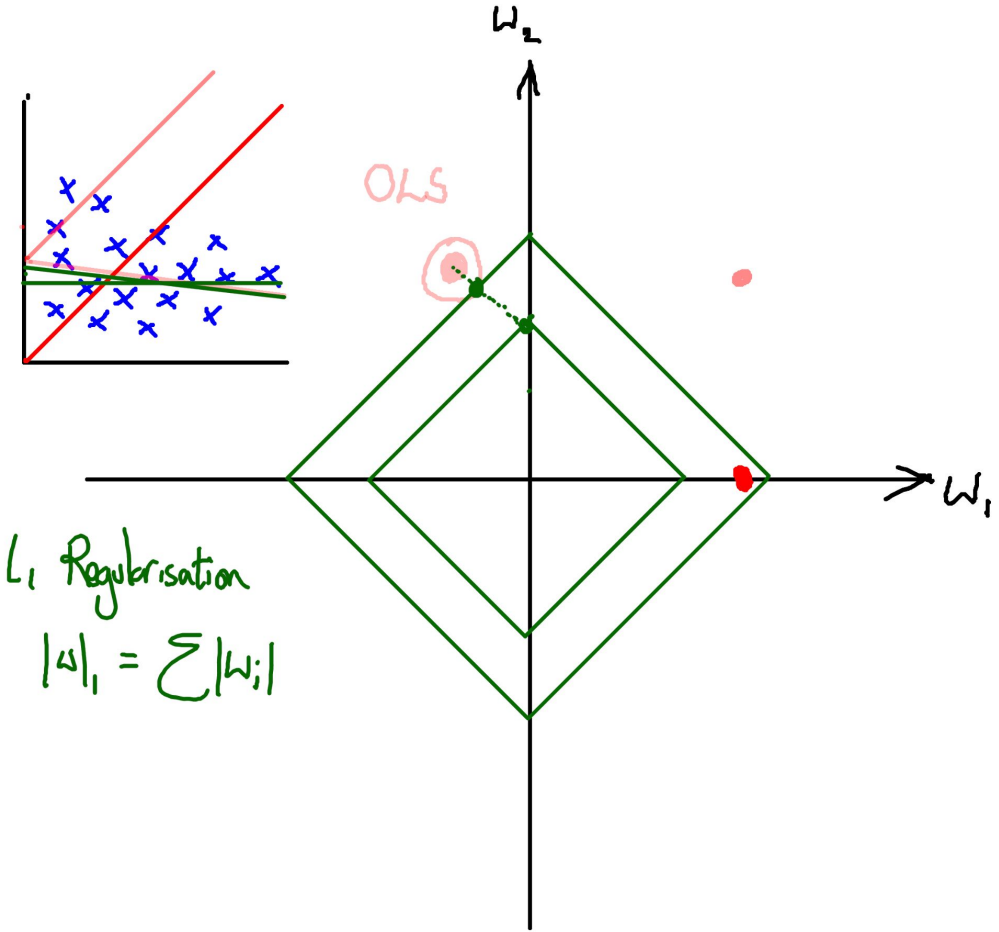
$$|w|_2^2 = \sum w_i^2$$



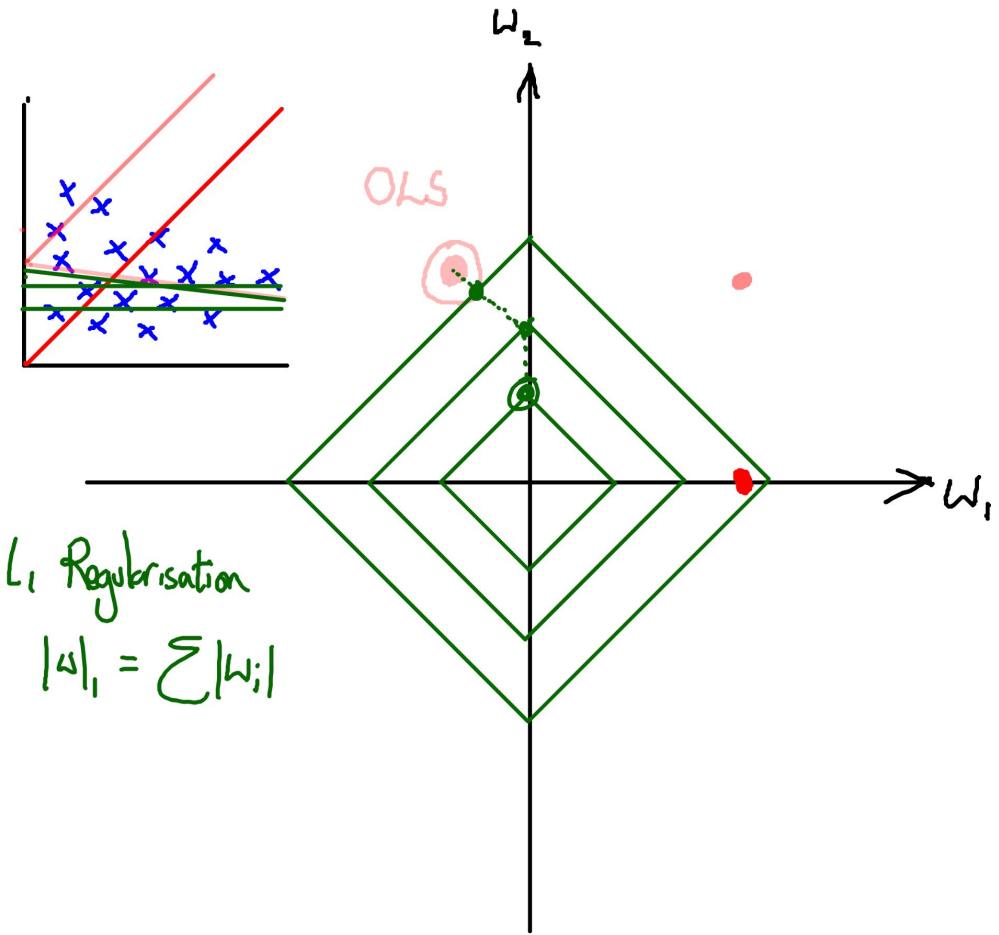
l_1 regularisation has the property that it will push many of the parameters to zero. This can be useful if you think only a few features are relevant.



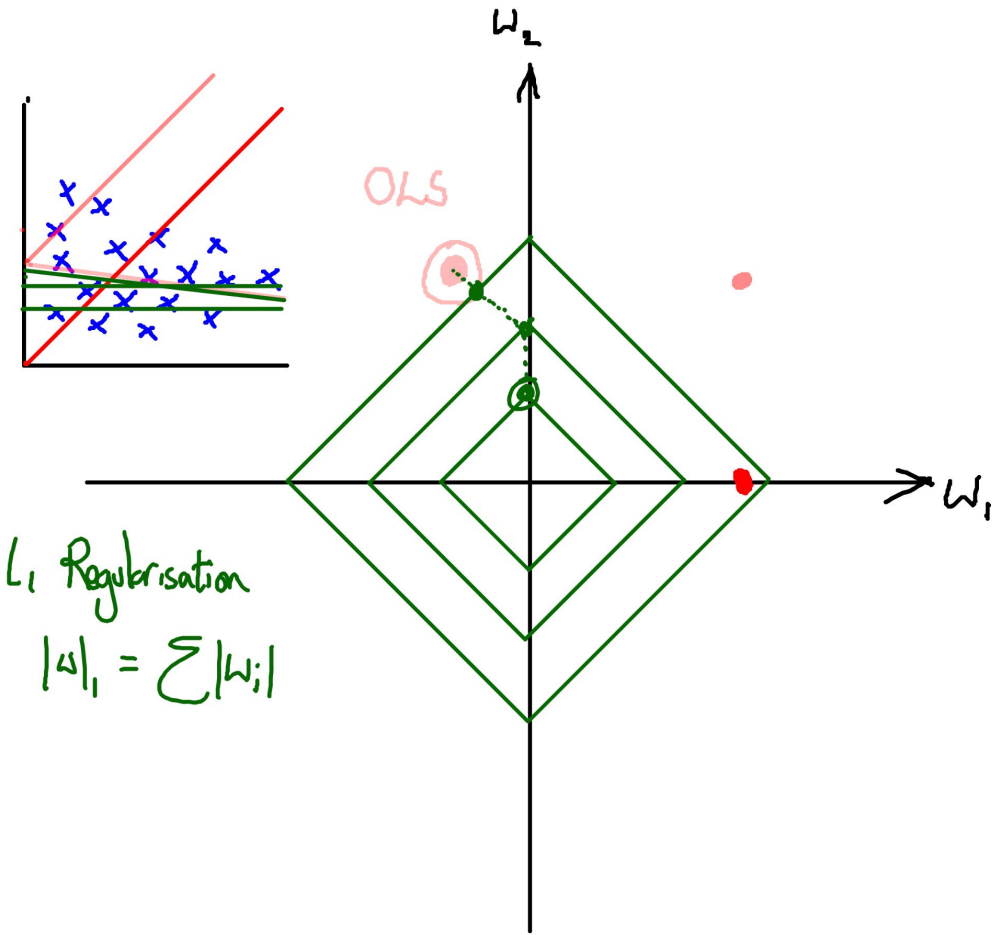
l_1 regularisation has the property that it will push many of the parameters to zero. This can be useful if you think only a few features are relevant.



l_1 regularisation has the property that it will push many of the parameters to zero. This can be useful if you think only a few features are relevant.



l_1 regularisation has the property that it will push many of the parameters to zero. This can be useful if you think only a few features are relevant.



l_1 regularisation has the property that it will push many of the parameters to zero. This can be useful if you think only a few features are relevant.

Elastic net regularisation has some l_1 and some l_2, mixed together.

Other approaches exist.

Summary / Take home messages

- We can write the sum square cost function as $E = (y - X\omega)^T (y - X\omega)$

Summary / Take home messages

- We can write the sum square cost function as $E = (y - X\omega)^T (y - X\omega)$
- We can differentiate, set to zero, and find a closed form solution for linear regression.

Summary / Take home messages

- We can write the sum square cost function as $E = (y - X\omega)^T (y - X\omega)$
- We can differentiate, set to zero, and find a closed form solution for linear regression.
- This is the same solution as when we use the maximum likelihood approach.

Summary / Take home messages

- We can write the sum square cost function as $E = (y - X\omega)^T (y - X\omega)$
- We can differentiate, set to zero, and find a closed form solution for linear regression.
- This is the same solution as when we use the maximum likelihood approach.
- If we don’t have a closed form solution, we need to optimise our parameters iteratively.

Summary / Take home messages

- We can write the sum square cost function as $E = (y - X\omega)^T (y - X\omega)$
- We can differentiate, set to zero, and find a closed form solution for linear regression.
- This is the same solution as when we use the maximum likelihood approach.
- If we don’t have a closed form solution, we need to optimise our parameters iteratively.
- Steepest descent is simple but often is slow.

Summary / Take home messages

- We can write the sum square cost function as $E = (y - X\omega)^T (y - X\omega)$
- We can differentiate, set to zero, and find a closed form solution for linear regression.
- This is the same solution as when we use the maximum likelihood approach.
- If we don’t have a closed form solution, we need to optimise our parameters iteratively.
- Steepest descent is simple but often is slow.
- Other options: line search, Newton’s method, conjugate gradient descent.

Summary / Take home messages

- We can write the sum square cost function as $E = (y - X\omega)^T (y - X\omega)$
- We can differentiate, set to zero, and find a closed form solution for linear regression.
- This is the same solution as when we use the maximum likelihood approach.
- If we don’t have a closed form solution, we need to optimise our parameters iteratively.
- Steepest descent is simple but often is slow.
- Other options: line search, Newton’s method, conjugate gradient descent.
- If we have too much data we could use mini-batching and stochastic gradient descent.

Summary / Take home messages

- We can write the sum square cost function as $E = (y - X\omega)^T (y - X\omega)$
- We can differentiate, set to zero, and find a closed form solution for linear regression.
- This is the same solution as when we use the maximum likelihood approach.
- If we don’t have a closed form solution, we need to optimise our parameters iteratively.
- Steepest descent is simple but often is slow.
- Other options: line search, Newton’s method, conjugate gradient descent.
- If we have too much data we could use mini-batching and stochastic gradient descent.
- If the model overfits, try regularising.