Mathematics for Machine Learning (70015)

Lecture 1.1 - Linear Regression Intro

Linear regression aims to provide a solution to the supervised learning problem; we are given a dataset of N examples of inputs and expected outputs, with a goal of predicting the correct output for a new input. Examples of this include image classification (such as digit classification) and translation. A curve fitting problem in 1-dimension has an input space $\in \mathbb{R}$, and an output space $\in \mathbb{R}$.

To tackle this problem mathematically, we need to first describe the curve fitting problem mathematically. As each input is associated with a single output, this is equivalent to a function in mathematics. We are given a dataset of N pairs, of inputs and outputs, where $\boldsymbol{x_n} \in \mathcal{X}$, which is usually $\mathbb{R}^D$ and $y_n \in \mathcal{Y}$ (usually $\mathbb{R}$ in this case); $\{(\boldsymbol{x_n}, y_n)\}_{n=1}^N$. The goal is to find a function that maps from the input space to the output space **well**; $f : \mathcal{X} \to \mathcal{Y}$.

We need to first find candidates for functions that can perform the predictions. Functions need to be parameterised, such that some numbers $\boldsymbol{\theta}$ map to a function. From here, we need to pick the 'best' function, thus requiring us to define what good and bad functions are. A good function has the property $f(\boldsymbol{x_i}, \boldsymbol{\theta^*}) \approx y_i$; the output of the function closely matches the outputs of the training points. This can be defined with a **loss function**, for example;

$$L(\boldsymbol{\theta}) = \sum_{i=1}^{N} (y_i - f(\boldsymbol{x_i}, \boldsymbol{\theta}))^2$$

Therefore, a good function is chosen by minimising the loss; $\boldsymbol{\theta^*} = \operatorname{argmin}_{\boldsymbol{\theta}} L(\boldsymbol{\theta})$.

Lecture 1.2 - Scalar Differentiation

We can plot the loss against the parameters for a function. This raises two questions; how to change the parameter to make the loss smaller and how we know if we can't get a better loss. The derivative is defined as the limit of the difference quotient (as usual);

$$f'(x) = \frac{\mathrm{d}f}{\mathrm{d}x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

Several examples of this are as follows;

$f(x)$	$f'(x)$
x^n	nx^{n-1}
$\sin(x)$	$\cos(x)$
$\tanh(x)$	$1 - \tanh^2(x)$
$e^x = \exp(x)$	e^x
$\log(x)$	$\frac{1}{x}$

There are also the following rules which combine the basic functions;

- sum rule — describes the derivative of sum of two functions

$$(f(x) + g(x))' = f'(x) + g'(x)$$

- product rule — similarly, for multiplication

$$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$$

- chain rule describes how to differentiate functions that are composed

$$(g \circ f)'(x) = (g(f(x)))' = g'(f(x))f'(x)$$

- quotient rule describes division, special case of the product rule

$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{(g(x))^2}$$

This tells us how to change the input in the function; the gradient tells us how much the output changes based on an increase in the input. We first compute the derivative function at a point to find the point's gradient. If the gradient is negative, this tells us the function will decrease if we increase the input (hence increase to minimise), and vice versa; decrease for positive gradients - this is the idea behind gradient descent.

Similarly, we know we are done (at a minimum for the loss function) when there is nothing that can be done to lower the output, hence the gradient must be 0. However, this isn't sufficient to tell us that we have reached a minimum, as a maximum also has a gradient of zero. A minimum has a decreasing function followed by an increasing function; hence the gradient of the gradient (second derivative) is positive.

However, this only gives us a local minima. We should be concerned with getting stuck in a local minima (rather than a global minima) when dealing with non-convex functions. Working through a simple example, with a linear regression problem (aiming to find an optimal a) - the final step takes the second derivative to verify we have a minimum;

$$\begin{aligned}
f(x) &= a \cdot x \\
L(a) &= \sum_{n=1}^{N} (f(x_n) - y_n)^2 \\
\frac{\mathrm{d}L}{\mathrm{d}a} &= \sum_{n=1}^{N} 2(ax_n - y_n)x_n \\
&= \sum_{n=1}^{N} 2ax_n^2 - 2x_ny_n \\
&= 0 \\
2a\sum_n x_n^2 &= \sum_n 2x_ny_n \\
a &= \frac{\sum_n 2x_ny_n}{\sum_n x_n^2} \\
\frac{\mathrm{d}^2L}{\mathrm{d}a^2} &= \sum_{n=1}^{N} 2x_n^2 \\
&\geq 0
\end{aligned}$$

Lecture 1.3 - Scalar-by-vector Differentiation

The previous example is too simple for real applications - we will need to differentiate by more parameters (vectors). Consider the following example, a polynomial, which has 4 vectors parametising it - each vector now corresponds to a single cubic polynomial;

$$f(x) = \theta_3x^3 + \theta_2x^2 + \theta_1x + \theta_0$$

$$
\begin{aligned}
&= \boldsymbol{\theta}^\top \boldsymbol{\phi}(x) \\
\boldsymbol{\phi}(x) &= \begin{bmatrix} x^3 & x^2 & x & 1 \end{bmatrix}^\top
\end{aligned}
$$

Our goal still remains to understand how a function changes with our parameter and to characterise what an optimum is for a function of a vector. Both of these change a multi-dimensional problem into many 1-dimensional problems.

We want to change it into a 1-dimensional question; instead of asking about a change to $\boldsymbol{\theta}$, we ask what happens when we move along a particular line / direction. A directional derivative is how much the function changes, when we move in a direction $\boldsymbol{v}$;

$$\nabla_{\boldsymbol{v}} L(\boldsymbol{\theta}) = \lim_{h \to 0} \frac{L(\boldsymbol{\theta} + h\boldsymbol{v}) - L(\boldsymbol{\theta})}{h}$$

The distance that we move away from the starting point ($\boldsymbol{\theta}$) is determined by **both** the norm / scale of the direction vector $\boldsymbol{v}$, as well as h. If we can understand how the function changes based on a change in **any** direction, we can fully characterise differentiation with respect to a vector.

Consider the following example, where we deal with two parameters (also notice the second equality holds as the values in violet are equal and cancel each other out);

$$
\begin{aligned}
\nabla_{\boldsymbol{v}} L(\boldsymbol{\theta}) &= \lim_{h \to 0} \frac{L(\theta_1 + hv_1, \theta_2 + hv_2) - L(\theta_1, \theta_2)}{h} \\
&= \lim_{h \to 0} \underbrace{\frac{L(\theta_1 + hv_1, \theta_2 + hv_2) - L(\theta_1, \theta_2 + hv_2)}{h}}_{\text{only change in first parameter}} + \underbrace{\frac{L(\theta_1, \theta_2 + hv_2) - L(\theta_1, \theta_2)}{h}}_{\text{only change in second parameter}} \\
&= \lim_{h \to 0} \frac{L(\theta_1 + h', \theta_2 + h'\frac{v_2}{v_1}) - L(\theta_1, \theta_2 + h'\frac{v_2}{v_1})}{\frac{h'}{v_1}} + \frac{L(\theta_1, \theta_2 + h'') - L(\theta_1, \theta_2)}{\frac{h''}{v_2}} \\
&= \frac{\partial L}{\partial \theta_1} v_1 + \frac{\partial L}{\partial \theta_2} v_2
\end{aligned}
$$

This means that we can find the gradient in any direction with the partial derivatives, as we chose arbitrary v_1, v_2. With a partial derivative, we change only one coordinate at a time - see the following for a function $f : \mathbb{R}^N \to \mathbb{R}$;

$$
\begin{aligned}
y &= f(\boldsymbol{x}) \\
x &= \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \\
\frac{\partial f}{\partial x_i} &= \lim_{h \to 0} \frac{f(x_1, \dots, x_{i-1}, x_i + h, x_{i+1}, \dots, x_N) - f(\boldsymbol{x})}{h}
\end{aligned}
$$

The Jacobian vector collects all partial derivatives into a row vector;

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \cdots & \frac{\partial f}{\partial x_N} \end{bmatrix} \in \mathbb{R}^{1 \times N}$$

We now want to know which direction to go in, to decrease the value of the function the most. The directional derivative can be written as the inner product;

$$\nabla_{\boldsymbol{v}} f(\boldsymbol{\theta}) = \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{\theta}} \boldsymbol{v} = \left| \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{\theta}} \right| |\boldsymbol{v}| \cos \beta$$

We can maximise this by making $\cos \beta$ as large as possible, since the norms of the two vectors are fixed. If we choose a unit vector $\boldsymbol{v}$, we want the largest value possible, hence $\cos \beta = 1$, so the angle between the vectors should be zero (such that $\beta = 0$). As such, we move in the direction of the Jacobian / gradient vector.

We can reuse the intuition from the 1-D case, where moving in either direction doesn't change your value - the directional derivative should be zero in **all directions** (hence the zero vector, $\mathbf{0}$). Additionally, to verify it's a minimum, we want the second directional derivative to also be positive in **all directions**.

Lecture 1.4 - Vector-by-vector Differentiation

Recall the motivating example of linear regression;

$$L(\boldsymbol{\theta}) = \sum_{n=1}^{N} (y_n - \boldsymbol{\phi}(x_n)^T\boldsymbol{\theta})^2 = ||\boldsymbol{y} - \boldsymbol{\Phi}(X)\boldsymbol{\theta}||^2$$

We could either manually take partial derivatives of L, which would be laborious, or consider it as a composition of a vector-to-vector function (the matrix multiplication) and a vector-to-scalar function (the norm of the vector squared);

$$f(\boldsymbol{g}(\boldsymbol{\theta})) \qquad f : \mathbb{R}^D \to \mathbb{R} \qquad g : \mathbb{R}^E \to \mathbb{R}^D$$

There is a multivariate chain rule, for scalars it is as follows (where f is a function of a and b, both of which are functions of t);

$$\frac{\mathrm{d}f(a(t), b(t))}{\mathrm{d}t} = \frac{\partial f}{\partial a}\frac{\mathrm{d}a}{\mathrm{d}t} + \frac{\partial f}{\partial b}\frac{\mathrm{d}b}{\mathrm{d}t}$$

This can be generalised as follows, for $\boldsymbol{g}(t) \in \mathbb{R}^D$, and both are vectors - allowing us to write it as an inner product;

$$\frac{\mathrm{d}f(\boldsymbol{g}(t))}{\mathrm{d}t} = \sum_{i=1}^{D} \frac{\partial f}{\partial g_i}\frac{\mathrm{d}g_i}{\mathrm{d}t} = \underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{g}}}_{\text{row}} \cdot \underbrace{\frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}t}}_{\text{col}}$$

This only works as we've defined the differentiation of a function with respect to a vector as a row vector - which we will use for the remainder of the course. Similarly, the second part of the sum is the derivative of a column vector, which remains a column vector. Consider the following example, with $f : \mathbb{R}^2 \to \mathbb{R}$ and $\boldsymbol{x} : \mathbb{R} \to \mathbb{R}^2$;

$$\begin{aligned}
f(\boldsymbol{x}) &= f(x_1, x_2) \\
&= x_1^2 + 2x_2 \\
\boldsymbol{x}(t) &= \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} \\
&= \begin{bmatrix} \sin(t) \\ \cos(t) \end{bmatrix} \\
\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}} &\in \mathbb{R}^{1\times 2} \\
\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} &\in \mathbb{R}^2 \\
\frac{\mathrm{d}f}{\mathrm{d}t} &= \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{x}}\frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}t} \\
&= \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} \end{bmatrix} \begin{bmatrix} \frac{\partial x_1}{\partial t} \\ \frac{\partial x_2}{\partial t} \end{bmatrix} \\
&= \begin{bmatrix} 2\sin(t) & 2 \end{bmatrix} \begin{bmatrix} \cos(t) \\ -\sin(t) \end{bmatrix} \\
&= 2\sin(t)(\cos(t) - 1)
\end{aligned}$$

A similar rule applies when differentiating with respect to a vector (note previously we only did it with respect to a scalar). Similarly, this just requires collecting all the partial derivatives, and the same chain rule applies for $\boldsymbol{g}(\boldsymbol{x}) \in \mathbb{R}^D$;

$$\frac{\partial f(\boldsymbol{g}(\boldsymbol{x}))}{\partial x_j} = \sum_{i=1}^{D} \frac{\partial f}{\partial g_i}\frac{\mathrm{d}g_i}{\mathrm{d}x_j}$$

Once collected, this becomes matrix multiplication, which can be written in vector form;

$$\frac{\mathrm{d}f(\boldsymbol{g}(\boldsymbol{x}))}{\mathrm{d}\boldsymbol{x}} = \underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{g}}}_{\text{row}} \cdot \underbrace{\frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}\boldsymbol{x}}}_{\text{mat}}$$

Note that the matrix is the derivative of a column vector ($\boldsymbol{g}$) with respect to an input vector $\boldsymbol{x}$. We instead put the elements in each row (similar to how we did it for a vector by scalar) - the elements of $\boldsymbol{g}$ (i) are along the column, and the dimensions of the derivative (j) are along the row. If f gave a vector as an output, we'd end up with a matrix by matrix multiplication.

Consider the following example, where we have $f : \mathbb{R}^N \to \mathbb{R}^M$, $\boldsymbol{y} = \boldsymbol{f}(\boldsymbol{x}) \in \mathbb{R}^M$, and $\boldsymbol{x} \in \mathbb{R}^N$;

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} f_1(x_1, \dots, x_N) \\ \vdots \\ f_M(x_1, \dots, x_N) \end{bmatrix}$$

The collection of all partial derivatives is called a Jacobian matrix;

$$\begin{bmatrix} \frac{\mathrm{d}y_1}{\mathrm{d}\boldsymbol{x}} \\ \vdots \\ \frac{\mathrm{d}y_M}{\mathrm{d}\boldsymbol{x}} \end{bmatrix} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_N} \\ \vdots & & \vdots \\ \frac{\partial f_M}{\partial x_1} & \dots & \frac{\partial f_M}{\partial x_N} \end{bmatrix} \in \mathbb{R}^{M \times N}$$

In general, a function $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$ has a gradient that is a matrix $\mathbb{R}^{M \times N}$, or has the number of target dimensions (M) $\times$ the number of input dimensions (N);

$$\mathrm{d}\boldsymbol{f}[m, n] = \frac{\partial f_m}{\partial x_n}$$

Recall that for matrix multiplication, the second dimension of the first matrix must match with the first dimension of the second matrix. A function composition is constrained that the output dimension of $\boldsymbol{h}$ must be the same as the input dimension of $\boldsymbol{g}$ to compute $\boldsymbol{g}(\boldsymbol{h}(\boldsymbol{x}))$ when we have $\boldsymbol{f}(\boldsymbol{x}) = (\boldsymbol{g} \circ \boldsymbol{h})(\boldsymbol{x})$. This ensures the shapes of the chain rule will **always** work out. For $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$, $\boldsymbol{g} : \mathbb{R}^L \to \mathbb{R}^M$, and $\boldsymbol{h} : \mathbb{R}^N \to \mathbb{R}^L$, we have the following shapes;

$$\underbrace{\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}}}_{M \times N} = \underbrace{\frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}\boldsymbol{h}}}_{M \times L} \underbrace{\frac{\mathrm{d}\boldsymbol{h}}{\mathrm{d}\boldsymbol{x}}}_{L \times N}$$

Consider the following example, of a matrix multiplication $\boldsymbol{f}(\boldsymbol{x}) = \boldsymbol{A}\boldsymbol{x}$ where we have $\boldsymbol{A} \in \mathbb{R}^{M \times N}$ and $\boldsymbol{x} \in \mathbb{R}^N$;

$$\begin{bmatrix} y_1 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}) \\ \vdots \\ f_M(\boldsymbol{x}) \end{bmatrix} \tag{1}$$

$$= \begin{bmatrix} A_{1,1}x_1 + A_{1,2}x_2 + \dots + A_{1,N}x_N \\ \vdots \\ A_{M,1}x_1 + A_{M,2}x_2 + \dots + A_{M,N}x_N \end{bmatrix} \tag{2}$$

$$f_i(\boldsymbol{x}) = \sum_{k=1}^{N} A_{i,k}x_k \tag{3}$$

$$\frac{\partial f_i}{\partial x_j} = \sum_k A_{i,k}\frac{\partial x_k}{\partial x_j} \tag{4}$$

$$= \sum_k A_{i,k}\delta_{kj} \tag{5}$$

$$= A_{i,j} \tag{6}$$

Note the following steps in particular;

(3) write out each scalar in the vector with index notation by writing the sum explicitly allowing us to take the derivative of an arbitrary element in the output vector with respect to an arbitrary element in the input vector

(4) take differential operator inside by sum rule

(5) matrix value is constant, so we end up taking a partial derivative of x_k by x_j - partial derivative only changes x_j and keeps everything else constant, hence it will be zero when $k \neq j$ and 1 when $k = j$ (indicator function, δ)

(6) end up with $A_{i,j}$ as it's the only case the indicator function is non-zero

Therefore, we get the result that $\frac{\mathrm{d}f}{\mathrm{d}x} = \boldsymbol{A} \in \mathbb{R}^{M \times N}$.

Consider the following, which is similar to the loss function, where $\boldsymbol{x} \in \mathbb{R}^N$, $\boldsymbol{A} \in \mathbb{R}^{M \times N}$, $\boldsymbol{e}, \boldsymbol{y} \in \mathbb{R}^M$;

$$\begin{aligned}
L(\boldsymbol{e}) &= \frac{1}{2}||\boldsymbol{e}||^2 \\
&= \frac{1}{2}\boldsymbol{e}^\top \boldsymbol{e} \\
\boldsymbol{e} &= \boldsymbol{y} - \boldsymbol{A}\boldsymbol{x} \\
\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{x}} &= \frac{\partial L}{\partial \boldsymbol{e}}\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{x}} \\
\frac{\partial L}{\partial e_i} &= \frac{\partial}{\partial e_i}\sum_j \frac{1}{2}e_j^2 \\
&= \sum_j \frac{1}{2}2e_j\frac{\partial e_j}{\partial e_i} \\
&= e_i \\
\frac{\partial L}{\partial \boldsymbol{e}} &= \boldsymbol{e}^\top \\
\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{x}} &= -\boldsymbol{A} \\
\frac{\partial L}{\partial \boldsymbol{x}} &= e^\top(-\boldsymbol{A}) \\
&= -(\boldsymbol{y} - \boldsymbol{A}\boldsymbol{x})^\top \boldsymbol{A}
\end{aligned}$$

Lecture 1.5 - Hessians

We still want to get the second derivative, in order to verify that we have a minima and not a maxima. The chain rule we have in place only works with derivatives of scalars, or column vectors (when differentiating with respect to vectors). We've currently defined the vertical axis to be outputs and the horizontal axis for variables we're differentiating by. This convention no longer holds when we are taking second derivatives, as we need to take the second derivative along a row vector;

$$\nabla_{\boldsymbol{v}} \underbrace{\left[\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{\theta}}\boldsymbol{v}\right]}_{\text{scalar}} = \underbrace{\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left[\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{\theta}}\boldsymbol{v}\right]}_{\text{row vector}} \boldsymbol{v}$$

Solving this in a way that only involves taking derivatives with respect to scalars;

$$\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{\theta}}\boldsymbol{v} = \sum_j \frac{\partial f}{\partial \theta_j}v_j \tag{1}$$

$$\frac{\partial}{\partial \theta_i}\left[\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{\theta}}\boldsymbol{v}\right] = \sum_j \frac{\partial}{\partial \theta_j}\frac{\partial f}{\partial \theta_j}v_j \tag{2}$$

$$= \sum_j \frac{\partial^2 f}{\partial \theta_i \partial \theta_j} v_j \tag{3}$$

$$\nabla_{\boldsymbol{v}} \left[\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{\theta}} \boldsymbol{v} \right] = \boldsymbol{v}^\top \boldsymbol{H} \boldsymbol{v} \tag{4}$$

This involves the following steps;

1. expand out in terms of summation components (partial derivatives multiplied by vector component)
2. take partial derivative of the scalar, using sum rule
3. obtain a row vector where we multiply against the second partial derivatives
4. the matrix $\boldsymbol{H}$ (Hessian) contains all second partial derivatives of the function f

We are at the minimum if $\boldsymbol{H}$ is positive definite ($\forall \boldsymbol{v}\ \boldsymbol{v}^\top \boldsymbol{H} \boldsymbol{v} \geq 0$) - positive eigenvalues.

Lecture 2 - Matrix Derivatives and Backpropagation

Defining a function with respect to a number of parameters, and taking a derivative (by doing many simple problems) can be time consuming.
Consider a loss function;

$$\begin{aligned}
L &= \frac{1}{2} ||\boldsymbol{e}||^2 && \boldsymbol{e} \in \mathbb{R}^N \\
\boldsymbol{e} &= \boldsymbol{y} - \boldsymbol{A}\boldsymbol{c} && \boldsymbol{A} \in \mathbb{R}^{N \times D} \\
\boldsymbol{c} &= \sin \boldsymbol{x} && \boldsymbol{x} \in \mathbb{R}^D \\
\frac{\partial L}{\partial x_j} &= \frac{\partial}{\partial x_j} \frac{1}{2} \sum_{n=1}^N \left(y_n - \sum_i A_{n,i} \sin x_i \right)^2 \\
&= \frac{1}{2} \cdot 2 \cdot \sum_{n=1}^N \left(y_n - \sum_i A_{n,i} \sin x_i \right) \cdot \frac{\partial}{\partial x_j} \left[y_n - \sum_i A_{n,i} \sin x_i \right] \\
&= \sum_{n=1}^N \left(y_n - \sum_i A_{n,i} \sin x_i \right) \sum_i \frac{\partial}{\partial x_j} (A_{n,i} \sin x_i) \\
&= \sum_{n=1}^N \left(y_n - \sum_i A_{n,i} \sin x_i \right) \cdot - \sum_i A_{n,i} \cos x_i \frac{\partial x_i}{\partial x_j} \\
&= \sum_{n=1}^N \left(y_n - \sum_i A_{n,i} \sin x_i \right) \cdot - \sum_i A_{n,i} \cos x_i \delta_{ij} \\
&= \sum_{n=1}^N \left(y_n - \sum_i A_{n,i} \sin x_i \right) \cdot - A_{n,j} \cos x_j \\
\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{x}} &= \underbrace{\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{e}}}_{1 \times N} \cdot \underbrace{\frac{\mathrm{d}\boldsymbol{e}}{\mathrm{d}\boldsymbol{c}}}_{N \times D} \cdot \underbrace{\frac{\mathrm{d}\boldsymbol{c}}{\mathrm{d}\boldsymbol{x}}}_{D \times D} \\
\frac{\partial L}{\partial e_j} &= \frac{\partial}{\partial e_j} \left[\frac{1}{2} \sum_{n=1}^N e_n^2 \right] \\
&= \frac{1}{2} \sum_{n=1}^N \frac{\partial}{\partial e_j} e_n^2 \\
&= \sum_n e_n \delta_{nj}
\end{aligned}$$

$$
\begin{aligned}
&= e_j \\
\frac{\partial e_i}{\partial c_j} &= \frac{\partial}{\partial c_j}\left[y_i - \sum_k A_{i,k}c_k\right] \\
&= -\sum_k A_{i,k}\frac{\partial c_k}{\partial c_j} \\
&= -\sum_k A_{i,k}\delta_{kj} \\
&= -A_{i,j}
\end{aligned}
$$

collect js along row and is down column

$$
\begin{aligned}
\frac{\mathrm{d}\boldsymbol{e}}{\mathrm{d}\boldsymbol{c}} &= -\boldsymbol{A} \\
\frac{\partial c_i}{\partial x_j} &= \frac{\partial}{\partial x_j}\sin x_i \\
&= \cos x_i \frac{\partial x_i}{\partial x_j} \\
&= (\cos x_i)\delta ij && \text{note}
\end{aligned}
$$

Note that previously we only had sums with the indicator function; in this case, we only have $\cos x_i$ along the diagonal, and zeroes elsewhere.

Consider the linear regression problem;

$$
\begin{aligned}
L(\boldsymbol{\theta}) &= \frac{1}{2}||\boldsymbol{y} - \boldsymbol{\Phi}(\boldsymbol{X})\boldsymbol{\theta}||^2 \\
\boldsymbol{\theta} &\in \mathbb{R}^3 \\
\boldsymbol{\Phi}(\boldsymbol{X}) &\in \mathbb{R}^{N\times 3} \\
\boldsymbol{y} &\in \mathbb{R}^N \\
\boldsymbol{\phi}(x) &= \begin{bmatrix} 1 \\ x \\ x^2 \end{bmatrix} \\
\boldsymbol{\Phi}(\boldsymbol{X}) &= \begin{bmatrix} \boldsymbol{\phi}^\top(x_1) \\ \boldsymbol{\phi}^\top(x_2) \\ \vdots \end{bmatrix} \\
f(x) &= \boldsymbol{\phi}(x)^\top\boldsymbol{\theta} \\
&= \theta_0 + \theta_1 x + \theta_2 x^2 \\
L(\boldsymbol{e}) &= \frac{1}{2}||\boldsymbol{e}||^2 \\
\boldsymbol{e} &= \boldsymbol{y} - \boldsymbol{\Phi}(\boldsymbol{X})\boldsymbol{\theta} \\
\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}} &= \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{e}}\cdot\frac{\mathrm{d}\boldsymbol{e}}{\mathrm{d}\boldsymbol{\theta}} \\
\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{e}} &= \boldsymbol{e}^\top \\
\frac{\mathrm{d}\boldsymbol{e}}{\mathrm{d}\boldsymbol{\theta}} &= -\boldsymbol{\Phi}(\boldsymbol{X}) \\
&= (\boldsymbol{y} - \boldsymbol{\Phi}(\boldsymbol{X})\boldsymbol{\theta})^\top \cdot -\boldsymbol{\Phi}(\boldsymbol{X}) \\
&= 0 \\
\boldsymbol{\theta}^\top\boldsymbol{\Phi}(\boldsymbol{X})^\top\boldsymbol{\Phi}(\boldsymbol{X}) &= \boldsymbol{y}^\top\boldsymbol{\Phi}(\boldsymbol{X})
\end{aligned}
$$

$$\boldsymbol{\theta}^\top \underbrace{[\boldsymbol{\Phi}(\boldsymbol{X})^\top \boldsymbol{\Phi}(\boldsymbol{X})][\boldsymbol{\Phi}(\boldsymbol{X})^\top \boldsymbol{\Phi}(\boldsymbol{X})]^{-1}}_{\boldsymbol{I}} = \boldsymbol{y}^\top \boldsymbol{\Phi}(\boldsymbol{X})[\boldsymbol{\Phi}(\boldsymbol{X})^\top \boldsymbol{\Phi}(\boldsymbol{X})]^{-1}$$

The basis function can also be a learnable feature, with $\phi : \mathbb{R}^D \to \mathbb{R}^L$ and σ being an activation function;

$$\begin{aligned}
&\boldsymbol{x} \in \mathbb{R}^D \\
&\boldsymbol{\theta} \in \mathbb{R}^L \\
&\quad f(\boldsymbol{x}) = \boldsymbol{\phi}(\boldsymbol{x})^\top \boldsymbol{\theta} \\
&\quad \boldsymbol{\phi}(x) = \sigma(\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b})
\end{aligned}$$

This can be repeated deeper by replacing the $\boldsymbol{x}$ with $\boldsymbol{\phi'}(\boldsymbol{x})$ (with $\boldsymbol{A'}$ and $\boldsymbol{b'}$ similarly), and so on. The idea is that we have a layer that takes in the output of the previous layer as follows (eventually, this stops with f_0 being x, the input);

$$f_l(f_{l-1}) = \sigma(\boldsymbol{A_l} f_{l-1} + \boldsymbol{b_l})$$

Note for brevity, we combine $\boldsymbol{A_i}$ and $\boldsymbol{b_i}$ as θ_i;

$$\begin{tikzcd}
x \arrow[r] & f_1 \arrow[r] & f_2 \arrow[r] & f_3 \arrow[r, dashed] & f_k \arrow[r] & L \\
& \theta_1 \arrow[u] & \theta_2 \arrow[u] & \theta_3 \arrow[u] & &
\end{tikzcd}$$

If we want to learn in this model, we need to take derivatives with respect to all parameters we've defined ($\boldsymbol{A}$ and $\boldsymbol{b}$ in each layer). However, the former is a matrix; hence we need to take the derivative of a function with respect to a matrix, such as (note that this is no longer matrix multiplication, even if they both look like matrices; will be explained later);

$$\frac{\mathrm{d}}{\mathrm{d}\theta}\boldsymbol{x}^\top \boldsymbol{A}(\theta)\boldsymbol{x} \text{ or } \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{A}}||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}||^2$$

When taking the derivative of a vector with respect to a vector, we had two indices which could be nicely represented in a matrix. But in the case of differentiating with respect to a matrix, we have the indices of the vector, as well as a grid of indices in the matrix.

Functions of matrices are no different to functions of vectors; they're still just functions of many different numbers - hence a function of a matrix $\boldsymbol{A} \in \mathbb{R}^{M \times N}$ remains a multivariate function.

$$\begin{aligned}
f(\boldsymbol{A}) &= ||\boldsymbol{A}\boldsymbol{x} - \boldsymbol{y}||^2 \\
f(A_{1,1}, A_{2,1}, \dots, A_{M,1}, \dots, A_{M,N}) &= \sum_i \left(\sum_{i,j} A_{i,j} x_j - y_i \right)
\end{aligned}$$

The chain rule remains the same; where we compute all the partial derivatives with respect to the elements;

$$\begin{aligned}
&f(\boldsymbol{g}) = ||\boldsymbol{g}||^2 \\
&\qquad \boldsymbol{g}(\boldsymbol{A}) = \boldsymbol{A}\boldsymbol{x} - \boldsymbol{y} \\
&\qquad \frac{\partial f}{\partial A_{i,j}} = \sum_k \frac{\partial f}{\partial g_k}\frac{\partial g_k}{\partial A_{i,j}}
\end{aligned}$$

Another example;

$$f(\boldsymbol{A}) = \boldsymbol{x}^\top \boldsymbol{A}\boldsymbol{x}$$

$$\frac{\partial f}{\partial \theta} = \sum_{j,k} \frac{\partial f}{\partial A_{j,k}} \frac{\partial A_{j,k}}{\partial \theta}$$

The difficulty lies in defining the notation that uses well-defined mathematical operations. The chain rule worked well before as we had a nice multiplication in the form of matrix multiplication; this is no longer the case here (see the note prior). Recall the previous notation;

$$\frac{\mathrm{d}f}{\mathrm{d}\theta} = \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{A}} \frac{\mathrm{d}\boldsymbol{A}}{\mathrm{d}\theta} \text{ and } \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{A}} = \frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{g}} \frac{\mathrm{d}\boldsymbol{g}}{\mathrm{d}\boldsymbol{A}}$$

A derivative of a vector with respect to a matrix is a rank 3 tensor (a vector can be seen as a rank 1 tensor, and a matrix as rank 2). This representation with higher-dimensional tensors generalises well. Recall that a function $\boldsymbol{f} : \mathbb{R}^N \to \mathbb{R}^M$ (M target dimensions and N input dimensions) has the following;

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{x}} \in \mathbb{R}^{M \times N}, \quad \mathrm{d}\boldsymbol{f}[m, n] = \frac{\partial f_m}{\partial x_n}$$

This generalises when the sizes of the inputs are matrices instead, for example $\boldsymbol{f} : \mathbb{R}^{M \times N} \to \mathbb{R}^{P \times Q}$, where the gradient is a tensor;

$$\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{X}} \in \mathbb{R}^{M \times N}, \quad \mathrm{d}\boldsymbol{f}[p, q, m, n] = \frac{\partial f_{p,q}}{\partial X_{m,n}}$$

Applying the chain rule to a particular matrix valued function - take the example of $f \in \mathbb{R}$ being a scalar function of $\boldsymbol{A} \in \mathbb{R}^{(P \times Q) \times (N \times M)}$, which is a function of $\boldsymbol{\theta} \in \mathbb{R}^L$;

$$\begin{aligned}
\underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{\theta}}}_{1 \times L} &= \underbrace{\frac{\mathrm{d}f}{\mathrm{d}\boldsymbol{A}}}_{1 \times (N \times M)} \underbrace{\frac{\mathrm{d}\boldsymbol{A}}{\mathrm{d}\boldsymbol{\theta}}}_{(N \times M) \times L} && (1) \\
&= \frac{\mathrm{d}f}{\mathrm{dvec}\boldsymbol{A}} \cdot \frac{\mathrm{dvec}\boldsymbol{A}}{\mathrm{d}\boldsymbol{\theta}} && (2) \\
\mathrm{vec} \begin{bmatrix} a & b \\ c & d \end{bmatrix} &= \begin{bmatrix} a \\ c \\ b \\ d \end{bmatrix} && (3)
\end{aligned}$$

(1) the issue is that we end up with a tensor $\mathbb{R}^{(N \times M) \times L}$

(2) we're now back to doing differentiating with vectors, leading to matrix multiplication

(3) note that changing a matrix into a vector gives a column vector of the matrix by columns

Automatic differentiation packages keep track of all these axes in these higher order tensors which contain these derivatives and figure out which axes to sum over by looking at the input shape of the function $f : \mathbb{R}^{N \times M} \to \mathbb{R}$, consistent with the matrix valued chain rule. The most unambiguous way to deal with this is to write it all as a vector, leading to matrix multiplication.

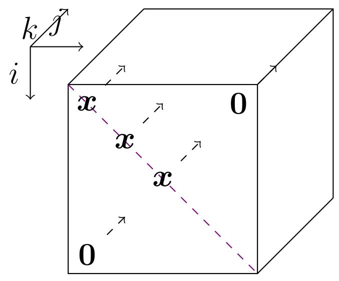
Example of Multivariate Function

The function is $f : \mathbb{R}^{N \times M} \to \mathbb{R}^N$, defined as;

$$\begin{aligned}
\boldsymbol{f} &= \underbrace{\boldsymbol{A}}_{N \times M} \underbrace{\boldsymbol{x}}_{M \times 1} \\
\frac{\mathrm{d}\boldsymbol{f}}{\mathrm{d}\boldsymbol{A}} &\in \mathbb{R}^{N \times (N \times M)} \\
\frac{\partial f_i}{\partial A_{j,k}} &= \frac{\partial}{\partial A_{j,k}} \left(\sum_m A_{i,m} x_m \right)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_m \frac{\partial A_{i,m}}{\partial A_{j,k}} x_m \\
&= \sum_m \delta_{nj}\delta_{mk} x_m \\
&= \delta_{ij} x_k
\end{aligned}
$$

This gives the following three-dimensional tensor, taking the convention of putting the outputs (i) along the column. Note that there are zeroes everywhere, other than the leading diagonal on i, j where the elements of $\boldsymbol{x}$ are.



Backpropagation

Recall that a neural network is essentially a stack of function estimators, such that the output of a previous function becomes the input of the next function. The loss function is typically something with respect to an error producing output (observed value y_i compared to some computed value f_i). As such, L is a function of all the parameters that make up the network; $L(\boldsymbol{A_1}, \boldsymbol{b_1}, \boldsymbol{A_2}, \boldsymbol{b_2}, \dots)$, which can also be written with $\boldsymbol{\theta}$ parameters. Consider the following example of a single-layer network;

$$\begin{tikzcd}
x \arrow[r] & z \arrow[r, "\sigma"] & f \\
& \boldsymbol{A}, \boldsymbol{b} \arrow[u] &
\end{tikzcd}$$

In this example, let σ (activation function) be tanh applied to some linear transformation (which we can denote $\boldsymbol{z}$), with $\boldsymbol{z} \in \mathbb{R}^M, \boldsymbol{x} \in \mathbb{R}^N, \boldsymbol{A} \in \mathbb{R}^{M \times N}, \boldsymbol{b} \in \mathbb{R}^M$. Also note that we have a loss function, L;

$$
\begin{aligned}
\boldsymbol{\theta} &= \{\boldsymbol{A}, \boldsymbol{b}\} \\
L(\boldsymbol{\theta}) &= \frac{1}{2}||\boldsymbol{e}||^2 \\
\boldsymbol{e} &= \boldsymbol{y} - \boldsymbol{f_\theta}(\boldsymbol{x}) \\
\boldsymbol{f} &= \tanh(\underbrace{\boldsymbol{Ax} + \boldsymbol{b}}_{\boldsymbol{z}}) \\
\underbrace{\frac{\partial L}{\partial \boldsymbol{e}}}_{1\times M} &= \boldsymbol{e}^\top \\
\underbrace{\frac{\partial \boldsymbol{e}}{\partial \boldsymbol{f}}}_{M\times M} &= -\boldsymbol{I} \\
\underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{b}}}_{M\times M} &= \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M\times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M\times M} \\
\underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{A}}}_{M\times(M\times N)} &= \underbrace{\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}}}_{M\times M} \underbrace{\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}}}_{M\times(M\times N)}
\end{aligned}
$$

$$\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}} = \operatorname{diag}(1 - \tanh^2(\boldsymbol{z}))$$
$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{b}} = \boldsymbol{I}$$
$$\frac{\partial \boldsymbol{z}}{\partial \boldsymbol{A}} = \begin{bmatrix} \boldsymbol{x}^\top & \cdots & \boldsymbol{0}^\top & \cdots & \boldsymbol{0}^\top \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \boldsymbol{0}^\top & \cdots & \boldsymbol{x}^\top & \cdots & \boldsymbol{0}^\top \\ \vdots & \ddots & \vdots & \ddots & \vdots \\ \boldsymbol{0}^\top & \cdots & \boldsymbol{0}^\top & \cdots & \boldsymbol{x}^\top \end{bmatrix}$$

If we blindly applied the rules we know, we'd be able to solve it with many computations of partial derivatives. The idea behind backpropagation is to find an order of computing partial derivatives that maximally reuses what we've computed. Consider the following;

$$L(\boldsymbol{\theta_1}, \boldsymbol{\theta_2}, \dots, \boldsymbol{\theta_K}) = ||\boldsymbol{y} - \boldsymbol{f_{\theta_K}}(\boldsymbol{f_{\theta_{K-1}}}(\dots \boldsymbol{f_{\theta_2}}(\boldsymbol{f_{\theta_1}}(\boldsymbol{x}))\dots))||^2$$

This has the following relation (anything that hasn't been previously computed is in violet);

$$\frac{\partial L}{\partial \boldsymbol{\theta_K}} = \frac{\partial L}{\partial \boldsymbol{f_K}} \frac{\partial \boldsymbol{f_K}}{\partial \boldsymbol{\theta_K}}$$
$$\frac{\partial L}{\partial \boldsymbol{\theta_{K-1}}} = \frac{\partial L}{\partial \boldsymbol{f_K}} \frac{\partial \boldsymbol{f_K}}{\partial \boldsymbol{f_{K-1}}} \frac{\partial \boldsymbol{f_{K-1}}}{\partial \boldsymbol{\theta_{K-1}}}$$
$$\frac{\partial L}{\partial \boldsymbol{\theta_{K-2}}} = \frac{\partial L}{\partial \boldsymbol{f_K}} \frac{\partial \boldsymbol{f_K}}{\partial \boldsymbol{f_{K-1}}} \frac{\partial \boldsymbol{f_{K-1}}}{\partial \boldsymbol{f_{K-2}}} \frac{\partial \boldsymbol{f_{K-2}}}{\partial \boldsymbol{\theta_{K-2}}}$$
$$\frac{\partial L}{\partial \boldsymbol{\theta_{K-3}}} = \frac{\partial L}{\partial \boldsymbol{f_K}} \frac{\partial \boldsymbol{f_K}}{\partial \boldsymbol{f_{K-1}}} \frac{\partial \boldsymbol{f_{K-1}}}{\partial \boldsymbol{f_{K-2}}} \frac{\partial \boldsymbol{f_{K-2}}}{\partial \boldsymbol{f_{K-3}}} \frac{\partial \boldsymbol{f_{K-3}}}{\partial \boldsymbol{\theta_{K-3}}}$$
$$\frac{\partial L}{\partial \boldsymbol{\theta_i}} = \frac{\partial L}{\partial \boldsymbol{f_K}} \frac{\partial \boldsymbol{f_K}}{\partial \boldsymbol{f_{K-1}}} \cdots \frac{\partial \boldsymbol{f_{i+1}}}{\partial \boldsymbol{f_i}} \frac{\partial \boldsymbol{f_i}}{\partial \boldsymbol{\theta_i}}$$

All the partial gradients that are carried around are vectors, being significantly more computationally efficient than matrices.

Lecture 3 - Probability and Statistics

Overfitting

Recall the curve fitting problem was to find to find a curve that predicts well for unseen inputs, by minimising the loss on the training points. The idea is that it needs to fit the training data well, to fit for unseen data.

$$L(\boldsymbol{\theta}) = \sum_n (f(\boldsymbol{x}_n; \boldsymbol{\theta}) - y_n)^2$$

Consider how many basis functions we should use (considering polynomial);

$$q \in 0, \dots, Q : \boldsymbol{\phi}(x) = \begin{bmatrix} 1 & \cdots & x^q \end{bmatrix}^\top$$

For each order we add, we have a more 'wiggly' function that approaches our training data. However, as we get closer to the training data, our predictions further away from our training points start to become more erratic.

$$f_M(\boldsymbol{x}_n; \boldsymbol{\theta}) = \sum_{m=0}^{M} x^m \theta_m$$

For models $M_1 \leq M_2$, the model M_2 can represent all functions that M_1 can - therefore increasing the order can never worsen the fit on the training data. Closed-form optimisation will find the exact minimum, hence (the training loss will always get smaller when we add basis functions);

$$\min_{\boldsymbol{\theta}} L_{M_2}(\boldsymbol{\theta}) \leq \min_{\boldsymbol{\theta}} L_{M_1}(\boldsymbol{\theta})$$

Typical practice is to split data into a training and a test set. The training set is actually used to find the parameters of a model and the test data shows how well the data predicts on unseen data. We can estimate some loss on the training set, using argmin (which finds the argument that minimises a function);

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L(\boldsymbol{\theta})$$

$$L_{\text{test}} = \frac{1}{N_{\text{test}}}(f(x_n; \boldsymbol{\theta}^*) - y_n)^2$$

The training error should decrease with higher polynomial degrees, whereas the test error should also decrease until a certain point, when it starts to increase (overfitting).

Probability Theory

We need to formalise this as follows;

- Ω set of all possible outcomes

  for a die; $\Omega = \{1, 2, 3, 4, 5, 6\}$
- $E = \{\omega_i \in \Omega\}_{i=1}^I$ event is a set of outcomes

  for an event that states the value of the die is greater than 3; $E = \{4, 5, 6\}$
- $\mathcal{A}$ set of all events (a set of sets)
- X, Y random variables (functions on outcomes to the target set τ, simplifies event structure)

$$X : \Omega \to \tau$$

Consider Ω to be all possible outcomes of a presidential election (can have varying granularity). This could be from the level of what each person voted, and so on. $X(\omega)$ is a simple binary outcome, whether person X is president or not (1 or 0, respectively).

For the scenario of flipping two coins, $\Omega = \{HT, TH, HH, TT\}$. Let $X(\omega)$ be the number of heads;

$$X(\omega) = \begin{cases} 0 & TT \\ 1 & HT, TH \\ 2 & HH \end{cases}$$

A probability measure is a function on sets with the following properties;

- maps from a particular element in the set of all possible events to a real value, larger than 0

$$P : \mathcal{A} \to \mathbb{R}, P(E) \geq 0$$

- probability of the set of outcomes is equal to 1 (something always happens)

$$P(\Omega) = 1$$

- union of events is the sum when there is no intersection

$$P\left(\bigcup_{i=1} \infty E_i\right) = \sum_i P(E_i) \text{ when } E_i \cap E_j = \varnothing\ \forall i, j, i \neq j$$

These axioms describe the reasoning about frequencies of recurring events and the plausibility of beliefs. Our goal is to compute probabilities on the outcomes of random variables (all possible outcomes of Ω

such that $X(\omega)$ is in S - what is the set of all outcomes that would give the random variable values that we care about);

$$P_X(S) = P(X \in S) = P(X^{-1}(S)) = P(\{\omega \in \Omega : X(\omega) \in S\})$$

We can also have multiple random variables as an outcome of the same event. Using the concrete example of the presidential election;

$$X(\omega) = \begin{cases} 1 & \text{person } X \text{ wins} \\ 0 & \text{otherwise} \end{cases}$$
$$Y(\omega) = \begin{cases} 1 & \text{person } X \text{ wins Arkansas} \\ 0 & \text{otherwise} \end{cases}$$

Similarly, we can ask probabilities on the pairs, with the same intuition;

$$P_{X,Y}(S) = P((X, Y) \in S) = P(\{\omega \in \Omega : (X(\omega), Y(\omega)) \in S\})$$

We might want to compute the probability of an event happening for a random variable. For a finite target set, we just need to specify the probability of each of these outcomes ($\frac{1}{6}$ for each die face). Note that for a single event, common notation could be $P(X = x) = p_X(x)$, or even lazier $p(x)$ (with the x implying a random variable X). Similar notation can be used for multiple random variables; $P(X = x, Y = y) = P_{X,Y}(x, y) = P(x, y)$. In the discrete case, probabilities juts need to be specified for each pair.

If we have the probabilities of multiple random variables, we can use this to find the probability of just one of the RVs. This is marginal probability. For example, if we have $p_{X,Y}(x, y)$ and we just want $p(x)$;

$$\begin{aligned} P(X = x) &= P((X, Y) \in \{((x, y) : y \in \tau_Y)\}) \\ &= \sum_{y \in \tau_Y} p_{X,Y}(x, y) \end{aligned}$$

An example of this is from *MacKay*; looking at a certain row of the joint table and normalise the occurrences on the same row to sum to 1. Conditional probability is as follows;

$$\begin{aligned} P(X = x \mid Y = y) &= \frac{P(X = x, Y = y)}{\sum\limits_{x'} P(X = x', Y = y)} \\ &= \frac{P(X = x, Y = y)}{P(Y = y)} \\ P(A \mid B) &= \frac{P(A \cap B)}{P(B)} \end{aligned}$$

The two rules are as follows (and Bayes' rule can be derived);

- sum rule $$p(x) = \sum_y p(x, y)$$
- product rule $$p(x, y) = p(y \mid x)p(x) = p(y)p(x \mid y)$$
- Bayes' rule $$p(y \mid x) = \frac{p(x \mid y)p(y)}{p(x)}$$

Statistical independence states that two events are independent if $P(A \cap B) = P(A)P(B)$ or $P(A \mid B) = P(A)$ - or A doesn't depend on B. Two random variables are independent if $p(X = x, Y = y) = p(x)p(y)$, also written as $X \perp\!\!\!\perp Y$.

However, when it's a continuous random variable, the probability of any particular outcome is zero. This is the case where we have $X : \Omega \to \mathbb{R}$ However, it's more meaningful to ask $P(a \leq X \leq b)$, which can be specified as follows;

$$P(a \leq X \leq b) = \int_a^b f_X(x)\,\mathrm{d}x = \int_a^b p(x)\,\mathrm{d}x$$

And similarly for multiple random variables;

$$P(a \leq X \leq b, c \leq Y \leq d) = \int_a^b \int_c^d f_{X,Y}(x, y)\,\mathrm{d}y\,\mathrm{d}x$$

The rules are similar;

- sum rule $p(x) = \int_y p(x, y)\mathrm{d}y$
- product and Bayes' rule are exactly the same (note we are using the shorthand $f_X(x) = p(x)$ when $x \in \mathbb{R}$)

The general form of a **moment** is as follows;

$$\mathbb{E}_X[g(X)] = \int p(x)g(x)\,\mathrm{d}x$$

For a distribution over a vector, the mean vector works similar to the partial derivatives from before (note $\boldsymbol{a}$ is just any 'direction');

$$\begin{aligned}
p(\boldsymbol{x}) &= p(x_1, x_2, \dots) \\
\mathbb{E}_{\boldsymbol{X}}[\boldsymbol{a}^\top \boldsymbol{x}] &= \int \boldsymbol{a}^\top \boldsymbol{x} p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \\
&= \int \sum_i a_i x_i p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \\
&= \sum_i a_i \int x_i p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \\
&= \sum_i a_i \int x_i p(x_i) p(\{x_j\}_{i \neq j} \mid x_i)\,\mathrm{d}\boldsymbol{x} \\
&= \sum_i a_i \int x_i p(x_i)\,\mathrm{d}x_i \underbrace{\int p(\{x_j\}_{i \neq j} \mid x_i)\,\mathrm{d}\{x\}}_{=1} \\
&= \boldsymbol{a}^\top \bar{\boldsymbol{x}}
\end{aligned}$$

A similar approach can be taken for variance;

$$\begin{aligned}
\mathbb{V}_{\boldsymbol{X}}[\boldsymbol{a}^\top \boldsymbol{x}] &= \int (\boldsymbol{a}^\top \boldsymbol{x} - \boldsymbol{a}^\top \bar{\boldsymbol{x}}) p(\boldsymbol{x})\,\mathrm{d}(\boldsymbol{x}) \\
&= \int \left(\sum_i a_i x_i - a_i - \bar{x}_i \right) \left(\sum_j a_j x_j - a_j - \bar{x}_j \right) p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} \\
&= \sum_i \sum_j a_i a_j \int (x_i - \bar{x}_i)(x_j - \bar{x}_j) p(x_i, x_j)\,\mathrm{d}\{x_i, x_j\} \\
&= \boldsymbol{x}^\top \boldsymbol{\Sigma} \boldsymbol{a}
\end{aligned}$$

The intuition for the direction, for variance, is that we look at the variance in the scalar random variables along a given direction.

Lecture 4 - Cross-validation

We can prevent overfitting by choosing a model (polynomial degree) which isn't too flexible. Generalisation loss represents how well our model performs in future predictions. The setting is that we train the model and deploy it for future predictions, making infinite predictions (limit of $N \to \infty$) in the future. We also incur a penalty for errors (loss function ℓ).

$$L_{\text{test}} = \lim_{N\to\infty} \frac{1}{N} \sum_{n=1}^{N} \ell(f(\boldsymbol{x_n}; \boldsymbol{\theta}^*), y_n)$$

We assume the pairs $(\boldsymbol{x_n}, y_n)$ come from the **same distribution** (both training and future data), hence $\boldsymbol{x_n}, y_n \sim p(\boldsymbol{x}, y)$. By the law of large numbers (generalisation error) - note that this is a moment of $p(\boldsymbol{x}, y)$;

$$L_{\text{test}} = \mathbb{E}_{p(\boldsymbol{x},y)}[\ell(f(\boldsymbol{x}; \boldsymbol{\theta}^*), y)]$$

Our goal is to estimate this loss without knowing the true joint distribution. We can get samples from $p(\boldsymbol{x}, y)$, and construct an estimator for L_{test} from these samples;

$$\hat{L}_{\text{test}} = \frac{1}{N_{\text{test}}} \sum_{n=1}^{N} \ell(f(\boldsymbol{x_n}; \boldsymbol{\theta}^*), y_n)$$

We are writing this without knowing what the density function is, just saying that it exists. As the number of testing points goes to infinity, the estimate approaches the true test loss.

Good ML practice splits data into a training set and a test set, both of which are iid samples from this distribution. Only on the training set do we **minimise** the loss. And loss is **measured** on the test set. As long as **no decisions** are based on the test set, $\hat{L}_{\text{test}}$ will be an unbiased estimate of the true average loss. This loss tells us what the error would've been on particular model. Unbiased means;

$$\mathbb{E}_{p(\boldsymbol{x_1},\boldsymbol{x_2},\ldots,y_1,y_2,\ldots)}[\hat{L}_{\text{test}}(\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, y_1, y_2, \ldots)] = L_{\text{test}}$$

We can analyse the variance of this estimator. As the test set becomes larger, the variance decreases.

$$\mathbb{V}_{\Pi_n p(\boldsymbol{x_n},y_n)}\left[\frac{1}{N}\sum_{n=1}^{N} \ell(f(\boldsymbol{x_n}; \boldsymbol{\theta}^*), y_n)\right] = \frac{1}{N}\mathbb{V}_{p(\boldsymbol{x},\boldsymbol{y})}[\ell(f(\boldsymbol{x_n}; \boldsymbol{\theta}^*), y)]$$

If we have a small variance, the error of the estimator is smaller. Chebyshev's inequality states that the probability of a variable being far away (more than k times the standard deviation) from the mean (the true value we want);

$$P(|X - \bar{X}| \geq k\sigma) \geq \frac{1}{k^2}$$

We want to evaluation a collection of models (different orders of polynomials) on the test set and then pick the model with the lowest test loss. However, we may want to ask whether this is still an unbiased estimate of the test loss. Note that $\boldsymbol{\theta}^*$ is now a function of the test set, therefore the estimator is no longer unbiased - therefore we cannot use the test set to select the model if we want an accurate estimate of the test loss;

$$\mathbb{E}_{\Pi_n p(\boldsymbol{x_n},y_n)}\left[\frac{1}{N}\sum_{n=1}^{N} \ell(f(\boldsymbol{x_n}; \boldsymbol{\theta}^*(\{\boldsymbol{x_i}, y_i\}_{i=1}^{N})), y_n)\right] \neq \mathbb{E}_{p(\boldsymbol{x},\boldsymbol{y})}[\ell(f(\boldsymbol{x_n}; \boldsymbol{\theta}^*), y)]$$

All labelled data is split into training data and test data. The training data is split further to a training set (to find the parameters of the model) and a validation set (to select the model). Once we've select the model based on the validation set, we can use the true test set (kept completely separate) to find an unbiased estimator. However, with a small validation set there would be large variance (possibly choosing wrong model), whereas a large validation set would lead to smaller training sets, hence wrong parameters could be chosen.

Cross-validation splits the data into training and validation sets in multiple ways, and computes the validation performance for each split. The average of these values is the cross-validation loss (no longer unbiased, however small).

1. split data in K different ways
2. for each model;
   (a) for each split
      i. find parameters of model
      ii. compute loss on validation set
   (b) calculate average validation loss for all splits
3. pick model with lowest cross-validation loss

Lecture 5 - Gradient Descent

An optimisation is can be formulated as an objective function $L : \mathbb{R}^D \to \mathbb{R}$ (parameter space to scalar) which roughly tells us how well the model performs on the training data, with an unconstrained minimisation (search parameter in the full space);

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, L(\boldsymbol{\theta})$$

Gradient-based optimisation is a class of methods which perform the following steps;

- pick starting point $\boldsymbol{\theta_0}$
- iterative update the parameters to give a sequence $\boldsymbol{\theta_1}, \dots, \boldsymbol{\theta_T}$
- choose the update by computing the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t})$
- repeat until we reach a stopping criterion (such as computation budget or gradient size)

The gradient descent algorithm is as follows, with a starting point $\boldsymbol{\theta_0}$ and a sequence of step sizes γ_t;

1. set $\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t})$
2. repeat the first step until the stopping criterion is met

Consider the following model (note the use of matrices), our goal is to find $\boldsymbol{\theta} \in \mathbb{R}^{D\times 1}$, such that $\boldsymbol{y} \approx \boldsymbol{X\theta}$;

$$\begin{aligned}
\mathcal{D} &= \{\boldsymbol{X}, \boldsymbol{y}\} && \text{dataset} \\
\boldsymbol{X} &= \begin{bmatrix} \boldsymbol{x_1} & \cdots & \boldsymbol{x_N} \end{bmatrix}^\top && \in \mathbb{R}^{N\times D} \\
\boldsymbol{y} &= \begin{bmatrix} y_1 & \cdots & y_N \end{bmatrix}^\top && \in \mathbb{R}^{N\times 1} \\
\boldsymbol{X}^\top &= \begin{bmatrix} \uparrow & \uparrow & & \uparrow \\ \boldsymbol{x_1} & \boldsymbol{x_2} & \cdots & \boldsymbol{x_N} \\ \downarrow & \downarrow & & \downarrow \\ 1 & 1 & & 1 \end{bmatrix}
\end{aligned}$$

The model and loss are as follows (assume a mean of zero and some variance) and the loss is the ℓ_2 norm. We are taking the gradient as the transpose of the derivative of L with respect to $\boldsymbol{\theta}$ (a column vector);

$$\begin{aligned}
f(\boldsymbol{x}, \boldsymbol{\theta}) &= \boldsymbol{x}^\top \boldsymbol{\theta} \\
y &= f(\boldsymbol{x}, \boldsymbol{\theta}) + \epsilon && \epsilon \sim \mathcal{N}(0, \sigma^2) \\
L(\boldsymbol{\theta}) &= \frac{1}{2\sigma^2} \sum_n (f(\boldsymbol{x_n}, \boldsymbol{\theta}) - y_n)^2
\end{aligned}$$

$$\begin{aligned}
&= \frac{1}{2\sigma^2}||\boldsymbol{y} - \boldsymbol{X\theta}||_2^2 \\
\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) &= \frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}}^\top \\
&= \frac{1}{\sigma^2}\boldsymbol{X}^\top(\boldsymbol{X\theta} - \boldsymbol{y}) && \text{setting} = 0\text{, to get pseudo-inverse} \Rightarrow \\
\frac{1}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{X\theta^*} &= \frac{1}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{y} && \Rightarrow \\
\boldsymbol{\theta^*} &= (\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y}
\end{aligned}$$

The gradient descent for linear regression can be written as follows, assuming constant step-sizes ($\gamma_t = \gamma$);

1. define the starting point $\boldsymbol{\theta_0}$
2. update $\boldsymbol{\theta_{t+1}}$

$$\begin{aligned}
\boldsymbol{\theta_{t+1}} &= \boldsymbol{\theta_t} - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t}) \\
&= \boldsymbol{\theta_t} - \gamma\frac{1}{\sigma^2}\boldsymbol{X}^\top(\boldsymbol{X\theta_t} - \boldsymbol{y}) \\
&= (\boldsymbol{I} - \frac{\gamma}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{\theta_t} + \frac{\gamma}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{y}
\end{aligned}$$

3. repeat until stopping criterion

We will study the gradient descent update equation (final line). This forms a geometric series ($x_{t+1} = ax_t + b$), which can be written as $\alpha^t x_0 + \beta$ (which applies to vectors as well as scalars).

$$\begin{aligned}
\boldsymbol{\theta_{t+1}} + \boldsymbol{\beta} &= \boldsymbol{A}(\boldsymbol{\theta_t} + \boldsymbol{\beta}) && \Rightarrow \\
\boldsymbol{\theta_{t+1}} &= \boldsymbol{A\theta_t} + (\boldsymbol{A} - \boldsymbol{I})\boldsymbol{\beta} && \Rightarrow \\
\boldsymbol{A} &= (\boldsymbol{I} - \frac{\gamma}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{X}) \\
(\boldsymbol{A} - \boldsymbol{I})\boldsymbol{\beta} &= \frac{\gamma}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{y} && \Rightarrow \\
-\frac{\gamma}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{X\beta} &= \frac{\gamma}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{y} && \Rightarrow \\
\boldsymbol{\beta} &= -(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\boldsymbol{y} \\
&= -\boldsymbol{\theta^*}
\end{aligned}$$

This iterative update gives the following result;

$$\boldsymbol{\theta_t} = (\boldsymbol{I} - \frac{\gamma}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{X})^t(\boldsymbol{\theta_0} - \boldsymbol{\theta^*}) + \boldsymbol{\theta^*}$$

Note that this tells us the gradient descent converges (as t gets larger) $\boldsymbol{\theta_t} \to \boldsymbol{\theta^*}$ if;

$$(\boldsymbol{I} - \frac{\gamma}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{X})^t(\boldsymbol{\theta_0} - \boldsymbol{\theta^*}) \to \boldsymbol{0}$$

In an $\mathbb{R}^D$ space, the space can be constructed by a span of basis vectors. For example, $\mathbb{R}^2 = \text{span}(\{\boldsymbol{e_1}, \boldsymbol{e_2}\})$ and $\boldsymbol{e_i} \perp \boldsymbol{e_j}, ||\boldsymbol{e_i}|| = 1$. Any vector in this space can be written as a linear combination of the basis vectors. For the standard basis, assume 2-dimensions, $\boldsymbol{x} = (x_1, x_2)^\top$, then $\boldsymbol{x}^\top\boldsymbol{e_i} = x_i$.

A left multiply of $\boldsymbol{Q}^{-1}$ on $\boldsymbol{x}$ is known as a change of basis from $\{\boldsymbol{e_1}, \boldsymbol{e_2}\}$ to $\{\boldsymbol{q_1}, \boldsymbol{q_2}\}$;

$$\begin{aligned}
\boldsymbol{Q} &= \begin{bmatrix} q_{1,1} & q_{1,2} \\ q_{2,1} & q_{2,2} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{q_1} & \boldsymbol{q_2} \end{bmatrix}
\end{aligned}$$

$$\begin{aligned}
\boldsymbol{Q}^{-1} &= \begin{bmatrix} q_{1,1} & q_{2,1} \\ q_{1,2} & q_{2,2} \end{bmatrix} \\
&= \begin{bmatrix} \boldsymbol{q_1}^\top \\ \boldsymbol{q_2}^\top \end{bmatrix} \\
\boldsymbol{z} &= \boldsymbol{Q}^{-1}\boldsymbol{x} \\
z_1 &= \boldsymbol{q_1}^\top \boldsymbol{x} \\
z_2 &= \boldsymbol{q_2}^\top \boldsymbol{x}
\end{aligned}$$

If we were to stretch this vector in the $\boldsymbol{Q}$ basis;

$$\begin{aligned}
\boldsymbol{z'} &= \boldsymbol{\Lambda}\boldsymbol{Q}^{-1}\boldsymbol{x} \\
z_1' &= \lambda_1 \boldsymbol{q_1}^\top \boldsymbol{x} \\
z_2' &= \lambda_2 \boldsymbol{q_2}^\top \boldsymbol{x} \\
\boldsymbol{\Lambda} &= \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix}
\end{aligned}$$

To return to the canonical basis, after stretching by $\boldsymbol{\Lambda}$ in the $\boldsymbol{Q}$ basis, we have $\boldsymbol{x'} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}\boldsymbol{x}$. Consider the case where we have $\boldsymbol{x'} = \boldsymbol{A}\boldsymbol{x}$, for some $\boldsymbol{A} \in \mathbb{R}^{D\times D}$. We want to find $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$. The goal is to find scalar λ (eigenvalue) and vector (eigenvector) $\boldsymbol{q}$ such that $\boldsymbol{A}\boldsymbol{q} = \lambda\boldsymbol{q}$, if solutions exist for this, we have $\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1}$ where $\boldsymbol{\Lambda} = \text{diag}(\lambda_1, \dots, \lambda_D)$. If $\boldsymbol{A}$ is symmetric, there will be D pairs of solutions of eigenvectors and eigenvalues, such that the column vectors ($\boldsymbol{Q} = (\boldsymbol{q_1}, \dots, \boldsymbol{q_D})$) form an orthonormal basis (such that $\boldsymbol{Q}^{-1} = \boldsymbol{Q}^\top$) and $\lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_D$. Additionally, if $\boldsymbol{A}$ is positive semi-definite, then $\lambda_D \geq 0$. To find this, we first assume $\boldsymbol{q} \neq \boldsymbol{0}$. By $\boldsymbol{A}\boldsymbol{q} = \lambda\boldsymbol{q}$, we can get $(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{q} = \boldsymbol{0}$. With the assumption, we find λ such that $\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0$ - once we have this, it can be plugged in and solved to obtain $\boldsymbol{q}$.

The determinant tells us how the volume is scaled by linear transformation. If the determinant is zero ($\det(\boldsymbol{A} - \lambda\boldsymbol{I}) = 0$), then some subspace in $\mathbb{R}^D$ is collapsed into a line $\{\boldsymbol{0}\}$, implying that there is some $\boldsymbol{q} \neq 0$ such that $(\boldsymbol{A} - \lambda\boldsymbol{I})\boldsymbol{q} = \boldsymbol{0}$. If λ is an eigenvalue of $\boldsymbol{A}$, then λ^t is an eigenvalue of $\boldsymbol{A}^t$. Additionally, $\lambda + \alpha$ is an eigenvalue of $\boldsymbol{A} + \alpha\boldsymbol{I}$;

$$\boldsymbol{A} + \alpha\boldsymbol{I} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^{-1} + \boldsymbol{Q}\alpha\boldsymbol{I}\boldsymbol{Q}^{-1} = \boldsymbol{Q}(\boldsymbol{\Lambda} + \alpha\boldsymbol{I})\boldsymbol{Q}^{-1} \Rightarrow \boldsymbol{\Lambda} + \alpha\boldsymbol{I} = \text{diag}(\lambda_1 + \alpha, \dots, \lambda_D + \alpha)$$

By combining these, if λ is an eigenvalue of $\boldsymbol{A}$, then it follows that $(\lambda+\alpha)^t$ is an eigenvalue of $(\boldsymbol{A}+\alpha\boldsymbol{I})^t$. Note that the corresponding eigenvector in the pair doesn't change.

The Rayleigh quotient is bounded by the eigenvalues of $\boldsymbol{A}$ (assuming symmetric);

$$\lambda_{\min}(\boldsymbol{A}) \leq \underbrace{R(\boldsymbol{A}, \boldsymbol{x}) = \frac{\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x}}{||\boldsymbol{x}||_2^2} = \frac{\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x}}{\boldsymbol{x}^\top\boldsymbol{x}}}_{\text{Rayleigh quotient}} \leq \lambda_{\max}(\boldsymbol{A}) \Rightarrow \lambda_{\min}(\boldsymbol{A})||\boldsymbol{x}||_2^2 \leq \boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x} \leq \lambda_{\max}(\boldsymbol{A})||\boldsymbol{x}||_2^2$$

This result comes from the following;

$$\begin{aligned}
\boldsymbol{x}^\top\boldsymbol{A}\boldsymbol{x} &= \boldsymbol{x}^\top\boldsymbol{Q}\boldsymbol{\Lambda}\underbrace{\boldsymbol{Q}^\top\boldsymbol{x}}_{\boldsymbol{z}} \\
&= \boldsymbol{z}^\top\boldsymbol{\Lambda}\boldsymbol{z} \\
\boldsymbol{x}^\top\boldsymbol{x} &= \boldsymbol{x}^\top\underbrace{\boldsymbol{Q}\boldsymbol{Q}^\top}_{\boldsymbol{I}}\boldsymbol{x} \\
&= \boldsymbol{z}^\top\boldsymbol{z} \\
&= ||\boldsymbol{z}||_2^2 \\
&= \sum_{d=1}^{D} z_D^2
\end{aligned}$$

$$
\begin{aligned}
\boldsymbol{z}^\top \boldsymbol{\Lambda} \boldsymbol{z} &= \sum_{d=1}^{D} \lambda_d z_d^2 \\
R(\boldsymbol{A}, \boldsymbol{x}) &= \frac{\boldsymbol{z}^\top \boldsymbol{\Lambda} \boldsymbol{z}}{||\boldsymbol{z}||_2^2} \\
\boldsymbol{z} &= \begin{bmatrix} z_1 \\ \vdots \\ z_D \end{bmatrix} \\
&= \sum_{d=1}^{D} \frac{z_d^2}{||\boldsymbol{z}||_2^2} \lambda_d && \text{weighted sum}
\end{aligned}
$$

Recall the gradient descent with constant step size. The ℓ_2 distance between $\boldsymbol{\theta_t}$ and $\boldsymbol{\theta^*}$;

$$
\begin{aligned}
\boldsymbol{\theta_t} &= (\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X})^t (\boldsymbol{\theta_0} - \boldsymbol{\theta^*}) + \boldsymbol{\theta^*} \\
||\boldsymbol{\theta_t} - \boldsymbol{\theta^*}||_2^2 &= ||(\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X})^t (\boldsymbol{\theta_0} - \boldsymbol{\theta^*})||_2^2 \\
&= |(\boldsymbol{\theta_0} - \boldsymbol{\theta^*})^\top \underbrace{(\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X})^{2t}}_{\boldsymbol{A}} \underbrace{(\boldsymbol{\theta_0} - \boldsymbol{\theta^*})}_{\boldsymbol{x}}|
\end{aligned}
$$

Note that this is in the form of the Rayleigh quotient, hence we can bound the distance;

$$
\lambda_{\text{min}}((\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X})^2)^t ||\boldsymbol{\theta_0} - \boldsymbol{\theta^*}||_2^2 \leq ||\boldsymbol{\theta_t} - \boldsymbol{\theta^*}||_2^2 \leq \lambda_{\text{max}}((\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X})^2)^t ||\boldsymbol{\theta_0} - \boldsymbol{\theta^*}||_2^2
$$

If $\lambda_{\text{max}}^t \to 0$ as $t \to \infty$, then we have convergence (as it's an upper bound) as it's a distance between the current solution and the optimal solution. For the rules, we will use the following definitions;

$$
\begin{aligned}
\lambda_{\text{min}} &= \lambda_{\text{min}}((\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X})^2) && \geq 0 \\
\lambda_{\text{max}} &= \lambda_{\text{max}}((\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X})^2)
\end{aligned}
$$

Therefore we have the following rules;

1. $\lambda_{\text{max}} < 1$ — always converge
2. $\lambda_{\text{min}} \geq 1$ — always diverge
3. $\lambda_{\text{min}} < 1$ but $\lambda_{\text{max}} \geq 1$ — convergence depends on $\boldsymbol{\theta_0}$

To derive the eigenvalues;

- if λ is an eigenvalue of $\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X}$ then λ^2 is an eigenvalue of $(\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X})^2$
- if λ is an eigenvalue of $\boldsymbol{X}^\top \boldsymbol{X}$ then $1 - \frac{\gamma\lambda}{\sigma^2}$ is an eigenvalue of $\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X}$

This gives us the following result (when combined) - note that $\boldsymbol{X}^\top \boldsymbol{X}$ is positive semi-definite, hence $\lambda \geq 0$;

$$
\boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{q} = \lambda \boldsymbol{q} \Leftrightarrow (\boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X})^2 = (1 - \frac{\gamma\lambda}{\sigma^2})^2 \boldsymbol{q}
$$

To ensure convergence, we want $\lambda_{\text{max}} < 1$, hence we have the constraint on the step size;

$$
\gamma < \frac{2\sigma^2}{\lambda_{\text{max}}(\boldsymbol{X}^\top \boldsymbol{X})}
$$

Generally, larger step-sizes shouldn't be used as we are unlikely to be lucky (and end up in a divergent case).

1. choose a step-size γ that's larger than the constraint
2. **randomly** initialise the parameter

$$\boldsymbol{\theta_0} = \boldsymbol{\theta^*} + \sum_{d=1}^{D} \alpha_d \boldsymbol{q_d}$$

3. iterative update

$$\boldsymbol{\theta_t} - \boldsymbol{\theta^*} = \sum_{d=1}^{D} \alpha_d \underbrace{(1 - \frac{\gamma\lambda_d}{\sigma^2})^t}_{k} \boldsymbol{q_d}$$

   If $\gamma < \frac{2\sigma^2}{\lambda_d}$ for a direction $\boldsymbol{q_d}$, then $k\boldsymbol{q_d} \to \boldsymbol{0}$. Otherwise, $k\boldsymbol{q_d}$ diverges. Gradient descent will diverge unless $\alpha_d = 0$ for all the diverging directions.

Whether the choice of γ is robust to the initialisation of $\boldsymbol{\theta_0}$ depends on the condition number of $\boldsymbol{X}^\top\boldsymbol{X}$.

In general, the loss function isn't quadratic nor is it convex. However, when we approach the optimum, we can take a local quadratic approximation (Taylor expansion of order 2).

Lecture 6 - More on Gradient Descent

We need to check whether the ℓ_2-norm $||\boldsymbol{\theta_t} - \boldsymbol{\theta^*}||_2^2 \to 0$ when we have $t \to \infty$. $\boldsymbol{\theta_t} - \boldsymbol{\theta^*} = \boldsymbol{A}^t(\boldsymbol{\theta_t} - \boldsymbol{\theta^*})$ where $\boldsymbol{A} = (\boldsymbol{I} - \frac{\gamma}{\sigma^2}\boldsymbol{X}^\top\boldsymbol{X})$. The convergence of gradient descent depends on $\lambda_{\max}(\boldsymbol{A}^2)$.

The robustness of the choice of step size depends on the condition number $\kappa(\boldsymbol{X}^\top\boldsymbol{X})$, with $\kappa \approx 1$ being well conditioned and $\kappa >> 1$ being ill conditioned (loss function is stretched across some direction). Possible ways of addressing this are as follows;

- **gradient descent with pre-conditioning**

  The condition number (mentioned before) is based on the data, and we can't generally control that. The idea of gradient descent with pre-conditioning is as follows, where we apply a linear transformation on the gradient vector;

  1. define $\Delta\boldsymbol{\theta_t} = \boldsymbol{0}$
  2. select a pre-conditioner $\boldsymbol{P_t}$
  3. set $\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \gamma_t \boldsymbol{P_t}^{-1}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t})$
  4. set $t \leftarrow t + 1$
  5. repeat 2 - 4 until stopping criterion

  Consider the following example, for linear regression (assuming constant step sizes γ and fixed pre-conditioner $\boldsymbol{P}$);

  1. set $\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \gamma\frac{1}{\sigma^2}\boldsymbol{P}^{-1}\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{\theta_t} - \boldsymbol{y})$
  2. set $t \leftarrow t + 1$
  3. repeat until stopping criterion

  Deriving the geometric sequence form, we see we need to look at the eigenvalues (with the same rules as before) of $(\boldsymbol{I} - \gamma\frac{1}{\sigma^2}\boldsymbol{P}^{-1}\boldsymbol{X}^\top\boldsymbol{X})^2$;

$$\begin{aligned}
\boldsymbol{\theta_{t+1}} &= \boldsymbol{\theta_t} - \gamma\frac{1}{\sigma^2}\boldsymbol{P}^{-1}\boldsymbol{X}^\top(\boldsymbol{X}\boldsymbol{\theta_t} - \boldsymbol{y}) \\
&= (\boldsymbol{I} - \gamma\frac{1}{\sigma^2}\boldsymbol{P}^{-1}\boldsymbol{X}^\top\boldsymbol{X})\boldsymbol{\theta_t} + \gamma\frac{1}{\sigma^2}\boldsymbol{P}^{-1}\boldsymbol{X}^\top\boldsymbol{y} && \Rightarrow \\
\boldsymbol{\theta_t} &= (\boldsymbol{I} - \gamma\frac{1}{\sigma^2}\boldsymbol{P}^{-1}\boldsymbol{X}^\top\boldsymbol{X})^t(\boldsymbol{\theta_0} - \boldsymbol{\theta^*}) + \boldsymbol{\theta^*}
\end{aligned}$$

As such, to ensure convergence at any initialisation, γ can be chosen as;

$$\gamma < \frac{2\sigma^2}{\lambda_{\max}(\boldsymbol{P}^{-1}\boldsymbol{X}^\top\boldsymbol{X})}$$

Similarly, our robustness depends on the condition number; $\kappa(\boldsymbol{P}^{-1}\boldsymbol{X}^\top\boldsymbol{X})$. As we want a well-conditioned optimisation, we can choose the pre-conditioner to be proportional; $\boldsymbol{P} \propto \boldsymbol{X}^\top\boldsymbol{X}$ - in practice we want $\kappa \approx 1$ with $\boldsymbol{P}^{-1}$ being easy to compute (expensive to invert). To construct $\boldsymbol{P_t}$;

- use low-rank / diagonal matrices
- have prior knowledge on $\nabla^2 L(\boldsymbol{\theta_t})$ (useful), for example in linear regression $\nabla^2 L(\boldsymbol{\theta_t}) \propto \boldsymbol{X}^\top\boldsymbol{X}$
- $\nabla^2 L(\boldsymbol{\theta_t})$ may not be constant (or even available) - approximate with statistics of gradients along the update trajectory; used by many adaptive learning rate methods

- **gradient descent with momentum**

  The intuition is to use some historical update information to update the current run. The momentum term is the difference between the current and next parameters. Note that α can depend on t, but it is typically fixed.

  1. define $\Delta\boldsymbol{\theta_t} = \boldsymbol{0}$ and momentum step-size α
  2. set $\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t}) + \alpha\Delta\boldsymbol{\theta_t}$
  3. set $\Delta\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_{t+1}} - \boldsymbol{\theta_t} = \alpha\Delta\boldsymbol{\theta_t} - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t})$
  4. set $t \leftarrow t + 1$
  5. repeat 2 - 4 until stopping criterion

  The key idea is to make the current iteration's updates closer to the previous iteration. This helps to speed up the iterations in flatter regions (where $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t})$ has a small magnitude). It also helps to alleviate oscillations (where $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t})$ points in a different direction). For the latter, consider the 1-D case; in step t we move to the right, but in $t + 1$, we move to the left (based on our gradient at that point) - the updates cancel slightly to give us an update of a smaller magnitude. Finally, it also helps to smooth out gradients if it's inexact.

- **line search**

  The linear search algorithm automatically adapts γ_t to ensure improvement;

  1. define a starting point $\boldsymbol{\theta_0}$ and set $t \leftarrow 0$
  2. compute the gradient $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t})$
  3. search $\gamma_t \in (\gamma_{\min}, \gamma_{\max})$ to minimise $L(\boldsymbol{\theta_t} - \gamma\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t}))$ with respect to γ
     - **backtracking line search** - shrinking step sizes, let τ be the maximum budget;
       (a) set a decreasing schedule, $\gamma_{\max} = \gamma^{(1)} > \gamma^{(2)} > \cdots > \gamma^{(\tau)} = \gamma_{\min}$
       (b) try sizes until it satisfies an acceptance criterion, for example $L(\boldsymbol{\theta_t}) - L(\boldsymbol{\theta_t} - \gamma^{(k)}\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t}))$ is big enough
       (c) set $\gamma_t \leftarrow \gamma^{(k)}$
     - **other methods** - for example, checking the Wolfe conditions

     The decreasing schedule is generally better as we want to do the large movements quickly (imagine a flat region).
  4. set $\boldsymbol{\theta_{t+1}} = \boldsymbol{\theta_t} - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t})$ and $t \leftarrow t + 1$
  5. repeat 2 - 4 until stopping criterion

  Note that the search space for γ is constrained to ensure we don't go over budget. This is used by nonlinear Conjugate Gradient or BFGS. It converges quickly if $\dim(\boldsymbol{\theta})$ is small.

Stochastic Gradient Descent

All the methods we've discussed require knowing the exact gradient (knowing the loss function and differentiating it). The objective function (LSE) can be written as follows (for regression);

$$\boldsymbol{g_t} = \nabla_{\boldsymbol{\theta_t}} L(\boldsymbol{\theta_t}) = \nabla_{\boldsymbol{\theta_t}} \sum_{n=1}^{N} (f(\boldsymbol{x_n}; \boldsymbol{\theta_t}) - y_n)^2$$

Generally, we can write the loss function as a loss on each individual point;

$$\boldsymbol{g_t} = \nabla_{\boldsymbol{\theta_t}} L(\boldsymbol{\theta_t}) = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta_t}} L_n(\boldsymbol{\theta_t})$$

However, in big data applications, N could be extremely large. As such, running a full batch gradient descent can be very expensive, as we need to compute L_n for each point as well as storing intermediate results of every $\nabla_{\boldsymbol{\theta_t}} L(\boldsymbol{\theta_t})$ - just for a single update step.

In SGD, we compute a stochastic estimator for the sum with a random subset $M < N$ terms - the gradient estimator is now random;

$$\hat{\boldsymbol{g}}_{\boldsymbol{t}} = \frac{N}{M} \sum_{m \in \mathcal{S}} \nabla_{\boldsymbol{\theta_t}} L_m(\boldsymbol{\theta_t})$$

If we continuously select "wrong" points, the gradient can end up pointing in the wrong direction. However, by randomising, we don't continue to keep picking the same point.

In practice, this is done by uniform (each point has the same probability of being chosen) sampling with replacement (same point can be chosen again, not taken out of distribution).

$$P(\mathcal{S} = \{m_1, m_2, \dots, m_M\}) = \frac{1}{N} \cdot \frac{1}{N} \cdot \dots = N^{-M}$$

This estimator is unbiased (first condition for convergence);

$$\mathbb{E}_{\mathcal{S}}[\hat{\boldsymbol{g}}_{\boldsymbol{t}}] = \boldsymbol{g_t} \Leftrightarrow \mathbb{E}_{\mathcal{S}}\left[\frac{N}{M} \sum_{m \in \mathcal{S}} \nabla_{\boldsymbol{\theta_t}} L_m(\boldsymbol{\theta_t})\right] = \sum_{n=1}^{N} \nabla_{\boldsymbol{\theta_t}} L_n(\boldsymbol{\theta_t})$$

We also need to choose the step sizes carefully (Robbins-Monro condition), then SGD will converge;

$$\sum_{t=1}^{\infty} \gamma_t = \infty$$
$$\sum_{t=1}^{\infty} \gamma_t^2 < \infty$$

This is a decreasing step size, but not decreasing too quickly - if it's constant, then the solution will swing.

We need to ensure that M is chosen such that it's small enough to be computed quickly, but not too small. The amount of error we will get depends on the error of the stochastic gradient (inversely scaled);

$$\mathbb{V}[\hat{\boldsymbol{g}}_{\boldsymbol{t}}] = \mathbb{V}_{\mathcal{S}}\left[\frac{N}{M} \sum_{m \in \mathcal{S}} \nabla_{\boldsymbol{\theta_t}} L_m(\boldsymbol{\theta_t})\right] = \frac{N^2}{M} \mathbb{V}_{m \sim \text{Uniform}(1,\dots N)}[\nabla_{\boldsymbol{\theta_t}} L_m(\boldsymbol{\theta_t})]$$

Lecture 7 - Ridge Regression and PCA

Ridge Regression

Overfitting can be avoided by choosing a model with the right complexity, with the validation data (same distribution with the test data, allowing for generalisation). Another technique to avoid overfitting is regularisation. Our model is as follows, with the goal of finding $\boldsymbol{\theta} \in \mathbb{R}^{D \times 1}$, we assume the noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$ has a known σ^2;

$$\begin{aligned}
\mathcal{D} &= \{\boldsymbol{X}, \boldsymbol{y}\} \\
\boldsymbol{X} &= \begin{bmatrix} \boldsymbol{x_1} & \cdots & \boldsymbol{x_N} \end{bmatrix}^\top && \in \mathbb{R}^{N \times D} \\
\boldsymbol{y} &= \begin{bmatrix} y_1 & \cdots & y_N \end{bmatrix}^\top && \in \mathbb{R}^{N \times 1} \\
\boldsymbol{y} &\approx \boldsymbol{X\theta} \\
f(\boldsymbol{x}, \boldsymbol{\theta}) &= \boldsymbol{x}^\top \boldsymbol{\theta} \\
y &= f(\boldsymbol{x}; \boldsymbol{\theta}) + \epsilon \\
L(\boldsymbol{\theta}) &= \frac{1}{2\sigma^2} \sum_n (f(\boldsymbol{x_n}; \boldsymbol{\theta}) - y_n)^2 \\
&= \frac{1}{2\sigma^2} ||\boldsymbol{y} - \boldsymbol{X\theta}||_2^2 && \text{matrix form}
\end{aligned}$$

Ridge regression adds an additional regularisation term to the loss function, controlled by the hyper-parameter λ;

$$L(\boldsymbol{\theta}) = \frac{1}{2\sigma^2} ||\boldsymbol{y} - \boldsymbol{X\theta}||_2^2 + \frac{\lambda}{2} ||\boldsymbol{\theta}||_2^2$$

The optimal solution for $\boldsymbol{\theta}$ is $\boldsymbol{\theta_R^*}$, and can be obtained by setting $\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) = 0$;

$$\begin{aligned}
\boldsymbol{\theta_R^*} &= \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\text{argmin}} \frac{1}{2\sigma^2} ||\boldsymbol{y} - \boldsymbol{X\theta}||_2^2 + \frac{\lambda}{2} ||\boldsymbol{\theta}||_2^2 \\
\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}) &= 0 && \Rightarrow \\
\frac{1}{\sigma^2} \boldsymbol{X}^\top (\boldsymbol{X\theta} - \boldsymbol{y}) + \lambda \boldsymbol{\theta} &= 0 && \Rightarrow \\
\left(\lambda \boldsymbol{I} + \frac{1}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X} \right) \boldsymbol{\theta^*} &= \frac{1}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{y} && \Rightarrow \\
\boldsymbol{\theta_R^*} &= (\sigma^2 \lambda \boldsymbol{I} + \boldsymbol{X}^\top \boldsymbol{X})^{-1} \boldsymbol{X}^\top \boldsymbol{y}
\end{aligned}$$

The optimal solution remains similar to the linear regression case; however the matrix that needs to be inverted is no longer simply $\boldsymbol{X}^\top \boldsymbol{X}$. As such, note that if $\lambda = 0$ then $\boldsymbol{\theta}_R^* = \boldsymbol{\theta^*}$. Similarly, gradient descent can be used to solve this, assuming constant step sizes $\gamma_t = \gamma$;

1. define starting point $\boldsymbol{\theta_0}$, set $t = 0$
2. update $t \leftarrow t + 1$, and update $\boldsymbol{\theta_{t+1}}$

$$\begin{aligned}
\boldsymbol{\theta_{t+1}} &= \boldsymbol{\theta_t} - \gamma_t \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta_t}) \\
&= \boldsymbol{\theta_t} - \gamma \left(\frac{1}{\sigma^2} \boldsymbol{X}^\top (\boldsymbol{X\theta_t} - \boldsymbol{y}) + \lambda \boldsymbol{\theta_t} \right) \\
&= \left((1 - \gamma\lambda) \boldsymbol{I} - \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{X} \right) \boldsymbol{\theta_t} + \frac{\gamma}{\sigma^2} \boldsymbol{X}^\top \boldsymbol{y}
\end{aligned}$$

3. repeat until stopping criterion

Non-linear regression uses a non-linear feature mapping $\phi(\cdot) : \mathbb{R}^D \to \mathbb{R}^p$ (typically $p > D$). With the non-linear ridge regression model, fitting ridge regression becomes the following;

$$f(\boldsymbol{x}, \boldsymbol{\theta}) = \phi(\boldsymbol{x})^\top \boldsymbol{\theta}$$

$$\begin{aligned}
y &= f(\boldsymbol{x}, \boldsymbol{\theta}) + \epsilon \\
\boldsymbol{\Phi} &= \begin{bmatrix} \phi(\boldsymbol{x_1}) & \cdots & \phi(\boldsymbol{x_N}) \end{bmatrix}^\top && \in \mathbb{R}^{N \times p} \\
\boldsymbol{\theta}^*_{\boldsymbol{R}} &= \underset{\boldsymbol{\theta} \in \boldsymbol{\Theta}}{\text{argmin}} \, \frac{1}{2\sigma^2} ||\boldsymbol{y} - \boldsymbol{\Phi\theta}||_2^2 + \frac{\lambda}{2} ||\boldsymbol{\theta}||_2^2 \\
&= (\sigma^2 \lambda \boldsymbol{I} + \boldsymbol{\Phi}^\top \boldsymbol{\Phi})^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y}
\end{aligned}$$

Consider regression with polynomial functions - the parameters we want to fit are the coefficients for each of the polynomial features;

$$f(x, \boldsymbol{\theta}) = \sum_{i=1}^{p} \theta_i x^{i-1}$$

We can obtain a number of models with different p values that all have a **training loss** close to 0 - however, they may have different **test loss**. Previously, we selected a model based on a validation dataset.

The ℓ_2 regulariser pushes the elements of $\boldsymbol{\theta}$ towards zero. Ridge regression helps balance between data fit and model simplicity; if for a given i, $\theta_i = 0$, then the feature x^{i-1} isn't in use.

$$R(\boldsymbol{\theta}) = ||\boldsymbol{\theta}||_2^2 = \sum_{i=1}^{p} \theta_i^2$$

Ridge regression gives an estimator of $\boldsymbol{\theta}$ with a smaller variance; giving a smaller expected test error than with just linear regression.

We assume there is no model mismatch, hence $\mathcal{D} = \{(\boldsymbol{x_n}, y_n)\}_{n=1}^N$ is generated by $y_n = f(\boldsymbol{x_n}; \boldsymbol{\theta_0}) + \epsilon_n$. This can also be written as $\mathcal{D} \sim p_{\text{data}}^N$. Note that $\boldsymbol{\theta_0}$ denotes the (unknown) ground truth parameters for the generation of the dataset. We also assume that the test data is also generated from the same process.

Similar to before, our goal is to estimate $\boldsymbol{\theta}$ using data $\mathcal{D}$, such that $\boldsymbol{\theta}^* \approx \boldsymbol{\theta_0}$ - the solution is written as $\boldsymbol{\theta}^*(\mathcal{D})$, when the dataset is given.

The expected ℓ_2 error for estimate is as follows (ideally we want an unbiased estimator with a low variance);

$$\begin{aligned}
\text{error}(\boldsymbol{\theta}^*) &= \mathbb{E}_{\mathcal{D} \sim p_{\text{data}}^N}[||\boldsymbol{\theta}^*(\mathcal{D}) - \boldsymbol{\theta_0}||_2^2] \\
&= ||\boldsymbol{\theta}^*(\mathcal{D}) - \mathbb{E}[\boldsymbol{\theta}^*(\mathcal{D})] + \mathbb{E}[\boldsymbol{\theta}^*(\mathcal{D})] - \boldsymbol{\theta_0}||_2^2 \\
&= \underbrace{||\boldsymbol{\theta}^*(\mathcal{D}) - \mathbb{E}[\boldsymbol{\theta}^*(\mathcal{D})]||_2^2}_{\text{variance}} + \underbrace{||\mathbb{E}[\boldsymbol{\theta}^*(\mathcal{D})] - \boldsymbol{\theta_0}||_2^2}_{\text{bias}^2} + \underbrace{\text{cross-term}}_{=0} \\
&= \underbrace{\text{tr}[\mathbb{V}_{\mathcal{D} \sim p_{\text{data}}^N}[\boldsymbol{\theta}^*(\mathcal{D})]]}_{\text{variance}} + \underbrace{||\mathbb{E}[\boldsymbol{\theta}^*(\mathcal{D})] - \boldsymbol{\theta_0}||_2^2}_{\text{bias}^2}
\end{aligned}$$

We are not only looking at the estimation error for one particular regression problem (one experiment). To compute the expectation, the experiment is computed multiple times, with each experiment sampling a new set of data from the previously mentioned generation process. The expected error can be written in matrix form;

$$\begin{aligned}
\text{Error}(\boldsymbol{\theta}^*) &= \mathbb{E}_{\mathcal{D} \sim p_{\text{data}}^N}[(\boldsymbol{\theta}^*(\mathcal{D}) - \boldsymbol{\theta_0})(\boldsymbol{\theta}^*(\mathcal{D}) - \boldsymbol{\theta_0})^\top] \\
&= \boldsymbol{b}(\boldsymbol{\theta}^*)\boldsymbol{b}(\boldsymbol{\theta}^*)^\top \boldsymbol{V}(\boldsymbol{\theta}^*) \\
\boldsymbol{b}(\boldsymbol{\theta}^*) &= \mathbb{E}_{\mathcal{D} \sim p_{\text{data}}^N}[\boldsymbol{\theta}^*(\mathcal{D})] - \boldsymbol{\theta_0} && \text{bias} \\
\boldsymbol{V}(\boldsymbol{\theta}^*) &= \mathbb{V}_{\mathcal{D} \sim p_{\text{data}}^N}[\boldsymbol{\theta}^*(\mathcal{D})] && \text{variance}
\end{aligned}$$

The expected prediction error is as follows;

$$y_{\text{test}} = \phi(\boldsymbol{x}_{\text{test}})^\top \boldsymbol{\theta_0} + \boldsymbol{\epsilon}$$

$$\begin{aligned}
f(\boldsymbol{x}_{\text{test}}, \boldsymbol{\theta}^*(\mathcal{D})) &= \phi(\boldsymbol{x}_{\text{test}})^\top \boldsymbol{\theta}^*(\mathcal{D})) \\
||y_{\text{test}} - f(\boldsymbol{x}_{\text{test}}, \boldsymbol{\theta}^*(\mathcal{D}))||_2^2 &= ||\phi(\boldsymbol{x}_{\text{test}})^\top \boldsymbol{\theta_0} - \phi(\boldsymbol{x}_{\text{test}})^\top \boldsymbol{\theta}^*(\mathcal{D})) + \boldsymbol{\epsilon}||_2^2 \\
&= ||\phi(\boldsymbol{x}_{\text{test}})^\top (\boldsymbol{\theta_0} - \boldsymbol{\theta}^*(\mathcal{D}))||_2^2 + ||\boldsymbol{\epsilon}||_2^2 + \text{cross-term} \\
\mathbb{E}[||\boldsymbol{\epsilon}||_2^2] &= \sigma^2 \\
\mathbb{E}[\text{cross-term}] &= 0 \\
||\phi(\boldsymbol{x}_{\text{test}})^\top (\boldsymbol{\theta_0} - \boldsymbol{\theta}^*(\mathcal{D}))||_2^2 &= \phi(\boldsymbol{x}_{\text{test}})^\top (\boldsymbol{\theta_0} - \boldsymbol{\theta}^*(\mathcal{D}))(\boldsymbol{\theta_0} - \boldsymbol{\theta}^*(\mathcal{D}))^\top \phi(\boldsymbol{x}_{\text{test}}) \\
\mathbb{E}[(\boldsymbol{\theta_0} - \boldsymbol{\theta}^*(\mathcal{D}))(\boldsymbol{\theta_0} - \boldsymbol{\theta}^*(\mathcal{D}))^\top] &= \text{Error}(\boldsymbol{\theta}^*) \\
\text{error}_{\text{pred}}(\boldsymbol{\theta}^*) &= \mathbb{E}_{\mathcal{D}\sim p_{\text{data}}^N}[\mathbb{E}_{(\boldsymbol{x}_{\text{test}}, y_{\text{test}})\sim p_{\text{data}}^N}[||y_{\text{test}} - f(\boldsymbol{x}_{\text{test}}, \boldsymbol{\theta}^*(\mathcal{D}))||_2^2]] \\
&= \mathbb{E}_{\boldsymbol{x}_{\text{test}}}[\phi(\boldsymbol{x}_{\text{test}})^\top \text{Error}(\boldsymbol{\theta}^*)\phi(\boldsymbol{x}_{\text{test}})] + \sigma^2
\end{aligned}$$

If we have two estimators $\boldsymbol{\theta_1}$ and $\boldsymbol{\theta_2}$ (for example, the former is the linear regressor and the latter is the ridge regressor) - if we can show the matrix form of the parameter estimator error of the first estimator is dominated by the matrix form of the latter, then we can show the expected prediction error of the former is smaller;

$$\text{Error}(\boldsymbol{\theta_1}) \preceq \text{Error}(\boldsymbol{\theta_2}) \Rightarrow \text{error}_{\text{pred}}(\boldsymbol{\theta_1}) \leq \text{error}_{\text{pred}}(\boldsymbol{\theta_2})$$

Note that $\text{Error}(\boldsymbol{\theta_1}) \preceq \text{Error}(\boldsymbol{\theta_2})$ means $\text{Error}(\boldsymbol{\theta_2}) - \text{Error}(\boldsymbol{\theta_1})$ is positive semi-definite, hence;

$$\forall \boldsymbol{\phi}\ [\boldsymbol{\phi}^\top (\text{Error}(\boldsymbol{\theta_1}) - \text{Error}(\boldsymbol{\theta_2}))\boldsymbol{\phi} \leq 0]$$

If we have a smaller estimation error, then we have a smaller prediction error.

Linear regression gives an unbiased estimator (the expectation of ϵ is zero, then the rest cancels out);

$$\mathbb{E}_{\mathcal{D}\sim p_{\text{data}}^N}[\boldsymbol{\theta}_{\boldsymbol{L}}^*(\mathcal{D})] = \mathbb{E}_{\mathcal{D}\sim p_{\text{data}}^N}[(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{\theta_0} + \boldsymbol{\epsilon})] = \boldsymbol{\theta_0}$$

However, ridge regression gives a biased estimator (as long as $\lambda \neq 0$);

$$\mathbb{E}_{\mathcal{D}\sim p_{\text{data}}^N}[\boldsymbol{\theta}_{\boldsymbol{R}}^*(\mathcal{D})] = (\sigma^2\lambda\boldsymbol{I} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\theta_0} \neq \boldsymbol{\theta_0}$$

If we know $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \boldsymbol{\Sigma})$ and $\boldsymbol{\epsilon'} = \boldsymbol{A}\boldsymbol{\epsilon}$, then $\boldsymbol{\epsilon'} \sim \mathcal{N}(0, \boldsymbol{A}\boldsymbol{\Sigma}\boldsymbol{A}^\top)$ The variance (for ridge regression) is as follows;

$$\begin{aligned}
\mathbb{V}_{\mathcal{D}\sim p_{\text{data}}^N}[\boldsymbol{\theta}_{\boldsymbol{R}}^*(\mathcal{D})] &= \mathbb{V}_{\mathcal{D}\sim p_{\text{data}}^N}[(\sigma^2\lambda\boldsymbol{I} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{\theta_0} + \boldsymbol{\epsilon})] \\
&= \mathbb{V}_{\mathcal{D}\sim p_{\text{data}}^N}[(\sigma^2\lambda\boldsymbol{I} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\epsilon}] \\
&= \sigma^2(\sigma^2\lambda\boldsymbol{I} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}(\sigma^2\lambda\boldsymbol{I} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi}) \\
&= \boldsymbol{V}(\lambda)
\end{aligned}$$

The variance for the linear regression estimator can be obtained from the above, by setting $\lambda = 0$;

$$\boldsymbol{V}(0) = \sigma^2(\boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}$$

The bias of the ridge regression estimator can be written as the follows (and similarly, set $\lambda = 0$ for the bias of the linear regression estimator);

$$\boldsymbol{b}(\lambda) = -\sigma^2\lambda(\sigma^2\lambda\boldsymbol{I} + \boldsymbol{\Phi}^\top\boldsymbol{\Phi})^{-1}\boldsymbol{\theta_0}$$

We can choose some λ in the range $0 \leq \lambda \leq \frac{2}{||\boldsymbol{\theta_0}||_2^2}$ such that;

$$\boldsymbol{b}(\lambda)\boldsymbol{b}(\lambda)^\top + \boldsymbol{V}(\lambda) \preceq \boldsymbol{V}(0) \Rightarrow \text{error}_{\text{pred}}(\boldsymbol{\theta}_{\boldsymbol{R}}^*) \leq \text{error}_{\text{pred}}(\boldsymbol{\theta}_{\boldsymbol{L}}^*)$$

This shows that ridge regression can achieve a better bias-variance trade-off in the parameter space. For any positive λ, ridge regression gives an estimator that has a smaller variance. However λ needs to be chosen carefully to ensure the bias isn't too large.

Principal Component Analysis

Regression is a supervised learning problem. In unsupervised learning, there is no output data y, and we want to figure out the underlying structure from data X.

The motivation for dimensionality reduction is that the data we observe daily tend to be high dimensional. For example; natural images (actual images) are a very small subset of all possible RGB images - in the majority of cases, it's just noise. Therefore, instead of a high dimensional vector of pixel data, the actual data lies in a lower dimensional space.

A user's data is typically quite sparse in a recommender system; for example a matrix of users' ratings on items - there may be millions of products, but very few will actually have ratings per user. Although the data may seem to be high dimensional, they only lie in a lower dimension subspace.

The problem can be set up as follows (no y, as it's unsupervised);

$$\begin{aligned}
\mathcal{D} &= \{\boldsymbol{x_1}, \dots, \boldsymbol{x_N}\} && (1)\\
\text{mean}(\boldsymbol{x_n}) &= \boldsymbol{0} && (2)\\
\boldsymbol{x_n} &\approx \boldsymbol{\tilde{x}_n} && (3)\\
&= \sum_{j=1}^{M} z_{nj}\boldsymbol{b_j} && (4)\\
z_{nj} &= \boldsymbol{b_j}^\top \boldsymbol{x_n} && (5)\\
\boldsymbol{B} &= [\boldsymbol{b_1}, \dots, \boldsymbol{b_M}] && (6)
\end{aligned}$$

Note the following;

(1) $\boldsymbol{x_n} \in \mathbb{R}^{D\times 1}$

(2) assume that we've already shifted the data

(3) an approximate projection in a lower-dimensional space, $M < D$

(5) note that the coordinates $\{z_{nj}\}$ are projections of the $\boldsymbol{x_n}$ vector onto a given basis.

(6) an orthonormal basis (which we want to figure out), $\boldsymbol{b_m} \in \mathbb{R}^{D\times 1}$

The goal is to minimise the ℓ_2 reconstruction error (consider the full orthonormal basis $\boldsymbol{B}_{\text{full}}$);

$$\boldsymbol{B}_{\text{full}} = \begin{bmatrix}\boldsymbol{b_1} & \cdots & \boldsymbol{b_M} & \boldsymbol{b_{M+1}} & \cdots & \boldsymbol{b_D}\end{bmatrix} \qquad \text{used in new basis, dropped}$$

$$\boldsymbol{x_n} = \underbrace{\sum_{j=1}^{M} z_{nj}\boldsymbol{b_j}}_{\boldsymbol{\tilde{x}_n}} + \sum_{j=M+1}^{D} z_{nj}\boldsymbol{b_j}$$

$$\boldsymbol{x_n} - \boldsymbol{\tilde{x}_n} = \sum_{j=M+1}^{D} z_{nj}\boldsymbol{b_j}$$

$$\begin{aligned}
L &= \frac{1}{N}\sum_{n=1}^{N} ||\boldsymbol{x_n} - \boldsymbol{\tilde{x}_n}||_2^2\\
&= \frac{1}{N}\sum_{n=1}^{N} ||\sum_{j=M+1}^{D} z_{nj}\boldsymbol{b_j}||_2^2\\
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{j=M+1}^{D} ||z_{nj}\boldsymbol{b_j}||_2^2\\
&= \frac{1}{N}\sum_{n=1}^{N}\sum_{j=M+1}^{D} z_{nj}^2
\end{aligned}$$

Note that the ℓ_2 norm can be put inside the (penultimate step) as the cross terms cancel out, since it is an orthonormal basis. The final step can be done as z_{nj} is a scalar, and the basis vectors are unit vectors. Plugging in the definition of z_{nj}, we get the following (notice the variance of $\{\boldsymbol{x_n}\}$);

$$L = \sum_{j=M+1}^{D} \boldsymbol{b_j}^\top \left(\frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x_n} \boldsymbol{x_n}^\top \right) \boldsymbol{b_j}$$

Therefore, our goal is to find the orthonormal basis $\boldsymbol{B}_\text{full}$ to minimise L. Note that $\boldsymbol{S} \in \mathbb{R}^{D \times D}$ is a symmetric matrix and positive semidefinite - for any $\boldsymbol{x} \in \mathbb{R}^{D \times 1}$; $\boldsymbol{x}^\top \boldsymbol{S} \boldsymbol{x} = \frac{1}{N} \sum_{n=1}^{N} (\boldsymbol{x}^\top \boldsymbol{x_n})^2 \geq 0$.

$$\begin{aligned}
\text{define } \boldsymbol{S} &= \frac{1}{N} \sum_{n=1}^{N} \boldsymbol{x_n} \boldsymbol{x_n}^\top \\
L &= \sum_{j=M+1}^{D} \boldsymbol{b_j}^\top \boldsymbol{S} \boldsymbol{b_j}
\end{aligned}$$

Assume the eigen decomposition of $\boldsymbol{S} = \boldsymbol{Q \Lambda Q}^\top$ ($\boldsymbol{Q}$ constructs an orthonormal basis of the $\mathbb{R}^D$ space);

$$\begin{aligned}
\boldsymbol{\Lambda} &= \text{diag}(\lambda_1, \dots, \lambda_D) && \lambda_1 \geq \dots \geq \lambda_D \geq 0 \\
\boldsymbol{b_j}^\top \boldsymbol{S} \boldsymbol{b_j} &= \boldsymbol{b_j}^\top \boldsymbol{Q \Lambda Q}^\top \boldsymbol{b_j} \\
&= \boldsymbol{\beta_j}^\top \boldsymbol{\Lambda} \boldsymbol{\beta_j} && \text{defined} \\
&= \sum_{d=1}^{D} \lambda_d \beta_{jd}^2 \\
\boldsymbol{\beta_j} &= \boldsymbol{Q}^\top \boldsymbol{b_j} \\
&= [\beta_{j1}, \dots, \beta_{jD}] \\
&= [\boldsymbol{q_1}^\top \boldsymbol{b_j}, \dots, \boldsymbol{q_D}^\top \boldsymbol{b_j}]
\end{aligned}$$

This gives us a constrained optimisation problem, where we have an orthonormal basis ($||\boldsymbol{b_j}||_2^2 = 1$ and vectors are perpendicular);

$$\min_{\boldsymbol{B}_\text{full}} L = \sum_{j=M+1}^{D} \sum_{d=1}^{D} \lambda_d \beta_{jd}^2$$

An iterative approach is as follows; first solve for $\boldsymbol{b_D}$, then solve $\boldsymbol{b_j}$ for $j = D-1, \dots, M+1$ - the constraint still applies that $\boldsymbol{b_D}$ is a unit vector;

$$\min_{\boldsymbol{b_D}} \sum_{d=1}^{D} \lambda_d \beta_{Dd}^2$$

Notice the following, hence it is a weighted sum of the eigenvalues;

$$\begin{aligned}
\beta_{Dd} &= \boldsymbol{b_D}^\top \boldsymbol{q_D} \\
\sum_{d=1}^{D} \beta_{Dd}^2 &= 1
\end{aligned}$$

Therefore, $\boldsymbol{b_D} = \boldsymbol{q_D}$, by ordering of the eigenvalues - it's just the eigenvector with the smallest eigenvalue. The iterative solution relies on proof by induction;

1. $\boldsymbol{b_D} = \boldsymbol{q_D}$
2. for $j = D-1, \dots, M+1$, assume $\boldsymbol{b_i} = \boldsymbol{q_i}$, where $i > j$ (already solved the previous optimisation problems)

(a) $\boldsymbol{b_j} \perp \boldsymbol{b_i},\ i > j \Rightarrow \boldsymbol{b_j} = \sum_{d=1}^{j} \beta_{jd}\boldsymbol{q_d}$

By the perpendicular constraint, if we consider $\boldsymbol{b_j}$ as a linear combination of the $\boldsymbol{q}$ vectors, it cannot contain the $\boldsymbol{q_i}$ vectors. If there is a contribution, the inner product wouldn't be equal to zero.

(b) $||\boldsymbol{b_j}||_2^2 = 1 \Rightarrow \sum_{d=1}^{j} \beta_{jd}^2 = 1$

(c) the minimisation problems is as follows, with respect to β_{jd};

$$\min_{\boldsymbol{\beta_j}} \sum_{d=1}^{j} \lambda_d \beta_{jd}^2$$

Therefore, $\boldsymbol{b_j} = \boldsymbol{q_j}$ (hence $\beta_{jj} = 1$, $\beta_{jd} = 0$ for $d \neq j$)

3. proof by induction tells us that $\boldsymbol{b_j} = \boldsymbol{q_j}$ for $j = M+1, \dots, D$

As such, we are projecting $\boldsymbol{x_n}$ to an orthogonal complement space;

$$\text{span}(\{\boldsymbol{q_j}\}_{j=M+1}^{D})^{\perp} = \{\boldsymbol{x} \in \mathbb{R}^{D\times 1} : \boldsymbol{x}^\top \boldsymbol{q_j} = 0, j = M+1, \dots, D\}$$

$$\boldsymbol{x_n} = \underbrace{\sum_{j=1}^{M} z_{nj}\boldsymbol{b_j}}_{\tilde{\boldsymbol{x}}_n} + \sum_{j=M+1}^{D} z_{nj}\boldsymbol{q_j} \qquad \boldsymbol{b_i} \perp \boldsymbol{q_j}$$

$$\tilde{\boldsymbol{x}}_n \in \text{span}(\{\boldsymbol{q_j}\}_{j=M+1}^{D})^{\perp}$$

In practice, PCA is done by doing the following;

1. start with the original dataset
2. centre the dataset by subtracting the mean from each data point
3. divide by the standard deviation to make the data unit free, the data will have variance 1 along each of the axes
4. compute eigenvalues and eigenvectors of the data covariance matrix
5. project data onto the principal subspace
6. undo the standardisation and move the projected data back into original space