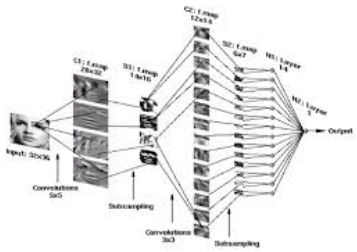
Advanced Machine Learning

Course Outline



Course Details and Topics

Course Structure

- Lectures
  - 11:00-11:45 Tuesday, Building 35 room 1005
  - 16:00-16:45 Tuesday, Building 44 room 1041 (L/T A)
  - 15:00-15:45 Thursday, Building 44 room 1041 (L/T A)
- Assessment
  - 80% Exam
  - 20% Problem Sheet

Problem Sheets

- I am going to provide many problem sheets
- One problem sheets will be marked and worth 20% (you will know which one this is)
- The other problem sheets are optional, but some small proportion of the questions will be on the exam
- I will go through the problem sheets, but if you have not attempted the questions you won't learn that much

What's in the Course

- This course is going to cover the core principles and mathematics behind machine learning
- It is not going to explicitly teach different machine learning algorithms, although some will be covered
- We are not looking at advanced algorithms but cover the principles, fish
- There are very good implementation available (e.g. scikit-learn)
- Along the way though we will meet (often many times) particular algorithms

Cracking the Code

- Mathematics is the language of machine learning
- You can do machine learning without mathematics, but if you want to develop and understand advanced algorithms then you have no choice
- This course invites you on a journey to crack the code of mathematics for machine learning
- If this isn't a challenge you want, then this is probably not the course for you

Topics

- Learning Theory
  - Bias-Variance
  - Overfitting, symmetry and regularisation
  - Ensembling, bagging and boosting
- Mathematics
  - Function Spaces: Kernel Methods and Gaussian Processes
  - Linear Algebra, embeddings, positive definiteness, subspace, determinants

Topics Continued

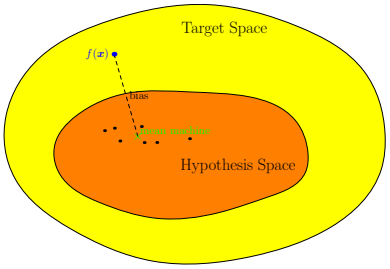
- Optimisation
  - Newton/Quasi-Newton Methods: convergence rates
  - SGD, momentum, ADAM
- Constrained Optimisation
  - KKT conditions
  - Duality Linear/Quadratic Programming
  - SVMs
- Convexity
  - Convex sets: linear constraints, PD matrices
  - Convex functions
  - SVMs, Lasso
  - Jensen's inequality

Topics Continued

- Probability
  - Naive Bayes
  - Gaussian Processes
  - Dependencies and Graphical Models
  - Expectations and MCMC
- Advanced Methods
  - Divergences: KL and Wasserstein
  - VAEs and GANs
  - Entropy and information theory
  - Variational Approximation

Advanced Machine Learning

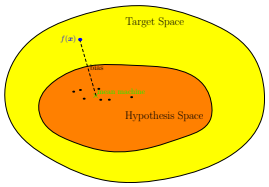
When Machine Learning Works



When ML Works, Bias Variance

Outline

1. **What Makes a Good Learning Machine?**
2. Bias-Variance Dilemma



What Makes a Good Learning Machine?

- We want to understand why some machine learning techniques work well and other don't
- To understand why these works we need to understand what makes a good learning machine
- For this we have to get conceptual and think about **generalisation** performance

  ***generalisation**: how well do we do on unseen data as opposed to the training data*

What Makes Machine Learning Hard?

- Typically we work in high dimensions (i.e. have many features)
- The problem can be over-constrained (i.e. we have conflicting data to deal with)—solve by minimising an error function
- The problem can be under-constrained (i.e. there are many possible solutions that are consistent with the data)—need to choose a plausible solution
- Typically in machine learning the data will be over-constrained in some dimensions and under-constrained in others
- We can't visualise the data to know what is going on

Least Squared Errors

- Suppose we want to learn some output y for a feature vector $\boldsymbol{x}$
- We construct a learning machine that makes a prediction $\hat{f}(\boldsymbol{x}|\boldsymbol{\theta})$
- We typically choose the machine to minimise a *training loss*

$$L_T(\mathcal{D}) = \sum_{(\boldsymbol{x},y)\in\mathcal{D}} \left(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}) - y\right)^2 = \sum_{i=1}^{m} \left(\hat{f}(\boldsymbol{x}_i|\boldsymbol{\theta}) - y_i\right)^2$$

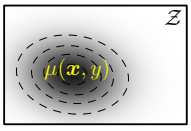
  where $\mathcal{D} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^m$ is a set of size m, sampled from a probability distribution $\mu(\boldsymbol{x},y)$

- We call this machine $\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_\mathcal{D})$

Generalisation Error

- We want to minimise the *generalisation loss* which in this case is

$$L_G(\mathcal{D}) = \sum_{(\boldsymbol{x},y)\in\mathcal{Z}} \mu(\boldsymbol{x},y) \left(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_\mathcal{D}) - y\right)^2$$

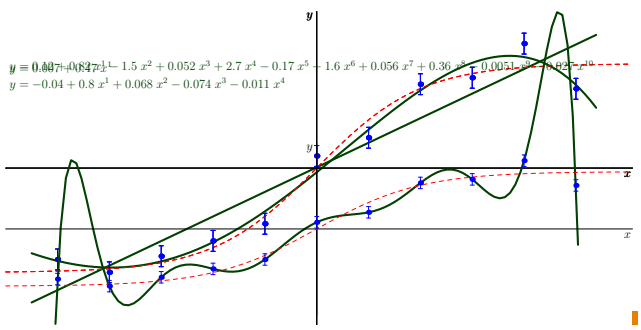


  (we can estimate this if we have some labelled examples $(\boldsymbol{x}_i, y_i)$ which we have not trained on)

- We want to minimise $L_G(\mathcal{D})$ but in practice we are minimising $L_T(\mathcal{D})$, *what could possibly go wrong?*

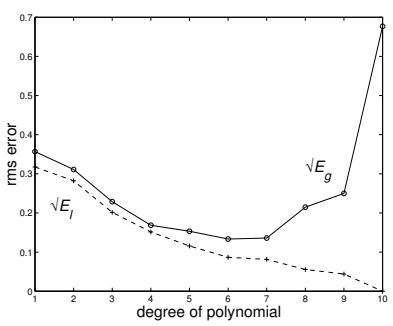
Too Simple or Too Complex?

- Fit $\hat{f}(x|\boldsymbol{\theta}_\mathcal{D})$ to data



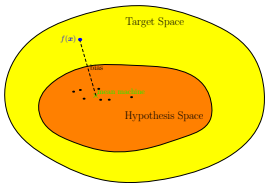
Measuring Generalisation Error for Regression

- Consider the regression example. The root mean squared error is



Outline

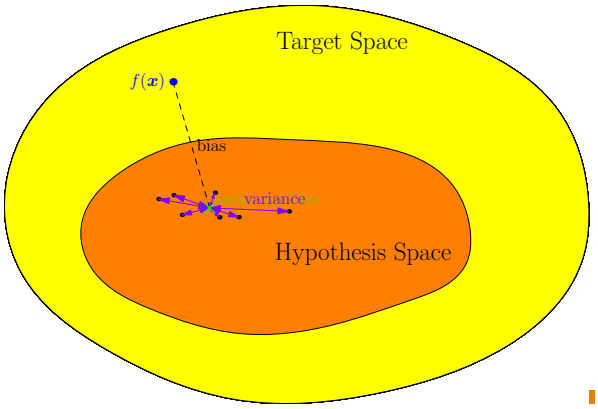
1. What Makes a Good Learning Machine?
2. **Bias-Variance Dilemma**



Expected Generalisation Performance

- Our generalisation performance will depend on our training set, $\mathcal{D}$
- To reason about generalisation we can ask what is the *expected generalisation loss*, when we average over all different data sets of size m drawn independently from $\mu(\boldsymbol{x},y)$
- For each data set, $\mathcal{D}$, we would learn a different approximator $\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}})$
- Note that in practice we only get one data set. We might be lucky and do better than the expected generalisation or we might be unlucky and do worse

Approximation and Estimation Errors



Mean Machine

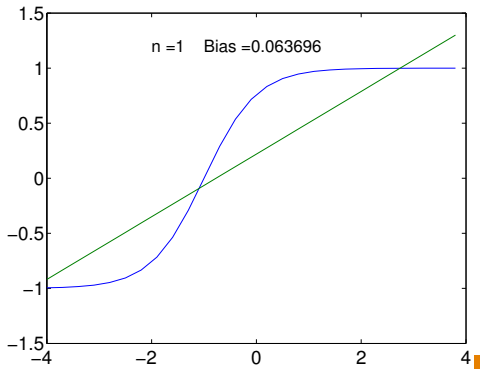
- To help understand generalisation we can consider the mean prediction with respect to machines trained with all data sets of size m

$$\hat{f}_m(\boldsymbol{x}) = \mathbb{E}_{\mathcal{D}}\Big[\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}})\Big]$$

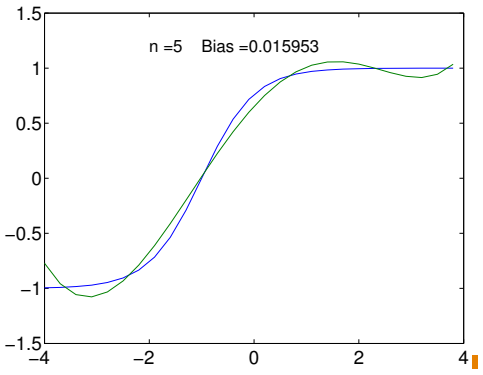
- We can define the **bias** to be generalisation performance of the mean machine

$$B = \sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\Big(\hat{f}_m(\boldsymbol{x}) - y\Big)^2$$

Regression Example $n = 1$



Regression Example $n = 5$



Bias and Variance

Consider the expected generalisation for data sets of size $|\mathcal{D}| = m$

$$\begin{aligned}
\bar{L}_G = \mathbb{E}_{\mathcal{D}}[L_G(\mathcal{D})] &= \mathbb{E}_{\mathcal{D}}\left[\sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\Big(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - y\Big)^2\right] \\
&= \sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\mathbb{E}_{\mathcal{D}}\left[\Big(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - y\Big)^2\right] \\
&= \sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\mathbb{E}_{\mathcal{D}}\left[\Big(\big(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - \hat{f}_m(\boldsymbol{x})\big) + \big(\hat{f}_m(\boldsymbol{x}) - y\big)\Big)^2\right] \\
&= \sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\Bigg(\mathbb{E}_{\mathcal{D}}\left[\Big(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - \hat{f}_m(\boldsymbol{x})\Big)^2 + \Big(\hat{f}_m(\boldsymbol{x}) - y\Big)^2\right] \\
&\qquad + 2\mathbb{E}_{\mathcal{D}}\Big[\Big(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - \hat{f}_m(\boldsymbol{x})\Big)\Big(\hat{f}_m(\boldsymbol{x}) - y\Big)\Big]\Bigg)
\end{aligned}$$

Cross Term

- The cross term vanishes

$$\begin{aligned}
C &= \mathbb{E}_{\mathcal{D}}\Big[\Big(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - \hat{f}_m(\boldsymbol{x})\Big)\Big(\hat{f}_m(\boldsymbol{x}) - y\Big)\Big] \\
&= \mathbb{E}_{\mathcal{D}}\Big[\Big(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - \hat{f}_m(\boldsymbol{x})\Big)\Big]\Big(\hat{f}_m(\boldsymbol{x}) - y\Big) \\
&= \Big(\mathbb{E}_{\mathcal{D}}\Big[\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}})\Big] - \hat{f}_m(\boldsymbol{x})\Big)\Big(\hat{f}_m(\boldsymbol{x}) - y\Big) \\
&= \Big(\hat{f}_m(\boldsymbol{x}) - \hat{f}_m(\boldsymbol{x})\Big)\Big(\hat{f}_m(\boldsymbol{x}) - y\Big) = 0
\end{aligned}$$

- Thus

$$\bar{L}_G = \sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\mathbb{E}_{\mathcal{D}}\left[\Big(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - \hat{f}_m(\boldsymbol{x})\Big)^2 + \Big(\hat{f}_m(\boldsymbol{x}) - y\Big)^2\right]$$

Bias and Variance

- We can write the expected generalisation loss as

$$\mathbb{E}_{\mathcal{D}}[L_G(\mathcal{D})] = \sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - \hat{f}_m(\boldsymbol{x})\right)^2\right] + \sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\left(\hat{f}_m(\boldsymbol{x}) - y\right)^2 = V + B$$

- Where B is the bias and V is the variance defined by

$$V = \sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - \hat{f}_m(\boldsymbol{x})\right)^2\right]$$

Bias-Variance Dilemma

- The bias measure the generalisation performance of the *mean machine* and is large if the machine is too simple to capture the changes in the function we want to learn
- The variance measures the variation in the prediction of the machines as we change the data set we train on

$$V = \sum_{\boldsymbol{x}\in\mathcal{X}} \mu(\boldsymbol{x},y)\mathbb{E}_{\mathcal{D}}\left[\left(\hat{f}(\boldsymbol{x}|\boldsymbol{\theta}_{\mathcal{D}}) - \hat{f}_m(\boldsymbol{x})\right)^2\right]$$

- The variance is usually large if we have a complex machine
- Striking the right balance is often the key to getting good results

Balancing Bias and Variance

- We want to choose a learning machine that is complex enough to capture the underlying function we are trying to learn, but otherwise as simple as possible
- There are a number of tricks to achieve this balance
- Some require us to preprocess the data to reduce the number of inputs
- Some machines cleverly adjust their own complexity
- This course looks at machines that achieve this balance

Lessons

- This course is about understanding machine learning techniques that work well
- Which one to use will depend on the data set
- One of the most useful intuitions about what works is the bias-variance framework
- The bias is high for simple machines that can't capture the data
- The variance is high for complex machines that are sensitive to the training set
- Good machines are powerful enough to capture complex data sets, but they can control their own capacity (ability to (over-)fit the data)

Advanced Machine Learning

Over-Fitting

Overfitting, regularisation, feature selection

Outline

1. **Over-fitting?**
2. Controlling Complexity
3. Hidden structure
4. Regularisation

Over-fitting

- Complex machine can **over-fit**

  ***over-fitting**: fitting the training data well at the cost of getting poorer generalisation performance*

- Three red cars. . .
- If we used an infinitely flexible machine we can fit our training data perfectly, but would have no generalisation ability

Binary Classification Task for You

Class 1 Class 2

Which Category?

- Which category does the following image belong to?

Spurious Rules

- You ask a learning machine to solve a task based on data
- It will find a rule that does this, but not necessary the rule you had in mind—machine learning isn't magic, it can't read your mind
- Infinitely flexible machines have an infinity of spurious rules they can learn—they are useless
- What should we do?

All Binary Functions

$\mathcal{D} = \{(000, 0), (010, 1), (110, 1), (001, 0), (101, 0)\}$

$\boldsymbol{x}_0 = 000 \quad y_0 = \begin{cases} 0 \\ \mathbf{x} \end{cases}$

$\boldsymbol{x}_1 = 100 \quad y_1 = \begin{cases} 0 \\ 1 \end{cases}$ unseen

$\boldsymbol{x}_2 = 010 \quad y_2 = \begin{cases} \mathbf{x} \\ 1 \end{cases}$

$\boldsymbol{x}_3 = 110 \quad y_3 = \begin{cases} \mathbf{x} \\ 1 \end{cases}$

$\boldsymbol{x}_4 = 001 \quad y_4 = \begin{cases} 0 \\ \mathbf{x} \end{cases}$ seen

$\boldsymbol{x}_5 = 101 \quad y_5 = \begin{cases} 0 \\ \mathbf{x} \end{cases}$

$\boldsymbol{x}_6 = 011 \quad y_6 = \begin{cases} 0 \\ 1 \end{cases}$

$\boldsymbol{x}_7 = 111 \quad y_7 = \begin{cases} 0 \\ 1 \end{cases}$ unseen

Are MLPs Universal Approximators?

- Yes and No
- Yes: If you give me any function, I can find an MLP that approximates that function to any desired accuracy
- No: If you give me an MLP, I can find a function with an arbitrary high approximation error
- Would an MLP that could approximate any function be useful?
- **Absolutely not!**

Outline

1. Over-fitting?
2. **Controlling Complexity**
3. Hidden structure
4. Regularisation

Controlling Complexity

- Infinitely flexible machine don't generalise (because any unseen data could have any value)
- **Machine learning only works because there is some structure in the data**
- A successful machine should capture this structure
- Even deep learning machines with millions of parameters only work because they successfully capture the structure of images or text
- Different learning machines have different performance on different problems because the data has different structure

Training Examples

- As we increase the number of training examples, we make it hard to find a spurious rule
- Bigger data sets allow us to use more complicated machines
- Part of the success of deep learning is because they use huge training sets—but this is only a part of their success
- (Labelled) data is often expensive to collect so we sometimes have no choice but to use a small training set
- One of the limitations of using deep learning comes because we often have limited data

Identifying Structure

- In some cases we know *a priori* some of the structure in the data
- In images we believe the identity of an object is invariant to translation and scaling
- The success of *convolutional neural networks* (CNNs) in deep learning is in large part because the convolutions respect translational invariance

Preprocessing

- Structure might often be obscure to the learning machine
- If we are trying to predict the spread of disease then a list of place names might be a lot less useful than their coordinates
- Imposing an ordering on an unordered set might **not** be useful

$$\{\text{“blue”} : 0,\ \text{“brown”} : 1,\ \text{“green”} : 2,\ \text{“black”} : 3\}$$

- Choosing an encoding that reflect meaningful structure is essential to successful machine learning

Automatic Preprocessing

- One view of deep learning is that each layer (particularly in CNNs) acts as a preprocessor
- That is, it finds filters that captures features salient to the problem being tackled
- For both images and texts we expect salient features to be spatially localised (CNN finds localised filter)
- The deep structure allows ever more complicated features to be captured—that is, we can find spatially localised features on different scales
- Having very large datasets and simple filters (the number of weights in the CNN layers tends to be small) stops overfitting

Outline

1. Over-fitting?
2. Controlling Complexity
3. **Hidden structure**
4. Regularisation

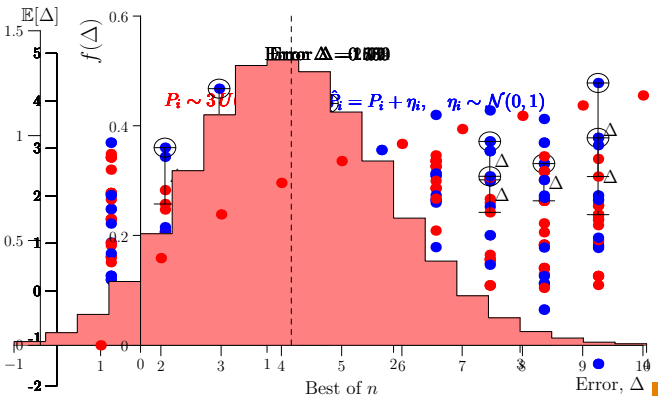
Hidden Structure

- Often the structure of data is invisible to us
- A very successful strategy is to try many different machine learning techniques and choose the best (stupid but effective)
- Often learning machines have adjustable parameters (hyper-parameters) that we have to set (they are the same for all input data, but change with the problem)
- We need to choose the hyper-parameters to fit the data in our problem
- Fine tuning hyper-parameter is important and almost always required (true in SVMs, MLP, deep learning)

Measuring Generalisation Performance

- Recall, we want to predict **unseen** data
- **You cannot use data that you have trained on!**—you will overfit
- Need to split your data up into training and validation set
- Use the validation set to choose the hyper-parameters
- You need a separate testing set if you want to measure your generalisation performance

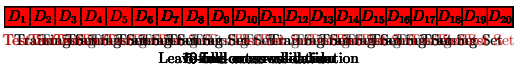
The Overfitting Game



Cross Validation

- If you want to use more data for training then you can use cross validation
- K-fold cross validation splits the data into K groups

$$\mathcal{D} = \{D_i\}_{i=1}^{P} \quad D_i = (\mathbf{x}_i, y_i)$$

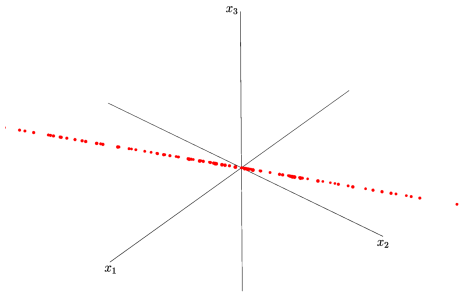


$$E_{g_7} = \frac{5.5 + 8.5 + 5.5 + 1.8 + 2.3 + 4.8 + 3.7 + 3.6 + 7.4 + 4.6 + 0.99 + 4.5 + 4.6 + 5.4 + 6.2 + 3.3 + 2.7}{20} = 4.3$$

- Leave-one-out cross-validation is extreme case

Hidden Structure

Can't spot low dimensional data by looking at numbers



Dimensionality Reduction

- We can sometimes simplify our machines by using less features
- We can project our data onto a lower dimensional sub-space (e.g. one with the maximum variation in the data: PCA)
- We can use clustering to find exemplars and recode our data in terms of distances from the exemplars (radial basis functions)
- Whether this helps depends on whether the information we discard is pertinent to the task we are trying to perform

Feature Selection

- Spurious features will allow us to find spurious rules (**over-fitting**)
- We can try different combinations of features to find the best set, although it rapidly becomes intractable to do this in all ways
- We can use various heuristics to decide which features to keep, but no heuristic is fail-safe method to find the best set of features
- Feature selection however can be powerful, often we can get very good results by keeping only a few variables
- As well as possibly improving generalisation we also get a more **interpretable** rule

Normalising Features

- Measuring a feature in millimeters or kilometers is going to make a lot of difference to the size of that feature
- Many learning algorithms are sensitive to the size of a feature (larger features are more important)
- If we don't know how important different features are then it makes sense to normalise our features. E.g.

$$x_i^\alpha \leftarrow \frac{x_i^\alpha - \hat{\mu}_i}{\hat{\sigma}_i}, \quad \hat{\mu} = \frac{1}{m}\sum_{\beta=1}^{m} x_i^\beta, \quad \hat{\sigma}_i^2 = \frac{1}{m-1}\sum_{\beta=1}^{m}(x_i^\beta - \hat{\mu}_i)^2$$

Outline

1. Over-fitting?
2. Controlling Complexity
3. Hidden structure
4. **Regularisation**

Explicit Regularisation

- As you've seen in the foundations of ML course, we can modify our error function to choose smoother functions

$$L = \sum_{k=1}^{m} \left(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_k - y_k\right)^2 + \nu\|\boldsymbol{w}\|^2$$

  (Good to normalise features)
- Second term is minimised when $w_i = 0$
- If w_i is large then

$$f(\boldsymbol{x}|\boldsymbol{w}) = \boldsymbol{w}^{\mathsf{T}}\boldsymbol{x} = \sum_{i=1}^{p} w_i x_i$$

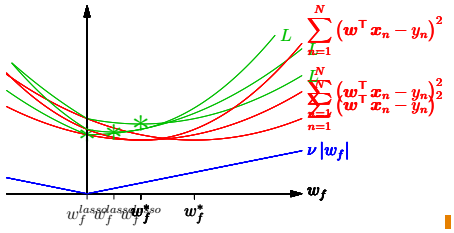
  varies rapidly as we change x_i

Lasso

- We can use other regularisers

$$L = \sum_{k=1}^{m} \left(\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_k - y_k\right)^2 + \nu \sum_{i=1}^{p} |w_i|$$

- Spurious features (e.g. colour of flag on energy consumption) will give us a small improvement in training error

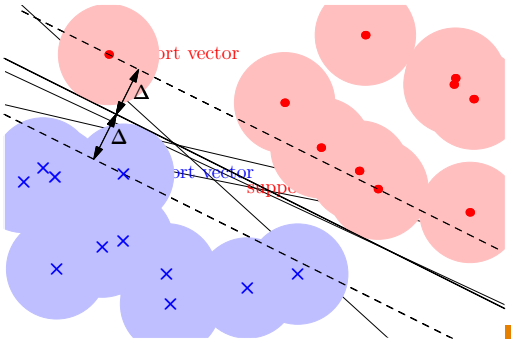


Implicit Regularisation

- In the last two examples we added an explicit regularisation term that made the function we learnt simpler
- Some learning machines do this less explicitly
- Some deep learning architectures do subtle averaging
- Sometimes the architecture biases the machine to find a simple solution

Maximum Margin Machines

- Perceptrons have many options to separate data



- SVMs choose the machine with the biggest margins

Success of SVMs

- SVMs regularise themselves by choosing the machine with the largest margin
- This ensures maximum stability to noise on the data
- It leads to very good generalisation on small datasets—usually beats everything else
- But you still need to normalise the features
- You also need to tune its hyper-parameters (C and sometimes γ)

Lessons

- Machine learning isn't magic
- It works when the learning machine is well attuned to the problem
- Sometimes you can help by preprocessing your data
- Sometimes there is a regularisation term that helps select a simpler machine
- Most machines have hyper-parameter that you tune to match the machine to the data
- Really clever machines try to do this matching automatically

Advanced Machine Learning

Symmetry

Inductive Bias, Symmetry, Invariance, Group theory

Outline

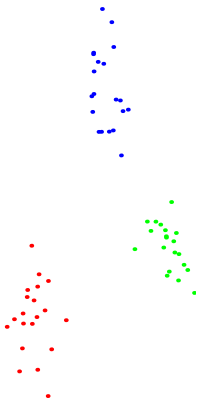
1. **Inductive Bias**
2. Invariance
3. Group Theory

Inductive Bias

- Learning machines differ in how they split up the space of inputs
- Perceptrons split the space using a hyperplane, RBFs split the space depending on some set of centres, MLPs successive split the world at different layers
- These differences are known as the **inductive bias** of the machine
- The different inductive bias lead to different performance
- How can we exploit the inductive bias to get good performance?

The Real World

- The problems that we care about come from a highly structured world where the solutions inherit some of the properties of that world
- Usually similar input have similar outputs (we don't expect to change one pixel in an image and that it should be classified differently)
- The classes are often quite well separated



Smoothness

- A natural assumption is smoothness (the classification surface or regression surface changes slowly as we change the input)
- Often there is a hyperparameter that controls this
  - The kernel parameter γ in SVMs (or ℓ in Gaussian Processes)
  - k in k-nearest neighbours
  - The size of the L_2 regulisation term in neural networks
- We often have to choose these using trial and error (using a validation set)

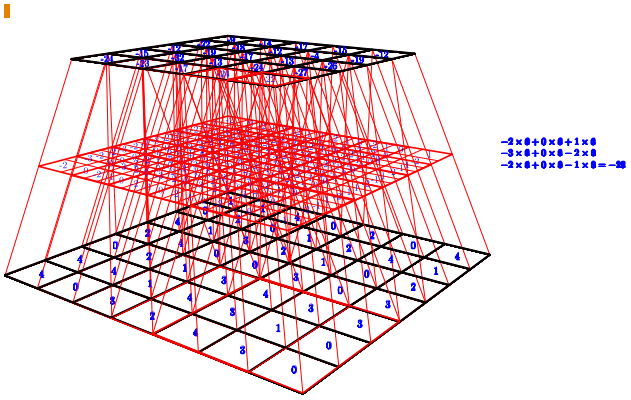
Outline

1. Inductive Bias
2. **Invariance**
3. Group Theory

Symmetry

- There are sometimes symmetries in the world we can exploit
- In image classification we consider an object should be classified the same wherever it is in the image
- The classification should be invariant to translations (scaling and perhaps rotations)
- The classification function possesses a translational symmetry
- Learning machines that respect translational invariance have far less propensity to overfit—they don't need to learn that the same object in a different part of the image is the same

Convolutional Neural Networks



Equivariance

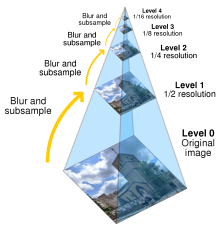
- CNNs are not translationally invariant
- The output will be different if we shift the input
- They are (up to edge effects) **equivariant** in that we would get the same result if we shifted both the input and the output in the same way
- They are not perfectly equivariant (they break at the pixel level and at the full image level—due to edges)
- But they are sufficient good that CNNs have dominated computer vision for years (ignoring visual transformers)

Other Symmetries

- The identity of an object should not change if we move the camera further away or rotate the camera
- CNNs are not naturally scale or rotationally invariant (although there are attempts to remedy this)
- Rotation invariance doesn't always reflect "the world"—people usually stand on their feet not on their heads
- Although you want rotational invariance when interpreting aerial images

Scale Symmetry

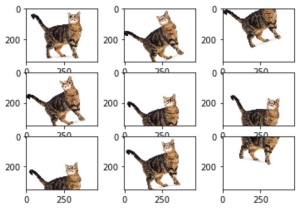
- CNNs are not scale invariant
- There are various pyramid image processing techniques to try and build in approximate scale invariance
- However, in many image classification tasks we concentrate on the object in the foreground
- Many datasets rescale objects in the image so they are roughly the same size



Learning Invariance

- Currently we are not smart enough to come up with networks that are scale invariant or rotational invariant or invariant to small distortions
- We therefore hope that our networks learns this by training on objects of different sizes, orientations, lighting conditions, etc.
- Knowing this is an invariance we want, it is now common practice to do data augmentation where we perform transformations that replicate physical plausible changes in the data
- We are removing spurious rules by (artificially) increasing our dataset size

Augmentations



- Data augmentation is standard practice in deep learning for images and makes a very considerable difference

Other Symmetries Exist

- In NLP it is now common to translate sentences to a foreign language and back to the original language to learn invariance to minor changes in the words used
- There are cases where we care about sets of objects, but not their ordering
- An example might be sets of objects in an image or vertices in a graph
- Here constructing operators that are equivariant (the same if we permute the inputs and outputs) can again reduce overfitting

Normalisation

- Sometimes clever normalisation can make a lot of difference to the performance of an algorithm
- We might do this by feature engineering (divide the length on the nose by the width between the persons eyes)
- We might do this by normalising the output—use cosine similarity rather than distance

$$\cos(\theta) = \frac{\boldsymbol{a}^\top \boldsymbol{b}}{\|\boldsymbol{a}\|\|\boldsymbol{b}\|}$$

- These make the results invariant to the distance of the camera or the length of $\boldsymbol{a}$ and $\boldsymbol{b}$

Outline

1. Inductive Bias
2. Invariance
3. **Group Theory**

Group Theory

- Symmetries are very well studied in mathematics
- The language of symmetries are **groups**
- These capture the structure of transformations that represent some symmetry
- Group theory is highly abstract and it is easy to lose sight of its connection to symmetry, but it has proved itself extraordinarily powerful in revealing structures

Groups

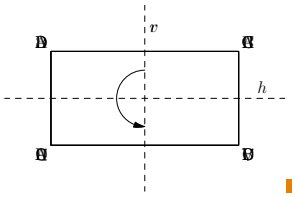
- Groups consists of a set (we can think of the elements as transformations), $\mathcal{G} = \{a,b,\cdots\}$
- And an operator "$\cdot$" (which we can often think of as composition)
- That is, $a \cdot b$ we can interpret as apply transformation b and then apply transformation a
- The set and transform form a group if they satisfy four axioms
  - ⋆ closure: for all $a,b \in \mathcal{G}$ then $a \cdot b \in \mathcal{G}$
  - ⋆ associativity: $a \cdot (b \cdot c) = (a \cdot b) \cdot c$
  - ⋆ existence of an identity e such that $a \cdot e = a$ for all a
  - ⋆ Every element, a, has an inverse a^{-1} such that $a^{-1} \cdot a = e$

Tossing a coin

- We consider a coin with two actions
  - ⋆ flip the coin: f
  - ⋆ leave the coin alone: e
- These form a group (the cyclic group C_2)
  - ⋆ Closure is obvious $f \cdot e = f$, $f \cdot f = e$, $e \cdot e = e$ and $e \cdot f = f$
  - ⋆ Associativity: e.g. $(e \cdot f) \cdot f = e \cdot (f \cdot f)$
  - ⋆ e is the identity
  - ⋆ $e^{-1} = e$ and $f^{-1} = f$

The Four Group

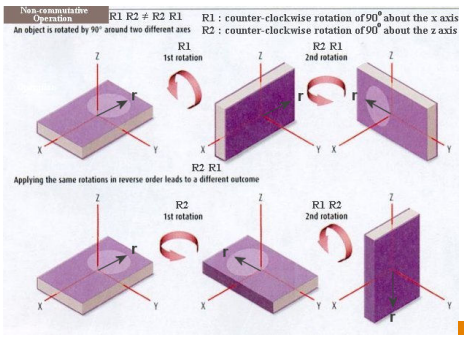
- A slightly less trivial example represents the set of transformations that leaves a rectangle invariant



	e	v	h	r
e	e	v	h	r
v	v	e	r	h
h	h	r	e	v
r	r	h	v	e

Non-commutativity

- Not all groups commute
- For 3-d rotations : $R_x \cdot R_z \neq R_z \cdot R_x$



Associativity

- We can think of the elements of a group as transformations acting on some object (e.g. rotations of a book)
- We consider $\cdot$ to denote composition so $a \cdot b$ denotes apply action b then action a
- When we say $c = a \cdot b$ we mean that action c is equivalent to apply action b followed by action a
- Suppose $g = a \cdot b$ and $h = b \cdot c$ then

$$g \cdot c = a \cdot h$$
$$(a \cdot b) \cdot c = a \cdot (b \cdot c)$$

- These correspond to an equivalent set of actions

Permutations

- The set of permutations of n objects form a group (known as the symmetric group S_n)
- The composition of a permutation is a permutation
- There is an identity (do nothing)
- For every permutation there is an inverse permutation
- Obviously S_n has $n!$ elements (you can reorder elements in n! ways)
- Permutations have a lot of structure (in terms of elements cycling, subgroups, etc.)

Infinite Groups

- The set of integers under addition form a group
- Its closed, has an identity, 0, and each element n has an inverse $-n$
- The group describes the set of discrete shifts
- The set of rational number under addition also forms a group
- Nothing new here, you've known about these since you were a toddler

Continuous Groups

- The set of reals under addition form a group and describes translations in one dimension
- The set of reals excluding 0 form a group under multiplication with an identity 1 and inverse of x being $1/x$
- These describe scale transformation
- These are not terrifically interesting groups (or at least groups you are so familiar with that they are now boring)
- Things get more interesting in high dimensional space

Lie Groups

- Continuous groups are known as Lie Groups
- They describe things like general translational invariance, rotational invariance, relativistic invariance
- The transformations can be represented by a set of matrices
- To study these one looks at infintesimal transformations and the algebra between these infintesimal transformations (Lie algebras)
- This is much studied in physics and occasionally studied in machine learning, although more on the sidelines

Rotations $SO(n)$

- The set of $n \times n$ orthogonal matrices (matrices, $\mathbf{M}$, such that $\mathbf{M}^\mathsf{T}\mathbf{M} = \mathbf{M}\mathbf{M}^\mathsf{T} = \mathbf{I}$) form a group $O(n)$
- Note that $\det(\mathbf{M}) = \pm 1$
- These correspond to rotations and reflections
- The group of matrices that don't correspond to reflections (where $\det(\mathbf{M}) = 1$) form a subgroup $SO(n)$
- $SO(2)$ consist of the 2-d rotation matrices

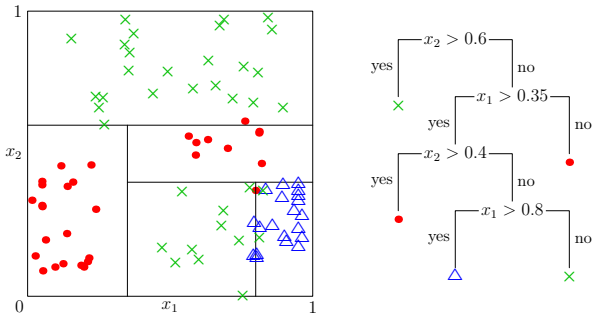
$$\begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$$

Summary

- Invariances are important as if a machine properly captures a true invariance it can massively limit the space of functions being approximated—reducing the chances of finding spurious rules
- Often it is hard to build-in invariances into the machine learning model and practitioners will often resort to data augmentation in the hope that these invariances can be learned
- There is a mathematical language around describing the structure of symmetries: **group theory**
- You should know *group theory* exists, it is occasionally used in machine learning, but not that much
- I'm not going to expect you to know any details about group theory

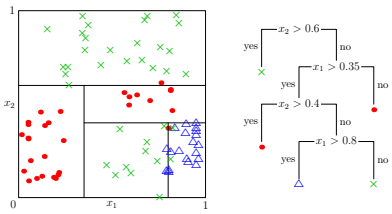
Advanced Machine Learning

Ensemble Methods



Decision Trees, Averaging, Bagging

Outline



Removing Variance By Averaging

- We can reduce the variance and hence improve our generalisation error by averaging over different learning machines
- There are a number of different techniques for doing this that go by the name of **ensemble methods** or **ensemble learning**
- This trick can be used with many different learning machines, but is clearly most practical for machine that can be trained quickly
- (nevertheless, even for deep learning taking the average response of many machines is usually done to win competitions)

Ensembling of Decision Trees

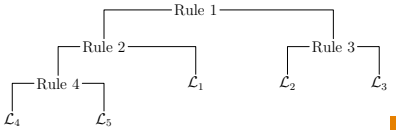
- One set of algorithms where ensembling are common place are decision trees
- These are particularly good for handling messy data
  - ⋆ categorical data
  - ⋆ mixture of data types
  - ⋆ missing data
  - ⋆ large data sets
  - ⋆ multiclass
- In many competitions ensembled trees, particularly *random forests* and *gradient boosting* beat all other techniques

Decision Trees

- A decision trees builds a binary tree to partition the data, $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | i = 1,\ldots,m\}$, into the leaves of the tree
- Each decision rule depends on a single feature
- At each step the rule is chosen that maximise the "*purity*" of the leaf nodes
- Decisions can be made on numerical values or categories

Partitioning

- Consider a classification problems with examples $(\boldsymbol{x}, y)$ belonging to some classes $y \in \mathcal{C}$
- The data is partitioned by the tree into leaves



- The proportion of data points in leaf $\mathcal{L}$ belonging to class c is

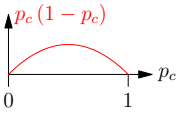
$$p_c(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{(\boldsymbol{x},y)\in\mathcal{L}} [\![y = c]\!]$$

where $[\![y = c]\!] = 1$ if $y = c$ and 0 otherwise

Leaf Purity

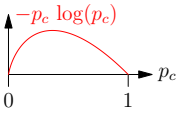
- Two different purity measures, $Q_m(\mathcal{L})$, for a leaf node $\mathcal{L}$ are commonly used
  - ⋆ **Gini index**

$$Q^g_m(\mathcal{L}) = \sum_{c\in\mathcal{C}} p_c(\mathcal{L})\,(1 - p_c(\mathcal{L}))$$

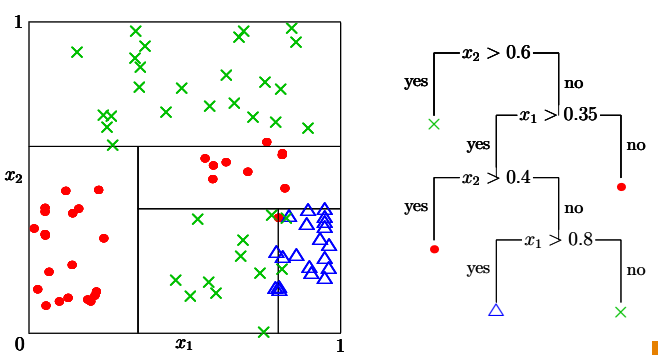


  - ⋆ **Cross-entropy**

$$Q^e_m(\mathcal{L}) = -\sum_{c\in\mathcal{C}} p_c(\mathcal{L}) \log(p_c(\mathcal{L}))$$



Building Decision Trees

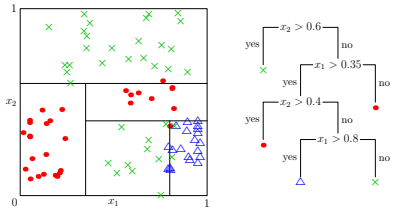


Observations

- Decision trees are very useful for exploring new data sets—the tree shows what features are most important
- Decision trees can also be used for regression problems
  - ⋆ Approximate function by a series of rules
  - ⋆ Reduce variance between data points assigned to leaf nodes
- CART is a classic implementation that builds **C**lassification **A**nd **R**egression **T**rees
- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets—high variance

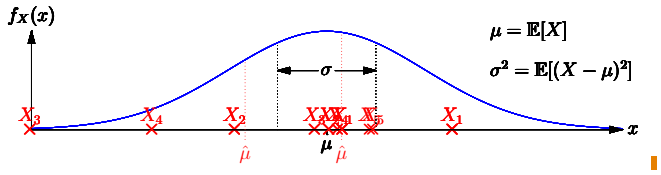
Outline

1. Decision Trees
2. **Bagging**



Error In The Means

- By taking the mean over many samples we can reduce the variance and thus improve our generalisation performance
- To get a feel for this consider estimating the mean of a random variable, X, from a number of samples ($n = 5$ in the example below)



Mean and Variance

- The expected value of the mean, $\hat{\mu}_n$, of n random **independent** variables, X_i, is the expected value $\mu = \mathbb{E}[X_i]$

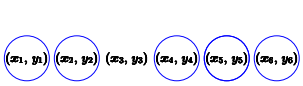
$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^n X_i\right] = \frac{1}{n}\sum_{i=1}^n \mathbb{E}[X_i] = \frac{1}{n}\sum_{i=1}^n \mu = \mu$$

- The variance is $\mathbb{E}\left[(\hat{\mu}_n - \mu)^2\right]$ or equivalently

$$\begin{aligned}\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^n (X_i-\mu)\right)^2\right] &= \frac{1}{n^2}\mathbb{E}\left[\sum_{i=1}^n (X_i-\mu)^2 + \sum_{i=1}^n\sum_{\substack{j=1\\ j\neq i}}^n (X_i-\mu)(X_j-\mu)\right] \\ &= \frac{1}{n^2}\sum_{i=1}^n\left(\mathbb{E}\left[(X_i-\mu)^2\right] + \sum_{\substack{j=1\\ j\neq i}}^n \mathbb{E}[X_i-\mu]\mathbb{E}[X_j-\mu]\right) \\ &= \frac{1}{n^2}\sum_{i=1}^n \sigma^2 = \frac{1}{n}\sigma^2\end{aligned}$$

Bootstrap Aggregation (Bagging)

- To reduce the variance in a learning machine (such as a decision tree) we can average over many machines
- To average many machines they must learn something different
- We only have one data set, but we can resample from the data set to make them look a bit different—this is known a **bootstrapping**



(x_5, y_5) (x_4, y_4) (x_5, y_5) (x_6, y_6) (x_2, y_2) (x_1, y_1)
(x_1, y_1) (x_5, y_5) (x_3, y_3) (x_2, y_2) (x_5, y_5) (x_2, y_2)
(x_6, y_6) (x_1, y_1) (x_3, y_3) (x_6, y_6) (x_3, y_3) (x_4, y_4)
(x_4, y_4) (x_3, y_3) (x_1, y_1) (x_6, y_6) (x_4, y_4) (x_6, y_6)
(x_3, y_3) (x_2, y_2) (x_3, y_3) (x_4, y_4) (x_6, y_6) (x_2, y_2)

Performance of Bagging

- For classification we get our different machines to vote
- For regression we can average the prediction of different machines
- Bagging improves the performance of decision trees
- However, we can usually do better using Boosting
- This is because our decision trees are correlated

Variance of Positive Correlated Variables

- If we calculate the variance of the mean of positively correlated variables with correlation ρ we find

$$\frac{1}{n^2}\mathbb{E}\left[\left(\sum_{i=1}^n X_i - \mu\right)^2\right] = \rho\sigma^2 + \frac{1-\rho}{n}\sigma^2$$

  ($\rho = \mathbb{E}[(X_i-\mu)(X_j-\mu)]/\sigma^2$)
- As $n \to \infty$ the second term vanishes, but we are left with the first term
- If we want to do well we need our learning machines to be unbiased and decorrelated

Random Forest

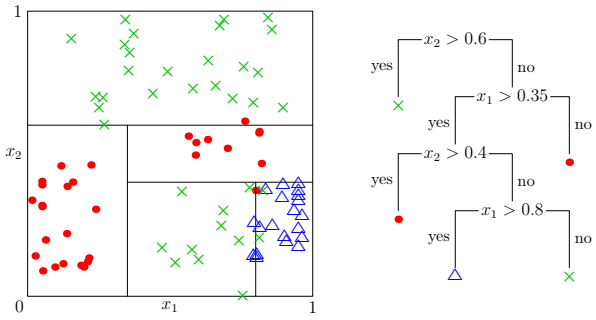
- In random forests we average much less correlated trees
- To do this for each tree we choose a subset of $p' \ll p$ of the features on which to split the tree
- Typically p' can range from 1 to $\sqrt{p}$
- The trees aren't that good, but are very decorrelated
- By averaging over a huge number of trees (order of 1000) we typically get good results
- Random Forest won (wins?) many competitions

Lessons

- Ensemble methods have proved themselves to be very powerful
- They work by averaging over different machines, trying to reduce their variance
- Here the variance comes from forcing the machines to learn different functions using Bootstrap Aggregation
- Tend to work best with very simple models (true of random forest and boosting)—seems to reduce over-fitting
- Random forest is very powerful, but gradient boosting is competitive

Advanced Machine Learning

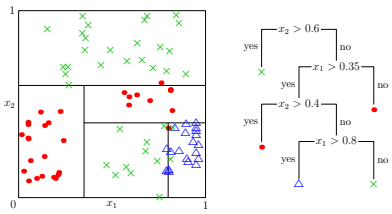
Boosting



Boosting, AdaBoost, Gradient Boosting

Outline

1. **Boosting**
2. AdaBoost
3. Gradient Boosting



Boosting

- In boosting we make a **strong learner** by using a weighted sum of **weak learners**

$$C_n(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i \hat{h}_i(\boldsymbol{x})$$

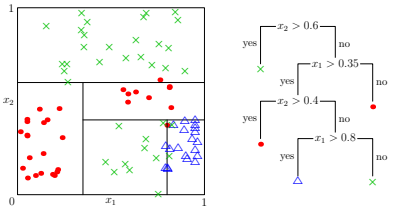
- Weak learners, $\hat{h}_i(\boldsymbol{x})$, are learning machine that do a little better than chance
- The trick is to choose the weights, α_i
- Because the weak learners do little better than chance we (miraculously) **don't** overfit that much

Shallow Trees

- One of the most effective type of weak learner are very shallow trees
- Sometimes we just use one variable (the stump) although usually we would use slightly deeper trees
- There are different algorithms for choosing the weights
  - ⋆ adaboost—a classic algorithm for binary classification
  - ⋆ gradient boosting—used for regression, trains a weak learner on the residual errors

Outline

1. Boosting
2. **AdaBoost**
3. Gradient Boosting



Boosting a Binary Classifier

- Suppose we have a binary classification task with data $\mathcal{D} = \{(\boldsymbol{x}^\mu, y^\mu) | \mu = 1, 2, \ldots, m\}$ with $y^\mu \in \{-1, 1\}$
- Our i^{th} weak learner provides a prediction $\hat{h}_i(\boldsymbol{x}^\mu) \in \{-1, 1\}$
- We ask, can we find a linear combination

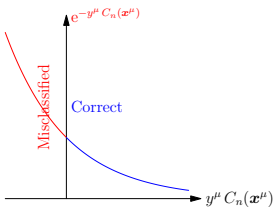
$$C_n(\boldsymbol{x}) = \alpha_1 \hat{h}_1(\boldsymbol{x}) + \alpha_2 \hat{h}_2(\boldsymbol{x}) + \cdots + \alpha_n \hat{h}_n(\boldsymbol{x})$$

- So that $\mathrm{sgn}\big(C_n(\boldsymbol{x})\big)$ is a strong learner?
- Note we want $y^\mu C_n(\boldsymbol{x}^\mu) > 0$

AdaBoost

- AdaBoost is a classic solution to this problem
- It assigns an "loss function"

$$L_n = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)}$$



- This punishes examples where there is an errors more than correct classifications

Iterative Learning

- We build up a strong learner iteratively (greedily)

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n \hat{h}_n(\boldsymbol{x})$$

- Defining $w_1^\mu = 1$ and $w_n^\mu = \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$ then

$$L_n(\alpha_n) = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu C_n(\boldsymbol{x}^\mu)} = \sum_{\mu=1}^{m} \mathrm{e}^{-y^\mu (C_{n-1}(\boldsymbol{x}^\mu) + \alpha_n \hat{h}_n(\boldsymbol{x}^\mu))}$$

$$= \sum_{\mu=1}^{m} w_n^\mu \mathrm{e}^{-\alpha_n y^\mu \hat{h}_n(\boldsymbol{x}^\mu)} = \mathrm{e}^{\alpha_n} \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu + \mathrm{e}^{-\alpha_n} \sum_{\mu: y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

$$= \mathrm{e}^{-\alpha_n} \sum_{\mu=1}^{m} w_n^\mu + (\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n}) \sum_{\mu: y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

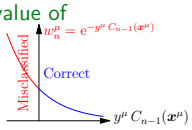
Choosing a Weak Classifier

- To minimise the loss

$$L_n(\alpha_n) = \mathrm{e}^{-\alpha_n}\sum_{\mu=1}^{m} w_n^\mu + (\mathrm{e}^{\alpha_n} - \mathrm{e}^{-\alpha_n})\sum_{\mu:y^\mu\neq\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$$

- We choose the weak learner with the lowest value of

$$\sum_{\mu:y^\mu\neq\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = \sum_{\mu:y^\mu\neq\hat{h}_n(\boldsymbol{x}^\mu)} \mathrm{e}^{-y^\mu C_{n-1}(\boldsymbol{x}^\mu)}$$



- That is, it misclassifies only where the other learners classify well

Choosing Weights

- We now choose the weight α_n to minimise the loss $L_n(\alpha_n)$

$$\frac{\partial L_n(\alpha_n)}{\partial \alpha_n} = \mathrm{e}^{\alpha_n}\sum_{\mu:y^\mu\neq\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu - \mathrm{e}^{-\alpha_n}\sum_{\mu:y^\mu=\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu = 0$$

- That is

$$\mathrm{e}^{2\alpha_n} = \frac{\sum_{\mu:y^\mu=\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu:y^\mu\neq\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \quad \text{or} \quad \alpha_n = \frac{1}{2}\log\left(\frac{\sum_{\mu:y^\mu=\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu:y^\mu\neq\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}\right)$$

Algorithm

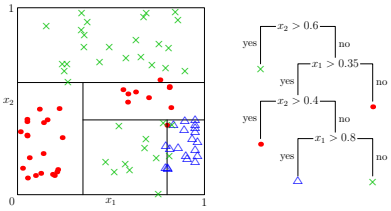
1. Start with a set of weak learners $\mathcal{W}$
2. Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1,2,\ldots,m$
3. Initially $w_1^\mu = 1$ (large weight, w_n^μ, means $(\boldsymbol{x}^\mu, y^\mu)$ is poorly classified)
4. Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\sum_{\mu:y^\mu\neq\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$
5. Update predictor $C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \alpha_n\hat{h}_n(\boldsymbol{x})$ where $\alpha_n = \frac{1}{2}\log\left(\frac{\sum_{\mu:y^\mu=\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu:y^\mu\neq\hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}\right)$
6. Update $w_{n+1}^\mu = w_n^\mu \mathrm{e}^{-y^\mu\alpha_n\hat{h}_n(\boldsymbol{x}^\mu)}$
7. Go to 4

Performance

- Adaboost works well with weak learners, usually out-performing bagging
- It doesn't work well with strong learners (tends to over-fit)
- It is limited to binary classification (there are generalisation, but they are difficult to get to work)
- It has fallen from fashion
- In contrast **gradient boosting** used for regression is very popular

Outline

1. Boosting
2. AdaBoost
3. **Gradient Boosting**



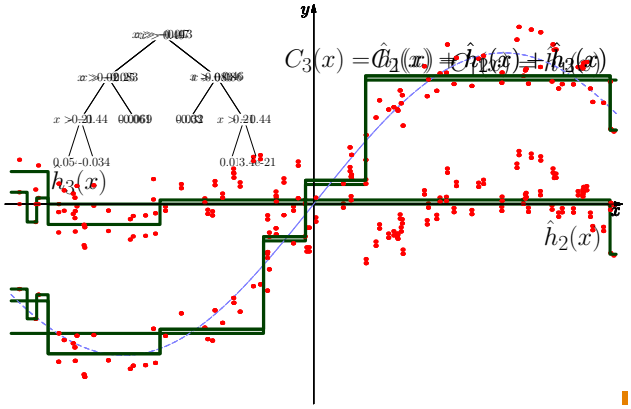
Gradient Boosting

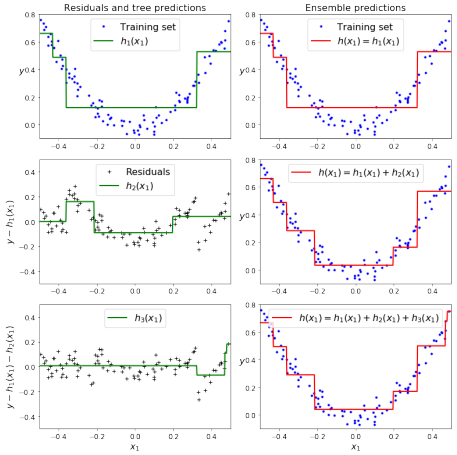
- In gradient boosting we again build a strong learner as a linear combination of weak learners

$$C_n(\boldsymbol{x}) = C_{n-1}(\boldsymbol{x}) + \hat{h}_n(\boldsymbol{x})$$

- Gradient boosting used on regression (again using decision trees)
- At each step $\hat{h}_n(\boldsymbol{x})$ is trained to predict the residual error, $\Delta_{n-1} = y - C_{n-1}(\boldsymbol{x})$, (i.e. the target minus the current prediction)
- (This difference looks a bit like a gradient hence the rather confusing name)

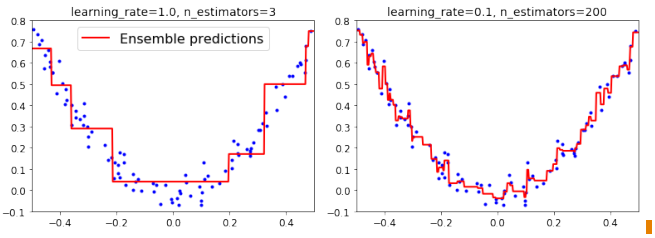
Fitting a Sin Wave





Keep On Going

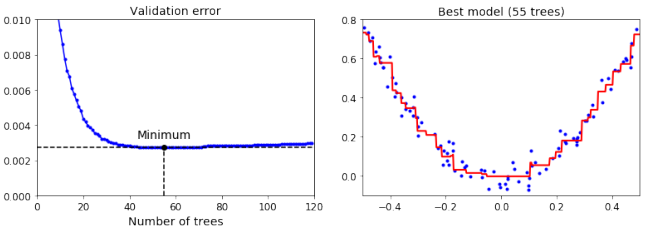
- We can keep on going



- But we will over-fit eventually

Early Stopping

- Like many algorithms we often get better results by early stopping



- Use cross-validation against a validation set to decide when to stop

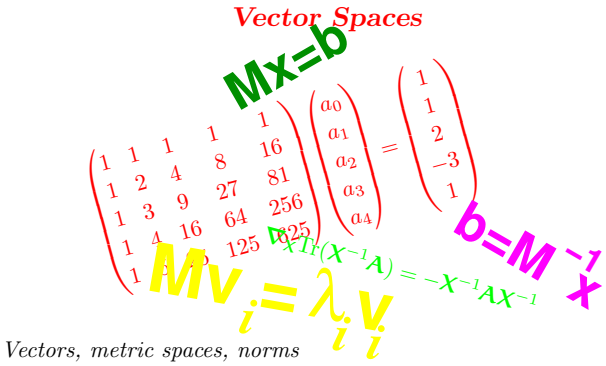
XGBoost

- XGBoost is an implementation of gradient boosting that won the Higg's Boson challenge and regularly wins Kaggle competitions
- XGBoost stands for eXtreme Gradient Boosting
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better
- It uses a cleverly chosen regularisation term to favour simple trees
- Finds a clever way to approximately minimise error plus regulariser very fast
- Rather a bodge of optimisation hacks

Conclusion

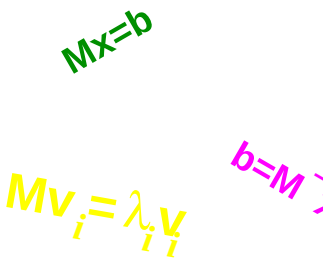
- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)—seems to reduce over-fitting
- XGBoost or GBM are currently the best methods for tabular data (particular for large training sets)—probably
- For images, signal and speech deep learning can give very significant advantage
- Probabilistic models can be better if you have a good model

Advanced Machine Learning



Vectors, metric spaces, norms

Outline



Matrices, Vectors and All That

- The language of machine learning is mathematics
- Sometimes we draw pretty pictures to explain the mathematics
- Much of the mathematics we will use involves vectors, matrices and functions
- You need to master the language of mathematics, otherwise you won't understand the algorithms
- I'm going to spend this lecture and the next revising the mathematics you need to know (but I'm going use a slightly posher language than you are probably used to)

Scalars (Fields)

- Vector spaces involve **fields** (numbers)—aka **scalars**
- These are quantities we can add together $(a+b)$ and multiply together $(a\times b)$
- Formally they form an Abelian group under addition with an identity 0 and excluding 0 an Abeilian group under multiplication and they are distributive

$$a\times(b+c)=a\times b+a\times c$$

- Although this sounds rather daunting don't panic. They behave like numbers. The field might be integers, rational numbers, reals, complex numbers or something a bit more exotic—but we will almost always consider reals

Vectors

- We often work with objects with many components (features)
- To help handle this we will use vector notation

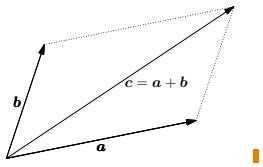
$$\boldsymbol{x}=\begin{pmatrix}x_1\\x_2\\\vdots\\x_n\end{pmatrix}$$

  - We represent vectors by bold symbols
  - All our vectors are column vectors by default
  - We treat them as $n\times 1$ matrix
- We write row vectors as transposes of column vectors

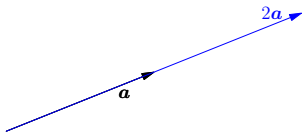
$$\boldsymbol{y}^{\mathsf{T}}=(y_1,y_2,\ldots,y_n)$$

Basic Vector Operations

- The basic vector operations are adding



- multiplying by a scalar (a number)



Vector Space

- A vector space, $\mathcal{V}$, is a set of vectors which satisfies

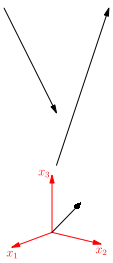
1. if $\boldsymbol{v},\boldsymbol{w}\in\mathcal{V}$ then $a\boldsymbol{v}\in\mathcal{V}$ and $\boldsymbol{v}+\boldsymbol{w}\in\mathcal{V}$ (closure)
2. $\boldsymbol{v}+\boldsymbol{w}=\boldsymbol{w}+\boldsymbol{v}$ (commutativity of addition)
3. $(\boldsymbol{u}+\boldsymbol{v})+\boldsymbol{w}=\boldsymbol{u}+(\boldsymbol{v}+\boldsymbol{w})$ (associativity of addition)
4. $\boldsymbol{v}+\boldsymbol{0}=\boldsymbol{v}$ (existence of additive identity $\boldsymbol{0}$)
5. $1\boldsymbol{v}=\boldsymbol{v}$ (existence of multiplicative identity 1)
6. $a(b\boldsymbol{v})=(ab)\boldsymbol{v}$ (distributive properties)
7. $a(\boldsymbol{v}+\boldsymbol{w})=a\boldsymbol{v}+a\boldsymbol{w}$
8. $(a+b)\boldsymbol{v}=a\boldsymbol{v}+b\boldsymbol{v}$

(You don't need to remember these)

- Just from these properties we can deduce other properties

$\mathbb{R}^n$

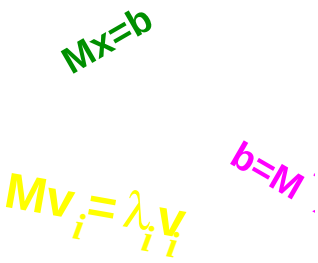
- When we first learn about vectors we think of them arrows in 3-D space
- If we centre them all at the origin then there is a one-to-one correspondence between vectors and points in space
- We call this vector space $\mathbb{R}^3$
- Any set of quantities $\boldsymbol{x}=(x_1,x_2,\ldots,x_n)^{\mathsf{T}}$ which satisfy the axioms above form a vector space $\mathbb{R}^n$
- Of course, we can't so easily draw pictures of high-dimensional vectors



Other Vector Spaces

- Vectors (i.e. $\mathbb{R}^n$) are not the only object that form vector spaces
- Matrices satisfy all the conditions of a vector space
- Infinite sequences form a vector space
- Functions form a vector space
  - Let $C(a,b)$ be the set of functions defined on the interval $[a,b]$
  - Note that if $f(x),g(x) \in C(a,b)$ then $af(x) \in C(a,b)$ and $f(x)+g(x) \in C(a,b)$
- Bounded vectors **don't** form a vector space

Outline



1. Vector Spaces
2. **Metrics (distances)**
3. Norms

Metrics

- Vector spaces become more interesting if we have a notion of distance
- We say $d(\boldsymbol{x},\boldsymbol{y})$ is a **proper distance** or **metric** if

1. $d(\boldsymbol{x},\boldsymbol{y}) \geq 0$ (non-negativity)
2. $d(\boldsymbol{x},\boldsymbol{y}) = 0$ iff $\boldsymbol{x} = \boldsymbol{y}$ (identity of indiscernibles)
3. $d(\boldsymbol{x},\boldsymbol{y}) = d(\boldsymbol{y},\boldsymbol{x})$ (symmetry)
4. $d(\boldsymbol{x},\boldsymbol{y}) \leq d(\boldsymbol{x},\boldsymbol{z}) + d(\boldsymbol{z},\boldsymbol{y})$ (triangular inequality)

- There are typically many possible distances (e.g. Euclidean distance, Manhattan distance, etc.)
- Often one or more condition isn't satisfied then we have a **pseudo-metric**

Mappings and Functions

- A function defines a mapping from one vector space to another (although the spaces might be the same), e.g.

$$f : \mathbb{R} \to \mathbb{R}$$

(f maps the reals onto reals, i.e. $f(x)$ takes a real x and gives you a new real number $y = f(x)$)

- We are often interested in functions that behave nicely
- E.g. They are continuous

Lipschitz Function

- One way to characterise well behaved function, $f(x)$ is if there exists a number $K < \infty$ such that for all x and y

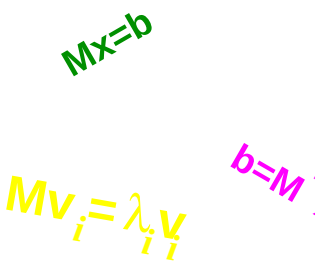
$$d(f(x),f(y)) \leq K d(x,y)$$

- This is known a **Lipschitz condition** and the function is said to be K-Lipschitz
- Note that such functions cannot have any jumps (i.e. they are continuous)
- The size of K measures the limit on the amplifying effect of the function

Contractive Mappings

- An interesting class of function are those for which $K < 1$
- These are said to be contractive mappings
- A famous theorem that applies to contractive mappings is the Banach fixed-point theorem which says there exists a unique fixed point such that $f(x) = x$
- This is used for example in showing that various algorithms will converge

Outline



1. Vector Spaces
2. Metrics (distances)
3. **Norms**

Norms

- Vector spaces are even more interesting with a notion of length
- **Norms** provide some measure of the size of a vector
- To formalise this we define the **norm** of an object $\boldsymbol{v}$ as $\|\boldsymbol{v}\|$ satisfying

1. $\|\boldsymbol{v}\| > 0$ if $\boldsymbol{v} \neq \boldsymbol{0}$ (non-negativity)
2. $\|a\boldsymbol{v}\| = a\|\boldsymbol{v}\|$ (linearity)
3. $\|\boldsymbol{u}+\boldsymbol{v}\| \leq \|\boldsymbol{u}\| + \|\boldsymbol{v}\|$ (triangular inequality)

- When some criteria aren't satisfied we have a **pseudo-norms**
- Norms provide a metric $d(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\|$ (they are metric spaces)

Vector Norms

- The familiar vector norm is the (Euclidean) two norm

$$\|\boldsymbol{v}\|_2 = \sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}$$

- Other norms exist, such as the p-norm ($p \geq 1$)

$$\|\boldsymbol{v}\|_p = \left(\sum_{i=1}^{n} |v_i|^p\right)^{1/p}$$

- Special cases include the 1-norm and the infinite norm

$$\|\boldsymbol{v}\|_1 = \sum_{i=1}^{n} |v_i| \qquad \|\boldsymbol{v}\|_\infty = \max_i |v_i|$$

- The 0-norm is a pseudo-norm as it does not satisfy condition 2

$$\|\boldsymbol{v}\|_0 = \text{number of non-zero components}$$

Matrix Norms

- We can define norms for other objects
- The norm of a matrix encodes how large the mapping is
- The Frobenius norm is defined by

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |A_{ij}|^2}$$

- Many other norms exist including 1-norm, max-norm, etc.
- For square matrices, some, but not all, norms satisfy the inequality

$$\|\mathbf{AB}\| \leq \|\mathbf{A}\| \times \|\mathbf{B}\|$$

Compatible Norms

- A vector and matrix norm are said to be compatible if

$$\|\mathbf{M}\boldsymbol{v}\|_b \leq \|\mathbf{M}\|_a \times \|\boldsymbol{v}\|_b$$

(Spectral and Euclidean norms are compatible)

- Norms provide quick ways to bound the maximum growth of a vector under a mapping induced by the matrix
- We will see that a measure of the sensitivity of a mapping is in terms of the ratio of its maximum effect to its minimum effect on a vector
- This is known as the **conditioning**, given by $\|\mathbf{M}\| \times \|\mathbf{M}^{-1}\|$

Why Should You Care?

- Deep learning involves multiply the input (which we can think of as a vector $\boldsymbol{x}$) by many layers
- In CNNs we have convolutional layers and dense layers
- The effect of applying these layers can be represented by a matrix multiplication $\boldsymbol{x}_n = \mathbf{L}_n\boldsymbol{x}_{n-1}$
- We also do other things like applying ReLU's or pooling that changes the magnitude, $\boldsymbol{x}_n$, of our representation
- If you are developing new architectures you want $\|\boldsymbol{x}_n\|$ neither to blow up or vanish
- This can be controlled by carefully choosing $\|\mathbf{L}_n\|$

Function Norms

- Functions can also have norms, for example, if $f(x)$ is defined in some interval $\mathcal{I}$

$$\|f\|_{L_2} = \sqrt{\int_{x\in\mathcal{I}} f^2(x)\mathrm{d}x}$$

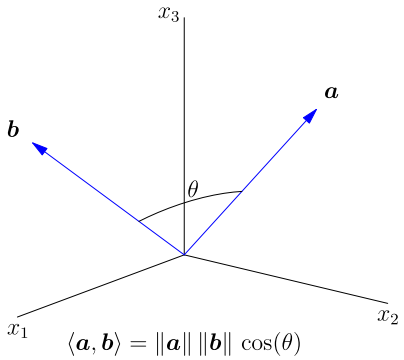
- The L_2 vector space is the set of function where $\|f\|_{L_2} < \infty$
- The L_1-norm is given by $\|f\|_{L_1} = \int_{x\in\mathcal{I}} |f(x)|\mathrm{d}x$
- The infinite-norm is given by $\|f\|_\infty = \max_{x\in\mathcal{I}} |f(x)|$

Summary

- Vector spaces with a distance (metric spaces) and vector spaces with a norm (normed vector spaces) are interesting objects
- They allow you to define a topology (open/closed sets, etc.)
- You can build up ideas about connectedness, continuity, contractive maps, fixed-point theorems, . . .
- For the most part we are going to consider an even more powerful vector space that has an inner-product defined

Advanced Machine Learning

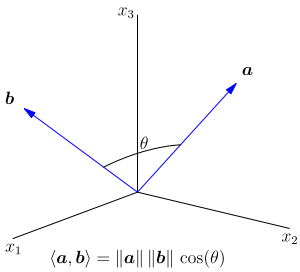
Inner Product Spaces



$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \|\boldsymbol{a}\| \, \|\boldsymbol{b}\| \cos(\theta)$$

Inner products, operators

Outline



$$\langle \boldsymbol{a}, \boldsymbol{b} \rangle = \|\boldsymbol{a}\| \, \|\boldsymbol{b}\| \cos(\theta)$$

Recap

- We have looked at vector spaces (closed sets where we can add elements and multiply them by a scalar)
- Recall that vector spaces don't just apply to normal vectors ($\mathbb{R}^n$), but to matrices, functions, sequences, random variables, . . .
- Proper distances or metrics, $d(\boldsymbol{x},\boldsymbol{y})$, allow us to construct ideas about geometry of the vector space
- Norms, $\|\boldsymbol{x}\|$, that allow us to reason about the size of vector
- Norm induce a distance, $d(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x}-\boldsymbol{y}\|$

Inner Products

- We will often consider objects with an *inner product*
- For vectors in $\mathbb{R}^n$

$$\langle \boldsymbol{x},\boldsymbol{y} \rangle = \boldsymbol{x}^\mathsf{T}\boldsymbol{y} = \sum_{i=1}^n x_i y_i$$

- For functions

$$\langle f,g \rangle = \int_{x\in\mathcal{I}} f(x)g(x)\mathrm{d}x$$

- For $m \times n$ matrices

$$\langle \mathbf{A},\mathbf{B} \rangle = \mathrm{Tr}\boldsymbol{A}^\mathsf{T}\mathbf{B} = \sum_{i=1}^m\sum_{j=1}^n A_{ij}B_{ij}$$

Axioms of Inner Products

- An inner product satisfies
  1. $\langle \boldsymbol{x},\boldsymbol{x} \rangle \geq 0$ for all $\boldsymbol{x} \in \mathcal{V}$
  2. $\langle \boldsymbol{x},\boldsymbol{x} \rangle = 0$ if and only if $\boldsymbol{x} = \boldsymbol{0}$
  3. $\langle \alpha\boldsymbol{x},\boldsymbol{y} \rangle = \alpha \langle \boldsymbol{x},\boldsymbol{y} \rangle$
  4. $\langle \boldsymbol{x},\boldsymbol{y}+\boldsymbol{z} \rangle = \langle \boldsymbol{x},\boldsymbol{y} \rangle + \langle \boldsymbol{x},\boldsymbol{z} \rangle$
  5. $\langle \boldsymbol{x},\boldsymbol{y} \rangle = \langle \boldsymbol{y},\boldsymbol{x} \rangle$
- We can show that $\|\boldsymbol{x}\| = \sqrt{\langle \boldsymbol{x},\boldsymbol{x} \rangle}$ satisfies the axioms of a norm, so that an inner-product space is a normed space
- The norm associated with the inner-product for vectors in $\mathbb{R}^n$ (i.e. $\langle \boldsymbol{x},\boldsymbol{y} \rangle = \boldsymbol{x}^\mathsf{T}\boldsymbol{y}$) is the Euclidean norm $\|\boldsymbol{x}\| = \sqrt{\boldsymbol{x}^\mathsf{T}\boldsymbol{x}}$

Cauchy-Schwarz Inequality

- One of the most important results of inner-product spaces, known as the **Cauchy-Schwarz inequality** is that

$$\langle \boldsymbol{x},\boldsymbol{y} \rangle^2 \leq \langle \boldsymbol{x},\boldsymbol{x} \rangle \langle \boldsymbol{y},\boldsymbol{y} \rangle = \|\boldsymbol{x}\|^2\|\boldsymbol{y}\|^2$$

- Or

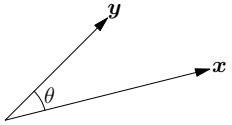
$$|\langle \boldsymbol{x},\boldsymbol{y} \rangle| \leq \|\boldsymbol{x}\| \, \|\boldsymbol{y}\|$$

- This is a very general result so for example

$$\left| \int f(x)g(x)\mathrm{d}x \right| \leq \sqrt{\left(\int f^2(x)\mathrm{d}x \right)\left(\int g^2(x)\mathrm{d}x \right)}$$

Angles Between Vectors

- A natural interpretation of the inner product is in providing a measure of the angle between vectors



$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x}^\mathsf{T}\boldsymbol{y} = \|\boldsymbol{x}\| \, \|\boldsymbol{y}\| \cos(\theta)$$

- Vectors are orthogonal if $\langle \boldsymbol{x},\boldsymbol{y} \rangle = 0$
- We can extend this idea to functions

$$\langle f(x),g(x) \rangle = \int_{x\in\mathcal{I}} f(x)g(x)\mathrm{d}x = \|f(x)\| \, \|g(x)\| \cos(\theta)$$

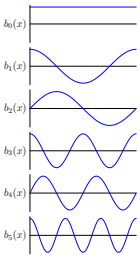
- Note that $\sin(x)$ and $\cos(x)$ are orthogonal in the interval $[0,2\pi]$

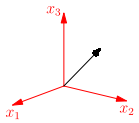
Basis Functions

- Any set of vectors $\{\boldsymbol{b}_i | i = 1,\ldots\}$ that span the space can be used as a basis or coordinate system
- The simplest and most useful case is when the vectors are orthogonal and normalised (i.e. $\|\boldsymbol{b}_i\| = 1$)
- In $\mathbb{R}^3$ we could use $\boldsymbol{b}_1 = \begin{pmatrix}1\\0\\0\end{pmatrix}$, $\boldsymbol{b}_2 = \begin{pmatrix}0\\1\\0\end{pmatrix}$, $\boldsymbol{b}_3 = \begin{pmatrix}0\\0\\1\end{pmatrix}$
- This is not unique as we can rotate our basis vectors
- For an orthogonal basis we can write any vector as $\hat{\boldsymbol{x}} = \begin{pmatrix}\boldsymbol{x}^\mathsf{T}\boldsymbol{b}_1\\ \boldsymbol{x}^\mathsf{T}\boldsymbol{b}_2\\ \boldsymbol{x}^\mathsf{T}\boldsymbol{b}_3\end{pmatrix}$

Orthogonal Functions

- For functions we can use any ortho-normal set of functions as a basis
- The most familiar are the Fourier functions $\sin(n\theta)$ and $\cos(n\theta)$
- Any function in $C(0,2\pi)$ can be represented by a point $\boldsymbol{f} = \begin{pmatrix} \langle f(x), b_0(x)\rangle \\ \langle f(x), b_1(x)\rangle \\ \vdots \end{pmatrix}$
- There might be an infinite number of components
- This is analogous to points in $\mathbb{R}^n$ (for large n)



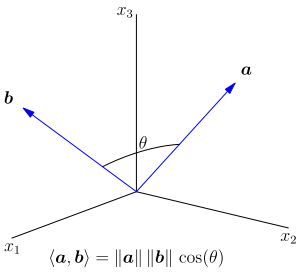


Algebraic Structure

- We have gone to these lengths as we want to show that many properties of vectors are shared by other objects (matrices, functions, etc.)
- The notions of distance (geometry), norms (size of vectors) and inner products (angles between vectors) provides a very rich set of concepts
- Vectors form the backbone of objects we will use repeated in machine learning
- The next piece of the jigsaw is to understand how we can transform these objects

Outline

1. Inner Products
2. Operators



Operators

- In machine learning we are interested in transforming our input vectors into some output predictions
- To accomplish this we will apply some mapping or operators on the vector $\mathcal{T} : \mathcal{V} \to \mathcal{V}'$
- This says that $\mathcal{T}$ maps some object $\boldsymbol{x} \in \mathcal{V}$ to an object $\boldsymbol{y} = \mathcal{T}[\boldsymbol{x}]$ in a new vector space $\mathcal{V}'$
- This new vector space may or may not be the same as the original vector space
- Our objects may be any object in a vector space such as a function

Linear Operators

- Operators are in general very complicated, but a particular nice set of operators are linear operators
- $\mathcal{T}$ is a linear operator if
  1. $\mathcal{T}[a\boldsymbol{x}] = a\mathcal{T}[\boldsymbol{x}]$
  2. $\mathcal{T}[\boldsymbol{x}+\boldsymbol{y}] = \mathcal{T}[\boldsymbol{x}] + \mathcal{T}[\boldsymbol{y}]$
- For normal vectors ($\boldsymbol{x} \in \mathbb{R}^n$) the most general linear operation is

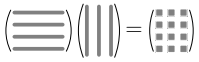
$$\mathcal{T}[\boldsymbol{x}] = \mathbf{M}\boldsymbol{x}$$

where $\mathbf{M}$ is a matrix

Matrix multiplication

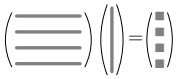
- For an $\ell \times m$ matrix $\mathbf{A}$ and an $m \times n$ matrix $\mathbf{B}$ we can compute the $(\ell \times n)$ product, $\mathbf{C} = \mathbf{AB}$, such that

$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj}$$



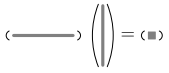
- Treating the vector $\boldsymbol{x}$ as a $n \times 1$ matrix then

$$\boldsymbol{y} = \mathbf{A}\boldsymbol{x} \quad \Rightarrow \quad y_i = \sum_j M_{ij} x_j$$



- Using the same matrix notation we can define the inner product as

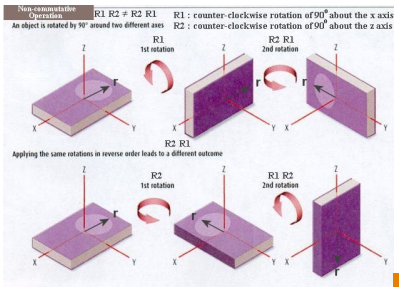
$$\langle \boldsymbol{x}, \boldsymbol{y}\rangle = \boldsymbol{x}^{\mathsf{T}}\boldsymbol{y} = \sum_{i=1}^{n} x_i y_i$$



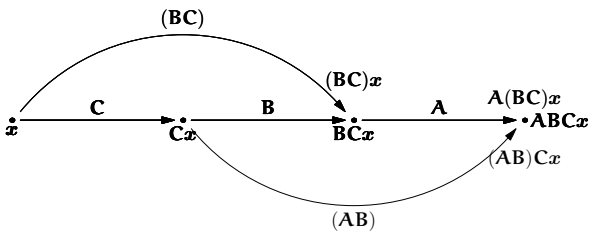
Non-commutativity

- In general $\mathbf{AB} \neq \mathbf{BA}$

$$\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \quad \begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix}$$



Associativity of Mappings



- For all $\boldsymbol{x}$ we have $\mathbf{A}(\mathbf{BC})\boldsymbol{x} = (\mathbf{AB})\mathbf{C}\boldsymbol{x}$
- This implies $\mathbf{A}(\mathbf{BC}) = (\mathbf{AB})\mathbf{C}$

Kernels

- The equivalent of a matrix for functions (i.e. a linear operator) is known as a kernel $K(x,y)$

$$g(x) = \mathcal{T}[f] = \int_{y \in \mathcal{I}} K(x,y) f(y) \mathrm{d}y$$

- Our domain does not need to be one dimensional, e.g.

$$g(\boldsymbol{x}) = \mathcal{T}[f] = \int_{\boldsymbol{y} \in \mathcal{I}} K(\boldsymbol{x},\boldsymbol{y}) f(\boldsymbol{y}) \mathrm{d}\boldsymbol{y}$$

- We shall soon see examples of high-dimensional kernels

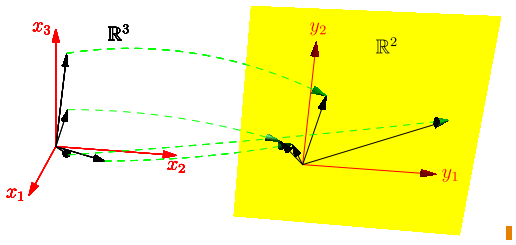
Kernels in Machine Learning

- Kernels are used heavily in machine learning
- In kernel methods such as SVM, SVR, Kernel-PCA
- They are also used in Gaussian Processes
- In all these cases we consider symmetric, positive semi-definite kernels
- Sometimes they can be interpreted as covariance between random functions

$$K(\boldsymbol{x},\boldsymbol{y}) = \mathbb{E}_{f \sim \mathcal{P}}\big[\big(f(\boldsymbol{x}) - \mu(\boldsymbol{x})\big)\big(f(\boldsymbol{y}) - \mu(\boldsymbol{y})\big)\big]$$

General Linear Mappings

- In general a linear operator will map vectors between different vector spaces
- E.g. $\mathbb{R}^3 \to \mathbb{R}^2$



Square Matrices

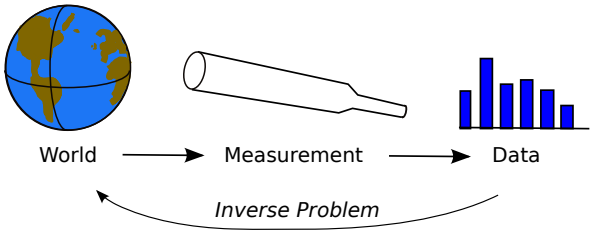
- We will spend a lot of time on operators that map from a vector space onto itself $\mathcal{T} : \mathcal{V} \to \mathcal{V}$
- For vectors in $\mathbb{R}^n$ such linear operators are represented by square matrices
- When there is a one-to-one mapping then we have a unique inverse
- We will study such mappings in detail in the next lecture

Summary

- We haven't covered much machine learning as such—sorry
- But mathematics is the language of machine learning and you have to get used to it
- Mathematics is like programming, if you don't understand the syntax and you can't write it down then its meaningless
- We've taken a high level view of inner product spaces and operator, this will pay us back later as we look at kernel methods

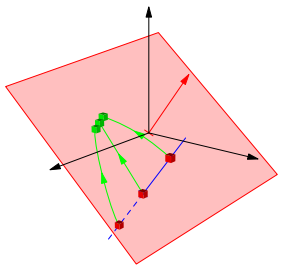
Advanced Machine Learning

Understand Mappings



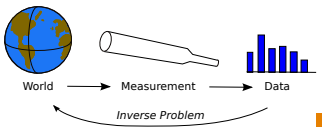
Mappings, Linear Maps, Solving Linear Systems

Outline



Transforming Data

- In the last lecture we spent time developing a sophisticate view of vector spaces and operators
- At a mathematical level machine learning can be viewed as performing an inverse mapping



- Although our mappings are not necessarily linear in either direction we learn a lot by understanding linear operators

Inverse Problems

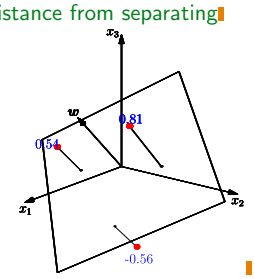
- Given m observations $\{(\boldsymbol{x}_k,y_k)|k=1,\ldots,m\}$ and p unknown $\boldsymbol{w}=(w_1,w_2,\ldots w_p)$ such that $\boldsymbol{x}_k^\mathsf{T}\boldsymbol{w}=y_k$ then to find $\boldsymbol{w}$
- Define the *design matrix* as the matrix of feature vectors

$$\mathbf{X}=\begin{pmatrix}\boldsymbol{x}_1^\mathsf{T}\\ \boldsymbol{x}_2^\mathsf{T}\\ \ldots\\ \boldsymbol{x}_m^\mathsf{T}\end{pmatrix}=\begin{pmatrix}x_{11}&x_{12}&\cdots&x_{1p}\\ x_{21}&x_{22}&\cdots&x_{2p}\\ \vdots&\vdots&\ddots&\vdots\\ x_{m1}&x_{m2}&\cdots&x_{mp}\end{pmatrix}$$

- and the target vector $\boldsymbol{y}=(y_1,y_2,\cdots,y_m)^\mathsf{T}$
- Then if $m=p$ we have $\boldsymbol{y}=\mathbf{X}\boldsymbol{w}$ or $\boldsymbol{w}=\mathbf{X}^{-1}\boldsymbol{y}$

Linear Regression

- $\boldsymbol{x}_k^\mathsf{T}\boldsymbol{w}$ depends on distance from separating



- If $m>p$ then $\mathbf{X}$ isn't square so doesn't have an inverse
- Worse unless the data is accurate $\boldsymbol{y}\approx\mathbf{X}\boldsymbol{w}\Rightarrow$ no "solution"
- Problem solved by Gauss to predict the orbit of the asteroid Ceres

Linear Least Squares

- The error of input pattern $\boldsymbol{x}_k$ is

$$\epsilon_k=\boldsymbol{x}_k^\mathsf{T}\boldsymbol{w}-y_k$$

- The squared error

$$E(\boldsymbol{w}|\mathcal{D})=\sum_{k=1}^m\left(\boldsymbol{x}_k^\mathsf{T}\boldsymbol{w}-y_k\right)^2=\sum_{k=1}^m\epsilon_k^2=\|\boldsymbol{\epsilon}\|^2$$

- We can define the error vector

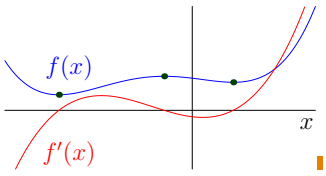
$$\boldsymbol{\epsilon}=\mathbf{X}\boldsymbol{w}-\boldsymbol{y}$$

(note that $\epsilon_k=\boldsymbol{x}_k^\mathsf{T}\boldsymbol{w}-y_k$)

- Minimising this error is known as the least squares problem

Finding a Minimum

- The minima of a one dimensional function, $f(x)$, are given by $f'(x)=0$



- The minima of an n-dimensions function $f(\boldsymbol{x})$ are given by the set of equations

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i}=0\quad\forall i=1,\ldots n$$

Gradients

- The **grad** operator $\boldsymbol{\nabla}$ is the gradient operator in high dimensions

$$\boldsymbol{\nabla}f(\boldsymbol{x})=\begin{pmatrix}\frac{\partial f(\boldsymbol{x})}{\partial x_1}\\ \frac{\partial f(\boldsymbol{x})}{\partial x_2}\\ \vdots\\ \frac{\partial f(\boldsymbol{x})}{\partial x_n}\end{pmatrix}$$

- The partial derivatives (curly d's)

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i}$$

means differentiate with respect to x_i treating all other components x_j as constants

Least Squares Solution

- The least squared solution is give by

$$\begin{aligned}\nabla E(\boldsymbol{w}|\mathcal{D}) &= \nabla\|\boldsymbol{\epsilon}\|^2 = \nabla\|\mathbf{X}\boldsymbol{w}-\boldsymbol{y}\|^2 \\ &= \nabla\left(\boldsymbol{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{w} - 2\boldsymbol{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\boldsymbol{y} + \boldsymbol{y}^\mathsf{T}\boldsymbol{y}\right) \\ &= 2\left(\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{w} - \mathbf{X}^\mathsf{T}\boldsymbol{y}\right) = 0\end{aligned}$$

- Or

$$\boldsymbol{w} = \left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)^{-1}\mathbf{X}^\mathsf{T}\boldsymbol{y} = \mathbf{X}^{+}\boldsymbol{y}$$

- $\mathbf{X}^{+} = \left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)^{-1}\mathbf{X}^\mathsf{T}$ is known as the pseudo inverse
- For non-square matrices Matlab uses the pseudo inverse so in Matlab we can write

```
w = X\y
```


Missing Bits of the Mathematics

- Note that $\|\boldsymbol{a}\|^2 = \boldsymbol{a}^\mathsf{T}\boldsymbol{a} = \sum_i a_i^2$

$$\begin{aligned}\|\mathbf{X}\boldsymbol{w}-\boldsymbol{y}\|^2 &= (\mathbf{X}\boldsymbol{w}-\boldsymbol{y})^\mathsf{T}(\mathbf{X}\boldsymbol{w}-\boldsymbol{y}) = (\boldsymbol{w}^\mathsf{T}\mathbf{X}^\mathsf{T}-\boldsymbol{y}^\mathsf{T})(\mathbf{X}\boldsymbol{w}-\boldsymbol{y}) \\ &= \boldsymbol{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{w} - 2\boldsymbol{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\boldsymbol{y} + \boldsymbol{y}^\mathsf{T}\boldsymbol{y}\end{aligned}$$

- Where we have used $\boldsymbol{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\boldsymbol{y} = \boldsymbol{y}^\mathsf{T}\mathbf{X}\boldsymbol{w}$, $\sum_{i,j} w_i X_{ji} y_j = \sum_{i,j} y_i X_{ij} w_j$
- Also $\nabla \boldsymbol{w}^\mathsf{T}\mathbf{M}\boldsymbol{w} = \mathbf{M}\boldsymbol{w} + \mathbf{M}^\mathsf{T}\boldsymbol{w}$
- If $\mathbf{M} = \mathbf{M}^\mathsf{T}$ (i.e. $\mathbf{M}$ is symmetric) then $\nabla \boldsymbol{w}^\mathsf{T}\mathbf{M}\boldsymbol{w} = 2\mathbf{M}\boldsymbol{w}$
- $(\mathbf{X}^\mathsf{T}\mathbf{X})^\mathsf{T} = \mathbf{X}^\mathsf{T}\mathbf{X}$ so that $\mathbf{X}^\mathsf{T}\mathbf{X}$ is symmetric

Computing Gradients

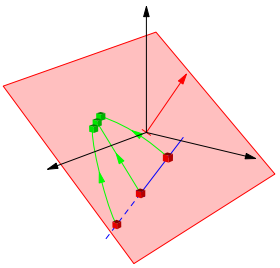
- To understand gradients we sometimes need to go back to components

$$\nabla \boldsymbol{w}^\mathsf{T}\mathbf{M}\boldsymbol{w} = \begin{pmatrix}\frac{\partial}{\partial w_1}\\ \frac{\partial}{\partial w_2}\\ \frac{\partial}{\partial w_3}\\ \vdots\end{pmatrix}\sum_{i,j} w_i M_{ij} w_j = \begin{pmatrix}\sum_j M_{1j}w_j + \sum_i w_i M_{i1}\\ \sum_j M_{2j}w_j + \sum_i w_i M_{i2}\\ \sum_j M_{3j}w_j + \sum_i w_i M_{i3}\\ \vdots\end{pmatrix}$$
$$= \mathbf{M}\boldsymbol{w} + \mathbf{M}^\mathsf{T}\boldsymbol{w}$$

- It is tedious to compute these things component-wise, but when you need to understand what is going on then go back to the basics

Outline

1. Mappings
2. Linear Maps

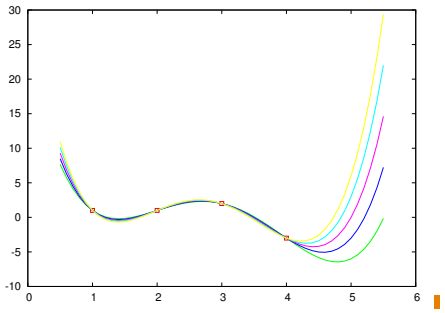


Solving Inverse Problems

- Gauss showed us how to solve **over-constrained** problems (we have more observations than parameters)
- We seek a solution which isn't necessarily exact but minimises an error
- But, what if we have more parameters than observations
- That is, we are **under-constrained**
- Note that in some directions you might be over-constrained and in other directions under-constrained
- This is very typical of most machine learning problems

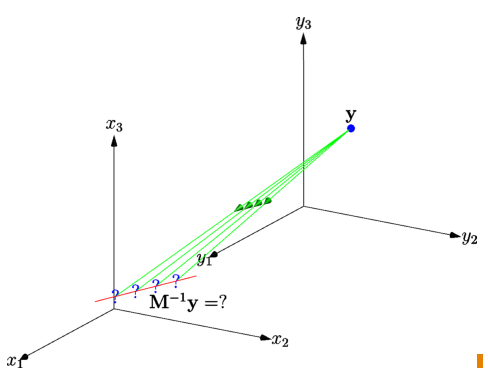
Under Constrained Systems

- If we have less data-points than parameters then there will be multiple solutions



What is the Inverse?

- Many points can map to the same points



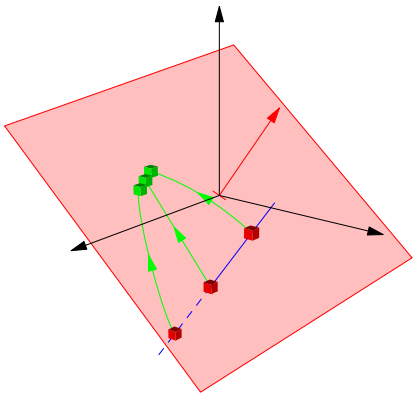
Under-constrained Systems

- The system is **under-constrained**
- We have more unknowns than equations
- The inverse is not unique
- Solving the inverse problem ($\boldsymbol{w} = \left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)^{-1}\mathbf{X}^\mathsf{T}\boldsymbol{y}$) is said to be **ill-posed**
- The inverse $\left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)^{-1}$ doesn't exist
- If we have a complicated learning machine and not sufficient data we often end with an ill-posed inverse problem (there are lots of sets of parameters that explain the data)

Ill-Conditions

- Singular matrices are rare (although they occur when we don't have enough data), but matrices that are close to being singular are common
- If a matrix is close to singular it is ill-conditioned
- Ill-conditioned matrices have some small eigenvalues
- All points get contracted towards a plane
- Large matrices are very often ill conditioned

Ill-Conditioned Matrices



Ill-Conditioning in ML

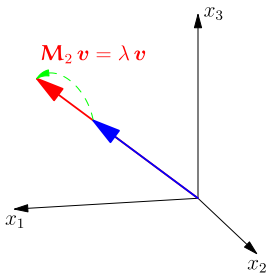
- Ill-conditioning in machine learning occurs when a very small change in the learning data causes a large change in the predictions of the learning machine
- In linear regression the matrix $\mathbf{X}^\mathsf{T}\mathbf{X}$ is ill-conditioned when we have as many data points as parameters
- Much of machine learning is concerned with making learning machines better conditioned
- Adding regularisers is one approach to achieve this

Summary

- Linear mappings are commonly used in machine learning algorithms such as regression
- We will often meet the pseudo-inverse involving inverting $\mathbf{X}^\mathsf{T}\mathbf{X}$
- They can be inherently unstable to noise in the inputs

Advanced Machine Learning

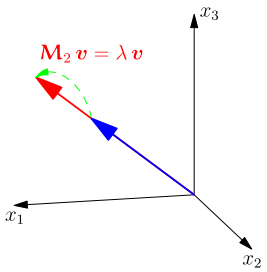
Eigensystems



Eigenvectors, Orthogonal Matrices, Eigenvector Decomposition, Rank

Outline

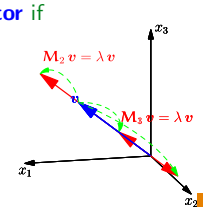
1. **Eigenvectors**
2. Orthogonal Matrices
3. Eigen Decomposition
4. Low Rank Approximation



Eigenvector equation

- Eigen-systems help us to understand mappings
- A vector $\boldsymbol{v}$ is said to be an **eigenvector** if

$$\mathbf{M}\boldsymbol{v} = \lambda\boldsymbol{v}$$



- $\mathbf{M}$ is square (i.e. $n\times n$)
- Where the number λ is the **eigenvalue**
- Eigenvalues play a fundamental role in understanding operators

Symmetric Matrices

- If $\mathbf{M}$ is an $n\times n$ **symmetric** matrix then it has n real orthogonal eigenvectors with real eigenvalues
- We denote the i^{th} eigenvector by $\boldsymbol{v}_i$ and the corresponding eigenvalue by λ_i so that

$$\mathbf{M}\boldsymbol{v}_i = \lambda_i\boldsymbol{v}_i$$

- Orthogonal means that if $i \neq j$ then

$$\boldsymbol{v}_i^\mathsf{T}\boldsymbol{v}_j = 0$$

- (We can always normalise eigenvectors if we want)

Proof of Orthogonality

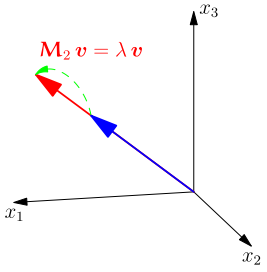
- $\left(\mathbf{M}\boldsymbol{v}_i = \lambda_i\boldsymbol{v}_i\right)^\mathsf{T}$ implies $\boldsymbol{v}_i^\mathsf{T}\mathbf{M}^\mathsf{T} = \lambda_i\boldsymbol{v}_i^\mathsf{T}$
- When $\mathbf{M}$ is symmetric then $\mathbf{M}\boldsymbol{v}_i = \lambda_i\boldsymbol{v}_i \Rightarrow \boldsymbol{v}_i^\mathsf{T}\mathbf{M} = \lambda_i\boldsymbol{v}_i^\mathsf{T}$
- Consider two eigenvectors $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ of $\mathbf{M}$

$$\begin{aligned}\boldsymbol{v}_i^\mathsf{T}\mathbf{M}\boldsymbol{v}_j &= (\boldsymbol{v}_i^\mathsf{T}\mathbf{M})\boldsymbol{v}_j = \lambda_i\boldsymbol{v}_i^\mathsf{T}\boldsymbol{v}_j\\ &= \boldsymbol{v}_i^\mathsf{T}(\mathbf{M}\boldsymbol{v}_j) = \lambda_j\boldsymbol{v}_i^\mathsf{T}\boldsymbol{v}_j\end{aligned}$$

- So either $\lambda_i = \lambda_j$ or $\boldsymbol{v}_i^\mathsf{T}\boldsymbol{v}_j = 0$
- If $\lambda_i = \lambda_j$ then any linear combination of $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ is an eigenvector ($\mathbf{M}(a\boldsymbol{v}_i + b\boldsymbol{v}_j) = \lambda_i(a\boldsymbol{v}_i + b\boldsymbol{v}_j)$). So I can choose two eigenvectors that are orthogonal to each other.

Outline

1. Eigenvectors
2. **Orthogonal Matrices**
3. Eigen Decomposition
4. Low Rank Approximation



Orthogonal Matrices

- We can construct an **orthogonal** matrix $\mathbf{V}$ from the eigenvectors

$$\mathbf{V} = (\boldsymbol{v}_1, \boldsymbol{v}_2, \cdots, \boldsymbol{v}_n)$$

- Matrix $\mathbf{V}$ is an $n\times n$ matrix
- Because of the orthogonality of the vectors $\boldsymbol{v}_i$

$$\mathbf{V}^\mathsf{T}\mathbf{V} = \begin{pmatrix}\boldsymbol{v}_1^\mathsf{T}\boldsymbol{v}_1 & \boldsymbol{v}_1^\mathsf{T}\boldsymbol{v}_2 & \cdots & \boldsymbol{v}_1^\mathsf{T}\boldsymbol{v}_n\\ \boldsymbol{v}_2^\mathsf{T}\boldsymbol{v}_1 & \boldsymbol{v}_2^\mathsf{T}\boldsymbol{v}_2 & \cdots & \boldsymbol{v}_2^\mathsf{T}\boldsymbol{v}_n\\ \vdots & \vdots & \ddots & \vdots\\ \boldsymbol{v}_n^\mathsf{T}\boldsymbol{v}_1 & \boldsymbol{v}_n^\mathsf{T}\boldsymbol{v}_2 & \cdots & \boldsymbol{v}_n^\mathsf{T}\boldsymbol{v}_n\end{pmatrix} = \begin{pmatrix}1 & 0 & \cdots & 0\\ 0 & 1 & \cdots & 0\\ \vdots & \vdots & \ddots & \vdots\\ 0 & 0 & \cdots & 1\end{pmatrix} = \mathbf{I}$$

The Other Way Around

- We have shown that $\mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{I}$
- Thus multiply both sides on the left by $\mathbf{V}$

$$\mathbf{V}\mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{V}$$

- $\mathbf{V}$ will have an inverse, $\mathbf{V}^{-1}$, such that $\mathbf{V}\mathbf{V}^{-1} = \mathbf{I}$
- Multiplying the equation on the right by $\mathbf{V}^{-1}$

$$\begin{aligned}(\mathbf{V}\mathbf{V}^\mathsf{T})\mathbf{V}\mathbf{V}^{-1} &= \mathbf{V}\mathbf{V}^{-1}\\ \mathbf{V}\mathbf{V}^\mathsf{T} &= \mathbf{I}\end{aligned}$$

- Note that, $\mathbf{V}^{-1} = \mathbf{V}^\mathsf{T}$ (definition of orthogonal matrix)

Invertible Matrices

- A matrix, $\mathbf{M}$, will be singular (uninvertible) if there exists a vector $\boldsymbol{x}$ ($\neq \mathbf{0}$) such that

$$\mathbf{M}\boldsymbol{x} = \mathbf{0}$$

- Now if there exists such a vector such that $\mathbf{V}\boldsymbol{x} = \mathbf{0}$ then multiply by $\mathbf{V}^\mathsf{T}$ we get

$$\mathbf{V}^\mathsf{T}\mathbf{V}\boldsymbol{x} = \mathbf{V}^\mathsf{T}\mathbf{0}$$
$$\boldsymbol{x} = \mathbf{0}$$

since $\mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{I}$

- Thus $\boldsymbol{V}$ is invertible

Rotations

- Orthogonal matrices satisfy $\mathbf{V}^\mathsf{T}\mathbf{V} = \mathbf{V}\mathbf{V}^\mathsf{T} = \mathbf{I}$
- As a consequent they define rotations (and possibly a reflection)
- Consider a vector $\boldsymbol{x}$ and $\boldsymbol{x}' = \boldsymbol{V}\boldsymbol{x}$, now

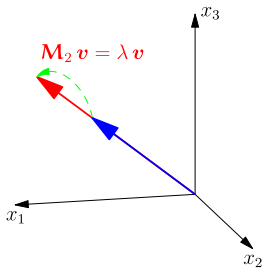
$$\|\boldsymbol{x}'\|_2^2 = \boldsymbol{x}'^\mathsf{T}\boldsymbol{x}' = (\boldsymbol{V}\boldsymbol{x})^\mathsf{T}(\boldsymbol{V}\boldsymbol{x}) = \boldsymbol{x}^\mathsf{T}\mathbf{V}^\mathsf{T}\mathbf{V}\boldsymbol{x} = \boldsymbol{x}^\mathsf{T}\boldsymbol{x} = \|\boldsymbol{x}\|_2^2$$

- Similarly if additionally $\boldsymbol{y}' = \boldsymbol{V}\boldsymbol{y}$ then

$$\langle \boldsymbol{x}', \boldsymbol{y}' \rangle = (\boldsymbol{V}\boldsymbol{x})^\mathsf{T}(\boldsymbol{V}\boldsymbol{y}) = \boldsymbol{x}^\mathsf{T}\mathbf{V}^\mathsf{T}\mathbf{V}\boldsymbol{y} = \boldsymbol{x}^\mathsf{T}\boldsymbol{y} = \langle \boldsymbol{x}, \boldsymbol{y} \rangle = \|\boldsymbol{x}\|_2 \|\boldsymbol{y}\|_2 \cos(\theta)$$

- Rotations and reflections preserve lengths and angles

Outline



Matrix Decomposition

- Taking the matrix of eigenvectors, $\boldsymbol{V}$, then

$$\mathbf{M}\mathbf{V} = \mathbf{M}(\boldsymbol{v}_1, \boldsymbol{v}_2, \ldots, \boldsymbol{v}_n) = (\lambda_1\boldsymbol{v}_1, \lambda_2\boldsymbol{v}_2, \ldots, \lambda_n\boldsymbol{v}_n) = \mathbf{V}\boldsymbol{\Lambda}$$

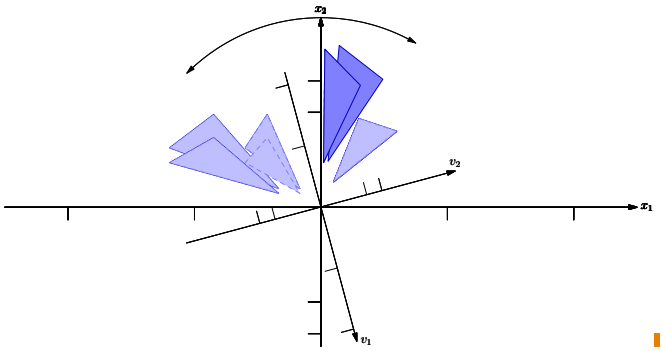
- where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$
- Now

$$\mathbf{M} = \mathbf{M}\mathbf{V}\mathbf{V}^\mathsf{T} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\mathsf{T}$$

- Very important *similarity transform*

Mappings by Symmetric Matrices

$$\mathbf{M} = \begin{pmatrix} 0.83 & -0.31 \\ -0.31 & 1.9 \end{pmatrix} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\mathsf{T} = \begin{pmatrix} \cos(-75) & \sin(-75) \\ -\sin(-75) & \cos(-75) \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 0.75 \end{pmatrix} \begin{pmatrix} \cos(75) & \sin(75) \\ -\sin(75) & \cos(75) \end{pmatrix}$$



Inverses

- For any square matrix

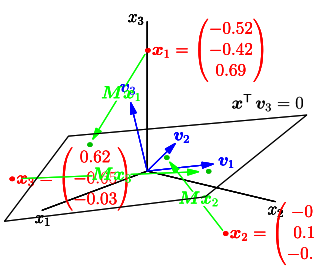
$$\mathbf{M} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\mathsf{T} \qquad \mathbf{M}^{-1} = \mathbf{V}\boldsymbol{\Lambda}^{-1}\mathbf{V}^\mathsf{T}$$

- Where $\boldsymbol{\Lambda}^{-1} = \mathrm{diag}(\frac{1}{\lambda_1}, \frac{1}{\lambda_2}, \ldots, \frac{1}{\lambda_n}) = \begin{pmatrix} \frac{1}{\lambda_1} & 0 & \cdots & 0 \\ 0 & \frac{1}{\lambda_2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \frac{1}{\lambda_n} \end{pmatrix}$
- Since

$$\mathbf{M}\mathbf{M}^{-1} = (\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\mathsf{T})(\mathbf{V}\boldsymbol{\Lambda}^{-1}\mathbf{V}^\mathsf{T}) = \mathbf{V}\boldsymbol{\Lambda}(\mathbf{V}^\mathsf{T}\mathbf{V})\boldsymbol{\Lambda}^{-1}\mathbf{V}^\mathsf{T}$$
$$= \mathbf{V}\boldsymbol{\Lambda}\boldsymbol{\Lambda}^{-1}\mathbf{V}^\mathsf{T} = \mathbf{V}\mathbf{V}^\mathsf{T} = \mathbf{I}$$

- I.e, Small eigenvalues become large eigenvalues and visa verse

Ill-Conditioning Again



$$\boldsymbol{M} = \begin{pmatrix} 0.80 & 0.01 & 0.25 \\ 0.01 & 0.85 & 0.48 \\ 0.25 & 0.48 & 0.5 \end{pmatrix} = \boldsymbol{V} \begin{pmatrix} 0.8 & 0 & 0 \\ 0 & 1.25 & 0 \\ 0 & 0 & 0.1 \end{pmatrix} \boldsymbol{V}^\mathsf{T}$$

$$\boldsymbol{M}^{-1} = \begin{pmatrix} 1.9 & 1.17 & -2.1 \\ 1.17 & 3.34 & -3.8 \\ -2.1 & -3.8 & 6.8 \end{pmatrix} = \boldsymbol{V} \begin{pmatrix} 1.25 & 0 & 0 \\ 0 & 0.8 & 0 \\ 0 & 0 & 10 \end{pmatrix} \boldsymbol{V}^\mathsf{T}$$

Condition Number

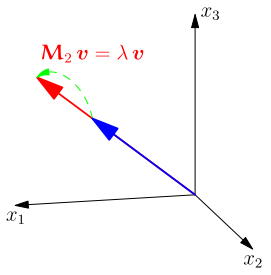
- Taking matrix inverses can be inherently unstable
- Any small error can be amplified by taking the inverse
- The stability of the inverse depends on the ratio of smallest eigenvalue to the largest eigenvalue (i.e. the biggest possible amplification compared to the smallest)
- Note that the Hilbert-norm of a matrix is the absolute value of the largest eigenvalue
- The condition number is given by

$$\|\mathbf{M}\|_H \times \|\mathbf{M}^{-1}\|_H = \frac{|\lambda_{\max}|}{|\lambda_{\min}|}$$

- Large condition number implies very ill-conditioned

Outline

1. Eigenvectors
2. Orthogonal Matrices
3. Eigen Decomposition
4. **Low Rank Approximation**



Rank of a Matrix

- The rank of a matrix, $\mathbf{M}$, is the number of non-zero eigenvalues
- The space spanned by the eigenvectors $\boldsymbol{v}_a$, $\boldsymbol{v}_b$, etc. with zero eigenvalue forms a **null space**
- Any vector in the null space will get projected to the zero vector

$$\mathbf{M}(a\boldsymbol{v}_a + b\boldsymbol{v}_b + \cdots) = \mathbf{0}$$

- A square matrix is said to be **rank deficient** if it has any eigenvectors with eigenvalue equal to 0
- This happens when the columns of the matrix are not linearly independent

"Inverting" Rank Deficient Matrices

- Rank deficient matrices are non-invertible (i.e. we don't know the vector $\boldsymbol{x}$ such that $\mathbf{M}\boldsymbol{x} = \boldsymbol{b}$) as we don't know the component of the $\boldsymbol{x}$ in the null space
- Although we don't know $\boldsymbol{x}$ we can find a vector, $\boldsymbol{x}$, that satisfies $\mathbf{M}\boldsymbol{x} = \boldsymbol{b}$
- Given a symmetric $n \times n$ matrix with k non-zero eigenvalues λ_1, λ_2, ..., λ_k we can construct a "pseudo inverse" $\mathbf{M}^+$ as $\mathbf{V}\boldsymbol{\Lambda}^+\mathbf{V}^\mathsf{T}$ where $\boldsymbol{\Lambda}^+ = \mathrm{diag}(\frac{1}{\lambda_1}, \frac{1}{\lambda_2}, \ldots, \frac{1}{\lambda_k}, 0, \ldots, 0)$
- This finds the vector $\boldsymbol{x}$ with no component in the null space (it is the solution with the smallest norm)
- This is a different to the pseudo inverse for non-square matrices

Low Rank Approximation

- Recall that matrices with large and small eigenvectors are ill-conditions so the inverse has the potential to greatly amplify any measurement error
- One work around is to set all small eigenvalues to zero and use the pseudo inverse
- Setting small eigenvalues to zero reduces the rank of the matrix and is an example of a low rank approximation
- Low rank approximations are much used to obtain approximate models for arrays of data (we will revisit this when we look at SVD)

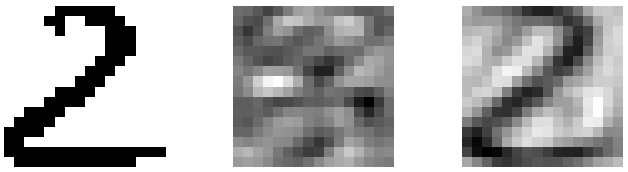
Summary

- Linear mappings are commonly used in machine learning algorithms such as regression
- We can understand symmetric operators by looking at their eigenvectors
- Any symmetric matrix can be decomposed as $\mathbf{M} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\mathsf{T}$
  - where $\mathbf{V}$ are orthogonal matrices whose rows are the eigenvector
  - and $\boldsymbol{\Lambda}$ is a diagonal matrix of the eigenvalues
- This decomposition allows us to understand inverse mappings

Advanced Machine Learning

Principal Component Analysis (PCA)

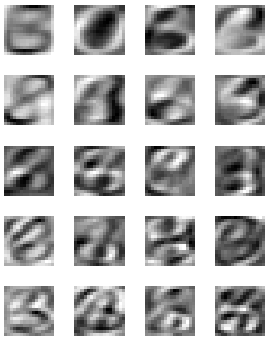
1.6 -1.1 -1.6 2.1 -0.52 2.8 0.72 0.7 -0.68 -0.41 -1.4 -1.5 -0.54 -0.62 1.3 -1.4 -0.27 0.74 0.77 -1



Covariance matrices, dimensionality reduction, PCA, Duality

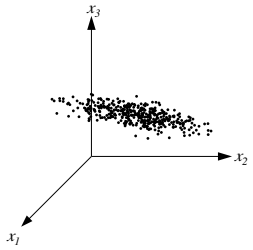
Outline

1. Covariance Matrices
2. Principal Component Analysis
3. Duality



Spread of Data

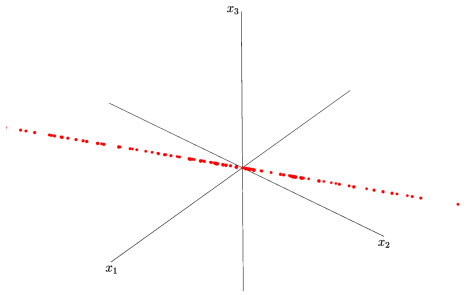
- Often data varies significantly in only some directions



- Reduce dimensions by projecting onto low dimensional subspace with maximum variation

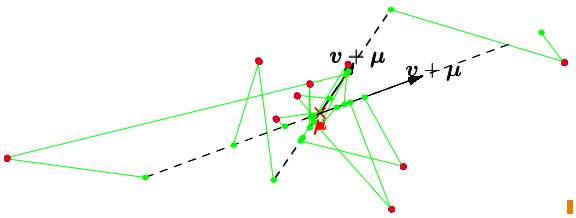
Looking is not Enough

Can't spot low dimensional data by looking at numbers



Dimensionality Reduction

- Often helpful to consider only directions where data varies significantly
- Want to find directions along which data has its greatest variation



Direction of Maximum Variation

- Look for the vector $\boldsymbol{v}$ with $\|\boldsymbol{v}\|^2 = 1$ to maximise

$$\sigma^2 = \frac{1}{m-1}\sum_{i=1}^{m}\left(\boldsymbol{v}^\mathsf{T}(\boldsymbol{x}_i - \boldsymbol{\mu})\right)^2$$

- This is a constrained optimisation problem
- Solve by maximising Lagrangian

$$\mathcal{L} = \frac{1}{m-1}\sum_{k=1}^{m}\left(\boldsymbol{v}^\mathsf{T}(\boldsymbol{x}_k - \boldsymbol{\mu})\right)^2 - \lambda\left(\|\boldsymbol{v}\|^2 - 1\right)$$

- λ is a Lagrange multiplier

Direction of Maximum Variation

- Expanding the Lagrangian

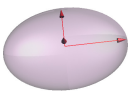
$$\begin{aligned}\mathcal{L} &= \frac{1}{m-1}\sum_{k=1}^{m}\left(\boldsymbol{v}^\mathsf{T}(\boldsymbol{x}_k - \boldsymbol{\mu})\right)^2 - \lambda\left(\|\boldsymbol{v}\|^2 - 1\right)\\ &= \frac{1}{m-1}\sum_{k=1}^{m}\left(\boldsymbol{v}^\mathsf{T}(\boldsymbol{x}_k - \boldsymbol{\mu})(\boldsymbol{x}_k - \boldsymbol{\mu})^\mathsf{T}\boldsymbol{v}\right) - \lambda\left(\|\boldsymbol{v}\|^2 - 1\right)\\ &= \boldsymbol{v}^\mathsf{T}\left(\frac{1}{m-1}\sum_{k=1}^{m}(\boldsymbol{x}_k - \boldsymbol{\mu})(\boldsymbol{x}_k - \boldsymbol{\mu})^\mathsf{T}\right)\boldsymbol{v} - \lambda\left(\|\boldsymbol{v}\|^2 - 1\right)\\ &= \boldsymbol{v}^\mathsf{T}\mathbf{C}\boldsymbol{v} - \lambda\left(\boldsymbol{v}^\mathsf{T}\boldsymbol{v} - 1\right)\end{aligned}$$

- Extrema of the Lagrangian

$$\nabla\mathcal{L} = 2(\mathbf{C}\boldsymbol{v} - \lambda\boldsymbol{v}) = 0 \qquad \Rightarrow \qquad \mathbf{C}\boldsymbol{v} = \lambda\boldsymbol{v}$$

Direction of Maximum Variation

- The eigenvectors are directions that are extrema of the variance



- The variance in direction $\boldsymbol{v}$ is equal to

$$\begin{aligned}\sigma^2 &= \frac{1}{m-1}\sum_{i=1}^{m}\left(\boldsymbol{v}^\mathsf{T}(\boldsymbol{x}_i - \boldsymbol{\mu})\right)^2\\ &= \boldsymbol{v}^\mathsf{T}\mathbf{C}\boldsymbol{v} = \lambda\boldsymbol{v}^\mathsf{T}\boldsymbol{v} = \lambda\end{aligned}$$

- The variance is maximised by the eigenvector with the maximum eigenvalue

Covariance Matrix

- The **covariance matrix** is defined as

$$\mathbf{C} = \frac{1}{m-1}\sum_{k=1}^{m}\left(\boldsymbol{x}_k - \boldsymbol{\mu}\right)\left(\boldsymbol{x}_k - \boldsymbol{\mu}\right)^{\mathsf{T}}$$

- The components C_{ij} measure how the i^{th} and j^{th} components co-vary

$$C_{ij} = \frac{1}{m-1}\sum_{k=1}^{m}\left(x_{ik} - \mu_i\right)\left(x_{jk} - \mu_j\right)$$

- C.f. covariance of random variables

$$\mathrm{Cov}(X,Y) = \mathbb{E}[(X - \mathbb{E}[X])(Y - \mathbb{E}[Y])]$$

Outer Product

- Remember that the outer-product of two vectors is defined as

$$\boldsymbol{x}\boldsymbol{y}^{\mathsf{T}} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}\begin{pmatrix} y_1 & y_2 & \cdots & y_n \end{pmatrix} = \begin{pmatrix} x_1y_1 & x_1y_2 & \cdots & x_1y_n \\ x_2y_1 & x_2y_2 & \cdots & x_2y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_ny_1 & x_ny_2 & \cdots & x_ny_n \end{pmatrix}$$

- C.f. Inner product

$$\boldsymbol{x}^{\mathsf{T}}\boldsymbol{y} = \begin{pmatrix} x_1 & x_2 & \cdots & x_n \end{pmatrix}\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = x_1y_1 + x_2y_2 + \cdots + x_ny_n$$

Matrix Form

- The covariance matrix is

$$\mathbf{C} = \frac{1}{m-1}\sum_{k=1}^{m}\left(\boldsymbol{x}_k - \boldsymbol{\mu}\right)\left(\boldsymbol{x}_k - \boldsymbol{\mu}\right)^{\mathsf{T}}$$

- Define the matrix

$$\mathbf{X} = \frac{1}{\sqrt{m-1}}\left(\boldsymbol{x}_1 - \boldsymbol{\mu}, \boldsymbol{x}_2 - \boldsymbol{\mu}, \cdots \boldsymbol{x}_m - \boldsymbol{\mu}\right)$$

- We can write the covariance matrix as

$$\mathbf{C} = \mathbf{X}\mathbf{X}^{\mathsf{T}}$$

Properties of Covariance Matrix

- The **quadratic form** of a vector and matrix is defined as

$$\boldsymbol{v}^{\mathsf{T}}\mathbf{M}\boldsymbol{v}$$

- The quadratic form of a covariance matrix is non-negative for any vector

$$\boldsymbol{v}^{\mathsf{T}}\mathbf{C}\boldsymbol{v} = \boldsymbol{v}^{\mathsf{T}}\mathbf{X}\mathbf{X}^{\mathsf{T}}\boldsymbol{v} = \boldsymbol{u}^{\mathsf{T}}\boldsymbol{u} = \|\boldsymbol{u}\|^2 \geq 0$$

where $\boldsymbol{u} = \mathbf{X}^{\mathsf{T}}\boldsymbol{v}$

- Matrices with non-negative quadratic forms are known as **positive semi-definite**

Eigenvalue Decomposition

- The eigenvectors of $\mathbf{C}$ with the largest eigenvalues are known as the **principal components**
- The eigenvalues are all greater than or equal to zero
- Recall an eigenvector $\boldsymbol{v}$ satisfies the equation

$$\mathbf{C}\boldsymbol{v} = \lambda\boldsymbol{v}$$

- Multiplying both sides by $\boldsymbol{v}^{\mathsf{T}}$

$$\boldsymbol{v}^{\mathsf{T}}\mathbf{C}\boldsymbol{v} = \lambda\boldsymbol{v}^{\mathsf{T}}\boldsymbol{v} = \lambda\|\boldsymbol{v}\|^2$$

but $\boldsymbol{v}^{\mathsf{T}}\mathbf{C}\boldsymbol{v} \geq 0$ and $\|\boldsymbol{v}\|^2 > 0$ so

$$\lambda = \frac{\boldsymbol{v}^{\mathsf{T}}\mathbf{C}\boldsymbol{v}}{\|\boldsymbol{v}\|^2} \geq 0$$

Surface Defined by Matrix

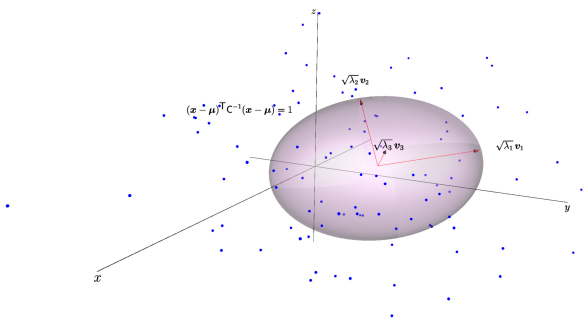
- The set of vectors $\boldsymbol{x}$ such that

$$\boldsymbol{x}^{\mathsf{T}}\mathbf{C}^{-1}\boldsymbol{x} = 1$$

defines a surface

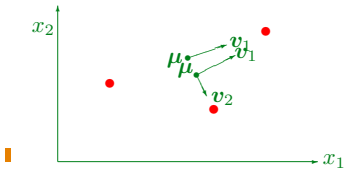
- The surface is an ellipsoid, $\mathcal{E}$
- The eigenvectors point in the direction of the principal axes of the ellipsoid
- The radii of the principal axes are equal to the square root of the eigenvalues

Ellipsoid and Eigen Space



Spanning Input Space

- A covariance matrix will have a zero eigenvalue only if there is no variation in the direction of the corresponding eigenvector
- A covariance matrix will have zero eigenvalues if the number of patterns are less than or equal to the number of dimensions
- A covariance matrix formed from $p+1$ patterns that are linearly independent (i.e. you cannot form any one out of p of the other patterns) will have no zero eigenvalues

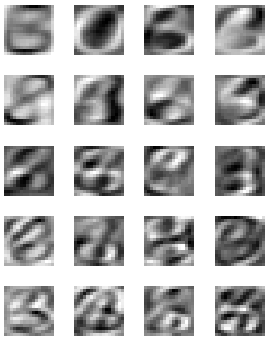


Positive Definite

- Matrices with no zero eigenvalues are called **full rank** matrices (as opposed to rank deficient)
- Full rank matrices are invertible, rank deficient matrices are singular and non-invertible
- Full rank covariance matrices have positive eigenvalues only and are said to be **positive definite**
- We would expect that when $m > p$ the covariance matrix will be positive definite unless there are some symmetries that linearly constrain the patterns

Outline

1. Covariance Matrices
2. **Principal Component Analysis**
3. Duality



Principal Component Analysis

- PCA occurs as follows
  - Construct the covariance matrix
  - Find the eigenvalues and eigenvectors
  - Keep the eigenvectors with the largest eigenvalues (principal components)
  - Project the inputs into the space spanned by the principal components
- We then use the projected inputs as inputs to our learning machine

Projection Matrix

- To project the inputs construct the projection matrix

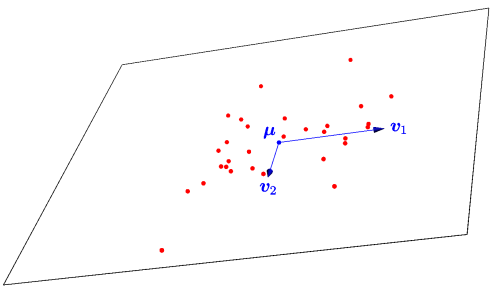
$$\mathbf{P} = \begin{pmatrix} \boldsymbol{v}_1^{\mathsf{T}} \\ \boldsymbol{v}_2^{\mathsf{T}} \\ \vdots \\ \boldsymbol{v}_k^{\mathsf{T}} \end{pmatrix}$$

- $k < p$ is the number of principal components we keep
- Given a p-dimensional input pattern $\boldsymbol{x}$ we can construct a k-dimensional representation $\boldsymbol{z}$

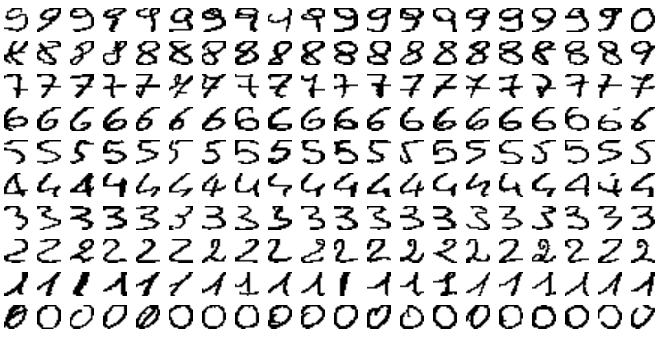
$$\boldsymbol{z} = \mathbf{P}(\boldsymbol{x} - \boldsymbol{\mu})$$

- Use $\boldsymbol{z}$ as our new inputs

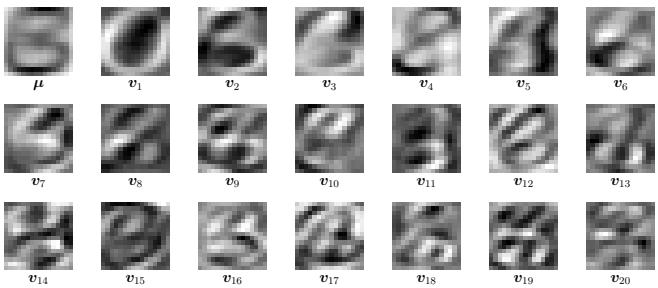
Subspace Projection



Hand Written Digits



Eigenvectors



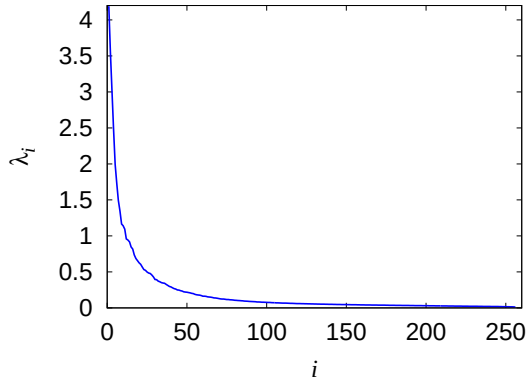
Reconstruction

- Projecting into a subspace of eigenvectors can be seen as approximating the inputs by

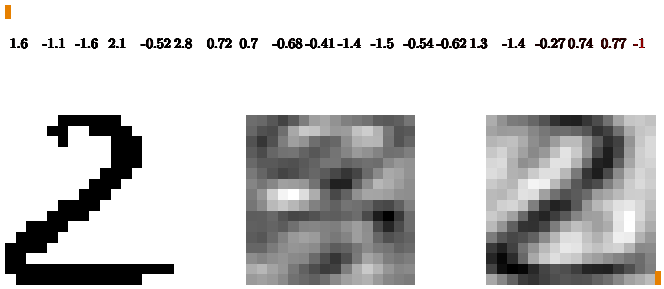
$$\hat{\boldsymbol{x}}_i = \boldsymbol{\mu} + \sum_{j=1}^{k} z_j^i \boldsymbol{v}_j, \qquad z_j^i = \boldsymbol{v}_j^{\mathsf{T}}(\boldsymbol{x}_i - \boldsymbol{\mu}), \qquad \|\boldsymbol{v}_j\| = 1$$

- Principle component analysis projects the data into a subspace of size m with the minimal approximation error $\mathbb{E}\left[\|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|^2\right]$
- The loss of "energy" (or squared error) is equal to the sum of the eigenvalues in the directions that are ignored

Eigenvalues for Digits

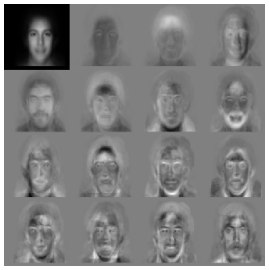


Reconstruction from Eigenvectors



Outline

1. Covariance Matrices
2. Principal Component Analysis
3. **Duality**



PCA for Images

- An image often contains around $p = 256 \times 256 = 64k$ pixels
- In standard PCA we would create an $p \times p$ matrix with over 4×10^9 elements
- This is intractable
- m images span at most a $m-1$ dimensional subspace
- Usually this subspace will be much smaller than the space of all images $m \ll p$

Dual Matrix

- The covariance $\mathbf{C} = \mathbf{X}\mathbf{X}^\mathsf{T}$ is a $p \times p$ matrix
- Consider the $m \times m$ matrix $\mathbf{D} = \mathbf{X}^\mathsf{T}\mathbf{X}$
- Suppose $\boldsymbol{v}$ is an eigenvector of $\mathbf{D}$

$$\begin{aligned}\mathbf{D}\boldsymbol{v} &= \lambda\boldsymbol{v}\\ \mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{v} &= \lambda\boldsymbol{v}\\ \mathbf{X}\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{v} &= \lambda\mathbf{X}\boldsymbol{v}\\ \mathbf{C}\mathbf{X}\boldsymbol{v} &= \lambda\mathbf{X}\boldsymbol{v} \quad\Rightarrow\quad \mathbf{C}\boldsymbol{u} = \lambda\boldsymbol{u}\end{aligned}$$

- $\boldsymbol{u} = \mathbf{X}\boldsymbol{v}$ (and $\boldsymbol{v} \propto \mathbf{X}^\mathsf{T}\boldsymbol{u}$)

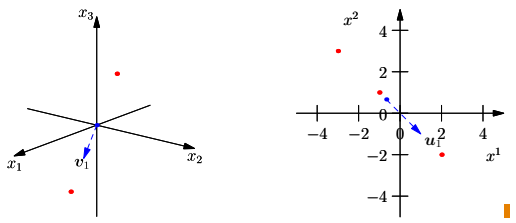
Dual Matrix

- Matrices $\mathbf{C} = \mathbf{X}\mathbf{X}^\mathsf{T}$ and $\mathbf{D} = \mathbf{X}^\mathsf{T}\mathbf{X}$ have the same eigenvalues
- Can use the dual $m \times m$ matrix $\mathbf{D}$ to find eigenvalues and eigenvectors of $\mathbf{C}$
- Note that $\mathbf{D} = \mathbf{X}^\mathsf{T}\mathbf{X}$ has components $D_{kl} \propto (\boldsymbol{x}_k - \boldsymbol{\mu})^\mathsf{T}(\boldsymbol{x}_l - \boldsymbol{\mu})$
- Takes $O(p \times m \times m)$ time to construct $\mathbf{D}$
- We work in a "dual space" which is the space spanned by the examples

What Does a Subspace Look Like?

- Consider $\boldsymbol{y}^1 = \begin{pmatrix}2\\4\\4\end{pmatrix}$, $\boldsymbol{y}^2 = \begin{pmatrix}8\\6\\2\end{pmatrix}$ with mean $\boldsymbol{\mu} = \begin{pmatrix}5\\5\\3\end{pmatrix}$
- Subtracting the mean $\boldsymbol{x}^i = \boldsymbol{y}^i - \boldsymbol{\mu}$ we can construct matrix

$$\mathbf{X} = \begin{pmatrix}x_1^1 & x_1^2\\ x_2^1 & x_2^2\\ x_3^1 & x_3^2\end{pmatrix} = \begin{pmatrix}-3 & 3\\ -1 & 1\\ 2 & -2\end{pmatrix}$$

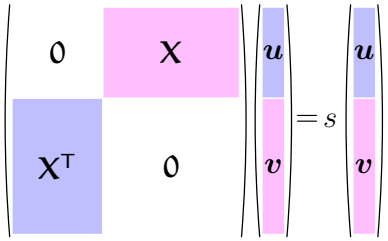


Summary

- PCA allows us to reduce the dimensionality of the inputs
- We project the inputs into a sub-space where the data varies the most
- We can work in either the original space ($\mathbf{X}\mathbf{X}^\mathsf{T}$) or the dual space ($\mathbf{X}^\mathsf{T}\mathbf{X}$)
- When we have many more features than examples (i.e. $p \gg m$) then it is more efficient working in the dual space
- We will see examples of dual spaces again when we look at SVMs

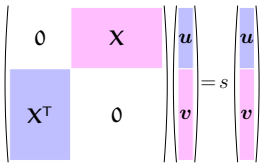
Advanced Machine Learning

Singular Value Decomposition (SVD)



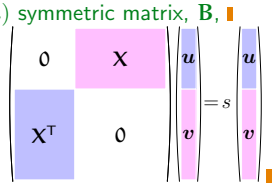
Singular Valued Decomposition, SVD, general linear maps

Outline



Singular Valued Decomposition

- Consider an arbitrary $n \times m$ matrix $\mathbf{X}$, and construct the $(n+m) \times (n+m)$ symmetric matrix, $\mathbf{B}$,



$\binom{\boldsymbol{u}}{\boldsymbol{v}}$ is an eigenvector of $\mathbf{B}$ with eigenvalue s

- We observe that

$$\mathbf{X}\boldsymbol{v} = s\boldsymbol{u} \qquad \mathbf{X}^\mathsf{T}\boldsymbol{u} = s\boldsymbol{v}$$
$$\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{v} = s\mathbf{X}^\mathsf{T}\boldsymbol{u} = s^2\boldsymbol{v} \qquad \mathbf{X}\mathbf{X}^\mathsf{T}\boldsymbol{u} = s\mathbf{X}\boldsymbol{v} = s^2\boldsymbol{u}$$

Eigenvectors

- Note that as $\mathbf{X}\boldsymbol{v} = s\boldsymbol{u}$ and $\mathbf{X}^\mathsf{T}\boldsymbol{u} = s\boldsymbol{v}$ then

$$\mathbf{X}(-\boldsymbol{v}) = (-s)\boldsymbol{u} \qquad \mathbf{X}^\mathsf{T}\boldsymbol{u} = (-s)(-\boldsymbol{v})$$

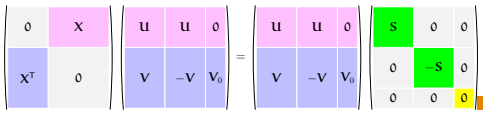
if $\binom{\boldsymbol{u}}{\boldsymbol{v}}$ is an eigenvector of $\mathbf{B}$ with eigenvalue s then so is $\binom{\boldsymbol{u}}{-\boldsymbol{v}}$ with eigenvalue $-s$

- If $n < m$ then $\mathbf{X}^\mathsf{T}\mathbf{X}$ is not full rank so some eigenvalues are zero
- As a consequence $m - n$ vectors exist such that $\boldsymbol{X}\boldsymbol{v} = 0$
- The eigenvalues and eigenvectors are

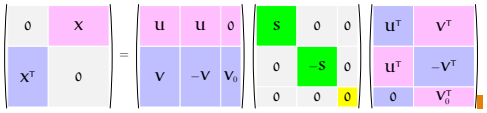
$$n \times \left(s_i, \binom{\boldsymbol{u}_i}{\boldsymbol{v}_i}\right) \quad n \times \left(-s_i, \binom{\boldsymbol{u}_i}{-\boldsymbol{v}_i}\right) \quad m - n \times \left(0, \binom{0}{\boldsymbol{v}_k}\right)$$

Matrix Decomposition

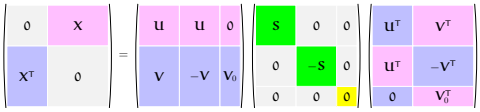
- Stacking the eigenvectors into a matrix



- Since the vectors $\binom{\boldsymbol{u}_i}{\boldsymbol{v}_i}$ are eigenvectors of a symmetric matrix they from an orthogonal matrix if they are normalised.
- Multiply on the right by the transpose of the orthogonal matrix



Normalisation Subtlety



- Multiplying out we have

$$\mathbf{X} = 2\mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T} \qquad \mathbf{X}^\mathsf{T} = 2\mathbf{V}\mathbf{S}\mathbf{U}^\mathsf{T}$$

- Now the vectors $\boldsymbol{u}_i$ and $\boldsymbol{v}_i$ form an orthogonal set as it satisfy

$$\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{v} = s^2\boldsymbol{v} \qquad \mathbf{X}\mathbf{X}^\mathsf{T}\boldsymbol{u} = s^2\boldsymbol{u}$$

- But they are not normalised (since $\binom{\boldsymbol{u}_i}{\boldsymbol{v}_i}$ is normalised). If we define $\tilde{\mathbf{U}} = \sqrt{2}\mathbf{U}$ and $\tilde{\mathbf{V}} = \sqrt{2}\mathbf{V}$ we find

$$\mathbf{X} = \tilde{\mathbf{U}}\mathbf{S}\tilde{\mathbf{V}}^\mathsf{T} \qquad \mathbf{X}^\mathsf{T} = \tilde{\mathbf{V}}\mathbf{S}\tilde{\mathbf{U}}^\mathsf{T}$$

SVD

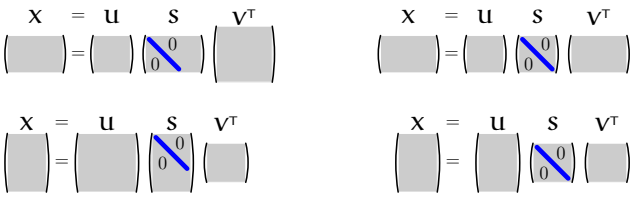
- Any matrix, $\mathbf{X}$, can be written as $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T}$
  - $\mathbf{U}$, $\mathbf{V}$ are orthogonal matrices
  - $\mathbf{S} = \mathrm{diag}(s_1, s_2, \ldots, s_n)$
- s_i can always be chosen to be positive and are known as **singular values**
- Singular value decomposition applies to both square and non-square matrices—they describe general linear mappings

Finding SVD

- Most libraries will compute the SVD for you
- They can do this by choosing the smaller of two matrices $\mathbf{X}\mathbf{X}^\mathsf{T}$ and $\mathbf{X}^\mathsf{T}\mathbf{X}$ and then compute the eigenvalues
- The singular values are the square root of the eigenvalues (notice that $\mathbf{X}\mathbf{X}^\mathsf{T}$ and $\mathbf{X}^\mathsf{T}\mathbf{X}$ are both positive semi-definite so the eigenvalues will be non-negative)
- It can compute the $\mathbf{U}$ matrix or $\boldsymbol{V}$ matrix by multiplying through by $\mathbf{X}$ or $\mathbf{X}^\mathsf{T}$ ($\mathbf{U} = \mathbf{X}\boldsymbol{V}\mathbf{S}^{-1}$ and $\boldsymbol{V} = \mathbf{X}^\mathsf{T}\mathbf{U}\mathbf{S}^{-1}$)
- In practice to perform PCA most people subtract the mean from their data and then perform SVD

Economical Forms of SVD

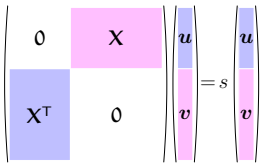
- Often the rows or columns of the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ that are not associated with a singular value are ignored



- In Matlab these are obtained using

```
>> [U, S, V] = svd(X)
>> [U, S, V] = svd(X,'econ'))
```


Outline



General Matrix

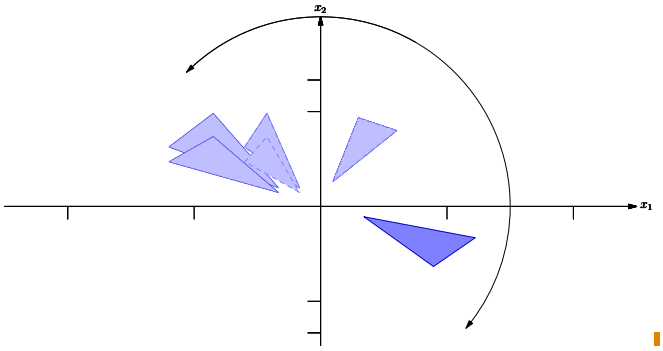
- Recall that we can compute the SVD for any matrix, $\mathbf{X}$
- As matrices describe the most general linear mapping

$$\boldsymbol{v} \to \mathcal{T}[\boldsymbol{v}] = \mathbf{X}\boldsymbol{v}$$

- We can use SVD to understand any linear mapping
- Thus any linear mapping can be seen as a rotation followed by a squashing or expansion independently in each coordinate followed by another rotation

Matrices

$$\mathbf{M} = \begin{pmatrix} -0.45 & 1.9 \\ -0.77 & -0.025 \end{pmatrix} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T} = \begin{pmatrix} \cos(-175) & \sin(-175) \\ -\sin(-175) & \cos(-175) \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 0.75 \end{pmatrix} \begin{pmatrix} \cos(75) & \sin(75) \\ -\sin(75) & \cos(75) \end{pmatrix}$$



Determinants

- The determinant, $|\mathbf{M}|$ of a matrix $\mathbf{M}$ is defined for square matrices
- It describes the change in volume under the mapping
- Now for any two matrices $|\mathbf{AB}| = |\mathbf{A}||\mathbf{B}|$
- Thus

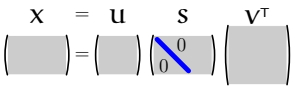
$$|\mathbf{M}| = |\mathbf{U}||\mathbf{S}||\mathbf{V}^\mathsf{T}|$$

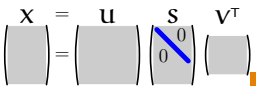
- For and orthogonal matrix $|\mathbf{U}| = \pm 1$
- Thus

$$|\mathbf{M}| = \pm|\mathbf{S}| = \pm\prod_i s_i$$

Non-Square Matrices

- When the matrices are non-square then the matrix of singular value matrix will either
  - ⋆ Squash some directions to zero
  - ⋆ Introduce new dimensions orthogonal to the vector





- The rank of an arbitrary matrix is the number of non-zero singular values (also number of linearly independent rows or columns)

Duality Revisited

- If $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T}$ then

$$\begin{aligned} \mathbf{C} &= \mathbf{X}\mathbf{X}^\mathsf{T} & \mathbf{D} &= \mathbf{X}^\mathsf{T}\mathbf{X} \\ &= \mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T}\mathbf{V}\mathbf{S}^\mathsf{T}\mathbf{U}^\mathsf{T} & &= \mathbf{V}\mathbf{S}^\mathsf{T}\mathbf{U}^\mathsf{T}\mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T} \\ &= \mathbf{U}(\mathbf{S}\mathbf{S}^\mathsf{T})\mathbf{U}^\mathsf{T} & &= \mathbf{V}(\mathbf{S}^\mathsf{T}\mathbf{S})\mathbf{V}^\mathsf{T} \end{aligned}$$

- If $\mathbf{X}$ is an $p \times m$ matrix then $\mathbf{S}\mathbf{S}^\mathsf{T}$ is a $p \times p$ diagonal matrix with elements $S^2_{ii} = s^2_i$
- $\mathbf{S}^\mathsf{T}\mathbf{S}$ is an $m \times m$ matrix with elements $S^2_{ii} = s^2_i$
- $\mathbf{U}$ and $\mathbf{V}$ are matrices of eigenvectors for $\mathbf{C}$ and $\mathbf{D}$
- The eigenvalues are $\lambda_i = S^2_{ii} = s^2_i$

$\mathbf{S}\mathbf{S}^\mathsf{T}$ and $\mathbf{S}^\mathsf{T}\mathbf{S}$

$$\mathbf{S} = \begin{pmatrix} s_1 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & s_2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_m & 0 & 0\cdots & 0 \end{pmatrix}$$

$$\mathbf{S}^\mathsf{T}\mathbf{S} = \begin{pmatrix} s_1^2 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & s_2^2 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_m^2 & 0 & 0\cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & 0\cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots\cdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0\cdots & 0 \end{pmatrix}$$

$$\mathbf{S}\mathbf{S}^\mathsf{T} = \begin{pmatrix} s_1^2 & 0 & \cdots & 0 \\ 0 & s_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_m^2 \end{pmatrix}$$

Having A Go

- It's really easy to verify this in MATLAB or OCTAVE

```
>> X = rand(3,2)
>> [U, S, V] = svd(X)
>> U*S*V'
>> U(:,1)'*U(:,2)
>> U'*U
>> U*U'
>> [Ua,L] = eig(X*X')
>> S*S'
```

- Test yourself!

Outline

$$\begin{pmatrix} 0 & X \\ X^{\mathsf{T}} & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = s \begin{pmatrix} u \\ v \end{pmatrix}$$

Linear Regression

- Given a set of data $\mathcal{D} = \{(\boldsymbol{x}_i, y_i) | k = 1, 2, \ldots, m\}$
- In linear regression we try to fit a linear model

$$f(\boldsymbol{x}|\boldsymbol{w}) = \boldsymbol{x}^{\mathsf{T}}\boldsymbol{w}$$

- Which we fit by minimising the squared error loss

$$L(\boldsymbol{w}) = \sum_{k=1}^{m} (f(\boldsymbol{x}_i|\boldsymbol{w}) - y_i)^2$$

Matrix Form

- In matrix from we write $L(\boldsymbol{w}) = \|\mathbf{X}\boldsymbol{w} - \boldsymbol{y}\|^2$

$$\mathbf{X} = \begin{pmatrix} \boldsymbol{x}_1^{\mathsf{T}} \\ \boldsymbol{x}_2^{\mathsf{T}} \\ \vdots \\ \boldsymbol{x}_m^{\mathsf{T}} \end{pmatrix} \qquad \boldsymbol{y} = \begin{pmatrix} y_1 \\ y_2, \\ \vdots \\ y_m \end{pmatrix}$$

- Then $\nabla L(\boldsymbol{w}^*) = 0$ implies

$$\boldsymbol{w}^* = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathsf{T}}\boldsymbol{y} = \mathbf{X}^{+}\boldsymbol{y}$$

- This is known as the pseudo-inverse

Using SVD

- Using $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^{\mathsf{T}}$ then

$$\begin{aligned} \mathbf{X}^{+} &= \left(\mathbf{X}^{\mathsf{T}}\mathbf{X}\right)^{-1}\mathbf{X}^{\mathsf{T}} \\ &= \left(\mathbf{V}\mathbf{S}^{\mathsf{T}}\mathbf{S}\mathbf{V}^{\mathsf{T}}\right)^{-1}\mathbf{V}\mathbf{S}^{\mathsf{T}}\mathbf{U}^{\mathsf{T}} \\ &= \mathbf{V}\left(\mathbf{S}^{\mathsf{T}}\mathbf{S}\right)^{-1}\mathbf{V}^{\mathsf{T}}\mathbf{V}\mathbf{S}^{\mathsf{T}}\mathbf{U}^{\mathsf{T}} \\ &= \mathbf{V}\left(\mathbf{S}^{\mathsf{T}}\mathbf{S}\right)^{-1}\mathbf{S}^{\mathsf{T}}\mathbf{U}^{\mathsf{T}} = \mathbf{V}\mathbf{S}^{+}\mathbf{U}^{\mathsf{T}} \end{aligned}$$

- If $m > p$

$$\mathbf{X}^{\mathsf{T}} = \left(\begin{array}{|c|c|c|c|c|c|c|} \hline & & & & & & \\ \hline \end{array}\right), \mathbf{S}^{\mathsf{T}} = \begin{pmatrix} s_1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & s_2 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & s_3 & \cdots & 0 & & 0\cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & s_p & 0 & 0\cdots & 0 \end{pmatrix}$$

Pseudo-Inverse of S

$$\mathbf{S}^{\mathsf{T}}\mathbf{S} = \begin{pmatrix} s_1^2 & 0 & \cdots & 0 \\ 0 & s_2^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_p^2 \end{pmatrix} \quad \left(\mathbf{S}^{\mathsf{T}}\mathbf{S}\right)^{-1} = \begin{pmatrix} s_1^{-2} & 0 & \cdots & 0 \\ 0 & s_2^{-2} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & s_p^{-2} \end{pmatrix}$$

$$\mathbf{S}^{+} = \left(\mathbf{S}^{\mathsf{T}}\mathbf{S}\right)^{-1}\mathbf{S}^{\mathsf{T}} = \begin{pmatrix} s_1^{-1} & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & s_2^{-1} & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & s_3^{-1} & \cdots & 0 & & 0\cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & s_p^{-1} & 0 & 0\cdots & 0 \end{pmatrix}$$

Ill-Conditioned Data Matrix

- Recall that

$$\boldsymbol{w}^* = \mathbf{X}^{+}\boldsymbol{y} = \mathbf{V}\mathbf{S}^{+}\mathbf{U}^{\mathsf{T}}\boldsymbol{y}$$

- If any of the singular values of $\mathbf{X}$ are small then $\mathbf{S}^{+}$ will magnify components in that direction
- Any errors in the target $\boldsymbol{y}$ will be magnified
- This leads to poor weights

Regularisation

- Consider linear regression with a regulariser

$$\begin{aligned} \mathcal{L}(\boldsymbol{w}) &= \|\mathbf{X}\boldsymbol{w} - \boldsymbol{y}\|^2 + \eta\|\boldsymbol{w}\|^2 \\ &= \boldsymbol{w}^{\mathsf{T}}\left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \eta\mathbf{I}\right)\boldsymbol{w} - 2\boldsymbol{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\boldsymbol{y} + \boldsymbol{y}^{\mathsf{T}}\boldsymbol{y} \end{aligned}$$

- Thus

$$\nabla\mathcal{L}(\boldsymbol{w}) = 2\left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \eta\mathbf{I}\right)\boldsymbol{w} - 2\mathbf{X}^{\mathsf{T}}\boldsymbol{y}$$

- and $\nabla\mathcal{L}(\boldsymbol{w}^*) = 0$ gives

$$\boldsymbol{w}^* = \left(\mathbf{X}^{\mathsf{T}}\mathbf{X} + \eta\mathbf{I}\right)^{-1}\mathbf{X}^{\mathsf{T}}\boldsymbol{y}$$

Regularisation Continued

- Using $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T}$

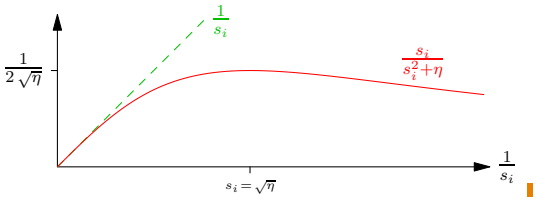
$$\begin{aligned}\boldsymbol{w}^* &= \left(\mathbf{X}^\mathsf{T}\mathbf{X} + \eta\mathbf{I}\right)^{-1}\mathbf{X}^\mathsf{T}\boldsymbol{y} \\ &= \mathbf{V}\left(\mathbf{S}^\mathsf{T}\mathbf{S} + \eta\mathbf{I}\right)^{-1}\mathbf{S}^\mathsf{T}\mathbf{U}^\mathsf{T}\boldsymbol{y}\end{aligned}$$

- where

$$\left(\mathbf{S}^\mathsf{T}\mathbf{S} + \eta\mathbf{I}\right)^{-1}\mathbf{S}^\mathsf{T} = \begin{pmatrix} \frac{s_1}{s_1^2+\eta} & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & \frac{s_2}{s_2^2+\eta} & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \frac{s_3}{s_3^2+\eta} & \cdots & 0 & & 0\cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{s_p}{s_p^2+\eta} & 0 & 0\cdots & 0 \end{pmatrix}$$

Effect of Regularisation

- Without regularisation if $s_i = 0$ the problem would be ill-posed (even $\mathbf{S}^+$ does not exist since s_i^{-1} would be ill defined) and if s_i is small then $\mathbf{S}^+$ is ill conditioned
- Using $\hat{\mathbf{S}}^+ = (\mathbf{S}^\mathsf{T}\mathbf{S} + \eta)^{-1}\mathbf{S}^\mathsf{T}$ instead of $\mathbf{S}^+$ then

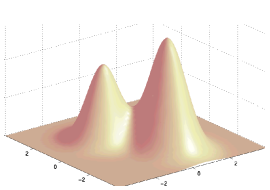
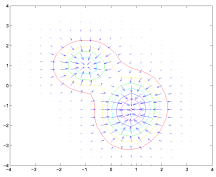


- Regularisation makes the machine much more stable (reduces the variance)

Summary

- Any matrix can be decomposed as $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\mathsf{T}$ where
  - $\mathbf{U}$ and $\mathbf{V}$ are orthogonal (rotation matrices)
  - $\mathbf{S} = \mathrm{diag}(s_1,\ldots,s_n)$ is a diagonal matrix of positive singular values
- This describes the most general linear transform
- The transform exploits the duality between $\mathbf{X}\mathbf{X}^\mathsf{T}$ and $\mathbf{X}^\mathsf{T}\mathbf{X}$
- In linear regression the pseudo-inverse involves the reciprocal of the singular values, which can lead to poor generalisation
- Regularisation improves the conditioning of the "inverse" matrix

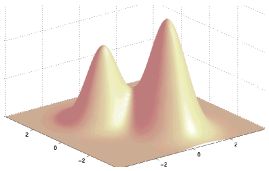
Advanced Machine Learning
Optimisation

$$z = \mathrm{e}^{-(x+1)^2-(y-1)^2} + 0.6\mathrm{e}^{-(x-1)^2-0.5(y+1)^2+0.1(x-3)(y-3)}$$

Gradient descent, quadratic minima, differing length scales

Outline

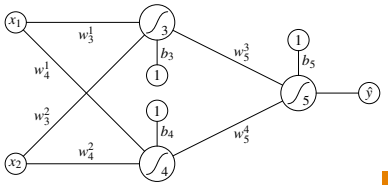


ML = Optimisation

- Many learning machines can be thought of as functions of the form
  $$\hat{y} = f(\boldsymbol{x}|\boldsymbol{w})$$
  (or more generally $\hat{\boldsymbol{y}} = \boldsymbol{f}(\boldsymbol{x}|\boldsymbol{w})$)
- Given an input pattern (set of features) $\boldsymbol{x}$ the learning machine makes a prediction $\hat{y}$
- We try to choose the parameters $\boldsymbol{w}$ so that the predictions are good
- In practice training a learning machine comes down to optimising some loss function

MLP

- We can depict a neural network such as an MLP by a diagram



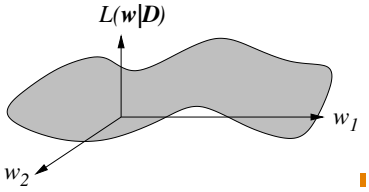
- Stands for the function ($\hat{y} = f(\boldsymbol{x}|\boldsymbol{w})$)
  $$\hat{y} = g\big(w_5^3 g(w_3^1 x_1 + w_3^2 x_2 + b_3) + w_5^4 g(w_4^1 x_1 + w_4^2 x_2 + b_4) + b_5\big)$$
  where, for example, $g(V) = \frac{1}{1+\mathrm{e}^{-V}}$

Training

- Given a (labelled) training dataset
  $$\mathcal{D} = \big\{(\boldsymbol{x}_k, y_k)\big|k = 1,\ldots,m\big\}$$
- We define an error or loss function that we want to minimise
  $$L(\boldsymbol{w}|\boldsymbol{\mathcal{D}}) = \frac{1}{m}\sum_{k=1}^{m}\big(f(\boldsymbol{x}_k|\boldsymbol{w}) - y_k\big)^2$$
- We then use the machine with the weights $\boldsymbol{w}^*$ which minimise $L(\boldsymbol{w}|\mathcal{D})$

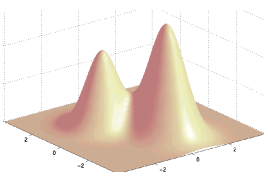
Computing Gradients

- $L(\boldsymbol{w}|\mathcal{D})$ is a complex function of the weights $\boldsymbol{w}$



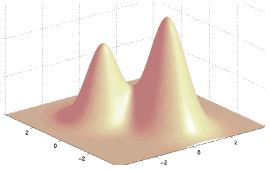
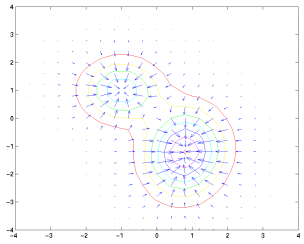
- To minimise we $L(\boldsymbol{w}|\mathcal{D})$ we compute the gradient $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D})$
- In MLP an efficient algorithm for computing the gradient is known as back-prop

Outline



Gradient Optimisation

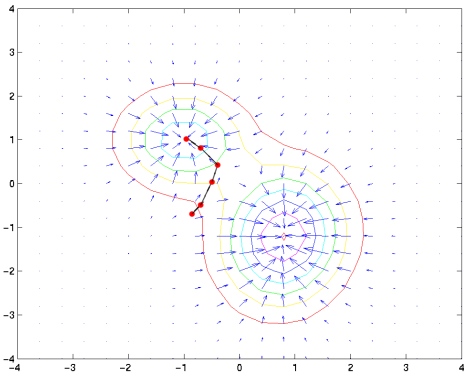
- A maximum or minimum occurs when $\boldsymbol{\nabla} L(\boldsymbol{w}|\mathcal{D}) = \boldsymbol{0}$
- E.g.
  $$L = \mathrm{e}^{-(x+1)^2-(y-1)^2} + 0.6\mathrm{e}^{-(x-1)^2-0.5(y+1)^2+0.1(x-3)(y-3)}$$

Gradient Descent

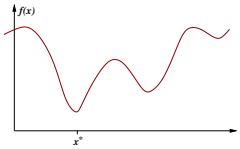
- For a simple function $L(\boldsymbol{w}|\mathcal{D})$ we can solve $\nabla L(\boldsymbol{w}|\mathcal{D}) = \mathbf{0}$ explicitly. E.g. the linear perceptron.
- For a non-linear functions we usually can't solve this set of simultaneous equations.
- We can find a maximum or minimum **iteratively**.
- If we know the gradient then we can follow the gradient.
  - Maximisation: $\boldsymbol{w} \rightarrow \boldsymbol{w}' = \boldsymbol{w} + r\nabla L(\boldsymbol{w}|\mathcal{D})$
  - Minimisation: $\boldsymbol{w} \rightarrow \boldsymbol{w}' = \boldsymbol{w} - r\nabla L(\boldsymbol{w}|\mathcal{D})$

Hill-Climbing



What Goes Right

- Almost all minima are quadratic (Morse's theorem).



- Taylor expanding around a minimum x^*

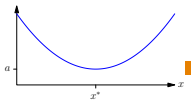
$$\begin{aligned} f(x) &= f(x^*) + (x-x^*)f'(x^*) + \frac{1}{2}(x-x^*)^2 f''(x^*) + \cdots \\ &= f(x^*) + \frac{1}{2}(x-x^*)^2 f''(x^*) + \frac{1}{3!}(x-x^*)^3 f'''(x^*) + \cdots \end{aligned}$$

- If $x - x^*$ is sufficiently small the higher order terms are negligible.

Newton's Method

- If we were in a quadratic minimum

$$f(x) = a + \frac{b}{2}(x-x^*)^2$$



- then

$$f'(x) = b(x-x^*), \qquad f''(x) = b$$

- so

$$x - x^* = \frac{f'(x)}{b} = \frac{f'(x)}{f''(x)}$$

- or

$$x^* = x - \frac{f'(x)}{f''(x)}$$

Newton's Method

- This is Newton's methods.
- For non-quadratic functions Newtons method converges **quadratically** provided we are sufficiently close to a minimum.
- If we are at a distance $x - x^* = \epsilon$ from the minima then after one cycle we will be a distance ϵ^2 after two cycles we will be at a distance ϵ^4, etc.
- If we are too far from a minimum we might go anywhere!
- We should follow the gradient until we are near the minimum.

Taylor's Expansion in High Dimensions

- We can generalise these results to many dimensions.
- The Taylor expansion of a function $f(\boldsymbol{x})$ about $\boldsymbol{x}_0$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + (\boldsymbol{x}-\boldsymbol{x}_0)^\mathsf{T}\nabla f(\boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_0)^\mathsf{T}\mathbf{H}(\boldsymbol{x}-\boldsymbol{x}_0) + \cdots$$

where $\mathbf{H}$ is the **Hessian** matrix with elements

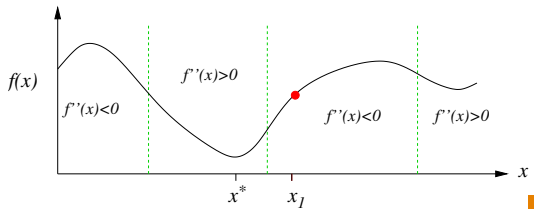
$$H_{ij} = \frac{\partial^2 f(\boldsymbol{x}_0)}{\partial x_i \partial x_j}$$

- Newton's method in high dimension is

$$\boldsymbol{x}^* = \boldsymbol{x} - \mathbf{H}^{-1}\nabla f(\boldsymbol{x})$$

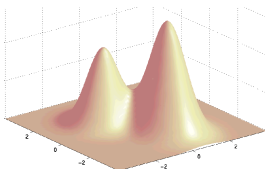
Using the Second Derivative

- If we are optimising N parameters the Hessian is an $N \times N$ matrix.
- It is time-consuming to compute (and prone to errors when coding)—for deep learning it is impossible even to store the Hessian.
- Away from minima they can be misleading



Outline

1. Motivation
2. Gradient Descent
3. **Why Gradient Descent is Difficult**



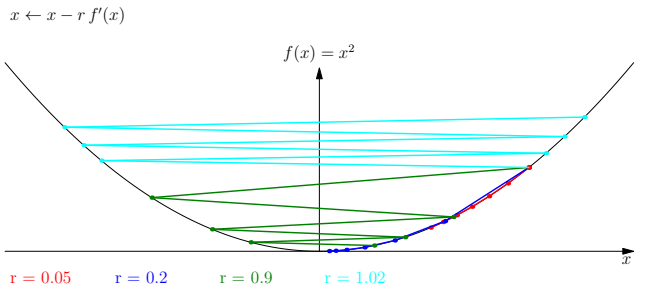
Step Size

- Gradient descent

$$\boldsymbol{x}' = \boldsymbol{x} - r\nabla f(\boldsymbol{x})$$

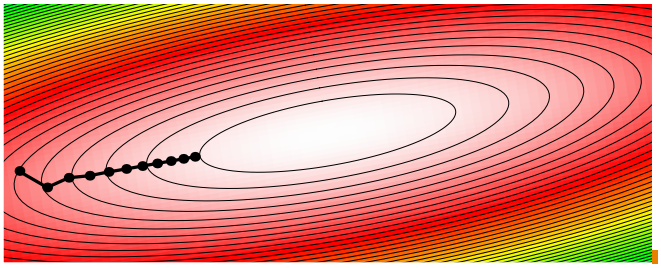
- Need to choose the learning rate of step size, r
- Too small steps takes lots of time
- Too large steps takes you away from a minimum

Step Size



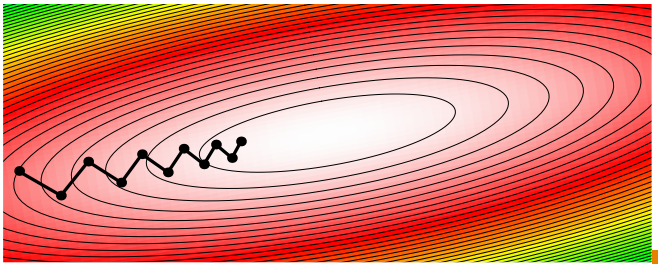
Higher Dimensions

- In higher dimensions the problem is that there are some directions you need to move a long way



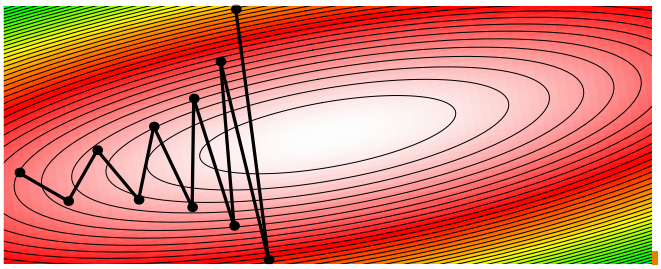
Getting There Quicker

- Increasing the step size speeds up convergence, but the direction of steepest descent doesn't point to the minimum



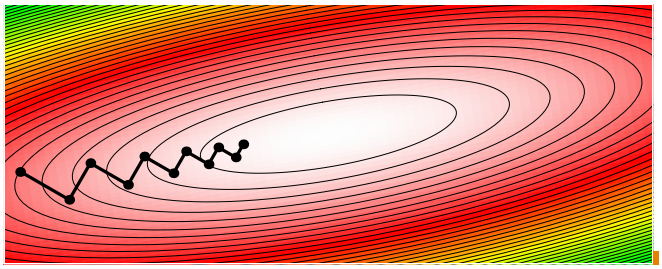
More Haste Less Speed

- Increasing the step size, just a little further, increases the rate of converge in one direction, but . . .



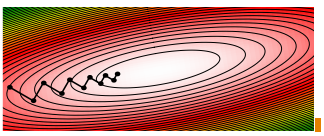
Line Minimisation

- We can systematically seek the minimum along a line of the gradient



Zig-Zag

- Note that in high dimensions gradient descent tends to zigzag



- If we computed the Hessian and used Newton's method we would jump straight to the minimum if we were in a quadratic potential
- However computing the Hessian is time consuming and misleading if we are not in a quadratic potential (i.e. far from the optimum)

Better Optimisation Algorithms

- Good optimisation algorithms often compute an approximation of the Hessian
- E.g. Conjugate gradient
  - ⋆ Performs Line Minimisation
  - ⋆ Uses gradient, but does not go along it
  - ⋆ For a quadratic minimum in d dimensions it reaches the minimum in d steps
- E.g. Levenberg-Marquardt
  - ⋆ Used on least squares problem only
  - ⋆ Uses linear approximation of function to approximate Hessian
  - ⋆ Adapts from hill-climbing to Newton method
  - ⋆ Avoids line-minimisation

Levenberg-Marquardt

- Want to minimise $\|\boldsymbol{\epsilon}(\boldsymbol{w})\|^2$ where $\epsilon_i(\boldsymbol{w}) = f(\boldsymbol{x}_i|\boldsymbol{w}) - y_i$
- Use linear approximation

$$\epsilon_i(\boldsymbol{w}) \approx \epsilon_i(\boldsymbol{w}^{(k)}) + (\boldsymbol{w} - \boldsymbol{w}^{(k)})\boldsymbol{\nabla}\epsilon_i(\boldsymbol{w}^{(k)})$$

  with $\boldsymbol{\nabla}\epsilon_i(\boldsymbol{w}^{(k)}) = \boldsymbol{\nabla} f(\boldsymbol{x}_i|\boldsymbol{w}^{(k)})$
- Solve quadratic minimisation of approximate error $\text{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w})$ with $\mathbf{J} = \boldsymbol{\nabla}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$

$$\begin{aligned} L_{approx}(\boldsymbol{w}) &= \|\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) + \mathbf{J}(\boldsymbol{w} - \boldsymbol{w}^{(k)})\|^2 \\ &= \boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})^\mathsf{T}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) + 2(\boldsymbol{w} - \boldsymbol{w}^{(k)})^\mathsf{T}\mathbf{J}^\mathsf{T}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)}) \\ &\quad + (\boldsymbol{w} - \boldsymbol{w}^{(k)})^\mathsf{T}\mathbf{J}^\mathsf{T}\mathbf{J}(\boldsymbol{w} - \boldsymbol{w}^{(k)}) \end{aligned}$$

Trust Region

- Solution given by $\boldsymbol{\nabla}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) = 0$ gives

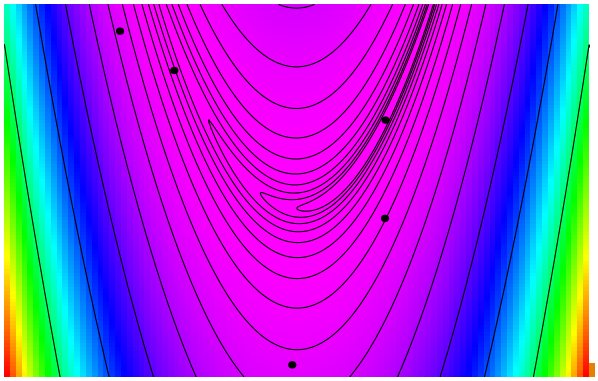
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^\mathsf{T}\mathbf{J}\right)^{-1}\mathbf{J}^\mathsf{T}\boldsymbol{\epsilon}(\boldsymbol{w}^{(k)})$$

- Can lead us in the wrong direction
- Instead use $\boldsymbol{w}^{(k+1)} = \text{argmin}_{\boldsymbol{w}} L_{approx}(\boldsymbol{w}) + \nu\|\boldsymbol{w} - \boldsymbol{w}^{(k)}\|^2$

$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^\mathsf{T}\mathbf{J} + \nu\mathbf{I}\right)^{-1}\mathbf{J}^\mathsf{T}\boldsymbol{\epsilon}(\boldsymbol{w})$$

- ν limits the step size
- If predicted reduction in error is accurate then reduce ν, else if predicted reduction in error is very poor increase ν

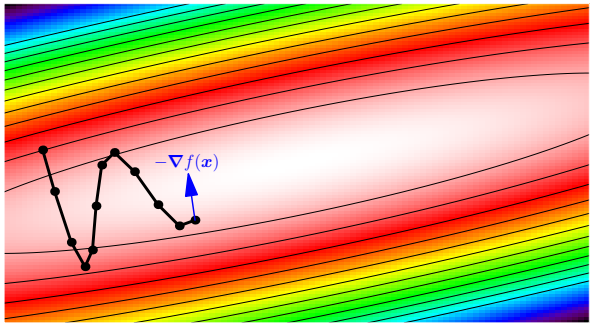
$\epsilon_1 = 10(x_2 - x_1^2)$ **and** $\epsilon_2 = 1 - x_1$



Summary

- There are some **non-gradient methods** (Nelder Mead, evolutionary strategies, Powell's method), but in very high dimensions these are not very competitive
- There are **gradient methods** (first order methods) that suffer from the problem of having to choose a single step size with conflicting requirements in different directions
- **Newton's method** (a second order method) requires computing the Hessian matrix, gives very fast convergent, but can take you in the wrong direction if you are not sufficiently close to a minimum
- There exist a number of **pseudo-Newton methods** (conjugate gradient, Levenberg-Marquardt, etc.) that approximates Newton's method often without explicitly computing the Hessian

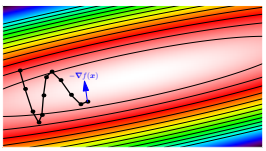
Advanced Machine Learning
Stochastic Gradient Descent



SGD, momentum, step size, ADAM

Outline

1. SGD
2. Momentum
3. Loss Landscapes



The Next Step

- In most machine learning problems we design a network and a loss function
- The rest requires simple optimisation
- Although Newton and Quasi-Newton methods ensure rapid convergence for many tasks this apparent advantage is delusionary
- Since the raise of deep learning, gradient descent and its variants have become dominant

Mini-Batch Learning

- Rather than computing the gradient of the loss function, e.g.

$$\nabla L_{\mathcal{D}}(\boldsymbol{w}) = \nabla \sum_{(\boldsymbol{x},y)\in\mathcal{D}} L\big(f(\boldsymbol{x}|\boldsymbol{w}),y\big)^2$$

- We often compute an approximation

$$\nabla L_{\mathcal{B}}(\boldsymbol{w}) = \nabla \sum_{(\boldsymbol{x},y)\in\mathcal{B}} L\big(f(\boldsymbol{x}|\boldsymbol{w}),y\big)^2$$

- where $\mathcal{B}\subset\mathcal{D}$ is a randomly sampled subset of the training set
- Usually $|\mathcal{B}| \ll |\mathcal{D}|$

Stochastic Gradient Descent

- Stochastic gradient descent (SGD) follows the gradient of mini-batches
- Computationally this is efficient because it is much quicker to compute $\nabla L_{\mathcal{B}}(\boldsymbol{w})$ than $\nabla L_{\mathcal{D}}(\boldsymbol{w})$
- The batch gradients add noise (are stochastic) as we are only sampling the dataset
- This noise might help escape local-minima (but I'm not sure there is compelling evidence for this)
- Taking many smaller steps of approximate gradients reduces the likelihood of divergence

Automatic Differentiation

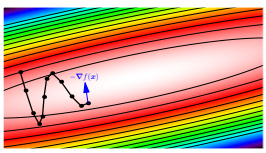
- For gradient descent we need to compute the gradient
- For most of my life this was just a pain
- You guys have it easy, modern deep learning frameworks do this for you automatically
- This is a *game changer*! We are no longer afraid of large models
- We can differentiate through algorithms allowing us to training incredible sophisticated networks

Convergence

- Asymptotic convergence of gradient descent (and SGD) is slower than quasi-Newton methods
- But there are three reasons why we don't really care
  - For complex models we spend a lot of time before we are close to a quadratic minima where asymptotic convergence kicks in
  - These days we often use ReLUs (rectified linear units) which are non-analytic. The convergence we discussed depends on a Taylor expansion that assumes analyticity
  - We want to minimise the generalisation error, but are only minimising a poor surrogate, namely the training error

Outline

1. SGD
2. Momentum
3. Loss Landscapes

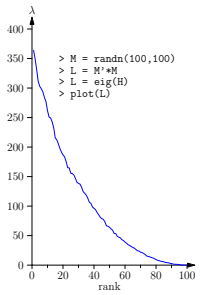


Step Size

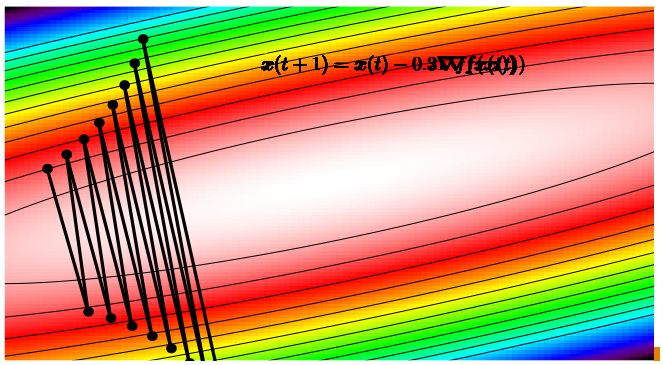
- The optimal step size depends on the gradient and the curvature (second derivative)
- For a quadratic minima the minimum is given by

$$x^* = x - \frac{f'(x)}{f''(x)}, \quad \boldsymbol{x}^* = \boldsymbol{x} - \mathbf{H}^{-1}\nabla f(\boldsymbol{x})$$

- In high dimensions the Hessian $\mathbf{H}$ will have a spectrum of eigenvalues
- This means there are different scales



Gradient Descent



Avoiding Divergence

- We saw in the last lecture that gradient descent can actually diverge from a local optimum if you have a too big step size
- If we use too large a step size we quickly get NaN errors
- If we only use SGD the step size is determined by the largest eigenvalue of the Hessian
- This seems impossibly slow, but weirdly straightforward SGD is often used in deep learning (although nearly always with momentum)

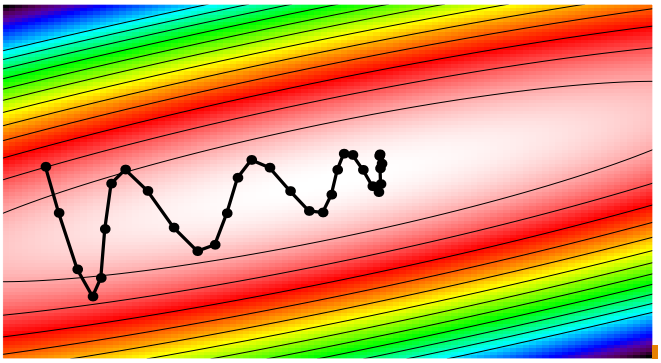
Momentum

- In high dimensions we zig-zag:
  - stepping consistently in directions with low curvature
  - jumping backwards and forwards past the minimum in directions with high curvature
- By introducing "momentum" we can increase our steps in low curvature directions and decrease it in high curvature directions

$$\boldsymbol{v}(t+1) = \gamma \boldsymbol{v}(t) - \eta \nabla f(\boldsymbol{x}(t))$$
$$\boldsymbol{x}(t+1) = \boldsymbol{x}(t) + \boldsymbol{v}(t+1)$$

$0 < \gamma < 1$ is a damping term

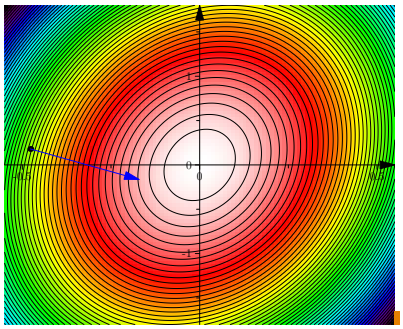
Gradient Descent with Momentum



Adaptive Methods

- A major difficulty of high dimensional optimisation is the existence of different scales
- That is, there are some directions were we have to move a lot (low curvature), while other directions we have to make small steps
- In adaptive methods we use a dynamic algorithm to rescale the step size of each parameter (weight)
- We can think of this as a "regularisation" that makes our basins of attraction more spherical

Rescaling Co-ordinates



AdaDelta

- We want to rescale the coordinates in proportion to the curvature in that direction
- To estimate the curvature we compute a running average of the squared gradients

$$S_i^g(t+1) = (1-\gamma) S_i^g(t) + \gamma \left(\frac{\partial L_{\mathcal{B}}(\boldsymbol{w}(t))}{\partial w_i(t)} \right)^2$$

- To estimate the relative scale of the weights we compute a running average of the squared weights

$$S_i^w(t+1) = (1-\gamma) S_i^w(t) + \gamma w_i(t)^2$$

AdaDelta

- The AdaDelta algorithm uses the update

$$w_i(t+1) = w_i(t) - \eta \sqrt{\frac{S_i^w(t+1)+\epsilon}{S_i^g(t+1)+\epsilon}} \frac{\partial L_{\mathcal{B}}(\boldsymbol{w}(t))}{\partial w_i(t)}$$

- Note that we can rescale the weights or rescale the gradients yet the update would be the same—this is one aspect of a covariant algorithm
- Because we are adaptively changing our step size we can use a single step size throughout the optimisation

Adding in Momentum

- AdaDelta doesn't use momentum (there is an argument this isn't so important as it has "regularised the landscape')'
- Nevertheless we can learn a momentum term

$$M_i(t+1) = (1-\beta)M_i(t) + \beta \frac{\partial L_{\mathcal{B}}(\boldsymbol{w}(t))}{\partial w_i(t)}$$

$$S_i(t+1) = (1-\gamma)S_i(t) + \gamma \left(\frac{\partial L_{\mathcal{B}}(\boldsymbol{w}(t))}{\partial w_i(t)}\right)^2$$

- These running averages have a lag time which we can remove

$$\hat{M}_i(t+1) = \frac{M_i(t+1)}{1-(1-\beta)^t} \qquad \hat{S}_i(t+1) = \frac{S(t+1)}{1-(1-\gamma)^t}$$

ADAM

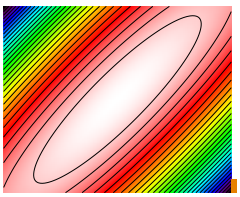
- Adaptive Moment Estimation (Adam) adapts the scale of the parameters, but also uses momentum
- Update weights

$$w_i(t+1) = w_i(t) - \frac{\eta}{\sqrt{\hat{S}_i(t+1)+\epsilon}} \hat{M}_i(t+1)$$

- ADAM and its variants are very successful in deep learning
- It is very robust so often results in a network learning which might not learn under standard SGD (with momentum)

Covariant Equations

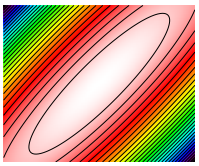
- Vector arithmetic, (matrix multiplication, addition, multiplying by a scale, computing gradients) has a covariant property that it is invariant to the coordinate system
- That is we can translate and rotate our coordinates and get the same result
- That is not true of element-wise multiplication of vectors
- Many ML algorithms do this (including ADAM and adaDelta), but they aren't invariant if we rotate our coordinates



Can't rescale coordinates

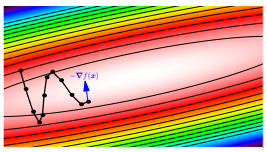
Correlated Weights

- Correlated weights will lead to skewed valleys
- Adaptive methods would be more efficient if they were rotationally invariant
- However, this would slow down the algorithm
- ADAM is a compromise between speed of implementation and effectiveness



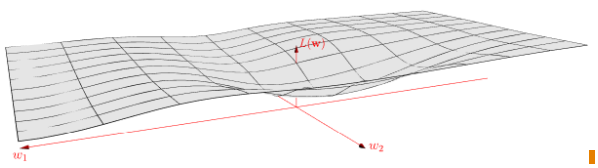
Outline

1. SGD
2. Momentum
3. **Loss Landscapes**



Loss Landscape

- A useful concept for understanding optimisation is the loss landscape
- For every value of the weights $\boldsymbol{w}$ there is some loss associated with it
- Note that this landscape is very high dimensional, so our intuition can be a bit misleading
- The landscape is also huge



Global and Local Minima

- Our objective is to minimise our loss
- This would mean finding the global minimum
- There are no algorithms that are guaranteed to find the global minimum
- The best we can hope is to find a local minimum by doing gradient descent
- In practice, our landscapes are so large that we are probably unable to find even a local minimum

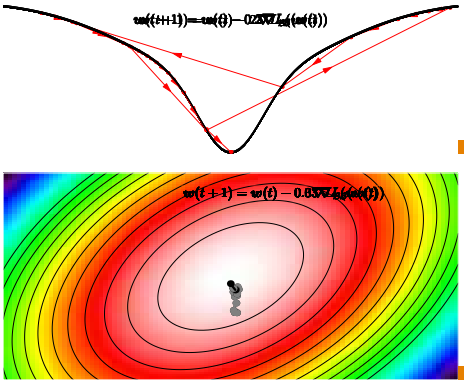
Noisy Optimisation

- Because we are using mini-batches we aren't actually following the correct gradients
- If we reduce our step size then our average gradient is closer to the true gradient
- Thus our optimisation is inherently noisy
- At least for deep learning there is no evidence that the weights stop changing even after learning for a huge amount of time

Valleys in Valleys in Valleys

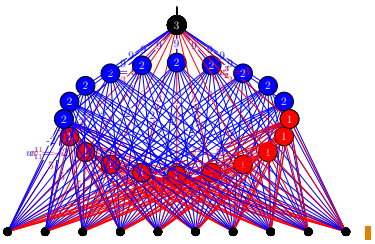
- My view of the loss landscape is that we have valleys inside bigger valleys insider bigger valleys, and so on
- If our noise is high (we use large step size), we can navigate away from local valleys and move towards the big valley
- But, we can't explore the small valleys
- Often when the rate of improvement slows down, practitioners will reduce their learning rate by a factor of 10 and the rate of improvement jumps up
- Some practitioners cycle through raising and lowering their learning rates which may speed up convergence

Reducing the Step Size



Symmetries

- When training neural networks (deep or shallow) there are typically many symmetries
- Examples come from exchanging the weights of neurons (or filters in CNNs) in two layers



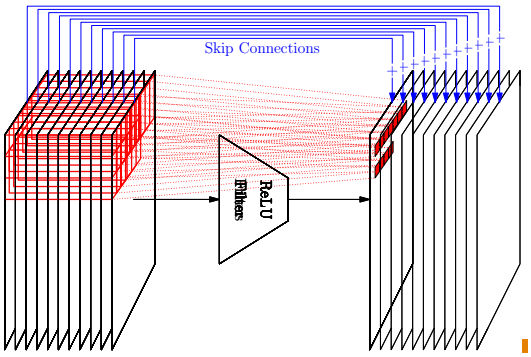
More Symmetries

- With n hidden nodes there are $n!$ symmetries
- Often if I multiply all the input and output weights to a node by -1 the network will be the same
- There are also continuous symmetries. For linear networks with two layers performing a mapping $\boldsymbol{W}_2\boldsymbol{W}_1$ then for any invertible matrix $\mathbf{U}$ (where $\mathbf{U}\mathbf{U}^{-1}=\mathbf{I}$)

$$\boldsymbol{W}_2\boldsymbol{W}_1 = \boldsymbol{W}_2\mathbf{I}\boldsymbol{W}_1 = \boldsymbol{W}_2(\mathbf{U}\mathbf{U}^{-1})\boldsymbol{W}_1 = (\boldsymbol{W}_2\mathbf{U})(\mathbf{U}^{-1}\boldsymbol{W}_1)$$

  so a network with weights $\boldsymbol{W}_2' = \boldsymbol{W}_2\mathbf{U}$ and $\boldsymbol{W}_1' = \mathbf{U}^{-1}\boldsymbol{W}_1$ will perform the same mapping
- There will be a huge space of equivalent networks

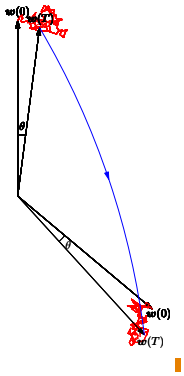
Skip Connections



- Breaks permutation symmetry. Used in Resnets, transformer, etc.

Symmetries

- The landscape potentially has a huge manifold with the same losses (neutral network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance

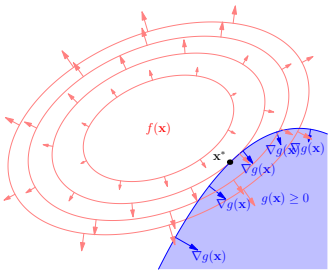


Lessons

- SGD together with automatic differentiation has revolutionised machine learning
- There are a number of techniques to get the step size right momentum, adaptive step size, ADAM
- Still need to understand that we are exploring a huge and complex loss landscape
- What works is problem specific (as always)

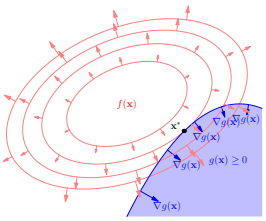
Advanced Machine Learning

Constrained Optimisation



Lagrangians, Inequalities, KKT, Linear Programming, Quadratic Programming, Duality

Outline



Optimisation with Constraints

- There are a number of important applications where we wish to minimise an objective function subject to inequality constraints
- A prominent example of this is support vector machines
- More generally there are a large number of kernel models that involve constraints
- However, constraints are ubiquitous in machine learning (e.g. in Wasserstein GANs)

Solving Constrained Optimisation Problems

- Suppose we have a problem

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{subject to } g(\boldsymbol{x}) = 0$$

- A standard procedure is to define the Lagrangian

$$\mathcal{L}(\boldsymbol{x},\alpha) = f(\boldsymbol{x}) - \alpha g(\boldsymbol{x})$$

  where α is known as a Lagrange multiplier
- In the extended space $(\boldsymbol{x},\alpha)$ we have to solve

$$\max_{\alpha}\min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x},\alpha)$$

Conditions on Optimum

- The optimisation problem is

$$\max_{\alpha}\min_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x},\alpha) \quad \text{where} \quad \mathcal{L}(\boldsymbol{x},\alpha) = f(\boldsymbol{x}) - \alpha g(\boldsymbol{x})$$

- Assuming differentiability

$$\boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x},\alpha) = \boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}) - \alpha \boldsymbol{\nabla}_{\boldsymbol{x}} g(\boldsymbol{x}) = 0$$
$$\frac{\partial \mathcal{L}}{\partial \alpha} = -g(\boldsymbol{x}) = 0$$

- The second condition is just the constraint
- But what about first condition: $\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}) = \alpha \boldsymbol{\nabla}_{\boldsymbol{x}} g(\boldsymbol{x})$?

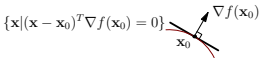
Note on Gradients

- Note that for any function $f(\boldsymbol{x})$ we can Taylor expand around $\boldsymbol{x}_0$

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + (\boldsymbol{x}-\boldsymbol{x}_0)^{\mathsf{T}}\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}_0) + \frac{1}{2}(\boldsymbol{x}-\boldsymbol{x}_0)^{\mathsf{T}}\mathbf{H}(\boldsymbol{x}-\boldsymbol{x}_0) + \ldots$$

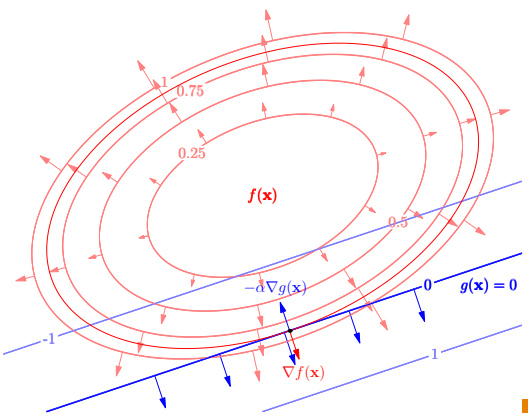
  where $\mathbf{H}$ is a matrix of second derivative known as the Hessian
- If we consider the set of points perpendicular to $\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}_0)$ which go through $\boldsymbol{x}_0$ (the tangent plane), these will have values

$$f(\boldsymbol{x}) = f(\boldsymbol{x}_0) + O(\|\boldsymbol{x}-\boldsymbol{x}_0\|^2)$$



  thus $\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x})$ is always orthogonal to the contour lines

Constrained Optima



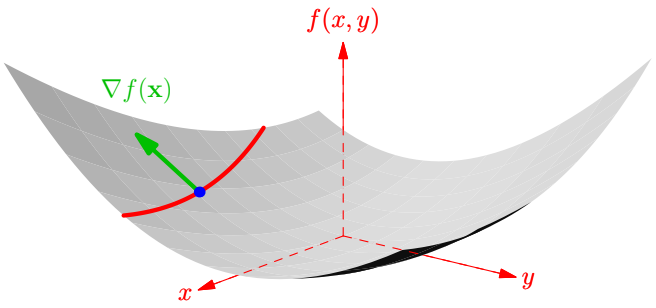
Example

- Minimise $f(\boldsymbol{x}) = x^2 + 2y^2 - xy$
- Subject to $g(\boldsymbol{x}) = x - 2y - 3 = 0$
- Writing $\mathcal{L} = f(\boldsymbol{x}) - \alpha g(\boldsymbol{x})$
- Condition for minima is $\boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{L} = 0$

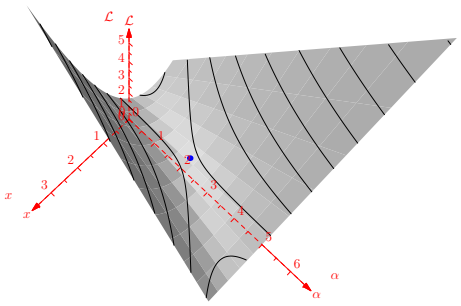
$$\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{pmatrix} 2x - y \\ -x + 4y \end{pmatrix} = \alpha \boldsymbol{\nabla}_{\boldsymbol{x}} g(\boldsymbol{x}) = \alpha \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

  and $\frac{\partial \mathcal{L}}{\partial \alpha} = -g(\boldsymbol{x}) = -x + 2y + 3 = 0$
- Solving simultaneous equations gives minima at $(x,y) = (\frac{3}{4}, -\frac{9}{8})$ with $\alpha = \frac{21}{8}$

Surface



Saddle-Point $y = -9/8$



Multiple Constraints

- Given an optimisation problem with multiple constraints

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{subject to } g_k(\boldsymbol{x}) = 0 \text{ for } k = 1, 2, \ldots, m$$

- We introduce multiple Lagrange multipliers

$$\mathcal{L}(\boldsymbol{x}, \boldsymbol{\alpha}) = f(\boldsymbol{x}) - \sum_{k=1}^{m} \alpha_k g_k(\boldsymbol{x})$$

- The condition for an optima is $\boldsymbol{\nabla}_{\boldsymbol{x}} \mathcal{L}(\boldsymbol{x}, \boldsymbol{\alpha}) = 0$ which implies

$$\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}) = \sum_{k=1}^{m} \alpha_k \boldsymbol{\nabla}_{\boldsymbol{x}} g_k(\boldsymbol{x})$$

plus the original constraints $\frac{\partial \mathcal{L}(\boldsymbol{x}, \boldsymbol{\alpha})}{\partial \alpha_k} = -g_k(\boldsymbol{x}) = 0$

Example

- Minimise $f(\boldsymbol{x}) = x^2 + 2y^2 + 5z^2 - xy - xz$ subject to $g_1(\boldsymbol{x}) = x - 2y - z - 3 = 0$ and $g_2(\boldsymbol{x}) = 2x + 3y + z - 2 = 0$
- Writing $\mathcal{L}(\boldsymbol{x}, \alpha) = f(\boldsymbol{x}) - \alpha_1 g_1(\boldsymbol{x}) - \alpha_2 g_2(\boldsymbol{x})$
- Condition for minima is $\boldsymbol{\nabla}_{\boldsymbol{x}} \mathcal{L} = 0$ or $\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}) = \sum_{k=1}^{2} \alpha_k \boldsymbol{\nabla}_{\boldsymbol{x}} g_k(\boldsymbol{x})$

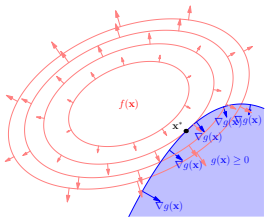
$$\begin{pmatrix} 2x - y - z \\ -x + 4y \\ 10z - x \end{pmatrix} = \alpha_1 \begin{pmatrix} 1 \\ -2 \\ -1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}$$

and $\frac{\partial \mathcal{L}}{\partial \alpha_i} = -g_i(\boldsymbol{x}) = 0$

- Solving simultaneous equations gives minima at $(\frac{37}{20}, -\frac{11}{20}, -\frac{1}{20})$ with $\alpha_1 = 3$ and $\alpha_2 = \frac{13}{20}$

Outline

1. Constrained Optimisation
2. **Inequalities**
3. Duality



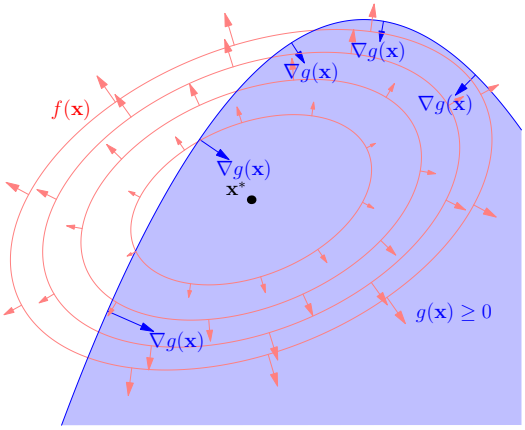
Inequality Constraints

- Suppose we have the problem

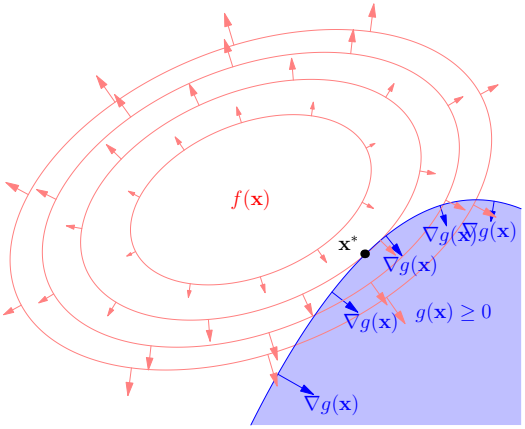
$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{subject to } g(\boldsymbol{x}) \geq 0$$

- Looks much more complicated, but
- Only two things can happen
  - Either a minimum, $\boldsymbol{x}^*$, of $f(\boldsymbol{x})$ satisfies $g(\boldsymbol{x}^*) > 0$
    - We then have an unconstrained optimisation problem
  - Otherwise, it satisfies $g(\boldsymbol{x}^*) = 0$
    - We have a constrained optimisation problem

Inside Region



On the Boundary



KKT Conditions

- To minimise $f(\boldsymbol{x})$ subject to $g(\boldsymbol{x}) \geq 0$

$$\mathcal{L}(\boldsymbol{x},\alpha) = f(\boldsymbol{x}) - \alpha g(\boldsymbol{x})$$

- Then $\nabla_{\boldsymbol{x}}\mathcal{L} = 0$ or

$$\nabla_{\boldsymbol{x}}\mathcal{L} = \nabla_{\boldsymbol{x}} f(\boldsymbol{x}) - \alpha \nabla_{\boldsymbol{x}} g(\boldsymbol{x}) = 0$$

- where either
  - $\alpha = 0$ and the solutions in the interior or
  - $\alpha > 0$ and $g(\boldsymbol{x}) = 0$, i.e. the solution is on the boundary
- These conditions are known as the Karush-Kuhn-Tucker conditions

Many Inequalities

- Given the problem

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{subject to } g_k(\boldsymbol{x}) \geq 0 \text{ for } k = 1,2,\ldots,m$$

- We introduce multiple Lagrange multipliers

$$\mathcal{L}(\boldsymbol{x},\alpha) = f(\boldsymbol{x}) - \sum_{k=1}^{m} \alpha_k g_k(\boldsymbol{x})$$

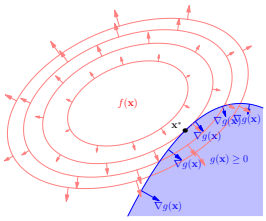
- The condition for an optima is

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{x}) = \sum_{k=1}^{m} \alpha_k \nabla_{\boldsymbol{x}} g_k(\boldsymbol{x})$$

- Plus the constraints that either $\alpha_k = 0$ or $\alpha_k > 0$ and $g_k(\boldsymbol{x}) = 0$

Outline

1. Constrained Optimisation
2. Inequalities
3. **Duality**



Solving the Lagrangian for x

- Consider minimising a function $f(\boldsymbol{x})$ subject to a set of constraints $g_i(\boldsymbol{x}) = 0$ or $g_i(\boldsymbol{x}) \leq 0$
- We can consider this a double optimisation problem

$$\max_{\alpha}\min_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha}) = \max_{\alpha}\min_{\boldsymbol{x}}\left(f(\boldsymbol{x}) + \sum_i \alpha_i g_i(\boldsymbol{x})\right)$$

where there would be constraints on α_i if we had an inequality constraint

Dual Problem

- If $f(\boldsymbol{x})$ and $g_i(\boldsymbol{x})$ are simple we can sometimes find a set of variables $\boldsymbol{x}^*(\boldsymbol{\alpha})$ that minimises the Lagrangian

$$\nabla_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x}^*(\boldsymbol{\alpha}),\boldsymbol{\alpha}) = 0$$

- This leaves us with the **dual problem**

$$\max_{\alpha}\mathcal{L}(\boldsymbol{x}^*(\boldsymbol{\alpha}),\boldsymbol{\alpha})$$

- If we had an inequality constraint $g_i(\boldsymbol{x}) \geq 0$ then we would have the additional constraint in the dual problem $\alpha_i \geq 0$

Linear Programming

- In linear programming we minimise a linear objective function $\boldsymbol{c}^{\mathsf{T}}\boldsymbol{x}$ subject to linear constraints $\boldsymbol{g}(\boldsymbol{x}) = \boldsymbol{M}\boldsymbol{x} - \boldsymbol{b} = 0$ (or $\boldsymbol{g}(\boldsymbol{x}) \geq 0$)
- The Lagrangian becomes

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha}) = \boldsymbol{c}^{\mathsf{T}}\boldsymbol{x} - \boldsymbol{\alpha}^{\mathsf{T}}(\boldsymbol{M}\boldsymbol{x} - \boldsymbol{b})$$

- An equivalent way of writing the Lagrangian is

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha}) = \boldsymbol{b}^{\mathsf{T}}\boldsymbol{\alpha} - \boldsymbol{x}^{\mathsf{T}}(\boldsymbol{M}^{\mathsf{T}}\boldsymbol{\alpha} - \boldsymbol{c})$$

- An entirely equivalent interpretation is that we maximise an objective function $\boldsymbol{b}^{\mathsf{T}}\boldsymbol{\alpha}$ subject to constraints $\boldsymbol{M}^{\mathsf{T}}\boldsymbol{\alpha} - \boldsymbol{c} = 0$ (or $\boldsymbol{M}^{\mathsf{T}}\boldsymbol{\alpha} - \boldsymbol{c} \leq 0$)

Linear Programming Example

- Suppose we eat potatoes and rice and we want to ensure that we get enough vitamin A and C

	Potatoes	Rice	Daily Requirement
Vitamin A	3	5	20
Vitamin C	5	2	24
Price	5	4	

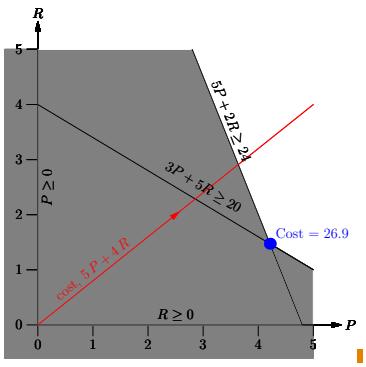
- We want to buy P kg potatoes and R kg of rice as cheaply as possible subject to fulfilling our vitamin requirement

$$\min_{P,R} 5P + 4R$$

subject to $P,R \geq 0$, $3P + 5R \geq 20$ and $5P + 2R \geq 24$

Linear Programming

- Minimise $5P + 4R$
- Subject to
  - $3P + 5R \geq 20$
  - $5P + 2R \geq 24$
  - $P,R \geq 0$



Lagrangian

- We can write the problem as a Lagrange problem

$$\min_{P,R}\max_{A,C} \quad 5P+4R-A(3P+5R-20)-C(5P+2R-24)$$

- subject to $P,R,A,B \geq 0$
- A and C are Lagrange multipliers for vitamin A and C
- We can rearrange the Lagrangian to obtain

$$\max_{A,C}\min_{P,R} \quad 20A+24C-P(3A+5C-5)-R(5A+2C-4)$$

Dual Problem

- The Lagrangian

$$\max_{A,C}\min_{P,R} \quad 20A+24C-P(3A+5C-5)-R(5A+2C-4)$$

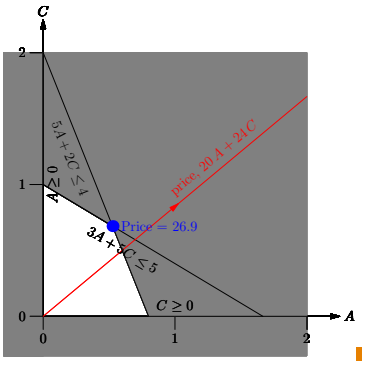
leads to the dual problem

$$\max_{A,C} 20A+24C$$
$$\text{subject to} \quad 3A+5C\leq 5 \quad 5A+2C\leq 4 \quad A,C\geq 0$$

- Consider someone selling vitamins A and C. They want to maximise the price of vitamins A and C, but their prices cannot exceed the price of the vitamins in potatoes or rice

Dual Linear Programme

- Maximise $20A+24C$
- Subject to
  - $3A+5C\leq 5$
  - $5A+2C\leq 4$
  - $A,C\geq 0$



Why?

- Why are we bothered about translating one linear programme into another?
- Sometime one form is massively easier to solve than the other
- This is because the first linear programme depends on the dimensionality of $\boldsymbol{x}$ while the second linear programme depends on the number of constraints (or dimensionality of $\boldsymbol{\alpha}$)
- This is important, for example, in Wasserstein GANs

Quadratic Programming

- A quadratic programme involves minimising a quadratic function $\boldsymbol{x}^{\mathsf{T}}\mathbf{Q}\boldsymbol{x}$ (with $\mathbf{Q}\succ 0$) subject to linear constraints $\mathbf{M}\boldsymbol{x}=\boldsymbol{b}$ (or $\mathbf{M}\boldsymbol{x}\leq\boldsymbol{b}$)
- We can define the Lagrangian

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha})=\boldsymbol{x}^{\mathsf{T}}\mathbf{Q}\boldsymbol{x}-\boldsymbol{\alpha}^{\mathsf{T}}(\mathbf{M}\boldsymbol{x}-\boldsymbol{b})$$

- Where the solution is given by $\max_{\boldsymbol{\alpha}}\min_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha})$
- If the constraints are inequality constraints then $\alpha_i\geq 0$

Solution to Quadratic Programming Problem

- Using

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha})=\boldsymbol{x}^{\mathsf{T}}\mathbf{Q}\boldsymbol{x}-\boldsymbol{\alpha}^{\mathsf{T}}(\mathbf{M}\boldsymbol{x}-\boldsymbol{b})$$

- Then

$$\boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha})=2\mathbf{Q}\boldsymbol{x}-\mathbf{M}^{\mathsf{T}}\boldsymbol{\alpha}$$

- So $\boldsymbol{\nabla}_{\boldsymbol{x}}\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha})=0$ implies

$$\boldsymbol{x}^*=\frac{1}{2}\mathbf{Q}^{-1}\mathbf{M}^{\mathsf{T}}\boldsymbol{\alpha}$$

Dual Quadratic Programming Problem

- Substituting $\boldsymbol{x}^*=\frac{1}{2}\mathbf{Q}^{-1}\mathbf{M}^{\mathsf{T}}\boldsymbol{\alpha}$ into

$$\mathcal{L}(\boldsymbol{x},\boldsymbol{\alpha})=\boldsymbol{x}^{\mathsf{T}}\mathbf{Q}\boldsymbol{x}-\boldsymbol{\alpha}^{\mathsf{T}}(\mathbf{M}\boldsymbol{x}-\boldsymbol{b})$$

- We get the dual problem

$$\max_{\boldsymbol{\alpha}}-\frac{1}{4}\boldsymbol{\alpha}^{\mathsf{T}}\mathbf{M}\mathbf{Q}^{-1}\mathbf{M}^{\mathsf{T}}\boldsymbol{\alpha}+\boldsymbol{\alpha}^{\mathsf{T}}\boldsymbol{b}$$

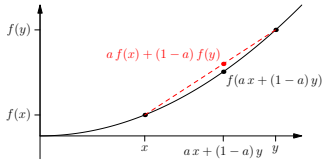
- If the constraints were inequality constraints then we have $\alpha_i\geq 0$
- We have exchanged one quadratic programme for another, but sometimes that very useful (e.g. SVMs)

Lessons

- A useful tool for performing constrained optimisation is the introduction of Lagrange multipliers
- This is particularly useful for problems with unique solutions (it will work when there are multiple solutions, but finding many saddle points is a pain)
- For inequality constraints we need to satisfy KKT conditions
- For simple situations (linear and quadratic programming) we can eliminate the original variables to obtain the dual problem

Advanced Machine Learning

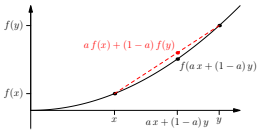
Convexity



Convex sets, convex functions, Jensen's inequality

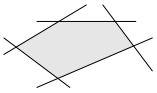
Outline

1. **Convex sets**
2. Convex functions
3. Jensen's inequality



Convex Regions

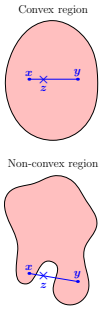
- Convex regions are familiar



- For any two points $\boldsymbol{x}$ and $\boldsymbol{y}$ in a region $\mathcal{R}$ then for any $a \in [0,1]$ if

$$\boldsymbol{z} = a\boldsymbol{x} + (1-a)\boldsymbol{y} \in \mathcal{R}$$

- then $\mathcal{R}$ is a convex region



Convex Sets

- For any set, $\mathcal{S}$, where addition and scalar multiplication is defined (e.g. a vector space) then:

  If for any two elements $\boldsymbol{x},\boldsymbol{y} \in \mathcal{S}$ and any $a \in [0,1]$

$$\boldsymbol{z} = a\boldsymbol{x} + (1-a)\boldsymbol{y} \in \mathcal{S}$$

  then $\mathcal{S}$ is said to be a convex set

Positive Semi-Definite Matrices

- Recall that a matrix $\mathbf{M}$ is positive semi-definite if for any vector $\boldsymbol{v}$

$$\boldsymbol{v}^\mathsf{T}\mathbf{M}\boldsymbol{v} \geq 0$$

  (i.e. any quadratic form of the matrix is non-negative)

- (We showed this also implies that all the eigenvalues are non-negative)

- We denote the fact that $\mathbf{M}$ is positive semi-definite by $\mathbf{M} \succeq 0$, and $\mathbf{M} \succ 0$ if it is positive definite

- The set of positive semi-definite (PSD) matrices (or kernels) form a convex set

Proof

- Consider any two arbitrarily chosen PSD matrices $\mathbf{M}_1$ and $\mathbf{M}_2$ and any $a \in [0,1]$ then let

$$\mathbf{M}_3 = a\mathbf{M}_1 + (1-a)\mathbf{M}_2$$

- Then for any vector $\boldsymbol{v}$

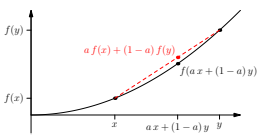
$$\begin{aligned}\boldsymbol{v}^\mathsf{T}\mathbf{M}_3\boldsymbol{v} &= \boldsymbol{v}^\mathsf{T}(a\mathbf{M}_1 + (1-a)\mathbf{M}_2)\boldsymbol{v}\\ &= a\boldsymbol{v}^\mathsf{T}\mathbf{M}_1\boldsymbol{v} + (1-a)\boldsymbol{v}^\mathsf{T}\mathbf{M}_2\boldsymbol{v}\\ &= am_1 + (1-a)m_2\end{aligned}$$

  where $m_1 = \boldsymbol{v}^\mathsf{T}\mathbf{M}_1\boldsymbol{v}$ and $m_2 = \boldsymbol{v}^\mathsf{T}\mathbf{M}_2\boldsymbol{v}$

- But $m_1, m_2 \geq 0$ since $\mathbf{M}_1, \mathbf{M}_2 \succeq 0$. Thus $am_1 + (1-a)m_2 \geq 0$ and so $\mathbf{M}_3 \succeq 0$ □

Outline

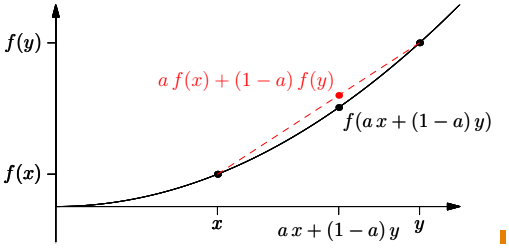
1. Convex sets
2. **Convex functions**
3. Jensen's inequality



Convex Functions

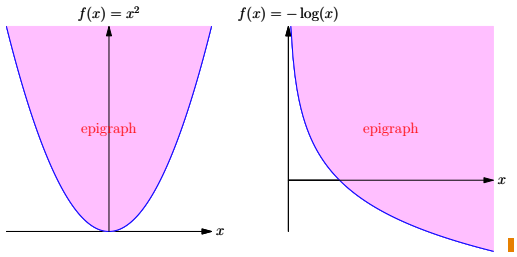
- Any function $f(x)$ is said to be a **convex function** if for any two points x and y and any $a \in [0,1]$

$$f(ax + (1-a)y) \leq af(x) + (1-a)f(y)$$



Epigraph

- The **epigraph** of a function is the area that lies above the function
- The epigraph of a convex function is a convex region



Convex-Down or Concave Functions

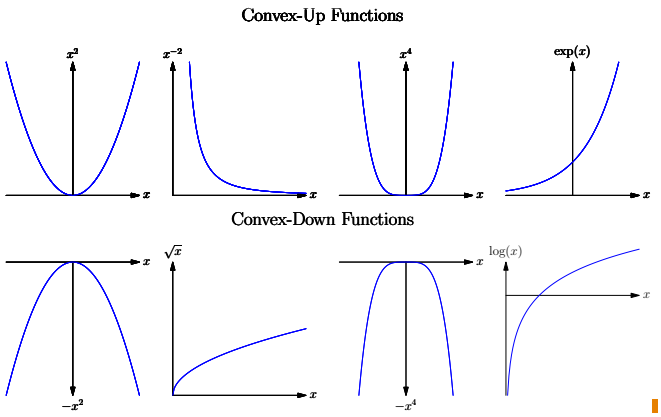
- Any function, $f(x)$, that satisfies the inverse inequality

$$f(ax + (1-a)y) \geq af(x) + (1-a)f(y)$$

for any points x and y and any $a \in [0,1]$ is said to be a **convex-down** or **concave** function

- Everything true for a convex(-up) function carries over to a convex-down function with a small modification
- If $f(x)$ is a convex-up function then $g(x) = -f(x)$ is a convex-down function
- The area that lies below a convex-down function is a convex region

Examples



Linear Functions

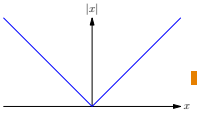
- Linear functions are given by

$$f(x) = mx + c$$

- They satisfy the **equality**

$$f(ax + (1-a)y) = af(x) + (1-a)f(y)$$

- As such they are both convex(-up) and convex-down function
- $|x|$ is a convex-up function



Strictly Convex Function

- Functions that satisfy the strict inequality (for $0 < a < 1$ and $x \neq y$)

$$f(ax + (1-a)y) < af(x) + (1-a)f(y)$$

are said to be **strictly convex functions**

- A strictly convex-down function satisfies the reverse strict inequality
- Strictly convex-(up or down) functions don't contain any linear regions

Convexity in High Dimensions

- If $f : \mathbb{R}^n \to \mathbb{R}$ (i.e. $f(\boldsymbol{x})$ maps high dimensional point $\boldsymbol{x} \in \mathbb{R}^n$ to a real value) satisfies

$$f(a\boldsymbol{x} + (1-a)\boldsymbol{y}) \leq af(\boldsymbol{x}) + (1-a)f(\boldsymbol{y})$$

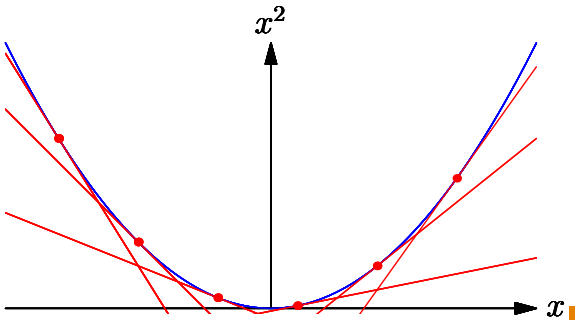
for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and any $a \in [0,1]$ then $f(\boldsymbol{x})$ is a convex function

- $\|\boldsymbol{x}\|_2^2 = \sum_i x_i^2$ is a (strictly) convex function
- $\|\boldsymbol{x}\|_1 = \sum_i |x_i|$ is a convex function

Properties of Convex Functions

- Convex functions lie on or above any tangent line

$$f(x) \geq f(x^*) + (x - x^*)f'(x^*)$$



Second Derivatives

- As $f(x)$ lies on or above its tangent line then for any $\epsilon > 0$

$$f'(x + \epsilon) \geq f'(x)$$

therefore $f''(x) = \lim_{\epsilon \to 0}(f'(x+\epsilon) - f'(x))/\epsilon \geq 0$ at all points x

- In high dimensions a convex function lies above its tangent plane

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*) + (\boldsymbol{x} - \boldsymbol{x}^*)^{\mathsf{T}} \boldsymbol{\nabla} f(\boldsymbol{x}^*)$$

- The matrix of second derivatives (the Hessian) must be positive semi-definite at all points $\boldsymbol{x}$

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1^2} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_1 \partial x_2} & \cdots \\ \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2 \partial x_1} & \frac{\partial^2 f(\boldsymbol{x})}{\partial x_2^2} & \cdots \\ \vdots & \vdots & \ddots \end{pmatrix} \succeq 0$$

Proving Convexity

- $f(x) = x^2$ is strictly convex as $f''(x) = 2 > 0$
- $f(x) = \mathrm{e}^{cx}$ is strictly convex as $f''(x) = c^2\mathrm{e}^{cx} > 0$
- $f(x) = \log(x)$ is strictly convex-down as $f''(x) = -\frac{1}{x^2} < 0$
- $f(x,y) = \frac{x^2}{y}$ is convex for $y > 0$ as

$$\mathbf{H} = \begin{pmatrix} \frac{\partial^2 f(x,y)}{\partial x^2} & \frac{\partial^2 f(x,y)}{\partial x \partial y} \\ \frac{\partial^2 f(x,y)}{\partial y \partial x} & \frac{\partial^2 f(x,y)}{\partial y^2} \end{pmatrix} = \begin{pmatrix} \frac{2}{y} & -\frac{2x}{y^2} \\ -\frac{2x}{y^2} & \frac{2x^2}{y^3} \end{pmatrix} = \frac{2}{y^3}\begin{pmatrix} y^2 & -xy \\ -xy & x^2 \end{pmatrix}$$

- Now $T = \mathrm{tr}\mathbf{H} = \frac{2}{y^3}(x^2 + y^2)$, $D = \det(\mathbf{H}) = 0$
- $\lambda_{1,2} = T/2 \pm \sqrt{T^2/4 - D} = \{0, T\} = \{0, \frac{2(x^2+y^2)}{y^3}\} \geq 0 \Rightarrow \mathbf{H} \succeq 0$

Sums of Convex Functions

- For any set of convex functions $f_1(x)$, $f_2(x)$, ... and any set of non-negative scalars a_1, a_2, ... then

$$g(x) = \sum_i a_i f_i(x)$$

is convex

- Proof

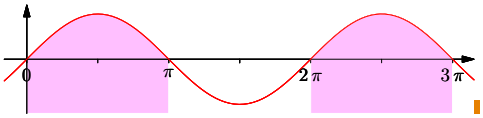
$$g''(x) = \sum_i a_i f_i''(x)$$

but $f_i''(x) \geq 0$ so $g''(x)$ is a sum on non-negative terms

- This generalises to higher dimensions as the sum of PSD matrices times positive factors is a PSD matix

Convex Functions Defined on Convex Sets

- All the properties we have discussed hold for functions defined on a convex set
- $\sin(x)$ is not generally neither convex up or down
- $\sin(x)$ for $x \in [0,\pi]$ is convex-down (its second derivative $-\sin(x)$ is less than or equal to 0 in this range)



- For a convex function defined on a non-convex set, $\mathcal{S}$, there exists points $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{S}$ such that for some $a \in [0,1]$ there will be points $\boldsymbol{z} = a\boldsymbol{x} + (1-a)\boldsymbol{y} \notin \mathcal{S}$ (the function isn't defined on such points)

Constraints

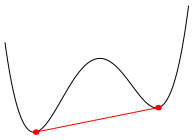
- Often we impose constraints on the set of points, e.g.

$$x_i > 0 \qquad \boldsymbol{a}^\mathsf{T}\boldsymbol{x} = b \qquad \boldsymbol{x}^\mathsf{T}\mathbf{M}\boldsymbol{x} \leq 1$$

- Linear constraints (e.g. $x_i > 0$ or $\boldsymbol{a}^\mathsf{T}\boldsymbol{x} = b$ or $\boldsymbol{a}^\mathsf{T}\boldsymbol{x} \leq b$) always define a convex region
- Multiple linear constraints always define a convex region
- Non-linear constraints may or may not define a convex region ($\{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x}^\mathsf{T}\mathbf{M}\boldsymbol{x} \leq 1, \mathbf{M} \succeq 0\}$ does while $\{\boldsymbol{x} \in \mathbb{R}^n | \boldsymbol{x}^\mathsf{T}\mathbf{M}\boldsymbol{x} \geq 1, \mathbf{M} \succeq 0\}$ doesn't)

Unique Minimum

- Strictly convex function have a unique minimum
- The existence of a local minimum would break convexity
  - ⋆ The line connecting a local minimum to a global minimum would be strictly decreasing
  - ⋆ Thus there are points next to the local minimum with lower values
  - ⋆ This is a contradiction



- This remains true if we consider convex functions that are constrained to live in a convex set

Convex Set of Minima

- If $f(\boldsymbol{x})$ is **convex** but not **strictly convex** then there might exist a convex set $\mathcal{M} \subset \mathcal{X}$ of minima such that for all $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{M}$ and any $\boldsymbol{z} \in \mathcal{X}$ we have $f(\boldsymbol{x}) = f(\boldsymbol{y}) \leq f(\boldsymbol{z})$
- This set of minima is convex, that is, if $\boldsymbol{x}, \boldsymbol{y} \in \mathcal{M}$ then for any $a \in [0,1]$ the point $\boldsymbol{z} = a\boldsymbol{x} + (1-a)\boldsymbol{y} \in \mathcal{M}$
- The sum of a convex function, $f(x)$, and a strictly convex function $g(x)$ will always be strictly convex since

$$f''(x) + g''(x) > 0$$

as $f''(x) \geq 0$ and $g''(x) > 0$

Linear Regression

- For linear regression the loss function

$$L(\boldsymbol{w}) = \|\mathbf{X}\boldsymbol{w} - \boldsymbol{y}\|^2 = \boldsymbol{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{w} - 2\boldsymbol{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\boldsymbol{y} + \boldsymbol{y}^\mathsf{T}\boldsymbol{y}$$

is convex

- Since the Hessian $\mathbf{H} = 2\boldsymbol{X}^\mathsf{T}\boldsymbol{X} \succeq 0$ (positive semi-definite) (For any vector $\boldsymbol{v}$ then $\boldsymbol{v}^\mathsf{T}\mathbf{H}\boldsymbol{v} = 2\boldsymbol{v}^\mathsf{T}\mathbf{X}^\mathsf{T}\mathbf{X}\boldsymbol{v} = 2\|\mathbf{X}\boldsymbol{v}\|^2 \geq 0$)
- If $\mathbf{H} \succ 0$ there will be a unique minima while if $\mathbf{H}$ has some zero eigenvalues there will be a family of solutions

Regularised Linear Regression

- In ridge regression we minimise a loss

$$L(\boldsymbol{w}) = \|\mathbf{X}\boldsymbol{w} - \boldsymbol{y}\|^2 + \eta\|\boldsymbol{w}\|^2 = \boldsymbol{w}^\mathsf{T}(\mathbf{X}^\mathsf{T}\mathbf{X} + \eta\mathbf{I})\boldsymbol{w} - 2\boldsymbol{w}^\mathsf{T}\mathbf{X}^\mathsf{T}\boldsymbol{y} + \boldsymbol{y}^\mathsf{T}\boldsymbol{y}$$

- Because $\|\boldsymbol{w}\|^2$ is strictly convex the loss function is strictly convex and so will have a unique solution
- Using an L_1 regulariser (Lasso)

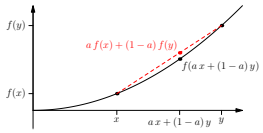
$$L(\boldsymbol{w}) = \|\mathbf{X}\boldsymbol{w} - \boldsymbol{y}\|^2 + \eta\|\boldsymbol{w}\|_1$$

again $\|\boldsymbol{w}\|_1$ is convex and so $L(\boldsymbol{w})$ will be convex

- Using an L_1 and an L_2 regulariser (elastic net) also gives a unique solution

Outline

1. Convex sets
2. Convex functions
3. **Jensen's inequality**



Jensen's Inequality

- In proving many properties of learning machines inequalities are really useful
- One of the most useful inequalities involve expectations of convex functions, this is known as **Jensen's Inequality**
- If $f(\boldsymbol{x})$ is a convex(-up) function then

$$\mathbb{E}[f(\boldsymbol{X})] \geq f(\mathbb{E}[\boldsymbol{X}])$$

- If $f(\boldsymbol{x})$ is a convex(-down) function then

$$\mathbb{E}[f(\boldsymbol{X})] \leq f(\mathbb{E}[\boldsymbol{X}])$$

Proof

- We said before that a convex function must lie on or above its tangent plane at any point $\boldsymbol{x}^*$

$$f(\boldsymbol{x}) \geq f(\boldsymbol{x}^*) + (\boldsymbol{x} - \boldsymbol{x}^*)^\mathsf{T} \boldsymbol{\nabla} f(\boldsymbol{x}^*)$$

- Taking $\boldsymbol{x}^* = \mathbb{E}[\boldsymbol{X}]$

$$f(\boldsymbol{X}) \geq f(\mathbb{E}[\boldsymbol{X}]) + (\boldsymbol{X} - \mathbb{E}[\boldsymbol{X}])^\mathsf{T} \boldsymbol{\nabla} f(\mathbb{E}[\boldsymbol{X}])$$

- Taking expectations of both sides

$$\begin{aligned}\mathbb{E}[f(\boldsymbol{X})] &\geq f(\mathbb{E}[\boldsymbol{X}]) + (\mathbb{E}[\boldsymbol{X}] - \mathbb{E}[\boldsymbol{X}])^\mathsf{T} \boldsymbol{\nabla} f(\mathbb{E}[\boldsymbol{X}]) \\ &= f(\mathbb{E}[\boldsymbol{X}]) \qquad \square\end{aligned}$$

Simple Proofs with Jensen's Inequality

- Since $f(x) = x^2$ is convex by Jensen's inequality

$$\mathbb{E}\left[X^2\right] \geq (\mathbb{E}[X])^2 \quad \text{or} \quad \mathbb{E}\left[X^2\right] - \mathbb{E}[X]^2 \geq 0$$

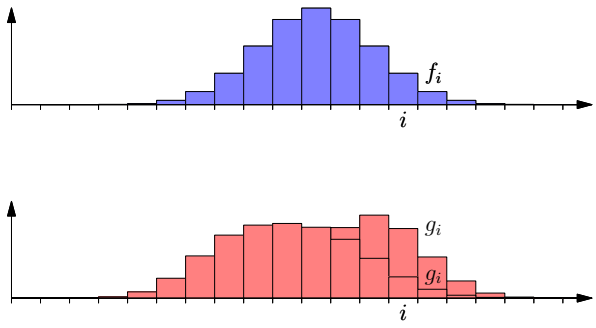
(i.e. variance are non-negative)

- The KL-divergence $\mathrm{KL}(f\|g)$ between two categorical probability distributions $(f_1, f_2, \ldots)$ and $(g_1, g_2, \ldots)$ is define as

$$\mathrm{KL}(f\|g) = -\sum_i f_i \log\left(\frac{g_i}{f_i}\right)$$

(note $f_i, g_i \geq 0$ and $\sum_i f_i = \sum_i g_i = 1$)

Kullback-Leibler Divergence



$$\mathrm{KL}(\boldsymbol{f}\|\boldsymbol{g}) = -\sum_{i=1}^{n} f_i \log\left(\frac{g_i}{f_i}\right) = 0.2355$$

Proof of Gibbs' Inequality

- To show that $\mathrm{KL}(f\|g) \geq 0$ (Gibbs' inequality) we note that since the logarithm is a convex-down function

$$\begin{aligned}\mathrm{KL}(f\|g) = -\sum_i f_i \log\left(\frac{g_i}{f_i}\right) &= \mathbb{E}_f\left[-\log\left(\frac{g_i}{f_i}\right)\right] \\ &\geq -\log\left(\mathbb{E}_f\left[\frac{g_i}{f_i}\right]\right) \\ &= -\log\left(\sum_i f_i \frac{g_i}{f_i}\right) = -\log\left(\sum_i g_i\right) = -\log(1) = 0\end{aligned}$$

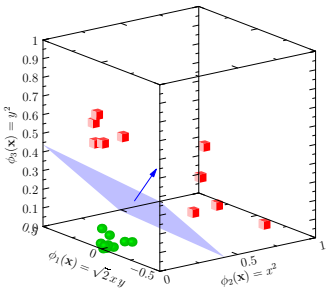
- We will meet KL-divergences later on

Lessons

- Although we haven't talked much about machine learning, convexity is heavily used in many machine learning applications
- A lot of ML algorithms involve convex functions e.g. SVMs
- As such they will have a unique minimum (or a convex set of minima)
- Convexity is an elegant idea which is relatively easy to prove theorems about
- One of the most useful tools is Jensen's inequality

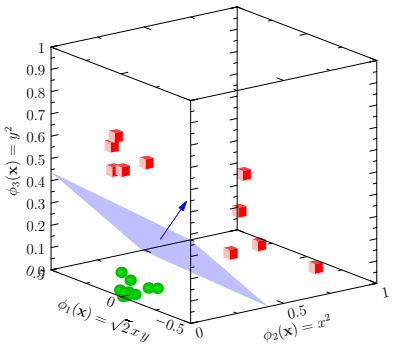
Advanced Machine Learning

Support Vector Machines



Support Vector Machines, maximum margins

Outline

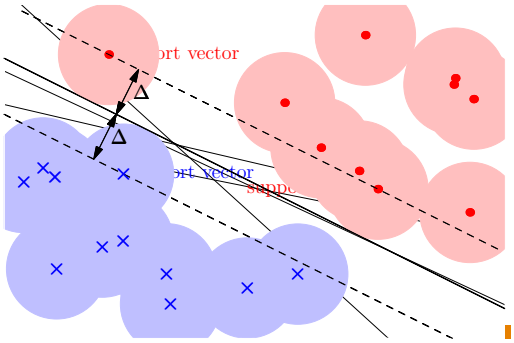


Support Vector Machines

- Support vector machines, when used right, often have the best generalisation results
- They are typically used on numerical data, but can and have been adapted to text, sequences, etc.
- Although not as trendy as deep learning, they will often be the method of choice on small data sets
- They subtly regularise themselves, choosing a solution that generalises well from a host of different solutions

Linear Separation of Data

- SVMs classify linearly separable data



- Finds maximum-margin separating plane

Extended Feature Space

- To increase the likelihood of linear-separability we often use a high-dimensional mapping

$$\boldsymbol{x} = (x_1, x_2, \ldots, x_p)^\mathsf{T} \rightarrow \boldsymbol{\phi}(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots, \phi_r(\boldsymbol{x}))^\mathsf{T}$$

$r \gg p$

- Finding the maximum margin hyper-plane is time consuming in "primal" form if r is large
- We can work in the "dual" space of patterns, then we only need to compute inner-products

$$\langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle = \boldsymbol{\phi}(\boldsymbol{x}_i)^\mathsf{T} \boldsymbol{\phi}(\boldsymbol{x}_j) = \sum_{k=1}^{r} \phi_k(\boldsymbol{x}_i)\phi_k(\boldsymbol{x}_j)$$

Kernel Trick

- If we choose a **positive semi-definite** kernel function $K(\boldsymbol{x}, \boldsymbol{y})$ then there exists functions $\boldsymbol{\phi}(\boldsymbol{x}) = (\phi_k(\boldsymbol{x}) | k = 1, 2, \ldots, r)$, such that

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle$$

(like an eigenvector decomposition of a matrix)

- Never need to compute $\phi_k(\boldsymbol{x}_i)$ explicitly as we only need the inner-product $\langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle = K(\boldsymbol{x}_i, \boldsymbol{x}_j)$ to compute maximum margin separating hyper-plane
- Sometimes $\boldsymbol{\phi}(\boldsymbol{x}_i)$ is an infinite dimensional vector so it is good we don't have to compute all the elements!

Kernel Functions

- Kernel functions are symmetric functions of two variable
- Strong restriction: *positive semi-definite*
- Examples

Quadratic kernel: $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \left(\boldsymbol{x}_1^\mathsf{T} \boldsymbol{x}_2\right)^2$

Gaussian (RBF) kernel: $K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \mathrm{e}^{-\gamma \|\boldsymbol{x}_1 - \boldsymbol{x}_2\|^2}$

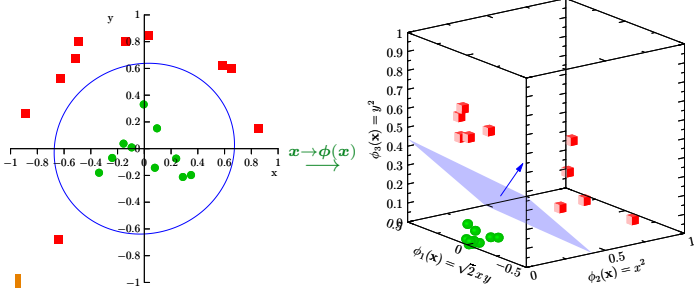
- Consider the mapping

$$\boldsymbol{x}_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \rightarrow \boldsymbol{\phi}(\boldsymbol{x}_i) = \begin{pmatrix} x_i^2 \\ y_i^2 \\ \sqrt{2} x_i y_i \end{pmatrix}$$

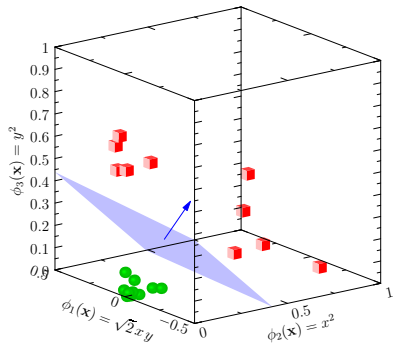
Non-linear Separation of Data

$$K(\boldsymbol{x}_1, \boldsymbol{x}_2) = \begin{pmatrix} x_1^2 & y_1^2 & \sqrt{2} x_1 y_1 \end{pmatrix} \begin{pmatrix} x_2^2 \\ y_2^2 \\ \sqrt{2} x_2 y_2 \end{pmatrix} = x_1^2 x_2^2 + y_1^2 y_2^2 + 2 x_1 y_1 x_2 y_2$$

$$= (x_1 x_2 + y_1 y_2)^2 = \left(\boldsymbol{x}_1^T \boldsymbol{x}_2\right)^2$$



Outline

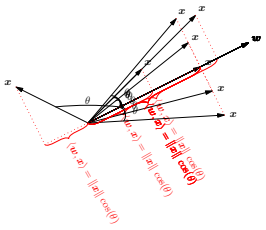


Inner Product

- Recall the inner or dot product in $\mathbb{R}^n$

$$\langle \boldsymbol{x}, \boldsymbol{y} \rangle = \boldsymbol{x} \cdot \boldsymbol{y} = \boldsymbol{x}^\mathsf{T}\boldsymbol{y} = \sum_{i=1}^{n} x_i y_i = \|\boldsymbol{x}\| \, \|\boldsymbol{y}\| \cos(\theta)$$

- If $\|\boldsymbol{w}\| = 1$ then $\langle \boldsymbol{x}, \boldsymbol{w} \rangle = \|\boldsymbol{x}\| \cos(\theta)$



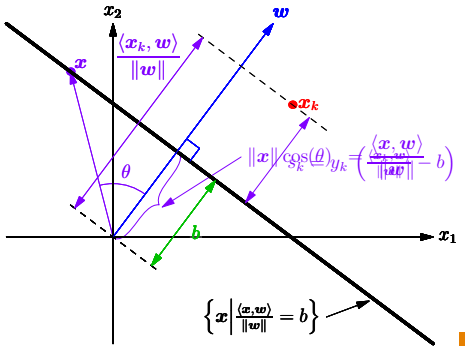
Maximise Margin

- Consider a linearly separable set of data
  - $\mathcal{D} = \{(\boldsymbol{x}_k, y_k)\}_{k=1}^m$
  - $y_k \in \{-1, 1\}$
- Our task is to find a separating plane defined by the orthogonal vector $\boldsymbol{w}$ and a threshold b such that

$$y_k \left(\frac{\langle \boldsymbol{w}, \boldsymbol{x}_k \rangle}{\|\boldsymbol{w}\|} - b \right) \geq \Delta$$

where Δ is the margin

Distance to hyperplanes



Constrained Optimisation

- Wish to find $\boldsymbol{w}$ and b to maximise Δ subject to constraints

$$y_k \left(\frac{\langle \boldsymbol{w}, \boldsymbol{x}_k \rangle}{\|\boldsymbol{w}\|} - b \right) \geq \Delta \quad \text{for all } k = 1, 2, \ldots, m$$

- If we divide through by Δ

$$y_k \left(\frac{\langle \boldsymbol{w}, \boldsymbol{x}_k \rangle}{\Delta \|\boldsymbol{w}\|} - \frac{b}{\Delta} \right) \geq 1 \quad \text{for all } k = 1, 2, \ldots, m$$

- Define $\hat{\boldsymbol{w}} = \boldsymbol{w}/(\Delta \|\boldsymbol{w}\|)$ and $\hat{b} = b/\Delta$

$$y_k \left(\langle \hat{\boldsymbol{w}}, \boldsymbol{x}_k \rangle - \hat{b} \right) \geq 1$$

Quadratic Programming Problem

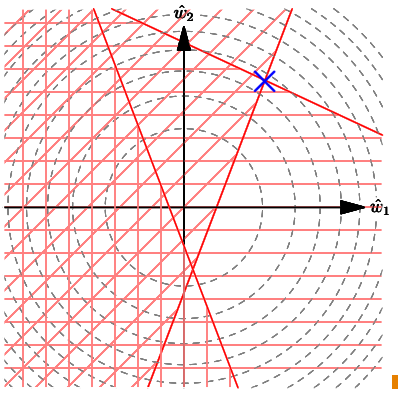
- Note that as $\hat{\boldsymbol{w}} = \boldsymbol{w}/(\Delta \|\boldsymbol{w}\|)$

$$\|\hat{\boldsymbol{w}}\| = \left\| \frac{\boldsymbol{w}}{\Delta \|\boldsymbol{w}\|} \right\| = \frac{1}{\Delta \|\boldsymbol{w}\|} \|\boldsymbol{w}\| = \frac{1}{\Delta}$$

- Minimising $\|\hat{\boldsymbol{w}}\|^2$ is equivalent to maximising the margin Δ
- Can write the optimisation problem as a *quadratic programming problem*

$$\min_{\hat{\boldsymbol{w}}, \hat{b}} \frac{\|\hat{\boldsymbol{w}}\|^2}{2} \quad \text{subject to } y_k \left(\langle \hat{\boldsymbol{w}}, \boldsymbol{x}_k \rangle - \hat{b} \right) \geq 1 \text{ for all } k = 1, 2, \ldots, m$$

Quadratic Programming in SVMs

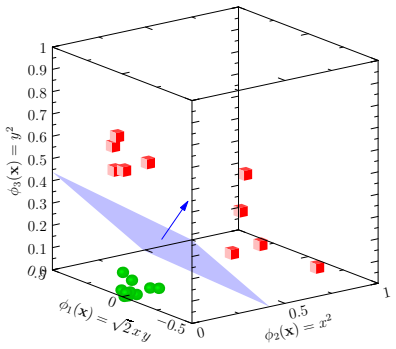


Quadratic Programming

- We have a quadratic programming problem for the weights $\hat{\boldsymbol{w}} = (\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_p)$ and bias $\hat{b}$ and m constraints
- This is a classic but fiddly optimisation problems
- It can be solved in $O(p^3)$ time (it involves inverting matrices) (phew it is not NP-complete!)
- We will see that there is an equivalent dual problem which allows us to use the kernel trick with time complexity $O(m^3)$

Outline

1. The Big Picture
2. Maximum Margins
3. Duality
4. Practice



Extended Feature Space

- We can generalise the SVM if we map all our features vectors to an extended feature space

$$\boldsymbol{x} \to \boldsymbol{\phi}(\boldsymbol{x})$$

- The components of $\boldsymbol{\phi}(\boldsymbol{x})$ will typically be (non-linear) functions of $\boldsymbol{x}$ (e.g. $\phi_1(\boldsymbol{x}) = x_1^2, \phi_2(\boldsymbol{x}) = x_2^2, \phi_3(\boldsymbol{x}) = \sqrt{2}x_1x_2$)
- We are free to choose whatever mappings we like
- There may be many more components of $\boldsymbol{\phi}(\boldsymbol{x})$ than of $\boldsymbol{x}$ making it easier to find a linear separation of the two classes
- But in the extended feature space (involving $\boldsymbol{\phi}(\boldsymbol{x}) = \big(\phi_1(\boldsymbol{x}), \phi_2(\boldsymbol{x}), \ldots, \phi_r(\boldsymbol{x})\big)$) the time complexity is $O(r^3)$

Lagrangian

- In the extended feature space we can find a separating plane (given by $\boldsymbol{w}$ and b) with maximum margine by solving the problem

$$\min_{\boldsymbol{w},b} \frac{\|\boldsymbol{w}\|^2}{2} \quad \text{subject to } y_k\big(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}_k)\rangle - b\big) \geq 1 \text{ for all } k = 1, 2, \ldots, m$$

- We can write this as a Lagrange problem

$$\min_{\boldsymbol{w},b} \max_{\boldsymbol{\alpha} \geq \boldsymbol{0}} \mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha})$$

where

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{\|\boldsymbol{w}\|^2}{2} - \sum_{k=1}^m \alpha_k \Big(y_k \big(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}_k)\rangle - b \big) - 1 \Big)$$

subject to $\alpha_k \geq 0$

Obtaining the Dual Form of the Problem

- Differentiating the Lagrangian with respect to $\boldsymbol{w}$

$$\mathcal{L}(\boldsymbol{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{k=1}^m \alpha_k \Big(y_k \big(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}_k)\rangle - b \big) - 1 \Big)$$

- $\boldsymbol{\nabla}_{\boldsymbol{w}} \mathcal{L} = \boldsymbol{w} - \sum_{k=1}^m \alpha_k y_k \boldsymbol{\phi}(\boldsymbol{x}_k) = 0$ implies that $\boldsymbol{w}^* = \sum_{k=1}^m \alpha_k y_k \boldsymbol{\phi}(\boldsymbol{x}_k)$
- $\frac{\partial \mathcal{L}}{\partial b} = \sum_{k=1}^m \alpha_k y_k = 0$ implies $\sum_{k=1}^m \alpha_k y_k = 0$
- Substituting back into the Lagrangian

$$\max_{\boldsymbol{\alpha} \geq 0} \sum_{k=1}^m \alpha_k - \frac{1}{2} \sum_{k,l=1}^m \alpha_k \alpha_l y_k y_l \langle \boldsymbol{\phi}(\boldsymbol{x}_k), \boldsymbol{\phi}(\boldsymbol{x}_l)\rangle$$

The Dual Problem

- The dual problem is now to find α_k's that maximise

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{k=1}^m \alpha_k - \frac{1}{2} \sum_{k,l=1}^m \alpha_k \alpha_l y_k y_l \langle \boldsymbol{\phi}(\boldsymbol{x}_k), \boldsymbol{\phi}(\boldsymbol{x}_l)\rangle$$

- subject to constraints

$$\sum_{k=1}^m \alpha_k y_k = 0 \qquad \forall k = 1, 2, \ldots, m \quad \alpha_k \geq 0$$

- The Hessian of $\mathcal{L}(\boldsymbol{\alpha})$ has elements $H_{kl} = -y_k y_l \langle \boldsymbol{\phi}(\boldsymbol{x}_k), \boldsymbol{\phi}(\boldsymbol{x}_l)\rangle$ so $\boldsymbol{v}^{\mathsf{T}} \mathbf{H} \boldsymbol{v} = -\big\|\sum_k v_k y_k \phi_k(\boldsymbol{x}_k)\big\|^2 \leq 0$ (note this is negative semi-definite so there is a unique maximum)

Kernel Trick

- We will show in the next lecture that if $K(\boldsymbol{x}, \boldsymbol{y})$ is a positive semi-definite function then it can always be written as

$$K(\boldsymbol{x}, \boldsymbol{y}) = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{y})\rangle$$

- As $\langle \boldsymbol{\phi}(\boldsymbol{x}_k), \boldsymbol{\phi}(\boldsymbol{x}_l)\rangle$ appears in the dual problem we can express the dual problem as finding α_k's that maximise

$$\mathcal{L}(\boldsymbol{\alpha}) = \sum_{k=1}^m \alpha_k - \frac{1}{2} \sum_{k,l=1}^m \alpha_k \alpha_l y_k y_l K(\boldsymbol{x}_k, \boldsymbol{x}_l)$$

- We therefore never have to compute $\boldsymbol{\phi}(\boldsymbol{x})$

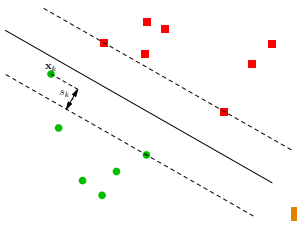
Sequential Minimal Optimisation

- One of the most efficient techniques for training SVMs is *Sequential Minimal Optimisation* or SMO
- This takes two Lagrange multipliers α_i and α_j and adjusts them to maximise the dual objective function
- This is very quick as it can be done in closed form
- Note that because $\sum_{k=1}^m y_k \alpha_k = 0$ we have to change at least two variables at the same time
- A heuristic is used to choose good pairs of α's to optimise
- Run until close to the optimum

Soft Margins

- We can relax the margin constraints by introducing *slack variables*, $s_k \geq 0$

$$y_k(\langle \boldsymbol{x}_k, \boldsymbol{w}\rangle - b) \geq 1 - s_k$$



- Minimise $\frac{\|\boldsymbol{w}\|^2}{2} + C\sum_{k=1}^n s_k$
- Larger C punishes slack variables more

Dual Problem with Slack Variables

- The Lagrangian with slack variables is

$$\mathcal{L} = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{k=1}^{m} s_k - \sum_{k=1}^{m}\alpha_k\left(y_k\left(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}_k)\rangle - b\right) - 1 + s_k\right) - \sum_{k=1}^{m}\beta_k s_k$$

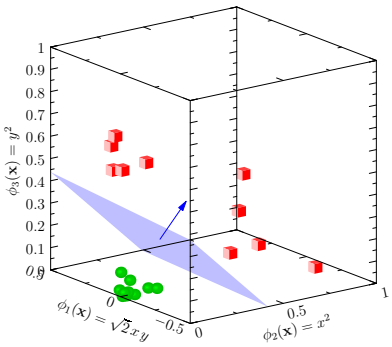
  where β_k are Lagrange multipliers that ensure $s_k \geq 0$ (note that $\beta_k \geq 0$—this is the KKT condition)

- Now minimising with respect to s_i

$$\frac{\partial \mathcal{L}}{\partial s_i} = C - \alpha_i - \beta_i = 0$$

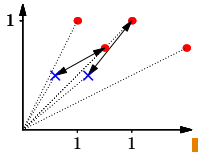
- Or $\alpha_i = C - \beta_i$. Since $\beta_i \geq 0$ the constraint is $\alpha_i \leq C$ (recall also $\alpha_i \geq 0$)

Outline



Getting SVMs to Work Well

- SVMs rely on distances between data points
- These will change relative to each other if we rescale some features but not other—giving different maximum-margin hyper-planes



- If we don't know what features are important (most often the case), then it is worth scaling each feature (for example, so their range is between 0 and 1 or their variance is 1)

Optimising C

- Recall that we can introduce soft-margins using slack variables where we minimise $\frac{\|\hat{\boldsymbol{w}}\|^2}{2} + C\sum_{k=1}^{m} s_k$ subject to constraints
- In practice it can make a huge difference to the performance if we change C
- Optimal C values changes by many orders of magnitude e.g. 2^{-5}–2^{15}
- Typically optimised by a grid search (start from 2^{-5} say and double until you reach 2^{15})
- Measure performance on a validation set

Choosing the Right Kernel Function

- There are kernels design for particular data types (e.g. string kernels for text or biological sequences)
- For numerical data, people tend to look at using no kernel (linear SVM), a radial basis function (Gaussian) kernel or polynomial kernels
- Kernels often come with parameters, e.g. the popular radial basis function kernel

$$K(\boldsymbol{x}, \boldsymbol{y}) = \mathrm{e}^{-\gamma\|\boldsymbol{x}-\boldsymbol{y}\|^2}$$

- Optimal γ values range over 2^{-15}–2^{3}

Multi-Class Problems

- By construction SVMs separate only two classes
- If we have a multi-class problem we have to use multiple SVMs
- There are two major ways practitioners do this

  One-versus-all: for each class, train a separate classify to determine that class versus all others

  All-pairs: train a classify for all pairs of classes
- In both cases choose the class which the classifier is most certain about
- Beware SVMs don't like imbalanced datasets

SVM Libraries

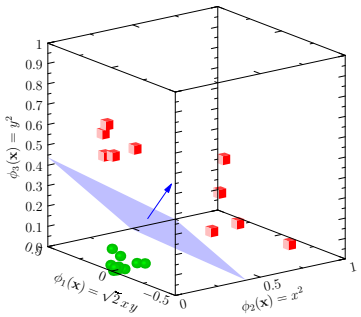
- Although SVMs have unique solutions, they require very well written optimisers
- If you have a large data set they can be very slow
- There are good libraries out there: svmlib, svm-lite, (now old), scikit-learn, etc.
- These will often automate normalisation of data and grid search for parameters

Conclusions

- We've seen how SVMs work
- We've learnt how to use them
- We've seen that we can find the maximum margin hyper-plane by solving a quadratic programming problem (with a unique solution)
- This is a convex optimisation problem with a unique optimum
- The **dual problem** of an SVM is particularly simple, especially if we use a positive semi-definite kernel (we explore these in the next lecture)

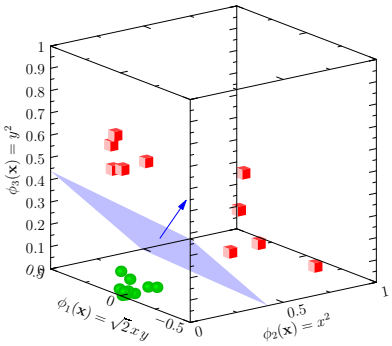
Advanced Machine Learning

Kernel Trick



The Kernel Trick, SVMs, Regression

Outline



SVM Kernels

- SVM Kernels are functions of two variables that can be factorised

$$K(\boldsymbol{x},\boldsymbol{y}) = \langle \boldsymbol{\phi}(\boldsymbol{x}),\boldsymbol{\phi}(\boldsymbol{y})\rangle = \sum_i \phi_i(\boldsymbol{x})\phi_i(\boldsymbol{y})$$

- where $\boldsymbol{\phi}(\boldsymbol{x}) = (\phi_1(\boldsymbol{x}),\phi_2(\boldsymbol{x}),\ldots)^{\mathsf{T}}$ and $\phi_i(\boldsymbol{x})$ are real valued functions of $\boldsymbol{x}$
- $K(\boldsymbol{x},\boldsymbol{y})$ will be positive semi-definite (because it is an inner-product)
- Furthermore, any positive semi-definite function will factorise
- This factorisation is not always obvious (we return to this later)

Dual Form

- Recall that the dual problem for an SVM is

$$\max_{\boldsymbol{\alpha}} \sum_{k=1}^m \alpha_k - \frac{1}{2}\sum_{k,l=1}^m \alpha_k\alpha_l y_k y_l \langle \boldsymbol{\phi}(\boldsymbol{x}_k),\boldsymbol{\phi}(\boldsymbol{x}_l)\rangle$$

- subject to $\sum_{k=1}^m y_k\alpha_k = 0$ and $0 \le \alpha_k (\le C)$
- But since $K(\boldsymbol{x}_k,\boldsymbol{x}_l) = \langle \boldsymbol{\phi}(\boldsymbol{x}_k),\boldsymbol{\phi}(\boldsymbol{x}_l)\rangle$ the dual problem becomes

$$\max_{\boldsymbol{\alpha}} \sum_{k=1}^m \alpha_k - \frac{1}{2}\sum_{k,l=1}^m \alpha_k\alpha_l y_k y_l K(\boldsymbol{x}_k,\boldsymbol{x}_l)$$

- This is the **kernel trick**—we never have to compute $\boldsymbol{\phi}(\boldsymbol{x})$!

Classifying New Data

- Having trained the SVM we now have to use it
- Given a new input $\boldsymbol{x}$ we decide on the class

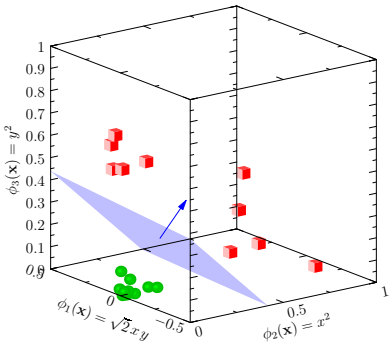
$$y = \mathrm{sgn}(\langle \boldsymbol{w},\boldsymbol{\phi}(\boldsymbol{x})\rangle - b) \qquad \text{but} \qquad \boldsymbol{w} = \sum_{k=1}^m \alpha_k y_k \boldsymbol{\phi}(\boldsymbol{x}_k)$$

- In the dual representation this becomes

$$\mathrm{sgn}\left(\sum_{k=1}^m \alpha_k y_k K(\boldsymbol{x}_k,\boldsymbol{x}) - b\right)$$

where we only need to sum over the non-zero α_k (i.e. the support vectors SVs)

Outline



Recap on Eigen Systems

- Recall for a symmetric $(n \times n)$ matrix $\mathbf{M}$ an eigenvector, $\boldsymbol{v}$

$$\mathbf{M}\boldsymbol{v} = \lambda\boldsymbol{v}$$

- There are n independent eigenvectors $\boldsymbol{v}^{(i)}$ with real eigenvalues $\lambda^{(i)}$
- The eigenvectors are orthogonal so that $\boldsymbol{v}^{(i)\mathsf{T}}\boldsymbol{v}^{(j)} = 0$ if $i \neq j$
- Forming a matrix of eigenvectors $\mathbf{V} = (\boldsymbol{v}^{(1)},\boldsymbol{v}^{(2)},\ldots\boldsymbol{v}^{(n)})$ the matrix satisfies

$$\mathbf{V}^{\mathsf{T}}\mathbf{V} = \mathbf{V}\mathbf{V}^{\mathsf{T}} = \mathbf{I}$$

- Such matrices are said to be orthogonal

Eigen Decomposition

- From the eigenvalue equation $\mathbf{M}\boldsymbol{v}^{(k)} = \lambda^{(k)}\boldsymbol{v}^{(k)}$

$$\mathbf{M}\mathbf{V} = \mathbf{V}\boldsymbol{\Lambda} \qquad \text{where} \quad \boldsymbol{\Lambda} = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

- Multiplying on the right by $\mathbf{V}^{\mathsf{T}}$ we get

$$\mathbf{M} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^{\mathsf{T}} = \sum_{k=1}^n \lambda^{(k)}\boldsymbol{v}^{(k)}\boldsymbol{v}^{(k)\mathsf{T}}$$

Or

$$M_{ij} = \sum_{k=1}^n \lambda^{(k)} v_i^{(k)} v_j^{(k)} = \sum_{k=1}^n u_i^{(k)} u_j^{(k)} = \langle \boldsymbol{u}_i,\boldsymbol{u}_j\rangle$$

$$u_i^{(k)} = \sqrt{\lambda^{(k)}}\, v_i^{(k)}$$

Eigenfunctions

- By analogy for a symmetric function of two variables we can define an *eigenfunction*

$$\int K(\boldsymbol{x},\boldsymbol{y})\psi(\boldsymbol{y})\mathrm{d}\boldsymbol{y} = \lambda\psi(\boldsymbol{x})$$

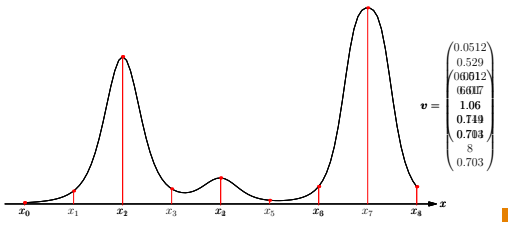
- In general there will be a denumerable set of eigenfunctions $\psi^{(k)}(\boldsymbol{x})$ where

$$K(\boldsymbol{x},\boldsymbol{y}) = \sum_k \lambda^{(k)}\psi^{(k)}(\boldsymbol{x})\psi^{(k)}(\boldsymbol{y})$$

- This is known as Mercer's theorem

Limit Process

- Consider sampling a function at a set of points

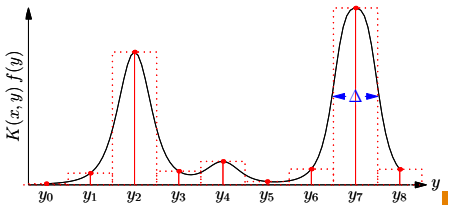


- In the limit where the number of sample points goes to infinity the vector more closely approximates a function
- Instead of the indices being numbers we use $k \leftarrow x_k$

Linear Operators

- Recall a linear function $\mathcal{T}[f(x)]$ can be represented by a kernel

$$\mathcal{T}[f(x)] = \int_{y\in\mathcal{I}} K(x,y)f(y)\mathrm{d}y \approx \Delta\sum_{j=1}^{n} K(x,y_j)f(y_j)$$



This is just a matrix equation with $M_{ij} = \Delta K(x_i,y_j)$

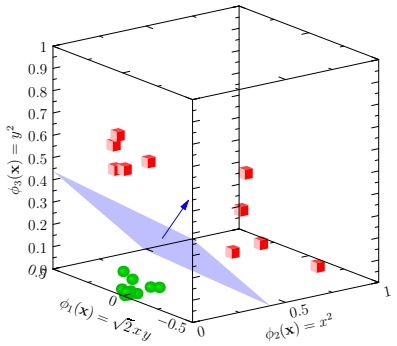
SVM Kernels

- If we define $\phi^{(k)}(\boldsymbol{x}) = \sqrt{\lambda^{(k)}}\psi^{(k)}(\boldsymbol{x})$ then

$$K(\boldsymbol{x},\boldsymbol{y}) = \sum_k \lambda^{(k)}\psi^{(k)}(\boldsymbol{x})\psi^{(k)}(\boldsymbol{y}) = \sum_k \phi^{(k)}(\boldsymbol{x})\phi^{(k)}(\boldsymbol{y})$$

- This is the definition of a SVM kernel we started with
- Note that for $\phi^{(k)}(\boldsymbol{x})$ to be real $\lambda^{(k)} \geq 0$ for all k
- If $\lambda^{(k)} < 0$ then $\langle\boldsymbol{\phi}(\boldsymbol{x}),\boldsymbol{\phi}(\boldsymbol{x})\rangle = \|\phi(\boldsymbol{x})\|^2$ might be negative and "distance" between points in the extended feature space can be negative!
- If we use a kernel that isn't positive semi-definite then the Hessian of the dual objective function will not be negative semi-definite and there will be a maximum where $\boldsymbol{\alpha}$ diverges

Outline



Positive Semi-Definite Kernels

- Kernels (or matrices) that have eigenvalues $\lambda^{(k)} \geq 0$ are called positive semi-definite
- (If the eigenvalues are strictly positive $\lambda^{(k)} > 0$ the kernels or matrices are called positive definite)
- Positive semi-definite kernels can always be decomposed into a sum of real functions

$$K(\boldsymbol{x},\boldsymbol{y}) = \langle\boldsymbol{\phi}(\boldsymbol{x}),\boldsymbol{\phi}(\boldsymbol{y})\rangle$$

Properties of Positive Semi-Definiteness

- Since

$$K(\boldsymbol{x},\boldsymbol{y}) = \big\langle\boldsymbol{\phi}(\boldsymbol{x}),\boldsymbol{\phi}(\boldsymbol{y})\big\rangle$$

- An immediate consequence is that for any function $f(\boldsymbol{x})$

$$\begin{aligned}\int f(\boldsymbol{x})K(\boldsymbol{x},\boldsymbol{y})f(\boldsymbol{y})\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} &= \int f(\boldsymbol{x})\big\langle\boldsymbol{\phi}(\boldsymbol{x}),\boldsymbol{\phi}(\boldsymbol{y})\big\rangle f(\boldsymbol{y})\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y}\\ &= \left\langle\int f(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})\mathrm{d}\boldsymbol{x}, \int f(\boldsymbol{y})\boldsymbol{\phi}(\boldsymbol{y})\mathrm{d}\boldsymbol{y}\right\rangle\\ &= \left\|\int f(\boldsymbol{x})\boldsymbol{\phi}(\boldsymbol{x})\mathrm{d}\boldsymbol{x}\right\|^2 \geq 0\end{aligned}$$

Positive Semi-Definiteness

- The following statements are equivalent
  - $K(\boldsymbol{x},\boldsymbol{y})$ is positive semi-definite (written $K(\boldsymbol{x},\boldsymbol{y}) \succeq 0$)
  - The eigenvalues of $K(\boldsymbol{x},\boldsymbol{y})$ are non-negative
  - The kernel can be written

$$K(\boldsymbol{x},\boldsymbol{y}) = \big\langle\boldsymbol{\phi}(\boldsymbol{x}),\boldsymbol{\phi}(\boldsymbol{y})\big\rangle$$

    where the $\phi^{(k)}(\boldsymbol{x})$'s are real functions
  - For any real function $f(x)$

$$\int f(\boldsymbol{x})K(\boldsymbol{x},\boldsymbol{y})f(\boldsymbol{y})\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} \geq 0$$

Adding Kernels

- We can construct SVM kernels from other kernels
- If $K_1(\boldsymbol{x},\boldsymbol{y})$ and $K_2(\boldsymbol{x},\boldsymbol{y})$ are valid kernels then so is $K_3(\boldsymbol{x},\boldsymbol{y}) = K_1(\boldsymbol{x},\boldsymbol{y}) + K_2(\boldsymbol{x},\boldsymbol{y})$

$$\begin{aligned} Q &= \int f(\boldsymbol{x}) K_3(\boldsymbol{x},\boldsymbol{y}) f(\boldsymbol{y}) \mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} \\ &= \int f(\boldsymbol{x}) \left(K_1(\boldsymbol{x},\boldsymbol{y}) + K_2(\boldsymbol{x},\boldsymbol{y})\right) f(\boldsymbol{y}) \mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} \\ &= \int f(\boldsymbol{x}) K_1(\boldsymbol{x},\boldsymbol{y}) f(\boldsymbol{y}) \mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} + \int f(\boldsymbol{x}) K_2(\boldsymbol{x},\boldsymbol{y}) f(\boldsymbol{y}) \mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} \geq 0 \end{aligned}$$

- If $K(\boldsymbol{x},\boldsymbol{y})$ is a valid kernel so is $cK(\boldsymbol{x},\boldsymbol{y})$ for $c > 0$

Product of Kernels

- If $K_1(\boldsymbol{x},\boldsymbol{y})$ and $K_2(\boldsymbol{x},\boldsymbol{y})$ are valid kernels then so is $K_3(\boldsymbol{x},\boldsymbol{y}) = K_1(\boldsymbol{x},\boldsymbol{y}) K_2(\boldsymbol{x},\boldsymbol{y})$
- Writing

$$K_1(\boldsymbol{x},\boldsymbol{y}) = \sum_i \phi_i^{(1)}(\boldsymbol{x})\phi_i^{(1)}(\boldsymbol{y}), \qquad K_2(\boldsymbol{x},\boldsymbol{y}) = \sum_j \phi_j^{(2)}(\boldsymbol{x})\phi_j^{(2)}(\boldsymbol{y})$$

then

$$\begin{aligned} K_3(\boldsymbol{x},\boldsymbol{y}) &= \sum_{i,j} \phi_i^{(1)}(\boldsymbol{x})\phi_i^{(1)}(\boldsymbol{y})\phi_j^{(2)}(\boldsymbol{x})\phi_j^{(2)}(\boldsymbol{y}) \\ &= \sum_{i,j} \phi_{ij}^{(3)}(\boldsymbol{x})\phi_{ij}^{(3)}(\boldsymbol{y}) = \left\langle \boldsymbol{\phi}^{(3)}(\boldsymbol{x}), \boldsymbol{\phi}^{(3)}(\boldsymbol{y}) \right\rangle \end{aligned}$$

where $\phi_{ij}^{(3)}(\boldsymbol{x}) = \phi_i^{(1)}(\boldsymbol{x})\phi_j^{(2)}(\boldsymbol{x})$

Exponentiating Kernels

- If $K(\boldsymbol{x},\boldsymbol{y})$ is a valid kernel so is $K^n(\boldsymbol{x},\boldsymbol{y})$ (by induction)
  - Assume $K(\boldsymbol{x},\boldsymbol{y}) \succeq 0$ this satisfies base case
  - If $K^{n-1}(\boldsymbol{x},\boldsymbol{y}) \succeq 0$ then

$$K^n(\boldsymbol{x},\boldsymbol{y}) = K^{n-1}(\boldsymbol{x},\boldsymbol{y}) K(\boldsymbol{x},\boldsymbol{y}) \succeq 0$$

- and $\exp(K(\boldsymbol{x},\boldsymbol{y}))$ is also a valid kernel since

$$\mathrm{e}^{K(\boldsymbol{x},\boldsymbol{y})} = \sum_{i=1} \frac{1}{i!} K^i(\boldsymbol{x},\boldsymbol{y}) = 1 + K(\boldsymbol{x},\boldsymbol{y}) + \frac{1}{2}K^2(\boldsymbol{x},\boldsymbol{y}) + \cdots$$

but each term in the sum is a kernel

RBF Kernel

- Now $\boldsymbol{x}^\mathsf{T}\boldsymbol{y} = \langle \boldsymbol{x},\boldsymbol{y} \rangle$ is a valid kernel because it is an inner product of functions $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{x}$
- For $\gamma > 0$ we have $2\gamma \boldsymbol{x}^\mathsf{T}\boldsymbol{y} \succeq 0$
- Thus $\exp(2\gamma \boldsymbol{x}^\mathsf{T}\boldsymbol{y}) \succeq 0$
- If $K(\boldsymbol{x},\boldsymbol{y}) \succeq 0$ then $g(\boldsymbol{x})K(\boldsymbol{x},\boldsymbol{y})g(\boldsymbol{y}) \succeq 0$

$$\int f(\boldsymbol{x})g(\boldsymbol{x})K(\boldsymbol{x},\boldsymbol{y})g(\boldsymbol{y})f(\boldsymbol{y})\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} = \int h(\boldsymbol{x})K(\boldsymbol{x},\boldsymbol{y})h(\boldsymbol{y})\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} \succeq 0$$

where $f(\boldsymbol{x})g(\boldsymbol{x}) = h(\boldsymbol{x})$

$$\mathrm{e}^{-\gamma \boldsymbol{x}^\mathsf{T}\boldsymbol{x}} \mathrm{e}^{2\gamma \boldsymbol{x}^\mathsf{T}\boldsymbol{y}} \mathrm{e}^{-\gamma \boldsymbol{y}^\mathsf{T}\boldsymbol{y}} = \mathrm{e}^{-\gamma \|\boldsymbol{x}-\boldsymbol{y}\|^2} \succeq 0$$

Other Kernels

- The success of SVMs has meant that researchers try to increase the area of application
- The condition that a SVM kernel must be positive semi-definite is quite restrictive
- There has been a cottage industry of researchers finding smart kernels for solving complicated problems
- The key to finding new kernels is to use the properties of kernels to build more complicated kernels from simpler ones

String Kernels

- One area where SVMs were very important is in document classification
- This requires comparing strings
- There are a large number of kernels developed to do this

Spectrum Kernel

- A simple way to compare documents is to collect a histogram of all occurrences of substrings of length p
- This is known as a p-spectrum
- A p-spectrum kernel counts the number of common substrings

$s = \texttt{statistics}$ $\quad \mathcal{S}_3(s) = \{\texttt{sta},\texttt{tat},\texttt{ati},\texttt{tis},\texttt{ist},\texttt{sti},\texttt{tic},\texttt{ics}\}$

$t = \texttt{computation}$ $\quad \mathcal{S}_3(t) = \{\texttt{com},\texttt{omp},\texttt{mpu},\texttt{put},\texttt{uta},\texttt{tat},\texttt{ati},\texttt{tio},\texttt{ion}\}$

- $K(s,t) = 2$ ("tat" and "ati")

All Subsequences Kernel

- A more sophisticated kernel is to count all of the common subsequences that occur in two documents
- Naively this would take an exponential amount of time to compute
- Using clever dynamic-programming techniques this can be done relatively efficiently
- This can even be extended to include sub-sequence matches with possible gaps between words

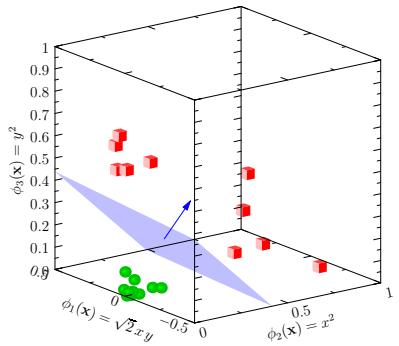
Other Kernel Applications

- String kernels for comparing subsequences are used in bioinformatics
- Kernels have been developed for comparing trees (e.g. for computer program evaluation, XML, etc.)
- Kernels have also been developed for comparing graphs (e.g. for comparing chemicals based on their molecular graph)

Fisher Kernels

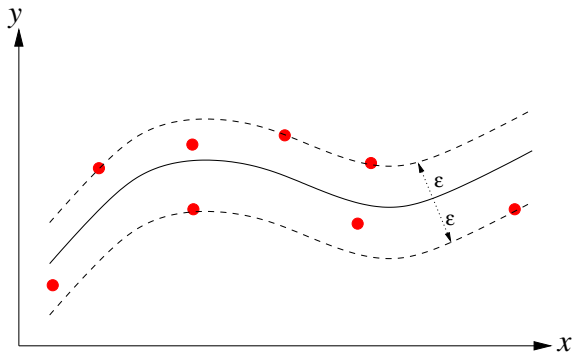
- In an attempt to build kernels that capture more domain knowledge, kernels are constructed from other learning machines
- An example of this are "Fisher kernels" whose features come from an Hidden Markov Model (HMM) trained on the data
- These tend to have better discriminative power than the underlying model (HMM), and has a better feature set than a SVM using a generic kernel

Outline



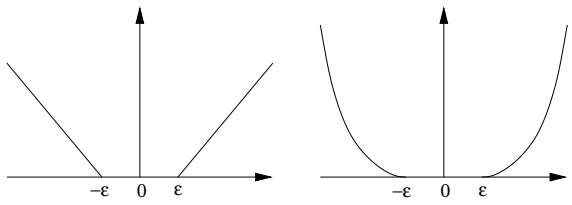
Regression with Margins

- SVMs can be modified to perform regression



Error Functions

- Can introduce slack variables with different errors



- This can be transformed to a quadratic programming problem

Ridge Regression Using Kernels

- We can also solve regression problems without using margins
- To solve a regression problem once again the problem is set up as a quadratic programming problem

$$\min_{\boldsymbol{w}} \lambda \|\boldsymbol{w}\|^2 + \sum_{i=1}^{m} \left(y_i - \boldsymbol{w}^{\mathsf{T}} \boldsymbol{\phi}(\boldsymbol{x}_i)\right)^2$$

- the $\|\boldsymbol{w}\|^2$ is a regularisation term
- By assuming $\boldsymbol{w} = \sum_i \alpha_i \boldsymbol{\phi}(\boldsymbol{x}_i)$ we obtain a quadratic equation for the α_i's which we can solve

Kernel Methods

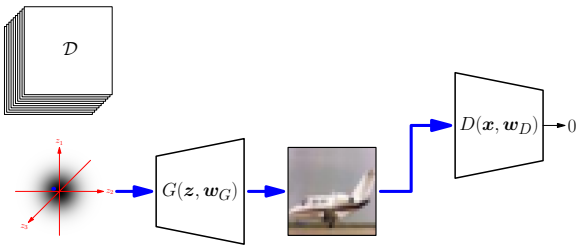
- Kernel methods where we project into an extended feature space are used with other linear algorithms
  - ⋆ Kernel Fisher discriminant analysis (KFDA)
  - ⋆ Kernel principle component analysis (KPCA)
  - ⋆ Kernel canonical correlation analysis (KCCA)
  - ⋆ Gaussian Processes
- These are also extremely powerful machine learning algorithms

Summary

- SVMs require a positive definite kernel function
- These can be built from simpler function
- There was a cottage industry of people creating new kernels for different application
- SVMs are just one example of a host of machine that
  - ⋆ use the kernel trick
  - ⋆ often use linear constraints
  - ⋆ tend to be convex optimisation problems

Advanced Machine Learning

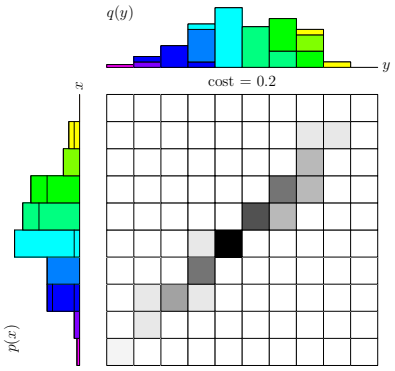
Wasserstein GANs



GANs, Wasserstein distance, Duality, WGANs

Outline

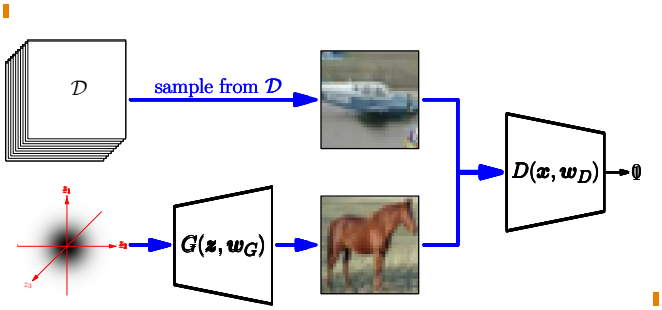
1. **GANs**
2. Wasserstein Distance
3. Wasserstein GANs



Generative Adversarial Networks

- One of the applications of Deep Learning that has most excited the public are **Generative Adversarial Networks** or GANs
- Their aim is to generate new random samples from the same distribution as some training set, $\mathcal{D}$
- Their number of real world applications are questionable
- But nobody cares because they are cool!
- *Out of date warning:* someone invented diffusion models

How GANs Work



Training GANs

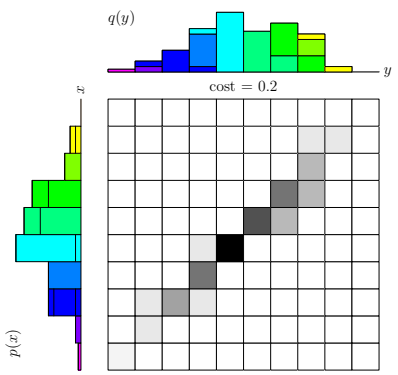
- The loss of the generator depends on its ability to trick the discriminator
- The loss of the discriminator depends on its ability not to be tricked
- We try to train the two networks simultaneously
- We hope that over time the generator produces better and better fakes

Problems of GANs

- GANs are notoriously difficult to train
- The generator and discriminator training can decouple
- Often the discriminator becomes too good at correctly identifying the generated images
- Then there can be little gradient information to help the generator as every small change in parameters doesn't significantly change the discriminator decision
- To try to solve this problem we first make a seemly unconnected diversion

Outline

1. GANs
2. **Wasserstein Distance**
3. Wasserstein GANs



Measuring Distances Between Distributions

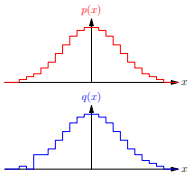
- In many machine learning tasks we want to minimise the distance between two probability distributions
- This requires that we can measure distances between probability distributions
- One prominent measure is the Kullback-Leibler or KL divergence

$$\mathrm{KL}(p\|q) = \int p(\boldsymbol{x}) \log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{y})}\right) \mathrm{d}\boldsymbol{x}$$

- This is very commonly used in ML (e.g. VAEs, Variational Approximation)

Trouble with KL

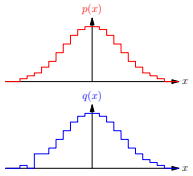
- KL-divergences are non-negative quantities that are minimised when the two probability distributions are the same
- They are not distances (they aren't symmetric and they don't satisfy the triangular inequality)
- We don't really care about this, but what we do care about is that if $q(\boldsymbol{x}) = 0$ when $p(\boldsymbol{x}) \neq 0$ then $\log\left(\frac{p(\boldsymbol{x})}{q(\boldsymbol{y})}\right)$ diverges



- We can therefore have distributes that seem very similar but their KL-divergence is huge (or infinite)

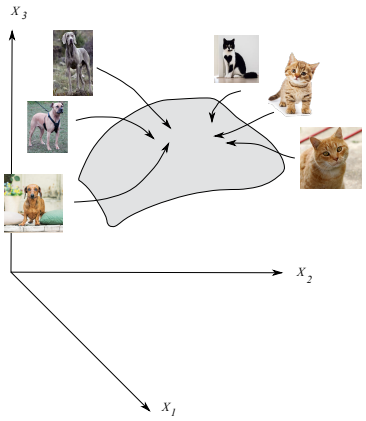
Wasserstein Distance

- A more benign measure of the differences between two probability functions is the **Wasserstein** or **Earth Moving** distance
- This is a true distance, but more importantly for us it measure distance in a very natural way so that distributions that are close has a small Wasserstein distance



- Although this seems contrived if our probability distribution represents the probability of a 128×128 matrix of real valued triples represents an image of dog, then it is easy to imagine that the Wasserstein distance may be more benign than the KL-divergence

High Probability Manifold



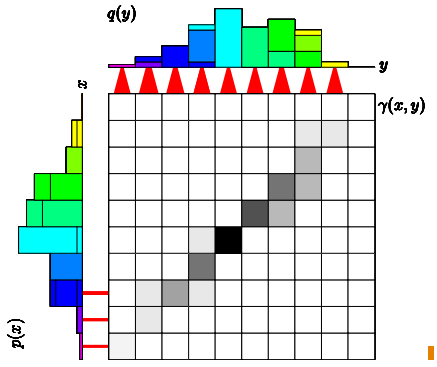
Transportion Policy

- But how do we formalise the Wasserstein distance?
- A good place to start is to define a transportation policy $\gamma(\boldsymbol{x},\boldsymbol{y})$ with

$$\int \gamma(\boldsymbol{x},\boldsymbol{y})\mathrm{d}\boldsymbol{y} = p(\boldsymbol{x}) \qquad \int \gamma(\boldsymbol{x},\boldsymbol{y})\mathrm{d}\boldsymbol{x} = q(\boldsymbol{y})$$

- This looks like a joint probability distribution, but we interpret $\gamma(\boldsymbol{x},\boldsymbol{y})$ as the amount of probability mass/density that we transfer from $p(\boldsymbol{x})$ to $q(\boldsymbol{y})$

Transportation Policy



The Cost of Transport

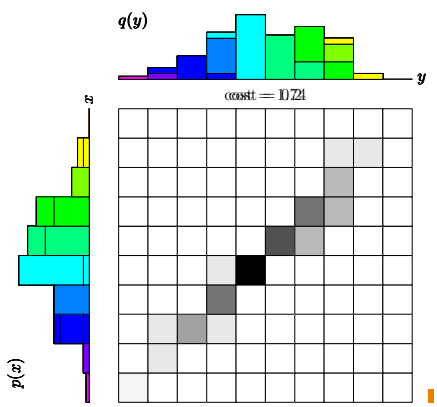
- We want to choose the transportation policy that minimises the amount of probability mass we need to move
- Let $d(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|$ be a distance measure then the cost of a transportation policy is

$$C(\gamma) = \int\int d(\boldsymbol{x},\boldsymbol{y})\gamma(\boldsymbol{x},\boldsymbol{y})\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y} = \mathbb{E}_\gamma[d(\boldsymbol{x},\boldsymbol{y})]$$

where we interpret $\gamma(\boldsymbol{x},\boldsymbol{y})$ as a probability distribution

- Usually we take $d(\boldsymbol{x},\boldsymbol{y})$ to be the Euclidean distance, but we can choose any distance

Transportation Cost



The Wasserstein Distance

- The Wasserstein distance $W(p,q)$ between two probability distributions is defined as

$$W(p,q) = \min_{\gamma\in\Lambda(p,q)} \mathbb{E}_\gamma[d(\boldsymbol{x},\boldsymbol{y})]$$

- Where $\Lambda(p,q)$ is the set of joint distributions $\gamma(\boldsymbol{x},\boldsymbol{y})$ such that

$$\int \gamma(\boldsymbol{x},\boldsymbol{y})\mathrm{d}\boldsymbol{y} = p(\boldsymbol{x}) \qquad \int \gamma(\boldsymbol{x},\boldsymbol{y})\mathrm{d}\boldsymbol{x} = q(\boldsymbol{y})$$

Computing the Wasserstein Distance

- To compute the Wasserstein distance we have to solve a minimisation task!
- This looks nasty, but it is a (continuous) linear programmming problem
- Suppose p and q were discrete distribution (i.e. $\boldsymbol{x}$ and $\boldsymbol{y}$ only take discrete points)
- Then we could treat each value of $\gamma(\boldsymbol{x},\boldsymbol{y})$ as an element of a vector $\boldsymbol{\gamma}$ and each value of $d(\boldsymbol{x},\boldsymbol{y})$ as an element of a vector $\boldsymbol{D}$
- Our objective is to choose $\boldsymbol{\gamma}$ to minimise $\boldsymbol{D}^\mathsf{T}\boldsymbol{\gamma}$

Constraints

$$\sum_j \gamma(\boldsymbol{x}_i,\boldsymbol{y}_j) = p(\boldsymbol{x}_i) \qquad \sum_i \gamma(\boldsymbol{x}_i,\boldsymbol{y}_j) = q(\boldsymbol{y}_j)$$

$$\mathbf{A}\boldsymbol{\gamma} = \boldsymbol{P}$$

$$\begin{pmatrix} 1 & 1 & \cdots & 1 & 0 & 0 & \cdots & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 1 & 1 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & 0 & \cdots & 1 & 1 & \cdots & 1 \\ 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & \cdots & 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 & 0 & 1 & \cdots & 0 & \cdots & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \cdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1 & \cdots & 0 & 0 & \cdots & 1 \end{pmatrix} \begin{pmatrix} \gamma(x_1,\,y_1) \\ \gamma(x_2,\,y_1) \\ \vdots \\ \gamma(x_n,\,y_1) \\ \gamma(x_1,\,y_2) \\ \gamma(x_2,\,y_2) \\ \vdots \\ \gamma(x_n,\,y_2) \\ \vdots \\ \gamma(x_1,\,y_n) \\ \gamma(x_2,\,y_n) \\ \vdots \\ \gamma(x_n,\,y_n) \end{pmatrix} = \begin{pmatrix} q(y_1) \\ q(y_2) \\ \vdots \\ q(y_n) \\ p(x_1) \\ p(x_2) \\ \vdots \\ p(x_n) \end{pmatrix}$$

Lagrange Formulation

- For discrete distributions

$$\min_{\boldsymbol{\gamma}} \boldsymbol{D}^\mathsf{T}\boldsymbol{\gamma}$$
$$\text{subject to} \quad \mathbf{A}\boldsymbol{\gamma} = \boldsymbol{P}, \quad \boldsymbol{\gamma} \geq 0$$

- Writing the Lagrangian

$$\mathcal{L}(\boldsymbol{\gamma},\boldsymbol{\alpha}) = \boldsymbol{D}^\mathsf{T}\boldsymbol{\gamma} - \boldsymbol{\alpha}^\mathsf{T}(\mathbf{A}^\mathsf{T}\boldsymbol{\gamma} - \boldsymbol{P})$$

where $\boldsymbol{\alpha}$ is a vector of Lagrange multipliers

- The solution to the discrete optimisation problem is given by

$$\min_{\boldsymbol{\gamma}} \max_{\boldsymbol{\alpha}} \mathcal{L}(\boldsymbol{\gamma},\boldsymbol{\alpha})$$

Dual Form

- We can rearrange

$$\begin{aligned}\mathcal{L}(\boldsymbol{\gamma},\boldsymbol{\alpha}) &= \boldsymbol{D}^\mathsf{T}\boldsymbol{\gamma} - \boldsymbol{\alpha}^\mathsf{T}(\mathbf{A}\boldsymbol{\gamma} - \boldsymbol{P}) \\ &= \boldsymbol{P}^\mathsf{T}\boldsymbol{\alpha} - \boldsymbol{\gamma}^\mathsf{T}(\mathbf{A}^\mathsf{T}\boldsymbol{\alpha} - \boldsymbol{D})\end{aligned}$$

- We note that $\boldsymbol{\gamma} \geq 0$ so the dual problem is to find a vector $\boldsymbol{\alpha}$ that maximises $\boldsymbol{P}^\mathsf{T}\boldsymbol{\alpha}$ subject to the constraints $\mathbf{A}^\mathsf{T}\boldsymbol{\alpha} \leq \boldsymbol{D}$
- Although the vector form allows us to make connections with our earlier discussion of linear programming, it is a little difficult to interpret

Explicit Form

- We can write a Lagrangian for the original problem

$$\begin{aligned}\mathcal{L} = \sum_{i,j} d(\boldsymbol{x}_i,\boldsymbol{y}_i)\gamma(\boldsymbol{x}_i,\boldsymbol{y}_j) - \sum_i \alpha(\boldsymbol{x}_i)\left(\sum_j \gamma(\boldsymbol{x}_i,\boldsymbol{y}_j) - p(\boldsymbol{x}_i)\right) \\ - \sum_j \beta(\boldsymbol{y}_j)\left(\sum_i \gamma(\boldsymbol{x}_i,\boldsymbol{y}_j) - q(\boldsymbol{y}_j)\right)\end{aligned}$$

subject to $\gamma(\boldsymbol{x}_i,\boldsymbol{y}_j) \geq 0$ where $\alpha(\boldsymbol{x}_i)$ and $\beta(\boldsymbol{y}_j)$ are Lagrange multipliers (they are components of $\boldsymbol{\alpha}$)

- Rearranging

$$\mathcal{L} = \sum_i \alpha(\boldsymbol{x}_i)p(\boldsymbol{x}_i) + \sum_j \beta(\boldsymbol{y}_j)q(\boldsymbol{y}_j) - \sum_{i,j}\gamma(\boldsymbol{x}_i,\boldsymbol{y}_j)(\alpha(\boldsymbol{x}_i) + \beta(\boldsymbol{y}_j) - d(\boldsymbol{x}_i,\boldsymbol{y}_i))$$

- This is eqivalent to maximising $\sum_i \alpha(\boldsymbol{x}_i)p(\boldsymbol{x}_i) + \sum_j \beta(\boldsymbol{y}_j)q(\boldsymbol{y}_j)$, subject to

$$\forall i,j \quad \alpha(\boldsymbol{x}_i) + \beta(\boldsymbol{y}_j) \leq d(\boldsymbol{x}_i,\boldsymbol{y}_j)$$

Continuous Form

- We can write a Lagrangian for the continuous problem

$$\begin{aligned}\mathcal{L} = \iint d(\boldsymbol{x},\boldsymbol{y})\gamma(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y} - \int \alpha(\boldsymbol{x})\left(\int \gamma(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} - p(\boldsymbol{x})\right)\mathrm{d}\boldsymbol{x} \\ - \int \beta(\boldsymbol{y})\left(\int \gamma(\boldsymbol{x},\boldsymbol{y})\,\mathrm{d}\boldsymbol{x} - q(\boldsymbol{y})\right)\mathrm{d}\boldsymbol{y}\end{aligned}$$

subject to $\gamma(\boldsymbol{x},\boldsymbol{y}) \geq 0$ where $\alpha(\boldsymbol{x})$ and $\beta(\boldsymbol{y})$ are Lagrange multiplier functions

- Rearranging

$$\mathcal{L} = \int \alpha(\boldsymbol{x})p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int \beta(\boldsymbol{y})q(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y} - \iint \gamma(\boldsymbol{x},\boldsymbol{y})(\alpha(\boldsymbol{x}) + \beta(\boldsymbol{y}) - d(\boldsymbol{x},\boldsymbol{y}))\,\mathrm{d}\boldsymbol{x}\,\mathrm{d}\boldsymbol{y}$$

- This is eqivalent to maximising $\int \alpha(\boldsymbol{x})p(\boldsymbol{x})\,\mathrm{d}\boldsymbol{x} + \int \beta(\boldsymbol{y})q(\boldsymbol{y})\,\mathrm{d}\boldsymbol{y}$, subject to

$$\alpha(\boldsymbol{x}) + \beta(\boldsymbol{y}) \leq d(\boldsymbol{x},\boldsymbol{y})$$

Dual Form Constraint

- We note that $\alpha(\boldsymbol{x}) + \beta(\boldsymbol{y}) \leq d(\boldsymbol{x},\boldsymbol{y})$ for all $\boldsymbol{x}$ and $\boldsymbol{y}$
- This has to be true when $\boldsymbol{x} = \boldsymbol{y}$ so that

$$\alpha(\boldsymbol{x}) + \beta(\boldsymbol{x}) \leq d(\boldsymbol{x},\boldsymbol{x}) = 0$$

- So $\beta(\boldsymbol{x}) = -\alpha(\boldsymbol{x}) - \epsilon(\boldsymbol{x})$ where $\epsilon(\boldsymbol{x}) \geq 0$
- But want to maximise

$$\int \alpha(\boldsymbol{x})p(\boldsymbol{x})\mathrm{d}\boldsymbol{x} + \int \beta(\boldsymbol{y})q(\boldsymbol{y})\mathrm{d}\boldsymbol{y} = \int \alpha(\boldsymbol{x})\,(p(\boldsymbol{x}) - q(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x} - \int q(\boldsymbol{x})\epsilon(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$$

- This is maximised when $\epsilon(\boldsymbol{x}) = 0$ i.e. $\beta(\boldsymbol{x}) = -\alpha(\boldsymbol{x})$

Dual Form

- Thus the dual problem is to find a function $\alpha(\boldsymbol{x})$—or a vector of functions $(\alpha(\boldsymbol{x}_i)|i)$—that maximises

$$\int \alpha(\boldsymbol{x})\,(p(\boldsymbol{x}) - q(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x}$$

- Subject to the constraint

$$\alpha(\boldsymbol{x}) - \alpha(\boldsymbol{y}) \leq d(\boldsymbol{x},\boldsymbol{y}) = \|\boldsymbol{x} - \boldsymbol{y}\|$$

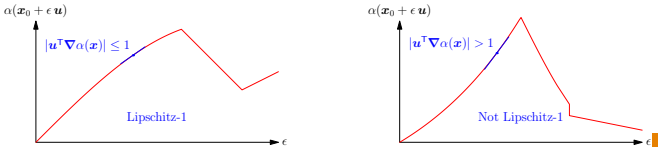
- This is a continuity constraint on the Lagrange multiplier function $\alpha(\boldsymbol{x})$ known as Lipschitz-1

Lipschitz-1 Functions

- We note for a Lipschitz-1 function and any unit vector $\boldsymbol{u}$

$$\boldsymbol{u}^\top \nabla \alpha(\boldsymbol{x}) = \lim_{\epsilon \to 0} \frac{\alpha(\boldsymbol{x}) - \alpha(\boldsymbol{x} + \epsilon \boldsymbol{u})}{\epsilon} \leq \frac{\epsilon}{\epsilon} = 1$$

- That is, at every point the gradient in all directions must be less than 1 (since the gradient defines the direction of greatest increase it is both necessary and sufficient for $\|\nabla \alpha(\boldsymbol{x})\| \leq 1$ everywhere)



Calculating the Wasserstein Distance

- To recall the big picture we want to compute the Wasserstein distance

$$W(p,q) = \min_{\gamma \in \Lambda(p,q)} \mathbb{E}_\gamma[d(\boldsymbol{x},\boldsymbol{y})]$$

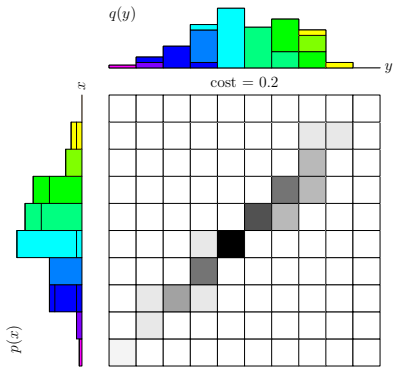
- For high dimensional objects $\gamma(\boldsymbol{x},\boldsymbol{y})$ would be a huge object to approximate
- Instead we can compute the Wasserstein distance in the dual formulation

$$W(p,q) = \max_{\alpha(\boldsymbol{x})} \int \alpha(\boldsymbol{x})\,(p(\boldsymbol{x}) - q(\boldsymbol{x}))\,\mathrm{d}\boldsymbol{x} = \max_{\alpha} \mathbb{E}_p[\alpha(\boldsymbol{X})] - \mathbb{E}_q[\alpha(\boldsymbol{X})]$$

subject to the constraint that $\alpha(\boldsymbol{x})$ is a Lipschitz-1 function

Outline

1. GANs
2. Wasserstein Distance
3. **Wasserstein GANs**



Back to GANs

- What has this got do with GANs?
- Suppose we want to minimise the distance between the distribution $p(\boldsymbol{x})$ of real images (of which $\mathcal{D}$ are samples) and the distribution $q(\boldsymbol{x})$ of images drawn from a generator
- We can use a normal GAN generator, $G(\boldsymbol{z},\boldsymbol{w}_G)$, that generates an image when given a random variable $\boldsymbol{z} \sim \mathcal{N}(\boldsymbol{0},\mathbf{I})$
- To do this we choose the weights, $\boldsymbol{w}_G$ of the generator to minimise

$$W(p,q) = \max_{\alpha(\boldsymbol{x})} (\mathbb{E}_{\boldsymbol{x}\sim p}[\alpha(\boldsymbol{x})] - \mathbb{E}_{\boldsymbol{x}\sim q}[\alpha(\boldsymbol{x})])$$

Estimating Expectations

- Although we can't compute $\mathbb{E}_p[\alpha(\boldsymbol{x})]$ and $\mathbb{E}_q[\alpha(\boldsymbol{x})]$ exactly, we can estimate them from samples

$$\mathbb{E}_p[\alpha(\boldsymbol{x})] \approx \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x}\in\mathcal{B}} \alpha(\boldsymbol{x}), \qquad \mathbb{E}_q[\alpha(\boldsymbol{x})] \approx \frac{1}{n} \sum_{i=1}^{n} \alpha(G(\boldsymbol{z}_i,\boldsymbol{w}_G))$$

- where $\mathcal{B} \subset \mathcal{D}$ is a minibatch of true images and $\boldsymbol{z}_i \sim \mathcal{N}(\boldsymbol{0},\mathbf{I})$
- From this we can choose $\boldsymbol{w}_G$ to minimise

$$C = \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x}\in\mathcal{B}} \alpha(\boldsymbol{x}) - \frac{1}{n} \sum_{i=1}^{n} \alpha(G(\boldsymbol{z}_i,\boldsymbol{w}_G))$$

The Critic

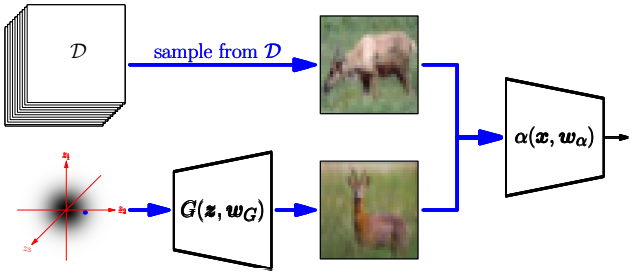
- For this quantity to approximate the Wasserstein distance we need to find a function $\alpha(\boldsymbol{x},\boldsymbol{w}_\alpha)$ that maximises C
- To do this we learn a second network, the critic or discriminator whose job it is to maximise

$$C = \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x}\in\mathcal{B}} \alpha(\boldsymbol{x},\boldsymbol{w}_\alpha) - \frac{1}{n} \sum_{i=1}^{n} \alpha(G(\boldsymbol{z}_i,\boldsymbol{w}_G),\boldsymbol{w}_\alpha)$$

- The network $\alpha(\boldsymbol{x},\boldsymbol{w}_\alpha)$ should be Lipschitz-1 (which we usually botched by, for example, by setting the spectral norm of the convolutional weight matrix to 1)

Wasserstein GANs

$$\max_{\boldsymbol{w}_\alpha} \min_{\boldsymbol{w}_G} \frac{1}{|\mathcal{B}|} \sum_{\boldsymbol{x}\in\mathcal{B}} \alpha(\boldsymbol{x},\boldsymbol{w}_\alpha) - \frac{1}{n} \sum_{i=1}^{n} \alpha(G(\boldsymbol{z}_i,\boldsymbol{w}_G),\boldsymbol{w}_\alpha)$$



Lesson

- Wasserstein GANs are, at least for me, one of the most elegant pieces of theory in recent years
- By trying to minimise the Wasserstein distance between the distribution of a generator and a true distribution we arrive at optimising two adversarial networks just like a GAN
- This uses a rather beautiful dual formulation
- It is claimed that W-GANs solve many of the problems of traditional GANs

Advanced Machine Learning

Probability

$Y = g(X)$				Ω
$y_{13} = g(x_{13})$	$y_{14} = g(x_{14})$	$y_{15} = g(x_{15})$	$y_{16} = g(x_{16})$	
$y_9 = g(x_9)$	$y_{10} = g(x_{10})$	$y_{11} = g(x_{11})$	$y_{12} = g(x_{12})$	
$y_5 = g(x_5)$	$y_6 = g(x_6)$	$y_7 = g(x_7)$	$y_8 = g(x_8)$	
$y_1 = g(x_1)$	$y_2 = g(x_2)$	$y_3 = g(x_3)$	$y_4 = g(x_4)$	

Probability, Random Variables, Expectations

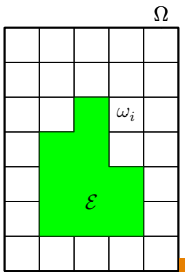
Outline

1. **Random Variables**
2. Expectations
3. Calculus of Probabilities

Ω				
x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{26}	x_{27}	x_{28}	x_{29}	x_{30}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_6	x_7	x_8	x_9	x_{10}
x_1	x_2	x_3	x_4	x_5

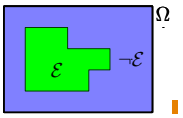
Modelling Uncertainty

- To model a world with uncertainty we consider some set of **elementary events** or **outcomes** Ω
- For the outcome of rolling a dice $\Omega = \{1,2,3,4,5,6\}$
- The elementary events ω_i are **mutually exclusive** $\omega_i \cap \omega_j = \emptyset$ and **exhaustive** $\bigcup_i \omega_i = \Omega$
- We consider **events** $\mathcal{E} = \bigcup_{i\in\mathcal{I}} \omega_i$
- E.g. For a dice throw $\mathcal{E} = \{2,4,6\}$



Probabilities

- We attribute a **probability**, $\mathbb{P}(\mathcal{E})$, to an event, $\mathcal{E}$, with the requirements
  - $0 \le \mathbb{P}(\mathcal{E}) \le 1$
  - $\mathbb{P}(\mathcal{E}) + \mathbb{P}(\neg\mathcal{E}) = 1$ where $\neg\mathcal{E} = \Omega \setminus \mathcal{E}$



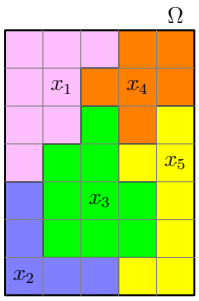
- In some cases we can interpret $\mathbb{P}(\mathcal{E})$ as the expected frequency of occurrence of a repetitive trial
- But $\mathbb{P}$(Pass COMP6208 exam) is something you do once
- Can think of probability as an informed belief that something might happen
- When our knowledge changes the probability changes

Random Variables

- We can define a **random variable**, X, by partition the set of outcomes Ω and assign a numbers to each partition
- E.g. for a dice

$$X = \begin{cases} 0 & \text{if } \omega \in \{1,3,5\} \\ 1 & \text{if } \omega \in \{2,4,6\} \end{cases}$$

- $\mathbb{P}(X = x_i) = \mathbb{P}(\mathcal{E}_i)$ where $\mathcal{E}_i$ is the event that corresponding to the partition with value x_i



What's In A Name

- We denote random variables with capital letters, X, Y, Z, etc.
- The symbol denote an object that can take one of a number of different values, but which one is still to be decided by chance
- When we write $\mathbb{P}(X)$ we can view this as short-hand for

$$(\mathbb{P}(X = x) \mid x \in \mathcal{X}) = (\mathbb{P}(X = x_1), \mathbb{P}(X = x_2), \ldots \mathbb{P}(X = x_n))$$

  where $\mathcal{X}$ is the set of possible values that X can take
- We treat random variables very differently to normal numbers (scalars) when we consider taking expectations

Function of Random Variables

- Any function, $Y = g(X)$, of a random variable, X, is a random variable

$Y = g(X)$				Ω
$y_{13} = g(x_{13})$	$y_{14} = g(x_{14})$	$y_{15} = g(x_{15})$	$y_{16} = g(x_{16})$	
$y_9 = g(x_9)$	$y_{10} = g(x_{10})$	$y_{11} = g(x_{11})$	$y_{12} = g(x_{12})$	
$y_5 = g(x_5)$	$y_6 = g(x_6)$	$y_7 = g(x_7)$	$y_8 = g(x_8)$	
$y_1 = g(x_1)$	$y_2 = g(x_2)$	$y_3 = g(x_3)$	$y_4 = g(x_4)$	

Continuous Spaces

- If the space of elementary events is continuous (e.g. for darts $\boldsymbol{x} = (x,y)$) then $\mathbb{P}(\boldsymbol{X} = \boldsymbol{x}) = 0$
- But if we consider a region, $\mathcal{R}$, then we can assign a probability to landing in the region $\mathbb{P}(\boldsymbol{X} \in \mathcal{R})$
- It is useful to work with **probability densities function** (PDF)

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = \lim_{\epsilon\to 0} \frac{\mathbb{P}(\boldsymbol{X} \in \mathcal{B}(\boldsymbol{x},\epsilon))}{|\mathcal{B}(\boldsymbol{x},\epsilon)|}$$

  where $\mathcal{B}(\boldsymbol{x},\epsilon)$ is a ball of radius ϵ around the point $\boldsymbol{x}$ and $|\mathcal{B}(\boldsymbol{x},\epsilon)|$ is the volume of the ball
- If we make a change of variable the volume $|\mathcal{B}(\boldsymbol{x},\epsilon)|$ might change so $f_{\boldsymbol{X}}(\boldsymbol{x})$ will change

Change of Variables

- Consider a region $\mathcal{R}$—we can describe this using different coordinate systems x or $y = g(x)$
- But

$$\mathbb{P}(X \in \mathcal{R}) = \int_{\mathcal{R}} f_X(x)\mathrm{d}x = \mathbb{P}(Y \in \mathcal{R}) = \int_{\mathcal{R}} f_Y(y)\mathrm{d}y$$

- As this is true for any region $\mathcal{R}$: $f_X(x)|\mathrm{d}x| = f_Y(y)|\mathrm{d}y|$
- Or

$$f_X(x) = f_Y(y)\left|\frac{\mathrm{d}y}{\mathrm{d}x}\right| = f_Y(g(x))|g'(x)|$$

- The probability density measured in units of probability per cm is different to that measured in units of probability per inch

Jacobian

- In high dimension if we make a change of variables $\boldsymbol{x} \to \boldsymbol{y}(\boldsymbol{x})$ (which can be seen as a change of random variables $\boldsymbol{X} \to \boldsymbol{Y}(\boldsymbol{X})$)
- Then

$$f_{\boldsymbol{X}}(\boldsymbol{x}) = f_{\boldsymbol{Y}}(\boldsymbol{y})|\det(\mathbf{J})|$$

where $\mathbf{J}$ is the Jacobian matrix

$$\mathbf{J} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \dots & \frac{\partial y_n}{\partial x_n} \end{pmatrix}$$

- Ensures integrals over volumes are the same

Meaning of Probability Densities

- Probability densities are not probabilities
- They are positive, but don't need to be less than 1
- Note that

$$f_X(x) = \lim_{\delta x \to 0} \frac{\mathbb{P}(x \le X < x + \delta x)}{\delta x}$$

- We can think of $f_X(x)\delta x$ as $\mathbb{P}(x \le X < x + \delta x)$
- Note that $f_X(x)\delta x \le 1$

Cumulative Distribution Functions

- We can define the **cumulative distribution function** (CDF)

$$F_X(x) = \mathbb{P}(X \le x) = \begin{cases} \sum_{i:x_i \le x} \mathbb{P}(X = x_i) \\ \int_{-\infty}^{x} f_X(y)\mathrm{d}y \end{cases}$$

- This is a function that goes from 0 to 1 as x goes from $-\infty$ to ∞
- We note that for continuous random variables

$$f_X(x) = \frac{\mathrm{d}F_X(x)}{\mathrm{d}x}$$

Outline

1. Random Variables
2. **Expectations**
3. Calculus of Probabilities

Ω

x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{26}	x_{27}	x_{28}	x_{29}	x_{30}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_6	x_7	x_8	x_9	x_{10}
x_1	x_2	x_3	x_4	x_5

Expectation

- We can define the expectation of $\boldsymbol{Y} = g(\boldsymbol{X})$ as

$$\mathbb{E}_{\boldsymbol{X}}[g(\boldsymbol{X})] = \begin{cases} \sum_{\boldsymbol{x} \in \mathcal{X}} g(\boldsymbol{x})\mathbb{P}(\boldsymbol{X} = \boldsymbol{x}) \\ \int g(\boldsymbol{x}) f_{\boldsymbol{X}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} \end{cases}$$

- The expectation of a constant c is

$$\mathbb{E}_{\boldsymbol{X}}[c] = \begin{cases} \sum_{\boldsymbol{x} \in \mathcal{X}} c\mathbb{P}(\boldsymbol{X} = \boldsymbol{x}) = c\sum_{\boldsymbol{x} \in \mathcal{X}} \mathbb{P}(\boldsymbol{X} = \boldsymbol{x}) = c \\ \int c f_{\boldsymbol{X}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = c\int f_{\boldsymbol{X}}(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = c \end{cases}$$

- Note $\mathbb{E}_X[\mathbb{E}_X[g(X)]] = \mathbb{E}_X[g(X)]$

Linearity of Expectation

- Because sums and integrals are linear operators

$$\sum_i (ax_i + by_i) = a\left(\sum_i x_i\right) + b\left(\sum_i y_i\right)$$

$$\int (af(\boldsymbol{x}) + bg(\boldsymbol{x}))\mathrm{d}\boldsymbol{x} = a\left(\int f(\boldsymbol{x})\mathrm{d}\boldsymbol{x}\right) + b\left(\int g(\boldsymbol{x})\mathrm{d}\boldsymbol{x}\right)$$

then expectations are linear

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$$

- Beware usually $\mathbb{E}[XY] \ne \mathbb{E}[X]\mathbb{E}[Y]$ (unless X and Y are independent)

Indicator Functions

- An indicator function has the property

$$[\![predicate]\!] = \begin{cases} 1 & \text{if } predicate \text{ is True} \\ 0 & \text{if } predicate \text{ is False} \end{cases}$$

(sometimes written $\boldsymbol{I}_A(x)$ where $A(x)$ is the predicate)

- We can obtain probabilities from expectations

$$\mathbb{P}(predicate) = \mathbb{E}[[\![predicate]\!]]$$

- E.g. The CDF is given by

$$F_X(x) = \mathbb{P}(X \le x) = \mathbb{E}[[\![X \le x]\!]]$$

Outline

1. Random Variables
2. Expectations
3. **Calculus of Probabilities**

Ω

x_{31}	x_{32}	x_{33}	x_{34}	x_{35}
x_{26}	x_{27}	x_{28}	x_{29}	x_{30}
x_{21}	x_{22}	x_{23}	x_{24}	x_{25}
x_{16}	x_{17}	x_{18}	x_{19}	x_{20}
x_{11}	x_{12}	x_{13}	x_{14}	x_{15}
x_{6}	x_{7}	x_{8}	x_{9}	x_{10}
x_{1}	x_{2}	x_{3}	x_{4}	x_{5}

Joint Probabilities

- Often we want to model complex processes where we have multiple random variables
- We can define the joint probability

$$p_{X,Y}(x,y) = \mathbb{P}(X=x, Y=y)$$

  i.e. the probability of the event where both $X=x$ and $Y=y$
- Clearly $\mathbb{P}(X,Y) = \mathbb{P}(Y,X)$

Marginalisation

- Probabilities are extremely easy to manipulate (although lots of people struggle)
- One of the most useful properties is known as **marginalisation**

$$\mathbb{P}(X) = \sum_{y\in\mathcal{Y}} \mathbb{P}(X, Y=y)$$

  where $\mathcal{Y}$ is the set of values that the random variable Y takes
- Note that when we write $\mathbb{P}(X)$ we are saying this is true for all values that X can take
- Although obvious and easy this is extremely useful

Conditional Probability

- We can also define the probability of an event X given that $Y=y$ has occurred

$$\mathbb{P}(X \mid Y=y) = \frac{\mathbb{P}(X, Y=y)}{\mathbb{P}(Y=y)}$$

- In constructing a model it is often much easier to specify conditional probabilities (because you know something) rather than joint probabilities
- When manipulating probabilities it is often easier to work with joint probabilities because we can simplify them by marginalising out random variables we are not interested in

Basic Calculus

- To obtain the joint probability we can use

$$\mathbb{P}(X,Y) = \mathbb{P}(X|Y)\mathbb{P}(Y) = \mathbb{P}(Y|X)\mathbb{P}(X)$$

- This generalises to more random variables

$$\mathbb{P}(X,Y,Z) = \mathbb{P}(X,Y|Z)\mathbb{P}(Z) = \mathbb{P}(X|Y,Z)\mathbb{P}(Y|Z)\mathbb{P}(Z)$$

- We can do this in a number of different ways

$$\mathbb{P}(X,Y,Z) = \mathbb{P}(Y,Z|X)\mathbb{P}(X) = \mathbb{P}(Z|Y,X)\mathbb{P}(Y|X)\mathbb{P}(X)$$

- Note that $\mathbb{P}(A,B \mid X,Y)$ means the probability of random variables A and B given that X and Y take particular values

Beware

- Conditional probabilities, $\mathbb{P}(X \mid Y)$ are probabilities for X, but not Y

$$\sum_{x\in\mathcal{X}} \mathbb{P}(X=x \mid Y) = 1$$
$$\sum_{y\in\mathcal{Y}} \mathbb{P}(X \mid Y=y) \neq 1$$

  (in general)
- Note that

$$\mathbb{E}_Y[\mathbb{P}(X \mid Y)] = \sum_{y\in\mathcal{Y}} \mathbb{P}(Y=y)\mathbb{P}(X|Y=y)$$
$$= \sum_{y\in\mathcal{Y}} \mathbb{P}(X, Y=y) = \mathbb{P}(X)$$

Causality

- Conditional probabilities does not imply causality
- We might have causal relationships

$$\mathbb{P}(\text{pass} \mid \text{study}) = 0.9 \qquad \mathbb{P}(\text{pass} \mid \neg\text{study}) = 0.2$$

- But if we know $\mathbb{P}(\text{study}) = 0.8$ then we can compute

$$\mathbb{P}(\text{pass},\text{study}) = \mathbb{P}(\text{pass} \mid \text{study})\mathbb{P}(\text{study}) = 0.9 \times 0.8 = 0.72$$
$$\mathbb{P}(\text{pass},\neg\text{study}) = \mathbb{P}(\text{pass} \mid \neg\text{study})\mathbb{P}(\neg\text{study}) = 0.2 \times 0.2 = 0.04$$

  and

$$\mathbb{P}(\text{study} \mid \text{pass}) = \frac{\mathbb{P}(\text{pass},\text{study})}{\mathbb{P}(\text{pass})}$$
$$= \frac{\mathbb{P}(\text{pass},\text{study})}{\mathbb{P}(\text{pass},\text{study}) + \mathbb{P}(\text{pass},\neg\text{study})} = \frac{0.72}{0.72+0.04} \approx 0.947$$

Independence

- Random variables X and Y are said to be **independent** if

$$\mathbb{P}(X,Y) = \mathbb{P}(X)\mathbb{P}(Y)$$

- Because $\mathbb{P}(X,Y) = \mathbb{P}(X|Y)\mathbb{P}(Y)$ and $\mathbb{P}(X,Y) = \mathbb{P}(Y|X)\mathbb{P}(X)$ independence implies

$$\mathbb{P}(X|Y) = \mathbb{P}(X) \qquad \mathbb{P}(Y|X) = \mathbb{P}(Y)$$

- Probabilistic independence implies a mathematical co-incident not necessarily causal independence
- However causal independence implies probabilistic independence
- If $X \in \{1,2,3,4,5,6\}$ represents the outcome of tossing a coin and $Y \in \{1,2,3,4,5,6\}$ the outcome of rolling a dice then X and Y are independent

Well Conducted Experiments

- In well conducted experiments we expect the results we obtain are independent
- Let $\mathcal{D} = (X_1, X_2, \ldots, X_m)$ represents possible outcomes from a set of m well conducted experiments then

$$\mathbb{P}(\mathcal{D}) = \prod_{i=1}^{m} \mathbb{P}(X_i)$$

- Denoting a possible sentence I might say by $\mathcal{S} = (W_1, W_2, \ldots, W_m)$ then

$$\mathbb{P}(\mathcal{S}) \neq \prod_{i=1}^{m} \mathbb{P}(W_i)$$

otherwise it's time I retired

Conditional Independence

- Let $K(d)$ be a random variable measuring the amount you know about ML on day d of your revision
- From you revision schedule you can write down your belief

$$\mathbb{P}(K(d) \mid K(d-1), K(d-2), \ldots K(1))$$

- But a very reasonable model is

$$\mathbb{P}(K(d) \mid K(d-1), K(d-2), \ldots K(1)) = \mathbb{P}(K(d) \mid K(d-1))$$

what you are going to know today will just depend on what you knew yesterday

- We say that $K(d)$ is **conditionally independent** on $K(d-2)$, $K(d-3)$, etc. given $K(d-1)$

Conclusion

- To work with probabilities you need to know
  - How to go back and forward between joint probabilities and conditional probabilities
  - How to marginalise out variables
- You need to understand that for continuous outcomes, it makes sense to talk about the probability density
- You need to know that expectations are linear operators and the expectation of a constant is the constant
- You need to understand independence

Advanced Machine Learning

Bayesian Inference

Bayes, Conjugate Priors, Uninformative Priors

Outline

1. **Bayes' Rule**
2. Conjugate Priors
3. Uninformative Priors

Dealing with Uncertainty

- In machine learning we are attempting to make inference under uncertainty
- The natural language for discussing uncertainty is probability
- The natural framework for making inferences is Bayesian statistics
- However, this requires that we encode our prior knowledge of the problem and specify a likelihood
- In consequence, probabilistic methods tend to be bespoke, rather then general purpose black boxes

Revision on Bayes

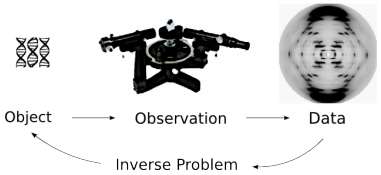
- Bayes' rule

$$\mathbb{P}(\mathcal{H}_i|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\mathcal{H}_i)\mathbb{P}(\mathcal{H}_i)}{\mathbb{P}(\mathcal{D})}$$

  - $\mathbb{P}(\mathcal{H}_i|\mathcal{D})$ is the **posterior** probability of a hypothesis $\mathcal{H}_i$ (i.e. the probability of $\mathcal{H}_i$ **after** we see the data)
  - $\mathbb{P}(\mathcal{D}|\mathcal{H}_i)$ is the **likelihood** of the data given the hypothesis. Note, that we calculated this from the forward problem
  - $\mathbb{P}(\mathcal{H}_i)$ is the **prior** probability (i.e. the probability of $\mathcal{H}_i$ **before** we see the data)
  - $\mathbb{P}(\mathcal{D})$ is the **evidence** or **marginal likelihood**

$$\mathbb{P}(\mathcal{D}) = \sum_{i=1}^{n} \mathbb{P}(\mathcal{H}_i, \mathcal{D}) = \sum_{i=1}^{n} \mathbb{P}(\mathcal{D}|\mathcal{H}_i)\,\mathbb{P}(\mathcal{H}_i)$$

Solving Inverse Problems



- We want the posterior $\mathbb{P}(\mathcal{H}_i|\mathcal{D})$ (i.e. the probability of what happened given some evidence)
- The Bayesian formalism converts this into the forward problem

$$\mathbb{P}(\mathcal{H}_i|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\mathcal{H}_i)\,\mathbb{P}(\mathcal{H}_i)}{\mathbb{P}(\mathcal{D})}$$

Bayesian Inference

- Bayes' rule says $\mathbb{P}(\mathcal{H}_i|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\mathcal{H}_i)\mathbb{P}(\mathcal{H}_i)}{\mathbb{P}(\mathcal{D})}$
- We calculate the likelihood $\mathbb{P}(\mathcal{D}|\mathcal{H}_i)$ (i.e. assuming the hypothesis, what is the chance of obtaining the data?)
- We consider the process of how the data is generated
- This uses the data we have (doesn't care about missing data)
- But we also need to know the prior $\mathbb{P}(\mathcal{H}_i)$
- Also, this can get difficult when we have many hypotheses

Evidence

- The normalisation term

$$\mathbb{P}(\mathcal{D}) = \sum_{i=1}^{n} \mathbb{P}(\mathcal{H}_i, \mathcal{D}) = \sum_{i=1}^{n} \mathbb{P}(\mathcal{D}|\mathcal{H}_i)\,\mathbb{P}(\mathcal{H}_i)$$

  tells you how likely the data is (given the prior and likelihood function)
- It is called the **marginal likelihood** or **evidence**
- If we have two models M_1 and M_2 we can do **model selection** by choosing the model with the largest evidence $\mathbb{P}(\mathcal{D} \mid M_1)$ or $\mathbb{P}(\mathcal{D} \mid M_2)$
- This also allows us to select hyperparameters for a model

Probability Density

- When we are working with continuous variables it is more natural to work with probability densities

$$f_X(x) = \lim_{\delta x \to 0} \frac{\mathbb{P}(x \le X < x + \delta x)}{\delta x}$$

- Note that densities are non-negative, but can be greater than 1 (they are not probabilities)
- However

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

  is a probability and is less than or equal to 1

Densities and Bayes

- Bayes' rule also applies to densities

$$\mathbb{P}(x \leq X < x + \delta x|Y) = \frac{\mathbb{P}(Y|x)\mathbb{P}(x \leq X < x + \delta x)}{\mathbb{P}(Y)}$$

- Dividing by δx and taking the limit $\delta x \to 0$

$$f_{X|Y}(x|Y) = \frac{\mathbb{P}(Y|x)\, f_X(x)}{\mathbb{P}(Y)}$$

- Similarly if X is discrete and Y continuous

$$\mathbb{P}(X|y) = \frac{f_{Y|X}(y|X)\mathbb{P}(X)}{f_Y(y)}$$

- If both X and Y are continuous

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)f_X(x)}{f_Y(y)}$$

Practical Bayesian Inference

- Often consider learning parameters $\boldsymbol{\theta}$

$$p(\boldsymbol{\theta}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{p(\mathcal{D})}$$

- This can be hard for large data sets as the posterior, $p(\boldsymbol{\theta}|\mathcal{D})$, is often a mess
- If we are lucky and have a simple likelihood then if we choose the right prior we end up with a posterior of the same form as the prior
- This occurs in some classic probabilistic inference problems, but as we will see soon it is also true for Gaussian Processes

Outline

1. Bayes' Rule
2. **Conjugate Priors**
3. Uninformative Priors

Learning a Probability

- Suppose we have a coin and we want to establish the probability of a head
- We want to learn this from a series of independent trials
- (Independent trials with two possible outcomes are known in probability theory as Bernoulli trials)
- Let X_i equal 1 if the i^{th} trial is a head and 0 otherwise
- If the probability of a head is p then the **likelihood** of a X_i is

$$\mathbb{P}(X_i|p) = p^{X_i}(1-p)^{1-X_i} = \begin{cases} p & \text{if } X_i = 1 \\ (1-p) & \text{if } X_i = 0 \end{cases}$$

Prior

- We may have a prior belief (e.g. we have made a few trials or we see the coin looks like a normal penny)
- We will suppose we can model our prior belief in terms of a **Beta distribution**

$$f(p) = \text{Beta}(p|a,b) = \frac{p^{a-1}(1-p)^{b-1}}{B(a,b)}$$

- $B(a,b)$ is just a normalisation constant

$$B(a,b) = \int_0^1 p^{a-1}(1-p)^{b-1}\mathrm{d}p = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

- This is a useful function for modelling the distribution of a random variable in the range 0 to 1

Uninformative Prior

- Suppose we have no idea about p what should we do?
- Laplace (one of the first Bayesian's) suggested giving equal weighting to all values of p
- This corresponds to a beta distribution with $a = b = 1$
- (Surprisingly other arguments suggest using $a = b = 0$ which provides a strong bias towards $p = 0$ and $p = 1$)
- Given enough data the prior is not so important and we will stick with Laplace for now

Independent Trials

- Using Bayes' rule

$$f(p|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|p)\, f(p)}{\mathbb{P}(\mathcal{D})}$$

- Assuming the trials are independent (a reasonably fair assumption for tossing coins) then the likelihood factorises

$$\begin{aligned}\mathbb{P}(\mathcal{D}|p) &= \prod_{i=1}^{n} p^{X_i}(1-p)^{1-X_i} \\ &= p^{X_1}(1-p)^{1-X_1}p^{X_2}(1-p)^{1-X_2}\cdots p^{X_n}(1-p)^{1-X_n} \\ &= p^{\sum_i X_i}(1-p)^{\sum_i (1-X_i)} = p^s(1-p)^{n-s}\end{aligned}$$

$s = \sum_i X_i$ (number of successes/heads)

Posterior

- Plugging in a prior $f(p) = \text{Beta}(p|a_0,b_0)$

$$f(p|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|p)\, f(p)}{\mathbb{P}(\mathcal{D})} = \frac{p^s(1-p)^{n-s} \times p^{a_0-1}(1-p)^{b_0-1}}{\mathbb{P}(\mathcal{D})\, B(a_0,b_0)}$$

- The denominator is a normalising factor

$$\begin{aligned}\mathbb{P}(\mathcal{D}) &= \int_0^1 \mathbb{P}(\mathcal{D}|p)\, f(p)\,\mathrm{d}p = \int_0^1 \frac{p^{s+a_0-1}(1-p)^{n-s+b_0-1}}{B(a_0,b_0)}\mathrm{d}p \\ &= \frac{B(s+a_0, n-s+b_0)}{B(a_0,b_0)}\end{aligned}$$

Conjugate Priors

- The posterior distribution is Beta distribution

$$f(p|\mathcal{D}) = \frac{p^{s+a_0-1}(1-p)^{n-s+b_0-1}}{B(s+a_0, n-s+b_0)} = \text{Beta}(p|s+a_0, n-s+b_0)$$

- Something rather nice happened
- Starting with a beta distributed prior $f(p) = \text{Beta}(p|a_0, b_0)$ for a set of Bernoulli trials we obtain a beta distributed posterior $f(p|\mathcal{D}) = \text{Beta}(p|a_0+s, b_0+n-s)$
- This is not always the case (often the posterior will be very complicated) but it happens for a few likelihoods and priors
- When the posterior is the same as the prior then the likelihood and prior distributions are said to be **conjugate**

Incremental Updating

- For independent data we can update incrementally $\mathcal{D} = (X_1, X_2, \ldots, X_n)$

$$f(p|X_1) = \frac{\mathbb{P}(X_1|p)f(p)}{\mathbb{P}(X_1)}$$

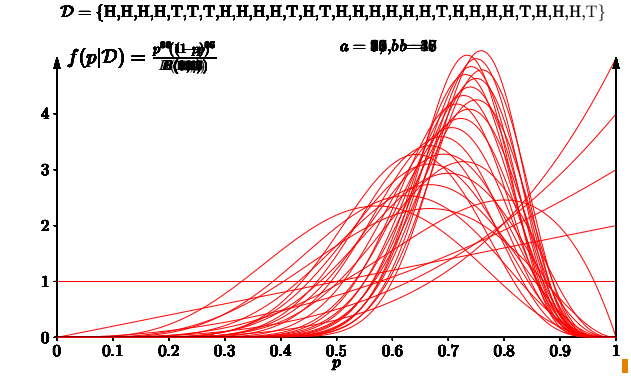
$$f(p|X_1, X_2) = \frac{\mathbb{P}(X_2|p)f(p|X_1)}{\mathbb{P}(X_2)}$$

$$\vdots \quad = \quad \vdots$$

$$f(p|X_1, X_2, \ldots, X_n) = \frac{\mathbb{P}(X_n|p)f(p|X_1, \ldots, X_{n-1})}{\mathbb{P}(X_n)}$$

- The posterior becomes the prior for the next piece of data
- For our problem the posterior is always Beta distributed

Example (p=0.7)



Estimating Prediction Errors

- A full Bayesian treatment gives a prediction of its own error
- Assuming $f(p|\mathcal{D}) = \text{Beta}(p|a,b)$
- The expected value of p is given by $a/(a+b) = 23/32 = 0.719$
- The standard deviation is

$$\sqrt{\frac{ab}{(a+b)^2(a+b+1)}} = 0.078$$

Poisson Likelihoods

- Let's look at a second example of conjugate priors
- Suppose we want to find the rate of traffic along a road between 1:00pm and 2:00pm
- We assume the number of cars is given by a Poisson distribution

$$\mathbb{P}(N) = \text{Pois}(N|\mu) = \frac{\mu^N}{N!}\mathrm{e}^{-\mu}$$

- μ is the rate of traffic per hour which we want to infer from observation taken on different days

Using Bayes

- Let us assume a Gamma distributed prior

$$p(\mu) = \Gamma(\mu|a_0, b_0) = \frac{b_0^{a_0}\mu^{a_0-1}\mathrm{e}^{-b_0\mu}}{\Gamma(a)}$$

- We will assume that we know nothing. The uninformative prior is $a_0 = b_0 = 0$
- The data is $\mathcal{D} = \{N_1, N_2, \ldots, N_n\}$
- The likelihood is $\text{Pois}(N_i|\mu)$

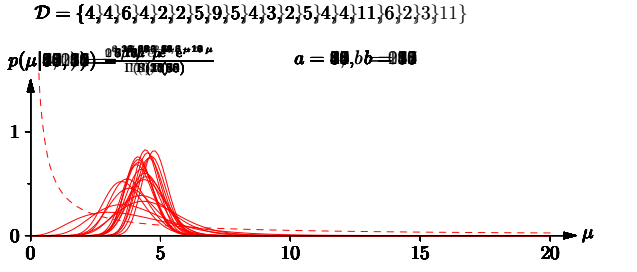
Posterior

- The posterior after seeing the first piece of data is

$$\begin{aligned} p(\mu|N_1) &\propto \mathbb{P}(N_1|\mu)\,p(\mu) \\ &\propto \frac{\mu^{N_1}}{N_1!}\mathrm{e}^{-\mu}\mu^{a_0-1}\mathrm{e}^{-b_0\mu} \\ &\propto \mu^{N_1+a_0-1}\mathrm{e}^{-(b_0+1)\mu} \end{aligned}$$

- The posterior is also a Gamma distribution $\Gamma(\mu|a_1, b_1)$ with $a_1 = a_0 + N_1$, $b_1 = b_0 + 1$

Example ($\mu = 5$)



$$\mathbb{E}[\mu] = \frac{a}{b} = \frac{96}{20} = 4.8 \qquad \sqrt{\mathbb{V}\text{ar}(\mu)} = \sqrt{\frac{a}{b^2}} = 0.49$$

Outline

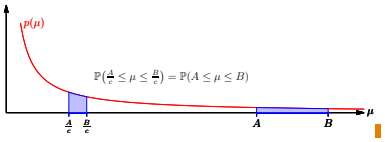
1. Bayes' Rule
2. Conjugate Priors
3. **Uninformative Priors**

Uninformative Priors

- What if we have no prior knowledge, what should we do?
- OK usually we know whether we should make a measurement using a micrometer, ruler or car mileage, but we might still know almost nothing
- This led to Bayesian statistics being labelled as *subjective*
- However Ed. Jaynes (the greatest proponent of Bayesian methods) argued that we could answer this using symmetry arguments

Uninformative Priors for Scale Parameter

- Why did we choose $a_0 = b_0 = 0$ implying a prior $p(\mu) = 1/\mu$?



- That is, we have no idea on what scale to measure μ

$$\int_A^B p(\mu)\mathrm{d}\mu = \int_{A/c}^{B/c} p(\mu)\mathrm{d}\mu = \int_A^B \frac{1}{c}p(\frac{\nu}{c})\mathrm{d}\nu = \int_A^B \frac{1}{c}p(\frac{\mu}{c})\mathrm{d}\mu$$

making a change of variables $\mu = \nu/c$

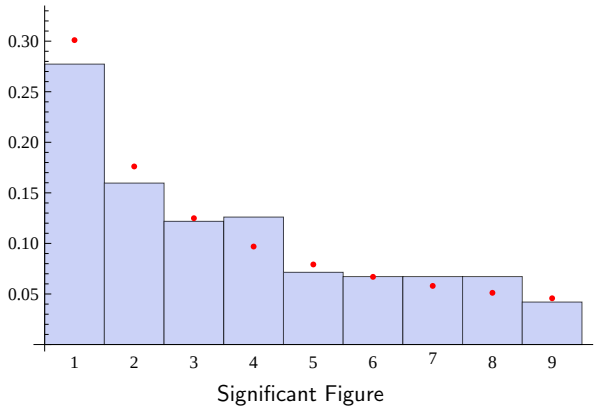
- Or $p(\mu) = \frac{1}{c}p(\frac{\mu}{c})$ implying $p(\mu) \propto \frac{1}{\mu}$

Benford's Law

- Numbers occurring in life (physical constants, amounts of money) should not depend on the units (scale) measuring them
- They should then be distributed as $p(x) \propto 1/x$
- A curious consequence of this is that the significant figure has a distribution

$$\mathbb{P}(\text{most s.f. of } x = n) = \frac{\int_n^{n+1}\frac{1}{x}\mathrm{d}x}{\int_1^{10}\frac{1}{x}\mathrm{d}x} = \frac{\int_{10n}^{10n+10}\frac{1}{x}\mathrm{d}x}{\int_{10}^{100}\frac{1}{x}\mathrm{d}x}$$
$$= \frac{\log(n+1)-\log(n)}{\log(10)} = \log_{10}\left(\frac{n+1}{n}\right)$$

Population Size of 238 Countries

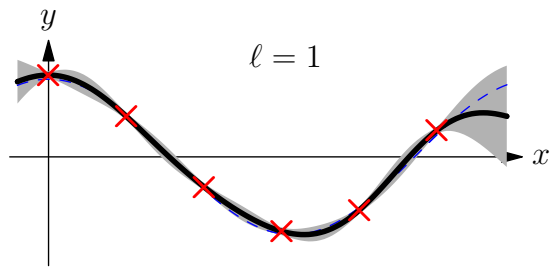


Conclusion

- Bayesian inference provides a coherent framework which we can use for machine learning
- However, it requires a model of what is happening
- In practice Bayesian methods are easy if the data is generated from a likelihood with a conjugate prior distribution—we have to be clever to choose the right prior
- We will see in the next lecture that much more frequently we will have likelihoods with no conjugate prior and we have to work much harder
- When we have no knowledge there are consistent ways to express our ignorance

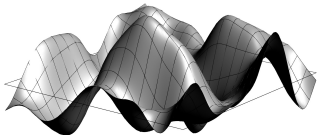
Advanced Machine Learning

Gaussian Processes



Gaussian Processes, regression

Outline



Gaussian Proccesses

- Gaussian processes (GPs) are a mathematically defined ensemble of functions
- They can be combined with Bayesian inference to give one of the most powerful regression techniques
- Although Bayesian they can be used in a black-box fashion due to the ubiquity of the prior
- Mathematically they are a bit complicated (because Gaussians involve the inverse of matrices which are a real pain to work with)
- In practice they aren't that difficult to use

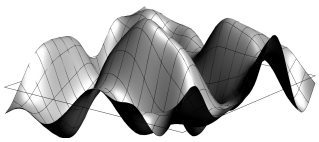
Regression

- In regression we try to fit a multi-dimensional function to our data
- (You can use Gaussian Processes for classification, e.g. by inferring the probabilities of being in a class, but we ignore this as regression is where GP excel)
- In regression we have some p dimensional feature vectors $\boldsymbol{x}_i$ and some target $y_i \in \mathbb{R}$
- Our task is to fit a function through all the data points

Priors on Functions

- We can think of a solution as a function $f(\boldsymbol{x})$
- We can put a prior probability distribution, $p(f)$, on a function, f, that prefers smooth functions
- We can then compute a posterior probability distribution on functions given the data, $p(f|\mathcal{D})$
- As a likelihood, $p(y_i|f(\boldsymbol{x}_i))$, we use the probability of observing y_i given the true function value is $f(\boldsymbol{x}_i)$
- In general, this would be next to impossible to compute, except in the special case where everything is Gaussian (normally) distributed

Outline



Gaussian Processes

- Gaussian Processes are probability distributions over functions
- (Functions can be viewed as vectors in an infinite dimensional vector space)
- In the Gaussian Process, $\mathcal{GP}(m,k)$, the probability of a function, f, is proportional

$$p(f|m,k) \propto \mathrm{e}^{-\frac{1}{2}\int (f(\boldsymbol{x})-m(\boldsymbol{x}))k^{-1}(\boldsymbol{x},\boldsymbol{y})(f(\boldsymbol{y})-m(\boldsymbol{y}))\mathrm{d}\boldsymbol{x}\mathrm{d}\boldsymbol{y}}$$

- The function $m(\boldsymbol{x})$ is the mean $\mathbb{E}[f(\boldsymbol{x})]$ (usually taken to be zero in most inference problems)

Meaning of GP

- To understand GP's we can discretise space, $\boldsymbol{x}$, into a lattice of points $\{\boldsymbol{x}_i\}$
- Then (assuming $m(\boldsymbol{x}) = 0$)

$$p(f|m,k) \propto \prod_i \mathrm{e}^{-\frac{f_i^2 k^{-1}(\boldsymbol{x}_i,\boldsymbol{x}_i)}{2} + f_i \sum_j k^{-1}(\boldsymbol{x}_i,\boldsymbol{x}_j) f_j}$$

  where $f_i = f(\boldsymbol{x}_i)$
- We see that the value of the function at each point is normally distributed with a mean that depends on functions at neighbouring points

Covariance function

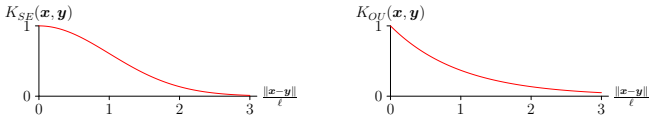
- $k(\boldsymbol{x},\boldsymbol{y})$ is a covariance function

$$\mathbb{E}\left[\left(f(\boldsymbol{x})-m(\boldsymbol{x})\right)\left(f(\boldsymbol{y})-m(\boldsymbol{y})\right)\right]=k(\boldsymbol{x},\boldsymbol{y})$$

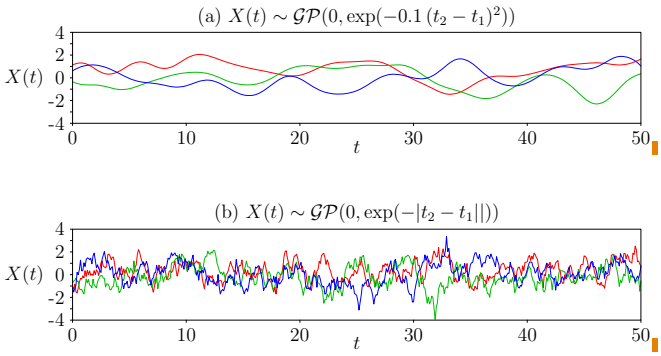
- This is sometimes know as a **kernel**—it must be positive semi-definite (just like in SVMs)
- It is a free "parameter" that the user gets to choose (although we can learn its parameters too)
- If $k(\boldsymbol{x},\boldsymbol{y})$ is a function of $\boldsymbol{x}-\boldsymbol{y}$ it is **"stationary"**
- If $k(\boldsymbol{x},\boldsymbol{y})$ is a function of $\|\boldsymbol{x}-\boldsymbol{y}\|$ it is also **"isometric"**

Popular Choices of GP Kernel Function

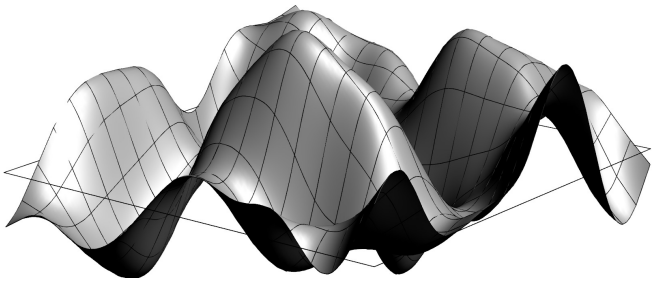
- Constant: $k_{\mathrm{C}}(\boldsymbol{x},\boldsymbol{y})=C$
- Gaussian noise: $k_{\mathrm{GN}}(\boldsymbol{x},\boldsymbol{y})=\sigma^2\delta_{\boldsymbol{x},\boldsymbol{y}}$
- Squared exponential: $k_{\mathrm{SE}}(\boldsymbol{x},\boldsymbol{y})=\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|^2}{2\ell^2}\right)$
- Ornstein–Uhlenbeck: $k_{\mathrm{OU}}(\boldsymbol{x},\boldsymbol{y})=\exp\left(-\frac{\|\boldsymbol{x}-\boldsymbol{y}\|}{\ell}\right)$



Gaussian Process Worlds

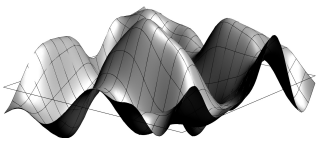


2-D Gaussian Processes



Outline

1. Introduction
2. Gaussian Processes
3. **Bayesian Inference**
4. Hyper-parameters



Observed Gaussian Processes

- Given some data points $\mathcal{D}=\left((\boldsymbol{x}_i,y_i)\middle|i=1,\ldots,m\right)$ the likelihood (assuming Gaussian error are independence of the data point) is given by

$$p(\mathcal{D}|f)=\prod_{i=1}^{m}\mathcal{N}\left(y_i\middle|f(\boldsymbol{x}_i),\sigma^2\right)$$

- Using a Gausssian Process prior we can compute a posterior using Bayes's rule
- The posterior is a Gaussian Process with a shifted mean and variance depending on the data-points
- This direct Bayesian derivation gives the answer involving the inverse matrix of the correlation function, $k^{-1}(\boldsymbol{x},\boldsymbol{y})$—this is a pain to work with

Alternative Derivation

- Denoting the target values as a vector $\boldsymbol{y}$ with elements y_i
- Denoting the matrices of covariances between data points as $\mathbf{K}$ with elements $k(\boldsymbol{x}_i,\boldsymbol{x}_j)$
- Denoting the covariance between the data points and a particular position, $\boldsymbol{x}_*$ as $\boldsymbol{k}_*$ with elements $k(\boldsymbol{x}_i,\boldsymbol{x}_*)$
- Denoting the variance a point $\boldsymbol{x}_*$ as $k_*=k(\boldsymbol{x}_*,\boldsymbol{x}_*)$
- Then the distribution of function values at points at $\boldsymbol{x}_i$ and $\boldsymbol{x}_*$ is

$$p(\boldsymbol{y},f_*)=\mathcal{N}\left(\begin{pmatrix}\boldsymbol{y}\\ f_*\end{pmatrix}\middle|\boldsymbol{0},\begin{pmatrix}\mathbf{K}+\sigma^2\mathbf{I} & \boldsymbol{k}_*\\ \boldsymbol{k}_*^{\mathsf{T}} & k_*\end{pmatrix}\right)$$

Conditional Probability

- To compute the posterior $p(f_*|\boldsymbol{y})$ we use

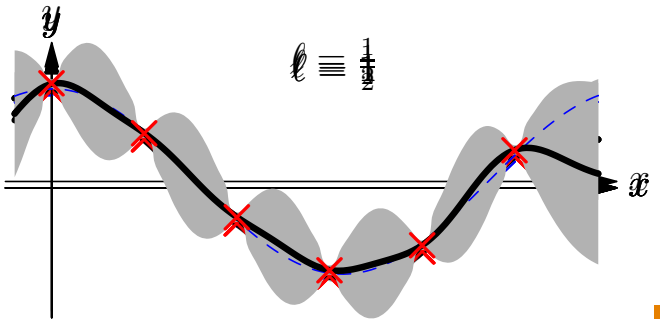
$$p(f_*|\boldsymbol{y})=\frac{p(f_*,\boldsymbol{y})}{p(\boldsymbol{y})}$$

- where $p(\boldsymbol{y})=\int p(f_*,\boldsymbol{y})\,\mathrm{d}f_*$
- Because all integrals are Gaussian we can compute the integral to obtain

$$p(f_*|\boldsymbol{y})=\mathcal{N}\left(f_*\middle|\boldsymbol{k}_*^{\mathsf{T}}(\mathbf{K}+\sigma^2\mathbf{I})^{-1}\boldsymbol{y},k-\boldsymbol{k}_*^{\mathsf{T}}(\mathbf{K}+\sigma^2\mathbf{I})^{-1}\boldsymbol{k}_*\right)$$

- Looks complicated, but numerically easy to evaluate

$$K(x,x') = \exp(-(x-x')^2/(2\ell^2))$$

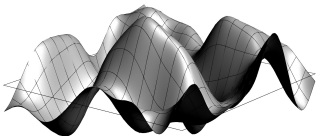


Multi-dimensional Regression

- I've shown a 1-D regression example because it is easy to visualise
- This might be used with a time series
- The much more typical situation in machine learning is for $\boldsymbol{x}$ to have many features so we are doing multi-dimensional regression
- Gaussian process inference were first used in spatial problems where it was known as **krigging**
- It was re-invented by the machine learning community who call it Gaussian Processes (GP)

Outline

1. Introduction
2. Gaussian Processes
3. Bayesian Inference
4. **Hyper-parameters**



Choosing the Correct Covariance Function

- Choosing the correct covariance function is critical
- Most covariance functions include a continuous **hyper-parameter** (e.g. the correlation length ℓ) that we have to choose correctly
- This is typical of many Bayesian problems were we have some set of hyper-parameters, $\boldsymbol{\phi}$, describing the model
- These are different to the normal parameters we learn (e.g. weights $\boldsymbol{w}$ or in GP the functions $f(\boldsymbol{x})$)
- In Bayesian inference we learn the posterior for these normal parameters

$$p(f|\mathcal{D},\boldsymbol{\phi}) = \frac{p(\mathcal{D}|f,\boldsymbol{\phi})p(f|\boldsymbol{\phi})}{p(\mathcal{D}|\boldsymbol{\phi})}$$

Evidence Framework

- The normalisation factor, $p(\mathcal{D}|\boldsymbol{\phi})$ is known as the **marginal likelihood** or **evidence**

$$p(\mathcal{D}|\boldsymbol{\phi}) = \int p(\mathcal{D}|f,\boldsymbol{\phi})p(f|\boldsymbol{\phi})\mathrm{d}f$$

- We can perform a Bayesian calculation at a second level by putting a prior on $\boldsymbol{\phi}$

$$p(\boldsymbol{\phi}|\mathcal{D}) = \frac{p(\mathcal{D}|\boldsymbol{\phi})p(\boldsymbol{\phi})}{p(\mathcal{D})}$$

- From this we can now marginalise out the hyper-parameters

$$p(f|\mathcal{D}) = \int p(f|\mathcal{D},\boldsymbol{\phi})p(\boldsymbol{\phi}|\mathcal{D})\mathrm{d}\boldsymbol{\phi}$$

Maximum-Likelihood-II

- The integral

$$p(f|\mathcal{D}) = \int p(f|\mathcal{D},\boldsymbol{\phi})p(\boldsymbol{\phi}|\mathcal{D})\mathrm{d}\boldsymbol{\phi}$$

usually can't be computed analytically and we have to use Monte Carlo methods (see later lecture)

- An alternative is to use the most likely hyper-parameter
- We can find this by using gradient search of $p(\mathcal{D}|\boldsymbol{\phi})$
- This is sometimes referred to as ML-II
- Normally even this can be difficult, but for GP its not too difficult

Evidence for GP

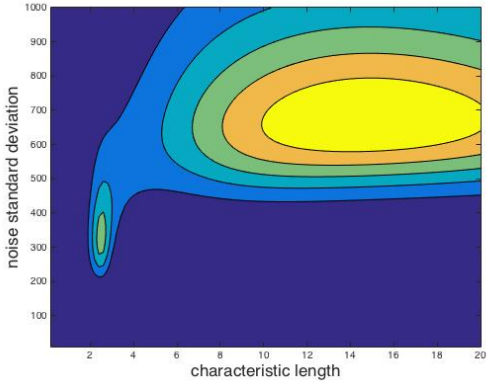
- For GP the (log)-evidence can be computed in closed form

$$\log(p(\mathcal{D}|\boldsymbol{\phi})) = -\frac{1}{2}\boldsymbol{y}^\mathsf{T}(\mathbf{K}+\sigma^2\mathbf{I})\boldsymbol{y} - \frac{1}{2}\log(|\mathbf{K}+\sigma^2\mathbf{I}|) - \frac{m}{2}\log(2\pi)$$

  - First term measures goodness of fit
  - Second term measure complexity of model
  - Last term is a common normalisation constant

- Can efficiently compute derivatives and find best parameters
- Could overfit!

Example (slightly pathological)

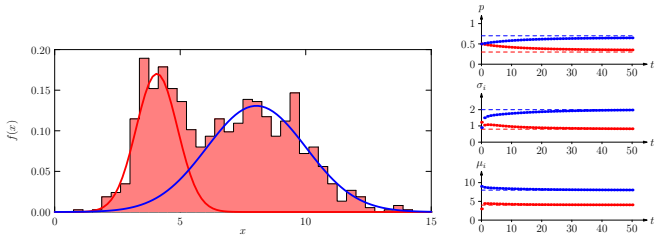


Conclusions

- Gaussian processes are very powerful for regression (and classification?)
- Because all calculations involve Gaussian integrals we can compute everything in closed form
- (Actually its a pain to do the mathematics because you end up working with inverse of matrices)
- Fairly generic (black-box) technique because the prior captures many continuity constraints
- We can use the evidence framework (probability of data) to do model selection and hyper-parameter optimisations

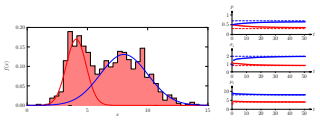
Advanced Machine Learning

Probabilistic Inference



Hierarchical Models, Mixture of Gaussians, Expectation Maximisation

Outline



Building Probabilistic Models

- To describe a system with uncertainty we use random variables, X, Y, Z, etc.
- We use the convention of writing random variables in capitals (this is sometimes confusing as when you observe a random variables it is no longer random)
- The variables are described by probability mass function $\mathbb{P}(X,Y,Z)$ or if our variables are continuous, but probability densities $f_{X,Y,Z}(x,y,z)$
- A major rule of probability is

$$\sum_X \mathbb{P}(X,Y,Z) = \mathbb{P}(Y,Z)$$

Conditional Probabilities

- When developing models it is often useful to consider conditional probabilities e.g. $\mathbb{P}(X,Y|Z)$ or $f_{X|Y,Z}(x|y,z)$
- A second major rule in probabilistic modelling is

$$\mathbb{P}(X,Y) = \mathbb{P}(X|Y)\mathbb{P}(Y) = \mathbb{P}(Y|X)\mathbb{P}(X)$$

- This is a mathematical identity that does not imply causality (it defines conditional probability)
- It is the origins of Bayes' rule: $\mathbb{P}(X|Y) = \dfrac{\mathbb{P}(Y|X)\mathbb{P}(X)}{\mathbb{P}(Y)}$

Discriminative Models

- We often think of our observations as given and the predictions as random variables
- For example we might be given some features $\boldsymbol{x}$ and we wish to predict a class $C \in \mathcal{C}$
- Our objective is then to find the probability $\mathbb{P}(C|\boldsymbol{x})$
- This is known as a **discriminative model**
- E.g. in *foundations of machine learning* you learnt how to find the Bayes' optimal discrimination surface

Generative Models

- Sometimes it is easy to think about the joint process of generating the features and outputs together
- This leads to a joint distribution $\mathbb{P}(\boldsymbol{X},Y)$ where $\boldsymbol{X}$ are your features and Y is your output you are trying to predict
- This is known as a **generative model**
- Generative models are often more natural to think about
- We can use them to do discrimination using

$$\mathbb{P}(Y|\boldsymbol{X}) = \frac{\mathbb{P}(\boldsymbol{X},Y)}{\mathbb{P}(\boldsymbol{X})} = \frac{\mathbb{P}(\boldsymbol{X},Y)}{\sum_Y \mathbb{P}(\boldsymbol{X},Y)}$$

Latent Variables

- Sometimes we have models that involve random variables that we don't observe and we don't care about
- These are called **latent variables**
- If we have a latent variable Z and observed variable $\boldsymbol{X}$ and we are predicting a variable Y then we would **marginalise** over the latent variable

$$\mathbb{P}(\boldsymbol{X},Y) = \sum_Z \mathbb{P}(\boldsymbol{X},Y,Z)$$

Modelling Virus

- Suppose we want to estimate the number of hospitalisation from Corona virus in the next month
- Our observable is the number of reported cases
- In our model we might want to estimate the number of actual cases
- This would be a latent variable (it is not an observable or our final target, but it is very useful intermediate in our model)
- This will be a random variable (we are uncertain, but we can build a probabilistic model giving a distribution of number of actual cases)

Hierarchical Models

- Of course, if I was really modelling the spread of a disease I would care about the probability, $f(C|A,V)$, of catching the disease, C, given the persons age A and the variant of the disease V
- I would want to know the distribution of ages $f(A)$ and try to infer the probability of different variants $\mathbb{P}(V)$
- I would care about the probability, $f(R|A,V)$, of cases being reported given age and variant
- And the probability, $f(H|A,V)$, of hospitalisation given A and V
- This would involve an elaborate (hierarchical) model with a large number of latent variables

Probabilistic Inference

- We can use Bayes' rules to learn a set of parameter $\boldsymbol{\Theta}$ that occur in our likelihood function

$$\mathbb{P}(\boldsymbol{\Theta}|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\boldsymbol{\Theta})\,\mathbb{P}(\boldsymbol{\Theta})}{\mathbb{P}(\mathcal{D})}$$

- This provides us a full probabilistic description of the parameters
- It doesn't overfit (we are not choosing the best)
- Bayesian inference provides a description of its own uncertainty
- We need to specify a likelihood and prior, but this is usually not difficult

Problem with Bayes

- Bayes is problematic because it is often hard
- The posterior is often not expressible as a nice probability function
- We need to compute the *evidence* or *margin likelihood* we use

$$\mathbb{P}(\mathcal{D}) = \sum_{\boldsymbol{\Theta}} \mathbb{P}(\mathcal{D}|\boldsymbol{\Theta})\,\mathbb{P}(\boldsymbol{\Theta})$$

- But sometimes the number of values that $\boldsymbol{\Theta}$ can take are so large that we cannot easily compute this
- Nevertheless we can usually do this using Monte Carlo techniques

Maximum A Posteriori (MAP) Solution

- One work around is to compute the mode of the posterior

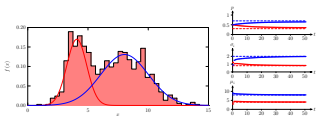
$$\boldsymbol{\Theta}_{\text{MAP}} = \operatorname*{argmax}_{\boldsymbol{\Theta}} f(\mathcal{D}|\boldsymbol{\Theta})\,f(\boldsymbol{\Theta}) = \operatorname*{argmax}_{\boldsymbol{\Theta}} \log(f(\mathcal{D}|\boldsymbol{\Theta})) + \log(f(\boldsymbol{\Theta}))$$

- We don't need to calculate $f(\mathcal{D})$ or explicitly calculate the posterior distribution
- But it is not Bayesian (despite what you are sometime told)—its not properly probabilistic
- You can overfit and you don't get an estimate of the error in your inference

Maximum Likelihood

- When we assume a uniform prior then the MAP solution is just maximising the likelihood
- Weirdly this hack was accepted as part of mainstream statistics even when Bayesian statistics was considered unscientific
- Maximum likelihood is often sufficient for *government work*, but it isn't the best you can do
- In high-dimensional problems using a non-uniform prior can make a big difference
- And, of course, doing a full probabilistic calculation has real advantages

Outline



Mixture of Gaussians

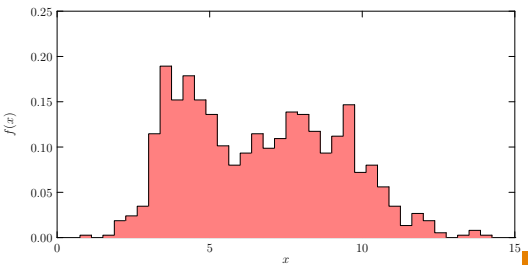
- Suppose we were observing the decays from two types of short-lived particle, A or B
- We observe the half life, X_i, but not the particle type
- We assume X_i is normally distributed with unknown means and variances: $\boldsymbol{\Theta} = \{\mu_A,\, \sigma_A^2,\, \mu_B,\, \sigma_B^2\}$
- Let $Z_i \in \{0,1\}$ be an indicator that particle i is of type A
- The probability of X_i is given by

$$f(X_i|Z_i,\boldsymbol{\Theta}) = Z_i\mathcal{N}\big(X_i|\mu_A,\sigma_A^2\big) + (1-Z_i)\mathcal{N}\big(X_i|\mu_B,\sigma_B^2\big)$$

Data

- Note that

$$f(X_i|\boldsymbol{\Theta}) = \sum_{Z_i\in\{0,1\}} f(X_i,Z_i|\boldsymbol{\Theta}) = \sum_{Z_i\in\{0,1\}} f(X_i|Z_i,\boldsymbol{\Theta})\,\mathbb{P}(Z_i)$$
$$= \mathbb{E}_{Z_i}[f(X_i|Z_i,\boldsymbol{\Theta})] = p\mathcal{N}\big(X_i|\mu_A,\sigma_A^2\big) + (1-p)\mathcal{N}\big(X_i|\mu_B,\sigma_B^2\big)$$



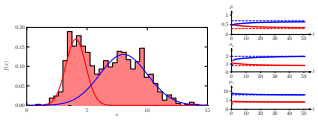
Maximum Likelihood

- To solve the model as a Bayesian we would have to assign priors to our parameters $\boldsymbol{\Theta} = (\mu_A, \sigma_A, \mu_B, \sigma_B, p)$
- This is doable, but complicated (we would also end up with a distribution for our parameters)
- Often we only want a reasonable estimate for some of our parameters (e.g. the half-lives μ_A and μ_B)
- A reasonable approach is to seek those parameters that maximise the likelihood of our observed data

$$f(\mathcal{D}|\boldsymbol{\Theta}) = \prod_{X \in \mathcal{D}} f(X|\boldsymbol{\Theta})$$

Outline

1. Building Probabilistic Models
2. Mixture of Gaussians
3. **Expectation Maximisation**



Maximum Likelihood with Latent Variables

- The maximum likelihood is a non-linear function of the parameters so cannot be immediately maximised
- If we knew which type of particle a data-point belongs to (Z_i) then it would be straightforward to maximise the likelihood
- As we don't we need to estimate $\mathbb{P}(Z_i = 1)$, but this depends on μ_A, σ_A^2, μ_B, σ_B^2 and p
- We could use a standard optimiser, but this is slightly inelegant

EM Algorithm

- Instead we can use an **expectation-maximisation algorithm** usually known as an **EM algorithm**
- We proceed iteratively by maximising the expected log-likelihood with respect to the current set of parameters

$$\Theta^{(t+1)} = \underset{\boldsymbol{\Theta}}{\text{argmax}} \sum_{\boldsymbol{Z}} \mathbb{P}\Big(\boldsymbol{Z}|\mathcal{D}, \boldsymbol{\Theta}^{(t)}\Big) \log(f(\mathcal{D}|\boldsymbol{Z}, \boldsymbol{\Theta}))$$

- It isn't obvious why this works

Why EM Algorithm Works

- The argument around why this works is quite involved
- Note that at each step we maximise

$$Q(\boldsymbol{\Theta}|\boldsymbol{\Theta}^{(t)}) = \sum_{\boldsymbol{Z} \in \{0,1\}^m} \mathbb{P}\Big(\boldsymbol{Z}|\mathcal{D}, \boldsymbol{\Theta}^{(t)}\Big) \log(f(\mathcal{D}|\boldsymbol{Z}, \boldsymbol{\Theta}))$$

- We can show that the maximum, $\boldsymbol{\Theta}^{(t+1)}$, is such that

$$\log\Big(f(\mathcal{D}|\boldsymbol{\Theta}^{(t+1)})\Big) - \log\Big(f(\mathcal{D}|\boldsymbol{\Theta}^{(t)})\Big) \geq Q(\boldsymbol{\Theta}^{(t+1)}|\boldsymbol{\Theta}^{(t)}) - Q(\boldsymbol{\Theta}^{(t)}|\boldsymbol{\Theta}^{(t)}) \geq 0$$

- The details are given in the supplemental notes

Conditional Latent Variables

- We need to compute the distribution of latent variables conditioned on the data and current estimated parameters
- For our problem

$$\mathbb{P}\Big(\boldsymbol{Z}|\mathcal{D}, \boldsymbol{\Theta}^{(t)}\Big) = \prod_{i=1}^m \mathbb{P}\Big(Z_i|X_i, \boldsymbol{\Theta}^{(t)}\Big)$$

where

$$\mathbb{P}\Big(Z_i = 1|X_i, \boldsymbol{\Theta}^{(t)}\Big) = \frac{p^{(t)} \mathcal{N}\Big(X_i|\mu_A^{(t)}, \sigma_A^{2(t)}\Big)}{p^{(t)} \mathcal{N}\Big(X_i|\mu_A^{(t)}, \sigma_A^{2(t)}\Big) + (1-p^{(t)}) \mathcal{N}\Big(X_i|\mu_B^{(t)}, \sigma_B^{2(t)}\Big)}$$

$$\mathbb{P}\Big(Z_i = 0|X_i, \boldsymbol{\Theta}^{(t)}\Big) = 1 - \mathbb{P}\Big(Z_i = 1|X_i, \boldsymbol{\Theta}^{(t)}\Big)$$

EM for Mixture of Gaussians

- Maximise with respect to parameters $\boldsymbol{\theta}$

$$Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)}) = \sum_{\boldsymbol{Z}} \mathbb{P}\Big(\boldsymbol{Z}|\mathcal{D}, \boldsymbol{\Theta}^{(t)}\Big) \log(f(\mathcal{D}|\boldsymbol{Z}, \boldsymbol{\Theta})) = \sum_{i=1}^m \sum_{Z_i} \mathbb{P}\Big(Z_i|\mathcal{D}, \boldsymbol{\Theta}^{(t)}\Big) \log(f(X_i|Z_i, \boldsymbol{\Theta}))$$

$$= \sum_{i=1}^m \sum_{Z_i \in \{0,1\}} \mathbb{P}\Big(Z_i|X_i, \boldsymbol{\theta}_i^{(t)}\Big) \Big(Z_i \log(p) + (1-Z_i)\log(1-p) - \frac{(X_i - \mu_{Z_i})^2}{2\sigma_{Z_i}^2} - \log\Big(\sqrt{2\pi}\sigma_{Z_i}\Big)\Big)$$

- Compute update equations

$$\frac{\partial Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})}{\partial \mu_k} = 0, \qquad \frac{\partial Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})}{\partial \sigma_k} = 0, \qquad \frac{\partial Q(\boldsymbol{\theta}|\boldsymbol{\theta}^{(t)})}{\partial p} = 0$$

Update Equations

- Means

$$\mu_{Z_i}^{(t+1)} = \frac{\sum_{i=1}^n \mathbb{P}\big(Z_i|X_i, \boldsymbol{\theta}^{(t)}\big) X_i}{\sum_{i=1}^n \mathbb{P}\big(Z_i|X_i, \boldsymbol{\theta}^{(t)}\big)},$$

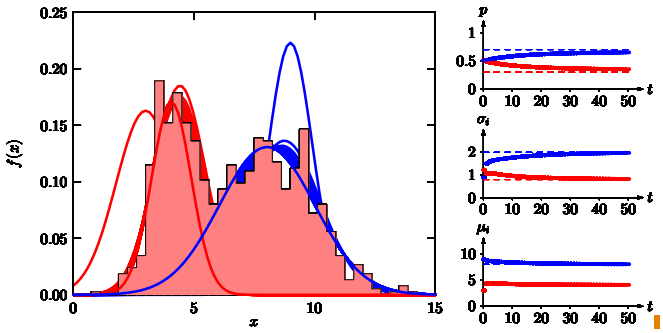
- Variances

$$(\sigma_{Z_i}^{(t+1)})^2 = \frac{\sum_{i=1}^n \mathbb{P}\big(Z_i|X_i, \boldsymbol{\theta}^{(t)}\big) (X_i - \mu_{Z_i}^{(t+1)})^2}{\sum_{i=1}^n \mathbb{P}\big(Z_i|X_i, \boldsymbol{\theta}^{(t)}\big)}$$

- Probability of being type 1

$$p^{(t+1)} = \frac{1}{n} \sum_{i=1}^n \mathbb{P}\Big(Z_i = 1 \mid X_i, \boldsymbol{\theta}_i^{(t)}\Big)$$

Example

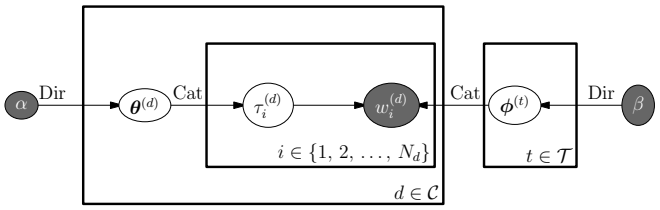


Summary

- Building probabilistic models is an intricate process
- Identifying random variables that describe the system is the first step
- Often we need to introduce variables that we don't observe and need to be marginalised out
- The EM algorithm provide one approach to maximising likelihoods or MAP solutions when we have latent variables
- It often gives nice update equations, but convergence can be slow

Advanced Machine Learning

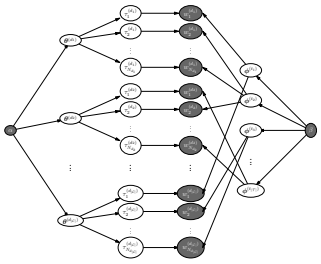
Graphical Models



Conditional Independence, Graphical models, LDA

Outline

1. **Graphical Models**
2. Cakes!
3. Latent Dirichlet Allocation



Graphical Models

- If we want to build large probabilistic inference systems
  - ⋆ AI Doctor
  - ⋆ Fault diagnostic system for a computer

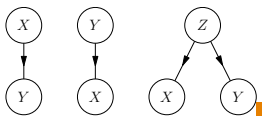
  we can describe this by introducing random variables, but it is helpful to graphically represent causal connections
- Graphical models allow us to do this
- It allows us to build a joint probability from which we can compute everything we want

Dependencies Between Variables

- In building a probabilistic model we want to know which random variables depend on each other directly and which don't
- Variables that don't will typically still be correlated
- If two random variables X and Y are correlated then
  - ⋆ X could affect Y
  - ⋆ Y could affect X
  - ⋆ X and Y could not influence each other, but both be affected by another random variable Z

Graphical Models

- **Bayesian Belief Networks** are a type of graphical models where we use a directed graphs to show causal relationships between random variables
- We could represent the three conditions described above by



- We can use these graphical representations to work out how to efficiently average over latent variables

Statistical Independence

- Two random variables are statistically independent if

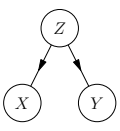
$$\mathbb{P}(X,Y) = \mathbb{P}(X)\mathbb{P}(Y)$$

- Equally this implies $\mathbb{P}(X|Y) = \mathbb{P}(X)$ and $\mathbb{P}(Y|X) = \mathbb{P}(Y)$
- Statistically independent variables are uncorrelated
- But statistical independence is often too powerful

Conditional Independence

- A weaker notion is conditional independence

$$\mathbb{P}(X,Y|Z) = \mathbb{P}(X|Z)\mathbb{P}(Y|Z)$$

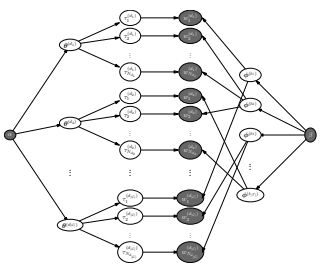


- Conditional independence implies that there is no direct causation
- But it doesn't imply zero correlation
- Conditional independence reduces computational complexity, e.g.

$$\mathbb{E}[XY] = \sum_{X,Y,Z} XY\mathbb{P}(X,Y,Z) = \sum_Z P(Z)\left(\sum_X XP(X|Z)\right)\left(\sum_Y YP(Y|Z)\right)$$

Outline

1. Graphical Models
2. **Cakes!**
3. Latent Dirichlet Allocation



Let Them Eat Cakes

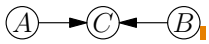
- I will go through a very simple example involving cakes
- It illustrates some simple principles
- In the subsidiary notes I present a very simple program for computing all the probabilities—I would encourage you to do this as it makes things much clearer

The Cake Scenario

- Abi and Ben both bake cakes and bring them into the coffee room
- Abi will bring in cakes 20% of the time: $\mathbb{P}(A=1)=0.2$
- Ben will bring in cakes 10% of the time: $\mathbb{P}(B=1)=0.1$
- 90% of the time if either Abi or Ben have put cakes in the coffee room there is some left when I enter
  $\mathbb{P}(C=1|A=1,B=0)=\mathbb{P}(C=1|A=0,B=1)=0.9$
- If they both make cake then there is always cake left
  $\mathbb{P}(C=1|A=1,B=1)=1$
- If neither Abi or Ben has made cake there is still a 5% chance someone else has put cake in the coffee room
  $\mathbb{P}(C=1|A=0,B=0)=0.05$

Computing with Probabilities

- Other probabilities I can deduce, e.g.
  $\mathbb{P}(C=0|A,B)=1-\mathbb{P}(C=1|A,B)$
- I can depict the causal relationship as



- The quantity that I really want is the joint probability

$$\mathbb{P}(A,B,C)=\mathbb{P}(C,B|A)\mathbb{P}(A)$$
$$=\mathbb{P}(C|A,B)\mathbb{P}(B|A)\mathbb{P}(A)=\mathbb{P}(C|A,B)\mathbb{P}(B)\mathbb{P}(A)$$

- Because $\mathbb{P}(B|A)=\mathbb{P}(B)$

Computing Expectations

- By using the joint probability and summing over all unknown quantities, we can compute expectations of anything we are interested in
- These sums are often sped up using knowledge of conditional independence
- To compute the probability of and event $\mathcal{E}$ we introduce an indicator function $[\![\mathcal{E}]\!]$ which is equal to 1 if the event happens and 0 otherwise
  $$\mathbb{P}(\mathcal{E})=\mathbb{E}[[\![\mathcal{E}]\!]]$$
- If E is a random variable equal to 1 if event $\mathcal{E}$ happens and 0 otherwise then $E=[\![\mathcal{E}]\!]$

Are There Any Cakes Left?

- We can use our model to compute the probabilities of there being cakes in the coffee room

$$\mathbb{P}(C=1)=\sum_{A,B,C\in\{0,1\}}[\![C=1]\!]\mathbb{P}(A,B,C)$$
$$=\sum_{A,B\in\{0,1\}}\mathbb{P}(C=1|A,B)\mathbb{P}(A)\mathbb{P}(B)=0.29$$

- The probability that Abi baked a cake is just 0.2 and for Ben its 0.1 (which is what we assume at the start)
- The probability of them both baking on a particular day is 0.02

Making Observation

- Making observations changes probabilities
- In graphical models observed random variables are shaded

- The probabilities conditioned on C is given by

$$\mathbb{P}(A,B|C)=\frac{\mathbb{P}(A,B,C)}{\mathbb{P}(C)}$$

where

$$\mathbb{P}(C)=\sum_{A,B\in\{0,1\}}\mathbb{P}(A,B,C)$$

Who Made Those Cakes?

- If we observe there are cakes

$$\mathbb{P}(A,B|C=1)=\mathbb{P}(A,B,C=1)/\mathbb{P}(C=1)$$

- A straightforward if tedious calculation shows

$$\mathbb{P}(A=1|C=1)=0.628,\quad \mathbb{P}(B=1|C=1)=0.317$$
$$\mathbb{P}(A=1,B=1|C=1)=0.069$$

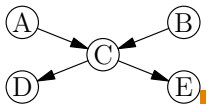
- Note $\mathbb{P}(A=1,B=1|C=1)\neq\mathbb{P}(A=1|C=1)\mathbb{P}(B=1|C=1)$
- When we observe C then A and B are no longer independent

Elaborate Cakes

- We can elaborate on our cake model
- We suppose that Dave likes cakes so if there is a cake in the coffee room there is a 80% chance that I will see him eating a cake: $\mathbb{P}(D=1|C=1)=0.8$
- Even if there are no cakes in the coffee room there is a 10% chance that Dave has bought his own cake:
  $\mathbb{P}(D=1|C=0)=0.1$
- Eli also likes cakes: there is a 60% chance that I will see her eating cakes if there are cakes in the coffee room:
  $\mathbb{P}(E=1|C=1)=0.6$
- But she never buys herself cakes $\mathbb{P}(E=1|C=0)=0$

Elaborate Graphical Model

- We can depict this situation as



- This allows us to break down the joint probability

$$\begin{aligned}\mathbb{P}(A,B,C,D,E) &= \mathbb{P}(C,D,E|A,B)\mathbb{P}(B)\mathbb{P}(A)\\ &= \mathbb{P}(D|C)\mathbb{P}(E|C)\mathbb{P}(C|A,B)\mathbb{P}(B)\mathbb{P}(A)\end{aligned}$$

- We use the conditional independence of D and E given C

Dependencies

- If we don't observe cakes then the probability of Dave and Eli eating cake are not independent

$$\mathbb{P}(D=1)=0.303, \qquad \mathbb{P}(E=1)=0.174$$
$$\mathbb{P}(D=1,E=1)=0.1392$$

so $\mathbb{P}(D,E)\neq\mathbb{P}(D)\mathbb{P}(E)$

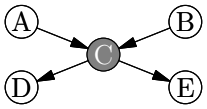
- This changes if we know there are cakes in the coffee room

$$\mathbb{P}(D=1|C=1)=0.8 \qquad \mathbb{P}(E=1|C=1)=0.6$$
$$\mathbb{P}(D=1,E=1|C=1)=0.48$$

so $\mathbb{P}(D=1,E=1|C=1)=\mathbb{P}(D=1|C=1)\mathbb{P}(E=1|C=1)$

Observations and Independence

- Making observations changes the probabilities and in some case the dependencies of random variables on each other



$A \not\perp\!\!\!\perp B$

$D \not\perp\!\!\!\perp E$

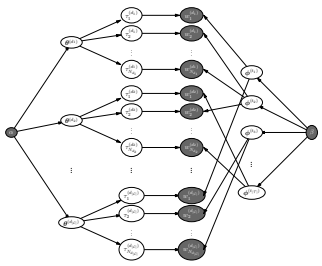
- There are rules to deduce the conditional independence from a graphical model given which variables have been observed—but these are details that you can look up if needed

Graphical Model Frameworks

- There are sophisticated frameworks for computing probabilities in Bayesian Belief Networks efficiently
- If our graph is a tree then we can evaluate probabilities efficiently
- When there are loops (so that a random variable both influences and is influenced by another random variables) then exact evaluation of expectations requires exhaustive summing over variables (which is often not tractable)
- There are various message passing algorithms designed to obtain approximations of expectations

Outline

1. Graphical Models
2. Cakes!
3. **Latent Dirichlet Allocation**



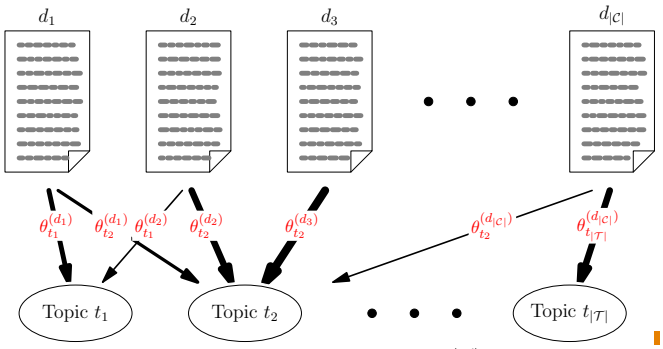
Model for Documents

- We consider a model for the words in a set of documents (we ignore word order)
- We consider a corpus $\mathcal{C}=\{d_i|i=1,2,\ldots|\mathcal{C}|\}$
- With documents consisting of words

$$d=\left(w_1^{(d)},w_2^{(d)},\ldots,w_{N_d}^{(d)}\right)$$

- We assume that there is a set of topics $\mathcal{T}=\{t_1,t_2,\ldots,t_{|\mathcal{T}|}\}$
- We associate a probability, $\theta_t^{(d)}$, that a word in document d relates to a topic t

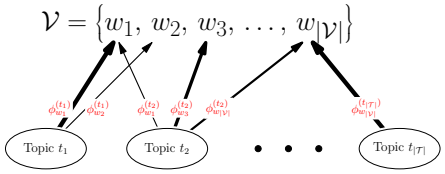
Documents and Topic



$$\boldsymbol{\theta}^{(d)}=(\theta_t^{(d)}|t\in\mathcal{T}) \qquad \theta_t^{(d)}\geq 0 \qquad \sum_{t=1}^{|\mathcal{T}|}\theta_t^{(d)}=1$$

Words and Topic

- We associate a probability $\phi_w^{(t)}$ that a word, w, is related to a topic t

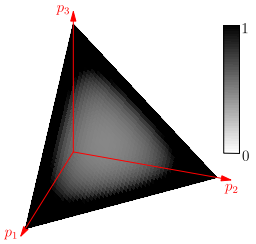


$$\boldsymbol{\phi}^{(t)}=(\phi_w^{(t)}|w\in\mathcal{V})$$

Dirichlet Allocation

- Most documents are predominantly about a few topics and most topic have a small number of words associated to them
- We can generate sparse vectors $\boldsymbol{\theta}^{(d)}$ and $\boldsymbol{\phi}^{(t)}$ from a Dirichlet distribution with small parameters $\boldsymbol{\alpha}$

$$\mathrm{Dir}(\boldsymbol{p}|\boldsymbol{\alpha}) = \Gamma\left(\sum_i \alpha_i\right) \prod_{i=1}^{n} \frac{p_i^{\alpha_i - 1}}{\Gamma(\alpha_i)}$$



$$\boldsymbol{\theta}^{(d)} \sim \mathrm{Dir}(\alpha \mathbf{1})$$
$$\boldsymbol{\phi}^{(t)} \sim \mathrm{Dir}(\beta \mathbf{1})$$

- $\sum_i p_i = 1$

Generating Document

- To generate a document we choose a topic for each word and a word for each topic

$$\forall d \in \mathcal{C} \quad \boldsymbol{\theta}^{(d)} \sim \mathrm{Dir}(\alpha \mathbf{1})$$
$$\forall t \in \mathcal{T} \quad \boldsymbol{\phi}^{(t)} \sim \mathrm{Dir}(\beta \mathbf{1})$$
$$\forall d \in \mathcal{C} \ \wedge \ \forall i \in \{1,2,\ldots,N_d\} \quad \tau_i^{(d)} \sim \mathrm{Cat}(\boldsymbol{\theta}^{(d)}), w_i^{(d)} \sim \mathrm{Cat}(\boldsymbol{\phi}^{(\tau_i^{(d)})})$$

- Where $\mathrm{Cat}(i|\boldsymbol{p}) = p_i$ is the categorical distribution (we choose one of a number of options)
- This model is known as **Latent Dirichlet Allocation**

LDA Graphical Model (version 1)

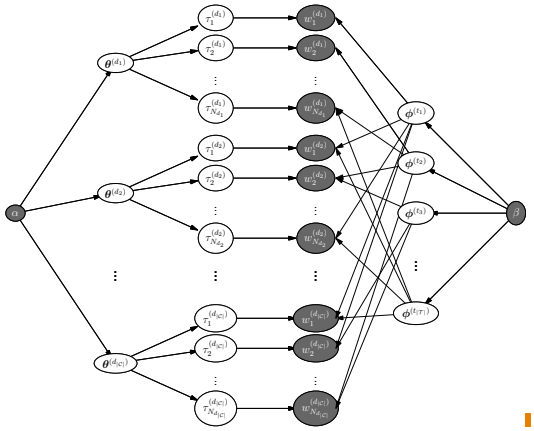
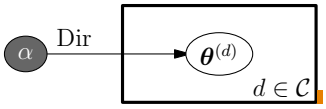


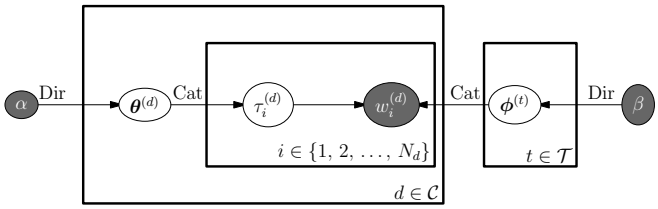
Plate Diagrams

- Drawing every random variable is tedious (and not really possible)
- A short-hand is to draw a box (plate) meaning repeat



- That is we generate vectors $\boldsymbol{\theta}^d$ from a Dirchelet distribution $\mathrm{Dir}(\boldsymbol{\theta}|\alpha\mathbf{1})$ for all documents in corpus $\mathcal{C}$

LDA Graphical Model (version 2)



- This is a lot more compact
- Personally, I find it hard to read, but you get used to it

Probabilistic Model

- The graphical Model is shorthand for the variables

$$\boldsymbol{W} = (\boldsymbol{w}^{(d)}|d \in \mathcal{C}) \quad \text{with} \quad \boldsymbol{w}^{(d)} = (w_1^{(d)}, w_2^{(d)}, \ldots, w_{N_d}^{(d)}), \quad \text{and} \quad w_i^{(d)} \in \mathcal{V}$$
$$\boldsymbol{T} = (\tau_i^{(d)}|d \in \mathcal{C} \ \wedge \ i \in \{1,2,\ldots,N_d\}) \quad \text{with} \quad \tau_i^{(d)} \in \mathcal{T}$$
$$\boldsymbol{\Theta} = (\boldsymbol{\theta}^{(d)}|d \in \mathcal{C}) \quad \text{with} \quad \boldsymbol{\theta}^{(d)} = (\theta_t^{(d)}|t \in \mathcal{T}) \in \Lambda^{|\mathcal{T}|}$$
$$\boldsymbol{\Phi} = (\boldsymbol{\phi}^{(t)}|t \in \mathcal{T}) \quad \text{with} \quad \boldsymbol{\phi}^{(t)} = (\phi_w^{(t)}|w \in \mathcal{V}) \in \Lambda^{|\mathcal{V}|}$$

- Distributed according to

$$\mathbb{P}(\boldsymbol{W},\boldsymbol{T},\boldsymbol{\Theta},\boldsymbol{\Phi}|\alpha,\beta) = \left(\prod_{t\in\mathcal{T}} \mathrm{Dir}\left(\boldsymbol{\phi}^{(t)}|\beta\mathbf{1}\right)\right) \left(\prod_{d\in\mathcal{C}} \mathrm{Dir}\left(\boldsymbol{\theta}^{(d)}|\alpha\mathbf{1}\right) \prod_{i=1}^{N_d} \mathrm{Cat}\left(\tau_i^{(d)}|\boldsymbol{\theta}^{(d)}\right) \mathrm{Cat}\left(w_i^{(d)}|\boldsymbol{\phi}^{(\tau_i^{(d)})}\right)\right)$$

Finding Topics

- We are given the set of words $\boldsymbol{W}$ and don't really care about τ_i^d the topic associated with word i in document d
- But we are interested in the words associated with each topic $\boldsymbol{\phi}^{(t_i)}$
- And the topics associated with each document $\boldsymbol{\theta}^{(d)}$
- To compute them we need to sample the probability distribution
- One way to do this is using Monte Carlo methods (see next lecture)

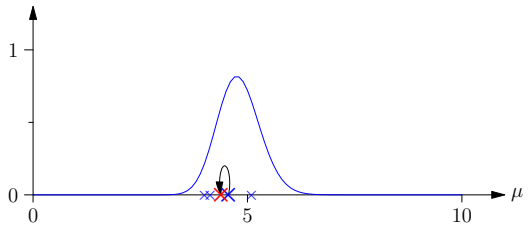
Summary

- Building probabilistic models is an intricate process
- Graphical models provide a representation showing the causal relationship between random variables
- This allows us to break down the joint probability of all the variables into conditional probabilities
- This is useful for building the model, but also can speed up evaluating expectations
- Making observations changes the probabilities of random variables
- It is possible to generate very rich models such as Latent Dirichlet Allocation (LDA)

Advanced Machine Learning

MCMC

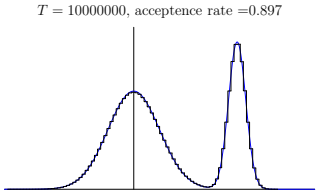
$\mathcal{D} = \{4, 4, 6, 4, 2, 2, 5, 9, 5, 4, 3, 2, 5, 4, 4, 11, 6, 2, 3, 11\}$



Monte Carlo methods, MCMC, Variational Methods

Outline

1. **Sampling**
2. Random Number Generation
3. MCMC



Bayesian Inference Gets Hard

- We saw that in some cases if we had a simple likelihood (normal, binomial, Poisson, multinomial) you can choose a conjugate prior (gamma-normal/Wishart, beta, gamma, Dirchlet) so that the posterior has the same form as the prior
- Very often we are working with more complex models where no conjugate prior exists
- The posterior is not described by a known distribution
- We have to work a lot harder—particularly with multivariate distributions

Bayesian Inference

- Recall our problem is that we are given some data $\mathcal{D}$
- Our posterior is given by

$$\mathbb{P}(\boldsymbol{\theta}|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})}{\mathbb{P}(\mathcal{D})} \quad \text{or} \quad f(\boldsymbol{\theta}|\mathcal{D}) = \frac{f(\mathcal{D}|\boldsymbol{\theta})\, f(\boldsymbol{\theta})}{f(\mathcal{D})}$$

- Where $\boldsymbol{\theta}$ are the parameters we are trying to infer
- But our likelihood (and/or prior) might be quite complicated
- Typically we don't have a closed form representation for our posterior distibution

Histograms, Samples and Means

- We could represent our posterior as a histogram, although for multivariate distributions (i.e. when we are modelling more than one variable) a histogram can be unwieldy
- A sample from the posterior distribution is often sufficient e.g. in our topic models (LDA) a typical set of topics is what we are after
- However, when samples vary a lot, often the most useful quantities are expectation, e.g.

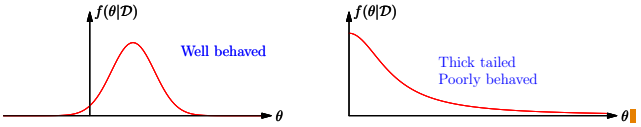
$$\mathbb{E}[\boldsymbol{\Theta}] \qquad \mathbb{E}\left[\Theta_i^2\right] - \mathbb{E}[\Theta_i]^2$$
$$\mathbb{E}[\Theta_i\Theta_j] - \mathbb{E}[\Theta_i]\mathbb{E}[\Theta_j] \qquad \mathbb{E}\left[\boldsymbol{\Theta}\boldsymbol{\Theta}^{\mathsf{T}}\right] - \mathbb{E}[\boldsymbol{\Theta}]\mathbb{E}[\boldsymbol{\Theta}]^{\mathsf{T}}$$

Sample Estimation

- If we can draw independent **deviates** (aka **variates**), $\boldsymbol{\Theta}_i$, from our posterior distribution then we can obtain an estimate of our expectation

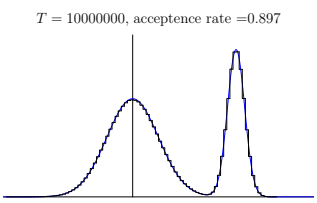
$$\mathbb{E}[g(\boldsymbol{\Theta})] \approx \frac{1}{n}\sum_{i=1}^{n} g(\boldsymbol{\Theta}_i)$$

- Provided our posterior distribution is well behaved the relative error in our estimate will drop off as $1/\sqrt{n}$



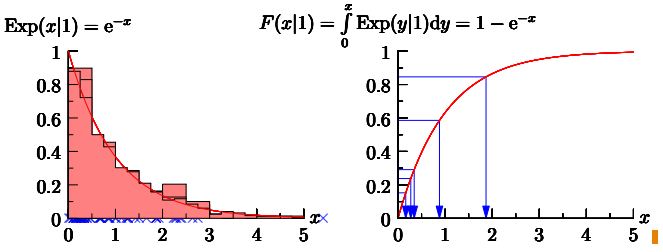
Outline

1. Sampling
2. **Random Number Generation**
3. MCMC



Drawing Random Samples

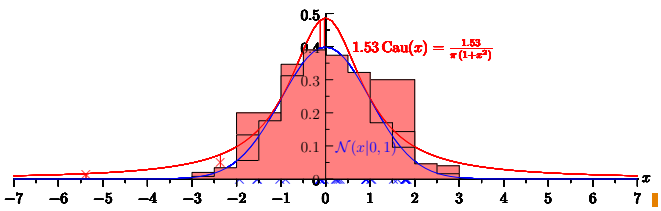
- Drawing (pseudo) random variables from a distribution is known as **Monte Carlo**
- For some very simple distributions we can use the **transformation methods** to transform a uniform distribution



Rejection Method

- The transformation method only works when we can easily compute the inverse *cumulative distribution function* (CDF)
- A more general technique is the **rejection method** where we generate deviates from $g_Y(y)$ such that $cg_Y(x) \geq f_X(x)$
- To draw deviates from $f_X(x)$ we draw a deviate $Y \sim g_Y$ and then accept the deviate with probability $f_X(Y)/(cg_Y(Y))$
- The expected rejection rate is $c - 1$
- Need to choose a good distribution $g_Y(y)$

Drawing Normal Deviates

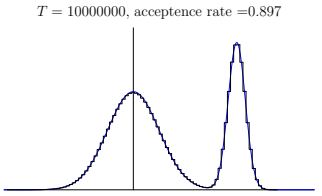


Problems with Rejection

- The rejection method is very general and often the method of choice (although for normal deviates there is a clever transformation method which is faster)
- However, for complicated probability distributions it can be difficult to find a good proposal distribution $g_Y(y)$
- This is particular true for multivariate distributions
- If the proposal distribution is poor c might be very high and the number of rejections is stupidly high

Outline

1. Sampling
2. Random Number Generation
3. MCMC



Detailed Balance

- Suppose we have a set of states $\mathcal{S}$ and want to draw sample from a probability distribution $\boldsymbol{\pi} = (\pi_i | i \in \mathcal{S})$
- We invent a dynamical system with a transition probability M_{ij} from state j to state i such that

$$M_{ij}\pi_j = M_{ji}\pi_i$$

- This is known as **detailed balance**
- Summing both sides over j

$$\sum_j M_{ij}\pi_j = \sum_j M_{ji}\pi_i = \pi_i \qquad \mathbf{M}\boldsymbol{\pi} = \boldsymbol{\pi}$$

Convergence of MCMC

- Suppose we start from a state $\boldsymbol{x}(0) = \sum_i c_i \boldsymbol{v}^{(i)}$ where the $\boldsymbol{v}^{(i)}$'s are an eigenvectors of the transition matrix $\mathbf{M}$ with eigenvalues λ_i
- It I apply $\mathbf{M}$ many times then

$$\boldsymbol{x}(t) = \mathbf{M}^t\boldsymbol{x}(0) = \mathbf{M}^t\sum_i c_i\boldsymbol{v}^{(i)} = \sum_i \lambda_i^t c_i \boldsymbol{v}^{(i)}$$

- And $\lim_{t\to\infty} \boldsymbol{x}(t) = \boldsymbol{v}^*$ where $\boldsymbol{v}^*$ is the eigenvector with the maximum eigenvalue
- Now $\|\mathbf{M}\boldsymbol{v}\|_1 \leq \|\mathbf{M}\|_1\|\boldsymbol{v}\|_1 = \|\boldsymbol{v}\|_1$ so the maximum eigenvalue is 1 with eigenvector $\boldsymbol{\pi}$ ($\mathbf{M}$ is known as a **stochastic matrix**)

Metropolis Algorithm

- A very easy way to achieve detailed balance is starting from state j choose a "neighbouring" state, i with equal probability
- We accept the move if either
  - $\pi_i > \pi_j$ or
  - we make the move with a probability π_i/π_j
- If $\pi_i > \pi_j$ then $M_{ij} = 1$ and $M_{ji} = \pi_j/\pi_i$. Thus

$$M_{ij}\pi_j = \pi_j \qquad M_{ji}\pi_i = \frac{\pi_j}{\pi_i}\pi_i = \pi_j$$

- Note that we require the state i to have the same number of neighbours as state j so that detailed balance is satisfied

Continuous Variables

- If we are working with continuous variables $\boldsymbol{\theta}$ then the equation for detailed balance for the transition probability $W(\boldsymbol{\theta} \to \boldsymbol{\theta}')$ is

$$W(\boldsymbol{\theta} \to \boldsymbol{\theta}')\pi(\boldsymbol{\theta}) = W(\boldsymbol{\theta}' \to \boldsymbol{\theta})\pi(\boldsymbol{\theta}')$$

- where $\pi(\boldsymbol{\theta})$ is the probability distribution we wish to sample from
- The update rule is to choose a nearby value $\boldsymbol{\theta}'$, compute $r = \pi(\boldsymbol{\theta}')/\pi(\boldsymbol{\theta})$ and accept the update with probability $\min(1, r)$
- We require that the probability of choosing $\boldsymbol{\theta}$ from $\boldsymbol{\theta}'$ is the same as the reverse

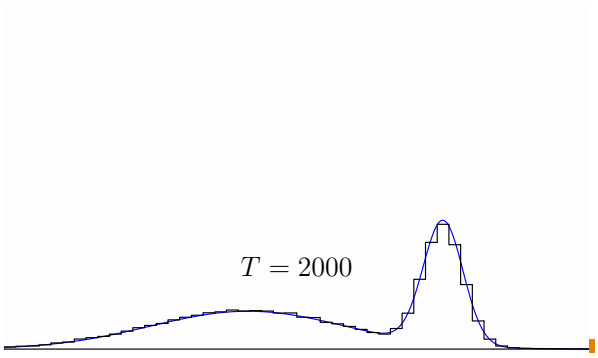
What Makes MCMC Nice

- Because we are free to choose where we move (and choose close by neighbours) $\pi(\boldsymbol{\theta}') \approx \pi(\boldsymbol{\theta})$ so that moves are not too infrequent
- Also very importantly the updates depend only on the ratio $\pi(\boldsymbol{\theta}')/\pi(\boldsymbol{\theta})$
- We only need to know our probabilities up to a multiplicative scaling factor
- For sampling from the posterior we only need to know the likelihood and prior $\mathbb{P}(\mathcal{D}|\boldsymbol{\theta})\mathbb{P}(\boldsymbol{\theta})$ (or $f(\mathcal{D}|\boldsymbol{\theta})f(\boldsymbol{\theta})$)
- We don't need to know $\mathbb{P}(\mathcal{D})$ which we generally don't know

What Makes MCMC Nasty

- It can take a long time until our states occur with the probability $\boldsymbol{\pi}$ (i.e. we have forgotten our initial state)
- We don't even know how long we have to wait
- Even when we have reached this *equilibration time* each sample is correlated with the previous sample
- To get a good approximation to the posterior expectation requires running for many times the equilibration time
- Note, if we are just finding sample averages then we can use all samples after equilibrating even if they are not independent

Burn-In



Proposals and Metropolis-Hastings

- We have some freedom in choosing a new proposal $\boldsymbol{\theta}'$ from our current position $\boldsymbol{\theta}$—a good choice can increase the acceptance rate making the MCMC more efficient
- We define the proposal distribution $p(\boldsymbol{\theta}'|\boldsymbol{\theta})$
- For the standard Metropolis algorithm to work we require $p(\boldsymbol{\theta}'|\boldsymbol{\theta}) = p(\boldsymbol{\theta}|\boldsymbol{\theta}')$
- In some cases (e.g when $\theta_i \geq 0$) this can be hard to achieve
- We can modify our update rule to accept a move with probability

$$\min\left(1, \frac{p(\boldsymbol{\theta}|\boldsymbol{\theta}')f(\mathcal{D}|\boldsymbol{\theta}')f(\boldsymbol{\theta}')}{p(\boldsymbol{\theta}'|\boldsymbol{\theta})f(\mathcal{D}|\boldsymbol{\theta})f(\boldsymbol{\theta})}\right)$$

Traffic Rate

- Consider monitoring the flow of traffic where we have data

$$\mathcal{D} = (N_1, N_2, \ldots, N_n)$$

  where N_i is the number of car that past on day i
- We assume $N_i \sim \text{Poi}(\mu)$ and want to infer μ
- The Poisson distribution has a beta conjugate prior
- We don't have any prior knowledge on μ so we use a non-informative prior $\text{Gam}(\mu|0,0) = 1/\mu$
- Note that we can solve this problem exactly—however, lets compare with MCMC

Proposal Distribution

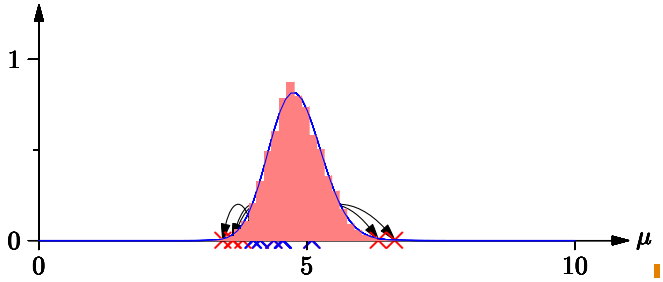
- If we can choose our proposal distribution $p(\mu'|\mu)$ to be close to the posterior distribution then our acceptance rate would be close to 1
- We choose $p(\mu'|\mu) = \text{Gam}(\mu'|\mu,\mu^2)$ which has $\mathbb{E}[\mu'] = \mu$ and variance 1
- We update with probability $\min(1,r)$ where

$$r = \frac{\text{Gam}(\mu|\mu'^2,\mu')\frac{1}{\mu'}\prod_{i=1}^n \text{Poi}(N_i|\mu')}{\text{Gam}(\mu'|\mu^2,\mu)\frac{1}{\mu}\prod_{i=1}^n \text{Poi}(N_i|\mu)}$$

$$= \frac{\mu\text{Gam}(\mu|\mu'^2,\mu')}{\mu'\text{Gam}(\mu'|\mu^2,\mu)} \mathrm{e}^{-n(\mu'-\mu)+\sum_{i=1}^n N_i \log\left(\frac{\mu'}{\mu}\right)}$$

MCMC in Practice

$$\mathcal{D} = \{4,4,6,4,2,2,5,9,5,4,3,2,5,4,4,11,6,2,3,11\}$$



MCMC Details

- To compute correct histograms you need to count samples where no move is made multiple times
- On modern computers its quite quick to compute millions of samples
- The code is not very difficult to write (although care is need to get everything correct)
- This can be used on complicated problems such as topic models (LDA) with thousands of parameters
- The accuracy of MCMC is slow if it takes a long time to sample the posterior distribution

The MCMC Industry

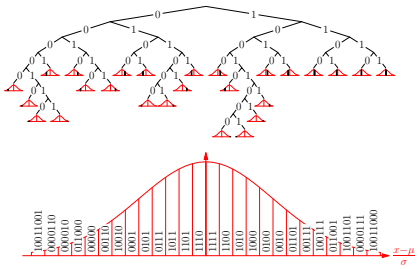
- MCMC provides a means to accurately sample from very complex models
- There have been many advanced techniques developed to improve MCMC performance
- E.g. hybrid MCMC simulates a dynamics to find good proposals with similar probability far from the starting point
- Often it seems that MCMC is complicated because there are so many optimisations, but often simple implementations are sufficient

Conclusions

- As soon as we use complex models we are no longer able to compute the posterior in closed form
- Monte Carlo techniques and particularly MCMC are a very general method for computing samples from the posterior
- These techniques have been highly developed, but very frequently even simple implementations are sufficient to do good inference
- Variational methods provide an approximate closed form solution to problems with complex likelihoods
- Variational methods are mathematically challenging, but are potentially far faster to compute than MCMC

Advanced Machine Learning

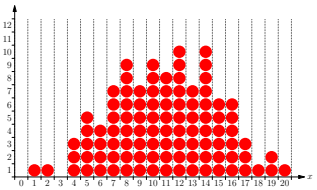
Entropy



Entropy, Coding, Maximum Entropy

Outline

1. Measuring Uncertainty
2. Code Length
3. Maximum Entropy



Measuring Uncertainty

- What is more uncertain tossing a coin three times or throwing a dice
- The answer depends on whether you care about the order of the coin tosses
- But, how do we answer such a question?
- Let X be a random variable denoting the possible outcomes
- Interestingly, Shannon entropy give a precise answer

$$H_X = -\sum_{x\in\mathcal{X}} \mathbb{P}(X=x)\log_2(\mathbb{P}(X=x))$$

Let's Calculate

- For an honest dice $D \in \{1,2,3,4,5,6\}$ and $\mathbb{P}(D=i)=1/6$ so

$$H_D = -\sum_{i=1}^{6}\frac{1}{6}\log_2\left(\frac{1}{6}\right) = -\log_2\left(\frac{1}{6}\right) = \log_2(6) \approx 2.584\text{bits}$$

- For an honest coin where we care about the order so $C \in \{000,001,\ldots,111\}$ the $\mathbb{P}(C=i)=\frac{1}{8}$ and

$$H_C = -\sum_{i=0}^{7}\frac{1}{8}\log_2\left(\frac{1}{8}\right) = -\log_2\left(\frac{1}{8}\right) = \log_2(8) = 3\text{bits}$$

- This clearly makes sense: there are more possible outcomes; all equally likely

Unordered Coin Toss

- What if we don't care about the order of the out-come then $\mathbb{P}(HHH)=\mathbb{P}(TTT)=1/8$, $\mathbb{P}(HHT)=\mathbb{P}(HTT)=3/8$ so

$$H_U = -\frac{1}{4}\log_2\left(\frac{1}{8}\right) - \frac{3}{4}\log_2\left(\frac{3}{8}\right) \approx 1.811\text{bits}$$

- This seems reasonable, although it is not obvious how you would determine this without using entropy
- But why Shannon entropy?

Additive Entropy

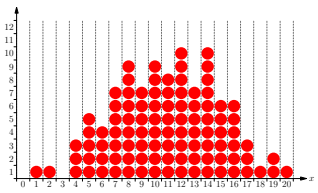
- If H_X and H_Y is the uncertainty of two independent random variable X and Y, what is the uncertainty of the combined event (X,Y)?

$$\begin{aligned}
H_{(X,Y)} &= -\sum_{X,Y}\mathbb{P}(X,Y)\log_2(\mathbb{P}(X,Y)) \\
&= -\sum_{X,Y}\mathbb{P}(X)\mathbb{P}(Y)\log_2(\mathbb{P}(X)\mathbb{P}(Y)) \\
&= -\sum_{X,Y}\mathbb{P}(X)\mathbb{P}(Y)\left(\log_2(\mathbb{P}(X))+\log_2(\mathbb{P}(Y))\right) \\
&= -\sum_{X}\mathbb{P}(X)\log_2(\mathbb{P}(X)) - \sum_{Y}\mathbb{P}(Y)\log_2(Y) = H_X + H_Y
\end{aligned}$$

- Shannon's entropy is one of the few functions that satisfy this condition

Outline

1. Measuring Uncertainty
2. Code Length
3. Maximum Entropy



Why Measure Entropy in Bits

- Suppose we had to communicate a message with 2^n equally likely outputs (e.g. the result of n-coin tosses)
- We can do this with a binary string with n bits (011..0)
- If there were 5 possible outcomes I could do this with 3 bits, but waste 3/8 of the message
- However if we have a batch of 3 independent messages each with 5 outcomes then there are 125 possible outcomes. We could communicate this with 8 bits. This would waste 3/128 of the message
- By batching together enough messages with N outcomes then we asymptotically need just $\log_2(N)$ bits

Different Probabilities

- We "showed" that if we had N events, X_i, each with probability, $\mathbb{P}(X_i) = 1/N$, we can code the outcomes with a message of length $-\log_2(\mathbb{P}(X_i)) = \log_2(N)$
- With a shorter message we would not be able to distinguish all possible outcomes from the message
- What happens if some of outcomes occur with a different probability

X_i:	1	2	3	4	5	6
$p(X_i)$:	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{4}$
Code:	000	001	010	011	10	11
$L = -\log_2(p(X_i))$:	3	3	3	3	2	2

Shannon's Entropy

- If the probabilities are not equal to 2^{-n} we can still find a code with a length very close to $-\log_2(\mathbb{P}(X))$ per message by transmitting a large number of messages
- The length of the message measures the amount of **surprise** on receiving the message
- Shannon's entropy is the expected length of the message to communicate a random variable X

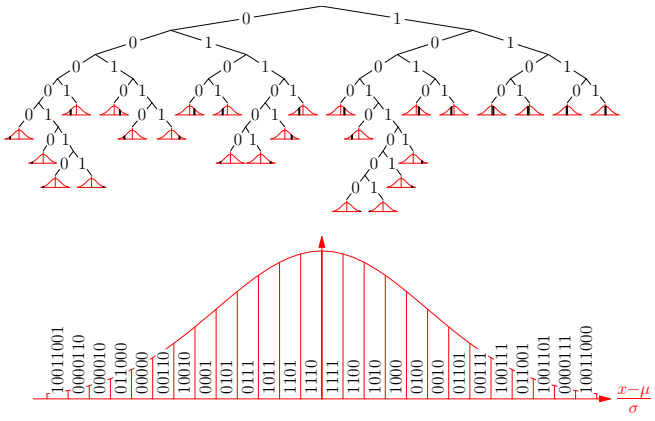
$$H_X = \mathbb{E}_X[-\log_2(\mathbb{P}(X))] = -\sum_{x\in\mathcal{X}} \mathbb{P}(X=x)\log_2(\mathbb{P}(X=x))$$

- The expected length is a measure of the uncertainty (how much information on average we need to convey the outcome)

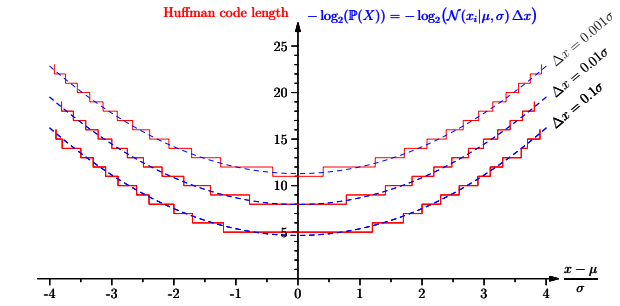
Real Codes

- Those of you with some computer science background will realise that we can't actually use different length strings in a code without paying some price
- We won't know where a code word ends so we can't decode the message
- An optimal solution is to use Huffman encoding where we associate the leaf of a tree with each code word
- Using the tree we can decode any message constructed using the tree
- There is a greedy algorithm for constructing the optimal tree

Coding Normals



Coding Normals to Accuracy Δx



bits and nats

- We have measured entropy in **bits** using

$$H_X = -\sum_{x\in\mathcal{X}} \mathbb{P}(X=x)\log_2\big(\mathbb{P}(X=x)\big)$$

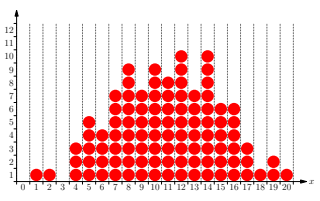
- Sometimes it is easier to use natural logarithms

$$H_X = -\sum_{x\in\mathcal{X}} \mathbb{P}(X=x)\ln\big(\mathbb{P}(X=x)\big)$$

- In this case the entropy is measured in **nats** with 1 nat equal to $\log_2(\mathrm{e})$ bits
- This is often easier when we want to do calculus on entropy

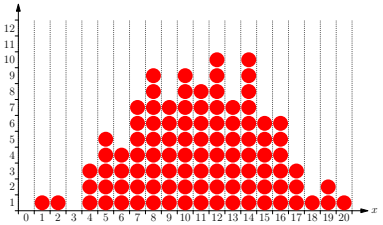
Outline

1. Measuring Uncertainty
2. Code Length
3. **Maximum Entropy**



Number of States

- Suppose I have N balls I them put in K boxes with coordinates x_i such that the mean is μ and variance is σ^2



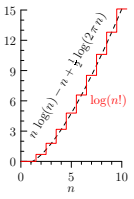
$$\mathbb{P}(\boldsymbol{n}) \propto \frac{N!}{n_1!n_2!\cdots n_K!}\left[\!\!\left[\sum_i \frac{n_i}{N}x_i = \mu\right]\!\!\right]\left[\!\!\left[\sum_i \frac{n_i}{N}(x_i-\mu)^2 = \sigma^2\right]\!\!\right]$$

Stirling's Approximation

- We can approximate the factorial $n!$ using **Stirling's approximation**

$$n! \approx \sqrt{2\pi n} n^n \mathrm{e}^{-n}$$

$$\log(n!) = n\log(n) - n + \frac{1}{2}\log(2\pi n)$$



- Using this in our formula for $\mathbb{P}(\boldsymbol{n})$ we have

$$\mathbb{P}(\boldsymbol{n}) \approx C\mathrm{e}^{-N\sum_i \frac{n_i}{N}\log\left(\frac{n_i}{N}\right)} \prod_{l=1}^{3} \left[\!\left[\sum_i \frac{n_i}{N} f_l(x_i) = v_l \right]\!\right]$$

where $(f_1(x_i), v_l) = \{(1,1), (x_i, \mu), ((x_i-\mu)^2, \sigma^2)\}$

Number of States and Entropy

- Let $p(x_i) = n_i/N$ be the proportion of balls in bin i then

$$\mathbb{P}(\boldsymbol{n}) \approx C\mathrm{e}^{NH_X} \prod_{l=1}^{3} \left[\!\left[\sum_i \frac{n_i}{N} f_l(x_i) = v_l \right]\!\right]$$

where

$$H_X = -\sum_i p(x_i)\log(p(x_i))$$

- That is, the "entropy" can be seen as a measure of the logarithm of the number of configurations
- When the number of balls, $N \to \infty$ the overwhelmingly likely configurations is the one that maximises the entropy subject to the observed mean and variance

Maximum Entropy Method

- When we are trying to infer a distribution given some observations then we can maximise the entropy subject to constraints—the entropy acts as a prior
- This is known as the **maximum entropy method**
- We can rationalise this as this is by far the most likely set of configurations consistent with the observations
- Alternatively we can see this as maximising our uncertainty given what we know—being as unbiased as possible
- It only gives a good approximation if all possibilities are equally likely

Knowing the Mean and Variance

- Consider a continuous random variable, X, with a known mean and second moment

$$\mathbb{E}[X] = \mu, \qquad \mathbb{E}[X^2] = \mu_2 = \mu^2 + \sigma^2$$

- To maximise the entropy subject to constraints consider

$$\mathcal{L}(f) = -\int f_X(x)\log(f_X(x))\,\mathrm{d}x + \lambda_0\left(\int f_X(x)\,\mathrm{d}x - 1\right) + \lambda_1\left(\int f_X(x)x\,\mathrm{d}x - \mu\right) + \lambda_2\left(\int f_X(x)x^2\,\mathrm{d}x - \mu_2\right)$$

- Thus

$$\frac{\delta\mathcal{L}(f)}{\delta f_X(x)} = -\log(f_X(x)) - 1 + \lambda_0 + \lambda_1 x + \lambda_2 x^2 = 0$$

- Or

$$f_X(x) = \mathrm{e}^{-1+\lambda_0+\lambda_1 x+\lambda_2 x^2}$$

Normal Distribution

- We have three constraints

$$\int \mathrm{e}^{-1+\lambda_0+\lambda_1 x+\lambda_2 x^2}\mathrm{d}x = 1$$

$$\int \mathrm{e}^{-1+\lambda_0+\lambda_1 x+\lambda_2 x^2} x\,\mathrm{d}x = \mu$$

$$\int \mathrm{e}^{-1+\lambda_0+\lambda_1 x+\lambda_2 x^2} x^2\mathrm{d}x = \mu_2 = \mu^2 + \sigma^2$$

- Solving for λ_0, λ_1 and λ_2 then

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-(x-\mu)^2/(2\sigma^2)}$$

- That is, the normal distribution is the maximum entropy distribution given we known the mean and variance

Using Maximum Entropy

- Maximum entropy is often used to infer distributions
- It can be very effective, but it might not work well if there are other constraints that we have not included
- The place that they work superbly well is in statistical physics
- The whole of statistical physics is about inferring distributions making observations of volume, pressure, etc.
- Temperature appears rather strangely as a Lagrange multiplier

Historic Entropy

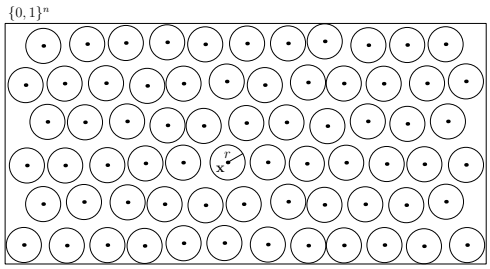
- Historically entropy was first introduced in statistical physics by Rudolf Clausius in 1865 (although Macquorn Rankine discussed it in 1850)
- Its interpretation as the number of states was introduced by Ludwig Boltzmann
- The person who got it all right was Josiah Willard Gibbs (and James Clerk Maxwell)
- Claude Shannon invented information theory base on entropy around 1948 (more on that in the next lecture)
- Ed Jaynes was the first to understand that statistical physics can be seen as an inference problem

Conclusion

- Entropy provides a measure of the disorder or uncertainty in a system
- It forms the basis of information theory which we will look at in the next lecture
- $-\log(\mathbb{P}(X = x))$ can be seen as the minimum length of a message to communication x
- This will be used as the basis of the minimum description length formalism also discussed in the next lecture
- Entropy can be used as a prior, which we often maximise subject to constraints to obtain an unbiased estimate

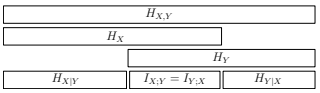
Advanced Machine Learning

Information Theory



Information, KL-divergence, Minimum Description Length

Outline



Communicating Via a Noisy Channel

- Information theory considers communicating down a (noisy) channel

$$X \sim \mathbb{P}(X) \xrightarrow{\text{noisy channel}} Y \sim \mathbb{P}(Y \mid X)$$

- We send a message X (with probability $\mathbb{P}(X)$) and receive a message Y with probability $\mathbb{P}(Y \mid X)$
- The uncertainty of the message sent, given we received a message y is

$$H_{X|Y=y} = -\sum_{x\in\mathcal{X}} \mathbb{P}(X=x \mid Y=y)\log(\mathbb{P}(X=x \mid Y=y))$$

- The expected uncertainty in the message sent is

$$H_{X|Y} = \sum_{y\in\mathcal{Y}} \mathbb{P}(Y=y)\, H_{X|Y=y} = -\sum_{x,y} \mathbb{P}(X=x, Y=y)\log(\mathbb{P}(X=x \mid Y=y))$$

Joint Entropy

- We can define the **joint entropy**

$$H_{X,Y} = -\sum_{x,y} P_{X,Y}(x,y)\log(P_{X,Y}(x,y))$$

- If the message we receive is independent of the message that is sent then $H_{X,Y} = H_X + H_Y$ (we saw this in the last lecture)
- $H_{X,Y} \neq H_X + H_Y$ if X and Y are correlated
- Since $\mathbb{P}(X,Y) = \mathbb{P}(Y|X)\mathbb{P}(X) = \mathbb{P}(X|Y)\mathbb{P}(Y)$ if follows

$$H_{X,Y} = H_X + H_{Y|X} = H_Y + H_{X|Y}$$

- Or $H_X - H_{X|Y} = H_Y - H_{Y|X}$

Mutual Information

- The amount of uncertainty about the message being sent, X, before receiving the message is $H_X = -\mathbb{E}_X[\log\mathbb{P}(X)]$
- Shannon define the *mutual information* to be the expected loss in uncertainty when we receive a message

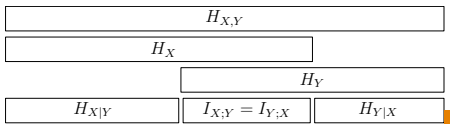
$$I_{X;Y} = H_X - H_{X|Y}$$

- Since $H_X - H_{X|Y} = H_Y - H_{Y|X}$ it follows

$$I_{X;Y} = I_{Y;X}$$

Channel Capacity

- We can summarise these relationships diagrammatically



- Shannon defined the *capacity* of a noisy channel as

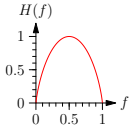
$$C = \max_{\mathbb{P}(X)} I_{X;Y}$$

- That is, you choose the probability distribution of the message to maximise the information gain

Independent Noise

- The simplest model of a noisy channel is a binary channel where each symbol is corrupted independently with a probability f

$$\mathbb{P}(X=1|Y=0) = \mathbb{P}(X=0|Y=1) = f$$



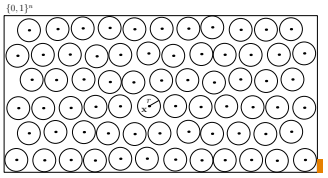
- An elementary calculations shows that

$$H_{X_i|Y_i} = -(1-f)\log(1-f) - f\log(f) = H(f)$$

- For a message of length n, $H_{X|Y} = nH(f)$

Error Correcting Codes

- To reduce the chance of misinterpreting a message we need to build an error correcting code
- We can do this dividing the space of binary messages into a set of Hamming balls



- A Hamming ball $B(\boldsymbol{x},r)$ is the set of strings that differ from n-dimensional binary string, $\boldsymbol{x}$, by at most r digits

Volume of Coding Space

- The expected number of errors in a string of length n given an error rate of f is nf
- For sufficiently large n we would expect all errors are smaller than $(f+\epsilon)n$ (for $\epsilon > 0$)
- If we make the radius of the Hamming ball $r = (f+\epsilon)n$ ($\epsilon > 0$) then we would expect no error for sufficiently large n
- An upper bound on the number of code words we can send in a string of length n is

$$\frac{2^n}{|B(\boldsymbol{x}_i, r)|} = c\sqrt{n}2^{I_{X;Y}}$$

Lower Bounds

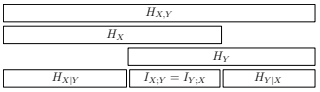
- Shannon also showed that choosing $2^{I_{X;Y}}$ random strings of length n the Hamming distance beween balls would be at least f with high probability
- This means that we can send information at rate of $I_{X;Y}$
- The maximum rate is given by the channel capacity $\max I_{X;Y}$
- If $f = 0.1$ then $C = I_{X;Y} = 0.469\text{bits}$ so we need codes of just over twice as long to communicate accurately over a noisy channel with a 10% corruption rate
- Unfortunately, we can't efficiently decode random code positions, so although we know Shannon's bound is achievable we don't have practical codes that do this

Using Mutual Information

- Mutual information is used quite often in machine learning
  - ⋆ Wikipedia mentions 14 applications
- Suppose we want to align two sets of images through some non-linear transformations
- One way of doing this is to choose the non-linear transformations that maximise the mutual information (or normalised mutual information) between the two sets of images

Outline

1. Information Theory
2. **KL-Divergence**
3. Minimum Description Length
4. Variational Auto-Encoders



KL-Divergence

- We have met the Kullback-Leibler divergence

$$\begin{aligned}\mathrm{KL}\big(p\big\|q\big) &= \mathbb{E}_{X\sim p(X)}\left[\log\left(\frac{p(X)}{q(X)}\right)\right]\\ &= -\mathbb{E}_{X\sim p(X)}[\log(q(X))] - H_X\end{aligned}$$

- Recall $-\log(q(X=x))$ is the length of code need to send a message x with a probability $q(X=x)$
- Thus $-\mathbb{E}_{X\sim p(X)}[\log(q(X))]$ is the expected length of message needed to code $X \sim p(X)$ using the optimal code for the distribution $q(X)$ that than $p(X)$
- $\mathrm{KL}\big(p\big\|q\big)$ is also known as the **relative entropy** and measures the expected extra length in coding $X \sim p(X)$ if we use the wrong distribution $q(X)$

Variational Approximation

- Recall we use MCMC in Bayesian inference because the posterior distribution is too complicated to write down in closed form
- In the variational approximation we approximate the posterior distribution by a simpler (typically factored distribution), e.g.

$$f(\boldsymbol{\theta} \mid \mathcal{D}) \approx g(\boldsymbol{\theta} \mid \boldsymbol{\phi}) = \prod_i g(\theta_i \mid \phi_i)$$

- The standard method for solving this is to maximise the **variational free energy**

$$\Phi(\boldsymbol{\phi}) = -\int g(\boldsymbol{\theta} \mid \boldsymbol{\phi}) \log\left(\frac{g(\boldsymbol{\theta} \mid \boldsymbol{\phi})}{f(\boldsymbol{\theta},\mathcal{D})}\right) \mathrm{d}\boldsymbol{\theta}$$

Evidence Lower Bound (ELBO)

- We can re-write the variational free energy as

$$\begin{aligned}\Phi(\boldsymbol{\phi}) &= -\int g(\boldsymbol{\theta} \mid \boldsymbol{\phi}) \log\left(\frac{g(\boldsymbol{\theta} \mid \boldsymbol{\phi})}{(f(\boldsymbol{\theta},\mathcal{D})/f(\mathcal{D}))\, f(\mathcal{D})}\right) \mathrm{d}\boldsymbol{\theta}\\ &= -\int g(\boldsymbol{\theta} \mid \boldsymbol{\phi}) \left(\log\left(\frac{g(\boldsymbol{\theta} \mid \boldsymbol{\phi})}{f(\boldsymbol{\theta} \mid \mathcal{D})}\right) - \log(f(\mathcal{D}))\right) \mathrm{d}\boldsymbol{\theta}\\ &= -\mathrm{KL}\big(g(\boldsymbol{\theta} \mid \boldsymbol{\phi})\big\|f(\boldsymbol{\theta} \mid \mathcal{D})\big) + \log(f(\mathcal{D}))\end{aligned}$$

- If we maximise $\Phi(\boldsymbol{\phi})$, we end up minimising the KL divergence between g and f so that $g \approx f$ and $\Phi(\boldsymbol{\phi}) \approx \log(f(\boldsymbol{D}))$
- That is, we choose the parameters of our simple factorised distribution so that it is close to the true posterior

Put Another Way

- We can rewrite the variational free energy as $\Phi(\boldsymbol{\phi}) = L_q(\boldsymbol{\phi}) + H_q(\boldsymbol{\phi})$ where

$$L_q(\boldsymbol{\phi}) = \int g(\boldsymbol{\theta} \mid \boldsymbol{\phi}) \big(\log\big(f(\mathcal{D}|\boldsymbol{\theta})\big) + \log(f(\boldsymbol{\theta}))\big) \mathrm{d}\boldsymbol{\phi}$$

  acts like an expected posterior term that is maximised when the data is well modelled (we put the probability density, $g(\boldsymbol{\theta} \mid \boldsymbol{\phi})$ where the $f(\boldsymbol{\theta},\mathcal{D})$ is large)
- The second term is an entropy

$$H_q(\boldsymbol{\phi}) = -\int g(\boldsymbol{\theta} \mid \boldsymbol{\phi}) \log\big(g(\boldsymbol{\theta} \mid \boldsymbol{\phi})\big) \mathrm{d}\boldsymbol{\phi}$$

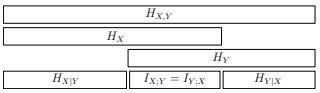
  That is, we maximise the uncertainty of the distribution $g(\boldsymbol{\theta} \mid \boldsymbol{\phi})$

Using Variational Methods

- Variational methods can be much faster than MCMC (although they tend to involve some iterations to minimise the variation free energy)
- The can produce very good approximations, although this is not guaranteed (depends on the problem)
- They can be extended (e.g. by minimising $\mathrm{KL}(g\|f)$ rather than $\mathrm{KL}(f\|g)$—this is known as *belief propagation*)
- MCMC is less elegant, but is a controlled approximation (we get better results by increasing the number of iterations)
- MCMC is slower, but on modern computers this isn't usually a problem

Outline

1. Information Theory
2. KL-Divergence
3. **Minimum Description Length**
4. Variational Auto-Encoders



Compression and Model Selection

- Outside of the Bayesian framework it is difficult to do model selection—most of ML isn't Bayesian
- When is it better to accept a more complex model for a better fit and when are we just over-fitting?
- Usually we answer this using a validation set, but this is not always possible
- One principled approach is to use the model that allows us to maximally compress the data
- If we are compressing the data then we are capturing features of the data

Alice and Bob

- Suppose Alice has data $\mathcal{D}=\{(\boldsymbol{x}_i,y_i)\mid i=1,2,\ldots,m\}$ while Bob has only the feature vectors $\{\boldsymbol{x}_i\mid i=1,2,\ldots,m\}$
- Alice wants to communicate y_i to Bob as efficiently as possible
- We suppose Alice & Bob have available a model $\hat{f}(\boldsymbol{x}|\boldsymbol{\theta})$
- Rather than sending the complete list $\{y_i\mid i=1,2,\ldots,m\}$ Alice can send Bob the parameter $\boldsymbol{\theta}$ and the errors

$$\delta_i = y_i - \hat{f}(\boldsymbol{x}_i|\boldsymbol{\theta})$$

- Assuming the δ_i's have a distribution p_δ then the cost of communicating an error to accuracy Δ is $-\log(p_\delta(\delta_i)\times\Delta)$

Description Length

- The **description length** for $\{y_i\mid i=1,2,\ldots,m\}$ is then the cost of transmitting $\boldsymbol{\theta}$ plus the cost of transmitting the errors

$$L=\sum_{k=1}^{n}\ell(\theta_k)-\sum_{i=1}^{m}\Big(\log\Big(p_\delta\Big(y_i-\hat{f}(\boldsymbol{x}_i|\boldsymbol{\theta})\Big)\Big)+\log(\Delta)\Big)$$

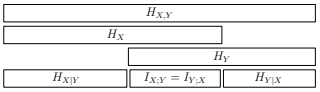
  where $\ell(\theta_k)$ is the number of bits need to communicate θ_k (we get to choose the accuracy if is worth encoding the parameters)
- To select between models we choose the model with the **minimum description length**
- Note that the accuracy Δ will lead to the same cost, $-m\log(\Delta)$, for all models so doesn't affect which model is selected

Minimum Description Length (MDL) Method

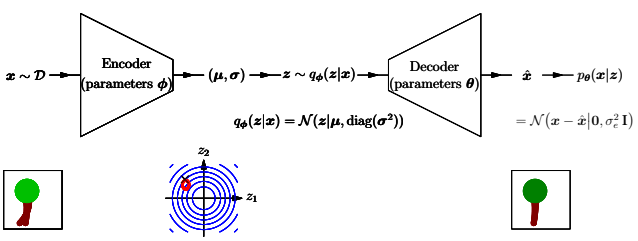
- The minimum description length method can be a powerful way of choosing between models
- Often it is the only principled method available
- It allows you to trade model accuracy against model complexity
- It can be fiddly as we need to determine the accuracy to which we should store the parameters of our model
- This isn't something we usually think about, but often we can get very good models even when we truncate the parameters to low precision

Outline

1. Information Theory
2. KL-Divergence
3. Minimum Description Length
4. **Variational Auto-Encoders**

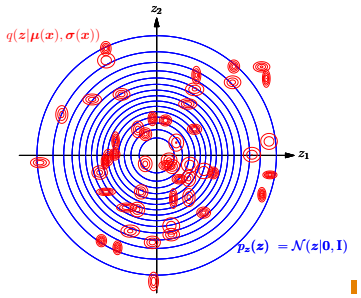


Variational Auto-Encoders VAE



$$\mathcal{L}=\mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\left[\mathrm{KL}\left(q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\middle\|\mathcal{N}(\mathbf{0},\mathbf{I})\right)-\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}(\boldsymbol{x})))\right]$$

Latent Space



$$\mathrm{KL}\big(q_{\theta}(\boldsymbol{z}|\boldsymbol{x})\big\|\mathcal{N}(\mathbf{0},\mathbf{I})\big)$$

Understanding the Loss Function

- The original paper derived the loss function as a variational approximation to maximising some posterior
- This is difficult to understand (at least, for me)
- It has a very natural explanation in terms of minimum description length
- Alice wants to communicate the images to Bob
- Alice uses the encoder to derive a (latent) code $q(\boldsymbol{z}|\boldsymbol{x})$ which she communicates to Bob
- She also communicates the errors $\boldsymbol{\delta} = \boldsymbol{x} - \bar{\boldsymbol{x}}$
- Bob uses the decoder to decode $q(\boldsymbol{z}|\boldsymbol{x})$ and $\boldsymbol{\delta}$ to repair the images

Description Length

- The loss

$$\mathcal{L} = \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\big[\mathrm{KL}\big(q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\big\|\mathcal{N}(\mathbf{0},\mathbf{I})\big) - \log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}(\boldsymbol{x})))\big]$$

  can be interpreted as

  - The cost of communicating the code $\mathrm{KL}\big(q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})\big\|\mathcal{N}(\mathbf{0},\mathbf{I})\big)$
  - Plus the cost to send the repair $\log(p_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}(\boldsymbol{x})))$

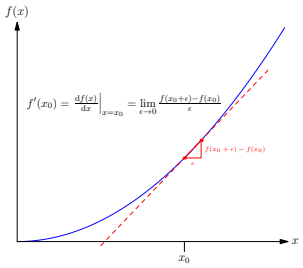
- We minimise the loss function equivalent to MDL
- What is really clever is that we can choose the accuracy of the code we send $q_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ to minimise the over-all cost

Conclusions

- Information theory has regularly been used in machine learning
- It requires some understanding and care to do it properly
- The KL-divergence (or relative entropy) is often used to make two probability distribution more alike
- The minimum description length is a powerful principle for model selection
- Variational Auto-Encoders have a very natural interpretation in terms of minimising a description length

Advanced Machine Learning

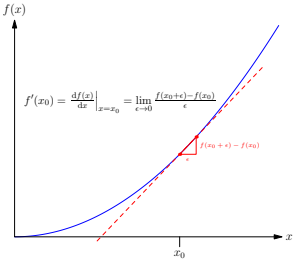
Differential Calculus



Differentiation, product and chain rules, vectors and matrices

Outline

1. **Why Calculus?**
2. Differentiation
3. Vector and Matrix Calculus



Why Calculus?

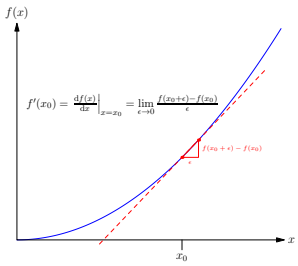
- Calculus is a fundamental tool of mathematical analysis
- In machine learning differentiation is fundamental tool in optimisation
- Integration is an essential tool in taking expectations over continuous distributions
- Both differentiation and integration crop up elsewhere
- This material will not be examined explicitly, but I assume elsewhere that you can do calculus

Back to Basics

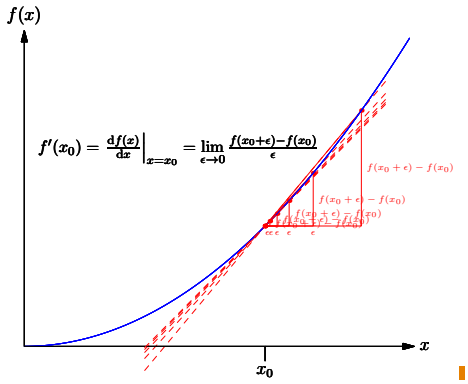
- You have all done A-level maths so should be familiar with the rules of calculus
- But, it is easy to forget the rules and sometimes we use quite sophisticated tricks
- Although the sophisticated tricks really speed up calculations, it pays to be able to understand where these tricks come from

Outline

1. Why Calculus?
2. **Differentiation**
3. Vector and Matrix Calculus



Differentiation



Polynomials

- $f(x) = x^2$

$$\frac{\mathrm{d}x^2}{\mathrm{d}x} = \lim_{\epsilon\to 0} \frac{(x+\epsilon)^2 - x^2}{\epsilon} = \lim_{\epsilon\to 0} \frac{(x^2 + 2\epsilon x + \epsilon^2) - x^2}{\epsilon}$$
$$= \lim_{\epsilon\to 0} 2x + \epsilon = 2x$$

- $(x+\epsilon)^n = (x+\epsilon)(x+\epsilon)\cdots(x+\epsilon) = x^n + n\epsilon x^{n-1} + O(\epsilon^2)$

$$\frac{\mathrm{d}x^n}{\mathrm{d}x} = \lim_{\epsilon\to 0} \frac{(x+\epsilon)^n - x^n}{\epsilon} = \lim_{\epsilon\to 0} nx^{n-1} + O(\epsilon) = nx^{n-1}$$

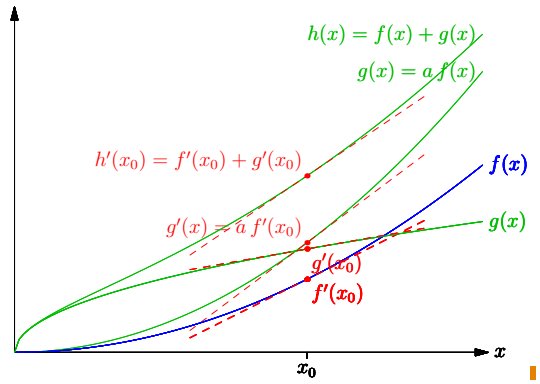
Linearity of derivatives

- Note that $f(x+\epsilon) = f(x) + \epsilon f'(x) + O(\epsilon^2)$ (from the definition of $f'(x)$)

$$\frac{\mathrm{d}(af(x)+bg(x))}{\mathrm{d}x} = \lim_{\epsilon\to 0} \frac{(af(x+\epsilon)+bg(x+\epsilon)) - (af(x)+bg(x))}{\epsilon}$$
$$= \lim_{\epsilon\to 0} \frac{a\epsilon f'(x) + b\epsilon g'(x) + O(\epsilon^2)}{\epsilon}$$
$$= af'(x) + bg'(x)$$

- **Differentiation is a linear operation!**

Linearity in Pictures



Product Rule

- Recall $f(x+\epsilon) = f(x) + \epsilon f'(x) + O(\epsilon^2)$
- If $h(x) = f(x)g(x)$

$$\begin{aligned} h'(x) &= \lim_{\epsilon\to 0} \frac{f(x+\epsilon)g(x+\epsilon) - f(x)g(x)}{\epsilon} \\ &= \lim_{\epsilon\to 0} \frac{\big(f(x) + \epsilon f'(x) + O(\epsilon^2)\big)\big(g(x) + \epsilon g'(x) + O(\epsilon^2)\big) - f(x)g(x)}{\epsilon} \\ &= \lim_{\epsilon\to 0} \frac{\epsilon(f'(x)g(x) + f(x)g'(x)) + O(\epsilon^2)}{\epsilon} = f'(x)g(x) + f(x)g'(x) \end{aligned}$$

- This is the **product rule**

Chain Rule

- Recall $f(x+\epsilon) = f(x) + \epsilon f'(x) + O(\epsilon^2)$
- Let $h(x) = f(g(x))$
- Then

$$\begin{aligned} h(x+\epsilon) &= f(g(x+\epsilon)) = f\big(g(x) + \epsilon g'(x) + O(\epsilon^2)\big) \\ &= f(g(x)) + \epsilon g'(x) f'(g(x)) + O(\epsilon^2) \end{aligned}$$

- Thus

$$h'(x) = \lim_{\epsilon\to 0} \frac{h(x+\epsilon) - h(x)}{\epsilon} = g'(x)\,f'(g(x))$$

- This is the famous **chain rule**. Together with the product rule it means you can differentiate almost everything

More on chain rules

- We can also write the chain rule as

$$\frac{\mathrm{d}f(g(x))}{\mathrm{d}x} = \frac{\mathrm{d}f(g)}{\mathrm{d}g}\frac{\mathrm{d}g(x)}{\mathrm{d}x}$$

- Sometimes this is neater or easier to remember

$$\begin{aligned} \frac{\mathrm{d}\mathrm{e}^{\cos(x^2)}}{\mathrm{d}x} &= \frac{\mathrm{d}\mathrm{e}^{\cos(x^2)}}{\mathrm{d}\cos(x^2)}\frac{\mathrm{d}\cos(x^2)}{\mathrm{d}x^2}\frac{\mathrm{d}x^2}{\mathrm{d}x} \\ &= \mathrm{e}^{\cos(x^2)}\big(-\sin(x^2)\big)\,2x \\ &= -2x\sin(x^2)\mathrm{e}^{\cos(x^2)} \end{aligned}$$

Inverse functions

- Suppose $g(y) = f^{-1}(y)$ is the inverse of $f(x)$ in the sense that $g(f(x)) = f^{-1}(f(x)) = x$
- Using the chain rule

$$\frac{\mathrm{d}g(f(x))}{\mathrm{d}x} = f'(x)g'(f(x)) = 1$$

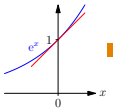
since $g(f(x)) = x$

- So $g'(f(x)) = 1/f'(x)$
- Writing $y = f(x)$ so that $x = f^{-1}(y) = g(y)$ we find $g'(y) = 1/f'(g(y))$ that is

$$\frac{\mathrm{d}g(y)}{\mathrm{d}y} = \frac{1}{f'(g(y))} \qquad \frac{\mathrm{d}f^{-1}(y)}{\mathrm{d}y} = \frac{1}{f'(f^{-1}(y))}$$

Exponentials

- Note that $a^{b+c} = a^b a^c$ (that is we multiply a together $b+c$ times)
- Now $\mathrm{e}^\epsilon \approx (1+\epsilon)$



- But $\mathrm{e}^{x+\epsilon} = \mathrm{e}^x\mathrm{e}^\epsilon = \mathrm{e}^x(1+\epsilon+O(\epsilon^2)) = \mathrm{e}^x + \epsilon\mathrm{e}^x + O(\epsilon^2)$

$$\frac{\mathrm{d}\mathrm{e}^x}{\mathrm{d}x} = \lim_{\epsilon\to 0}\frac{\mathrm{e}^{x+\epsilon} - \mathrm{e}^x}{\epsilon} = \lim_{\epsilon\to 0}\frac{\epsilon\mathrm{e}^x + O(\epsilon^2)}{\epsilon} = \mathrm{e}^x$$

Functions of Exponentials

- What about $f(x) = \mathrm{e}^{cx}$

$$\frac{\mathrm{d}\mathrm{e}^{cx}}{\mathrm{d}x} = \frac{\mathrm{d}\mathrm{e}^{cx}}{\mathrm{d}cx}\frac{\mathrm{d}cx}{\mathrm{d}x} = c\mathrm{e}^{cx}$$

- More generally using the chain rule

$$\frac{\mathrm{d}\mathrm{e}^{g(x)}}{\mathrm{d}x} = g'(x)\mathrm{e}^{g(x)}$$

- Also $a^{bc} = (a^b)^c$ (that is we multiply a together $b \times c$ times)

$$\frac{\mathrm{d}a^x}{\mathrm{d}x} = \frac{\mathrm{d}(\mathrm{e}^{\ln(a)})^x}{\mathrm{d}x} = \frac{\mathrm{d}\mathrm{e}^{\ln(a)x}}{\mathrm{d}x} = \ln(a)\mathrm{e}^{\ln(a)x} = \ln(a)a^x$$

Natural Logarithms

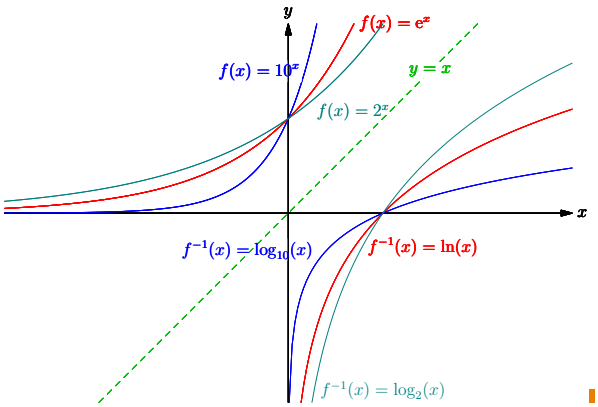
- The natural logarithm is defined as the inverse of e^x

$$\ln(\mathrm{e}^x) = x \qquad \mathrm{e}^{\ln(y)} = y$$

- Recall that if $g(y) = f^{-1}(y)$ then $g'(y) = 1/f'(g(y))$
- Consider $g(y) = \ln(y)$ and $f(x) = \mathrm{e}^x$ (with $f'(x) = \mathrm{e}^x$)

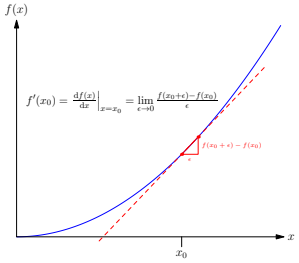
$$\frac{\mathrm{d}\ln(y)}{\mathrm{d}y} = \frac{1}{\mathrm{e}^{\ln(y)}} = \frac{1}{y}$$

Exponentials and Logarithms



Outline

1. Why Calculus?
2. Differentiation
3. **Vector and Matrix Calculus**



Derivatives in High Dimensions

- When working with functions $f : \mathbb{R}^n \to \mathbb{R}$ in many dimensions then there will typically be different derivative in different directions
- To compute the derivative in a direction $\boldsymbol{u} \in \mathbb{R}^n$ (where $\|\boldsymbol{u}\| = 1$) at a point $\boldsymbol{x} \in \mathbb{R}^n$ we use

$$\partial_{\boldsymbol{u}} F(\boldsymbol{x}) = \lim_{\epsilon \to 0} \frac{f(\boldsymbol{x} + \epsilon \boldsymbol{u}) - f(\boldsymbol{x})}{\epsilon}$$

- If $\boldsymbol{u} = \boldsymbol{\delta}_i = (0,\ldots,0,1,0,\ldots,0)$ (i.e. $u_i = 1$) then

$$\frac{\partial f(\boldsymbol{x})}{\partial x_i} = \lim_{\epsilon \to 0} \frac{f(\boldsymbol{x} + \epsilon \boldsymbol{\delta}_i) - f(\boldsymbol{x})}{\epsilon}$$

Taylor

- If we expand $f(\boldsymbol{x} + \epsilon \boldsymbol{u})$ to first order in ϵ

$$f(\boldsymbol{x} + \epsilon \boldsymbol{u}) = f(\boldsymbol{x}) + \epsilon \boldsymbol{u}^{\mathsf{T}} \boldsymbol{g}(\boldsymbol{x}) + O(\epsilon^2)$$

  then $g_i(\boldsymbol{x}) = \frac{\partial f(\boldsymbol{x})}{\partial x_i}$
- Recall we defined the vector of first order derivatives of $f(\boldsymbol{x})$ to be the gradient

$$\boldsymbol{\nabla} f(\boldsymbol{x}) = \begin{pmatrix} \frac{\partial f(\boldsymbol{x})}{\partial x_1} \\ \frac{\partial f(\boldsymbol{x})}{\partial x_2} \\ \vdots \\ \frac{\partial f(\boldsymbol{x})}{\partial x_n} \end{pmatrix}$$

- Thus

$$f(\boldsymbol{x} + \epsilon \boldsymbol{u}) = f(\boldsymbol{x}) + \epsilon \boldsymbol{u}^{\mathsf{T}} \boldsymbol{\nabla} f(\boldsymbol{x}) + O(\epsilon^2)$$

  This is the start of the high-dimensional Taylor expansion

Computing Gradients 1

- We can compute the gradient by writing out $f(\boldsymbol{x})$ componentwise and performing the partial derivative with respect to x_i

$$\boldsymbol{\nabla} \boldsymbol{w}^{\mathsf{T}} \mathbf{M} \boldsymbol{w} = \begin{pmatrix} \frac{\partial}{\partial w_1} \\ \frac{\partial}{\partial w_2} \\ \frac{\partial}{\partial w_3} \\ \vdots \end{pmatrix} \sum_{i,j} w_i M_{ij} w_j = \begin{pmatrix} \sum_j M_{1j} w_j + \sum_i w_i M_{i1} \\ \sum_j M_{2j} w_j + \sum_i w_i M_{i2} \\ \sum_j M_{3j} w_j + \sum_i w_i M_{i3} \\ \vdots \end{pmatrix}$$
$$= \mathbf{M} \boldsymbol{w} + \mathbf{M}^{\mathsf{T}} \boldsymbol{w}$$

- It is tedious to compute these things component-wise, but when you need to understand what is going on then go back to the basics

Computing Gradients 2

- A slicker way is just to expand $f(\boldsymbol{x} + \epsilon \boldsymbol{u})$
- Consider $f(\boldsymbol{x}) = \boldsymbol{x}^{\mathsf{T}} \mathbf{M} \boldsymbol{x} + \boldsymbol{a}^{\mathsf{T}} \boldsymbol{x}$

$$\begin{aligned} f(\boldsymbol{x} + \epsilon \boldsymbol{u}) &= (\boldsymbol{x} + \epsilon \boldsymbol{u})^{\mathsf{T}} \mathbf{M} (\boldsymbol{x} + \epsilon \boldsymbol{u}) + \boldsymbol{a}^{\mathsf{T}} (\boldsymbol{x} + \epsilon \boldsymbol{u}) \\ &= f(\boldsymbol{x}) + \epsilon \left(\boldsymbol{u}^{\mathsf{T}} \mathbf{M} \boldsymbol{x} + \boldsymbol{x}^{\mathsf{T}} \mathbf{M} \boldsymbol{u} + \boldsymbol{a}^{\mathsf{T}} \boldsymbol{u} \right) + O(\epsilon^2) \\ &= f(\boldsymbol{x}) + \epsilon \boldsymbol{u}^{\mathsf{T}} \left(\mathbf{M} \boldsymbol{x} + \mathbf{M}^{\mathsf{T}} \boldsymbol{x} + \boldsymbol{a} \right) + O(\epsilon^2) \end{aligned}$$

  using $\boldsymbol{x}^{\mathsf{T}} \mathbf{M} \boldsymbol{u} = \boldsymbol{u}^{\mathsf{T}} \mathbf{M}^{\mathsf{T}} \boldsymbol{x}$ and $\boldsymbol{a}^{\mathsf{T}} \boldsymbol{u} = \boldsymbol{u}^{\mathsf{T}} \boldsymbol{a}$
- But $f(\boldsymbol{x} + \epsilon \boldsymbol{u}) = f(\boldsymbol{x}) + \epsilon \boldsymbol{u}^{\mathsf{T}} \boldsymbol{\nabla} f(\boldsymbol{x}) + O(\epsilon^2)$ so

$$\boldsymbol{\nabla} f(\boldsymbol{x}) = \mathbf{M} \boldsymbol{x} + \mathbf{M}^{\mathsf{T}} \boldsymbol{x} + \boldsymbol{a}$$

Differentiating Matrices

- Often we have loss functions with respect to a matrix $\mathbf{W}$, e.g.

$$L(\mathbf{W}) = (\boldsymbol{a}^{\mathsf{T}} \mathbf{W} \boldsymbol{b} - c)^2$$

- We might want to find the minimum with respect to $\mathbf{W}$
- This occurs at a point $\mathbf{W}^*$ where $L(\mathbf{W})$ does not increase as we change $\mathbf{W}$ in any way
- That is, we seek a $\mathbf{W}^*$ such that, for any matrices $\mathbf{U}$

$$L(\mathbf{W}^* + \epsilon \mathbf{U}) - L(\mathbf{W}^*) = O(\epsilon^2)$$

Generalised Gradient

- We can generalise the idea of gradient to matrices

$$\frac{\partial L(\mathbf{W})}{\partial \mathbf{W}} = \begin{pmatrix} \frac{\partial L(\mathbf{W})}{\partial W_{11}} & \frac{\partial L(\mathbf{W})}{\partial W_{12}} & \cdots & \frac{\partial L(\mathbf{W})}{\partial W_{1m}} \\ \frac{\partial L(\mathbf{W})}{\partial W_{21}} & \frac{\partial L(\mathbf{W})}{\partial W_{22}} & \cdots & \frac{\partial L(\mathbf{W})}{\partial W_{2m}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L(\mathbf{W})}{\partial W_{n1}} & \frac{\partial L(\mathbf{W})}{\partial W_{n2}} & \cdots & \frac{\partial L(\mathbf{W})}{\partial W_{nm}} \end{pmatrix}$$

- From an identical argument we used for vectors

$$L(\mathbf{W} + \epsilon \mathbf{U}) = L(\mathbf{W}) + \epsilon \mathrm{tr} \mathbf{U}^{\mathsf{T}} \frac{\partial L(\mathbf{W})}{\partial \mathbf{W}} + O(\epsilon^2)$$

  where

$$\mathrm{tr} \mathbf{U}^{\mathsf{T}} \mathbf{G} = \sum_i \left[\mathbf{U}^{\mathsf{T}} \mathbf{G} \right]_{ii} = \sum_{ij} U_{ji} G_{ji} = \sum_{ij} U_{ij} G_{ij} = \langle \mathbf{U}, \mathbf{G} \rangle$$

Example

- Suppose

$$L(\boldsymbol{W}) = \left(\boldsymbol{a}^\mathsf{T}\boldsymbol{W}\boldsymbol{b} - c\right)^2$$

then

$$\begin{aligned} L(\boldsymbol{W} + \epsilon\boldsymbol{U}) &= \left(\boldsymbol{a}^\mathsf{T}(\boldsymbol{W} + \epsilon\boldsymbol{U})\boldsymbol{b} - c\right)^2 = \left(\boldsymbol{a}^\mathsf{T}\boldsymbol{W}\boldsymbol{b} + \epsilon\boldsymbol{a}^\mathsf{T}\boldsymbol{U}\boldsymbol{b} - c\right)^2 \\ &= L(\boldsymbol{W}) + 2\epsilon\left(\boldsymbol{a}^\mathsf{T}\boldsymbol{W}\boldsymbol{b} - c\right)\left(\boldsymbol{a}^\mathsf{T}\boldsymbol{U}\boldsymbol{b}\right) + O(\epsilon^2) \end{aligned}$$

- Now

$$\boldsymbol{a}^\mathsf{T}\boldsymbol{U}\boldsymbol{b} = \sum_{ij} a_i U_{ij} b_j = \sum_{ij} U_{ji} a_j b_i = \mathrm{tr}\boldsymbol{U}^\mathsf{T}\boldsymbol{a}\boldsymbol{b}^\mathsf{T}$$

Thus $\frac{\partial L(\boldsymbol{W})}{\partial \boldsymbol{W}} = 2\left(\boldsymbol{a}^\mathsf{T}\boldsymbol{W}\boldsymbol{b} - c\right)\boldsymbol{a}\boldsymbol{b}^\mathsf{T}$

Traces

- The trace of a matrix is the sum of its diagonal elements

$$\mathrm{tr}\boldsymbol{A} = \mathrm{tr}\boldsymbol{A}^\mathsf{T} = \sum_i A_{ii}$$

- Clearly $\mathrm{tr}c\boldsymbol{A} = c\mathrm{tr}\boldsymbol{A}$
- Also $\mathrm{tr}(\boldsymbol{A} + \boldsymbol{B}) = \mathrm{tr}\boldsymbol{A} + \mathrm{tr}\boldsymbol{B}$
- We note that

$$\mathrm{tr}\boldsymbol{A}\boldsymbol{B} = \sum_{i,j} A_{ij}B_{ji} = \sum_{i,j} B_{ij}A_{ji} = \mathrm{tr}\boldsymbol{B}\boldsymbol{A}$$

- It follows that

$$\mathrm{tr}\boldsymbol{A}\boldsymbol{B}\boldsymbol{C}\boldsymbol{D} = \mathrm{tr}\boldsymbol{D}\boldsymbol{A}\boldsymbol{B}\boldsymbol{C} = \mathrm{tr}\boldsymbol{C}\boldsymbol{D}\boldsymbol{A}\boldsymbol{B} = \mathrm{tr}\boldsymbol{B}\boldsymbol{C}\boldsymbol{D}\boldsymbol{A}$$

Quick Matrix Differentiation

- Let

$$\partial_{\boldsymbol{U}} f(\boldsymbol{X}) = \lim_{\epsilon\to 0} \frac{f(\boldsymbol{X} + \epsilon\boldsymbol{U}) - f(\boldsymbol{X})}{\epsilon} = \mathrm{tr}\boldsymbol{U}^\mathsf{T}\frac{\partial f(\boldsymbol{X})}{\partial \boldsymbol{X}}$$

- E.g.

$$\begin{aligned} \partial_{\boldsymbol{U}}\mathrm{tr}\boldsymbol{A}\boldsymbol{X}\boldsymbol{B} &= \lim_{\epsilon\to 0}\frac{1}{\epsilon}\mathrm{tr}\boldsymbol{A}\left(\boldsymbol{X} + \epsilon\boldsymbol{U}\right)\boldsymbol{B} - \mathrm{tr}\boldsymbol{A}\boldsymbol{X}\boldsymbol{B} \\ &= \mathrm{tr}\,\boldsymbol{A}\boldsymbol{U}\boldsymbol{B} = \mathrm{tr}\,\boldsymbol{B}^\mathsf{T}\boldsymbol{U}^\mathsf{T}\boldsymbol{A}^\mathsf{T} = \mathrm{tr}\,\boldsymbol{U}^\mathsf{T}\boldsymbol{A}^\mathsf{T}\boldsymbol{B}^\mathsf{T} \end{aligned}$$

thus

$$\frac{\partial \mathrm{tr}\boldsymbol{A}\boldsymbol{X}\boldsymbol{B}}{\partial \boldsymbol{X}} = \boldsymbol{A}^\mathsf{T}\boldsymbol{B}^\mathsf{T}$$

Log Determinants

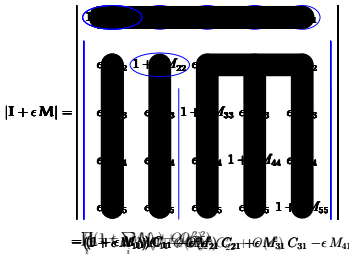
- We often come across logarithms of determinants of matrices, $\log(|\boldsymbol{M}|)$
- For GP we want to choose $\mathbf{K}$ to maximise the marginal likelihood, $\log\left(|\mathbf{K} + \sigma^2\mathbf{I}|\right)$
- To find the derivative of $\log(|\boldsymbol{X}|)$ we consider

$$\begin{aligned} \log(|\boldsymbol{X} + \epsilon\boldsymbol{U}|) &= \log\left(|\boldsymbol{X}(\mathbf{I} + \epsilon\boldsymbol{X}^{-1}\boldsymbol{U})|\right) \\ &= \log\left(|\boldsymbol{X}||\mathbf{I} + \epsilon\boldsymbol{X}^{-1}\boldsymbol{U}|\right) \\ &= \log(|\boldsymbol{X}|) + \log\left(|\mathbf{I} + \epsilon\boldsymbol{X}^{-1}\boldsymbol{U}|\right) \end{aligned}$$

  - ⋆ Using $|\boldsymbol{A}\boldsymbol{B}| = |\boldsymbol{A}||\boldsymbol{B}|$
  - ⋆ Using $\log(ab) = \log(a) + \log(b)$

Determinants

$$\begin{aligned} |\mathbf{I} + \epsilon\mathbf{M}| &= \begin{vmatrix} 1 + \epsilon M_{11} & \epsilon M_{12} \\ \epsilon M_{21} & 1 + \epsilon M_{22} \end{vmatrix} = (1 + \epsilon M_{11})(1 + \epsilon M_{22}) - \epsilon^2 M_{21}M_{12} \\ &= 1 + \epsilon(M_{11} + M_{22}) + O(\epsilon^2) \end{aligned}$$



$$= (1 + \epsilon M_{11})(1 + \epsilon M_{22})(1 + \epsilon M_{33})(1 + \epsilon M_{44})(1 + \epsilon M_{55}) + \epsilon M_{21}C_{21} + O(\epsilon^2) - \epsilon M_{31}C_{31} - \epsilon M_{41}C_{41} + \epsilon M_{51}C_{51}$$

Putting it Together

- Recall

$$\begin{aligned} \log(|\boldsymbol{X} + \epsilon\boldsymbol{U}|) - \log(|\boldsymbol{X}|) &= \log\left(|\mathbf{I} + \epsilon\boldsymbol{X}^{-1}\boldsymbol{U}|\right) \\ &= \log\left(1 + \epsilon\mathrm{tr}\,\boldsymbol{X}^{-1}\boldsymbol{U} + O(\epsilon)^2\right) \\ &= \epsilon\mathrm{tr}\,\boldsymbol{X}^{-1}\boldsymbol{U} + O(\epsilon)^2 \\ &= \epsilon\mathrm{tr}\,\boldsymbol{U}^\mathsf{T}\left(\boldsymbol{X}^{-1}\right)^\mathsf{T} + O(\epsilon) \end{aligned}$$

using $\log(1 + x) = x + \frac{x^2}{2} + \cdots$

- Thus $\partial_{\boldsymbol{U}}\log(|\boldsymbol{X}|) = \mathrm{tr}\,\boldsymbol{U}^\mathsf{T}\left(\boldsymbol{X}^{-1}\right)^\mathsf{T}$
- Or

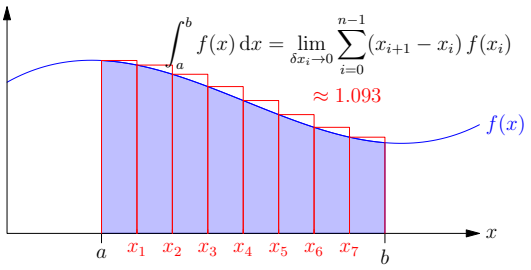
$$\frac{\partial \log(|\boldsymbol{X}|)}{\partial \boldsymbol{X}} = \left(\boldsymbol{X}^{-1}\right)^\mathsf{T}$$

Summary

- With care you can differentiate most expressions
- The chain and product rule are incredibly powerful tools
- We can generalise differentiation to vectors and matrices
- There are a number of surprisingly useful results see **The Matrix Cookbook**
- Next stop: integration

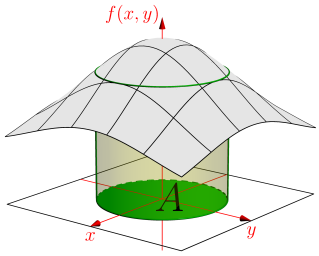
Advanced Machine Learning

Integral Calculus



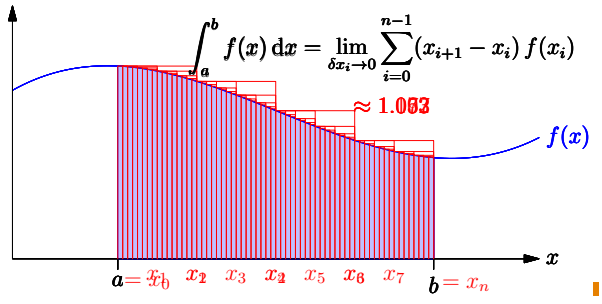
Riemann Integration, integration by parts, gaussian integrals

Outline



Riemann Integral

- Integrals represent area beneath a curve



Linearity of Integration

- Integration is a linear operator

$$\begin{aligned}
\int_a^b (r f(x) + s g(x))\,\mathrm{d}x &= \lim_{\delta x_i \to 0} \sum_{i=0}^{n-1} (x_{i+1} - x_i)\,(r f(x_i) + s g(x_i)) \\
&= \lim_{\delta x_i \to 0} \left(\sum_{i=0}^{n-1} (x_{i+1} - x_i) r f(x_i) + \sum_{i=0}^{n-1} (x_{i+1} - x_i) s g(x_i) \right) \\
&= \lim_{\delta x_i \to 0} \left(r \sum_{i=0}^{n-1} (x_{i+1} - x_i) f(x_i) + s \sum_{i=0}^{n-1} (x_{i+1} - x_i) g(x_i) \right) \\
&= r \lim_{\delta x_i \to 0} \sum_{i=0}^{n-1} (x_{i+1} - x_i) f(x_i) + s \lim_{\delta x_i \to 0} \sum_{i=0}^{n-1} (x_{i+1} - x_i) g(x_i) \\
&= r \int_a^b f(x)\mathrm{d}x + s \int_a^b f(x)\mathrm{d}x
\end{aligned}$$

Fundamental Law of Calculus

- Let

$$I(a,x) = \int_a^x f(z)\mathrm{d}z = \lim_{\delta z_i \to 0} \sum_{i=0}^{n-1} (z_{i+1} - z_i) f(z_i)$$

- Now for small δx

$$I(a,x+\delta x) = \int_a^{x+\delta x} f(z)\mathrm{d}z = \lim_{\delta z_i \to 0} \sum_{i=0}^{n-1} (z_{i+1} - z_i) f(z_i) + \delta x f(x)$$

- Thus

$$\frac{\mathrm{d}I(a,x)}{\mathrm{d}x} = \lim_{\delta x \to 0} \frac{I(x+\delta x) - I(x)}{\delta x} = \lim_{\delta x \to 0} \frac{\delta x f(x)}{\delta x} = f(x)$$

The Other Way Around

- Consider

$$\begin{aligned}
\int_a^b \frac{\mathrm{d}f(x)}{\mathrm{d}x}\mathrm{d}x &= \int_a^b \lim_{\delta x \to 0} \frac{f(x+\delta x) - f(x)}{\delta x}\mathrm{d}x \\
&= \lim_{x_{i+1} - x_i \to 0} \sum_{i=0}^{n-1} (x_{i+1} - x_i) \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \\
&= \lim_{x_{i+1} - x_i \to 0} \sum_{i=0}^{n-1} (f(x_{i+1}) - f(x_i)) \\
&= (f(x_1) - f(x_0)) + (f(x_2) - f(x_1)) + (f(x_3) - f(x_2)) + \cdots \\
&\quad + (f(x_{n-1}) - f(x_{n-2})) + (f(x_n) - f(x_{n-1})) \\
&= f(x_n) - f(x_0) = f(b) - f(a)
\end{aligned}$$

- We can think of integration as an **anti-derivative** it undoes differentiation

Indefinite Integrals

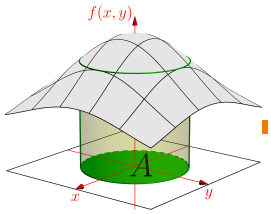
- So far we have considered **definite integrals** where we integrate between two points (a and b)
- However, when think about integration as an anti-derivative, it is useful to think of a function $F(x) = \int f(x)\mathrm{d}x$
- So that $F'(x) = f(x)$
- However the function $F(x)$, $F(x)+1$, $F(x)+\pi$, etc. all have the same derivative so $F(x)$ is only defined up to an additive constant
- Note that the definite integral is given by

$$\int_a^b f(x)\mathrm{d}x = F(b) - F(a)$$

Multiple Integrals

- For functions involving many independent variables (e.g. $f(x,y)$, $f(x,y,z)$, $f(\boldsymbol{x})$) we can integrate over multiple dimensions
- For example

$$\iint_A f(x,y)\mathrm{d}x\mathrm{d}y$$

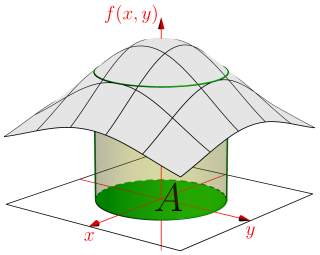


- It gets tedious writing multiple integral signs and I tend to write just one

$$\int \cdots \int f(x_1, x_2, \ldots, x_n)\mathrm{d}x_1\mathrm{d}x_2\cdots\mathrm{d}x_n = \int f(\boldsymbol{x})\mathrm{d}\boldsymbol{x}$$

Outline

1. Defining Integrals
2. **Doing Integrals**
3. Gaussian Integrals



Performing Integration

- A key method for performing integrals is through knowledge of the anti-derivative
- If we know $F'(x) = f(x)$ then $F(x) + c = \int f(x)\mathrm{d}x$
- E.g. we know that $\mathrm{d}x^n/\mathrm{d}x = nx^{n-1}$ therefore

$$\int x^{n-1}\mathrm{d}x = \frac{1}{n}\int \frac{\mathrm{d}x^n}{\mathrm{d}x}\mathrm{d}x = \frac{x^n}{n} + c$$

and

$$\int_a^b x^{n-1}\mathrm{d}x = \frac{b^n}{n} - \frac{a^n}{n}$$

Is Integration Straightforward?

- We saw due to the product and chain rules that we can differentiate almost anything. Given integration is the anti-derivative can we integrate anything?
- Products and compositions

$$\int f(x)g(x)\mathrm{d}x = ? \qquad \int f(g(x))\mathrm{d}x = ?$$

- Unfortunately, unlike differentiation we don't have a small parameter we can expand in
- In general integration is hard

Integration by Parts

- Recall the product rule $\frac{\mathrm{d}f(x)g(x)}{\mathrm{d}x} = \frac{\mathrm{d}f(x)}{\mathrm{d}x}g(x) + f(x)\frac{\mathrm{d}g(x)}{\mathrm{d}x}$
- Integrating we get

$$\begin{aligned}\int_a^b \frac{\mathrm{d}f(x)g(x)}{\mathrm{d}x}\mathrm{d}x &= \int_a^b \frac{\mathrm{d}f(x)}{\mathrm{d}x}g(x)\mathrm{d}x + \int_a^b f(x)\frac{\mathrm{d}g(x)}{\mathrm{d}x}\mathrm{d}x \\ &= [f(x)g(x)]_a^b = f(b)g(b) - f(a)g(a)\end{aligned}$$

- Unfortunately we get two integrals, but we can turn this around

$$\int_a^b f(x)\frac{\mathrm{d}g(x)}{\mathrm{d}x}\mathrm{d}x = [f(x)g(x)]_a^b - \int_a^b \frac{\mathrm{d}f(x)}{\mathrm{d}x}g(x)\mathrm{d}x$$

whether this is helpful depends on $f(x)$ and $g(x)$

Example of Integration by Parts

- Consider

$$\begin{aligned}\Pi(z) &= \int_0^\infty x^z \mathrm{e}^{-x}\mathrm{d}x = \int_0^\infty x^z \frac{\mathrm{d}(-\mathrm{e}^{-x})}{\mathrm{d}x}\mathrm{d}x \\ &= \left[x^z(-\mathrm{e}^{-x})\right]_0^\infty - \int_0^\infty \frac{\mathrm{d}x^z}{\mathrm{d}x}(-\mathrm{e}^{-x})\mathrm{d}x \\ &= \int_0^\infty (zx^{z-1})\mathrm{e}^{-x}\mathrm{d}x = z\int_0^\infty x^{z-1}\mathrm{e}^{-x}\mathrm{d}x = z\Pi(z-1)\end{aligned}$$

- Thus $\Pi(z) = z\Pi(z-1)$, but

$$\Pi(0) = \int_0^\infty \mathrm{e}^{-z}\mathrm{d}z = \left[-\mathrm{e}^{-x}\right]_0^\infty = -\mathrm{e}^{-\infty} - (-\mathrm{e}^0) = 1$$

- Now
  $\Pi(n) = n\Pi(n-1) = n(n-1)\Pi(n-2) = n(n-1)(n-2)\ldots 1 = n!$

Substitution

- We can make a transformation from x to $u = u(x)$

$$\begin{aligned}\int_a^b f(x)\mathrm{d}x &= \lim_{\delta x_i \to 0}\sum_{i=0}^{n-1} f(x_i)(x_{i+1} - x_i) \\ &= \lim_{\delta u_i \to 0}\sum_{i=0}^{n-1} f(x(u_i))\frac{x(u_{i+1}) - x(u_i)}{u_{i+1} - u_i}(u_{i+1} - u_i) \\ &= \int_{u(a)}^{u(b)} f(x(u))\frac{\mathrm{d}x(u)}{\mathrm{d}u}\mathrm{d}u\end{aligned}$$

  - where u_i is such that $x(u_i) = x_i$ or $u_i = u(x_i)$ where $u(x)$ is the inverse of $x(u)$
  - using $\lim_{\delta u_i \to 0}\frac{x(u_{i+1}) - x(u_i)}{u_{i+1} - u_i} = \frac{\mathrm{d}x(u_i)}{\mathrm{d}u}$

Example of Integration by Substitution

- We consider $I(n) = \int_0^\infty x^n \mathrm{e}^{-x^2/2}\mathrm{d}x$
- Let $u(x) = x^2/2$ or $x(u) = \sqrt{2u}$ so that

$$\frac{\mathrm{d}x(u)}{\mathrm{d}u} = \frac{1}{\sqrt{2u}} \qquad u(0) = 0 \qquad u(\infty) = \infty$$

- Thus

$$\begin{aligned}I(n) &= \int_0^\infty \left(\sqrt{2u}\right)^n \mathrm{e}^{-u}\frac{1}{\sqrt{2u}}\mathrm{d}u \\ &= 2^{\frac{n-1}{2}}\int_0^\infty u^{\frac{n-1}{2}}\mathrm{e}^{-u}\mathrm{d}u = 2^{\frac{n-1}{2}}\Pi\left(\frac{n-1}{2}\right)\end{aligned}$$

- $I(1) = 1$, $I(3) = 2 \times 1! = 2$, $I(5) = 2^2 \times 2! = 8$, but
  $I(0) = \Pi(-1/2)/\sqrt{2}$, $I(2) = \sqrt{2}\Pi(1/2) = \Pi(-1/2)/\sqrt{2}$

Changing Variables in Multidimensional Space

- When changing variables in many dimensions $\boldsymbol{x} \to \boldsymbol{u}$ the change of variables involves the Jacobian

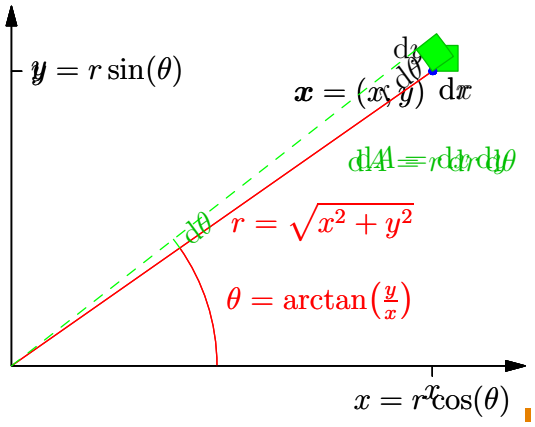
$$\int f(\boldsymbol{x})\mathrm{d}\boldsymbol{x} = \int f(\boldsymbol{x}(\boldsymbol{u}))|\det(\mathbf{J})|\mathrm{d}\boldsymbol{u}, \qquad \boldsymbol{J} = \begin{pmatrix} \frac{\partial x_1}{\partial u_1} & \frac{\partial x_1}{\partial u_2} & \cdots & \frac{\partial x_1}{\partial u_n} \\ \frac{\partial x_1}{\partial u_2} & \frac{\partial x_2}{\partial u_2} & \cdots & \frac{\partial x_2}{\partial u_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial u_1} & \frac{\partial x_n}{\partial u_2} & \cdots & \frac{\partial x_n}{\partial u_n} \end{pmatrix}$$

- E.g. transforming from Cartesian coordinates (x,y) to polar coordinates (r,θ) then $x = r\cos(\theta)$ and $y = r\sin(\theta)$

$$\begin{aligned}|\det(\mathbf{J})| &= \left|\det\begin{pmatrix} \frac{\partial r\cos(\theta)}{\partial r} & \frac{\partial r\cos(\theta)}{\partial \theta} \\ \frac{\partial r\sin(\theta)}{\partial r} & \frac{\partial r\sin(\theta)}{\partial \theta} \end{pmatrix}\right| = \left|\det\begin{pmatrix} \cos(\theta) & -r\sin(\theta) \\ \sin(\theta) & r\cos(\theta) \end{pmatrix}\right| \\ &= r\left(\cos^2(\theta) + \sin^2(\theta)\right) = r\end{aligned}$$

- That is, $\mathrm{d}x\mathrm{d}y = r\mathrm{d}r\mathrm{d}\theta$

Change of Variables in Pictures



Differentiating Through the Integral

- A trick that sometimes works is differentiating through an integral, e.g. consider finding moments

$$M_n = \mathbb{E}[X^n] = \int_{-\infty}^{\infty} x^n f_X(x)\mathrm{d}x$$

- We can define a momentum generating function

$$Z(\ell) = \int_{-\infty}^{\infty} \mathrm{e}^{\ell x} f_X(x)\mathrm{d}x$$

- Then $M_n = Z^{(n)}(0)$

$$\left.\frac{\mathrm{d}^n Z(\ell)}{\mathrm{d}\ell^n}\right|_{\ell=0} = \int_{-\infty}^{\infty} \left.\frac{\mathrm{d}^n \mathrm{e}^{\ell x}}{\mathrm{d}\ell^n}\right|_{\ell=0} f_X(x)\mathrm{d}x = \int_{-\infty}^{\infty} x^n f_X(x)\mathrm{d}x = M_n$$

Cumulant Generating Function

- Note that $\mathrm{e}^{\ell x} = 1 + \ell x + \frac{1}{2}\ell^2 x^2 + \frac{1}{3!}\ell^3 x^3 + \cdots$
- So

$$Z(\ell) = \int_{-\infty}^{\infty} \mathrm{e}^{\ell x} f_X(x)\mathrm{d}x = 1 + \ell M_1 + \frac{1}{2}\ell^2 M_2 + \frac{1}{3!}\ell^3 M_3 + \cdots$$

- Now using $\log(1+\epsilon) = \epsilon - \frac{1}{2}\epsilon^2 + \frac{1}{3}\epsilon^3 + \cdots$

$$G(\ell) = \log(Z(\ell)) = \ell M_1 + \frac{1}{2}\ell^2(M_2 - M_1^2) + \frac{1}{3!}\ell^3(M_3 - 3M_2M_1 + 2M_1^3) + \cdots$$

- So that $\kappa_n = G^{(n)}(0)$, with $\kappa_1 = M_1$ (the mean), $\kappa_2 = M_2 - M_1^2$ (the variance), $\kappa_3 = M_3 - 3M_2M_1 + 2M_1^3$ (the third cumulant related to the skewness)

More Integration

- Although we have a few tricks, integration is hard
- Surprisingly integration sometimes is easier when carried out in the complex plane
- This is a beautiful part of mathematics (due largely to Cauchy)—but beyond the scope of this course
- Interestingly, also there is an algorithm that allows us to integrate a lot of function. It is sufficiently complicated that you need to write a computer algorithm of considerable complexity to implement it. Most symbolic manipulation packages (e.g. Mathematica) have implemented some part of this algorithm

Special Functions

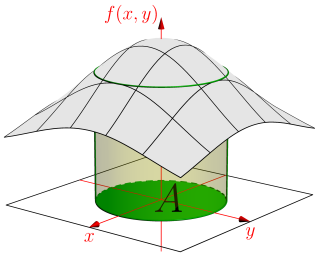
- There are integrals with no known closed form solution
- We saw that $\Pi(z) = \int_0^{\infty} x^z \mathrm{e}^{-x}\mathrm{d}x$ satisfies $\Pi(z) = z\Pi(z-1)$
- For integer n then $\Pi(n) = n!$ but for general z, the integal $\Pi(z)$ can't be written in terms of elementary functions
- We consider $\Pi(z)$ as a special function in its own right
- Although, history has left us with the gamma function instead

$$\Gamma(z) = \int_0^{\infty} x^{z-1}\mathrm{e}^{-x}\mathrm{d}x = \Pi(z-1)$$

- Other special function defined by integrals exist (e.g. the Bessel , Aire, hypergeometric, elliptic, error functions, . . .)

Outline

1. Defining Integrals
2. Doing Integrals
3. **Gaussian Integrals**



Gaussian Integrals

- Gaussian integrals are integrals involving e^{-x^2}, e.g.

$$\int_{-\infty}^{\infty} \mathrm{e}^{-x^2}\mathrm{d}x \qquad \int_{-\infty}^{\infty} x^4 \mathrm{e}^{-ax^2-bx}\mathrm{d}x$$

- They are important in computing integrals with respect to the normal distribution

$$\mathcal{N}(x|\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma}\mathrm{e}^{-(x-\mu)^2/(2\sigma^2)}$$

- The great news is that these integrals are all doable
- The bad news is that they are quite tricky to do

The Gaussian Integral

- The integral over a Gaussian is surprisingly difficult

$$I_1 = \int_{-\infty}^{\infty} \mathrm{e}^{-x^2/2}\mathrm{d}x$$

- There is a nice trick which is to consider

$$I_1^2 = \int_{-\infty}^{\infty} \mathrm{e}^{-x^2/2}\mathrm{d}x \int_{-\infty}^{\infty} \mathrm{e}^{-y^2/2}\mathrm{d}y = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} \mathrm{e}^{-(x^2+y^2)/2}\mathrm{d}x\mathrm{d}y$$

- Making the change of variables $r = \sqrt{x^2+y^2}$ and $\theta = \arctan(y/x)$ (so that $x = r\cos(\theta)$, $y = r\sin(\theta)$ and $x^2 + y^2 = r^2$)

$$I_1^2 = \int_0^{2\pi} \mathrm{d}\theta \int_0^{\infty} r e^{-r^2/2}\mathrm{d}r = 2\pi \int_0^{\infty} r e^{-r^2/2}\mathrm{d}r$$

The Gaussian Integral Continued

- From before

$$I_1^2 = 2\pi \int_0^\infty r e^{-r^2/2} \mathrm{d}r$$

- Finally let $u = r^2/2$ so that $\mathrm{d}u/\mathrm{d}r = r$ or $\mathrm{d}u = r\mathrm{d}r$ we get

$$I_1^2 = 2\pi \int_0^\infty e^{-u} \mathrm{d}u = 2\pi$$

- So that $I_1 = \sqrt{2\pi}$
- Incidentally, $I_1 = \sqrt{2}\Pi(-1/2)$ so $\Pi(-1/2) = \Gamma(1/2) = \sqrt{\pi}$

Normal Distribution

- We consider

$$I_2 = \int_{-\infty}^{\infty} \mathrm{e}^{-(x-\mu)^2/(2\sigma^2)} \mathrm{d}x$$

- Making the change of variables $z = (x-\mu)/\sigma$ so that $\mathrm{d}z = \mathrm{d}x/\sigma$ or $\mathrm{d}x = \sigma \mathrm{d}z$. Then

$$I_2 = \sigma \int_{-\infty}^{\infty} \mathrm{e}^{-z^2/2} \mathrm{d}z = \sigma I_1 = \sqrt{2\pi}\sigma$$

- Note that the *probability density function* (PDF) for a normally distributed random variable is given by

$$\mathcal{N}(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} \mathrm{e}^{-(x-\mu)^2/(2\sigma^2)}$$

Multi-dimensional Gaussians

- Consider

$$I_3 = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{1}{2}\|\boldsymbol{x}\|_2^2} \mathrm{d}x_1 \cdots \mathrm{d}x_n$$

where $\boldsymbol{x} = (x_1, x_2, \ldots, x_n)^\mathsf{T}$

- Note that $\|\boldsymbol{x}\|_2^2 = x_1^2 + x_2^2 + \cdots + x_n^2$ and using $\mathrm{e}^{\sum_i a_i} = \prod_i \mathrm{e}^{a_i}$

$$\begin{aligned} I_3 &= \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{1}{2}\sum_{i=1}^n x_i^2} \mathrm{d}x_1 \cdots \mathrm{d}x_n \\ &= \prod_{i=1}^n \int_{-\infty}^{\infty} \mathrm{e}^{-x_i^2/2} \mathrm{d}x_i = \prod_{i=1}^n \sqrt{2\pi} = (2\pi)^{n/2} \end{aligned}$$

Full Multi-variate Normal

- Consider

$$I_4 = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\mathsf{T} \boldsymbol{\Xi}^{-1} (\boldsymbol{x}-\boldsymbol{\mu})} \mathrm{d}x_1 \cdots \mathrm{d}x_n$$

- Let $\boldsymbol{\Xi}^{-1} = \boldsymbol{V}\boldsymbol{\Lambda}^{-1}\boldsymbol{V}^\mathsf{T}$ and make the change of variables $\boldsymbol{y} = \boldsymbol{V}^\mathsf{T}(\boldsymbol{x}-\boldsymbol{\mu})$
- The Jacobian $\mathbf{J}$ has elements (note that $\boldsymbol{x} = \boldsymbol{V}\boldsymbol{y} + \boldsymbol{\mu}$)

$$J_{ij} = \frac{\partial x_i}{\partial y_j} = \frac{\partial}{\partial y_j}\left(\sum_{k=1}^n V_{ik} y_k + \mu_i\right) = V_{ij}$$

- So that $\mathbf{J} = \mathbf{V}$ and consequently $|\det(\mathbf{J})| = |\det(\mathbf{V})| = 1$ then

$$I_4 = \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} \mathrm{e}^{-\frac{1}{2}\boldsymbol{y}^\mathsf{T}\boldsymbol{\Lambda}^{-1}\boldsymbol{y}} \mathrm{d}y_1 \cdots \mathrm{d}y_n. = \prod_{i=1}^n \int_{-\infty}^{\infty} \mathrm{e}^{-y_i^2/(2\lambda_i)} \mathrm{d}y_i = \prod_i \sqrt{2\pi\lambda_i}$$

Determinants

- Using the facts, that $\boldsymbol{\Xi} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T}$ then

$$\det(\boldsymbol{\Xi}) = \det(\boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T}) = \det(\boldsymbol{V})\det(\boldsymbol{\Lambda})\det(\boldsymbol{V}^\mathsf{T}) = \det(\boldsymbol{\Lambda}) = \prod_{i=1}^n \lambda_i$$

using $\det(\mathbf{A}\mathbf{B}) = \det(\mathbf{A})\det(\mathbf{B})$ and $\det(\mathbf{V}) = 1$

- Recall $I_4 = \prod_i \sqrt{2\pi\lambda_i} = (2\pi)^{n/2}\sqrt{\det(\boldsymbol{\Xi})}$
- We note for an $n \times n$ matrix $\mathbf{M}$ then $\det(c\mathbf{M}) = c^n \det(\mathbf{M})$ so that

$$I_4 = (2\pi)^{n/2}\sqrt{\det(\boldsymbol{\Xi})} = \sqrt{\det(2\pi\boldsymbol{\Xi})}$$

- Finally, we get that for the PDF of a normal to integrate to 1

$$\mathcal{N}(\boldsymbol{x} \mid \boldsymbol{\mu}, \boldsymbol{\Xi}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Xi})}} \mathrm{e}^{-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^\mathsf{T}\boldsymbol{\Xi}^{-1}(\boldsymbol{x}-\boldsymbol{\mu})}$$

Summary

- Integration is extra-ordinarily useful as a tool of analysis
- It occurs when you work with probabilities densities for continuous random variables
- Integration is beautiful, but hard—often impossible
- Normal distributions lucky almost always give raise to integrals that can be computed in closed form, although often it requires quite a bit of work
- Making friends with integration will give you a super-power that not too many people share