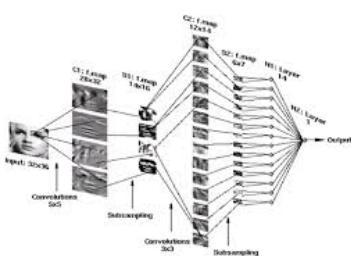


## Course Outline



## Course Details and Topics

## 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

## 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 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

- Notes on Moodle and

<https://ecs-vlc.github.io/aice1005/>  
<https://tinyurl.com/bddhrhcw>

- 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

## 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

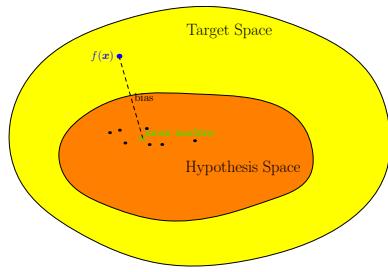
## Topics

- 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

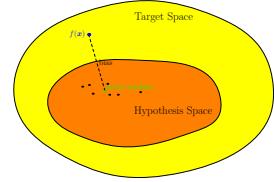
## Topics Continued

- 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

## When Machine Learning Works



When ML Works, Bias Variance



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  $\mathbf{x}$
- We construct a learning machine that makes a prediction  $\hat{f}(\mathbf{x}|\boldsymbol{\theta})$
- We typically choose the machine to minimise a *training loss*

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

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

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

## Generalisation Error

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

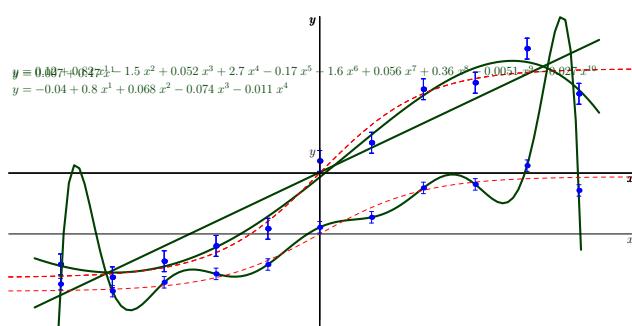
$$L_G(\boldsymbol{\theta}_{\mathcal{D}}) = \sum_{(\mathbf{x}, y) \in \mathcal{Z}} \mu(\mathbf{x}, y) (\hat{f}(\mathbf{x}|\boldsymbol{\theta}_{\mathcal{D}}) - y)^2$$

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

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

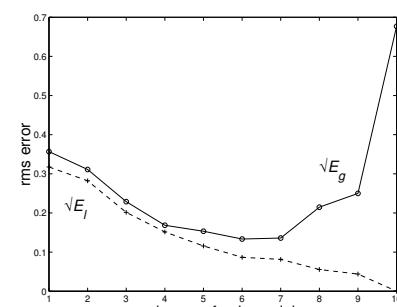
## Too Simple or Too Complex?

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

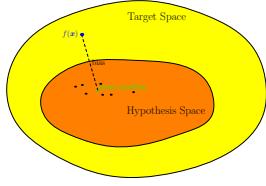


## Measuring Generalisation Error for Regression

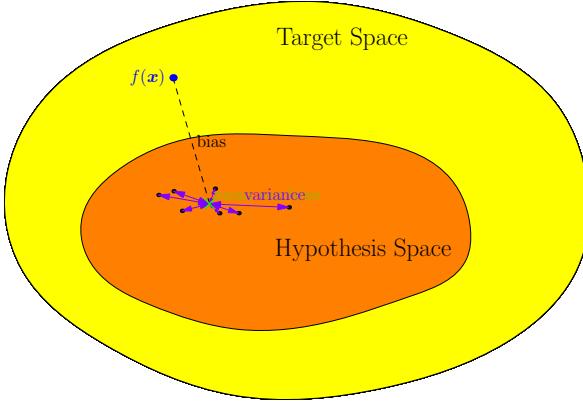
- Consider the regression example. The root mean squared error is



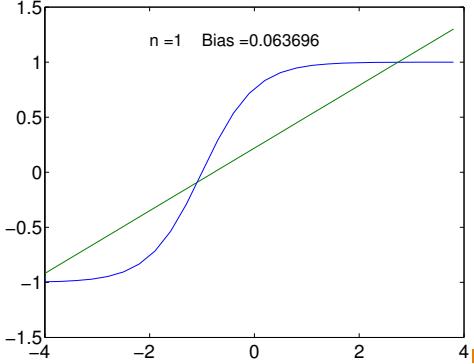
1. What Makes a Good Learning Machine?
2. Bias-Variance Dilemma



### Approximation and Estimation Errors



### Regression Example $n = 1$



### Bias and Variance

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

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

- 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  $\mu(x, y)$
- For each data set,  $\mathcal{D}$ , we would learn a different approximator  $\hat{f}(x|\theta_{\mathcal{D}})$
- 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

### Mean Machine

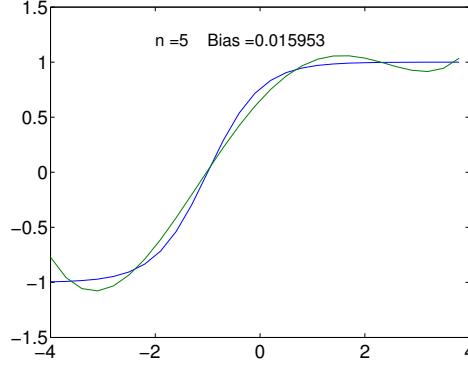
- To help understand generalisation we can consider the mean prediction with respect to machines trained with all data sets of size  $m$

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

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

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

### Regression Example $n = 5$



### Cross Term

- The cross term vanishes

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

- Thus

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

- We can write the expected generalisation loss as

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

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

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

- 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

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

- 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).

**Over-Fitting**

Overfitting, regularisation, feature selection

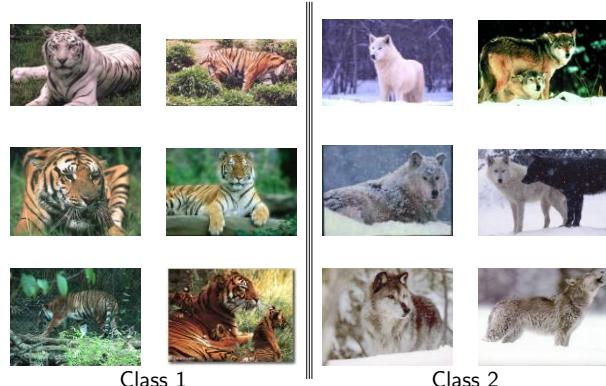
1. Over-fitting?
2. Controlling Complexity
3. Hidden structure
4. Regularisation

**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****Which Category?****Spurious Rules**

- Which category does the following image belong to?



- You ask a learning machine to solve a task based on data ■
- It will find a rule that does this, but not necessarily 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****Are MLPs Universal Approximators?**

$$\begin{aligned}
 x_0 &= 000 \quad y_0 = \begin{cases} 0 \\ \text{x} \end{cases} \\
 x_1 &= 100 \quad y_1 = \begin{cases} 0 \\ 1 \end{cases} \quad \text{unseen} \\
 x_2 &= 010 \quad y_2 = \begin{cases} \text{x} \\ 1 \end{cases} \\
 x_3 &= 110 \quad y_3 = \begin{cases} \text{x} \\ 1 \end{cases} \\
 x_4 &= 001 \quad y_4 = \begin{cases} 0 \\ \text{x} \end{cases} \quad \text{seen} \\
 x_5 &= 101 \quad y_5 = \begin{cases} 0 \\ \text{x} \end{cases} \\
 x_6 &= 011 \quad y_6 = \begin{cases} 0 \\ 1 \end{cases} \\
 x_7 &= 111 \quad y_7 = \begin{cases} 0 \\ 1 \end{cases} \quad \text{unseen}
 \end{aligned}$$

- 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! ■

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



- 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
  - { "blue" : 0, "brown" : 1, "green" : 2, "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

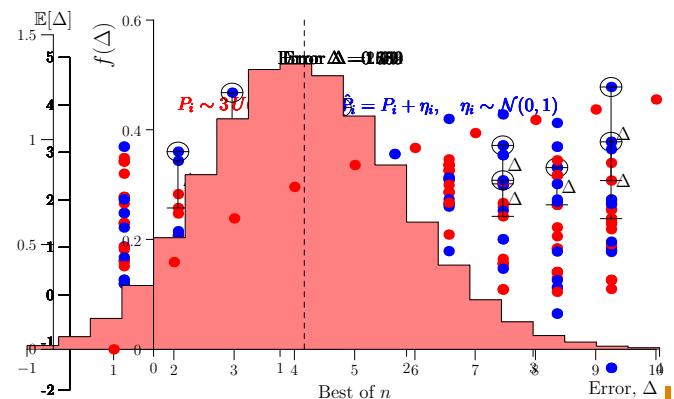
- Recall, we want to predict **unseen** data
- You cannot use data that you have trained on—you will overfit
- Need to split your data up into training and validation set
- Use the validation set to choose the hyper-parameters
- You need a separate testing set if you want to measure your generalisation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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

17

## The Overfitting Game



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

18

## 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

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

D <sub>1</sub>	D <sub>2</sub>	D <sub>3</sub>	D <sub>4</sub>	D <sub>5</sub>	D <sub>6</sub>	D <sub>7</sub>	D <sub>8</sub>	D <sub>9</sub>	D <sub>10</sub>	D <sub>11</sub>	D <sub>12</sub>	D <sub>13</sub>	D <sub>14</sub>	D <sub>15</sub>	D <sub>16</sub>	D <sub>17</sub>	D <sub>18</sub>	D <sub>19</sub>	D <sub>20</sub>
----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------	-----------------

Training Data: D<sub>1</sub>, D<sub>2</sub>, ..., D<sub>19</sub> (all data points except the last one)

Testing Data: D<sub>20</sub> (the last data point)

Leave-one-out cross-validation

$$MSE = \frac{5.5 + 8.5 + 1.8 + 2.3 + 4.8 + 3.7 + 3.6 + 7.4 + 4.6 + 0.99 + 4.5 + 4.6 + 5.4 + 6.2 + 3.3 + 2.7}{19} = 4.3$$

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

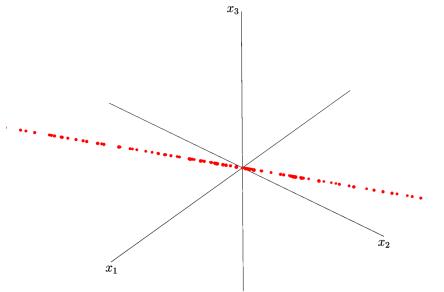
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

19

## Hidden Structure

Can't spot low dimensional data by looking at numbers



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

20

## 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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

21

## 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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

22

## 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.

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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

23

## Outline

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



24

## Explicit Regularisation

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

$$L = \sum_{k=1}^m (\mathbf{w}^\top \mathbf{x}_k - y_k)^2 + \nu \|\mathbf{w}\|^2$$

(Good to normalise features)

- Second term is minimised when  $w_i = 0$
- If  $w_i$  is large then

$$f(\mathbf{x}|\mathbf{w}) = \mathbf{w}^\top \mathbf{x} = \sum_{i=1}^p w_i x_i$$

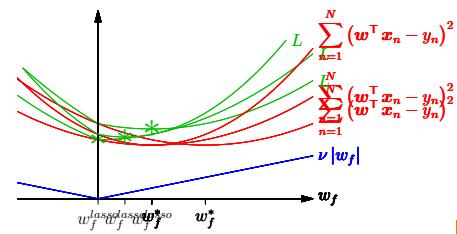
varies rapidly as we change  $x_i$

## Lasso

- We can use other regularisers

$$L = \sum_{k=1}^m (\mathbf{w}^\top \mathbf{x}_k - y_k)^2 + \nu \sum_{i=1}^p |w_i|$$

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

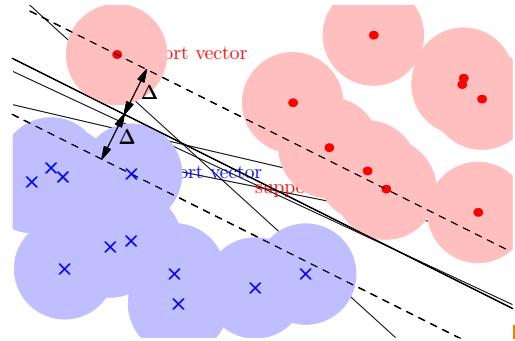


## Implicit Regularisation

- 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 margin

## 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  $\gamma$ )

## 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

### Symmetry



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



*Inductive Bias, Symmetry, Invariance, 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?

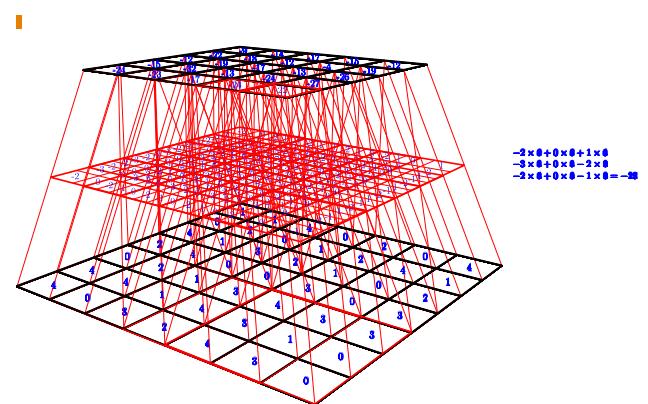
### Smoothness

- 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  $\gamma$  in SVMs (or  $\ell$  in Gaussian Processes)
  - ★  $k$  in  $k$ -nearest neighbours
  - ★ The size of the  $L_2$  regularisation term in neural networks
- We often have to choose these using trial and error (using a validation set)

### 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

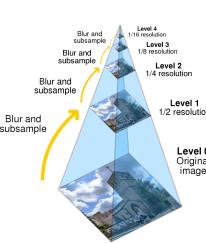
### Convolutional Neural Networks



- 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)

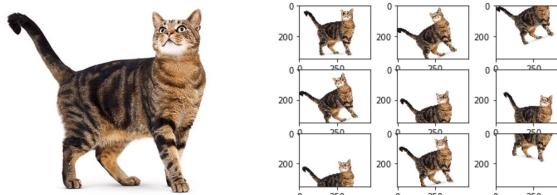
## Scale Symmetry

- 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



## Augmentations

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



- 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

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

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

- 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

## 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

## 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

## Outline

1. Inductive Bias
2. Invariance
3. Group Theory



- 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

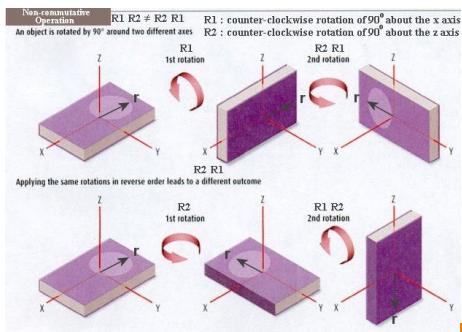
- Symmetries are very well studied in mathematics
- The language of symmetries are **groups**
- These capture the structure of transformations that represent some symmetry
- Group theory is highly abstract and it is easy to lose sight of its connection to symmetry, but it has proved itself extraordinarily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

## 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  $f^{-1} = f$

## Non-commutativity

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



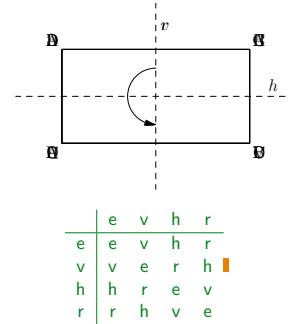
## Permutations

- 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.)

- Groups consists of a set (we can think of the elements as transformations),  $\mathcal{G} = \{a, b, \dots\}$
- And an operator “.” (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  $a^{-1} \cdot a = e$

## The Four Group

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



## 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

## 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

- 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 terribly interesting groups (or at least groups you are so familiar with that they are now boring)
- Things get more interesting in high dimensional space

- 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 infinitesimal transformations and the algebra between these infinitesimal 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  $\mathbf{M}^T \mathbf{M} = \mathbf{M} \mathbf{M}^T = \mathbf{I}$ ) 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

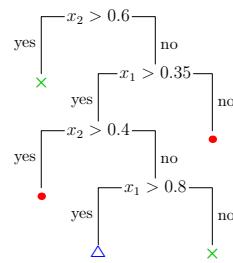
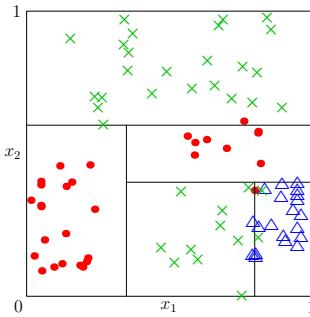
$$\begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix}$$

## 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

## 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)

## Decision Trees

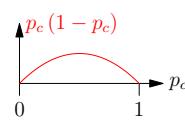
- A decision trees builds a binary tree to partition the data,  $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, \dots, m\}$ , into the leaves of the tree
- Each decision rule depends on a single feature
- At each step the rule is chosen that maximise the “*purity*” of the leaf nodes
- Decisions can be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

## Leaf Purity

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

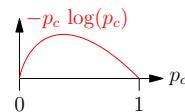
### Gini index

$$Q_m^g(\mathcal{L}) = \sum_{c \in \mathcal{C}} p_c(\mathcal{L})(1 - p_c(\mathcal{L}))$$

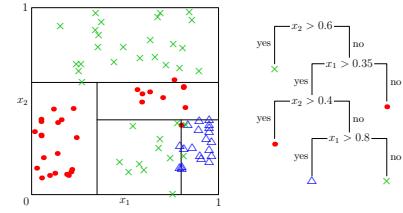


### Cross-entropy

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



- Decision Trees**
- Bagging**

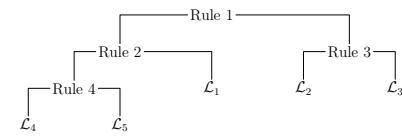


## Ensembling of Decision Trees

- 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

## Partitioning

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

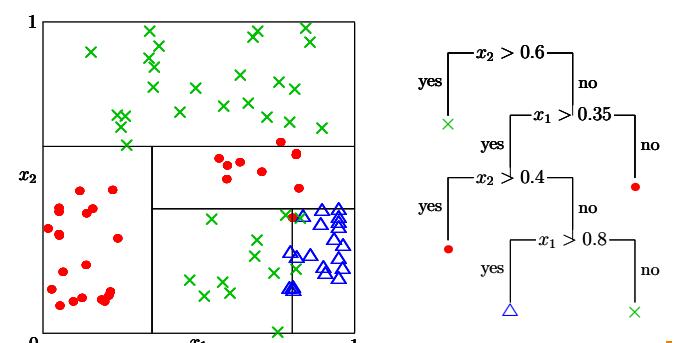


- The proportion of data points in leaf  $\mathcal{L}$  belonging to class  $c$  is

$$p_c(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{(\mathbf{x}, y) \in \mathcal{L}} \llbracket y = c \rrbracket$$

where  $\llbracket y = c \rrbracket = 1$  if  $y = c$  and 0 otherwise

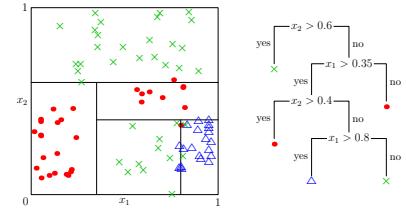
## Building Decision Trees



## 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 Classification And Regression Trees
- Decision trees depend strongly on the early decisions and so vary a lot for slightly different data sets—high variance

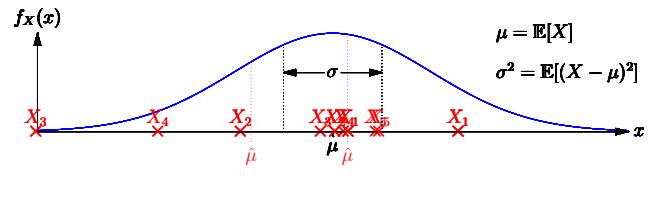
## Outline



- Decision Trees
- 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  $\mu = \mathbb{E}[X_i]$

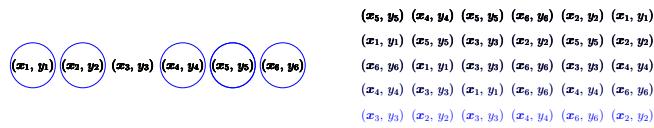
$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n} \sum_{i=1}^n X_i\right] = \frac{1}{n} \sum_{i=1}^n \mathbb{E}[X_i] = \frac{1}{n} \sum_{i=1}^n \mu = \mu$$

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

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

## 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 as bootstrapping



## Performance of Bagging

- Bootstrapping is an early form of data augmentation
- 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  $\rho$  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  $n \rightarrow \infty$  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

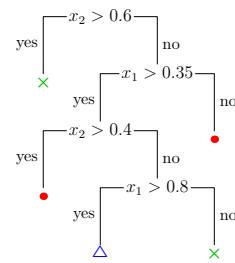
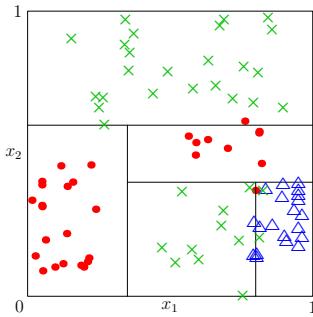
- In random forests we average much less correlated trees
- To do this for each tree we choose a subset of  $p' \ll p$  of the features on which to split the tree
- Typically  $p'$  can range from 1 to  $\sqrt{p}$
- The trees aren't that good, but are very decorrelated
- By averaging over a huge number 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

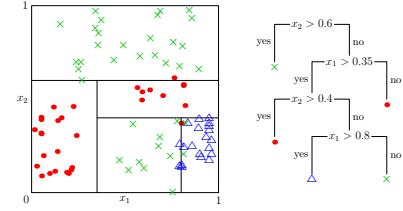
# Advanced Machine Learning

## Boosting



*Boosting, AdaBoost, Gradient Boosting*

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



## Boosting

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

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

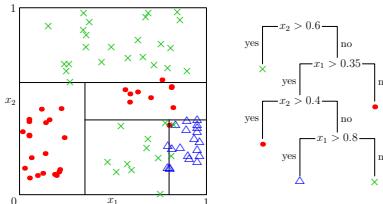
- Weak learners,  $\hat{h}_i(\mathbf{x})$ , are learning machine that do a little better than chance
- The trick is to choose the weights,  $\alpha_i$
- Because the weak learners do little better than chance we (miraculously) **don't overfit** that much

## Shallow Trees

- One of the most effective type of weak learner are very shallow trees
- Sometimes we just use one variable (the stump)
- 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**
4. **Dropout**



## Boosting a Binary Classifier

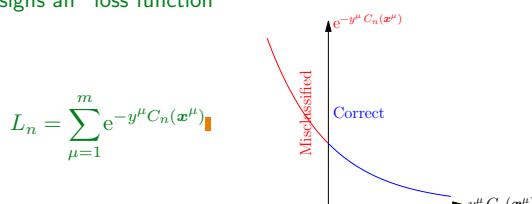
- Suppose we have a binary classification task with data  $\mathcal{D} = \{(\mathbf{x}^\mu, y^\mu) | \mu = 1, 2, \dots, m\}$  with  $y^\mu \in \{-1, 1\}$
- Our  $i^{th}$  weak learner provides a prediction  $\hat{h}_i(\mathbf{x}^\mu) \in \{-1, 1\}$
- We ask, can we find a linear combination

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

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

## AdaBoost

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



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

## Iterative Learning

- We build up a strong learner iteratively (greedily)

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

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

$$\begin{aligned} L_n(\alpha_n) &= \sum_{\mu=1}^m e^{-y^\mu C_n(\mathbf{x}^\mu)} = \sum_{\mu=1}^m e^{-y^\mu (C_{n-1}(\mathbf{x}^\mu) + \alpha_n \hat{h}_n(\mathbf{x}^\mu))} \\ &= \sum_{\mu=1}^m w_n^\mu e^{-\alpha_n y^\mu \hat{h}_n(\mathbf{x}^\mu)} = e^{\alpha_n} \sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu + e^{-\alpha_n} \sum_{\mu: y^\mu = \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu \\ &= e^{-\alpha_n} \sum_{\mu: y^\mu \neq \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu + (e^{\alpha_n} - e^{-\alpha_n}) \sum_{\mu: y^\mu = \hat{h}_n(\mathbf{x}^\mu)} w_n^\mu \end{aligned}$$

## Choosing a Weak Classifier

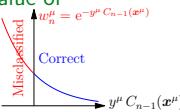
## Choosing Weights

- To minimise the loss

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

- We choose the weak learner with the lowest value of

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



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

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

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

- That is

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

## Algorithm

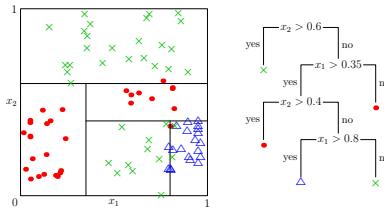
- Start with a set of weak learners  $\mathcal{W}$
- Associate a weight,  $w_n^\mu$ , with every data point  $(\mathbf{x}^\mu, y^\mu)$ ,  $\mu = 1, 2, \dots, m$
- Initially  $w_1^\mu = 1$  (large weight,  $w_n^\mu$ , means  $(\mathbf{x}^\mu, y^\mu)$  is poorly classified)
- Choose the weak learning,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

- Boosting
- AdaBoost
- Gradient Boosting**
- Dropout



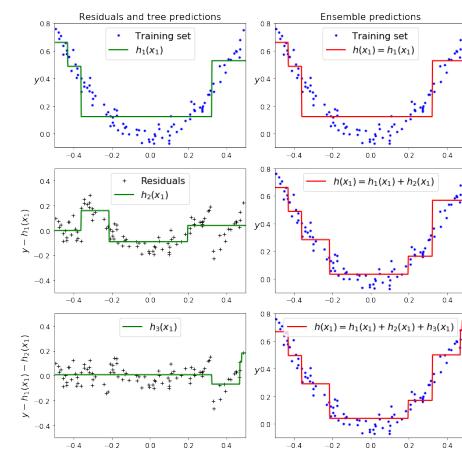
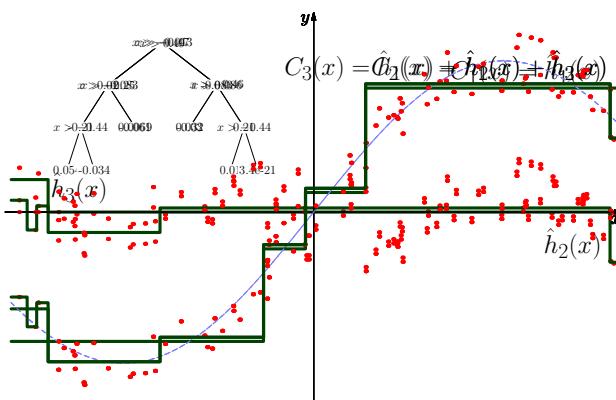
## Gradient Boosting

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

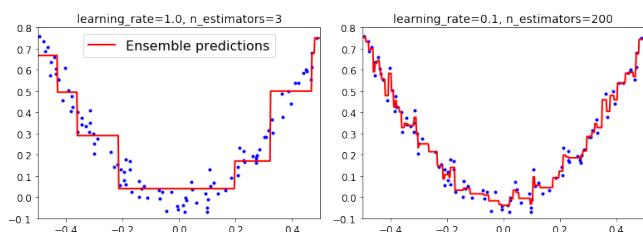
$$C_n(\mathbf{x}) = C_{n-1}(\mathbf{x}) + \hat{h}_n(\mathbf{x})$$

- Gradient boosting used on regression (again using decision trees)
- At each step  $\hat{h}_n(\mathbf{x})$  is trained to predict the **residual error**,  $\Delta_{n-1} = y - C_{n-1}(\mathbf{x})$ , (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

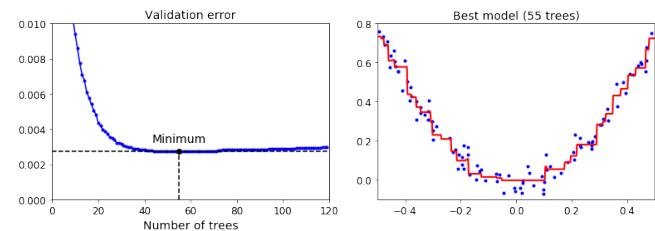


- We can keep on going



- But we will over-fit eventually

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



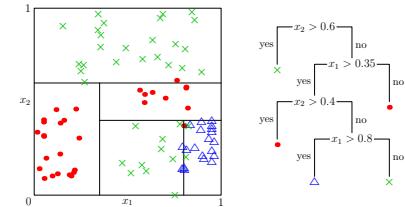
- Use cross-validation against a validation set to decide when to stop

## XGBoost

- XGBoost is an implementation of gradient boosting that won the Higgs's Boson challenge and regularly wins Kaggle competitions
- XGBoost stands for eXtreme Gradient Boosting
- 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
- It was much faster than most gradient boosting algorithms and scales to billions of training data points—although GBM is often better

## Outline

- Boosting
- AdaBoost
- Gradient Boosting
- Dropout

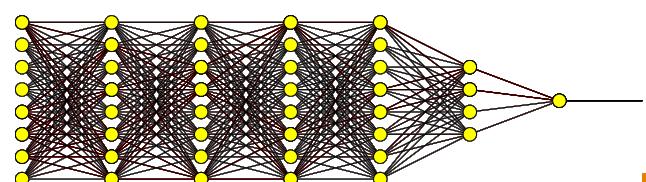


## Ensembling in Deep Learning

- For most machine learning ensembling different machines usually gives a reasonable improvement in performance
- The machines should have roughly the same performance
- Of course, this comes at the price of having to train multiple machines
- One can try to train a machine to decide how to combine different machines (stacking), but beware, it is very easy to overfit
- Usually better to average predictions for regression or do majority voting for classification problems

## Dropout

- For deep learning we can control for over-fitting using dropout



- This can be seen as ensembling lots of much simpler machines

## 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

**Vector Spaces**

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 & 16 \\ 1 & 3 & 9 & 27 & 81 \\ 1 & 4 & 16 & 64 & 256 \\ 1 & 5 & 25 & 125 & 625 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ a_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 2 \\ -3 \\ 1 \end{pmatrix}$$

$$Mx = b$$

$$MV_i = \lambda_i V_i$$

$$b = M^{-1}x$$

Vectors, metric spaces, norms

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

**Mx=b**

$$Mv_i = \lambda_i v_i$$

$$b = M^{-1}x$$

## 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 to use a slightly posher language than you are probably used to)

## Vectors

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

$$y^T = (y_1, y_2, \dots, y_n)$$

## Vector Space

- A vector space,  $\mathcal{V}$ , is a set of vectors which satisfies
  1. if  $v, w \in \mathcal{V}$  then  $av \in \mathcal{V}$  and  $v + w \in \mathcal{V}$  (closure)
  2.  $v + w = w + v$  (commutativity of addition)
  3.  $(u + v) + w = u + (v + w)$  (associativity of addition)
  4.  $v + 0 = v$  (existence of additive identity 0)
  5.  $1v = v$  (existence of multiplicative identity 1)
  6.  $a(bv) = (ab)v$  (distributive properties)
  7.  $a(v + w) = av + aw$
  8.  $(a + b)v = av + bv$

(You don't need to remember these)
- Just from these properties we can deduce other properties

## Scalars (Fields)

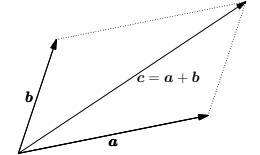
- Vector spaces involve **fields** (numbers)—aka **scalars**
- These are quantities we can add together ( $a + b$ ) and multiply together ( $a \times b$ )
- Formally they form an Abelian group under addition with an identity 0 and excluding 0 an Abelian group under multiplication and they are distributive

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

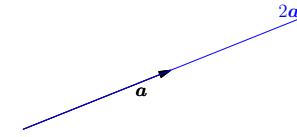
- Although this sounds rather daunting don't panic! They behave like numbers. The field might be integers, rational numbers, reals, complex numbers or something a bit more exotic—but we will almost always consider reals!

## Basic Vector Operations

- The basic vector operations are adding

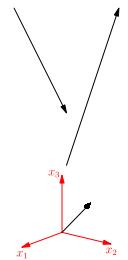


- multiplying by a scalar (a number)



$$\mathbb{R}^n$$

- When we first learn about vectors we think of them as 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  $x = (x_1, x_2, \dots, x_n)^T$  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



- 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

$Mx=b$

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

$Mv_i = \lambda_i v_i$

$b=M$

## Metrics

- Vector spaces become more interesting if we have a notion of distance
- We say  $d(x,y)$  is a **proper distance** or **metric** if
  1.  $d(x,y) \geq 0$  (non-negativity)
  2.  $d(x,y) = 0$  iff  $x = y$  (identity of indiscernibles)
  3.  $d(x,y) = d(y,x)$  (symmetry)
  4.  $d(x,y) \leq d(x,z) + d(z,y)$  (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.

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

( $f$  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  $y$ 

$$d(f(x), f(y)) \leq Kd(x, y)$$
- 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

$Mx=b$

$b=M$

## 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  $v$  as  $\|v\|$  satisfying
  1.  $\|v\| > 0$  if  $v \neq 0$  (non-negativity)
  2.  $\|av\| = |a|\|v\|$  (linearity)
  3.  $\|u + v\| \leq \|u\| + \|v\|$  (triangular inequality)
- When some criteria aren't satisfied we have a **pseudo-norms**
- Norms provide a metric  $d(x,y) = \|x - y\|$  (they are metric spaces)

## Vector Norms

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

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

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

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

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

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

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

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

## 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

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

(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 multiplying the input (which we can think of as a vector  $x$ ) 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  $x_n = \mathbf{L}_n x_{n-1}$
- We also do other things like applying ReLU's or pooling that changes the magnitude,  $x_n$ , of our representation
- If you are developing new architectures you want  $\|x_n\|$  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  $\mathcal{I}$

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

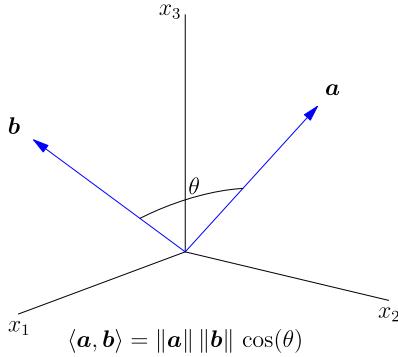
- The  $L_2$  vector space is the set of functions where  $\|f\|_{L_2} < \infty$
- The  $L_1$ -norm is given by  $\|f\|_{L_1} = \int_{x \in \mathcal{I}} |f(x)| dx$
- The infinite-norm is given by  $\|f\|_\infty = \max_{x \in \mathcal{I}} |f(x)|$

## Summary

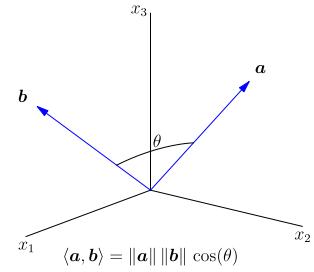
- Vector spaces with a distance (metric spaces) and vector spaces with a norm (normed vector spaces) are interesting objects
- They allow you to define a topology (open/closed sets, etc.)
- You can build up ideas about connectedness, continuity, contractive maps, fixed-point theorems, . . .
- For the 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



Inner products, operators



## 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,  $d(x, y)$ , allow us to construct ideas about geometry of the vector space
- Norms,  $\|x\|$ , that allow us to reason about the size of vector
- Norm induce a distance,  $d(x, y) = \|x - y\|$

## Inner Products

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

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

- For functions

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

- For  $m \times n$  matrices

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

## Cauchy-Schwarz Inequality

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

$$|\langle x, y \rangle| \leq \|x\| \|y\|$$

- Or

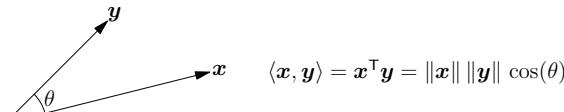
$$|\langle x, y \rangle|^2 \leq \langle x, x \rangle \langle y, y \rangle = \|x\|^2 \|y\|^2$$

- This is a very general result so for example

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

## Angles Between Vectors

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



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

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

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

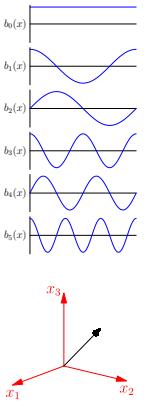
## Basis Functions

- Any set of vectors  $\{b_i | i = 1, \dots\}$  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.  $\|b_i\| = 1$ )
- In  $\mathbb{R}^3$  we could use  $b_1 = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ ,  $b_2 = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ ,  $b_3 = \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix}$
- This is not unique as we can rotate our basis vectors
- For an orthogonal basis we can write any vector as  $\hat{x} = \begin{pmatrix} x^T b_1 \\ x^T b_2 \\ x^T b_3 \end{pmatrix}$

## Orthogonal Functions

## Algebraic Structure

- 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  $f = \begin{pmatrix} \langle f(x), b_0(x) \rangle \\ \langle f(x), b_1(x) \rangle \\ \vdots \end{pmatrix}$
- There might be an infinite number of components
- This is analogous to points in  $\mathbb{R}^n$  (for large  $n$ )



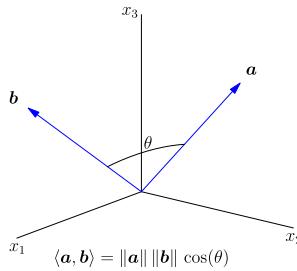
- We have gone to these lengths as we want to show that many properties of vectors are shared by other objects (matrices, functions, etc.)

- 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 repeatedly 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  $\mathcal{T} : \mathcal{V} \rightarrow \mathcal{V}'$
- This says that  $\mathcal{T}$  maps some object  $x \in \mathcal{V}$  to an object  $y = \mathcal{T}[x]$  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
  - $\mathcal{T}[ax] = a\mathcal{T}[x]$
  - $\mathcal{T}[x + y] = \mathcal{T}[x] + \mathcal{T}[y]$
- For normal vectors ( $x \in \mathbb{R}^n$ ) the most general linear operation is

$$\mathcal{T}[x] = Mx$$

where  $M$  is a matrix

## Matrix multiplication

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

$$C_{ij} = \sum_{k=1}^m A_{ik} B_{kj} \quad \left( \begin{array}{c|c|c} \hline & \cdots & \cdots \\ \hline \cdots & & \cdots \\ \hline k & & j \end{array} \right) \left( \begin{array}{c|c|c} \hline & \cdots & \cdots \\ \hline \cdots & & \cdots \\ \hline i & & \cdots \end{array} \right) = \left( \begin{array}{c|c|c} \hline & \cdots & \cdots \\ \hline \cdots & & \cdots \\ \hline i & & j \end{array} \right)$$

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

$$y = Ax \Rightarrow y_i = \sum_j M_{ij} x_j \quad \left( \begin{array}{c|c|c} \hline & \cdots & \cdots \\ \hline \cdots & & \cdots \\ \hline i & & \cdots \end{array} \right) \left( \begin{array}{c} \hline \cdots \\ \hline j \\ \hline \cdots \end{array} \right) = \left( \begin{array}{c} \hline \cdots \\ \hline i \\ \hline \cdots \end{array} \right)$$

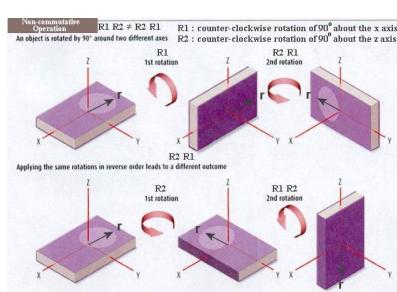
- Using the same matrix notation we can define the inner product as

$$\langle x, y \rangle = x^T y = \sum_{i=1}^n x_i y_i \quad \longleftrightarrow \left( \begin{array}{c} \hline \cdots \\ \hline i \\ \hline \cdots \end{array} \right) \left( \begin{array}{c} \hline \cdots \\ \hline j \\ \hline \cdots \end{array} \right) = \langle \cdots \rangle$$

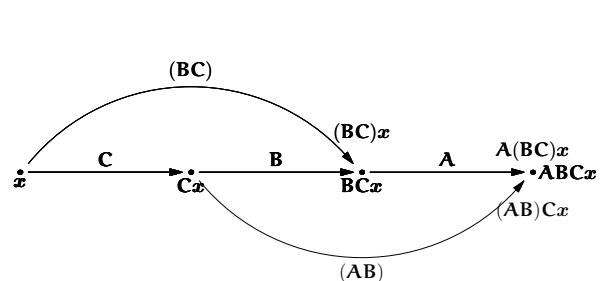
## Non-commutativity

- In general  $AB \neq BA$

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



## Associativity of Mappings



- For all  $x$  we have  $A(BC)x = (AB)Cx$
- This implies  $A(BC) = (AB)C$

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

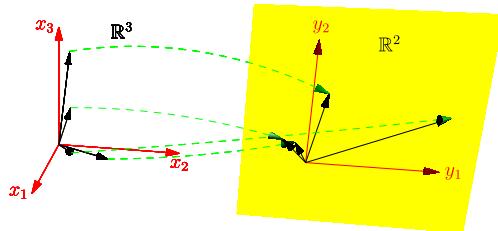
$$g(x) = \mathcal{T}[f] = \int_{y \in \mathcal{X}} K(x,y) f(y) dy$$

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

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

- We shall soon see examples of high-dimensional kernels

## General Linear Mappings



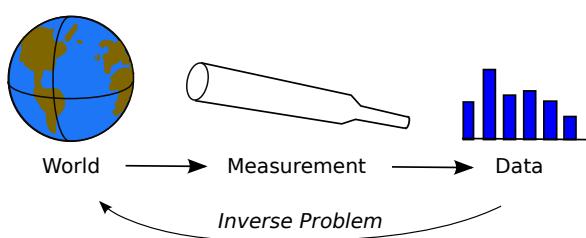
- In general a linear operator will map vectors between different vector spaces
- E.g.  $\mathbb{R}^3 \rightarrow \mathbb{R}^2$

- We will spend a lot of time on operators that map from a vector space onto itself  $\mathcal{T} : \mathcal{V} \rightarrow \mathcal{V}$
- 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

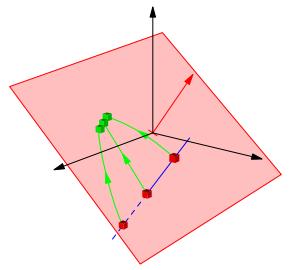
## Summary

- 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

### Understand Mappings



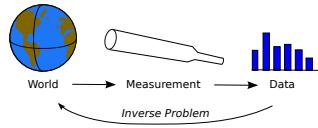
1. Mappings
2. Linear Maps



Mappings, Linear Maps, Solving Linear Systems

### Transforming Data

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



### Inverse Problems

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

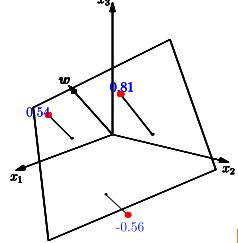
- Given  $m$  observations  $\{(\mathbf{x}_k, y_k) | k = 1, \dots, m\}$  and  $p$  unknown  $\mathbf{w} = (w_1, w_2, \dots, w_p)$  such that  $\mathbf{x}_k^\top \mathbf{w} = y_k$  then to find  $\mathbf{w}$
- Define the *design matrix* as the matrix of feature vectors

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

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

### Linear Regression

- $\mathbf{x}_k^\top \mathbf{w}$  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  $\mathbf{y} \approx \mathbf{X}\mathbf{w} \Rightarrow$  no "solution"
- Problem solved by Gauss to predict the orbit of the asteroid Ceres

### Linear Least Squares

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

$$\epsilon_k = \mathbf{x}_k^\top \mathbf{w} - y_k$$

- The squared error

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

- We can define the error vector

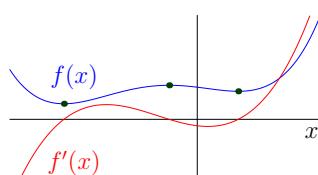
$$\boldsymbol{\epsilon} = \mathbf{X}\mathbf{w} - \mathbf{y}$$

(note that  $\epsilon_k = \mathbf{x}_k^\top \mathbf{w} - y_k$ )

- 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  $n$ -dimensional function  $f(\mathbf{x})$  are given by the set of equations

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

### Gradients

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

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

- The partial derivatives (curly d's)

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

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

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

- Or

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

$\mathbf{X}^+ = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$  is known as the pseudo inverse

- For non-square matrices Matlab uses the pseudo inverse so in Matlab we can write

$$\mathbf{w} = \mathbf{X} \setminus \mathbf{y}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

9

## Computing Gradients

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

$$\begin{aligned}\nabla \mathbf{w}^T \mathbf{M} \mathbf{w} &= \left( \frac{\partial}{\partial w_1} \frac{\partial}{\partial w_2} \frac{\partial}{\partial w_3} \vdots \right) \sum_{i,j} w_i M_{ij} w_j = \left( \begin{array}{c} \sum_j M_{1j} w_j + \sum_i w_i M_{i1} \\ \sum_j M_{2j} w_j + \sum_i w_i M_{i2} \\ \sum_j M_{3j} w_j + \sum_i w_i M_{i3} \\ \vdots \end{array} \right) \\ &= \mathbf{M} \mathbf{w} + \mathbf{M}^T \mathbf{w}\end{aligned}$$

- 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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

11

## 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

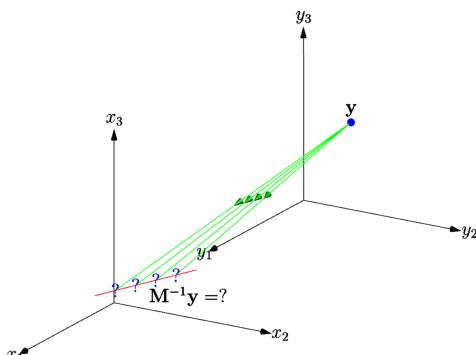
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

13

## What is the Inverse?

- Many points can map to the same points



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

15

## Missing Bits of the Mathematics

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

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

- Where we have used  $\mathbf{w}^T \mathbf{X}^T \mathbf{y} = \mathbf{y}^T \mathbf{X} \mathbf{w}$   $\sum_{i,j} w_i X_{ji} y_j = \sum_{i,j} y_i X_{ij} w_j$

- Also  $\nabla \mathbf{w}^T \mathbf{M} \mathbf{w} = \mathbf{M} \mathbf{w} + \mathbf{M}^T \mathbf{w}$

- If  $\mathbf{M} = \mathbf{M}^T$  (i.e.  $\mathbf{M}$  is symmetric) then  $\nabla \mathbf{w}^T \mathbf{M} \mathbf{w} = 2\mathbf{M} \mathbf{w}$

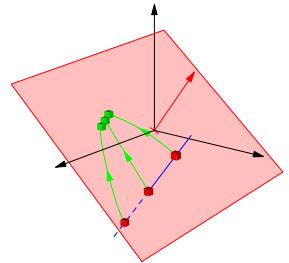
- $(\mathbf{X}^T \mathbf{X})^T = \mathbf{X}^T \mathbf{X}$  so that  $\mathbf{X}^T \mathbf{X}$  is symmetric

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

10

## Outline



- Mappings
- Linear Maps

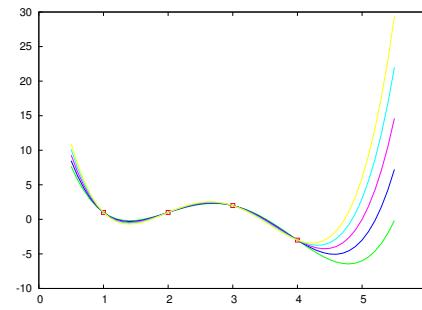
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

12

## Under Constrained Systems

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



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

14

## Under-constrained Systems

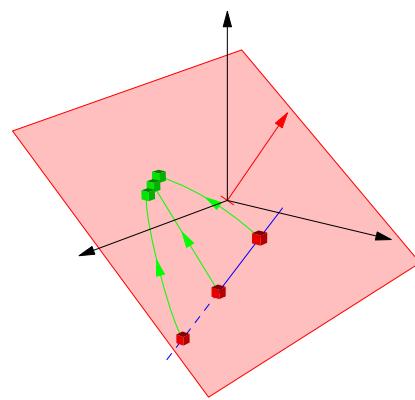
- The system is **under-constrained**
- We have more unknowns than equations
- The inverse is not unique
- Solving the inverse problem ( $\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$ ) is said to be **ill-posed**
- The inverse  $(\mathbf{X}^T \mathbf{X})^{-1}$  doesn't exist
- 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)

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

16

- 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



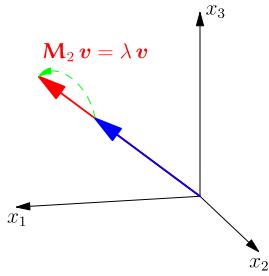
### III-Conditioning in ML

- 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  $X^T X$  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

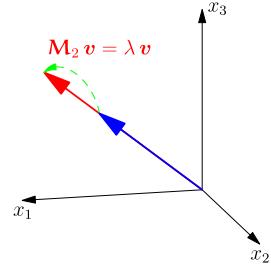
### Summary

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

### Eigen systems



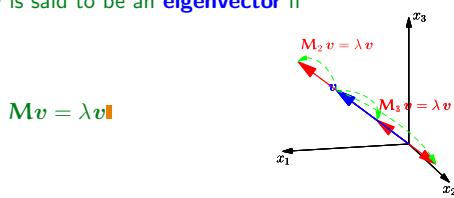
Eigenvectors, Orthogonal Matrices, Eigenvector Decomposition, Rank



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

### Eigenvector equation

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



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

### Symmetric Matrices

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

$$Mv_i = \lambda_i v_i$$

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

$$v_i^\top v_j = 0$$

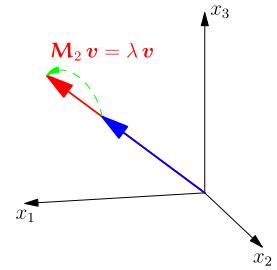
- (We can always normalise eigenvectors if we want)

### Proof of Orthogonality

- $(Mv_i = \lambda_i v_i)^\top$  implies  $v_i^\top M^\top = \lambda_i v_i^\top$
  - When  $M$  is symmetric then  $Mv_i = \lambda_i v_i \Rightarrow v_i^\top M = \lambda_i v_i^\top$
  - Consider two eigenvectors  $v_i$  and  $v_j$  of  $M$
- $$\begin{aligned} v_i^\top M v_j &= (v_i^\top M) v_j = \lambda_i v_i^\top v_j \\ &= v_i^\top (M v_j) = \lambda_j v_i^\top v_j \end{aligned}$$
- So either  $\lambda_i = \lambda_j$  or  $v_i^\top v_j = 0$
  - If  $\lambda_i = \lambda_j$  then any linear combination of  $v_i$  and  $v_j$  is an eigenvector ( $M(av_i + bv_j) = \lambda_i(av_i + bv_j)$ ). 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  $V$  from the eigenvectors

$$V = (v_1, v_2, \dots, v_n)$$

- Matrix  $V$  is an  $n \times n$  matrix
- Because of the orthogonality of the vectors  $v_i$

$$V^\top V = \begin{pmatrix} v_1^\top v_1 & v_1^\top v_2 & \dots & v_1^\top v_n \\ v_2^\top v_1 & v_2^\top v_2 & \dots & v_2^\top v_n \\ \vdots & \vdots & \ddots & \vdots \\ v_n^\top v_1 & v_n^\top v_2 & \dots & v_n^\top v_n \end{pmatrix} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 \end{pmatrix} = I$$

### The Other Way Around

- We have shown that  $V^\top V = I$
- Thus multiply both sides on the left by  $V$

$$VV^\top V = V$$

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

$$\begin{aligned} (VV^\top)VV^{-1} &= VV^{-1} \\ VV^\top &= I \end{aligned}$$

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

## Invertible Matrices

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

$$Mx = 0$$

- Now if there exists such a vector such that  $Vx = 0$  then multiply by  $V^T$  we get

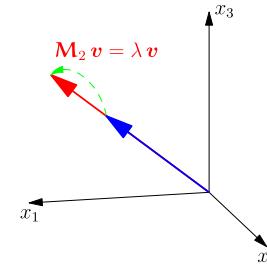
$$V^T V x = V^T 0$$

$$x = 0$$

since  $V^T V = I$

- Thus  $V$  is invertible

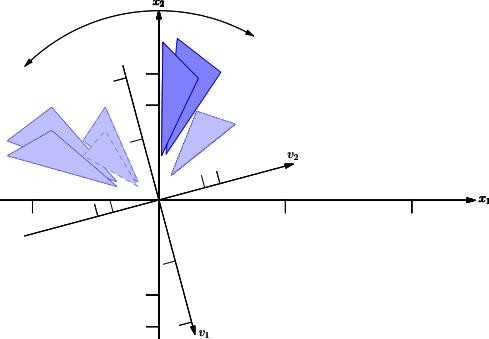
- Eigenvectors
- Orthogonal Matrices
- Eigen Decomposition**
- Low Rank Approximation



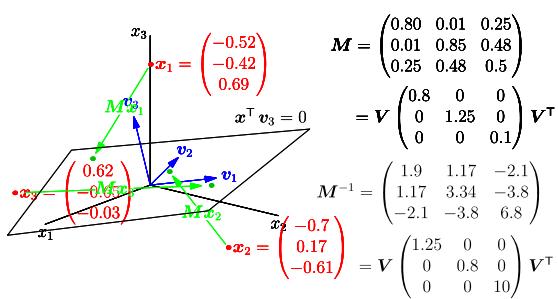
## Outline

## Mappings by Symmetric Matrices

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



## III-Conditioning Again



## Rotations

- Orthogonal matrices satisfy  $V^T V = V V^T = I$

- As a consequence they define rotations (and possibly a reflection)

- Consider a vector  $x$  and  $x' = Vx$ , now

$$\|x'\|_2^2 = x'^T x' = (Vx)^T (Vx) = x^T V^T V x = x^T x = \|x\|_2^2$$

- Similarly if additionally  $y' = Vy$  then

$$\langle x', y' \rangle = (Vx)^T (Vy) = x^T V^T V y = x^T y = \langle x, y \rangle = \|x\|_2 \|y\|_2 \cos(\theta)$$

- Rotations and reflections preserve lengths and angles

## Matrix Decomposition

- Taking the matrix of eigenvectors,  $V$ , then

$$MV = M(v_1, v_2, \dots, v_n) = (\lambda_1 v_1, \lambda_2 v_2, \dots, \lambda_n v_n) = V\Lambda$$

- where  $\Lambda = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_n) = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$

- Now

$$M = MVV^T = V\Lambda V^T$$

- Very important similarity transform

## Inverses

- For any square matrix

$$M = V\Lambda V^T \quad M^{-1} = V\Lambda^{-1}V^T$$

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

- Since

$$MM^{-1} = (V\Lambda V^T)(V\Lambda^{-1}V^T) = V\Lambda(V^T V)\Lambda^{-1}V^T = V\Lambda\Lambda^{-1}V^T = VV^T = I$$

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

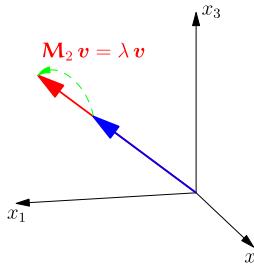
## Condition Number

- 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

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

- Large condition number implies very ill-conditioned

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



- The rank of a matrix,  $M$ , is the number of non-zero eigenvalues
- The space spanned by the eigenvectors  $v_a, v_b$ , etc. with zero eigenvalue forms a **null space**
- Any vector in the null space will get projected to the zero vector
 
$$M(av_a + bv_b + \dots) = 0$$
- 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  $x$  such that  $Mx = b$ ) as we don't know the component of the  $x$  in the null space
- Although we don't know  $x$  we can find a vector,  $x$ , that satisfies  $Mx = b$
- Given a symmetric  $n \times n$  matrix with  $k$  non-zero eigenvalues  $\lambda_1, \lambda_2, \dots, \lambda_k$  we can construct a "pseudo inverse"  $M^+$  as  $V\Lambda^+V^T$  where  $\Lambda^+ = \text{diag}(\frac{1}{\lambda_1}, \frac{1}{\lambda_2}, \dots, \frac{1}{\lambda_k}, 0, \dots, 0)$
- This finds the vector  $x$  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

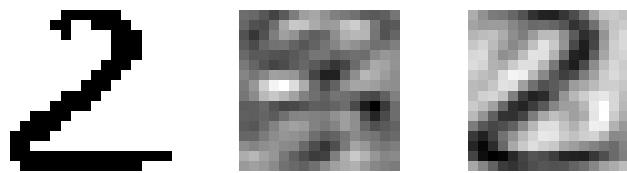
- Recall that matrices with large and small eigenvectors are ill-conditioned so the inverse has the potential to greatly amplify any measurement error
- One work around is to set all small eigenvalues to zero and use the **pseudo inverse**
- Setting small eigenvalues to zero reduces the rank of the matrix 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

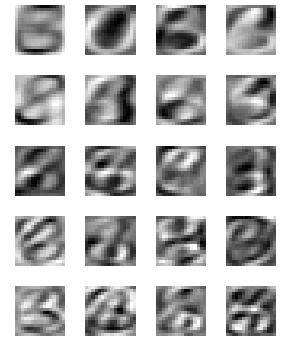
- Linear mappings are commonly used in machine learning algorithms such as regression
- We can understand symmetric operators by looking at their eigenvectors
- Any symmetric matrix can be decomposed as  $M = V\Lambda V^T$ 
  - ★ where  $V$  are orthogonal matrices whose rows are the eigenvectors
  -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

### 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



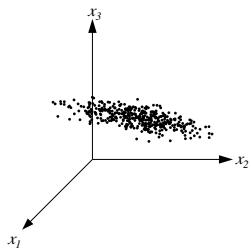
Adam Prügel-Bennett COMP6208 Advanced Machine Learning 1

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

Adam Prügel-Bennett COMP6208 Advanced Machine Learning 2

### Spread of Data

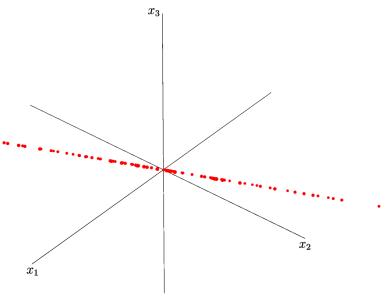
- Often data varies significantly in only some directions



- 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

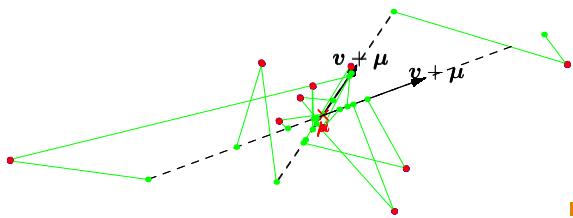


Adam Prügel-Bennett COMP6208 Advanced Machine Learning 3

Adam Prügel-Bennett COMP6208 Advanced Machine Learning 4

### Dimensionality Reduction

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



Adam Prügel-Bennett COMP6208 Advanced Machine Learning 5

### Direction of Maximum Variation

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

$$\sigma^2 = \frac{1}{m-1} \sum_{i=1}^m (v^\top (x_i - \mu))^2$$

- This is a constrained optimisation problem

- Solve by maximising Lagrangian

$$\mathcal{L} = \frac{1}{m-1} \sum_{k=1}^m (v^\top (x_k - \mu))^2 - \lambda (\|v\|^2 - 1)$$

- $\lambda$  is a Lagrange multiplier

Adam Prügel-Bennett COMP6208 Advanced Machine Learning 6

### Direction of Maximum Variation

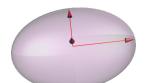
- Expanding the Lagrangian

$$\begin{aligned} \mathcal{L} &= \frac{1}{m-1} \sum_{k=1}^m (v^\top (x_k - \mu))^2 - \lambda (\|v\|^2 - 1) \\ &= \frac{1}{m-1} \sum_{k=1}^m (v^\top (x_k - \mu)(x_k - \mu)^\top v) - \lambda (\|v\|^2 - 1) \\ &= v^\top \left( \frac{1}{m-1} \sum_{k=1}^m (x_k - \mu)(x_k - \mu)^\top \right) v - \lambda (\|v\|^2 - 1) \\ &= v^\top C v - \lambda (v^\top v - 1) \end{aligned}$$

- Extrema of the Lagrangian

$$\nabla \mathcal{L} = 2(Cv - \lambda v) = 0 \Rightarrow Cv = \lambda v$$

- The eigenvectors are directions that are extrema of the variance



- The variance in direction  $v$  is equal to

$$\begin{aligned} \sigma^2 &= \frac{1}{m-1} \sum_{i=1}^m (v^\top (x_i - \mu))^2 \\ &= v^\top C v = \lambda v^\top v = \lambda \end{aligned}$$

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

Adam Prügel-Bennett COMP6208 Advanced Machine Learning 7

Adam Prügel-Bennett COMP6208 Advanced Machine Learning 8

## Covariance Matrix

## Outer Product

- The covariance matrix is defined as

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

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

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

- C.f. covariance of random variables

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

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

$$\mathbf{x}\mathbf{y}^\top = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} (y_1 \ y_2 \ \cdots \ y_n) = \begin{pmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n y_1 & x_n y_2 & \cdots & x_n y_n \end{pmatrix}$$

- C.f. Inner product

$$\mathbf{x}^\top \mathbf{y} = (x_1 \ x_2 \ \cdots \ x_n) \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$$

## Matrix Form

## Properties of Covariance Matrix

- The covariance matrix is

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

- Define the matrix

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

- We can write the covariance matrix as

$$\mathbf{C} = \mathbf{X}\mathbf{X}^\top$$

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

$$\mathbf{v}^\top \mathbf{M} \mathbf{v}$$

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

$$\mathbf{v}^\top \mathbf{C} \mathbf{v} = \mathbf{v}^\top \mathbf{X} \mathbf{X}^\top \mathbf{v} = \mathbf{u}^\top \mathbf{u} = \|\mathbf{u}\|^2 \geq 0$$

where  $\mathbf{u} = \mathbf{X}^\top \mathbf{v}$

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

## Eigenvalue Decomposition

## Surface Defined by Matrix

- 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  $\mathbf{v}$  satisfies the equation

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

- Multiplying both sides by  $\mathbf{v}^\top$

$$\mathbf{v}^\top \mathbf{C} \mathbf{v} = \lambda \mathbf{v}^\top \mathbf{v} = \lambda \|\mathbf{v}\|^2$$

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

$$\lambda = \frac{\mathbf{v}^\top \mathbf{C} \mathbf{v}}{\|\mathbf{v}\|^2} \geq 0$$

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

$$\mathbf{x}^\top \mathbf{C}^{-1} \mathbf{x} = 1$$

defines a surface

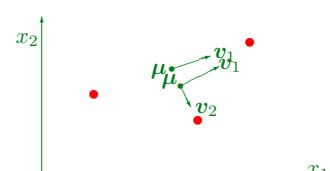
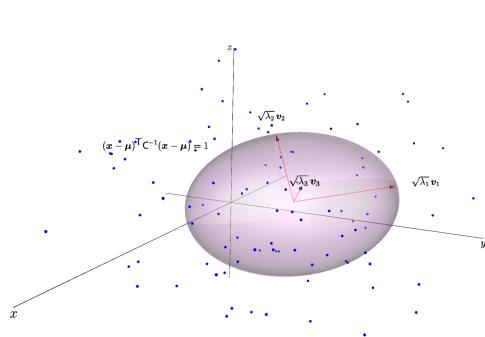
- The surface is an ellipsoid,  $\mathcal{E}$

- 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

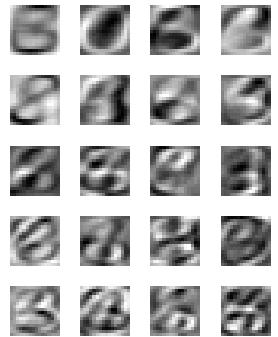
## Ellipsoid and Eigen Space

## 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

- 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



- Covariance Matrices
- Principal Component Analysis
- 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

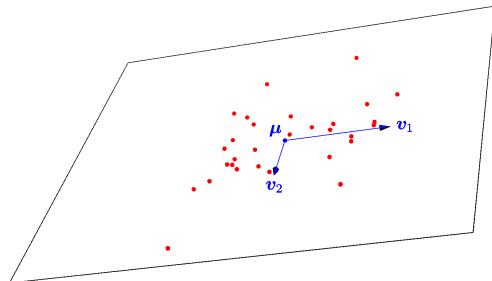
$$\mathbf{P} = \begin{pmatrix} \mathbf{v}_1^\top \\ \mathbf{v}_2^\top \\ \vdots \\ \mathbf{v}_k^\top \end{pmatrix}$$

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

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

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

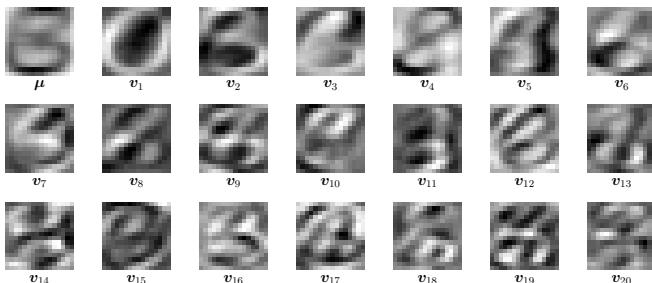
## Subspace Projection



## Hand Written Digits

99999999949999999990  
88888888888888888889  
77777777777777777777  
6666666666666666666666  
5555555555555555555555  
4444444444444444444444  
3333333333333333333333  
2222222222222222222222  
1111111111111111111111  
0000000000000000000000

## Eigenvectors



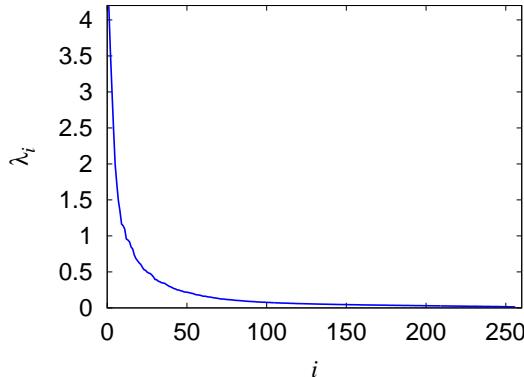
## Reconstruction

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

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

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

## Eigenvalues for Digits

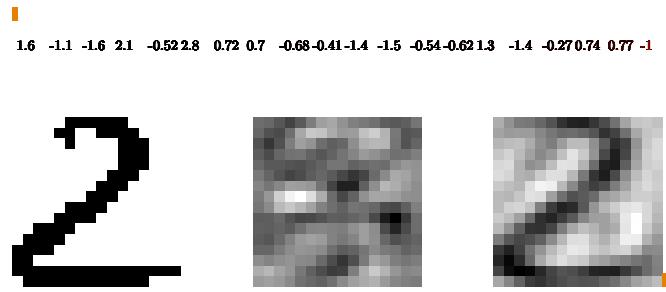


Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

25

## Reconstruction from Eigenvectors



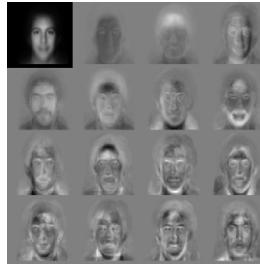
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

26

## 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  $4 \times 10^9$  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  $m \ll p$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

27

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

28

## Dual Matrix

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

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

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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

29

Adam Prügel-Bennett

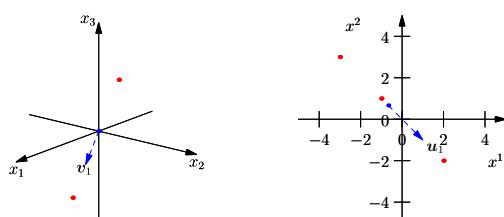
COMP6208 Advanced Machine Learning

30

## What Does a Subspace Look Like?

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

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



## 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 ( $\mathbf{X}\mathbf{X}^T$ ) or the dual space ( $\mathbf{X}^T\mathbf{X}$ )
- 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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

31

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

32

**Singular Value Decomposition (SVD)**

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

1. **Singular Value Decomposition**
2. General Linear Mappings
3. Linear Regression Revisited

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

*Singular Valued Decomposition, SVD, general linear maps*

**Singular Valued Decomposition**

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

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

$\begin{pmatrix} u \\ v \end{pmatrix}$  is an eigenvector of  $B$  with eigenvalue  $s$

- We observe that

$$\begin{aligned} Xv &= su \\ X^T Xv &= s X^T u = s^2 v \\ X X^T u &= s X v = s^2 u \end{aligned}$$

- Note that as  $Xv = su$  and  $X^T u = sv$  then

$$X(-v) = (-s)u \quad X^T u = (-s)(-v)$$

if  $\begin{pmatrix} u \\ v \end{pmatrix}$  is an eigenvector of  $B$  with eigenvalue  $s$  then so is  $\begin{pmatrix} u \\ -v \end{pmatrix}$  with eigenvalue  $-s$

- If  $n < m$  then  $X^T X$  is not full rank so some eigenvalues are zero
- As a consequence  $m - n$  vectors exist such that  $Xv = 0$
- The eigenvalues and eigenvectors are

$$n \times \left( s_i, \begin{pmatrix} u_i \\ v_i \end{pmatrix} \right) \quad n \times \left( -s_i, \begin{pmatrix} u_i \\ -v_i \end{pmatrix} \right) \quad m - n \times \left( 0, \begin{pmatrix} 0 \\ v_k \end{pmatrix} \right)$$

**Matrix Decomposition**

- Stacking the eigenvectors into a matrix

$$\begin{pmatrix} 0 & X \\ X^T & 0 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} u & u & 0 \\ v & -v & v_0 \end{pmatrix} \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} u^T & v^T \end{pmatrix}$$

- Since the vectors  $\begin{pmatrix} u_i \\ v_i \end{pmatrix}$  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

$$\begin{pmatrix} 0 & X \\ X^T & 0 \end{pmatrix} = \begin{pmatrix} u & u & 0 \\ v & -v & v_0 \end{pmatrix} \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} u^T & v^T \\ u^T & -v^T \\ 0 & v_0^T \end{pmatrix}$$

**SVD**

- Any matrix,  $X$ , can be written as  $X = USV^T$ 
  - $U, V$  are orthogonal matrices
  - $S = \text{diag}(s_1, s_2, \dots, s_n)$
- $s_i$  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

**Normalisation Subtlety**

$$\begin{pmatrix} 0 & X \\ X^T & 0 \end{pmatrix} = \begin{pmatrix} u & u & 0 \\ v & -v & v_0 \end{pmatrix} \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} u^T & v^T \\ u^T & -v^T \\ 0 & v_0^T \end{pmatrix}$$

- Multiplying out we have

$$X = 2USV^T \quad X^T = 2VSU^T$$

- Now the vectors  $u_i$  and  $v_i$  form an orthogonal set as it satisfy

$$X^T X v = s^2 v \quad X X^T u = s^2 u$$

- But they are not normalised (since  $\begin{pmatrix} u_i \\ v_i \end{pmatrix}$  is normalised). If we define  $\tilde{U} = \sqrt{2}U$  and  $\tilde{V} = \sqrt{2}V$  we find

$$X = \tilde{U}S\tilde{V}^T \quad X^T = \tilde{V}S\tilde{U}^T$$

**Finding SVD**

- Most libraries will compute the SVD for you
- They can do this by choosing the smaller of two matrices  $XX^T$  and  $X^T X$  and then compute the eigenvalues
- The singular values are the square root of the eigenvalues (notice that  $XX^T$  and  $X^T X$  are both positive semi-definite so the eigenvalues will be non-negative)
- It can compute the  $U$  matrix or  $V$  matrix by multiplying through by  $X$  or  $X^T$  ( $U = XVS^{-1}$  and  $V = X^TUS^{-1}$ )
- 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

$$\begin{pmatrix} \mathbf{X} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{U} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{S} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{V}^T \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{X} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{U} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{S} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{V}^T \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{X} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{U} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{S} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{V}^T \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{X} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{U} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{S} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{V}^T \\ \vdots \end{pmatrix}$$

- In Matlab these are obtained using

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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

9

## Outline

- Singular Value Decomposition
- General Linear Mappings
- Linear Regression Revisited

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

## General Matrix

- Recall that we can compute the SVD for any matrix,  $\mathbf{X}$

- As matrices describe the most general linear mapping

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

- 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

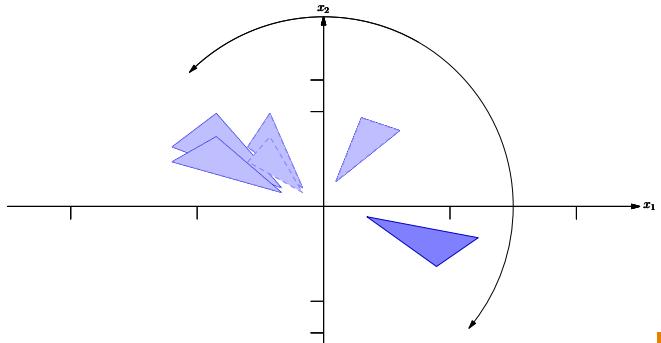
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

11

## Matrices

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



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

12

## 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

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

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

- Thus

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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

13

## 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

$$\begin{pmatrix} \mathbf{X} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{U} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{S} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{V}^T \\ \vdots \end{pmatrix}$$

$$\begin{pmatrix} \mathbf{X} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{U} \\ \vdots \end{pmatrix} \begin{pmatrix} \mathbf{S} \\ 0 \end{pmatrix} \begin{pmatrix} \mathbf{V}^T \\ \vdots \end{pmatrix}$$

- 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  $\mathbf{X} = \mathbf{USV}^T$  then

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

- If  $\mathbf{X}$  is an  $p \times m$  matrix then  $\mathbf{SS}^T$  is a  $p \times p$  diagonal matrix with elements  $S_{ii}^2 = s_i^2$

- $\mathbf{S}^T \mathbf{S}$  is an  $m \times m$  matrix with elements  $S_{ii}^2 = s_i^2$

- $\mathbf{U}$  and  $\mathbf{V}$  are matrices of eigenvectors for  $\mathbf{C}$  and  $\mathbf{D}$

- The eigenvalues are  $\lambda_i = S_{ii}^2 = s_i^2$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

15

## $\mathbf{SS}^T$ and $\mathbf{S}^T \mathbf{S}$

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

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

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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

16

- 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!

- Singular Value Decomposition
- General Linear Mappings
- Linear Regression Revisited

## Linear Regression

## Matrix Form

- Given a set of data  $\mathcal{D} = \{(\mathbf{x}_i, y_i) | i = 1, 2, \dots, m\}$

- In linear regression we try to fit a linear model

$$f(\mathbf{x}|\mathbf{w}) = \mathbf{x}^\top \mathbf{w}$$

- Which we fit by minimising the squared error loss

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

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

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_m^\top \end{pmatrix} \quad \mathbf{y} = \begin{pmatrix} y_1 \\ y_2, \\ \vdots \\ y_m \end{pmatrix}$$

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

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y} = \mathbf{X}^+ \mathbf{y}$$

- This is known as the pseudo-inverse

## Using SVD

## Pseudo-Inverse of S

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

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

- If  $m > p$

$$\mathbf{X}^\top = \begin{pmatrix} \mathbf{U}^\top \\ \mathbf{U}^\top \\ \mathbf{U}^\top \\ \mathbf{U}^\top \\ \mathbf{U}^\top \\ \mathbf{U}^\top \end{pmatrix}, \mathbf{S}^\top = \begin{pmatrix} s_1 & 0 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & s_2 & 0 & \cdots & 0 & 0 & \cdots & 0 \\ 0 & 0 & s_3 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & s_p & 0 & 0 & \cdots 0 \end{pmatrix}$$

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

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

## III-Conditioned Data Matrix

## Regularisation

- Recall that

$$\mathbf{w}^* = \mathbf{X}^+ \mathbf{y} = \mathbf{V}\mathbf{S}^+ \mathbf{U}^\top \mathbf{y}$$

- 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  $\mathbf{y}$  will be magnified

- This leads to poor weights

- Consider linear regression with a regulariser

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

- Thus

$$\nabla \mathcal{L}(\mathbf{w}) = 2(\mathbf{X}^\top \mathbf{X} + \eta \mathbf{I}) \mathbf{w} - 2\mathbf{X}^\top \mathbf{y}$$

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

$$\mathbf{w}^* = (\mathbf{X}^\top \mathbf{X} + \eta \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y}$$

- Using  $\mathbf{X} = \mathbf{USV}^T$

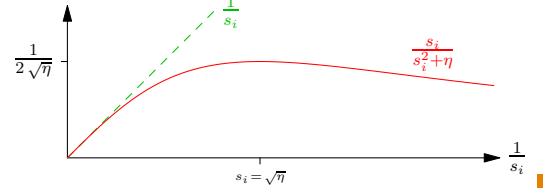
$$\begin{aligned}\mathbf{w}^* &= (\mathbf{X}^T \mathbf{X} + \eta \mathbf{I})^{-1} \mathbf{X}^T \mathbf{y} \\ &= \mathbf{V} (\mathbf{S}^T \mathbf{S} + \eta \mathbf{I})^{-1} \mathbf{S}^T \mathbf{U}^T \mathbf{y}\end{aligned}$$

- where

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

- 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

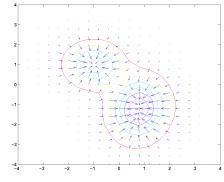
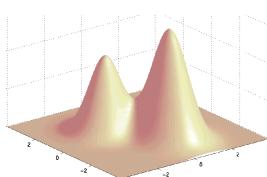
- Using  $\hat{\mathbf{S}}^+ = (\mathbf{S}^T \mathbf{S} + \eta \mathbf{I})^{-1} \mathbf{S}^T$  instead of  $\mathbf{S}^+$  then



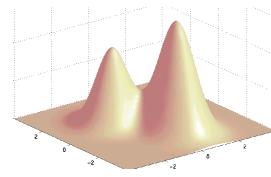
- Regularisation makes the machine much more stable (reduces the variance)

## Summary

- Any matrix can be decomposed as  $\mathbf{X} = \mathbf{USV}^T$  where
  - $\mathbf{U}$  and  $\mathbf{V}$  are orthogonal (rotation matrices)
  - $\mathbf{S} = \text{diag}(s_1, \dots, s_n)$  is a diagonal matrix of positive singular values
- This describes the most general linear transform
- The transform exploits the duality between  $\mathbf{X}\mathbf{X}^T$  and  $\mathbf{X}^T\mathbf{X}$
- 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



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



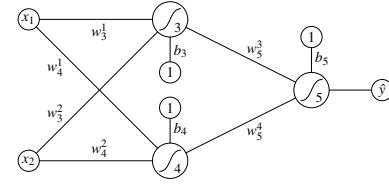
$$z = e^{-(x+1)^2 - (y-1)^2} + 0.6e^{-(x-1)^2 - 0.5(y+1)^2 + 0.1(x-3)(y-3)}$$

Gradient descent, quadratic minima, differing length scales

## ML = Optimisation

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

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



- Stands for the function ( $\hat{y} = f(\mathbf{x}|\mathbf{w})$ )

$$\hat{y} = g(w_5^3 g(w_3^1 x_1 + w_3^2 x_2 + b_3) + w_5^4 g(w_4^1 x_1 + w_4^2 x_2 + b_4) + b_5)$$

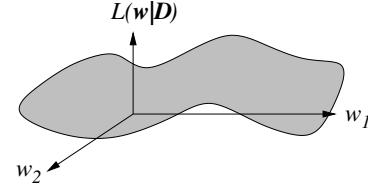
where, for example,  $g(V) = \frac{1}{1+e^{-V}}$

## Training

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

## Computing Gradients

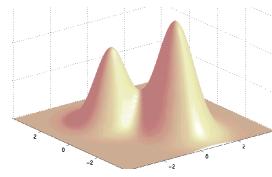
- $L(\mathbf{w}|\mathcal{D})$  is a complex function of the weights  $\mathbf{w}$



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

## Outline

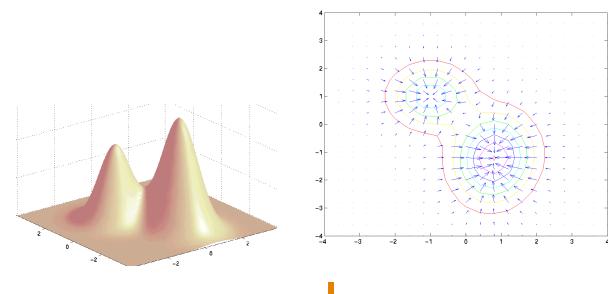
1. Motivation
2. Gradient Descent
3. Why Gradient Descent is Difficult



## Gradient Optimisation

- A maximum or minimum occurs when  $\nabla L(\mathbf{w}|\mathcal{D}) = \mathbf{0}$
- E.g.

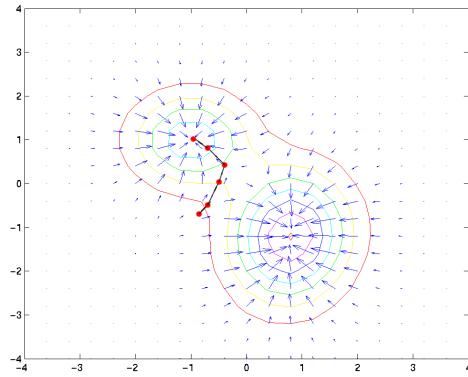
$$L = e^{-(x+1)^2 - (y-1)^2} + 0.6e^{-(x-1)^2 - 0.5(y+1)^2 + 0.1(x-3)(y-3)}$$



## Gradient Descent

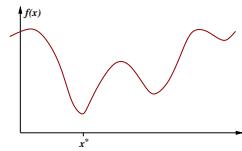
## Hill-Climbing

- For a simple function  $L(\mathbf{w}|\mathcal{D})$  we can solve  $\nabla L(\mathbf{w}|\mathcal{D}) = \mathbf{0}$  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:  $\mathbf{w} \rightarrow \mathbf{w}' = \mathbf{w} + r \nabla L(\mathbf{w}|\mathcal{D})$
  - Minimisation:  $\mathbf{w} \rightarrow \mathbf{w}' = \mathbf{w} - r \nabla L(\mathbf{w}|\mathcal{D})$



## What Goes Right

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



- Taylor expanding around a minimum  $x^*$

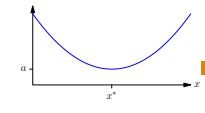
$$\begin{aligned} f(x) &= f(x^*) + (x - x^*)f'(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \dots \\ &= f(x^*) + \frac{1}{2}(x - x^*)^2 f''(x^*) + \frac{1}{3!}(x - x^*)^3 f'''(x^*) + \dots \end{aligned}$$

- 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

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

- so

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

- or

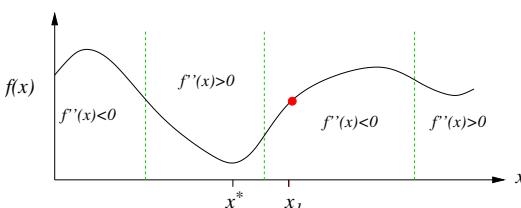
$$x^* = x - \frac{f'(x)}{f''(x)}$$

- This is Newton's method.

- For non-quadratic functions Newton's 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  $\epsilon^2$ . After two cycles we will be at a distance  $\epsilon^4$ , etc.
- If we are too far from a minimum we might go anywhere!
- We should follow the gradient until we are near the minimum.

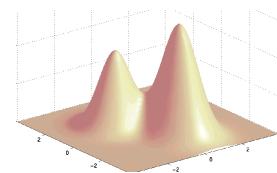
## Using the Second Derivative

- 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

- Motivation
- Gradient Descent
- Why Gradient Descent is Difficult**



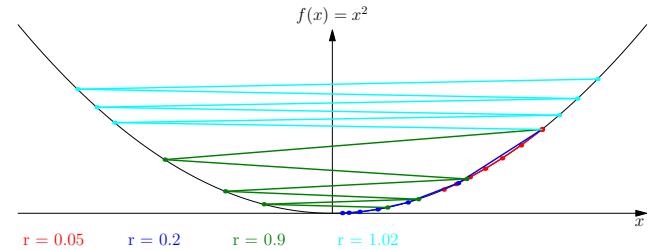
## Step Size

- Gradient descent

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

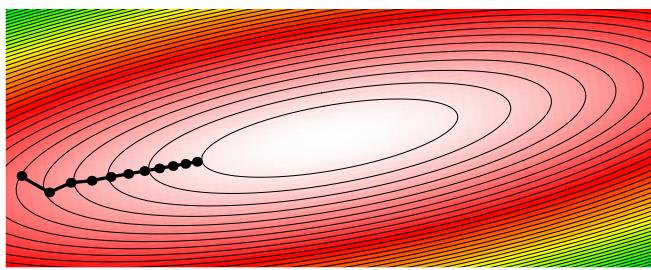
- Need to choose the learning rate of step size,  $r$
- Too small steps takes lots of time
- Too large steps takes you away from a minimum

$$x \leftarrow x - r f'(x)$$



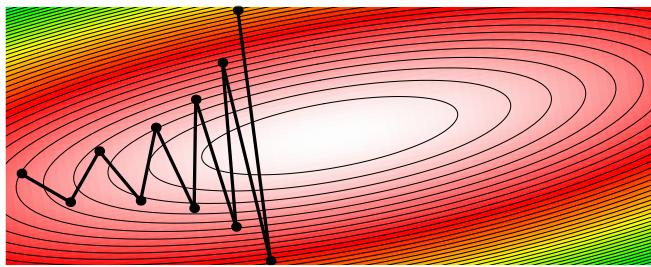
## Higher Dimensions

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



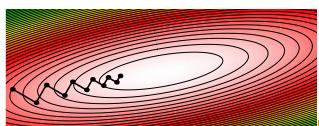
## More Haste Less Speed

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



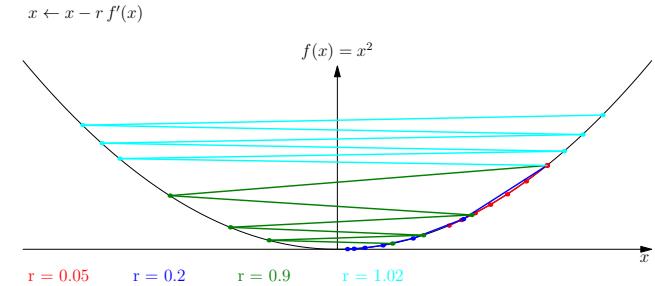
## Zig-Zag

- Note that in high dimensions gradient descent tends to zigzag



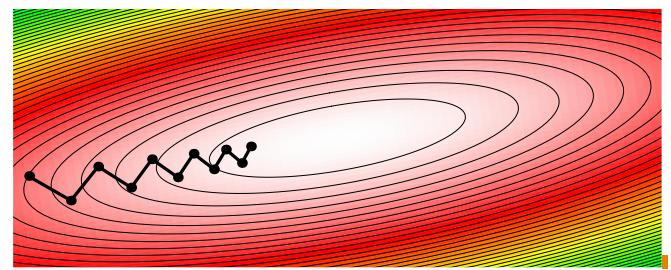
- If we computed the Hessian and used Newton's method we would jump straight to the minimum if we were in a quadratic potential
- However computing the Hessian is time consuming and misleading if we are not in a quadratic potential (i.e. far from the optimum)

## Step Size



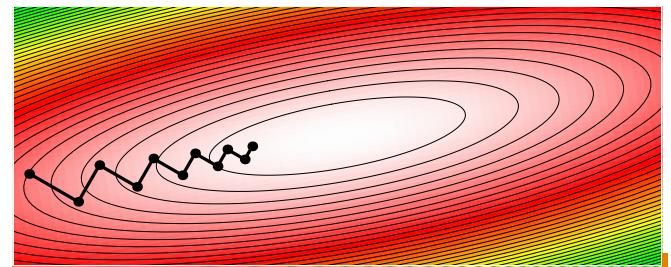
## Getting There Quicker

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



## Line Minimisation

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



## 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

## Trust Region

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

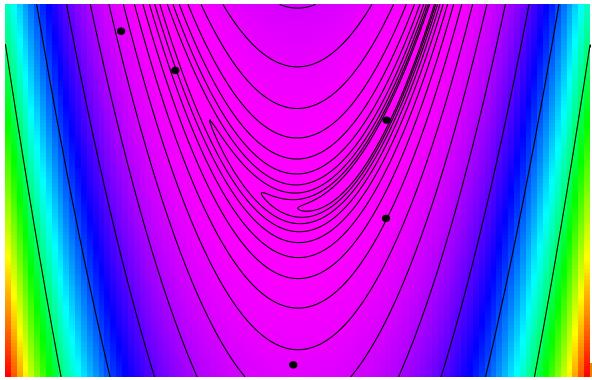
$$\epsilon_i(\mathbf{w}) \approx \epsilon_i(\mathbf{w}^{(k)}) + (\mathbf{w} - \mathbf{w}^{(k)}) \nabla \epsilon_i(\mathbf{w}^{(k)})$$

with  $\nabla \epsilon_i(\mathbf{w}^{(k)}) = \nabla f(\mathbf{x}_i|\mathbf{w}^{(k)})$

- Solve quadratic minimisation of approximate error  
 $\operatorname{argmin}_{\mathbf{w}} L_{approx}(\mathbf{w})$  with  $\mathbf{J} = \nabla \epsilon(\mathbf{w}^{(k)})$

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

$$\epsilon_1 = 10(x_2 - x_1^2) \text{ and } \epsilon_2 = 1 - x_1$$



- Solution given by  $\nabla_{\mathbf{w}} L_{approx}(\mathbf{w}) = 0$  gives

$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - (\mathbf{J}^\top \mathbf{J})^{-1} \mathbf{J}^\top \epsilon(\mathbf{w}^{(k)})$$

- Can lead us in the wrong direction

- Instead use  $\mathbf{w}^{(k+1)} = \operatorname{argmin}_{\mathbf{w}} L_{approx}(\mathbf{w}) + \nu \|\mathbf{w} - \mathbf{w}^{(k)}\|^2$

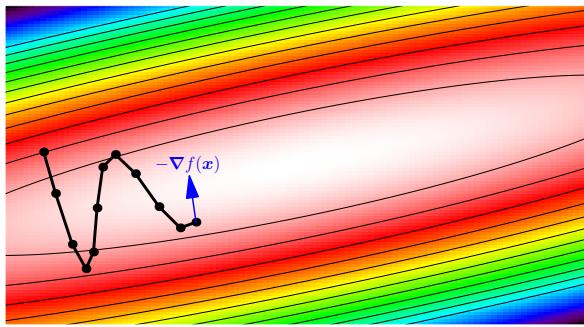
$$\mathbf{w}^{(k+1)} = \mathbf{w}^{(k)} - (\mathbf{J}^\top \mathbf{J} + \nu \mathbf{I})^{-1} \mathbf{J}^\top \epsilon(\mathbf{w})$$

- $\nu$  limits the step size

- If predicted reduction in error is accurate then reduce  $\nu$ , else if predicted reduction in error is very poor increase  $\nu$

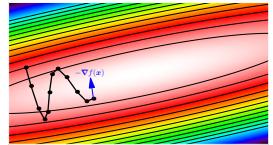
## Summary

- 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



*SGD, momentum, step size, ADAM*

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 delusory
- 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.

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

- We often compute an approximation

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

- 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  $\nabla L_{\mathcal{B}}(\mathbf{w})$  than  $\nabla L_{\mathcal{D}}(\mathbf{w})$
- 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

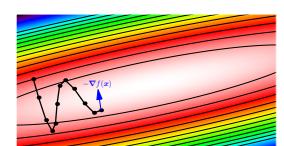
- For gradient descent we need to compute the gradient
- For most of my life this was just a pain
- You guys have it easy, modern deep learning frameworks do this for you automatically
- This is a *game changer!* We are no longer afraid of large models
- We can differentiate through algorithms allowing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

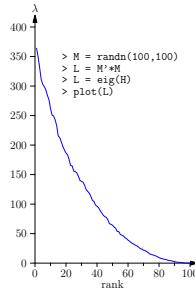
## Gradient Descent

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

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

- In high dimensions the Hessian  $\mathbf{H}$  will have a spectrum of eigenvalues

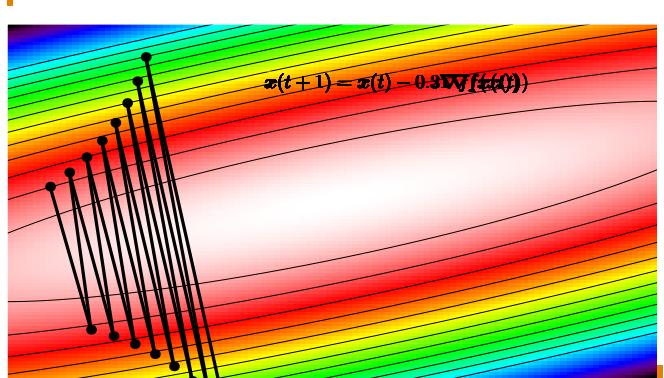
- This means there are different scales



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

9



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

10

## 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)

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

11

## 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

$$\begin{aligned} v(t+1) &= \gamma v(t) - \eta \nabla f(x(t)) \\ x(t+1) &= x(t) + v(t+1) \end{aligned}$$

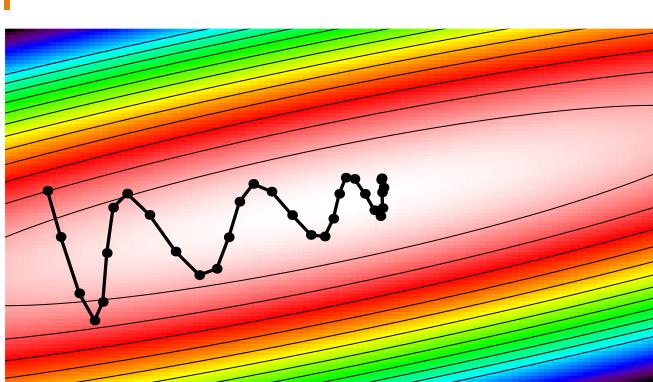
$0 < \gamma < 1$  is a damping term

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

12

## Gradient Descent with Momentum



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

13

## Adaptive Methods

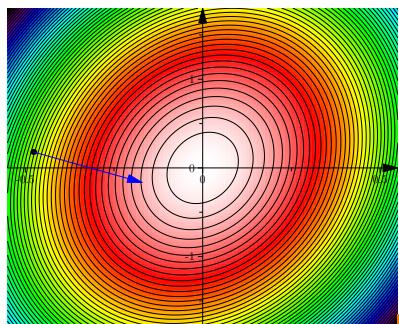
- A major difficulty of high dimensional optimisation is the existence of different scales
- That is, there are some directions where we have to move a lot (low curvature), while other directions we have to make small steps
- In adaptive methods we use a dynamic algorithm to rescale the step size of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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

14

## Rescaling Co-ordinates



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

15

## 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
- $$S_i^g(t+1) = (1 - \gamma) S_i^g(t) + \gamma \left( \frac{\partial L_B(\mathbf{w}(t))}{\partial w_i(t)} \right)^2$$
- To estimate the relative scale of the weights we compute a running average of the squared weights
- $$S_i^w(t+1) = (1 - \gamma) S_i^w(t) + \gamma w_i(t)^2$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

16

- The AdaDelta algorithm uses the update

$$w_i(t+1) = w_i(t) - \eta \sqrt{\frac{S_i^w(t+1) + \epsilon}{S_i^g(t+1) + \epsilon}} \frac{\partial L_B(\mathbf{w}(t))}{\partial w_i(t)}$$

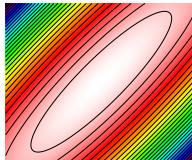
- Note that we can rescale the weights or rescale the gradients yet the update would be the same – this is one aspect of a covariant algorithm
- Because we are adaptively changing our step size we can use a single step size throughout the optimisation

## 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)

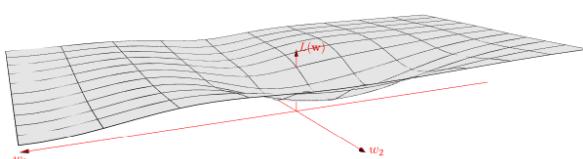
## Correlated Weights

- 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



## Loss Landscape

- A useful concept for understanding optimisation is the loss landscape
- For every value of the weights  $\mathbf{w}$  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



- 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

$$M_i(t+1) = (1 - \beta) M_i(t) + \beta \frac{\partial L_B(\mathbf{w}(t))}{\partial w_i(t)}$$

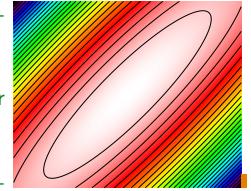
$$S_i(t+1) = (1 - \gamma) S_i(t) + \gamma \left( \frac{\partial L_B(\mathbf{w}(t))}{\partial w_i(t)} \right)^2$$

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

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

## 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

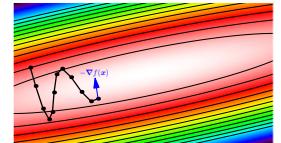


Can't rescale coordinates

- 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

## Outline

- SGD
- Momentum
- Loss Landscapes

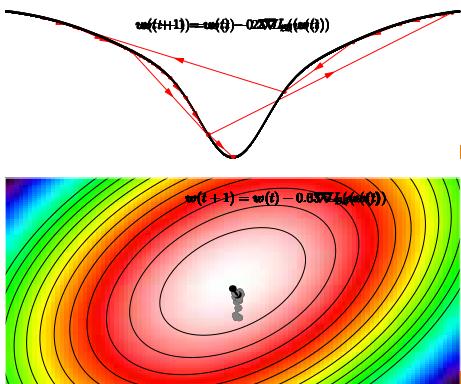


## Global and Local Minima

- 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

- 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

## Reducing the Step Size



## 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  $\mathbf{W}_2\mathbf{W}_1$  then for any invertible matrix  $\mathbf{U}$  (where  $\mathbf{UU}^{-1} = \mathbf{I}$ )

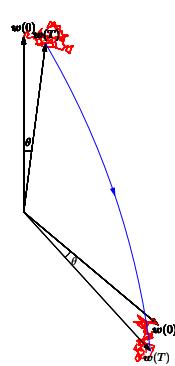
$$\mathbf{W}_2\mathbf{W}_1 = \mathbf{W}_2\mathbf{I}\mathbf{W}_1 = \mathbf{W}_2(\mathbf{U}\mathbf{U}^{-1})\mathbf{W}_1 = (\mathbf{W}_2\mathbf{U})(\mathbf{U}^{-1}\mathbf{W}_1)$$

so a network with weights  $\mathbf{W}'_2 = \mathbf{W}_2\mathbf{U}$  and  $\mathbf{W}'_1 = \mathbf{U}^{-1}\mathbf{W}_1$  will perform the same mapping

- There will be a huge space of equivalent networks

## Symmetries

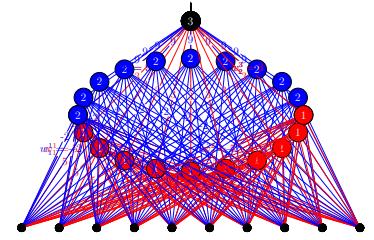
- The landscape potentially has a huge manifold with the same losses (neural network)
- Find solution close to start
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation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



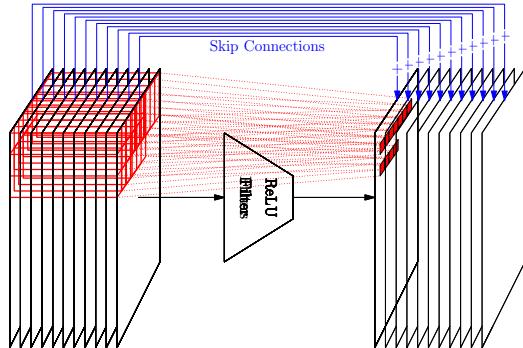
- My view of the loss landscape is that we have valleys inside bigger valleys inside bigger valleys, and so on
- If our noise is high (we use large step size), we can navigate away from local valleys and move towards the big valley
- But, we can't explore the small valleys
- Often when the rate of improvement 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

## 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



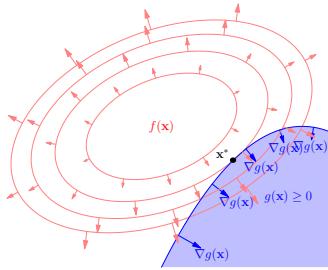
## Skip Connections



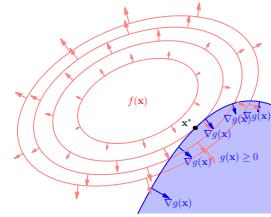
- Breaks permutation symmetry. Used in Resnets, transformer, etc.

## 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)



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



1. Constrained Optimisation
2. Inequalities
3. Duality

## 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

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

- A standard procedure is to define the Lagrangian

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

where  $\alpha$  is known as a Lagrange multiplier

- In the extended space  $(\mathbf{x}, \alpha)$  we have to solve

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

## Conditions on Optimum

- The optimisation problem is

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

- Assuming differentiability

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

$$\frac{\partial \mathcal{L}}{\partial \alpha} = -g(\mathbf{x}) = 0$$

- The second condition is just the constraint

- But what about first condition:  $\nabla_{\mathbf{x}} f(\mathbf{x}) = \alpha \nabla_{\mathbf{x}} g(\mathbf{x})$ ?

## Note on Gradients

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

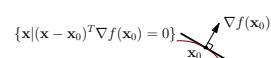
$$f(\mathbf{x}) = f(\mathbf{x}_0) + (\mathbf{x} - \mathbf{x}_0)^T \nabla_{\mathbf{x}} f(\mathbf{x}_0) + \frac{1}{2} (\mathbf{x} - \mathbf{x}_0)^T \mathbf{H} (\mathbf{x} - \mathbf{x}_0) + \dots$$

where  $\mathbf{H}$  is a matrix of second derivative known as the Hessian

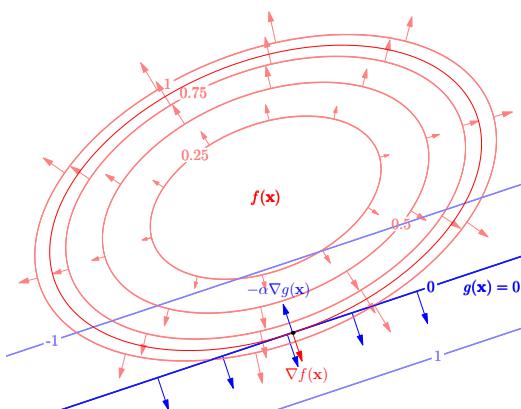
- If we consider the set of points perpendicular to  $\nabla_{\mathbf{x}} f(\mathbf{x}_0)$  which go through  $\mathbf{x}_0$  (the tangent plane), these will have values

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

thus  $\nabla_{\mathbf{x}} f(\mathbf{x})$  is always orthogonal to the contour lines



## Constrained Optima



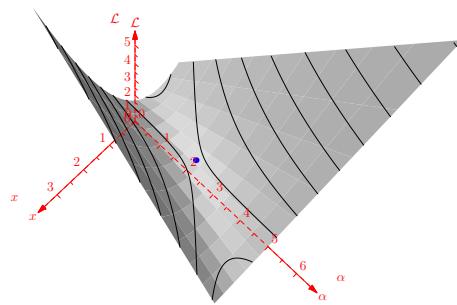
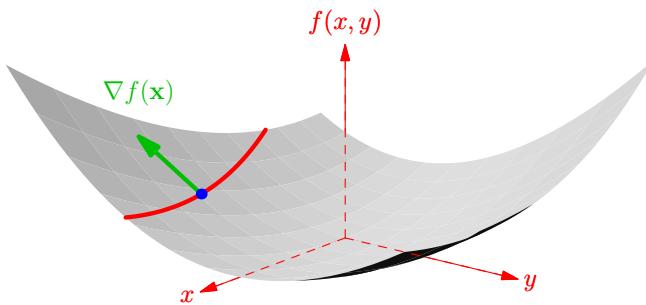
## Example

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

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

$$\text{and } \frac{\partial \mathcal{L}}{\partial \alpha} = -g(\mathbf{x}) = -x + 2y + 3 = 0$$

- Solving simultaneous equations gives minima at  $(x, y) = (\frac{3}{4}, -\frac{9}{8})$  with  $\alpha = \frac{21}{8}$



### Multiple Constraints

- Given an optimisation problem with multiple constraints

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

- We introduce multiple Lagrange multipliers

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

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

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

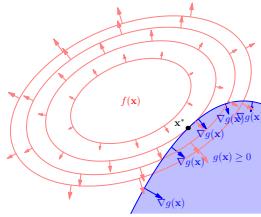
plus the original constraints  $\frac{\partial \mathcal{L}(\mathbf{x}, \boldsymbol{\alpha})}{\partial \alpha_k} = -g_k(\mathbf{x}) = 0$

### Example

- Minimise  $f(\mathbf{x}) = x^2 + 2y^2 + 5z^2 - xy - xz$  subject to  $g_1(\mathbf{x}) = x - 2y - z - 3 = 0$  and  $g_2(\mathbf{x}) = 2x + 3y + z - 2 = 0$
  - Writing  $\mathcal{L}(\mathbf{x}, \boldsymbol{\alpha}) = f(\mathbf{x}) - \alpha_1 g_1(\mathbf{x}) - \alpha_2 g_2(\mathbf{x})$
  - Condition for minima is  $\nabla_{\mathbf{x}} \mathcal{L} = 0$  or  $\nabla_{\mathbf{x}} f(\mathbf{x}) = \sum_{k=1}^2 \alpha_k \nabla_{\mathbf{x}} g_k(\mathbf{x})$
- $$\begin{pmatrix} 2x - y - z \\ -x + 4y \\ 10z - x \end{pmatrix} = \alpha_1 \begin{pmatrix} 1 \\ -2 \\ -1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}$$
- and  $\frac{\partial \mathcal{L}}{\partial \alpha_i} = -g_i(\mathbf{x}) = 0$
- 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

- Constrained Optimisation
- Inequalities
- Duality



- Suppose we have the problem

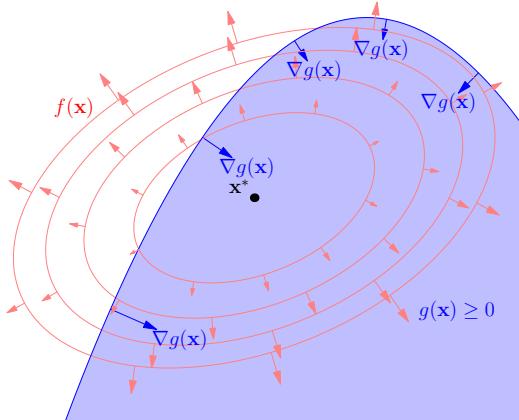
$$\min_{\mathbf{x}} f(\mathbf{x}) \text{ subject to } g(\mathbf{x}) \geq 0$$

- Looks much more complicated, but

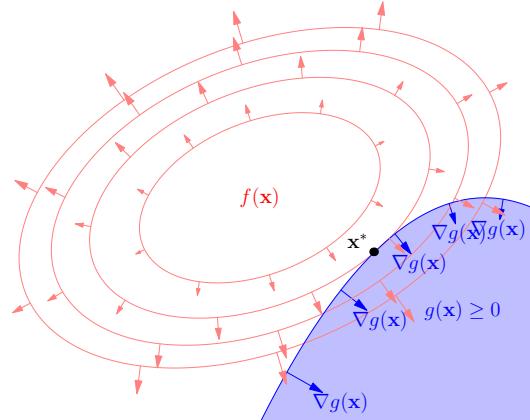
- Only two things can happen

- Either a minimum,  $\mathbf{x}^*$ , of  $f(\mathbf{x})$  satisfies  $g(\mathbf{x}^*) > 0$ 
  - We then have an unconstrained optimisation problem
- Otherwise, it satisfies  $g(\mathbf{x}^*) = 0$ 
  - We have a constrained optimisation problem

### Inside Region



### On the Boundary



## KKT Conditions

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

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

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

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

- where either

- $\alpha = 0$  and the solutions in the interior
- $\alpha > 0$  and  $g(\mathbf{x}) = 0$ , i.e. the solution is on the boundary

- These conditions are known as the Karush-Kuhn-Tucker conditions

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

17

## Many Inequalities

- Given the problem

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

- We introduce multiple Lagrange multipliers

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

- The condition for an optima is

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

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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

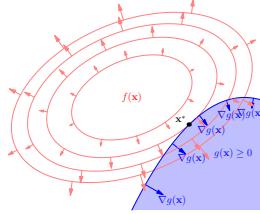
18

## Outline

### 1. Constrained Optimisation

### 2. Inequalities

### 3. Duality



## Solving the Lagrangian for $x$

- Consider minimising a function  $f(\mathbf{x})$  subject to a set of constraints  $g_i(\mathbf{x}) = 0$  or  $g_i(\mathbf{x}) \leq 0$

- We can consider this a double optimisation problem

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

where there would be constraints on  $\alpha_i$  if we had an inequality constraint

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

19

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

20

## Dual Problem

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

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

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

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

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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

21

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

21

## 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

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

- The Lagrangian becomes

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

- An equivalent way of writing the Lagrangian is

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

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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

22

## Linear Programming

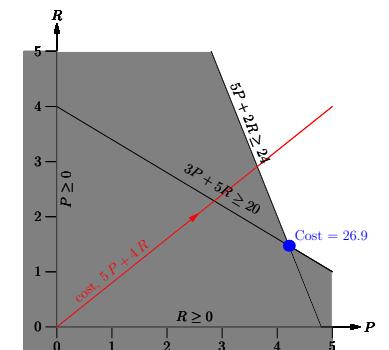
- Minimise  $5P + 4R$

- Subject to

$$3P + 5R \geq 20$$

$$5P + 2R \geq 24$$

$$P, R \geq 0$$



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

23

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

24

- We can write the problem as a Lagrange problem

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

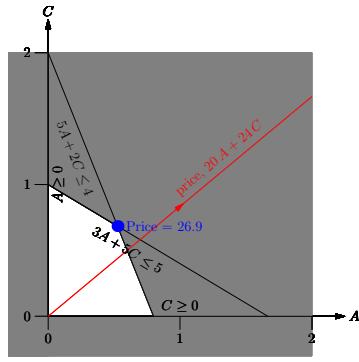
- subject to  $P, R, A, B \geq 0$

- $A$  and  $C$  are Lagrange multipliers for vitamin A and C

- We can rearrange the Lagrangian to obtain

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

## Dual Linear Programme



- Maximise  $20A + 24C$

- Subject to

- $3A + 5C \leq 5$
- $5A + 2C \leq 4$
- $A, C \geq 0$

## Quadratic Programming

- A quadratic programme involves minimising a quadratic function  $x^T Q x$  (with  $Q \succ 0$ ) subject to linear constraints  $Mx = b$  (or  $Mx \leq b$ )

- We can define the Lagrangian

$$\mathcal{L}(x, \alpha) = x^T Q x - \alpha^T (Mx - b)$$

- Where the solution is given by  $\max_{\alpha} \min_x \mathcal{L}(x, \alpha)$

- If the constraints are inequality constraints then  $\alpha_i \geq 0$

## Dual Quadratic Programming Problem

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

$$\mathcal{L}(x, \alpha) = x^T Q x - \alpha^T (Mx - b)$$

- We get the dual problem

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

- 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)

- The Lagrangian

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

leads to the dual problem

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

- 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!

## 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  $x$  while the second linear programme depends on the number of constraints (or dimensionality of  $\alpha$ )
- This is important, for example, in Wasserstein GANs

## Solution to Quadratic Programming Problem

- Using

$$\mathcal{L}(x, \alpha) = x^T Q x - \alpha^T (Mx - b)$$

- Then

$$\nabla_x \mathcal{L}(x, \alpha) = 2Qx - M^T\alpha$$

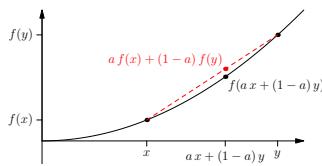
- So  $\nabla_x \mathcal{L}(x, \alpha) = 0$  implies

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

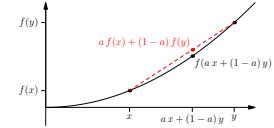
## Lessons

- 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

### Convexity



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

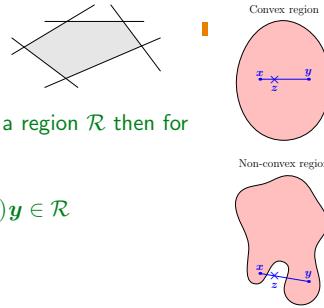


Convex sets, convex functions, Jensen's inequality

### Convex Regions

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

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



### 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  $x, y \in \mathcal{S}$  and any  $a \in [0,1]$

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

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

### Positive Semi-Definite Matrices

- Recall that a matrix  $M$  is positive semi-definite if for any vector  $v$

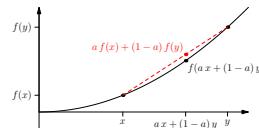
$$v^T M v \geq 0$$

(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  $M$  is positive semi-definite by  $M \succeq 0$ , and  $M \succ 0$  if it is positive definite
- The set of positive semi-definite (PSD) matrices (or kernels) form a convex set.

### Outline

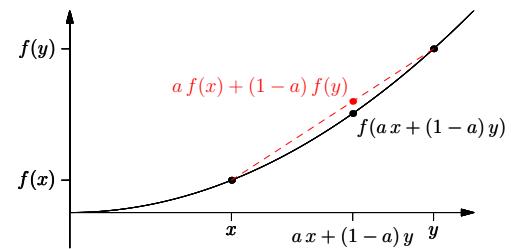
1. Convex sets
2. Convex functions
3. Jensen's inequality



### Convex Functions

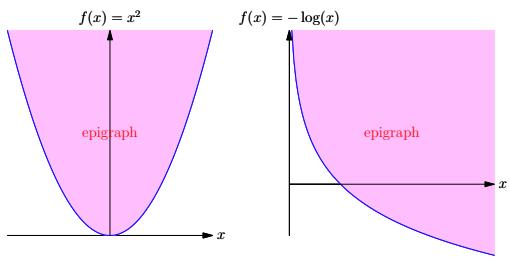
- Any function  $f(x)$  is said to be a **convex function** if for any two points  $x$  and  $y$  and any  $a \in [0,1]$

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



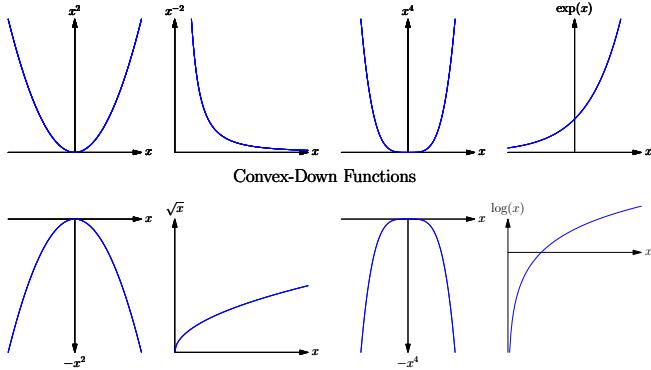
## Epigraph

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



## Examples

### Convex-Up Functions



## Strictly Convex Function

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

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

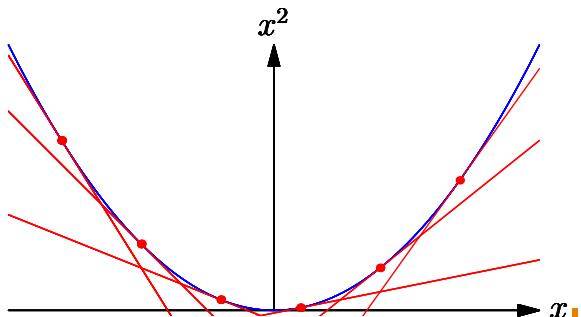
are said to be **strictly convex functions**

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

## Properties of Convex Functions

- Convex functions lie on or above any tangent line

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



## 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

## Linear Functions

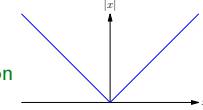
- Linear functions are given by

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

- 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



## Convexity in High Dimensions

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

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

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

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

## Second Derivatives

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

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

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

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

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + (\mathbf{x} - \mathbf{x}^*)^\top \nabla f(\mathbf{x}^*)$$

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

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

## Proving Convexity

- $f(x) = x^2$  is strictly convex as  $f''(x) = 2 > 0$
- $f(x) = e^{cx}$  is strictly convex as  $f''(x) = c^2 e^{cx} > 0$
- $f(x) = \log(x)$  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

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

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

## Sums of Convex Functions

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

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

is convex

- Proof

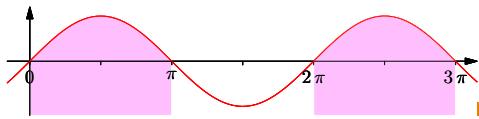
$$g''(x) = \sum_i a_i f''_i(x)$$

but  $f''_i(x) \geq 0$  so  $g''(x)$  is a sum of non-negative terms

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

## Constraints

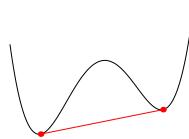
- All the properties we have discussed hold for functions defined on a convex set
- $\sin(x)$  is not generally neither convex up or down
- $\sin(x)$  for  $x \in [0, \pi]$  is convex-down (its second derivative  $-\sin(x)$  is less than or equal to 0 in this range)



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

## 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



## Linear Regression

- For linear regression the loss function

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

is convex

- Since the Hessian  $\mathbf{H} = 2\mathbf{X}^\top \mathbf{X} \succeq 0$  (positive semi-definite)
   
(For any vector  $\mathbf{v}$  then  $\mathbf{v}^\top \mathbf{H} \mathbf{v} = 2\mathbf{v}^\top \mathbf{X}^\top \mathbf{X} \mathbf{v} = 2\|\mathbf{X}\mathbf{v}\|^2 \geq 0$ )
- If  $\mathbf{H} \succ 0$  there will be a unique minimal 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

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

- Because  $\|\mathbf{w}\|^2$  is strictly convex the loss function is strictly convex and so will have a unique solution

- Using an  $L_1$  regulariser (Lasso)

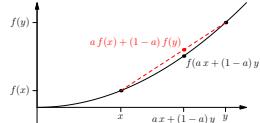
$$L(\mathbf{w}) = \|\mathbf{X}\mathbf{w} - \mathbf{y}\|^2 + \eta \|\mathbf{w}\|_1$$

again  $\|\mathbf{w}\|_1$  is convex and so  $L(\mathbf{w})$  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  $f(x)$  is a convex(-up) function then

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

- If  $f(x)$  is a convex(-down) function then

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

## Proof

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

$$f(\mathbf{x}) \geq f(\mathbf{x}^*) + (\mathbf{x} - \mathbf{x}^*)^\top \nabla f(\mathbf{x}^*)$$

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

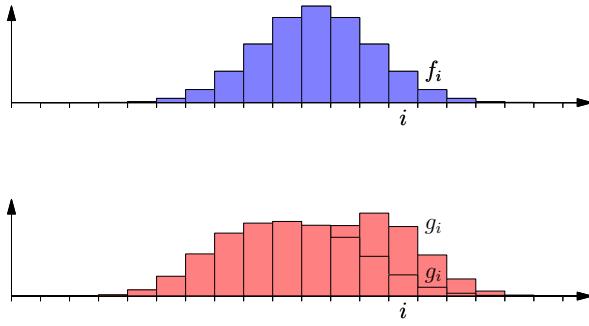
$$f(\mathbf{X}) \geq f(\mathbb{E}[\mathbf{X}]) + (\mathbf{X} - \mathbb{E}[\mathbf{X}])^\top \nabla f(\mathbb{E}[\mathbf{X}])$$

- Taking expectations of both sides

$$\begin{aligned} \mathbb{E}[f(\mathbf{X})] &\geq f(\mathbb{E}[\mathbf{X}]) + (\mathbb{E}[\mathbf{X}] - \mathbb{E}[\mathbf{X}])^\top \nabla f(\mathbb{E}[\mathbf{X}]) \\ &= f(\mathbb{E}[\mathbf{X}]) \end{aligned}$$

□

## Kullback-Leibler Divergence



$$KL(f||g) = - \sum_{i=1}^n f_i \log\left(\frac{g_i}{f_i}\right) = 0.235$$

## 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

## Simple Proofs with Jensen's Inequality

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

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

(i.e. variance are non-negative)

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

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

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

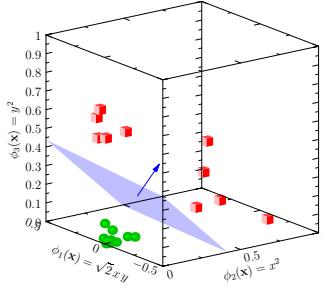
## Proof of Gibbs' Inequality

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

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

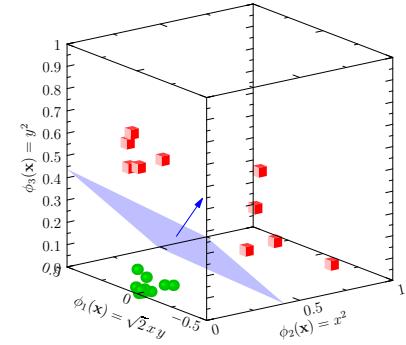
- We will meet KL-divergences later on

## Support Vector Machines



Support Vector Machines, maximum margins

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

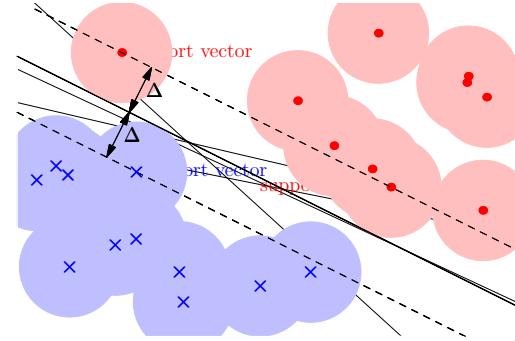


## Support Vector Machines

- 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

$$\mathbf{x} = (x_1, x_2, \dots, x_p)^T \rightarrow \phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_r(\mathbf{x}))^T$$

$$r \gg p$$

- 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

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

## Kernel Trick

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

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

(like an eigenvector decomposition of a matrix)

- Never need to compute  $\phi_k(\mathbf{x}_i)$  explicitly as we only need the inner-product  $\langle \phi(\mathbf{x}_i), \phi(\mathbf{x}_j) \rangle = K(\mathbf{x}_i, \mathbf{x}_j)$  to compute maximum margin separating hyper-plane
- Sometimes  $\phi(\mathbf{x}_i)$  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 variables
- Strong restriction: *positive semi-definite*
- Examples

$$\text{Quadratic kernel: } K(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^T \mathbf{x}_2)^2$$

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

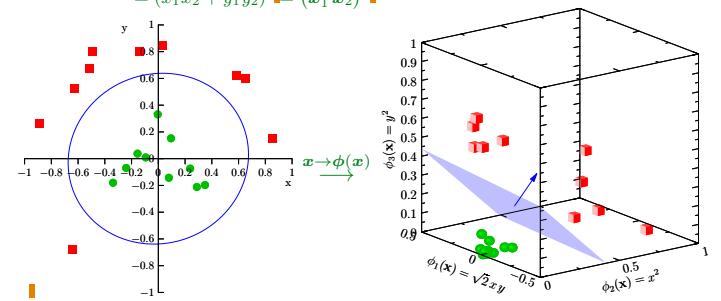
- Consider the mapping

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

## Non-linear Separation of Data

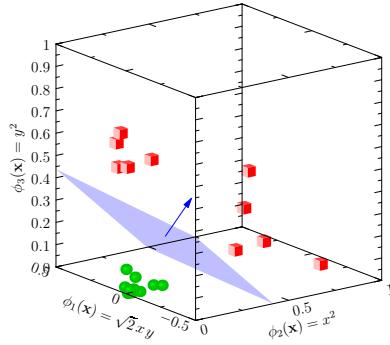
$$K(\mathbf{x}_1, \mathbf{x}_2) = (x_1^2 - y_1^2 - \sqrt{2}x_1y_1)(x_2^2 - y_2^2 - \sqrt{2}x_2y_2) = x_1^2x_2^2 + y_1^2y_2^2 + 2x_1y_1x_2y_2$$

$$= (x_1x_2 + y_1y_2)^2 = (\mathbf{x}_1^T \mathbf{x}_2)^2$$



## Outline

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



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

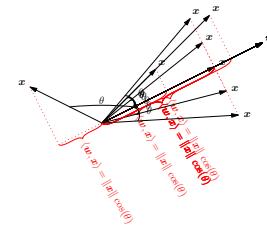
9

## Inner Product

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

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

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



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

10

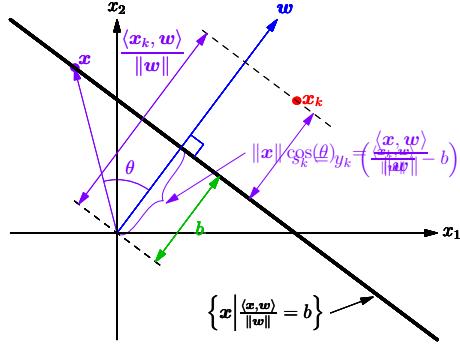
## Maximise Margin

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

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

where  $\Delta$  is the margin

## Distance to hyperplanes



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

11

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

12

## Constrained Optimisation

- Wish to find  $\mathbf{w}$  and  $b$  to maximise  $\Delta$  subject to constraints

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

- If we divide through by  $\Delta$

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

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

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

## Quadratic Programming Problem

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

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

- Minimising  $\|\hat{\mathbf{w}}\|^2$  is equivalent to maximising the margin  $\Delta$

- Can write the optimisation problem as a *quadratic programming problem*

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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

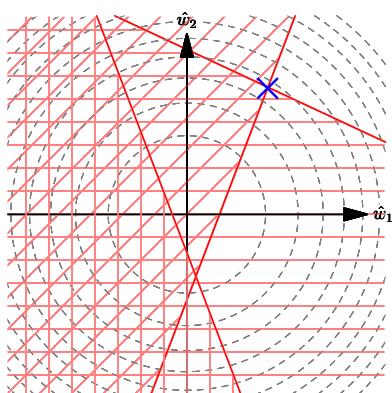
13

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

14

## Quadratic Programming in SVMs



## Quadratic Programming

- We have a quadratic programming problem for the weights  $\hat{\mathbf{w}} = (\hat{w}_1, \hat{w}_2, \dots, \hat{w}_p)$  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  $O(m^3)$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

15

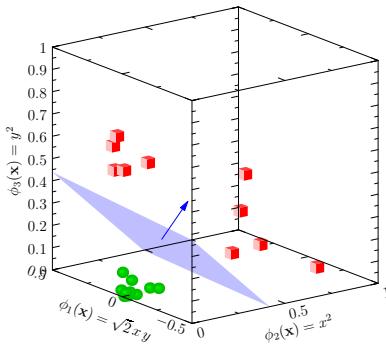
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

16

## 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

$$\mathbf{x} \rightarrow \phi(\mathbf{x})$$

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

## Lagrangian

- In the extended feature space we can find a separating plane (given by  $\mathbf{w}$  and  $b$ ) with maximum margin by solving the problem
- $$\min_{\mathbf{w}, b} \frac{\|\mathbf{w}\|^2}{2} \text{ subject to } y_k(\langle \mathbf{w}, \phi(\mathbf{x}_k) \rangle - b) \geq 1 \text{ for all } k = 1, 2, \dots, m$$
- We can write this as a Lagrange problem

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

where

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

subject to  $\alpha_k \geq 0$

## Obtaining the Dual Form of the Problem

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

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

- $\nabla_{\mathbf{w}} \mathcal{L} = \mathbf{w} - \sum_{k=1}^m \alpha_k y_k \phi(\mathbf{x}_k) = 0$  implies that  $\mathbf{w}^* = \sum_{k=1}^m \alpha_k y_k \phi(\mathbf{x}_k)$

- $\frac{\partial \mathcal{L}}{\partial b} = \sum_{k=1}^m \alpha_k y_k = 0$  implies  $\sum_{k=1}^m \alpha_k y_k = 0$

- Substituting back into the Lagrangian

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

## The Dual Problem

- The dual problem is now to find  $\alpha_k$ 's that maximise

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

- subject to constraints

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

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

## Sequential Minimal Optimisation

- One of the most efficient techniques for training SVMs is *Sequential Minimal Optimisation* or SMO
- This takes two Lagrange multipliers  $\alpha_i$  and  $\alpha_j$  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  $\alpha$ 's to optimise
- Run until close to the optimum

## Kernel Trick

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

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

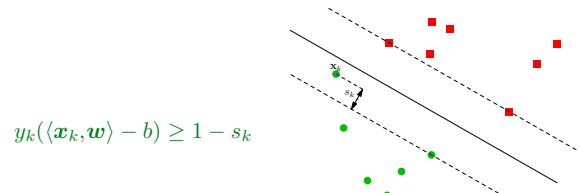
- As  $\langle \phi(\mathbf{x}_k), \phi(\mathbf{x}_l) \rangle$  appears in the dual problem we can express the dual problem as finding  $\alpha_k$ 's that maximise

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

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

## Soft Margins

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



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

## Dual Problem with Slack Variables

- The Lagrangian with slack variables is

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

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

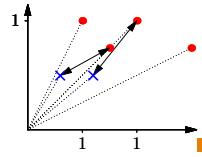
- Now minimising with respect to  $s_i$

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

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

## 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 others—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)

## Choosing the Right Kernel Function

- There are kernels designed 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

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

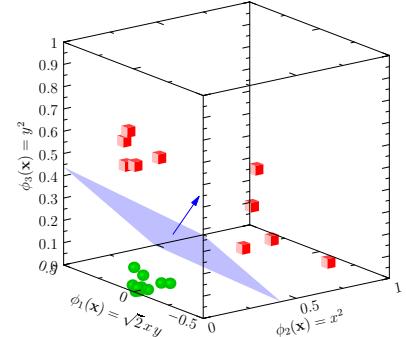
- Optimal  $\gamma$  values range over  $2^{-15}$ – $2^3$

## 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

## Outline

- The Big Picture
- Maximum Margins
- Duality
- Practice



## Optimising C

- Recall that we can introduce soft-margins using slack variables where we minimise  $\frac{\|\mathbf{w}\|^2}{2} + C\sum_{k=1}^m s_k$  subject to constraints
- In practice it can make a huge difference to the performance if we change  $C$
- Optimal  $C$  values change 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

## Multi-Class Problems

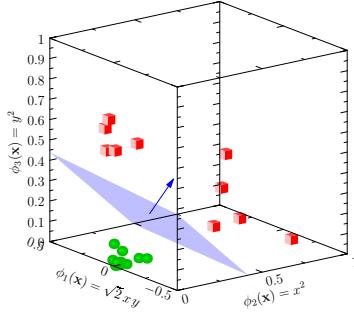
- By construction SVMs separate only two classes
- If we have a multi-class problem we have to use multiple SVMs
- There are two major ways practitioners do this
  - One-versus-all:** for each class, train a separate classifier to determine that class versus all others
  - All-pairs:** train a classifier 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

## 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

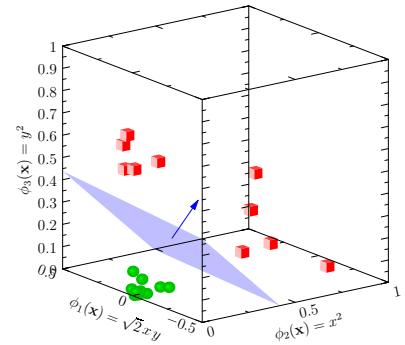
## Kernel Trick



The Kernel Trick, SVMs, Regression

## Outline

1. The Kernel Trick
2. Positive Semi-Definite Kernels
3. Kernel Properties
4. Beyond Classification



## SVM Kernels

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

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

- where  $\phi(\mathbf{x}) = (\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots)^T$  and  $\phi_i(\mathbf{x})$  are real valued functions of  $\mathbf{x}$
- $K(\mathbf{x}, \mathbf{y})$  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

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

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

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

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

## Classifying New Data

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

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

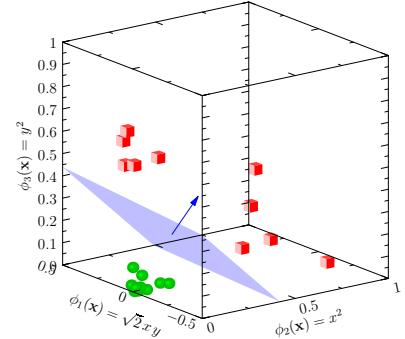
- In the dual representation this becomes

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

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

## Outline

1. The Kernel Trick
2. Positive Semi-Definite Kernels
3. Kernel Properties
4. Beyond Classification



## Recap on Eigen Systems

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

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

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

$$\mathbf{V}^\top \mathbf{V} = \mathbf{V} \mathbf{V}^\top = \mathbf{I}$$

- Such matrices are said to be orthogonal

## Eigen Decomposition

- From the eigenvalue equation  $\mathbf{M}\mathbf{v}^{(k)} = \lambda^{(k)}\mathbf{v}^{(k)}$

$$\mathbf{M}\mathbf{V} = \mathbf{V}\Lambda \quad \text{where} \quad \Lambda = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$$

- Multiplying on the right by  $\mathbf{V}^\top$  we get

$$\mathbf{M} = \mathbf{V}\Lambda\mathbf{V}^\top = \sum_{k=1}^n \lambda^{(k)} \mathbf{v}^{(k)} \mathbf{v}^{(k)\top}$$

Or

$$M_{ij} = \sum_{k=1}^n \lambda^{(k)} v_i^{(k)} v_j^{(k)} = \sum_{k=1}^n u_i^{(k)} u_j^{(k)} = \langle \mathbf{u}_i, \mathbf{u}_j \rangle$$

$$u_i^{(k)} = \sqrt{\lambda^{(k)}} v_i^{(k)}$$

## Eigenfunctions

## Limit Process

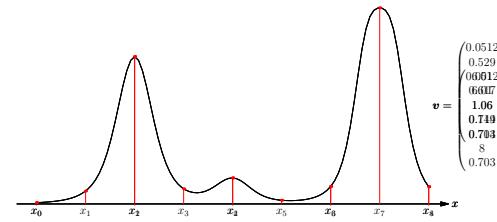
- By analogy for a symmetric function of two variables we can define an eigenfunction

$$\int K(\mathbf{x}, \mathbf{y}) \psi(\mathbf{y}) d\mathbf{y} = \lambda \psi(\mathbf{x})$$

- In general there will be a denumerable set of eigenfunctions  $\psi^{(k)}(\mathbf{x})$  where

$$K(\mathbf{x}, \mathbf{y}) = \sum_k \lambda^{(k)} \psi^{(k)}(\mathbf{x}) \psi^{(k)}(\mathbf{y})$$

- This is known as Mercer's theorem

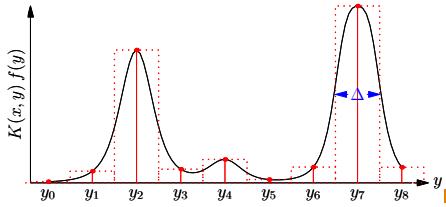


- In the limit where the number of sample points goes to infinity the vector more closely approximates a function
- Instead of the indices being numbers we use  $k \leftarrow x_k$

## Linear Operators

- Recall a linear function  $\mathcal{T}[f(x)]$  can be represented by a kernel

$$\mathcal{T}[f(x)] = \int_{y \in \mathcal{I}} K(x, y) f(y) dy \approx \Delta \sum_{j=1}^n K(x, y_j) f(y_j)$$



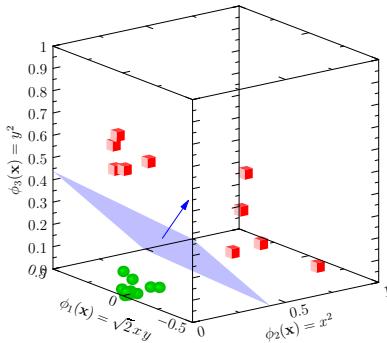
This is just a matrix equation with  $M_{ij} = \Delta K(x_i, y_j)$

## SVM Kernels

- If we define  $\phi^{(k)}(\mathbf{x}) = \sqrt{\lambda^{(k)}} \psi^{(k)}(\mathbf{x})$  then
- $K(\mathbf{x}, \mathbf{y}) = \sum_k \lambda^{(k)} \psi^{(k)}(\mathbf{x}) \psi^{(k)}(\mathbf{y}) = \sum_k \phi^{(k)}(\mathbf{x}) \phi^{(k)}(\mathbf{y})$
- This is the definition of a SVM kernel we started with
- Note that for  $\phi^{(k)}(\mathbf{x})$  to be real  $\lambda^{(k)} \geq 0$  for all  $k$
- If  $\lambda^{(k)} < 0$  then  $\langle \phi(\mathbf{x}), \phi(\mathbf{x}) \rangle = \|\phi(\mathbf{x})\|^2$  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  $\alpha$  diverges

## Outline

- The Kernel Trick
- Positive Semi-Definite Kernels
- Kernel Properties
- Beyond Classification



## Properties of Positive Semi-Definiteness

- Since

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

- An immediate consequence is that for any function  $f(\mathbf{x})$

$$\begin{aligned} \int f(\mathbf{x}) K(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} &= \int f(\mathbf{x}) \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &= \left\langle \int f(\mathbf{x}) \phi(\mathbf{x}) d\mathbf{x}, \int f(\mathbf{y}) \phi(\mathbf{y}) d\mathbf{y} \right\rangle \\ &= \left\| \int f(\mathbf{x}) \phi(\mathbf{x}) d\mathbf{x} \right\|^2 \geq 0 \end{aligned}$$

- 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

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

## Positive Semi-Definiteness

- The following statements are equivalent

- \*  $K(\mathbf{x}, \mathbf{y})$  is positive semi-definite (written  $K(\mathbf{x}, \mathbf{y}) \succeq 0$ )
- \* The eigenvalues of  $K(\mathbf{x}, \mathbf{y})$  are non-negative
- \* The kernel can be written

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

where the  $\phi^{(k)}(\mathbf{x})$ 's are real functions

- \* For any real function  $f(\mathbf{x})$

$$\int f(\mathbf{x}) K(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0$$

## Adding Kernels

- We can construct SVM kernels from other kernels

- If  $K_1(\mathbf{x}, \mathbf{y})$  and  $K_2(\mathbf{x}, \mathbf{y})$  are valid kernels then so is  

$$K_3(\mathbf{x}, \mathbf{y}) = K_1(\mathbf{x}, \mathbf{y}) + K_2(\mathbf{x}, \mathbf{y})$$

$$\begin{aligned} Q &= \int f(\mathbf{x}) K_3(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &= \int f(\mathbf{x}) (K_1(\mathbf{x}, \mathbf{y}) + K_2(\mathbf{x}, \mathbf{y})) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \\ &= \int f(\mathbf{x}) K_1(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} + \int f(\mathbf{x}) K_2(\mathbf{x}, \mathbf{y}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0 \end{aligned}$$

- If  $K(\mathbf{x}, \mathbf{y})$  is a valid kernel so is  $cK(\mathbf{x}, \mathbf{y})$  for  $c > 0$

## Product of Kernels

- If  $K_1(\mathbf{x}, \mathbf{y})$  and  $K_2(\mathbf{x}, \mathbf{y})$  are valid kernels then so is  

$$K_3(\mathbf{x}, \mathbf{y}) = K_1(\mathbf{x}, \mathbf{y}) K_2(\mathbf{x}, \mathbf{y})$$

- Writing

$$K_1(\mathbf{x}, \mathbf{y}) = \sum_i \phi_i^{(1)}(\mathbf{x}) \phi_i^{(1)}(\mathbf{y}), \quad K_2(\mathbf{x}, \mathbf{y}) = \sum_j \phi_j^{(2)}(\mathbf{x}) \phi_j^{(2)}(\mathbf{y})$$

then

$$\begin{aligned} K_3(\mathbf{x}, \mathbf{y}) &= \sum_{i,j} \phi_i^{(1)}(\mathbf{x}) \phi_i^{(1)}(\mathbf{y}) \phi_j^{(2)}(\mathbf{x}) \phi_j^{(2)}(\mathbf{y}) \\ &= \sum_{i,j} \phi_{ij}^{(3)}(\mathbf{x}) \phi_{ij}^{(3)}(\mathbf{y}) = \langle \phi^{(3)}(\mathbf{x}), \phi^{(3)}(\mathbf{y}) \rangle \end{aligned}$$

where  $\phi_{ij}^{(3)}(\mathbf{x}) = \phi_i^{(1)}(\mathbf{x}) \phi_j^{(2)}(\mathbf{x})$

## Exponentiating Kernels

- If  $K(\mathbf{x}, \mathbf{y})$  is a valid kernel so is  $K^n(\mathbf{x}, \mathbf{y})$  (by induction)

- Assume  $K(\mathbf{x}, \mathbf{y}) \succeq 0$  this satisfies base case
- If  $K^{n-1}(\mathbf{x}, \mathbf{y}) \succeq 0$  then

$$K^n(\mathbf{x}, \mathbf{y}) = K^{n-1}(\mathbf{x}, \mathbf{y}) K(\mathbf{x}, \mathbf{y}) \succeq 0$$

- and  $\exp(K(\mathbf{x}, \mathbf{y}))$  is also a valid kernel since

$$e^{K(\mathbf{x}, \mathbf{y})} = \sum_{i=1}^{\infty} \frac{1}{i!} K^i(\mathbf{x}, \mathbf{y}) = 1 + K(\mathbf{x}, \mathbf{y}) + \frac{1}{2} K^2(\mathbf{x}, \mathbf{y}) + \dots$$

but each term in the sum is a kernel

## RBF Kernel

- Now  $\mathbf{x}^\top \mathbf{y} = \langle \mathbf{x}, \mathbf{y} \rangle$  is a valid kernel because it is an inner product of functions  $\phi(\mathbf{x}) = \mathbf{x}$
- For  $\gamma > 0$  we have  $2\gamma \mathbf{x}^\top \mathbf{y} \succeq 0$
- Thus  $\exp(2\gamma \mathbf{x}^\top \mathbf{y}) \succeq 0$
- If  $K(\mathbf{x}, \mathbf{y}) \succeq 0$  then  $g(\mathbf{x}) K(\mathbf{x}, \mathbf{y}) g(\mathbf{y}) \succeq 0$

$$\int f(\mathbf{x}) g(\mathbf{x}) K(\mathbf{x}, \mathbf{y}) g(\mathbf{y}) f(\mathbf{y}) d\mathbf{x} d\mathbf{y} = \int h(\mathbf{x}) K(\mathbf{x}, \mathbf{y}) h(\mathbf{y}) d\mathbf{x} d\mathbf{y} \geq 0$$

where  $f(\mathbf{x}) g(\mathbf{x}) = h(\mathbf{x})$

$$e^{-\gamma \mathbf{x}^\top \mathbf{y}} e^{2\gamma \mathbf{x}^\top \mathbf{y}} e^{-\gamma \mathbf{y}^\top \mathbf{y}} = e^{-\gamma \|\mathbf{x} - \mathbf{y}\|^2} \succeq 0$$

## 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 = statistics  S3(s) = {sta,tat,ati,tis,ist,st,i,tic,ics}
t = computation S3(t) = {com,omp,mpu,put,uta,tat,ati,tio,ion}
```

$$K(s, t) = 2 ("tat" \text{ and } "ati")$$

## All Subsequences Kernel

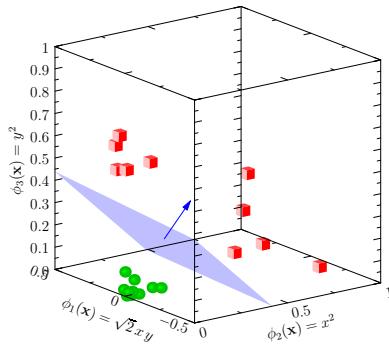
- A more sophisticated kernel is to count all of the common subsequences that occur in two documents
- Naively this would take an exponential amount of time to compute
- Using clever dynamic-programming techniques this can be done relatively efficiently
- This can even be extended to include sub-sequence matches with possible gaps between words

- 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)

- 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

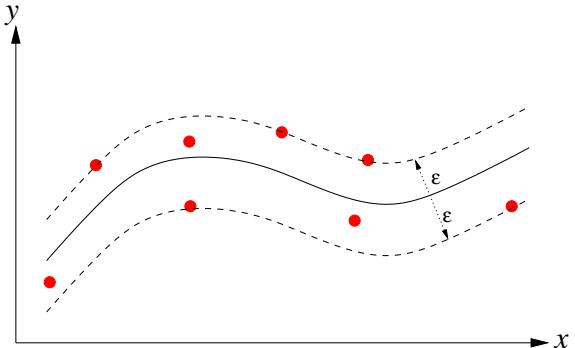
## Outline

1. The Kernel Trick
2. Positive Semi-Definite Kernels
3. Kernel Properties
4. Beyond Classification



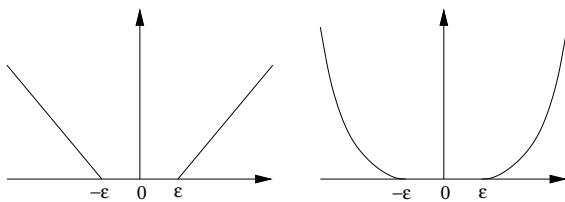
## Regression with Margins

- 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

$$\min_{\mathbf{w}} \lambda \|\mathbf{w}\|^2 + \sum_{i=1}^m (y_i - \mathbf{w}^\top \phi(\mathbf{x}_i))^2$$

- the  $\|\mathbf{w}\|^2$  is a regularisation term
- By assuming  $\mathbf{w} = \sum_i \alpha_i \phi(\mathbf{x}_i)$  we obtain a quadratic equation for the  $\alpha_i$ 's which we can solve

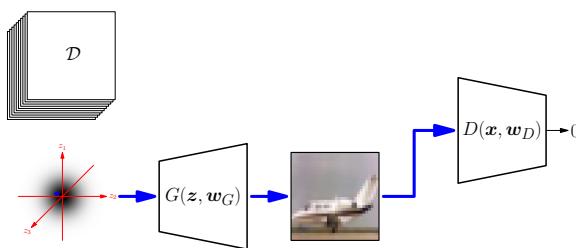
## Kernel Methods

- 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

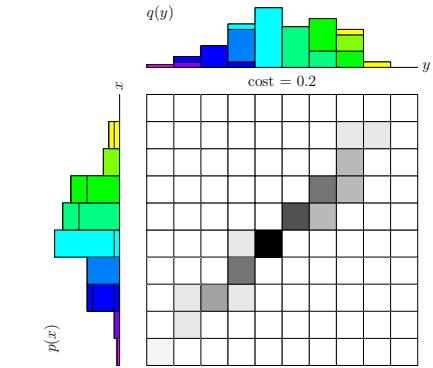
## Summary

- SVMs require a positive definite kernel function
- These can be built from simpler functions
- There was a cottage industry of people creating new kernels for different applications
- SVMs are just one example of a host of machine learning algorithms
  - use the kernel trick
  - often use linear constraints
  - tend to be convex optimisation problems

### Wasserstein GANs



*GANs, Wasserstein distance, Duality, WGANs*



### 1. GANs

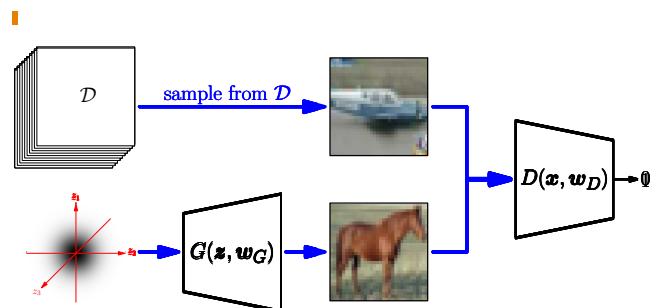
### 2. Wasserstein Distance

### 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

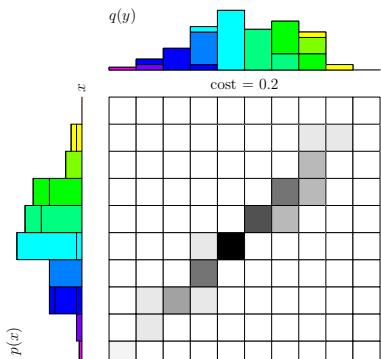
- GANs are notoriously difficult to train
- The generator and discriminator training can decouple
- Often the discriminator becomes too good at correctly identifying the generated images
- Then there can be little gradient information to help the generator as every small change in parameters doesn't 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



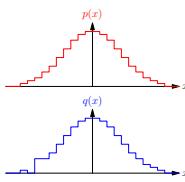
## Measuring Distances Between Distributions

- 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
- $$KL(p\|q) = \int p(x) \log \left( \frac{p(x)}{q(x)} \right) dx$$
- This is very commonly used in ML (e.g. VAEs, Variational Approximation)

## Trouble with KL

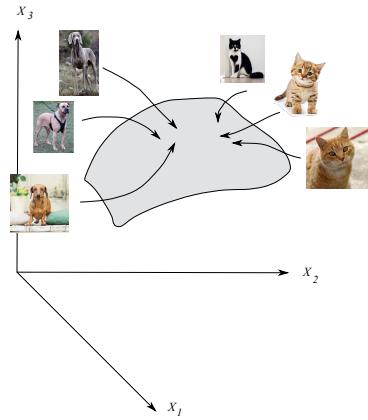
- KL-divergences are non-negative quantities that are minimised when the two probability distributions are the same
- They are not distances (they aren't symmetric and they don't satisfy the triangular inequality)

We don't really care about this, but what we do care about is that if  $q(x) = 0$  when  $p(x) \neq 0$  then  $\log\left(\frac{p(x)}{q(x)}\right)$  diverges

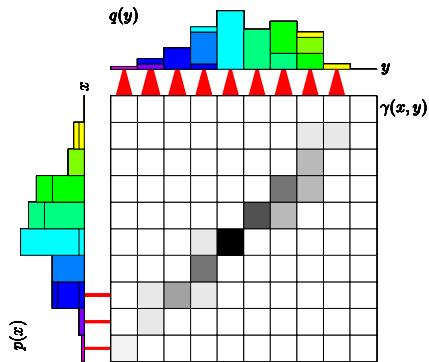


- We can therefore have distributions that seem very similar but their KL-divergence is huge (or infinite)

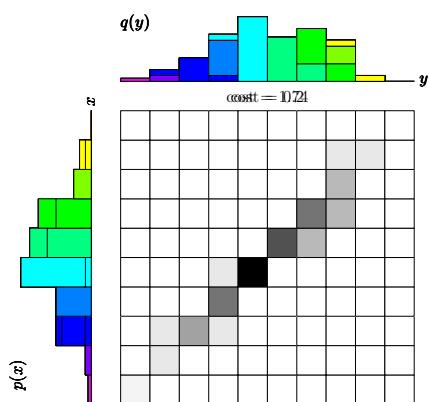
## High Probability Manifold



## Transportation Policy

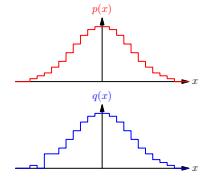


## Transportation Cost



## 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 measures distance in a very natural way so that distributions that are close have a small Wasserstein distance

- Although this seems contrived if our probability distribution represents the probability of a  $128 \times 128$  matrix of real valued triples representing an image of dog, then it is easy to imagine that the Wasserstein distance may be more benign than the KL-divergence

## Transportation Policy

- But how do we formalise the Wasserstein distance?
- A good place to start is to define a transportation policy  $\gamma(x,y)$  with

$$\int \gamma(x,y) dy = p(x) \quad \int \gamma(x,y) dx = q(y)$$

- This looks like a joint probability distribution, but we interpret  $\gamma(x,y)$  as the amount of probability mass/density that we transfer from  $p(x)$  to  $q(y)$

## The Cost of Transport

- We want to choose the transportation policy that minimises the amount of probability mass we need to move
- Let  $d(x,y) = \|x - y\|$  be a distance measure then the cost of a transportation policy is

$$C(\gamma) = \int \int d(x,y) \gamma(x,y) dx dy = \mathbb{E}_{\gamma}[d(x,y)]$$

where we interpret  $\gamma(x,y)$  as a probability distribution

- Usually we take  $d(x,y)$  to be the Euclidean distance, but we can choose any distance

## The Wasserstein Distance

- The Wasserstein distance  $W(p,q)$  between two probability distributions is defined as

$$W(p,q) = \min_{\gamma \in \Lambda(p,q)} \mathbb{E}_{\gamma}[d(x,y)]$$

- Where  $\Lambda(p,q)$  is the set of joint distributions  $\gamma(x,y)$  such that

$$\int \gamma(x,y) dy = p(x) \quad \int \gamma(x,y) dx = q(y)$$

## Computing the Wasserstein Distance

## Constraints

- To compute the Wasserstein distance we have to solve a minimisation task!
- This looks nasty, but it is a (continuous) linear programming problem!
- Suppose  $p$  and  $q$  were discrete distribution (i.e.  $\mathbf{x}$  and  $\mathbf{y}$  only take discrete points)
- Then we could treat each value of  $\gamma(\mathbf{x}, \mathbf{y})$  as an element of a vector  $\boldsymbol{\gamma}$  and each value of  $d(\mathbf{x}, \mathbf{y})$  as an element of a vector  $\mathbf{D}$
- Our objective is to choose  $\boldsymbol{\gamma}$  to minimise  $\mathbf{D}^T \boldsymbol{\gamma}$

$$\sum_j \gamma(\mathbf{x}_i, \mathbf{y}_j) = p(\mathbf{x}_i)$$

$$\sum_i \gamma(\mathbf{x}_i, \mathbf{y}_j) = q(\mathbf{y}_j)$$

$$\mathbf{A}\boldsymbol{\gamma} = \mathbf{P}$$

$$\begin{pmatrix} 1 & 1 & \dots & 1 & 0 & 0 & \dots & 0 & \dots & \dots & 0 & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 & \dots & \dots & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & \dots & \dots & 1 & 1 & \dots & 1 \\ 1 & 0 & \dots & 0 & 1 & 0 & \dots & 0 & \dots & \dots & 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 & 0 & 1 & \dots & 0 & \dots & \dots & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 1 & 0 & 0 & \dots & 1 & \dots & \dots & 0 & 0 & \dots & 1 \end{pmatrix} = \begin{pmatrix} \gamma(x_1, y_1) \\ \gamma(x_2, y_1) \\ \vdots \\ \gamma(x_n, y_1) \\ \gamma(x_1, y_2) \\ \gamma(x_2, y_2) \\ \vdots \\ \gamma(x_n, y_2) \\ \vdots \\ \gamma(x_1, y_n) \\ \gamma(x_2, y_n) \\ \vdots \\ \gamma(x_n, y_n) \end{pmatrix} = \begin{pmatrix} q(y_1) \\ q(y_2) \\ \vdots \\ q(y_n) \\ p(x_1) \\ p(x_2) \\ \vdots \\ p(x_n) \end{pmatrix}$$

## Lagrange Formulation

- For discrete distributions

$$\min_{\boldsymbol{\gamma}} \mathbf{D}^T \boldsymbol{\gamma}$$

subject to  $\mathbf{A}\boldsymbol{\gamma} = \mathbf{P}, \boldsymbol{\gamma} \geq 0$

- Writing the Lagrangian

$$\mathcal{L}(\boldsymbol{\gamma}, \boldsymbol{\alpha}) = \mathbf{D}^T \boldsymbol{\gamma} - \boldsymbol{\alpha}^T (\mathbf{A}^T \boldsymbol{\gamma} - \mathbf{P})$$

where  $\boldsymbol{\alpha}$  is a vector of Lagrange multipliers

- The solution to the discrete optimisation problem is given by

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

## Explicit Form

## Continuous Form

- We can write a Lagrangian for the original problem

$$\mathcal{L} = \sum_{i,j} d(\mathbf{x}_i, \mathbf{y}_j) \gamma(\mathbf{x}_i, \mathbf{y}_j) - \sum_i \alpha(\mathbf{x}_i) \left( \sum_j \gamma(\mathbf{x}_i, \mathbf{y}_j) - p(\mathbf{x}_i) \right) - \sum_j \beta(\mathbf{y}_j) \left( \sum_i \gamma(\mathbf{x}_i, \mathbf{y}_j) - q(\mathbf{y}_j) \right)$$

subject to  $\gamma(\mathbf{x}_i, \mathbf{y}_j) \geq 0$  where  $\alpha(\mathbf{x}_i)$  and  $\beta(\mathbf{y}_j)$  are Lagrange multipliers (they are components of  $\boldsymbol{\alpha}$ )

- Rearranging

$$\mathcal{L} = \sum_i \alpha(\mathbf{x}_i) p(\mathbf{x}_i) + \sum_j \beta(\mathbf{y}_j) q(\mathbf{y}_j) - \sum_{i,j} \gamma(\mathbf{x}_i, \mathbf{y}_j) (\alpha(\mathbf{x}_i) + \beta(\mathbf{y}_j) - d(\mathbf{x}_i, \mathbf{y}_j))$$

- This is equivalent to maximising  $\sum_i \alpha(\mathbf{x}_i) p(\mathbf{x}_i) + \sum_j \beta(\mathbf{y}_j) q(\mathbf{y}_j)$ , subject to

$$\forall i, j \quad \alpha(\mathbf{x}_i) + \beta(\mathbf{y}_j) \leq d(\mathbf{x}_i, \mathbf{y}_j)$$

- We can write a Lagrangian for the continuous problem

$$\mathcal{L} = \iint d(\mathbf{x}, \mathbf{y}) \gamma(\mathbf{x}, \mathbf{y}) d\mathbf{x} d\mathbf{y} - \int \alpha(\mathbf{x}) \left( \int \gamma(\mathbf{x}, \mathbf{y}) d\mathbf{y} - p(\mathbf{x}) \right) d\mathbf{x} - \int \beta(\mathbf{y}) \left( \int \gamma(\mathbf{x}, \mathbf{y}) d\mathbf{x} - q(\mathbf{y}) \right) d\mathbf{y}$$

subject to  $\gamma(\mathbf{x}, \mathbf{y}) \geq 0$  where  $\alpha(\mathbf{x})$  and  $\beta(\mathbf{y})$  are Lagrange multiplier functions

- Rearranging

$$\mathcal{L} = \int \alpha(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} + \int \beta(\mathbf{y}) q(\mathbf{y}) d\mathbf{y} - \iint \gamma(\mathbf{x}, \mathbf{y}) (\alpha(\mathbf{x}) + \beta(\mathbf{y}) - d(\mathbf{x}, \mathbf{y})) d\mathbf{x} d\mathbf{y}$$

- This is equivalent to maximising  $\int \alpha(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} + \int \beta(\mathbf{y}) q(\mathbf{y}) d\mathbf{y}$ , subject to

$$\alpha(\mathbf{x}) + \beta(\mathbf{y}) \leq d(\mathbf{x}, \mathbf{y})$$

## Dual Form Constraint

## Dual Form

- We note that  $\alpha(\mathbf{x}) + \beta(\mathbf{y}) \leq d(\mathbf{x}, \mathbf{y})$  for all  $\mathbf{x}$  and  $\mathbf{y}$
- This has to be true when  $\mathbf{x} = \mathbf{y}$  so that

$$\alpha(\mathbf{x}) + \beta(\mathbf{x}) \leq d(\mathbf{x}, \mathbf{x}) = 0$$

- So  $\beta(\mathbf{x}) = -\alpha(\mathbf{x}) - \epsilon(\mathbf{x})$  where  $\epsilon(\mathbf{x}) \geq 0$

- But want to maximise

$$\int \alpha(\mathbf{x}) p(\mathbf{x}) d\mathbf{x} + \int \beta(\mathbf{y}) q(\mathbf{y}) d\mathbf{y} = \int \alpha(\mathbf{x}) (p(\mathbf{x}) - q(\mathbf{x})) d\mathbf{x} - \int q(\mathbf{x}) \epsilon(\mathbf{x}) d\mathbf{x}$$

- This is maximised when  $\epsilon(\mathbf{x}) = 0$  i.e.  $\beta(\mathbf{x}) = -\alpha(\mathbf{x})$

- Thus the dual problem is to find a function  $\alpha(\mathbf{x})$ —or a vector of functions  $(\alpha(\mathbf{x}_i)|i)$ —that maximises

$$\int \alpha(\mathbf{x}) (p(\mathbf{x}) - q(\mathbf{x})) d\mathbf{x}$$

- Subject to the constraint

$$\alpha(\mathbf{x}) - \alpha(\mathbf{y}) \leq d(\mathbf{x}, \mathbf{y}) = \|\mathbf{x} - \mathbf{y}\|$$

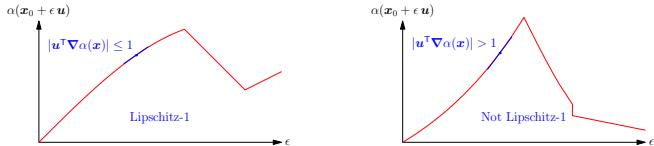
- This is a continuity constraint on the Lagrange multiplier function  $\alpha(\mathbf{x})$  known as Lipschitz-1

## Lipschitz-1 Functions

- We note for a Lipschitz-1 function and any unit vector  $\mathbf{u}$

$$\mathbf{u}^\top \nabla \alpha(\mathbf{x}) = \lim_{\epsilon \rightarrow 0} \frac{\alpha(\mathbf{x}) - \alpha(\mathbf{x} + \epsilon \mathbf{u})}{\epsilon} \leq \frac{\epsilon}{\epsilon} = 1$$

- 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  $\|\nabla \alpha(\mathbf{x})\| \leq 1$  everywhere)



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

25

## Calculating the Wasserstein Distance

- To recall the big picture we want to compute the Wasserstein distance

$$W(p, q) = \min_{\gamma \in \Lambda(p, q)} \mathbb{E}_{\gamma}[d(\mathbf{x}, \mathbf{y})]$$

- For high dimensional objects  $\gamma(\mathbf{x}, \mathbf{y})$  would be a huge object to approximate
- Instead we can compute the Wasserstein distance in the dual formulation

$$W(p, q) = \max_{\alpha(\mathbf{x})} \int \alpha(\mathbf{x}) (p(\mathbf{x}) - q(\mathbf{x})) d\mathbf{x} = \max_{\alpha} \mathbb{E}_p[\alpha(\mathbf{X})] - \mathbb{E}_q[\alpha(\mathbf{X})]$$

subject to the constraint that  $\alpha(\mathbf{x})$  is a Lipschitz-1 function

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

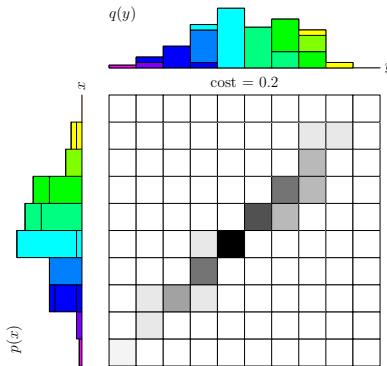
26

## Outline

### 1. GANs

### 2. Wasserstein Distance

### 3. Wasserstein GANs



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

27

## Back to GANs

- What has this got do with GANs?
- Suppose we want to minimise the distance between the distribution  $p(\mathbf{x})$  of real images (of which  $\mathcal{D}$  are samples) and the distribution  $q(\mathbf{x})$  of images drawn from a generator
- We can use a normal GAN generator,  $G(z, w_G)$ , that generates an image when given a random variable  $z \sim \mathcal{N}(0, I)$
- To do this we choose the weights,  $w_G$  of the generator to minimise

$$W(p, q) = \max_{\alpha(\mathbf{x})} (\mathbb{E}_{\mathbf{x} \sim p}[\alpha(\mathbf{x})] - \mathbb{E}_{\mathbf{x} \sim q}[\alpha(\mathbf{x})])$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

28

## Estimating Expectations

- Although we can't compute  $\mathbb{E}_p[\alpha(\mathbf{x})]$  and  $\mathbb{E}_q[\alpha(\mathbf{x})]$  exactly, we can estimate them from samples

$$\mathbb{E}_p[\alpha(\mathbf{x})] \approx \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \alpha(\mathbf{x}), \quad \mathbb{E}_q[\alpha(\mathbf{x})] \approx \frac{1}{n} \sum_{i=1}^n \alpha(G(z_i, w_G))$$

- where  $\mathcal{B} \subset \mathcal{D}$  is a minibatch of true images and  $z_i \sim \mathcal{N}(0, I)$
- From this we can choose  $w_G$  to minimise

$$C = \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \alpha(\mathbf{x}) - \frac{1}{n} \sum_{i=1}^n \alpha(G(z_i, w_G))$$

Adam Prügel-Bennett

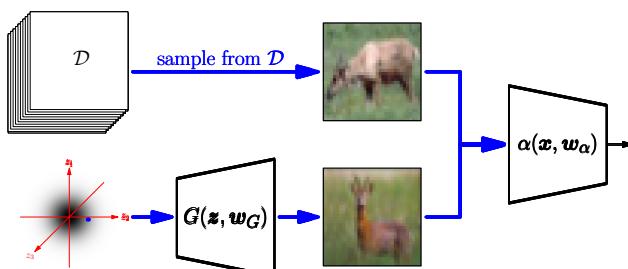
COMP6208 Advanced Machine Learning

29

## Wasserstein GANs

1

$$\max_{w_\alpha} \min_{w_G} \frac{1}{|\mathcal{B}|} \sum_{\mathbf{x} \in \mathcal{B}} \alpha(\mathbf{x}, w_\alpha) - \frac{1}{n} \sum_{i=1}^n \alpha(G(z_i, w_G), w_\alpha)$$



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

31

## 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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

32

### Probability

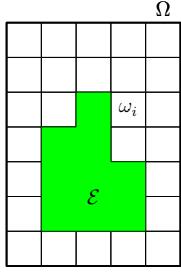
$Y = g(X)$				$\Omega$
$y_{13} = g(x_{13})$	$y_{14} = g(x_{14})$	$y_{15} = g(x_{15})$	$y_{16} = g(x_{16})$	
$y_9 = g(x_9)$	$y_{10} = g(x_{10})$	$y_{11} = g(x_{11})$	$y_{12} = g(x_{12})$	
$y_5 = g(x_5)$	$y_6 = g(x_6)$	$y_7 = g(x_7)$	$y_8 = g(x_8)$	
$y_1 = g(x_1)$	$y_2 = g(x_2)$	$y_3 = g(x_3)$	$y_4 = g(x_4)$	

$x_{31}$	$x_{32}$	$x_{33}$	$x_{34}$	$x_{35}$
$x_{26}$	$x_{27}$	$x_{28}$	$x_{29}$	$x_{30}$
$x_{21}$	$x_{22}$	$x_{23}$	$x_{24}$	$x_{25}$
$x_{16}$	$x_{17}$	$x_{18}$	$x_{19}$	$x_{20}$
$x_{11}$	$x_{12}$	$x_{13}$	$x_{14}$	$x_{15}$
$x_6$	$x_7$	$x_8$	$x_9$	$x_{10}$
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$

Probability, Random Variables, Expectations

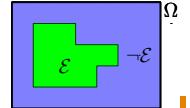
### Modelling Uncertainty

- To model a world with uncertainty we consider some set of elementary events or outcomes  $\Omega$
- For the outcome of rolling a dice  $\Omega = \{1, 2, 3, 4, 5, 6\}$
- The elementary events  $\omega_i$  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  $\mathcal{E} = \{2, 4, 6\}$



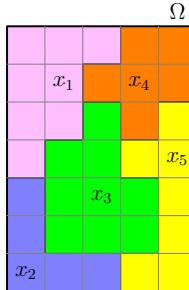
### Probabilities

- We attribute a probability,  $\mathbb{P}(\mathcal{E})$ , to an event,  $\mathcal{E}$ , with the requirements
  - $0 \leq \mathbb{P}(\mathcal{E}) \leq 1$
  - $\mathbb{P}(\mathcal{E}) + \mathbb{P}(\neg \mathcal{E}) = 1$  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  $\mathbb{P}(\text{Pass COMP6208 exam})$  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  $\Omega$  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  $(\mathbb{P}(X = x) | x \in \mathcal{X}) = (\mathbb{P}(X = x_1), \mathbb{P}(X = x_2), \dots, \mathbb{P}(X = x_n))$  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

$Y = g(X)$				$\Omega$
$y_{13} = g(x_{13})$	$y_{14} = g(x_{14})$	$y_{15} = g(x_{15})$	$y_{16} = g(x_{16})$	
$y_9 = g(x_9)$	$y_{10} = g(x_{10})$	$y_{11} = g(x_{11})$	$y_{12} = g(x_{12})$	
$y_5 = g(x_5)$	$y_6 = g(x_6)$	$y_7 = g(x_7)$	$y_8 = g(x_8)$	
$y_1 = g(x_1)$	$y_2 = g(x_2)$	$y_3 = g(x_3)$	$y_4 = g(x_4)$	

### Continuous Spaces

- If the space of elementary events is continuous (e.g. for darts  $x = (x, y)$ ) then  $\mathbb{P}(X = x) = 0$
  - But if we consider a region,  $\mathcal{R}$ , then we can assign a probability to landing in the region  $\mathbb{P}(X \in \mathcal{R})$
  - It is useful to work with probability densities function (PDF)
- $$f_X(x) = \lim_{\epsilon \rightarrow 0} \frac{\mathbb{P}(X \in \mathcal{B}(x, \epsilon))}{|\mathcal{B}(x, \epsilon)|}$$
- where  $\mathcal{B}(x, \epsilon)$  is a ball of radius  $\epsilon$  around the point  $x$  and  $|\mathcal{B}(x, \epsilon)|$  is the volume of the ball
- If we make a change of variable the volume  $|\mathcal{B}(x, \epsilon)|$  might change so  $f_X(x)$  will change

## Change of Variables

- Consider a region  $\mathcal{R}$ —we can describe this using different coordinate systems  $x$  or  $y = g(x)$
- But

$$\mathbb{P}(X \in \mathcal{R}) = \int_{\mathcal{R}} f_X(x) dx = \mathbb{P}(Y \in \mathcal{R}) = \int_{\mathcal{R}} f_Y(y) dy$$

- As this is true for any region  $\mathcal{R}$ :  $f_X(x)|dx| = f_Y(y)|dy|$

- Or

$$f_X(x) = f_Y(y) \left| \frac{dy}{dx} \right| = f_Y(g(x))|g'(x)|$$

- 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  $\mathbf{x} \rightarrow \mathbf{y}(\mathbf{x})$  (which can be seen as a change of random variables  $\mathbf{X} \rightarrow \mathbf{Y}(\mathbf{X})$ )

- Then

$$f_{\mathbf{X}}(\mathbf{x}) = f_{\mathbf{Y}}(\mathbf{y}) |\det(\mathbf{J})|$$

where  $\mathbf{J}$  is the Jacobian matrix

$$\mathbf{J} = \begin{pmatrix} \frac{\partial y_1}{\partial x_1} & \frac{\partial y_1}{\partial x_2} & \dots & \frac{\partial y_1}{\partial x_n} \\ \frac{\partial y_2}{\partial x_1} & \frac{\partial y_2}{\partial x_2} & \dots & \frac{\partial y_2}{\partial x_n} \\ \vdots & \vdots & \dots & \vdots \\ \frac{\partial y_n}{\partial x_1} & \frac{\partial y_n}{\partial x_2} & \dots & \frac{\partial y_n}{\partial x_n} \end{pmatrix}$$

- 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

$$f_X(x) = \lim_{\delta x \rightarrow 0} \frac{\mathbb{P}(x \leq X < x + \delta x)}{\delta x}$$

- We can think of  $f_X(x)\delta x$  as  $\mathbb{P}(x \leq X < x + \delta x)$
- Note that  $f_X(x)\delta x \leq 1$

## Cumulative Distribution Functions

- We can define the **cumulative distribution function** (CDF)

$$F_X(x) = \mathbb{P}(X \leq x) = \begin{cases} \sum_{i: x_i \leq x} \mathbb{P}(X = x_i) \\ \int_{-\infty}^x f_X(y) dy \end{cases}$$

- This is a function that goes from 0 to 1 as  $x$  goes from  $-\infty$  to  $\infty$
- We note that for continuous random variables

$$f_X(x) = \frac{dF_X(x)}{dx}$$

## Outline

- Random Variables
- Expectations
- Calculus of Probabilities

$\Omega$				
$x_{31}$	$x_{32}$	$x_{33}$	$x_{34}$	$x_{35}$
$x_{26}$	$x_{27}$	$x_{28}$	$x_{29}$	$x_{30}$
$x_{21}$	$x_{22}$	$x_{23}$	$x_{24}$	$x_{25}$
$x_{16}$	$x_{17}$	$x_{18}$	$x_{19}$	$x_{20}$
$x_{11}$	$x_{12}$	$x_{13}$	$x_{14}$	$x_{15}$
$x_6$	$x_7$	$x_8$	$x_9$	$x_{10}$
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$

## Expectation

- We can define the expectation of  $\mathbf{Y} = g(\mathbf{X})$  as

$$\mathbb{E}_{\mathbf{X}}[g(\mathbf{X})] = \begin{cases} \sum_{\mathbf{x} \in \mathcal{X}} g(\mathbf{x}) \mathbb{P}(\mathbf{X} = \mathbf{x}) \\ \int g(\mathbf{x}) f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} \end{cases}$$

- The expectation of a constant  $c$  is

$$\mathbb{E}_{\mathbf{X}}[c] = \begin{cases} \sum_{\mathbf{x} \in \mathcal{X}} c \mathbb{P}(\mathbf{X} = \mathbf{x}) = c \sum_{\mathbf{x} \in \mathcal{X}} \mathbb{P}(\mathbf{X} = \mathbf{x}) = c \\ \int c f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} = c \int f_{\mathbf{X}}(\mathbf{x}) d\mathbf{x} = c \end{cases}$$

- Note  $\mathbb{E}_{\mathbf{X}}[\mathbb{E}_{\mathbf{X}}[g(\mathbf{X})]] = \mathbb{E}_{\mathbf{X}}[g(\mathbf{X})]$

## Linearity of Expectation

- Because sums and integrals are linear operators

$$\begin{aligned} \sum_i (ax_i + by_i) &= a \left( \sum_i x_i \right) + b \left( \sum_i y_i \right) \\ \int (af(\mathbf{x}) + bg(\mathbf{x})) d\mathbf{x} &= a \left( \int f(\mathbf{x}) d\mathbf{x} \right) + b \left( \int g(\mathbf{x}) d\mathbf{x} \right) \end{aligned}$$

then expectations are linear

$$\mathbb{E}[aX + bY] = a\mathbb{E}[X] + b\mathbb{E}[Y]$$

- Beware usually  $\mathbb{E}[XY] \neq \mathbb{E}[X]\mathbb{E}[Y]$  (unless  $X$  and  $Y$  are independent)

## Indicator Functions

- An indicator function has the property

$$[\![\text{predicate}]\!] = \begin{cases} 1 & \text{if predicate is True} \\ 0 & \text{if predicate is False} \end{cases}$$

(sometimes written  $I_A(x)$  where  $A(x)$  is the predicate)

- We can obtain probabilities from expectations

$$\mathbb{P}(\text{predicate}) = \mathbb{E}[\![\text{predicate}]\!]$$

- E.g. The CDF is given by

$$F_X(x) = \mathbb{P}(X \leq x) = \mathbb{E}[\![X \leq x]\!]$$

1. Random Variables
2. Expectations
3. Calculus of Probabilities

$\Omega$				
$x_{31}$	$x_{32}$	$x_{33}$	$x_{34}$	$x_{35}$
$x_{26}$	$x_{27}$	$x_{28}$	$x_{29}$	$x_{30}$
$x_{21}$	$x_{22}$	$x_{23}$	$x_{24}$	$x_{25}$
$x_{16}$	$x_{17}$	$x_{18}$	$x_{19}$	$x_{20}$
$x_{11}$	$x_{12}$	$x_{13}$	$x_{14}$	$x_{15}$
$x_6$	$x_7$	$x_8$	$x_9$	$x_{10}$
$x_1$	$x_2$	$x_3$	$x_4$	$x_5$

- 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

$$\mathbb{P}(X | Y = y) = \frac{\mathbb{P}(X, Y = y)}{\mathbb{P}(Y = y)}$$

- 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  $\mathbb{P}(A, B | X, Y)$  means the probability of random variables  $A$  and  $B$  given that  $X$  and  $Y$  take particular values

## Causality

- Conditional probabilities does not imply causality
- We might have causal relationships

$$\mathbb{P}(\text{pass} | \text{study}) = 0.9 \quad \mathbb{P}(\text{pass} | \neg\text{study}) = 0.2$$

- But if we know  $\mathbb{P}(\text{study}) = 0.8$  then we can compute

$$\mathbb{P}(\text{pass, study}) = \mathbb{P}(\text{pass} | \text{study})\mathbb{P}(\text{study}) = 0.9 \times 0.8 = 0.72$$

$$\mathbb{P}(\text{pass}, \neg\text{study}) = \mathbb{P}(\text{pass} | \neg\text{study})\mathbb{P}(\neg\text{study}) = 0.2 \times 0.2 = 0.04$$

and

$$\begin{aligned} \mathbb{P}(\text{study} | \text{pass}) &= \frac{\mathbb{P}(\text{pass, study})}{\mathbb{P}(\text{pass})} \\ &= \frac{\mathbb{P}(\text{pass, study})}{\mathbb{P}(\text{pass, study}) + \mathbb{P}(\text{pass}, \neg\text{study})} = \frac{0.72}{0.72 + 0.04} \approx 0.947 \end{aligned}$$

## 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

$$\mathbb{P}(X|Y) = \mathbb{P}(X) \quad \mathbb{P}(Y|X) = \mathbb{P}(Y)$$

- Probabilistic independence implies a mathematical co-incident not necessarily causal independence
- However causal independence implies probabilistic independence
- If  $X \in \{0, 1\}$  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  $\mathcal{D} = (X_1, X_2, \dots, X_m)$  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  $\mathcal{S} = (W_1, W_2, \dots, W_m)$  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 your revision schedule you can write down your belief

$$\mathbb{P}(K(d) | K(d-1), K(d-2), \dots, K(1))$$

- But a very reasonable model is

$$\mathbb{P}(K(d) | K(d-1), K(d-2), \dots, K(1)) = \mathbb{P}(K(d) | K(d-1))$$

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

## Bayesian Inference



Bayes, Conjugate Priors, Uninformative Priors



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 than general purpose black boxes

## Revision on Bayes

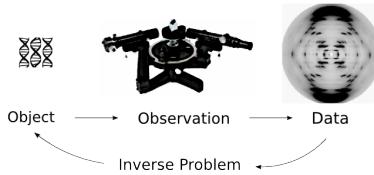
- Bayes' rule

$$\mathbb{P}(\mathcal{H}_i|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\mathcal{H}_i)\mathbb{P}(\mathcal{H}_i)}{\mathbb{P}(\mathcal{D})}$$

- ★  $\mathbb{P}(\mathcal{H}_i|\mathcal{D})$  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  $\mathbb{P}(\mathcal{D} | M_1)$  or  $\mathbb{P}(\mathcal{D} | M_2)$
- 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

$$f_X(x) = \lim_{\delta x \rightarrow 0} \frac{\mathbb{P}(x \leq X < x + \delta x)}{\delta x}$$

- Note that densities are non-negative, but can be greater than 1 (they are not probabilities)
- However

$$\mathbb{P}(a \leq X \leq b) = \int_a^b f_X(x) dx$$

is a probability and is less than or equal to 1

## Densities and Bayes

## Practical Bayesian Inference

- 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  $\delta x$  and taking the limit  $\delta x \rightarrow 0$

$$f_{X|Y}(x|Y) = \frac{\mathbb{P}(Y|x)f_X(x)}{\mathbb{P}(Y)}$$

- 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

$$f_{X|Y}(x|y) = \frac{f_{Y|X}(y|x)f_X(x)}{f_Y(y)}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

9

- Often consider learning parameters  $\theta$

$$p(\theta|\mathcal{D}) = \frac{p(\mathcal{D}|\theta)p(\theta)}{p(\mathcal{D})}$$

- This can be hard for large data sets as the posterior,  $p(\theta|\mathcal{D})$ , 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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

10

## Outline



1. Bayes' Rule
2. Conjugate Priors
3. Uninformative Priors

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

11

## 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

$$\mathbb{P}(X_i|p) = p^{X_i}(1-p)^{1-X_i} = \begin{cases} p & \text{if } X_i = 1 \\ (1-p) & \text{if } X_i = 0 \end{cases}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

12

## 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

$$B(a,b) = \int_0^1 p^{a-1}(1-p)^{b-1} dp = \frac{\Gamma(a)\Gamma(b)}{\Gamma(a+b)}$$

- This is a useful function for modelling the distribution of a random variable in the range 0 to 1

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

13

## 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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

14

## 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

$$\begin{aligned} \mathbb{P}(\mathcal{D}|p) &= \prod_{i=1}^n p^{X_i}(1-p)^{1-X_i} \\ &= p^{X_1}(1-p)^{1-X_1}p^{X_2}(1-p)^{1-X_2}\dots p^{X_n}(1-p)^{1-X_n} \\ &= p^{\sum_i X_i}(1-p)^{\sum_i (1-X_i)} = p^s(1-p)^{n-s} \end{aligned}$$

$$s = \sum_i X_i \text{ (number of successes/heads)}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

15

## 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

$$\begin{aligned} \mathbb{P}(\mathcal{D}) &= \int_0^1 \mathbb{P}(\mathcal{D}|p)f(p) dp = \int_0^1 \frac{p^{s+a_0-1}(1-p)^{n-s+b_0-1}}{B(a_0,b_0)} dp \\ &= \frac{B(s+a_0, n-s+b_0)}{B(a_0,b_0)} \end{aligned}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

16

## Conjugate Priors

- The posterior distribution is Beta distribution

$$f(p|\mathcal{D}) = \frac{p^{s+a_0-1}(1-p)^{n-s+b_0-1}}{B(s+a_0, n-s+b_0)} = \text{Beta}(p|s+a_0, n-s+b_0)$$

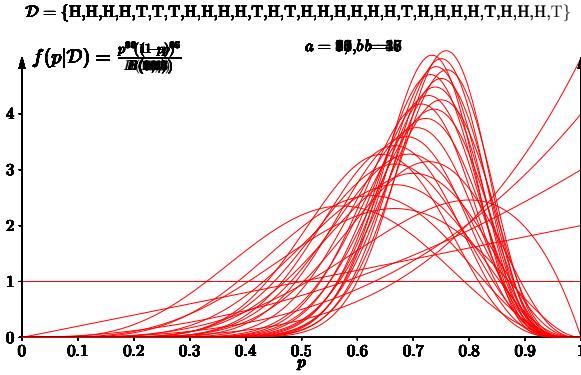
- Something rather nice happened

- 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**

## Example ( $p=0.7$ )



## 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

$$\mathbb{P}(N) = \text{Pois}(N|\mu) = \frac{\mu^N}{N!} e^{-\mu}$$

- $\mu$  is the rate of traffic per hour which we want to infer from observation taken on different days

## Posterior

- The posterior after seeing the first piece of data is

$$\begin{aligned} p(\mu|N_1) &\propto \mathbb{P}(N_1|\mu)p(\mu) \\ &\propto \frac{\mu^{N_1}}{N_1!} e^{-\mu} \mu^{a_0-1} e^{-b_0\mu} \\ &\propto \mu^{N_1+a_0-1} e^{-(b_0+1)\mu} \end{aligned}$$

- The posterior is also a Gamma distribution  $\Gamma(\mu|a_1, b_1)$  with  $a_1 = a_0 + N_1$ ,  $b_1 = b_0 + 1$

## Incremental Updating

- For independent data we can update incrementally  
 $\mathcal{D} = (X_1, X_2, \dots, X_n)$

$$\begin{aligned} f(p|X_1) &= \frac{\mathbb{P}(X_1|p)f(p)}{\mathbb{P}(X_1)} \\ f(p|X_1, X_2) &= \frac{\mathbb{P}(X_2|p)f(p|X_1)}{\mathbb{P}(X_2)} \\ &\vdots = \vdots \\ f(p|X_1, X_2, \dots, X_n) &= \frac{\mathbb{P}(X_n|p)f(p|X_1, \dots, X_{n-1})}{\mathbb{P}(X_n)} \end{aligned}$$

- The posterior becomes the prior for the next piece of data
- For our problem the posterior is always Beta distributed

## 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

$$\sqrt{\frac{ab}{(a+b)^2(a+b+1)}} = 0.078$$

## Using Bayes

- Let us assume a Gamma distributed prior

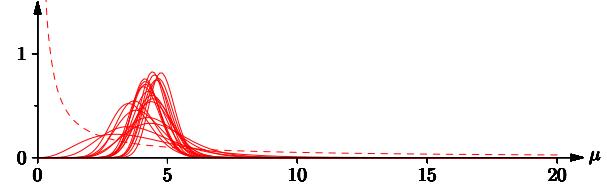
$$p(\mu) = \Gamma(\mu|a_0, b_0) = \frac{b_0^{a_0} \mu^{a_0-1} e^{-b_0\mu}}{\Gamma(a)}$$

- We will assume that we know nothing. The uninformative prior is  $a_0 = b_0 = 0$
- The data is  $\mathcal{D} = \{N_1, N_2, \dots, N_n\}$
- The likelihood is  $\text{Pois}(N_i|\mu)$

## Example ( $\mu = 5$ )

$$\mathcal{D} = \{4, 4, 6, 4, 2, 2, 5, 9, 5, 4, 3, 2, 5, 4, 4, 11, 6, 2, 3, 11\}$$

$$p(\mu|N_1, N_2, \dots, N_n) = \frac{\mu^{96} e^{-50\mu}}{\Gamma(97)} \quad a = 96, b = 50$$



$$\mathbb{E}[\mu] = \frac{a}{b} = \frac{96}{50} = 4.8 \quad \sqrt{\text{Var}(\mu)} = \sqrt{\frac{a}{b^2}} = \sqrt{\frac{96}{50^2}} = 0.49$$

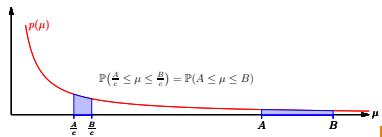
1. Bayes' Rule
2. Conjugate Priors
3. Uninformative Priors



- 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  $\mu$

$$\int_A^B p(\mu) d\mu = \int_{A/c}^{B/c} p(\mu) d\mu = \int_A^B \frac{1}{c} p\left(\frac{\nu}{c}\right) d\nu = \int_A^B \frac{1}{c} p\left(\frac{\mu}{c}\right) d\mu$$

making a change of variables  $\mu = \nu/c$

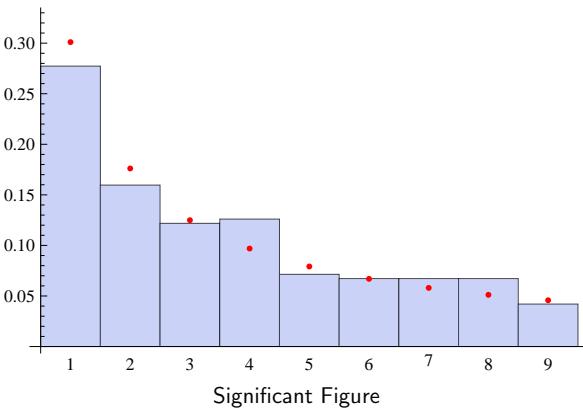
- Or  $p(\mu) = \frac{1}{c} p\left(\frac{\mu}{c}\right)$  implying  $p(\mu) \propto \frac{1}{\mu}$

### 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

$$\begin{aligned} \mathbb{P}(\text{most s.f. of } x = n) &= \frac{\int_n^{n+1} \frac{1}{x} dx}{\int_1^{10} \frac{1}{x} dx} = \frac{\int_{10^n}^{10^{n+1}} \frac{1}{x} dx}{\int_{10}^{100} \frac{1}{x} dx} \\ &= \frac{\log(n+1) - \log(n)}{\log(10)} = \log_{10}\left(\frac{n+1}{n}\right) \end{aligned}$$

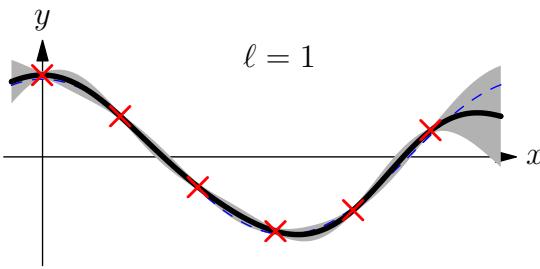
### Population Size of 238 Countries



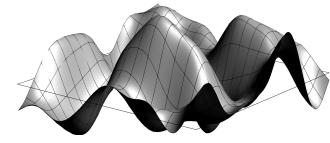
### Conclusion

- 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

### Gaussian Processes



Gaussian Processes, regression



### Gaussian Processes

- 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

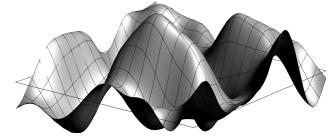
- In regression we try to fit a multi-dimensional function to our data
- (You can use Gaussian Processes for classification, e.g. by inferring the probabilities of being in a class, but we ignore this as regression is where GP excel)
- In regression we have some  $p$  dimensional feature vectors  $\mathbf{x}_i$  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  $f(\mathbf{x})$
- 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,  $p(y_i|f(\mathbf{x}_i))$ , we use the probability of observing  $y_i$  given the true function value is  $f(\mathbf{x}_i)$
- In general, this would be next to impossible to compute, except in the special case where everything is Gaussian (normally) distributed

### Outline

1. Introduction
2. Gaussian Processes
3. Bayesian Inference
4. Hyper-parameters



### 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
- $$p(f|m, k) \propto e^{-\frac{1}{2} \int (f(\mathbf{x}) - m(\mathbf{x})) k^{-1}(\mathbf{x}, \mathbf{y}) (f(\mathbf{y}) - m(\mathbf{y})) d\mathbf{x} d\mathbf{y}}$$
- The function  $m(\mathbf{x})$  is the mean  $\mathbb{E}[f(\mathbf{x})]$  (usually taken to be zero in most inference problems)

### Meaning of GP

- To understand GP's we can discretise space,  $\mathbf{x}$ , into a lattice of points  $\{\mathbf{x}_i\}$
  - Then (assuming  $m(\mathbf{x}) = 0$ )
- $$p(f|m, k) \propto \prod_i e^{-\frac{f_i^2 k^{-1}(\mathbf{x}_i, \mathbf{x}_i)}{2} + f_i \sum_j k^{-1}(\mathbf{x}_i, \mathbf{x}_j) f_j}$$
- where  $f_i = f(\mathbf{x}_i)$
- 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

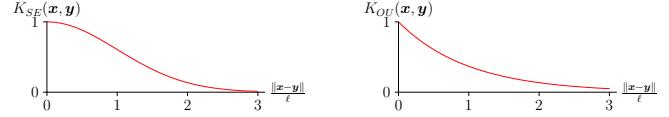
- $k(\mathbf{x}, \mathbf{y})$  is a covariance function

$$\mathbb{E}[(f(\mathbf{x}) - m(\mathbf{x})) (f(\mathbf{y}) - m(\mathbf{y}))] = k(\mathbf{x}, \mathbf{y})$$

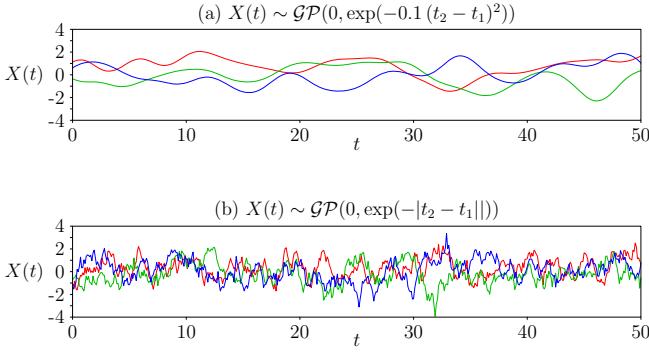
- This is sometimes known as a **kernel**—it must be positive semi-definite (just like in SVMs)
- It is a free “parameter” that the user gets to choose (although we can learn its parameters too)
- If  $k(\mathbf{x}, \mathbf{y})$  is a function of  $\mathbf{x} - \mathbf{y}$  it is “**stationary**”
- If  $k(\mathbf{x}, \mathbf{y})$  is a function of  $\|\mathbf{x} - \mathbf{y}\|$  it is also “**isometric**”

## Popular Choices of GP Kernel Function

- Constant:  $k_C(\mathbf{x}, \mathbf{y}) = C\mathbf{I}$
- Gaussian noise:  $k_{GN}(\mathbf{x}, \mathbf{y}) = \sigma^2 \delta_{\mathbf{x}, \mathbf{y}}$
- Squared exponential:  $k_{SE}(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|^2}{2\ell^2}\right)$
- Ornstein–Uhlenbeck:  $k_{OU}(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|}{\ell}\right)$

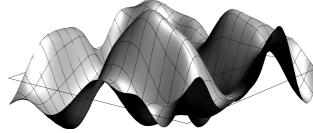


## Gaussian Process Worlds



## Outline

1. Introduction
2. Gaussian Processes
3. Bayesian Inference
4. Hyper-parameters



## Alternative Derivation

- Denoting the target values as a vector  $\mathbf{y}$  with elements  $y_i$
- Denoting the matrices of covariances between data points as  $\mathbf{K}$  with elements  $k(\mathbf{x}_i, \mathbf{x}_j)$
- Denoting the covariance between the data points and a particular position,  $\mathbf{x}_*$  as  $\mathbf{k}_*$  with elements  $k(\mathbf{x}_i, \mathbf{x}_*)$
- Denoting the variance at point  $\mathbf{x}_*$  as  $k_* = k(\mathbf{x}_*, \mathbf{x}_*)$
- Then the distribution of function values at points at  $\mathbf{x}_i$  and  $\mathbf{x}_*$  is

$$p(\mathbf{y}, f_*) = \mathcal{N}\left(\begin{pmatrix} \mathbf{y} \\ f_* \end{pmatrix} \middle| \mathbf{0}, \begin{pmatrix} \mathbf{K} + \sigma^2 \mathbf{I} & \mathbf{k}_* \\ \mathbf{k}_*^\top & k_* \end{pmatrix}\right)$$

## Conditional Probability

- To compute the posterior  $p(f_* | \mathbf{y})$  we use

$$p(f_* | \mathbf{y}) = \frac{p(f_*, \mathbf{y})}{p(\mathbf{y})}$$

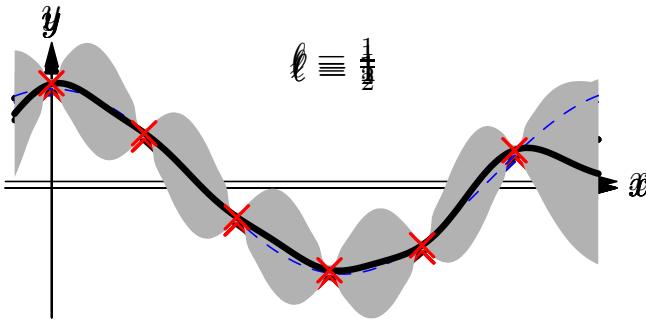
- where  $p(\mathbf{y}) = \int p(f_*, \mathbf{y}) df_*$

- Because all integrals are Gaussian we can compute the integral to obtain

$$p(f_* | \mathbf{y}) = \mathcal{N}\left(f_* \middle| \mathbf{k}_*^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{y}, k_* - \mathbf{k}_*^\top (\mathbf{K} + \sigma^2 \mathbf{I})^{-1} \mathbf{k}_*\right)$$

- Looks complicated, but numerically easy to evaluate

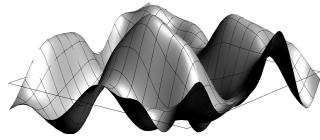
$$K(x, x') = \exp(-(x - x')^2/(2\ell^2))$$



- 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  $x$  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 **kriging**
- It was re-invented by the machine learning community who call it Gaussian Processes (GP)

## Outline

- Introduction
- Gaussian Processes
- Bayesian Inference
- Hyper-parameters



## Evidence Framework

- The normalisation factor,  $p(\mathcal{D}|\phi)$  is known as the **marginal likelihood** or **evidence**

$$p(\mathcal{D}|\phi) = \int p(\mathcal{D}|f, \phi)p(f|\phi)df$$

- We can perform a Bayesian calculation at a second level by putting a prior on  $\phi$

$$p(\phi|\mathcal{D}) = \frac{p(\mathcal{D}|\phi)p(\phi)}{p(\mathcal{D})}$$

- From this we can now marginalise out the hyper-parameters

$$p(f|\mathcal{D}) = \int p(f|\mathcal{D}, \phi)p(\phi|\mathcal{D})d\phi$$

## Evidence for GP

- For GP the (log)-evidence can be computed in closed form

$$\log(p(\mathcal{D}|\phi)) = -\frac{1}{2}\mathbf{y}^\top(\mathbf{K} + \sigma^2\mathbf{I})\mathbf{y} - \frac{1}{2}\log(|\mathbf{K} + \sigma^2\mathbf{I}|) - \frac{m}{2}\log(2\pi)$$

- 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!

## 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  $\ell$ ) that we have to choose correctly
- This is typical of many Bayesian problems where we have some set of hyper-parameters,  $\phi$ , describing the model
- These are different to the normal parameters we learn (e.g. weights  $w$  or in GP the functions  $f(x)$ )
- In Bayesian inference we learn the posterior for these normal parameters

$$p(f|\mathcal{D}, \phi) = \frac{p(\mathcal{D}|f, \phi)p(f|\phi)}{p(\mathcal{D}|\phi)}$$

## Maximum-Likelihood-II

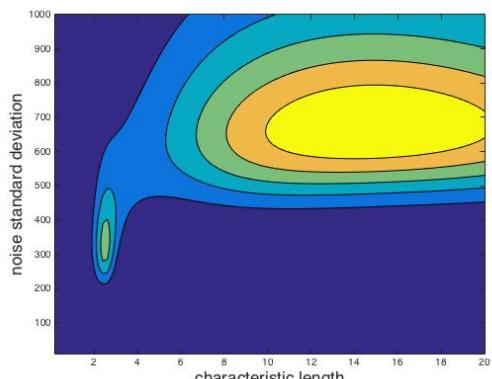
- The integral

$$p(f|\mathcal{D}) = \int p(f|\mathcal{D}, \phi)p(\phi|\mathcal{D})d\phi$$

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  $p(\mathcal{D}|\phi)$
- This is sometimes referred to as ML-II
- Normally even this can be difficult, but for GP its not too difficult

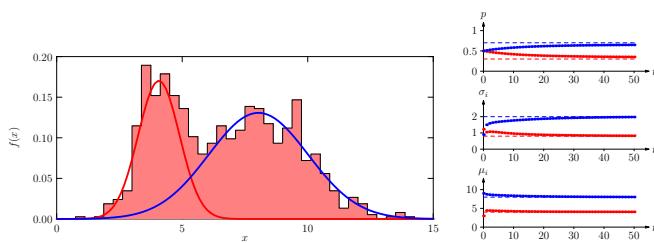
## Example (slightly pathological)



## Conclusions

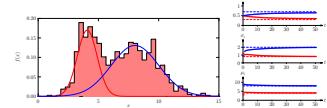
- Gaussian processes are very powerful for regression (and classification?)
- Because all calculations involve Gaussian integrals we can compute everything in closed form
- (Actually its a pain to do the mathematics because you end up working with inverse of matrices)
- Fairly generic (black-box) technique because the prior captures many continuity constraints
- We can use the evidence framework (probability of data) to do model selection and hyper-parameter optimisations

## Probabilistic Inference



Hierarchical Models, Mixture of Gaussians, Expectation Maximisation

1. Building Probabilistic Models
2. Mixture of Gaussians
3. Expectation Maximisation



## 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 variable 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:  $\mathbb{P}(X|Y) = \frac{\mathbb{P}(Y|X)\mathbb{P}(X)}{\mathbb{P}(Y)}$

## Discriminative Models

- We often think of our observations as given and the predictions as random variables
- For example we might be given some features  $x$  and we wish to predict a class  $C \in \mathcal{C}$
- Our objective is then to find the probability  $\mathbb{P}(C|x)$
- 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  $\mathbb{P}(\mathbf{X},Y)$  where  $\mathbf{X}$  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

$$\mathbb{P}(Y|\mathbf{X}) = \frac{\mathbb{P}(\mathbf{X},Y)}{\mathbb{P}(\mathbf{X})} = \frac{\mathbb{P}(\mathbf{X},Y)}{\sum_Y \mathbb{P}(\mathbf{X},Y)}$$

## 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  $\mathbf{X}$  and we are predicting a variable  $Y$  then we would **marginalise** over the latent variable

$$\mathbb{P}(\mathbf{X},Y) = \sum_Z \mathbb{P}(\mathbf{X},Y,Z)$$

## 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)

- 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

### 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

$$\mathbb{P}(\mathcal{D}) = \sum_{\Theta} \mathbb{P}(\mathcal{D}|\Theta) \mathbb{P}(\Theta)$$

- But sometimes the number of values that  $\Theta$  can take are so large that we cannot easily compute this
- Nevertheless we can usually do this using Monte Carlo techniques

### 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

### 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:  $\Theta = \{\mu_A, \sigma_A^2, \mu_B, \sigma_B^2\}$
- Let  $Z_i \in \{0,1\}$  be an indicator that particle  $i$  is of type  $A$
- The probability of  $X_i$  is given by

$$f(X_i|Z_i, \Theta) = Z_i \mathcal{N}(X_i|\mu_A, \sigma_A^2) + (1 - Z_i) \mathcal{N}(X_i|\mu_B, \sigma_B^2)$$

- We can use Bayes' rules to learn a set of parameter  $\Theta$  that occur in our likelihood function

$$\mathbb{P}(\Theta|\mathcal{D}) = \frac{\mathbb{P}(\mathcal{D}|\Theta) \mathbb{P}(\Theta)}{\mathbb{P}(\mathcal{D})}$$

- 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

### Maximum A Posteriori (MAP) Solution

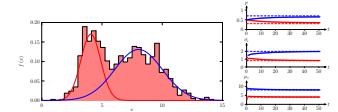
- One work around is to compute the mode of the posterior

$$\Theta_{\text{MAP}} = \underset{\Theta}{\operatorname{argmax}} f(\mathcal{D}|\Theta) f(\Theta) = \underset{\Theta}{\operatorname{argmax}} \log(f(\mathcal{D}|\Theta)) + \log(f(\Theta))$$

- 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) – it's not properly probabilistic
- You can overfit and you don't get an estimate of the error in your inference

### Outline

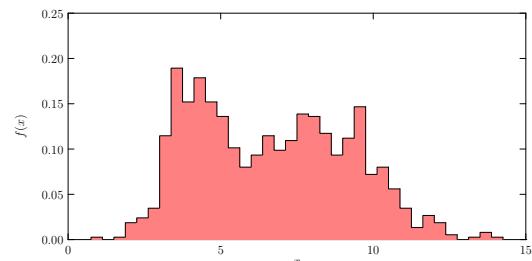
1. Building Probabilistic Models
2. Mixture of Gaussians
3. Expectation Maximisation



### Data

- Note that

$$\begin{aligned} f(X_i|\Theta) &= \sum_{Z_i \in \{0,1\}} f(X_i, Z_i|\Theta) = \sum_{Z_i \in \{0,1\}} f(X_i|Z_i, \Theta) \mathbb{P}(Z_i) \\ &= \mathbb{E}_{Z_i}[f(X_i|Z_i, \Theta)] = p \mathcal{N}(X_i|\mu_A, \sigma_A^2) + (1 - p) \mathcal{N}(X_i|\mu_B, \sigma_B^2) \end{aligned}$$



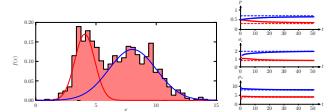
## Maximum Likelihood

- To solve the model as a Bayesian we would have to assign priors to our parameters  $\Theta = (\mu_A, \sigma_A, \mu_B, \sigma_B, p)$
- 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  $\mu_A$  and  $\mu_B$ )
- A reasonable approach is to seek those parameters that maximise the likelihood of our observed data

$$f(\mathcal{D}|\Theta) = \prod_{X \in \mathcal{D}} f(X|\Theta)$$

## Outline

- Building Probabilistic Models
- Mixture of Gaussians
- 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  $\mu_A, \sigma_A^2, \mu_B, \sigma_B^2$  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

$$\Theta^{(t+1)} = \operatorname{argmax}_{\Theta} \sum_{Z} \mathbb{P}(Z|\mathcal{D}, \Theta^{(t)}) \log(f(\mathcal{D}|Z, \Theta))$$

- 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

$$Q(\Theta|\Theta^{(t)}) = \sum_{Z \in \{0,1\}^m} \mathbb{P}(Z|\mathcal{D}, \Theta^{(t)}) \log(f(\mathcal{D}|Z, \Theta))$$

- We can show that the maximum,  $\Theta^{(t+1)}$ , is such that
$$\log(f(\mathcal{D}|\Theta^{(t+1)})) - \log(f(\mathcal{D}|\Theta^{(t)})) \geq Q(\Theta^{(t+1)}|\Theta^{(t)}) - Q(\Theta^{(t)}|\Theta^{(t)}) \geq 0$$
- 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

$$\mathbb{P}(Z|\mathcal{D}, \Theta^{(t)}) = \prod_{i=1}^m \mathbb{P}(Z_i|X_i, \Theta^{(t)})$$

where

$$\begin{aligned} \mathbb{P}(Z_i = 1|X_i, \Theta^{(t)}) &= \frac{p^{(t)} \mathcal{N}(X_i|\mu_A^{(t)}, \sigma_A^{2(t)})}{p^{(t)} \mathcal{N}(X_i|\mu_A^{(t)}, \sigma_A^{2(t)}) + (1-p^{(t)}) \mathcal{N}(X_i|\mu_B^{(t)}, \sigma_B^{2(t)})} \\ \mathbb{P}(Z_i = 0|X_i, \Theta^{(t)}) &= 1 - \mathbb{P}(Z_i = 1|X_i, \Theta^{(t)}) \end{aligned}$$

## EM for Mixture of Gaussians

- Maximise with respect to parameters  $\theta$

$$\begin{aligned} Q(\theta|\theta^{(t)}) &= \sum_Z \mathbb{P}(Z|\mathcal{D}, \Theta^{(t)}) \log(f(\mathcal{D}|Z, \Theta)) = \sum_{i=1}^m \sum_{Z_i} \mathbb{P}(Z_i|\mathcal{D}, \Theta^{(t)}) \log(f(X_i|Z_i, \Theta)) \\ &= \sum_{i=1}^m \sum_{Z_i \in \{0,1\}} \mathbb{P}(Z_i|X_i, \theta^{(t)}) (Z_i \log(p) + (1-Z_i) \log(1-p) \\ &\quad - \frac{(X_i - \mu_{Z_i})^2}{2\sigma_{Z_i}^2} - \log(\sqrt{2\pi}\sigma_{Z_i})) \end{aligned}$$

- Compute update equations

$$\begin{aligned} \frac{\partial Q(\theta|\theta^{(t)})}{\partial \mu_k} &= 0, & \frac{\partial Q(\theta|\theta^{(t)})}{\partial \sigma_k} &= 0, & \frac{\partial Q(\theta|\theta^{(t)})}{\partial p} &= 0 \end{aligned}$$

## Update Equations

- Means

$$\mu_{Z_i}^{(t+1)} = \frac{\sum_{i=1}^n \mathbb{P}(Z_i|X_i, \theta^{(t)}) X_i}{\sum_{i=1}^n \mathbb{P}(Z_i|X_i, \theta^{(t)})},$$

- Variances

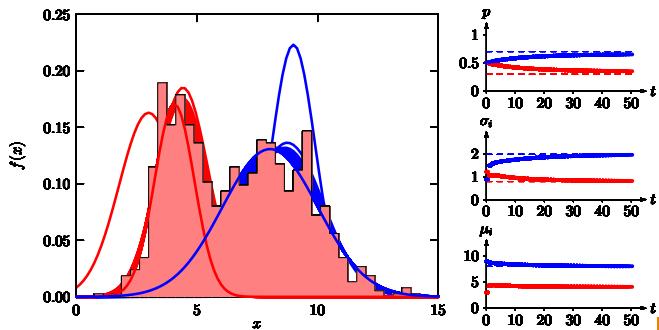
$$(\sigma_{Z_i}^{(t+1)})^2 = \frac{\sum_{i=1}^n \mathbb{P}(Z_i|X_i, \theta^{(t)}) (X_i - \mu_{Z_i}^{(t+1)})^2}{\sum_{i=1}^n \mathbb{P}(Z_i|X_i, \theta^{(t)})}$$

- Probability of being type 1

$$p^{(t+1)} = \frac{1}{n} \sum_{i=1}^n \mathbb{P}(Z_i = 1 | X_i, \theta_i^{(t)})$$

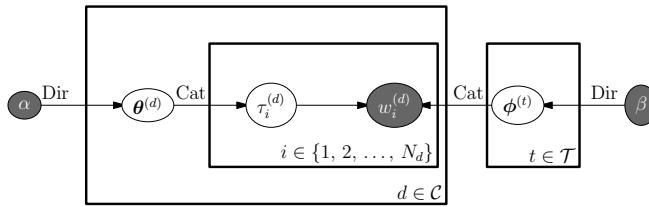
## Example

## Summary

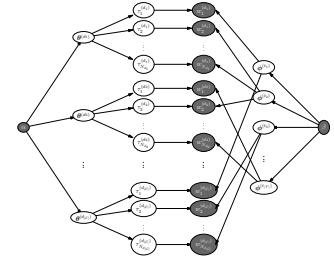


- Building probabilistic models is an intricate process
- Identifying random variables that describe the system is the first step
- Often we need to introduce variables that we don't observe and need to be marginalised out
- The EM algorithm provide one approach to maximising likelihoods or MAP solutions when we have latent variables
- It often gives nice update equations, but convergence can be slow

### Graphical Models



1. Graphical Models
2. Cakes!
3. Latent Dirichlet Allocation



Conditional Independence, Graphical models, LDA

### Graphical Models

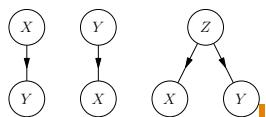
- If we want to build large probabilistic inference systems
  - ★ AI Doctor
  - ★ Fault diagnostic system for a computer
- we can describe this by introducing random variables, but it is helpful to graphically represent causal connections
- Graphical models allow us to do this
- It allows us to build 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

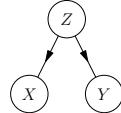
### Statistical Independence

- 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

$$\mathbb{P}(X,Y|Z) = \mathbb{P}(X|Z)\mathbb{P}(Y|Z)$$

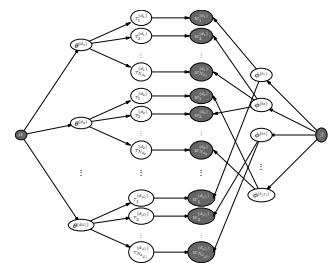


- Conditional independence implies that there is no direct causation
- But it doesn't imply zero correlation
- Conditional independence reduces computational complexity, e.g.

$$\mathbb{E}[XY] = \sum_{X,Y,Z} XY\mathbb{P}(X,Y,Z) = \sum_Z P(Z) \left( \sum_X X P(X|Z) \right) \left( \sum_Y Y P(Y|Z) \right)$$

### 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

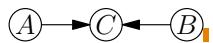
## The Cake Scenario

- 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

$$\begin{aligned} \mathbb{P}(A, B, C) &= \mathbb{P}(C, B|A)\mathbb{P}(A) \\ &= \mathbb{P}(C|A, B)\mathbb{P}(B|A)\mathbb{P}(A) = \mathbb{P}(C|A, B)\mathbb{P}(B)\mathbb{P}(A) \end{aligned}$$

- Because  $\mathbb{P}(B|A) = \mathbb{P}(B)$

## Are There Any Cakes Left?

- We can use our model to compute the probabilities of there being cakes in the coffee room

$$\begin{aligned} \mathbb{P}(C = 1) &= \sum_{A, B, C \in \{0, 1\}} \mathbb{P}(C = 1)\mathbb{P}(A, B, C) \\ &= \sum_{A, B \in \{0, 1\}} \mathbb{P}(C = 1|A, B)\mathbb{P}(A)\mathbb{P}(B) = 0.29 \end{aligned}$$

- 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

## 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

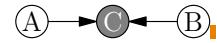
$$\mathbb{P}(A = 1|C = 1) = 0.628, \quad \mathbb{P}(B = 1|C = 1) = 0.317$$

$$\mathbb{P}(A = 1, B = 1|C = 1) = 0.069$$

- 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

## 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

$$\mathbb{P}(C) = \sum_{A, B \in \{0, 1\}} \mathbb{P}(A, B, C)$$

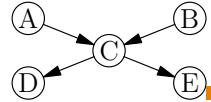
## Elaborate Cakes

- 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

## Dependencies

- 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  $C$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

17

- If we don't observe cakes then the probability of Dave and Eli eating cake are not independent

$$\begin{aligned}\mathbb{P}(D = 1) &= 0.303, & \mathbb{P}(E = 1) &= 0.174 \\ \mathbb{P}(D = 1, E = 1) &= 0.1392\end{aligned}$$

$$\text{so } \mathbb{P}(D, E) \neq \mathbb{P}(D) \mathbb{P}(E)$$

- This changes if we know there are cakes in the coffee room

$$\begin{aligned}\mathbb{P}(D = 1 | C = 1) &= 0.8 & \mathbb{P}(E = 1 | C = 1) &= 0.6 \\ \mathbb{P}(D = 1, E = 1 | C = 1) &= 0.48\end{aligned}$$

$$\text{so } \mathbb{P}(D = 1, E = 1 | C = 1) = \mathbb{P}(D = 1 | C = 1) \mathbb{P}(E = 1 | C = 1)$$

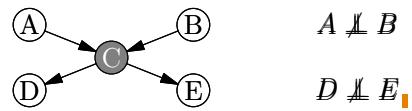
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

18

## Observations and Independence

- Making observations changes the probabilities and in some case the dependencies of random variables on each other



- 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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

19

## 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

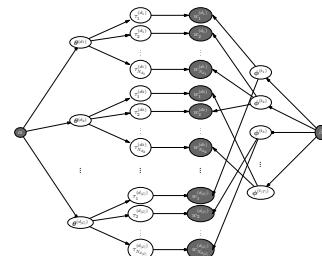
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

20

## Outline

- Graphical Models
- Cakes!
- Latent Dirichlet Allocation



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

21

## Model for Documents

- We consider a model for the words in a set of documents (we ignore word order)
- We consider a corpus  $\mathcal{C} = \{d_i | i = 1, 2, \dots, |\mathcal{C}|\}$
- With documents consisting of words

$$d = (w_1^{(d)}, w_2^{(d)}, \dots, w_{N_d}^{(d)})$$

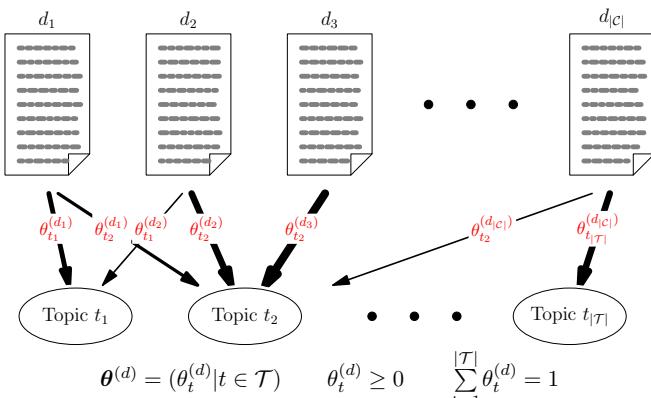
- We assume that there is a set of topics  $\mathcal{T} = \{t_1, t_2, \dots, t_{|\mathcal{T}|}\}$
- We associate a probability,  $\theta_t^{(d)}$ , that a word in document  $d$  relates to a topic  $t$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

22

## Documents and Topic



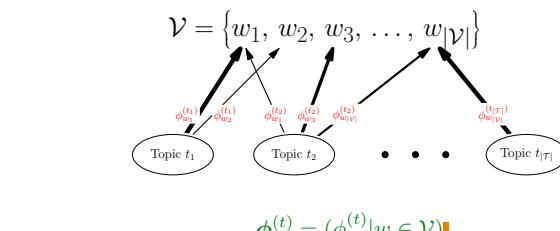
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

23

## Words and Topic

- We associate a probability  $\phi_w^{(t)}$  that a word,  $w$ , is related to a topic  $t$



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

24

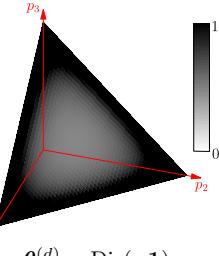
## Dirichlet Allocation

- 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  $\theta^{(d)}$  and  $\phi^{(t)}$  from a Dirichlet distribution with small parameters  $\alpha$

$$\text{Dir}(\boldsymbol{p}|\alpha) = \Gamma\left(\sum_i \alpha_i\right) \prod_{i=1}^n \frac{\alpha_i - 1}{\Gamma(\alpha_i)} \theta^{(d)} \sim \text{Dir}(\alpha \mathbf{1}) \quad \phi^{(t)} \sim \text{Dir}(\beta \mathbf{1})$$

$$\sum p_i = 1$$



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

25

## Generating Document

- To generate a document we choose a topic for each word and a word for each topic

$$\forall d \in \mathcal{C} \quad \theta^{(d)} \sim \text{Dir}(\alpha \mathbf{1})$$

$$\forall t \in \mathcal{T} \quad \phi^{(t)} \sim \text{Dir}(\beta \mathbf{1})$$

$$\forall d \in \mathcal{C} \wedge \forall i \in \{1, 2, \dots, N_d\} \quad \tau_i^{(d)} \sim \text{Cat}(\theta^{(d)}) \quad w_i^{(d)} \sim \text{Cat}(\phi^{(\tau_i^{(d)})})$$

- Where  $\text{Cat}(i|p) = p_i$  is the categorical distribution (we choose one of a number of options)

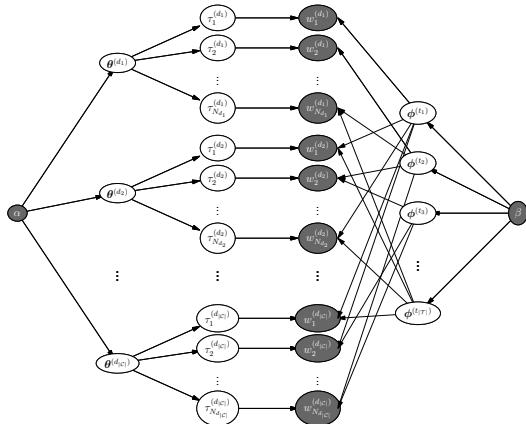
- This model is known as **Latent Dirichlet Allocation**

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

26

## LDA Graphical Model (version 1)



Adam Prügel-Bennett

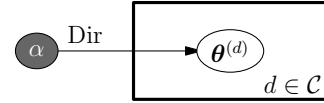
COMP6208 Advanced Machine Learning

27

## Plate Diagrams

- Drawing every random variable is tedious (and not really possible)

- A short-hand is to draw a box (plate) meaning repeat



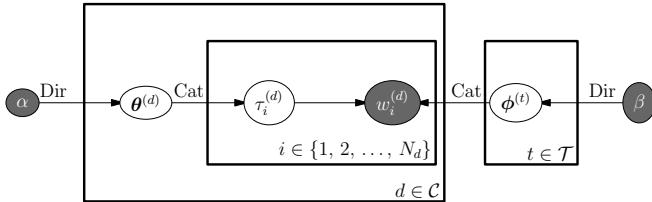
- That is we generate vectors  $\theta^d$  from a Dirichlet distribution  $\text{Dir}(\theta|\alpha \mathbf{1})$  for all documents in corpus  $\mathcal{C}$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

28

## LDA Graphical Model (version 2)



- This is a lot more compact

- Personally, I find it hard to read, but you get used to it

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

29

## Probabilistic Model

- The graphical Model is shorthand for the variables

$$W = (w^{(d)} | d \in \mathcal{C}) \quad \text{with} \quad w^{(d)} = (w_1^{(d)}, w_2^{(d)}, \dots, w_{N_d}^{(d)}), \quad \text{and} \quad w_i^{(d)} \in \mathcal{V}$$

$$T = (\tau_i^{(d)} | d \in \mathcal{C} \wedge i \in \{1, 2, \dots, N_d\}) \quad \text{with} \quad \tau_i^{(d)} \in \mathcal{T}$$

$$\Theta = (\theta^{(d)} | d \in \mathcal{C}) \quad \text{with} \quad \theta^{(d)} = (\theta_t^{(d)} | t \in \mathcal{T}) \in \Lambda^{|\mathcal{T}|}$$

$$\Phi = (\phi^{(t)} | t \in \mathcal{T}) \quad \text{with} \quad \phi^{(t)} = (\phi_w^{(t)} | w \in \mathcal{V}) \in \Lambda^{|\mathcal{V}|}$$

- Distributed according to

$$\begin{aligned} \mathbb{P}(W, T, \Theta, \Phi | \alpha, \beta) &= \left( \prod_{t \in \mathcal{T}} \text{Dir}(\phi^{(t)} | \beta \mathbf{1}) \right) \\ &\quad \left( \prod_{d \in \mathcal{C}} \text{Dir}(\theta^{(d)} | \alpha \mathbf{1}) \prod_{i=1}^{N_d} \text{Cat}(\tau_i^{(d)} | \theta^{(d)}) \text{Cat}(w_i^{(d)} | \phi^{(\tau_i^{(d)})}) \right) \end{aligned}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

30

## Finding Topics

- We are given the set of words  $W$  and don't really care about  $\tau_i^d$  the topic associated with word  $i$  in document  $d$
- But we are interested in the words associated with each topic  $\phi^{(t)}$
- And the topics associated with each document  $\theta^{(d)}$
- To compute them we need to sample the probability distribution
- One way to do this is using Monte Carlo methods (see next lecture)

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

31

## Summary

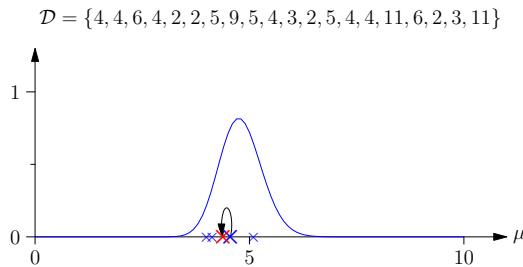
- Building probabilistic models is an intricate process
- Graphical models provide a representation showing the causal relationship between random variables
- This allows us to break down the joint probability of all the variables into conditional probabilities
- This is useful for building the model, 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)

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

32

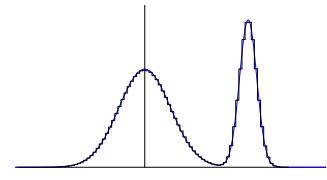
## MCMC



Monte Carlo methods, MCMC, Variational Methods

$T = 10000000$ , acceptance rate = 0.897

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, Dirichlet) 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 distribution

## 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.

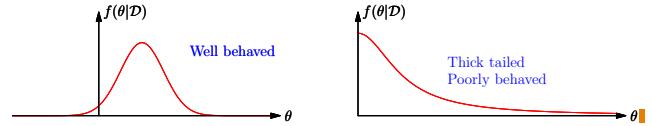
$$\begin{aligned}\mathbb{E}[\boldsymbol{\Theta}] &= \mathbb{E}[\boldsymbol{\Theta}_i^2] - \mathbb{E}[\boldsymbol{\Theta}_i]^2 \\ \mathbb{E}[\boldsymbol{\Theta}_i \boldsymbol{\Theta}_j] &= \mathbb{E}[\boldsymbol{\Theta}_i] \mathbb{E}[\boldsymbol{\Theta}_j] \\ \mathbb{E}[\boldsymbol{\Theta} \boldsymbol{\Theta}^T] &= \mathbb{E}[\boldsymbol{\Theta}] \mathbb{E}[\boldsymbol{\Theta}]^T\end{aligned}$$

## 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

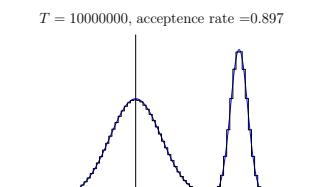
$$\mathbb{E}[g(\boldsymbol{\Theta})] \approx \frac{1}{n} \sum_{i=1}^n g(\boldsymbol{\Theta}_i)$$

- Provided our posterior distribution is well behaved the relative error in our estimate will drop off as  $1/\sqrt{n}$



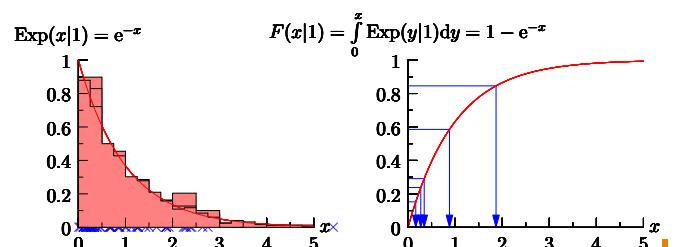
## Outline

1. Sampling
2. Random Number Generation
3. MCMC

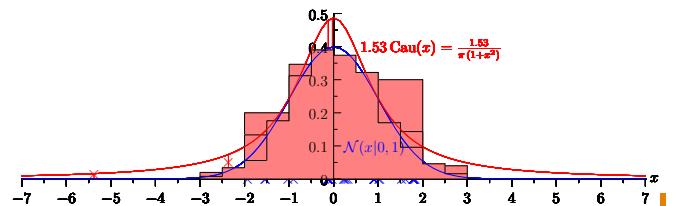


## Drawing Random Samples

- 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



- 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  $g_Y(y)$

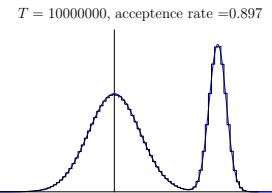


### Problems with Rejection

- 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 particularly true for multivariate distributions
- If the proposal distribution is poor  $c$  might be very high and the number of rejections is stupidly high

### Outline

- Sampling
- Random Number Generation
- MCMC



### Detailed Balance

- Suppose we have a set of states  $\mathcal{S}$  and want to draw samples from a probability distribution  $\pi = (\pi_i | i \in \mathcal{S})$
- 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  $j$

$$\sum_j M_{ij}\pi_j = \sum_j M_{ji}\pi_i = \pi_i \quad \mathbf{M}\pi = \pi$$

### Convergence of MCMC

- Suppose we start from a state  $x(0) = \sum_i c_i v^{(i)}$  where the  $v^{(i)}$ 's are eigenvectors of the transition matrix  $\mathbf{M}$  with eigenvalues  $\lambda_i$
  - If I apply  $\mathbf{M}$  many times then
- $$x(t) = \mathbf{M}^t x(0) = \mathbf{M}^t \sum_i c_i v^{(i)} = \sum_i \lambda_i^t c_i v^{(i)}$$
- $\lim_{t \rightarrow \infty} x(t) = v^*$  where  $v^*$  is the eigenvector with the maximum eigenvalue
  - Now  $\|\mathbf{M}v\|_1 \leq \|\mathbf{M}\|_1 \|v\|_1 = \|v\|_1$  so the maximum eigenvalue is 1 with eigenvector  $\pi$  ( $\mathbf{M}$  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  $\pi_i/\pi_j$
  - If  $\pi_i > \pi_j$  then  $M_{ij} = 1$  and  $M_{ji} = \pi_j/\pi_i$ . Thus
- $$M_{ij}\pi_j = \pi_j \quad M_{ji}\pi_i = \frac{\pi_j}{\pi_i}\pi_i = \pi_j$$
- Note that we require the state  $i$  to have the same number of neighbours as state  $j$  so that detailed balance is satisfied

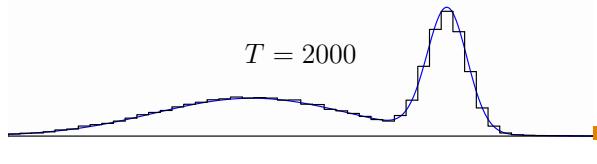
### Continuous Variables

- If we are working with continuous variables  $\theta$  then the equation for detailed balance for the transition probability  $W(\theta \rightarrow \theta')$  is
- $$W(\theta \rightarrow \theta')\pi(\theta) = W(\theta' \rightarrow \theta)\pi(\theta')$$
- where  $\pi(\theta)$  is the probability distribution we wish to sample from
  - The update rule is to choose a nearby value  $\theta'$ , compute  $r = \pi(\theta')/\pi(\theta)$  and accept the update with probability  $\min(1, r)$
  - We require that the probability of choosing  $\theta$  from  $\theta'$  is the same as the reverse

- Because we are free to choose where we move (and choose close by neighbours)  $\pi(\theta') \approx \pi(\theta)$  so that moves are not too infrequent
- Also very importantly the updates depend only on the ratio  $\pi(\theta')/\pi(\theta)$
- 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  $\mathbb{P}(\mathcal{D}|\theta)\mathbb{P}(\theta)$  (or  $f(\mathcal{D}|\theta)f(\theta)$ )
- We don't need to know  $\mathbb{P}(\mathcal{D})$  which we generally don't know

- It can take a long time until our states occur with the probability  $\pi$  (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  $\theta'$  from our current position  $\theta$ —a good choice can increase the acceptance rate making the MCMC more efficient
- We define the proposal distribution  $p(\theta'|\theta)$
- For the standard Metropolis algorithm to work we require  $p(\theta'|\theta) = p(\theta|\theta')$
- 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

$$\min\left(1, \frac{p(\theta|\theta')f(\mathcal{D}|\theta')f(\theta')}{p(\theta'|\theta)f(\mathcal{D}|\theta)f(\theta)}\right)$$

### Traffic Rate

- Consider monitoring the flow of traffic where we have data

$$\mathcal{D} = (N_1, N_2, \dots, N_n)$$

where  $N_i$  is the number of cars that pass on day  $i$

- We assume  $N_i \sim \text{Poi}(\mu)$  and want to infer  $\mu$
- The Poisson distribution has a beta conjugate prior
- We don't have any prior knowledge on  $\mu$  so we use a non-informative prior  $\text{Gam}(\mu|0,0) = 1/\mu$
- Note that we can solve this problem exactly—however, let's compare with MCMC

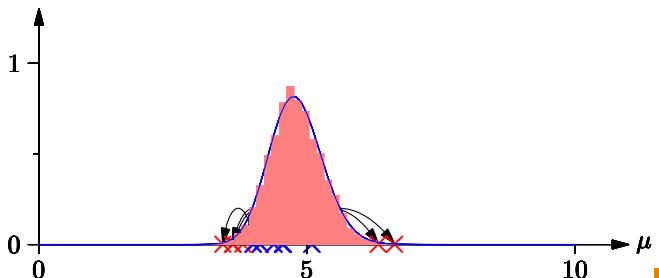
### Proposal Distribution

- 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

$$\begin{aligned} r &= \frac{\text{Gam}(\mu|\mu'^2, \mu') \frac{1}{\mu} \prod_{i=1}^n \text{Poi}(N_i|\mu')}{\text{Gam}(\mu'|\mu^2, \mu) \frac{1}{\mu} \prod_{i=1}^n \text{Poi}(N_i|\mu)} \\ &= \frac{\mu \text{Gam}(\mu|\mu'^2, \mu')}{\mu' \text{Gam}(\mu'|\mu^2, \mu)} e^{-n(\mu' - \mu) + \sum_{i=1}^n N_i \log\left(\frac{\mu'}{\mu}\right)} \end{aligned}$$

### MCMC in Practice

$$\mathcal{D} = \{4, 4, 6, 4, 2, 2, 5, 9, 5, 4, 3, 2, 5, 4, 4, 11, 6, 2, 3, 11\}$$



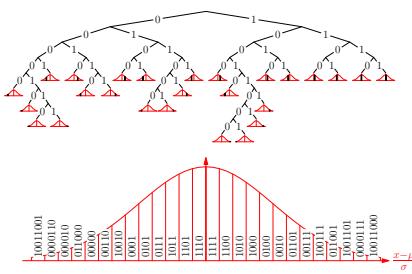
### MCMC Details

- To compute correct histograms you need to count samples where no move is made multiple times
- On modern computers it's quite quick to compute millions of samples
- The code is not very difficult to write (although care is needed 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

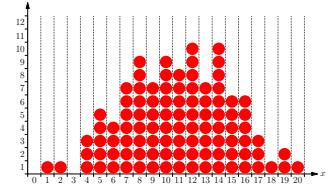
- MCMC provides a means to accurately sample from very complex models
- There have been many advanced techniques developed to improve MCMC performance
- E.g. hybrid MCMC simulates a dynamics to find good proposals with similar probability far from the starting point
- Often it seems that MCMC is complicated because there are so many optimisations, but often simple implementations are sufficient

- 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

### Entropy



Entropy, Coding, 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  $C \in \{000,001,\dots,111\}$  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

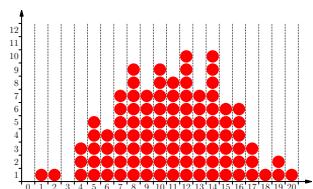
- What if we don't care about the order of the outcome then  $\mathbb{P}(HHH) = \mathbb{P}(TTT) = 1/8$ ,  $\mathbb{P}(HHT) = \mathbb{P}(HTT) = 3/8$  so
$$H_U = -\frac{1}{4} \log_2 \left( \frac{1}{8} \right) - \frac{3}{4} \log_2 \left( \frac{3}{8} \right) \approx 1.811 \text{ bits}$$
- This seems reasonable, 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  $(X,Y)$ ?
$$\begin{aligned} H_{(X,Y)} &= - \sum_{X,Y} \mathbb{P}(X,Y) \log_2(\mathbb{P}(X,Y)) \\ &= - \sum_{X,Y} \mathbb{P}(X)\mathbb{P}(Y) \log_2(\mathbb{P}(X)\mathbb{P}(Y)) \\ &= - \sum_{X,Y} \mathbb{P}(X)\mathbb{P}(Y) (\log_2(\mathbb{P}(X)) + \log_2(\mathbb{P}(Y))) \\ &= - \sum_X \mathbb{P}(X) \log_2(\mathbb{P}(X)) - \sum_Y \mathbb{P}(Y) \log_2(\mathbb{P}(Y)) = H_X + H_Y \end{aligned}$$
- 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

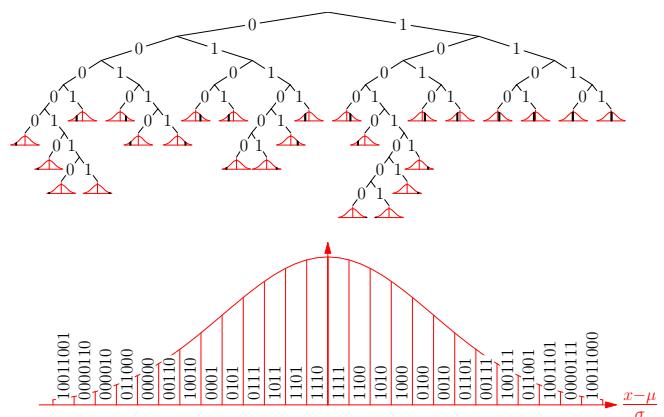
## Shannon's Entropy

- 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)

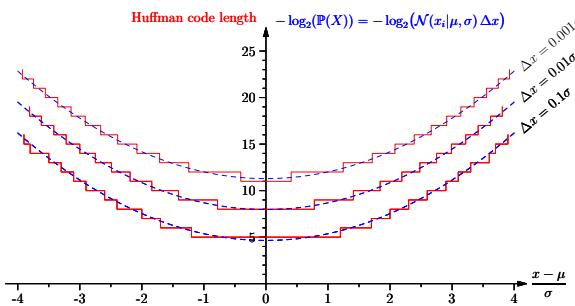
## Real Codes

- 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 $\Delta x$

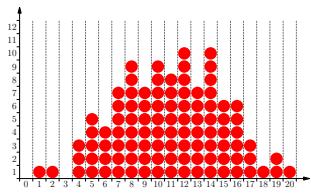


## bits and nats

- We have measured entropy in **bits** using
$$H_X = -\sum_{x \in \mathcal{X}} \mathbb{P}(X=x) \log_2(\mathbb{P}(X=x))$$
- Sometimes it is easier to use natural logarithms
$$H_X = -\sum_{x \in \mathcal{X}} \mathbb{P}(X=x) \ln(\mathbb{P}(X=x))$$
- In this case the entropy is measured in **nats** with 1 nat equal to  $\log_2(e)$  bits
- This is often easier when we want to do calculus on entropy

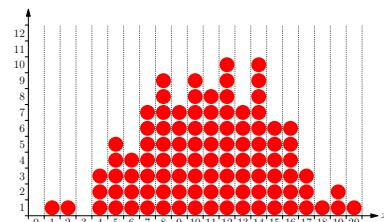
## Outline

- Measuring Uncertainty
- Code Length
- 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  $\mu$  and variance is  $\sigma^2$



$$\mathbb{P}(\mathbf{n}) \propto \frac{N!}{n_1! n_2! \dots n_K!} \left[ \sum_i \frac{n_i}{N} x_i = \mu \right] \left[ \sum_i \frac{n_i}{N} (x_i - \mu)^2 = \sigma^2 \right]$$

## Stirling's Approximation

- We can approximate the factorial  $n!$  using **Stirling's approximation**

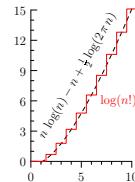
$$n! \approx \sqrt{2\pi n} n^n e^{-n}$$

$$\log(n!) = n \log(n) - n + \frac{1}{2} \log(2\pi n)$$

- Using this in our formula for  $\mathbb{P}(n)$  we have

$$\mathbb{P}(n) \approx C e^{-N \sum_i \frac{n_i}{N} \log(\frac{n_i}{N})} \prod_{l=1}^3 \left[ \sum_i \frac{n_i}{N} f_l(x_i) = v_l \right]$$

where  $(f_l(x_i), v_l) = \{(1,1), (x_i, \mu), ((x_i - \mu)^2, \sigma^2)\}$



## Number of States and Entropy

- Let  $p(x_i) = n_i/N$  be the proportion of balls in bin  $i$  then

$$\mathbb{P}(n) \approx C e^{N H_X} \prod_{l=1}^3 \left[ \sum_i \frac{n_i}{N} f_l(x_i) = v_l \right]$$

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,  $N \rightarrow \infty$  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

$$\mathbb{E}[X] = \mu, \quad \mathbb{E}[X^2] = \mu_2 = \mu^2 + \sigma^2$$

- To maximise the entropy subject to constraints consider

$$\begin{aligned} \mathcal{L}(f) = & - \int f_X(x) \log(f_X(x)) dx + \lambda_0 \left( \int f_X(x) dx - 1 \right) \\ & + \lambda_1 \left( \int f_X(x) x dx - \mu \right) + \lambda_2 \left( \int f_X(x) x^2 dx - \mu_2 \right) \end{aligned}$$

- 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

$$f_X(x) = e^{-1+\lambda_0+\lambda_1 x+\lambda_2 x^2}$$

## Normal Distribution

- We have three constraints

$$\begin{aligned} \int e^{-1+\lambda_0+\lambda_1 x+\lambda_2 x^2} dx &= 1 \\ \int e^{-1+\lambda_0+\lambda_1 x+\lambda_2 x^2} x dx &= \mu \\ \int e^{-1+\lambda_0+\lambda_1 x+\lambda_2 x^2} x^2 dx &= \mu_2 = \mu^2 + \sigma^2 \end{aligned}$$

- Solving for  $\lambda_0$ ,  $\lambda_1$  and  $\lambda_2$  then

$$f_X(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}$$

- 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

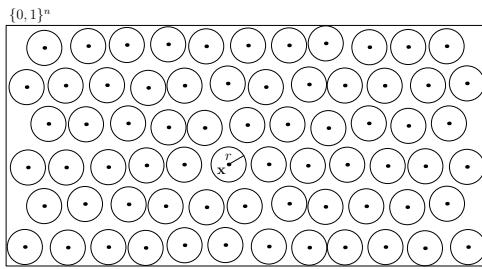
## Historic Entropy

- 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 communicate  $x$
- 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

## Information Theory



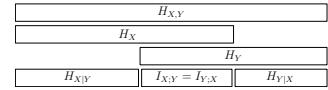
Information, KL-divergence, Minimum Description Length

### 1. Information Theory

### 2. KL-Divergence

### 3. Minimum Description Length

### 4. Variational Auto-Encoders



## Communicating Via a Noisy Channel

- Information theory considers communicating down a (noisy) channel

$$X \sim \mathbb{P}(X) \xrightarrow{\text{noisy channel}} Y \sim \mathbb{P}(Y | X)$$

- We send a message  $X$  (with probability  $\mathbb{P}(X)$ ) and receive a message  $Y$  with probability  $\mathbb{P}(Y | X)$

- The uncertainty of the message sent, given we received a message  $y$  is

$$H_{X|Y=y} = - \sum_{x \in \mathcal{X}} \mathbb{P}(X=x | Y=y) \log(\mathbb{P}(X=x | Y=y))$$

- The expected uncertainty in the message sent is

$$H_{X|Y} = \sum_{y \in \mathcal{Y}} \mathbb{P}(Y=y) H_{X|Y=y} = - \sum_{x,y} \mathbb{P}(X=x, Y=y) \log(\mathbb{P}(X=x | Y=y))$$

## 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

$$I_{X;Y} = I_{Y;X}$$

## Independent Noise

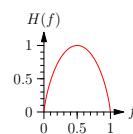
- The simplest model of a noisy channel is a binary channel where each symbol is corrupted independently with a probability  $f$

$$\mathbb{P}(X=1|Y=0) = \mathbb{P}(X=0|Y=1) = f$$

- An elementary calculations shows that

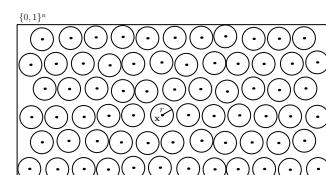
$$H_{X_i|Y_i} = -(1-f)\log(1-f) - f\log(f) = H(f)$$

- 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  $B(x,r)$  is the set of strings that differ from  $n$ -dimensional binary string,  $x$ , by at most  $r$  digits

## Volume of Coding Space

## Lower Bounds

- 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

$$\frac{2^n}{|B(\mathbf{x}_i, r)|} = c\sqrt{n}2^{I_{X,Y}}$$

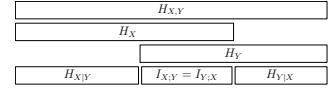
- Shannon also showed that choosing  $2^{I_{X,Y}}$  random strings of length  $n$  the Hamming distance between balls would be at least  $r$  with high probability
- This means that we can send information at rate of  $I_{X,Y}$
- The maximum rate is given by the channel capacity  $\max I_{X,Y}$
- If  $f = 0.1$  then  $C = I_{X,Y} = 0.469$  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

- Information Theory
- KL-Divergence**
- Minimum Description Length
- Variational Auto-Encoders



## KL-Divergence

- We have met the Kullback-Leibler divergence

$$\begin{aligned} \text{KL}(p \| q) &= \mathbb{E}_{X \sim p(X)} \left[ \log \left( \frac{p(X)}{q(X)} \right) \right] \\ &= -\mathbb{E}_{X \sim p(X)} [\log(q(X))] - H_X \end{aligned}$$

- 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  $q(X)$  than  $p(X)$
- $\text{KL}(p \| q)$  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.

$$f(\boldsymbol{\theta} | \mathcal{D}) \approx g(\boldsymbol{\theta} | \boldsymbol{\phi}) = \prod_i g(\theta_i | \phi_i)$$

- The standard method for solving this is to maximise the **variational free energy**

$$\Phi(\boldsymbol{\phi}) = - \int g(\boldsymbol{\theta} | \boldsymbol{\phi}) \log \left( \frac{g(\boldsymbol{\theta} | \boldsymbol{\phi})}{f(\boldsymbol{\theta}, \mathcal{D})} \right) d\boldsymbol{\theta}$$

## Evidence Lower Bound (ELBO)

- We can re-write the variational free energy as

$$\begin{aligned} \Phi(\boldsymbol{\phi}) &= - \int g(\boldsymbol{\theta} | \boldsymbol{\phi}) \log \left( \frac{g(\boldsymbol{\theta} | \boldsymbol{\phi})}{(f(\boldsymbol{\theta}, \mathcal{D}) / f(\mathcal{D})) f(\mathcal{D})} \right) d\boldsymbol{\theta} \\ &= - \int g(\boldsymbol{\theta} | \boldsymbol{\phi}) \left( \log \left( \frac{g(\boldsymbol{\theta} | \boldsymbol{\phi})}{f(\boldsymbol{\theta} | \mathcal{D})} \right) - \log(f(\mathcal{D})) \right) d\boldsymbol{\theta} \\ &= -\text{KL}(g(\boldsymbol{\theta} | \boldsymbol{\phi}) \| f(\boldsymbol{\theta} | \mathcal{D})) + \log(f(\mathcal{D})) \end{aligned}$$

- If we maximise  $\Phi(\boldsymbol{\phi})$ , we end up minimising the KL divergence between  $g$  and  $f$  so that  $g \approx f$  and  $\Phi(\boldsymbol{\phi}) \approx \log(f(\mathcal{D}))$
- 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

$$L_q(\boldsymbol{\phi}) = \int g(\boldsymbol{\theta} | \boldsymbol{\phi}) (\log(f(\mathcal{D} | \boldsymbol{\theta})) + \log(f(\boldsymbol{\theta}))) d\boldsymbol{\theta}$$

acts like an expected posterior term that is maximised when the data is well modelled (we put the probability density,  $g(\boldsymbol{\theta} | \boldsymbol{\phi})$  where the  $f(\boldsymbol{\theta}, \mathcal{D})$  is large)

- The second term is an entropy

$$H_q(\boldsymbol{\phi}) = - \int g(\boldsymbol{\theta} | \boldsymbol{\phi}) \log(g(\boldsymbol{\theta} | \boldsymbol{\phi})) d\boldsymbol{\theta}$$

That is, we maximise the uncertainty of the distribution  $g(\boldsymbol{\theta} | \boldsymbol{\phi})$

## Using Variational Methods

- Variational methods can be much faster than MCMC (although they tend to involve some iterations to minimise the variation free energy)■
- They can produce very good approximations, although this is not guaranteed (depends on the problem)■
- They can be extended (e.g. by minimising  $\text{KL}(g\|f)$  rather than  $\text{KL}(f\|g)$ —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■

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

17

## 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■

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

19

## Description Length

- The **description length** for  $\{y_i \mid i = 1, 2, \dots, m\}$  is then the cost of transmitting  $\theta$  plus the cost of transmitting the errors

$$L = \sum_{k=1}^n \ell(\theta_k) - \sum_{i=1}^m \left( \log(p_\delta(y_i - \hat{f}(\mathbf{x}_i|\theta))) + \log(\Delta) \right)$$

where  $\ell(\theta_k)$  is the number of bits need to communicate  $\theta_k$  (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  $\Delta$  will lead to the same cost,  $-m \log(\Delta)$ , for all models so doesn't affect which model is selected■

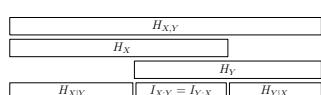
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

21

## Outline

1. Information Theory
2. KL-Divergence
3. Minimum Description Length
4. Variational Auto-Encoders



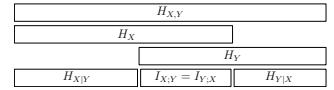
Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

23

## Outline

1. Information Theory
2. KL-Divergence
3. Minimum Description Length
4. Variational Auto-Encoders



Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

18

## Alice and Bob

- Suppose Alice has data  $\mathcal{D} = \{(\mathbf{x}_i, y_i) \mid i = 1, 2, \dots, m\}$  while Bob has only the feature vectors  $\{\mathbf{x}_i \mid i = 1, 2, \dots, m\}$ ■
- Alice wants to communicate  $y_i$  to Bob as efficiently as possible■
- We suppose Alice & Bob have available a model  $\hat{f}(\mathbf{x}|\theta)$ ■
- Rather than sending the complete list  $\{y_i \mid i = 1, 2, \dots, m\}$  Alice can send Bob the parameter  $\theta$  and the errors

$$\delta_i = y_i - \hat{f}(\mathbf{x}_i|\theta)$$

- Assuming the  $\delta_i$ 's have a distribution  $p_\delta$  then the cost of communicating an error to accuracy  $\Delta$  is  $-\log(p_\delta(\delta_i) \times \Delta)$ ■

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

20

## Minimum Description Length (MDL) Method

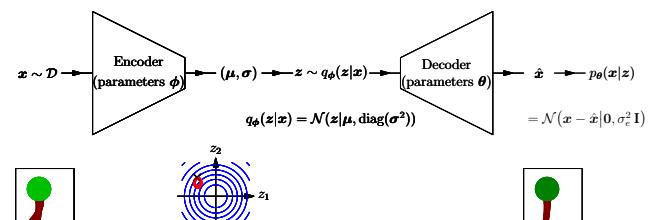
- The minimum description length method can be a powerful way of choosing between models■
- Often it is the only principled method available■
- It allows you to trade model accuracy against model complexity■
- It can be fiddly as we need to determine the accuracy to which we should store the parameters 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■

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

22

## Variational Auto-Encoders VAE



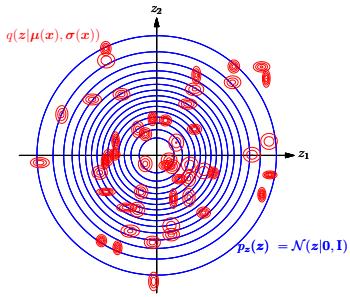
$$\mathcal{L} = \mathbb{E}_{x \sim \mathcal{D}} [\text{KL}(q_\theta(z|x) \parallel \mathcal{N}(\mathbf{0}, \mathbf{I})) - \log(p_\theta(x|z(x)))]$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

24

## Latent Space



$$\text{KL}(q_\theta(z|x) \parallel \mathcal{N}(0, I))$$

## 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  $q(z|x)$  which she communicates to Bob
- She also communicates the errors  $\delta = x - \bar{x}$
- Bob uses the decoder to decode  $q(z|x)$  and  $\delta$  to repair the images

## Description Length

- The loss

$$\mathcal{L} = \mathbb{E}_{x \sim D} [\text{KL}(q_\theta(z|x) \parallel \mathcal{N}(0, I)) - \log(p_\theta(x|z(x)))]$$

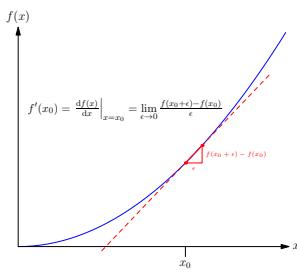
can be interpreted as

- The cost of communicating the code  $\text{KL}(q_\theta(z|x) \parallel \mathcal{N}(0, I))$
- Plus the cost to send the repair  $\log(p_\theta(x|z(x)))$
- We minimise the loss function equivalent to MDL
- What is really clever is that we can choose the accuracy of the code we send  $q_\theta(z|x)$  to minimise the over-all cost

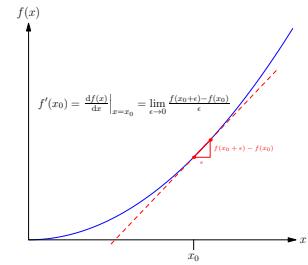
## Conclusions

- 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

### Differential Calculus



1. Why Calculus?
2. Differentiation
3. Vector and Matrix Calculus



*Differentiation, product and chain rules, vectors and matrices*

### Why Calculus?

### Back to Basics

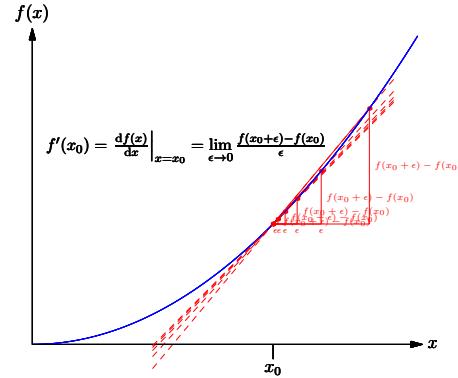
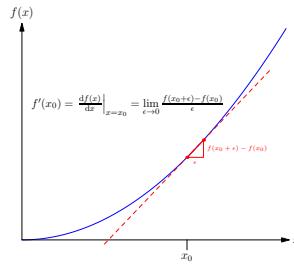
- Calculus is a fundamental tool of mathematical analysis
- In machine learning differentiation is fundamental tool in optimisation
- Integration is an essential tool in taking expectations over continuous distributions
- Both differentiation and integration crop up elsewhere
- This material will not be examined explicitly, but I assume elsewhere that you can do calculus

- 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

### Differentiation

1. Why Calculus?
2. Differentiation
3. Vector and Matrix Calculus

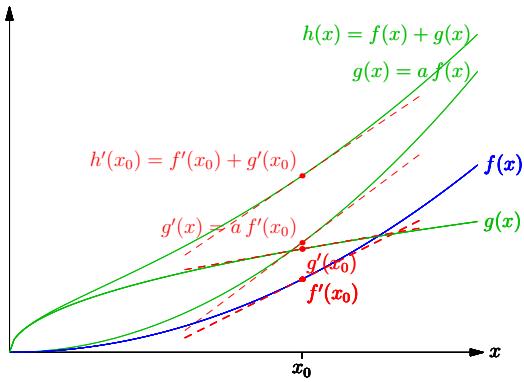


### Polynomials

### Linearity of derivatives

- $f(x) = x^2$
$$\frac{dx^2}{dx} = \lim_{\epsilon \rightarrow 0} \frac{(x + \epsilon)^2 - x^2}{\epsilon} = \lim_{\epsilon \rightarrow 0} \frac{(x^2 + 2\epsilon x + \epsilon^2) - x^2}{\epsilon} = \lim_{\epsilon \rightarrow 0} 2x + \epsilon = 2x$$
- $(x + \epsilon)^n = (x + \epsilon)(x + \epsilon) \cdots (x + \epsilon) = x^n + n\epsilon x^{n-1} + O(\epsilon^2)$
$$\frac{dx^n}{dx} = \lim_{\epsilon \rightarrow 0} \frac{(x + \epsilon)^n - x^n}{\epsilon} = \lim_{\epsilon \rightarrow 0} nx^{n-1} + O(\epsilon) = nx^{n-1}$$

- Note that  $f(x + \epsilon) = f(x) + \epsilon f'(x) + O(\epsilon^2)$  (from the definition of  $f'(x)$ )
$$\begin{aligned} \frac{d(af(x) + bg(x))}{dx} &= \lim_{\epsilon \rightarrow 0} \frac{(af(x + \epsilon) + bg(x + \epsilon)) - (af(x) + bg(x))}{\epsilon} \\ &= \lim_{\epsilon \rightarrow 0} \frac{a\epsilon f'(x) + b\epsilon g'(x) + O(\epsilon^2)}{\epsilon} \\ &= af'(x) + bg'(x) \end{aligned}$$
- Differentiation is a linear operation!



- Recall  $f(x + \epsilon) = f(x) + \epsilon f'(x) + O(\epsilon^2)$
- If  $h(x) = f(x)g(x)$

$$\begin{aligned} h'(x) &= \lim_{\epsilon \rightarrow 0} \frac{f(x + \epsilon)g(x + \epsilon) - f(x)g(x)}{\epsilon} \\ &= \lim_{\epsilon \rightarrow 0} \frac{(f(x) + \epsilon f'(x) + O(\epsilon^2))(g(x) + \epsilon g'(x) + O(\epsilon^2)) - f(x)g(x)}{\epsilon} \\ &= \lim_{\epsilon \rightarrow 0} \frac{\epsilon(f'(x)g(x) + f(x)g'(x)) + O(\epsilon^2)}{\epsilon} = f'(x)g(x) + f(x)g'(x) \end{aligned}$$

- 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

$$\begin{aligned} h(x + \epsilon) &= f(g(x + \epsilon)) = f(g(x) + \epsilon g'(x) + O(\epsilon^2)) \\ &= f(g(x)) + \epsilon g'(x) f'(g(x)) + O(\epsilon^2) \end{aligned}$$

- Thus

$$h'(x) = \lim_{\epsilon \rightarrow 0} \frac{h(x + \epsilon) - h(x)}{\epsilon} = g'(x) f'(g(x))$$

- This is the famous **chain rule**. Together with the product rule it means you can differentiate almost everything!

## 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

$$\frac{dg(f(x))}{dx} = f'(x)g'(f(x)) = 1$$

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

$$\frac{dg(y)}{dy} = \frac{1}{f'(g(y))} \quad \frac{df^{-1}(y)}{dy} = \frac{1}{f'(f^{-1}(y))}$$

## Functions of Exponentials

- What about  $f(x) = e^{cx}$

$$\frac{de^{cx}}{dx} = \frac{de^{cx}}{dcx} \frac{dcx}{dx} = ce^{cx}$$

- More generally using the chain rule

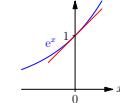
$$\frac{de^{g(x)}}{dx} = g'(x)e^{g(x)}$$

- Also  $a^{bc} = (a^b)^c$  (that is we multiply  $a$  together  $b \times c$  times)

$$\frac{da^x}{dx} = \frac{d(e^{\ln(a)x})}{dx} = \frac{de^{\ln(a)x}}{dx} = \ln(a)e^{\ln(a)x} = \ln(a)a^x$$

## Exponentials

- Note that  $a^{b+c} = a^b a^c$  (that is we multiply  $a$  together  $b+c$  times)



- Now  $e^\epsilon \approx (1 + \epsilon)$

- But  $e^{x+\epsilon} = e^x e^\epsilon = e^x(1 + \epsilon + O(\epsilon^2)) = e^x + \epsilon e^x + O(\epsilon^2)$

$$\frac{de^x}{dx} = \lim_{\epsilon \rightarrow 0} \frac{e^{x+\epsilon} - e^x}{\epsilon} = \lim_{\epsilon \rightarrow 0} \frac{\epsilon e^x + O(\epsilon^2)}{\epsilon} = e^x$$

## Natural Logarithms

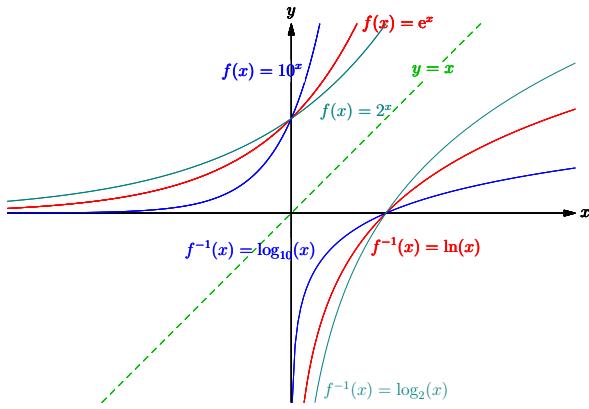
- The natural logarithm is defined as the inverse of  $e^x$

$$\ln(e^x) = x \quad e^{\ln(y)} = y$$

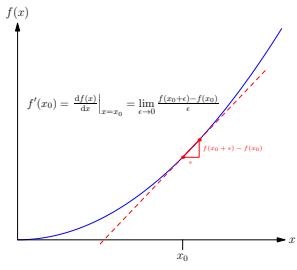
- Recall that if  $g(y) = f^{-1}(y)$  then  $g'(y) = 1/f'(g(y))$

- Consider  $g(y) = \ln(y)$  and  $f(x) = e^x$  (with  $f'(x) = e^x$ )

$$\frac{d\ln(y)}{dy} = \frac{1}{e^{\ln(y)}} = \frac{1}{y}$$



1. Why Calculus?
2. Differentiation
3. Vector and Matrix Calculus



## Derivatives in High Dimensions

- When working with functions  $f : \mathbb{R}^n \rightarrow \mathbb{R}$  in many dimensions then there will typically be different derivative in different directions
- To compute the derivative in a direction  $\mathbf{u} \in \mathbb{R}^n$  (where  $\|\mathbf{u}\| = 1$ ) at a point  $\mathbf{x} \in \mathbb{R}^n$  we use

$$\partial_{\mathbf{u}} F(\mathbf{x}) = \lim_{\epsilon \rightarrow 0} \frac{f(\mathbf{x} + \epsilon \mathbf{u}) - f(\mathbf{x})}{\epsilon}$$

- If  $\mathbf{u} = \delta_i = (0, \dots, 0, 1, 0, \dots, 0)$  (i.e.  $u_i = 1$ ) then

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \lim_{\epsilon \rightarrow 0} \frac{f(\mathbf{x} + \epsilon \delta_i) - f(\mathbf{x})}{\epsilon}$$

## Computing Gradients 1

- We can compute the gradient by writing out  $f(\mathbf{x})$  componentwise and performing the partial derivative with respect to  $x_i$

$$\nabla w^T M w = \left( \begin{array}{c} \frac{\partial}{\partial w_1} \\ \frac{\partial}{\partial w_2} \\ \frac{\partial}{\partial w_3} \\ \vdots \end{array} \right) \sum_{i,j} w_i M_{ij} w_j = \left( \begin{array}{c} \sum_j M_{1j} w_j + \sum_i w_i M_{i1} \\ \sum_j M_{2j} w_j + \sum_i w_i M_{i2} \\ \sum_j M_{3j} w_j + \sum_i w_i M_{i3} \\ \vdots \end{array} \right) = Mw + M^T w$$

- 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

## Differentiating Matrices

- Often we have loss functions with respect to a matrix  $\mathbf{W}$ , e.g.

$$L(\mathbf{W}) = (\mathbf{a}^T \mathbf{W} \mathbf{b} - c)^2$$

- 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  $\mathbf{U}$

$$L(\mathbf{W}^* + \epsilon \mathbf{U}) - L(\mathbf{W}^*) = O(\epsilon^2)$$

## Computing Gradients 2

- A slicker way is just to expand  $f(\mathbf{x} + \epsilon \mathbf{u})$

- Consider  $f(\mathbf{x}) = \mathbf{x}^T \mathbf{M} \mathbf{x} + \mathbf{a}^T \mathbf{x}$

$$\begin{aligned} f(\mathbf{x} + \epsilon \mathbf{u}) &= (\mathbf{x} + \epsilon \mathbf{u})^T \mathbf{M} (\mathbf{x} + \epsilon \mathbf{u}) + \mathbf{a}^T (\mathbf{x} + \epsilon \mathbf{u}) \\ &= f(\mathbf{x}) + \epsilon (\mathbf{u}^T \mathbf{M} \mathbf{x} + \mathbf{x}^T \mathbf{M} \mathbf{u} + \mathbf{a}^T \mathbf{u}) + O(\epsilon^2) \\ &= f(\mathbf{x}) + \epsilon \mathbf{u}^T (\mathbf{M} \mathbf{x} + \mathbf{M}^T \mathbf{x} + \mathbf{a}) + O(\epsilon^2) \end{aligned}$$

using  $\mathbf{x}^T \mathbf{M} \mathbf{u} = \mathbf{u}^T \mathbf{M}^T \mathbf{x}$  and  $\mathbf{a}^T \mathbf{u} = \mathbf{u}^T \mathbf{a}$

- But  $f(\mathbf{x} + \epsilon \mathbf{u}) = f(\mathbf{x}) + \epsilon \mathbf{u}^T \nabla f(\mathbf{x}) + O(\epsilon^2)$  so

$$\nabla f(\mathbf{x}) = \mathbf{M} \mathbf{x} + \mathbf{M}^T \mathbf{x} + \mathbf{a}$$

## 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

$$L(\mathbf{W} + \epsilon \mathbf{U}) = L(\mathbf{W}) + \epsilon \text{tr} \mathbf{U}^T \frac{\partial L(\mathbf{W})}{\partial \mathbf{W}} + O(\epsilon^2)$$

where

$$\text{tr} \mathbf{U}^T \mathbf{G} = \sum_i [\mathbf{U}^T \mathbf{G}]_{ii} = \sum_{ij} U_{ji} G_{ji} = \sum_{ij} U_{ij} G_{ij} = \langle \mathbf{U}, \mathbf{G} \rangle$$

## Example

- Suppose

$$L(\mathbf{W}) = (\mathbf{a}^\top \mathbf{W} \mathbf{b} - c)^2$$

then

$$\begin{aligned} L(\mathbf{W} + \epsilon \mathbf{U}) &= (\mathbf{a}^\top (\mathbf{W} + \epsilon \mathbf{U}) \mathbf{b} - c)^2 = (\mathbf{a}^\top \mathbf{W} \mathbf{b} + \epsilon \mathbf{a}^\top \mathbf{U} \mathbf{b} - c)^2 \\ &= L(\mathbf{W}) + 2\epsilon (\mathbf{a}^\top \mathbf{W} \mathbf{b} - c) (\mathbf{a}^\top \mathbf{U} \mathbf{b}) + O(\epsilon^2) \end{aligned}$$

- Now

$$\mathbf{a}^\top \mathbf{U} \mathbf{b} = \sum_{ij} a_i U_{ij} b_j = \sum_{ij} U_{ji} a_j b_i = \text{tr } \mathbf{U}^\top \mathbf{a} \mathbf{b}^\top$$

$$\text{Thus } \frac{\partial L(\mathbf{W})}{\partial \mathbf{W}} = 2(\mathbf{a}^\top \mathbf{W} \mathbf{b} - c) \mathbf{a} \mathbf{b}^\top$$

## Traces

- The trace of a matrix is the sum of its diagonal elements

$$\text{tr } \mathbf{A} = \text{tr } \mathbf{A}^\top = \sum_i A_{ii}$$

- Clearly  $\text{tr } c \mathbf{A} = c \text{tr } \mathbf{A}$

- Also  $\text{tr } (\mathbf{A} + \mathbf{B}) = \text{tr } \mathbf{A} + \text{tr } \mathbf{B}$

- We note that

$$\text{tr } \mathbf{A} \mathbf{B} = \sum_{i,j} A_{ij} B_{ji} = \sum_{i,j} B_{ij} A_{ji} = \text{tr } \mathbf{B} \mathbf{A}$$

- It follows that

$$\text{tr } \mathbf{A} \mathbf{B} \mathbf{C} \mathbf{D} = \text{tr } \mathbf{D} \mathbf{A} \mathbf{B} \mathbf{C} = \text{tr } \mathbf{C} \mathbf{D} \mathbf{A} \mathbf{B} = \text{tr } \mathbf{B} \mathbf{C} \mathbf{D} \mathbf{A}$$

## Quick Matrix Differentiation

- Let

$$\partial_{\mathbf{U}} f(\mathbf{X}) = \lim_{\epsilon \rightarrow 0} \frac{f(\mathbf{X} + \epsilon \mathbf{U}) - f(\mathbf{X})}{\epsilon} = \text{tr } \mathbf{U}^\top \frac{\partial f(\mathbf{X})}{\partial \mathbf{X}}$$

- E.g.

$$\begin{aligned} \partial_{\mathbf{U}} \text{tr } \mathbf{A} \mathbf{X} \mathbf{B} &= \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} \text{tr } \mathbf{A} (\mathbf{X} + \epsilon \mathbf{U}) \mathbf{B} - \text{tr } \mathbf{A} \mathbf{X} \mathbf{B} \\ &= \text{tr } \mathbf{A} \mathbf{U} \mathbf{B} = \text{tr } \mathbf{B}^\top \mathbf{U}^\top \mathbf{A}^\top = \text{tr } \mathbf{U}^\top \mathbf{A}^\top \mathbf{B}^\top \end{aligned}$$

thus

$$\frac{\partial \text{tr } \mathbf{A} \mathbf{X} \mathbf{B}}{\partial \mathbf{X}} = \mathbf{A}^\top \mathbf{B}^\top$$

## Log Determinants

- We often come across logarithms of determinants of matrices,  $\log(|\mathbf{M}|)$
- For GP we want to choose  $\mathbf{K}$  to maximise the marginal likelihood,  $\log(|\mathbf{K} + \sigma^2 \mathbf{I}|)$
- To find the derivative of  $\log(|\mathbf{X}|)$  we consider

$$\begin{aligned} \log(|\mathbf{X} + \epsilon \mathbf{U}|) &= \log(|\mathbf{X}(\mathbf{I} + \epsilon \mathbf{X}^{-1} \mathbf{U})|) \\ &= \log(|\mathbf{X}| |\mathbf{I} + \epsilon \mathbf{X}^{-1} \mathbf{U}|) \\ &= \log(|\mathbf{X}|) + \log(|\mathbf{I} + \epsilon \mathbf{X}^{-1} \mathbf{U}|) \end{aligned}$$

★ Using  $|\mathbf{AB}| = |\mathbf{A}| |\mathbf{B}|$

★ Using  $\log(ab) = \log(a) + \log(b)$

## Determinants

$$|\mathbf{I} + \epsilon \mathbf{M}| = \begin{vmatrix} 1 + \epsilon M_{11} & \epsilon M_{12} \\ \epsilon M_{21} & 1 + \epsilon M_{22} \end{vmatrix} = (1 + \epsilon M_{11})(1 + \epsilon M_{22}) - \epsilon^2 M_{21} M_{12} \\ = 1 + \epsilon(M_{11} + M_{22}) + O(\epsilon^2)$$

$$|\mathbf{I} + \epsilon \mathbf{M}| = \begin{vmatrix} 1 + \epsilon M_{11} & \epsilon M_{12} & \epsilon M_{13} & \cdots & \epsilon M_{1n} \\ \epsilon M_{21} & 1 + \epsilon M_{22} & \epsilon M_{23} & \cdots & \epsilon M_{2n} \\ \epsilon M_{31} & \epsilon M_{32} & 1 + \epsilon M_{33} & \cdots & \epsilon M_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \epsilon M_{n1} & \epsilon M_{n2} & \epsilon M_{n3} & \cdots & 1 + \epsilon M_{nn} \end{vmatrix} = (1 + \epsilon M_{11})(1 + \epsilon M_{22})(1 + \epsilon M_{33}) \cdots (1 + \epsilon M_{nn}) C_{11} C_{22} \cdots C_{nn} + \epsilon M_{12} C_{11} C_{23} \cdots C_{nn} - \epsilon M_{41} C_{41} + \epsilon M_{51} C_{51} + \cdots$$

## Putting it Together

- Recall

$$\begin{aligned} \log(|\mathbf{X} + \epsilon \mathbf{U}|) - \log(|\mathbf{X}|) &= \log(|\mathbf{I} + \epsilon \mathbf{X}^{-1} \mathbf{U}|) \\ &= \log(1 + \epsilon \text{tr } \mathbf{X}^{-1} \mathbf{U} + O(\epsilon)^2) \\ &= \epsilon \text{tr } \mathbf{X}^{-1} \mathbf{U} + O(\epsilon)^2 \\ &= \epsilon \text{tr } \mathbf{U}^\top (\mathbf{X}^{-1})^\top + O(\epsilon) \end{aligned}$$

using  $\log(1 + x) = x + \frac{x^2}{2} + \cdots$

- Thus  $\partial_{\mathbf{U}} \log(|\mathbf{X}|) = \text{tr } \mathbf{U}^\top (\mathbf{X}^{-1})^\top$

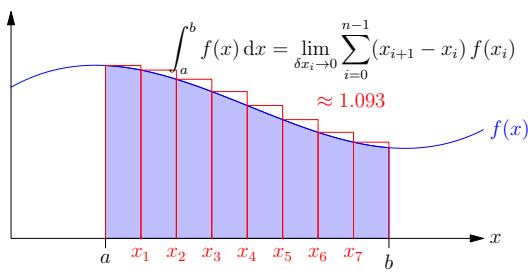
- Or

$$\frac{\partial \log(|\mathbf{X}|)}{\partial \mathbf{X}} = (\mathbf{X}^{-1})^\top$$

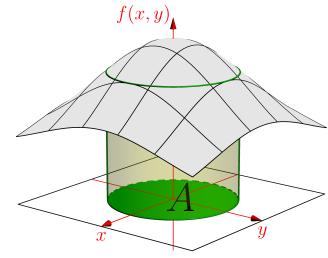
## Summary

- 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

### Integral Calculus



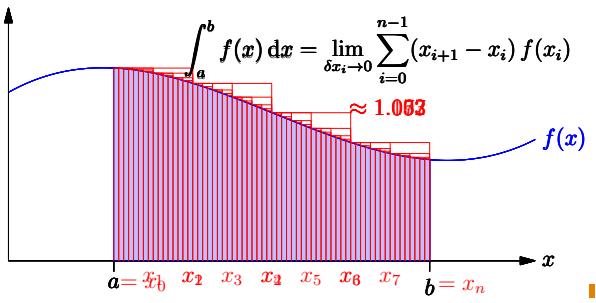
Riemann Integration, integration by parts, gaussian integrals



1. Defining Integrals
2. Doing Integrals
3. Gaussian Integrals

### Riemann Integral

- Integrals represent area beneath a curve



### Linearity of Integration

- Integration is a linear operator

$$\begin{aligned} \int_a^b (rf(x) + sg(x)) dx &= \lim_{\delta x_i \rightarrow 0} \sum_{i=0}^{n-1} (x_{i+1} - x_i) (rf(x_i) + sg(x_i)) \\ &= \lim_{\delta x_i \rightarrow 0} \left( \sum_{i=0}^{n-1} (x_{i+1} - x_i) rf(x_i) + \sum_{i=0}^{n-1} (x_{i+1} - x_i) sg(x_i) \right) \\ &= \lim_{\delta x_i \rightarrow 0} \left( r \sum_{i=0}^{n-1} (x_{i+1} - x_i) f(x_i) + s \sum_{i=0}^{n-1} (x_{i+1} - x_i) g(x_i) \right) \\ &= r \lim_{\delta x_i \rightarrow 0} \sum_{i=0}^{n-1} (x_{i+1} - x_i) f(x_i) + s \lim_{\delta x_i \rightarrow 0} \sum_{i=0}^{n-1} (x_{i+1} - x_i) g(x_i) \\ &= r \int_a^b f(x) dx + s \int_a^b g(x) dx \end{aligned}$$

### Fundamental Law of Calculus

- Let

$$I(a, x) = \int_a^x f(z) dz = \lim_{\delta z_i \rightarrow 0} \sum_{i=0}^{n-1} (z_{i+1} - z_i) f(z_i)$$

- Now for small  $\delta x$

$$I(a, x + \delta x) = \int_a^{x+\delta x} f(z) dz = \lim_{\delta z_i \rightarrow 0} \sum_{i=0}^{n-1} (z_{i+1} - z_i) f(z_i) + \delta x f(x)$$

- Thus

$$\frac{dI(a, x)}{dx} = \lim_{\delta x \rightarrow 0} \frac{I(x + \delta x) - I(x)}{\delta x} = \lim_{\delta x \rightarrow 0} \frac{\delta x f(x)}{\delta x} = f(x)$$

### The Other Way Around

- Consider

$$\begin{aligned} \int_a^b \frac{df(x)}{dx} dx &= \int_a^b \lim_{\delta x \rightarrow 0} \frac{f(x + \delta x) - f(x)}{\delta x} dx \\ &= \lim_{x_{i+1} - x_i \rightarrow 0} \sum_{i=0}^{n-1} (x_{i+1} - x_i) \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} \\ &= \lim_{x_{i+1} - x_i \rightarrow 0} \sum_{i=0}^{n-1} (f(x_{i+1}) - f(x_i)) \\ &= (f(x_1) - f(x_0)) + (f(x_2) - f(x_1)) + (f(x_3) - f(x_2)) + \dots \\ &\quad + (f(x_{n-1}) - f(x_{n-2})) + (f(x_n) - f(x_{n-1})) \\ &= f(x_n) - f(x_0) = f(b) - f(a) \end{aligned}$$

- We can think of integration as an **anti-derivative** it undoes differentiation

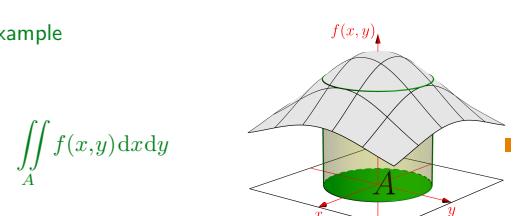
### Indefinite Integrals

- 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  $F(x) = \int f(x) dx$
- 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

$$\int_a^b f(x) dx = F(b) - F(a)$$

### Multiple Integrals

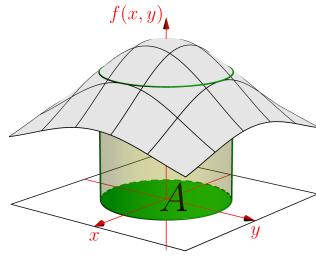
- For functions involving many independent variables (e.g.  $f(x, y)$ ,  $f(x, y, z)$ ,  $f(\mathbf{x})$ ) we can integrate over multiple dimensions
- For example



- It gets tedious writing multiple integral signs and I tend to write just one

$$\int \dots \int f(x_1, x_2, \dots, x_n) dx_1 dx_2 \dots dx_n = \int f(\mathbf{x}) d\mathbf{x}$$

1. Defining Integrals
2. Doing Integrals
3. Gaussian Integrals



### 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

$$\int f(x)g(x)dx = ? \quad \int f(g(x))dx = ?$$

- Unfortunately, unlike differentiation we don't have a small parameter we can expand in.
- In general integration is hard.

### Example of Integration by Parts

- Consider

$$\begin{aligned} \Pi(z) &= \int_0^\infty x^z e^{-x} dx = \int_0^\infty x^z \frac{d(-e^{-x})}{dx} dx \\ &= [x^z (-e^{-x})]_0^\infty - \int_0^\infty \frac{dx^z}{dx} (-e^{-x}) dx \\ &= \int_0^\infty (zx^{z-1}) e^{-x} dx = z \int_0^\infty x^{z-1} e^{-x} dx = z\Pi(z-1) \end{aligned}$$

- Thus  $\Pi(z) = z\Pi(z-1)$ , but

$$\Pi(0) = \int_0^\infty e^{-z} dz = [-e^{-x}]_0^\infty = -e^{-\infty} - (-e^0) = 1$$

- Now

$$\Pi(n) = n\Pi(n-1) = n(n-1)\Pi(n-2) = n(n-1)(n-2)\dots 1 = n!$$

### Example of Integration by Substitution

- We consider  $I(n) = \int_0^\infty x^n e^{-x^2/2} dx$

- Let  $u(x) = x^2/2$  or  $x(u) = \sqrt{2u}$  so that

$$\frac{dx(u)}{du} = \frac{1}{\sqrt{2u}} \quad u(0) = 0 \quad u(\infty) = \infty$$

- Thus

$$\begin{aligned} I(n) &= \int_0^\infty (\sqrt{2u})^n e^{-u} \frac{1}{\sqrt{2u}} du \\ &= 2^{\frac{n-1}{2}} \int_0^\infty u^{\frac{n-1}{2}} e^{-u} du = 2^{\frac{n-1}{2}} \Pi\left(\frac{n-1}{2}\right) \end{aligned}$$

- $I(1) = 1$ ,  $I(3) = 2 \times 1! = 2$ ,  $I(5) = 2^2 \times 2! = 8$ , but  
 $I(0) = \Pi(-1/2)/\sqrt{2}$ ,  $I(2) = \sqrt{2}\Pi(1/2) = \Pi(-1/2)/\sqrt{2}$

- A key method for performing integrals is through knowledge of the anti-derivative.

- If we know  $F'(x) = f(x)$  then  $F(x) + c = \int f(x)dx$

- E.g. we know that  $dx^n/dx = nx^{n-1}$  therefore

$$\int x^{n-1} dx = \frac{1}{n} \int \frac{dx^n}{dx} dx = \frac{x^n}{n} + c$$

and

$$\int_a^b x^{n-1} dx = \frac{b^n}{n} - \frac{a^n}{n}$$

### Integration by Parts

- Recall the product rule  $\frac{df(x)g(x)}{dx} = \frac{df(x)}{dx}g(x) + f(x)\frac{dg(x)}{dx}$

- Integrating we get

$$\begin{aligned} \int_a^b \frac{df(x)g(x)}{dx} dx &= \int_a^b \frac{df(x)}{dx} g(x) dx + \int_a^b f(x) \frac{dg(x)}{dx} dx \\ &= [f(x)g(x)]_a^b = f(b)g(b) - f(a)g(a) \end{aligned}$$

- Unfortunately we get two integrals, but we can turn this around

$$\int_a^b f(x) \frac{dg(x)}{dx} dx = [f(x)g(x)]_a^b - \int_a^b \frac{df(x)}{dx} g(x) dx$$

whether this is helpful depends on  $f(x)$  and  $g(x)$

### Substitution

- We can make a transformation from  $x$  to  $u = u(x)$

$$\begin{aligned} \int_a^b f(x) dx &= \lim_{\delta x_i \rightarrow 0} \sum_{i=0}^{n-1} f(x_i)(x_{i+1} - x_i) \\ &= \lim_{\delta u_i \rightarrow 0} \sum_{i=0}^{n-1} f(x(u_i)) \frac{x(u_{i+1}) - x(u_i)}{u_{i+1} - u_i} (u_{i+1} - u_i) \\ &= \int_{u(a)}^{u(b)} f(x(u)) \frac{dx(u)}{du} du \end{aligned}$$

\* where  $u_i$  is such that  $x(u_i) = x_i$  or  $u_i = u(x_i)$  where  $u(x)$  is the inverse of  $x(u)$ .

$$* \text{ using } \lim_{\delta u_i \rightarrow 0} \frac{x(u_{i+1}) - x(u_i)}{u_{i+1} - u_i} = \frac{dx(u_i)}{du}$$

### Changing Variables in Multidimensional Space

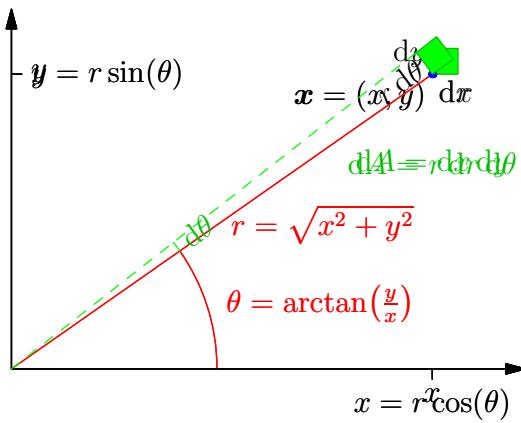
- When changing variables in many dimensions  $\mathbf{x} \rightarrow \mathbf{u}$  the change of variables involves the Jacobian

$$\int f(\mathbf{x}) d\mathbf{x} = \int f(\mathbf{x}(\mathbf{u})) |\det(\mathbf{J})| d\mathbf{u}, \quad \mathbf{J} = \begin{pmatrix} \frac{\partial x_1}{\partial u_1} & \frac{\partial x_1}{\partial u_2} & \dots & \frac{\partial x_1}{\partial u_n} \\ \frac{\partial x_2}{\partial u_1} & \frac{\partial x_2}{\partial u_2} & \dots & \frac{\partial x_2}{\partial u_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial x_n}{\partial u_1} & \frac{\partial x_n}{\partial u_2} & \dots & \frac{\partial x_n}{\partial u_n} \end{pmatrix}$$

- E.g. transforming from Cartesian coordinates  $(x, y)$  to polar coordinates  $(r, \theta)$  then  $x = r\cos(\theta)$  and  $y = r\sin(\theta)$

$$\begin{aligned} |\det(\mathbf{J})| &= \left| \det \begin{pmatrix} \frac{\partial r\cos(\theta)}{\partial r} & \frac{\partial r\cos(\theta)}{\partial \theta} \\ \frac{\partial r\sin(\theta)}{\partial r} & \frac{\partial r\sin(\theta)}{\partial \theta} \end{pmatrix} \right| = \left| \det \begin{pmatrix} \cos(\theta) & -r\sin(\theta) \\ \sin(\theta) & r\cos(\theta) \end{pmatrix} \right| \\ &= r(\cos^2(\theta) + \sin^2(\theta)) = r \end{aligned}$$

- That is,  $dxdy = rdrd\theta$



- A trick that sometimes works is differentiating through an integral, e.g. consider finding moments

$$M_n = \mathbb{E}[X^n] = \int_{-\infty}^{\infty} x^n f_X(x) dx$$

- We can define a momentum generating function

$$Z(\ell) = \int_{-\infty}^{\infty} e^{\ell x} f_X(x) dx$$

- Then  $M_n = Z^{(n)}(0)$

$$\left. \frac{d^n Z(\ell)}{d\ell^n} \right|_{\ell=0} = \int_{-\infty}^{\infty} \left. \frac{d^n e^{\ell x}}{d\ell^n} \right|_{\ell=0} f_X(x) dx = \int_{-\infty}^{\infty} x^n f_X(x) dx = M_n$$

## Cumulant Generating Function

- Note that  $e^{\ell x} = 1 + \ell x + \frac{1}{2}\ell^2 x^2 + \frac{1}{3!}\ell^3 x^3 + \dots$
  - So
- $$Z(\ell) = \int_{-\infty}^{\infty} e^{\ell x} f_X(x) dx = 1 + \ell M_1 + \frac{1}{2}\ell^2 M_2 + \frac{1}{3!}\ell^3 M_3 + \dots$$
- Now using  $\log(1 + \epsilon) = \epsilon - \frac{1}{2}\epsilon^2 + \frac{1}{3}\epsilon^3 + \dots$
- $$G(\ell) = \log(Z(\ell)) = \ell M_1 + \frac{1}{2}\ell^2(M_2 - M_1^2) + \frac{1}{3!}\ell^3(M_3 - 3M_2 M_1 + 2M_1^3) + \dots$$
- So that  $\kappa_n = G^{(n)}(0)$ , with  $\kappa_1 = M_1$  (the mean),  $\kappa_2 = M_2 - M_1^2$  (the variance),  $\kappa_3 = M_3 - 3M_2 M_1 + 2M_1^3$  (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 functions. 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

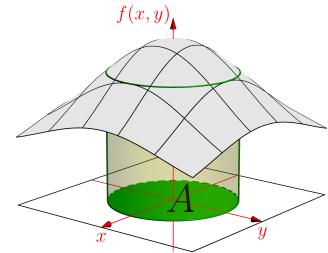
- There are integrals with no known closed form solution
  - We saw that  $\Pi(z) = \int_0^\infty x^z e^{-x} dx$  satisfies  $\Pi(z) = z\Pi(z-1)$
  - For integer  $n$  then  $\Pi(n) = n!$  but for general  $z$ , the integral  $\Pi(z)$  can't be written in terms of elementary functions
  - We consider  $\Pi(z)$  as a special function in its own right
  - Although, history has left us with the gamma function instead
- $$\Gamma(z) = \int_0^\infty x^{z-1} e^{-x} dx = \Pi(z-1)$$
- Other special function defined by integrals exist (e.g. the Bessel, Airy, hypergeometric, elliptic, error functions, . . . )

## Gaussian Integrals

- Gaussian integrals are integrals involving  $e^{-x^2}$ , e.g.
- $$\int_{-\infty}^{\infty} e^{-x^2} dx \quad \int_{-\infty}^{\infty} x^4 e^{-ax^2-bx} dx$$
- They are important in computing integrals with respect to the normal distribution
- $$\mathcal{N}(x|\mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}$$
- The great news is that these integrals are all doable
  - The bad news is that they are quite tricky to do

## Outline

1. Defining Integrals
2. Doing Integrals
3. Gaussian Integrals



## The Gaussian Integral

- The integral over a Gaussian is surprisingly difficult

$$I_1 = \int_{-\infty}^{\infty} e^{-x^2/2} dx$$

- There is a nice trick which is to consider

$$I_1^2 = \int_{-\infty}^{\infty} e^{-x^2/2} dx \int_{-\infty}^{\infty} e^{-y^2/2} dy = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} e^{-(x^2+y^2)/2} dx dy$$

- 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  $x^2 + y^2 = r^2$ )

$$I_1^2 = \int_0^{2\pi} d\theta \int_0^{\infty} r e^{-r^2/2} dr = 2\pi \int_0^{\infty} r e^{-r^2/2} dr$$

## The Gaussian Integral Continued

## Normal Distribution

- From before

$$I_1^2 = 2\pi \int_0^\infty r e^{-r^2/2} dr$$

- Finally let  $u = r^2/2$  so that  $du/dr = r$  or  $du = r dr$  we get

$$I_1^2 = 2\pi \int_0^\infty e^{-u} du = 2\pi$$

- So that  $I_1 = \sqrt{2\pi}$

- Incidentally,  $I_1 = \sqrt{2}\Pi(-1/2)$  so  $\Pi(-1/2) = \Gamma(1/2) = \sqrt{\pi}$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

25

$$I_2 = \int_{-\infty}^\infty e^{-(x-\mu)^2/(2\sigma^2)} dx$$

- Making the change of variables  $z = (x - \mu)/\sigma$  so that  $dz = dx/\sigma$  or  $dx = \sigma dz$ . Then

$$I_2 = \sigma \int_{-\infty}^\infty e^{-z^2/2} dz = \sigma I_1 = \sqrt{2\pi}\sigma$$

- Note that the *probability density function* (PDF) for a normally distributed random variable is given by

$$\mathcal{N}(x | \mu, \sigma^2) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

26

## Multi-dimensional Gaussians

- Consider

$$I_3 = \int_{-\infty}^\infty \cdots \int_{-\infty}^\infty e^{-\frac{1}{2}\|\mathbf{x}\|_2^2} d\mathbf{x}_1 \cdots d\mathbf{x}_n$$

where  $\mathbf{x} = (x_1, x_2, \dots, x_n)^\top$

- Note that  $\|\mathbf{x}\|_2^2 = x_1^2 + x_2^2 + \cdots + x_n^2$  and using  $e^{\sum_i a_i} = \prod_i e^{a_i}$

$$\begin{aligned} I_3 &= \int_{-\infty}^\infty \cdots \int_{-\infty}^\infty e^{-\frac{1}{2} \sum_{i=1}^n x_i^2} d\mathbf{x}_1 \cdots d\mathbf{x}_n \\ &= \prod_{i=1}^n \int_{-\infty}^\infty e^{-x_i^2/2} dx_i = \prod_{i=1}^n \sqrt{2\pi} = (2\pi)^{n/2} \end{aligned}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

27

## Full Multi-variate Normal

- Consider

$$I_4 = \int_{-\infty}^\infty \cdots \int_{-\infty}^\infty e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})} d\mathbf{x}_1 \cdots d\mathbf{x}_n$$

- Let  $\boldsymbol{\Sigma}^{-1} = \mathbf{V}\boldsymbol{\Lambda}^{-1}\mathbf{V}^\top$  and make the change of variables  $\mathbf{y} = \mathbf{V}^\top(\mathbf{x}-\boldsymbol{\mu})$

- The Jacobian  $\mathbf{J}$  has elements (note that  $\mathbf{x} = \mathbf{V}\mathbf{y} + \boldsymbol{\mu}$ )

$$J_{ij} = \frac{\partial x_i}{\partial y_j} = \frac{\partial}{\partial y_j} \left( \sum_{k=1}^n V_{ik} y_k + \mu_i \right) = V_{ij}$$

- So that  $\mathbf{J} = \mathbf{V}$  and consequently  $|\det(\mathbf{J})| = |\det(\mathbf{V})| = 1$  then

$$I_4 = \int_{-\infty}^\infty \cdots \int_{-\infty}^\infty e^{-\frac{1}{2}\mathbf{y}^\top \boldsymbol{\Lambda}^{-1} \mathbf{y}} dy_1 \cdots dy_n = \prod_{i=1}^n \int_{-\infty}^\infty e^{-y_i^2/(2\lambda_i)} dy_i = \prod_i \sqrt{2\pi\lambda_i}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

28

## Determinants

- Using the facts, that  $\boldsymbol{\Sigma} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top$  then

$$\det(\boldsymbol{\Sigma}) = \det(\mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\top) = \det(\mathbf{V}) \det(\boldsymbol{\Lambda}) \det(\mathbf{V}^\top) = \det(\boldsymbol{\Lambda}) = \prod_{i=1}^n \lambda_i$$

using  $\det(\mathbf{AB}) = \det(\mathbf{A}) \det(\mathbf{B})$  and  $\det(\mathbf{V}) = 1$

- Recall  $I_4 = \prod_i \sqrt{2\pi\lambda_i} = (2\pi)^{n/2} \sqrt{\det(\boldsymbol{\Sigma})}$

- We note for an  $n \times n$  matrix  $\mathbf{M}$  then  $\det(c\mathbf{M}) = c^n \det(\mathbf{M})$  so that

$$I_4 = (2\pi)^{n/2} \sqrt{\det(\boldsymbol{\Sigma})} = \sqrt{\det(2\pi\boldsymbol{\Sigma})}$$

- Finally, we get that for the PDF of a normal to integrate to 1

$$\mathcal{N}(\mathbf{x} | \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{\sqrt{\det(2\pi\boldsymbol{\Sigma})}} e^{-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^\top \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

29

## 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

Adam Prügel-Bennett

COMP6208 Advanced Machine Learning

30