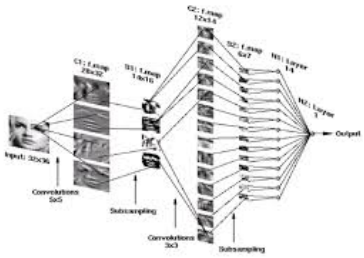
Advanced Machine Learning

Course Outline



When ML Works, Bias Variance

Advanced Machine Learning

- I've changed this course this year
  - Removing a lot of deep learning material (taught elsewhere)
  - I have made it much more grounded on theory
- This course will introduce you to the most successful machine learning methods currently available
- These tend to be mathematically sophisticated
- This is unapologetically an advanced course (the give away is in the name)

Course Structure

- Lectures
  - 15:00–15:45 Monday 7/3009 (L/T A)—lecture
  - 9:00–9:45 Tuesday 65/1133—lecture
  - 12:00–12:45 Friday 7/3009 (L/T A)—project work
- Assessment
  - 60% exam
  - 40% group project (split into two reports)

Group Projects

- There is a substantial group project (60 hours)
- You form groups of 3–4 people
- You come up with your own projects (look at Kaggle competition if you have no ideas)
- Each group will write a report of 3 pages
  - On data exploration by end of week 8
  - On evalutating ML by end of week 11
- We will use the Friday session to support the groups

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
- 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, structure and regularisation
  - Ensembling, bagging and boosting
- Mathematics
  - Function Spaces: Kernel Methods and Gaussian Processes
  - Linear Algebra, embeddings, positive definiteness, subspace, determinants

Topics Continued

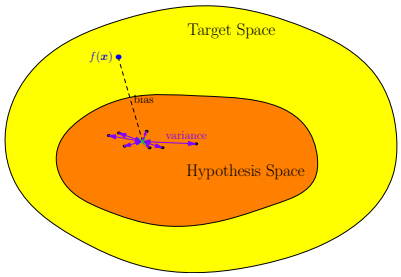
- Optimisation
  - Newton/Quasi-Newton Methods: convergence rates
  - SGD, momentum, ADAM
- Constrainted Optimisation
  - KKT conditions
  - Duality Linear/Quadratic Programming
  - SVMs
- Convexity
  - Convex sets: linear constraints, PD matrices
  - Convex functions
  - SVMs, Lasso
  - Jensen's inequality

Topics Continued

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

Advanced Machine Learning

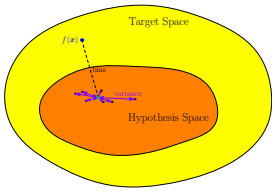
When Machine Learning Works



When ML Works, Bias Variance

Outline

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



What Makes a Good Learning Machine?

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

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

What Makes Machine Learning Hard?

- Typically 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 function $f(\boldsymbol{x})$
- We construct a learning machine that makes a prediction $\hat{f}(\boldsymbol{x}|\mathcal{D})$
- We typically choose the weights to minimise a *training error*

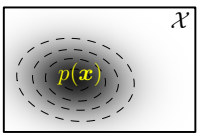
$$E_T(\mathcal{D}) = \sum_{\boldsymbol{x}\in\mathcal{D}} \left(\hat{f}(\boldsymbol{x}|\mathcal{D}) - f(\boldsymbol{x})\right)^2 = \sum_{i=1}^{m} \left(\hat{f}(\boldsymbol{x}_i|\mathcal{D}) - y_i\right)^2$$

where $\mathcal{D} = \{(\boldsymbol{x}_i, y_i = f(\boldsymbol{x}_i))\}_{i=1}^m$ is a set of size m, sampled from the set of all inputs, $\mathcal{X}$, according to a probability distribution $p(\boldsymbol{x})$ describing where our data is

Generalisation Error

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

$$E_G(\mathcal{D}) = \sum_{\boldsymbol{x}\in\mathcal{X}} p(\boldsymbol{x}) \left(\hat{f}(\boldsymbol{x}|\mathcal{D}) - f(\boldsymbol{x})\right)^2$$

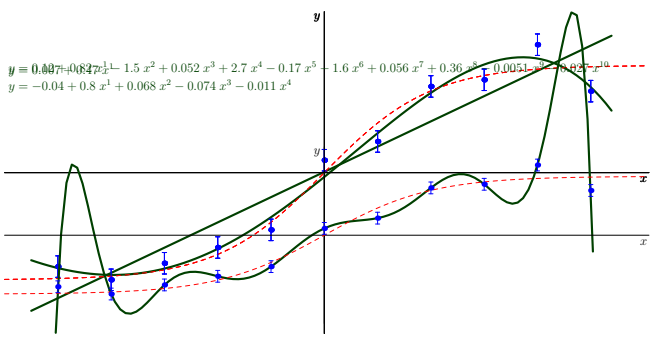


(we can estimate this if we have some examples with known labels $y_i = f(\boldsymbol{x}_i)$ which we have not trained on)

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

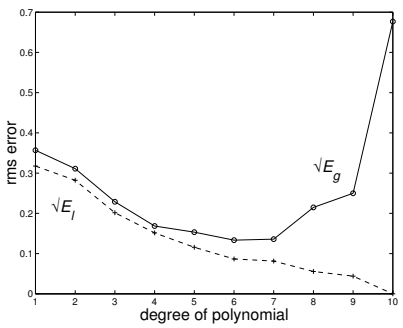
Too Simple or Too Complex?

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

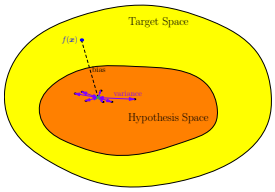


Measuring Generalisation Error for Regression

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



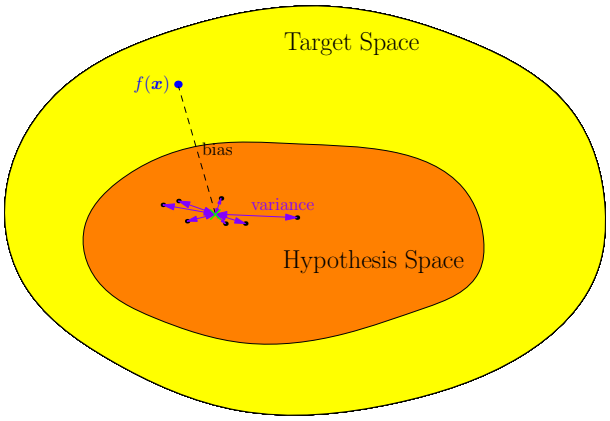
Outline



Expected Generalisation Performance

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

Approximation and Estimation Errors



Mean Machine

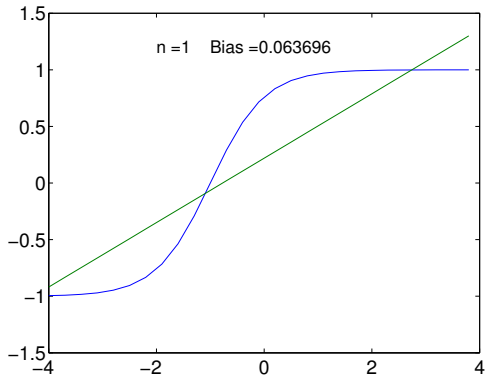
- To help understand generalisation we can consider the mean prediction with respect to machines trained with all data sets of size m

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

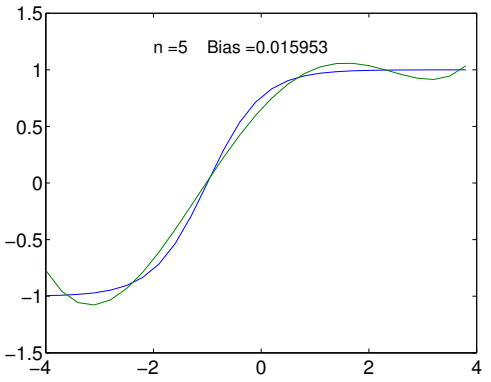
- We can define the **bias** to be generalisation performance of the mean machine

$$B = \sum_{\boldsymbol{x}\in\mathcal{X}} p(\boldsymbol{x})\left(\hat{f}_m(\boldsymbol{x}) - f(\boldsymbol{x})\right)^2$$

Regression Example $n = 1$



Regression Example $n = 5$



Bias and Variance

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

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

Cross Term

- The cross term vanishes

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

- Thus

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

Bias and Variance

- We can write the expected generalisation as

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

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

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

Bias-Variance Dilemma

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

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

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

Balancing Bias and Variance

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

Lessons

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

Advanced Machine Learning

Over-Fitting

Overfitting, regularisation, feature selection

Outline

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

Over-fitting

- Complex machine can **over-fitting**

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

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

Binary Classification Task for You

Class 1 Class 2

Which Category?

- Which category does the following image belong to?

Spurious Rules

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

All Binary Functions

$x_0 = 000 \quad y_0 = \begin{cases} 0 \\ 1 \end{cases}$

$x_1 = 100 \quad y_1 = \begin{cases} 0 \\ 1 \end{cases}$ unseen

$x_2 = 010 \quad y_2 = \begin{cases} 0 \\ 1 \end{cases}$

$x_3 = 110 \quad y_3 = \begin{cases} 0 \\ 1 \end{cases}$

$x_4 = 001 \quad y_4 = \begin{cases} 0 \\ 1 \end{cases}$ seen

$x_5 = 101 \quad y_5 = \begin{cases} 0 \\ 1 \end{cases}$

$x_6 = 011 \quad y_6 = \begin{cases} 0 \\ 1 \end{cases}$

$x_7 = 111 \quad y_7 = \begin{cases} 0 \\ 1 \end{cases}$ unseen

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

Are MLPs Universal Approximators?

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

Outline

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

Controlling Complexity

- Infinitely flexible machine don't generalise (because any unseen data could have any value)
- **Machine learning only works because we believe 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
- 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
- The position of words in a sentence does not change its meaning (it might change the meaning of the sentence) and CNNs are often successful in understanding text

Preprocessing

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

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

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

Automatic Preprocessing

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

Outline

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

Hidden Structure

- Often the structure of data is invisible to us
- A very successful strategy is to try many different machine learning techniques and choose the best (stupid but effective)
- Often learning machines have adjustable parameters (hyper-parameters) that we have to set (they are the same for all input data, but change with the problem)
- We need to choose the hyper-parameters to fit the data in our problem
- Fine tuning 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 validation set to choose hyper-parameters
- You need a separate testing set if you want to measure you generalisation performance

Cross Validation

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

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

Training Set

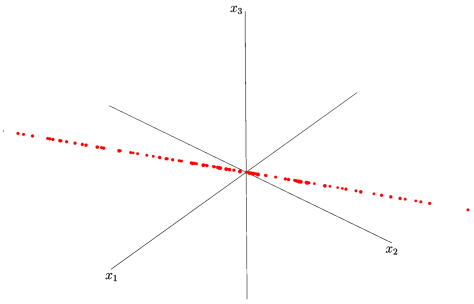
Leave-one-out cross validation

$$E_{\text{cv}} = \frac{5.5 + 3.5 + 1.8 + 4.8 + 3.7 + 3.6 + 7.4 + 4.6 + 0.99 + 4.5 + 4.6 + 5.4 + 6.2 + 3.3 + 2.7}{15} = 4.3$$

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

Hidden Structure

Can't spot low dimensional data by looking at numbers



Dimensionality Reduction

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

Feature Selection

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

Normalising Data

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

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

Outline

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

Explicit Regularisation

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

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

(Good to normalise data)

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

$$f(\boldsymbol{x}|\boldsymbol{w}) = \boldsymbol{w}^\mathsf{T}\boldsymbol{x}_n = \sum_{j=1}^p w_j\, x_j$$

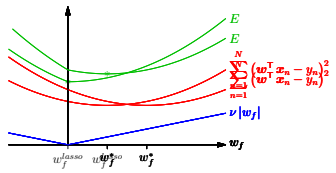
varies rapidly as we change x_i

Lasso

- We can us other regularisers

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

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



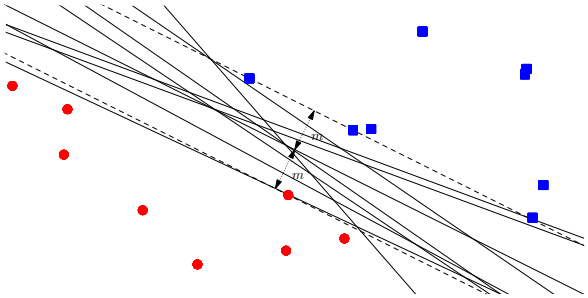
- Does automatic feature selection

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 chooses the machine with the biggest margins

Success of SVMs

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

Lessons

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

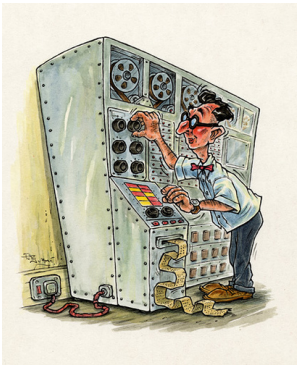
Advanced Machine Learning

Project

Details, Ideas, Research Methods

Outline

1. **Rules of the Game**
2. Ideas
3. How to Do a ML Project



Rules

- You will work in teams of 3-5
- Next week each team will email me a sheet with their team members and a team name
- You will give a brief presentation in week 2 (1 minute talk + 1 minutes of questions/suggestions)
- Write a project brief (1 page) for week 3
- In week 8 each team submits a 3 page report on data exploration
- In week 11 each team submits a 3 page report on using ML

Help

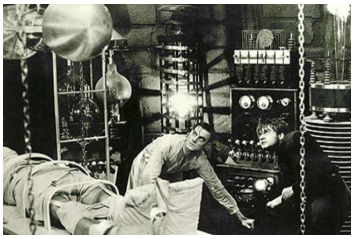
- There is a team of PhD students, Jon and myself to help
- We are only going to help in the Friday lecture slot
- We will give you ideas and steer you in the right direction
- We are available (nearly) every week
- To give you useful feedback I've split your report into two. First part on data exploration due before Easter

Marking

- The project is worth 40% (20% and 20%) (60 hours)
- The marks are assigned on the group reports
- If groups don't work I will split you up
- Each group can submit 1 page extra to indicate who should get the credit (signed by all group members)
- The exam is worth 60% and is the big differentiator
- You also get to give group presentations, just for the fun of it!

Outline

1. Rules of the Game
2. **Ideas**
3. How to Do a ML Project



Subject

- There is no restriction, but we award marks for work on "advanced machine learning"
- We take a very broad view of what that constitutes—its the whole process
- We are expecting you to go beyond taking a data set and applying a machine learning tool out of the box
- You can use any tools and any programming language you like (clearly acknowledging its use of course)

What Should You Do?

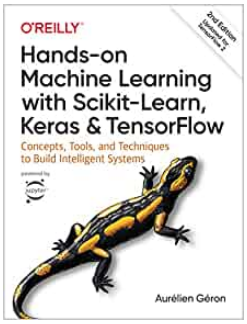
- Find a question, find a data set and make predictions
- There are lots of data sources available, datasets text/images on the web (decent please)
- Machine learning challenges
  - //pascallin2.ecs.soton.ac.uk/Challenges/
  - //home.web.cern.ch/about/updates/2014/05/higgs-boson-machine-learning-challenge
  - //www.kaggle.com/
- Beat IBM Watson at Jeopardy

Covering My Back

- Any projects involving
  - Humans
  - Personal data
  - Creating new intelligent life forms
- Must have ethical approval

Outline

1. Rules of the Game
2. Ideas
3. **How to Do a ML Project**



How to Do a Machine Learning Project?

- You are going to learn how to do a machine learning project by doing it
- I'm asking you not only to do the project, but to find out how to do a ML project
- The book "Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems" by Aurélien Géron takes you through this process
- In the Appendix there is a check list (also on a web page—see course page)

Eight Steps

1. Frame the problem and look at the big picture
2. Get the data
3. Explore the data to gain insights
4. Prepare the data to better expose the underlying data patterns to Machine Learning algorithms
5. Explore many different models and short-list the best ones
6. Fine-tune your models and combine them into a great solution
7. Present your solution
8. Launch, monitor, and maintain your system

1. Frame the Problem and Look at the Big Picture

1. Define the objective in business terms
2. How will your solution be used?
3. What are the current solutions/workarounds (if any)?
4. How should you frame this problem (supervised/unsupervised, online/offline, etc.)?
5. How should performance be measured?
6. Is the performance measure aligned with the business objective?
7. What would be the minimum performance needed to reach the business objective?
8. What are comparable problems? Can you reuse experience or tools?
9. Is human expertise available?

10. How would you solve the problem manually?
11. List the assumptions you (or others) have made so far
12. Verify assumptions if possible

2. Get the Data

- **Note: automate as much as possible so you can easily get fresh data**

1. List the data you need and how much you need
2. Find and document where you can get that data
3. Check how much space it will take
4. Check legal obligations, and get authorization if necessary
5. Get access authorizations
6. Create a workspace (with enough storage space)
7. Get the data
8. Convert the data to a format you can easily manipulate (without changing the data itself)

9. Ensure sensitive information is deleted or protected (e.g., anonymized)
10. Check the size and type of data (time series, sample, geographical, etc.)
11. Sample a test set, put it aside, and never look at it (no data snooping!)

3. Explore the Data

- **Note: try to get insights from a field expert for these steps.**

1. Create a copy of the data for exploration (sampling it down to a manageable size if necessary)
2. Create a Jupyter notebook to keep a record of your data exploration
3. Study each attribute and its characteristics:
   - Name
   - Type (categorical, int/float, bounded/unbounded, text, structured, etc)
   - Number of missing values
   - Noisiness and type of noise (stochastic, outliers, rounding errors, etc.)

   - Possibly useful for the task?
   - Type of distribution (Gaussian, uniform, logarithmic, etc)
4. For supervised learning tasks, identify the target attribute(s)
5. Visualise the data
6. Study the correlations between attributes
7. Study how you would solve the problem manually
8. Identify the promising transformations you may want to apply
9. Identify extra data that would be useful
10. Document what you have learned

4. Prepare the Data

- Notes:
  - Work on copies of the data (keep the original dataset intact)
  - Write functions for all data transformations you apply, for five reasons:
    - So you can easily prepare the data the next time you get a fresh dataset
    - So you can apply these transformations in future project
    - To clean and prepare the test set
    - To clean and prepare new data instances once your solution is live
    - To make it easy to treat your preparation choices as hyperparameters

4. Prepare the Data continued

1. Data cleaning
   - Fix or remove outliers (optional)
   - Fill in missing values (e.g., with zero, mean, median. . .) or drop their rows (or columns)
2. Feature selection (optional):
   - Drop the attributes that provide no useful information for the task
3. Feature engineering, where appropriate:
   - Discretize continuous features
   - Decompose features (e.g., categorical, date/time, etc.)
   - Add promising transformations of features (e.g., $\log(x)$, $\sqrt{x}$, x^2, e^x , etc.)

   - Aggregate features into promising new features
4. Feature scaling: standardise or normalise features

5. Short-List Promising Models

1. Train many quick and dirty models from different categories (e.g., linear, naiveBayes, SVM, Random Forests, neural net, etc.) using standard parameters
2. Measure and compare their performance
   - For each model, use N-fold cross-validation and compute the mean and standard deviation of the performance measure on the N folds
3. Analyse the most significant variables for each algorithm
4. Analyse the types of errors the models make
   - What data would a human have used to avoid these errors?
5. Have a quick round of feature selection and engineering
6. Have one or two more quick iterations of the five previous steps

7. Short-list the top three to five most promising models, preferring models that make different types of errors

6. Fine-Tune the System

- Notes:
  - You will want to use as much data as possible for this step, especially as you move toward the end of fine-tuning
  - As always automate what you can

1. Fine-tune the hyperparameters using cross-validation
   - Treat your data transformation choices as hyperparameters, especially when you are not sure about them (e.g., should I replace missing values with zero or with the median value? Or just drop the rows?)
   - Unless there are very few hyperparameter values to explore, prefer random search over grid search. If training is very long, you may prefer a Bayesian optimisation approach

2. Try Ensemble methods. Combining your best models will often perform better than running them individually

3. Once you are confident about your final model, measure its performance on the test set to estimate the generalization error

7. Present Your Solution

1. Document what you have done
2. Create a nice presentation
   - Make sure you highlight the big picture first
3. Explain why your solution achieves the business objective
4. Don't forget to present interesting points you noticed along the way
   - Describe what worked and what did not
   - List your assumptions and your system's limitations
5. Ensure your key findings are communicated through beautiful visualizations or easy-to-remember statements (e.g., "the median income is the number-one predictor of housing prices")

8. Launch

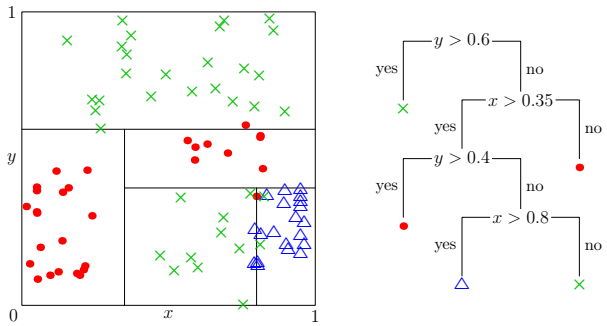
- Get your solution ready for production (plug into production data inputs, write unit tests, etc.)
- Write monitoring code to check your system's live performance at regular intervals and trigger alerts when it drops
  - ⋆ Beware of slow degradation too: models tend to "rot" as data evolves
  - ⋆ Measuring performance may require a human pipeline (e.g., via a crowd-sourcing service)
  - ⋆ Also monitor your inputs' quality (e.g., a malfunctioning sensor sending random values, or another team's output becoming stale). This is particularly important for online learning systems
- Retrain your models on a regular basis on fresh data (automate as much as possible)

In Conclusion

- Its a long list
- Not everything might be relevant
- But there is quite a lot of wisdom, so review the list regularly during the project
- You are doing this for you!
- For me, I only want two short reports—I've got to mark them

Advanced Machine Learning

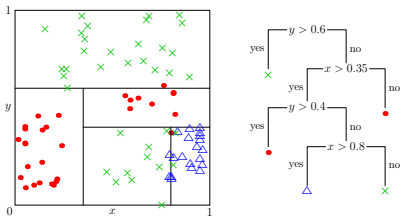
Ensemble Methods



Decision Trees, Bagging, Boosting

Outline

1. **Decision Trees**
2. Bagging
3. Boosting



Removing Variance By Averaging

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

Ensembling of Decision Trees

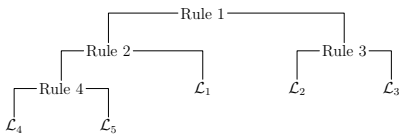
- One set of algorithms where ensembling are common place are decision trees
- These are particularly good for handling messy data
  - ⋆ categorical data
  - ⋆ mixture of data types
  - ⋆ missing data
  - ⋆ large data sets
  - ⋆ multiclass
- In many competitions ensembled trees, particularly *random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

Decision Trees

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

Partitioning

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



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

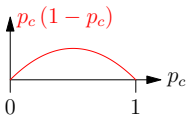
$$p_c(\mathcal{L}) = \frac{1}{|\mathcal{L}|} \sum_{(\boldsymbol{x},y)\in\mathcal{L}} [\![y = c]\!]$$

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

Leaf Purity

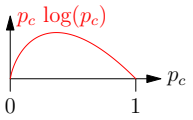
- Two different purity measures, $Q_m(\mathcal{L})$, for a leaf node $\mathcal{L}$ are commonly used
  - ⋆ **Gini index**

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

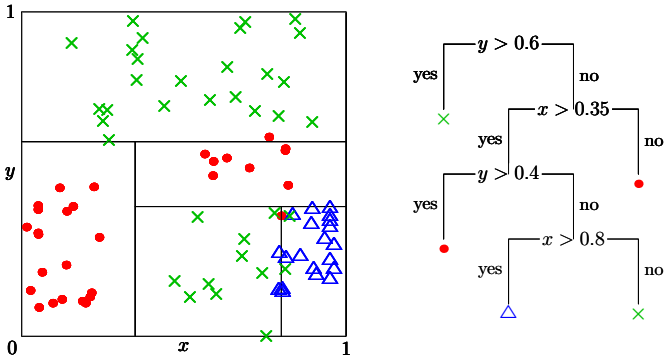


  - ⋆ **Cross-entropy**

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



Building Decision Trees

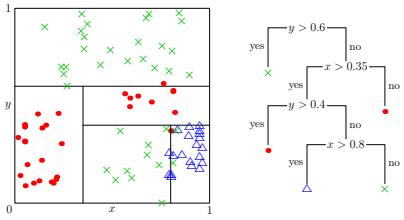


Observations

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

Outline

1. Decision Trees
2. **Bagging**
3. Boosting



Mean and Variance

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

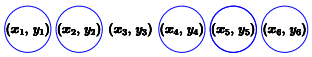
$$\mathbb{E}[\hat{\mu}_n] = \mathbb{E}\left[\frac{1}{n}\sum_{i=1}^{n} X_i\right] = \frac{1}{n}\sum_{i=1}^{n}\mathbb{E}[X_i] = \frac{1}{n}\sum_{i=1}^{n}\mu = \mu$$

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

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

Bootstrap Aggregation (Bagging)

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



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

Performance of Bagging

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

Variance of Positive Correlated Variables

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

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

$(\rho = \mathbb{E}[(X_i-\mu)(X_j-\mu)]/\sigma^2)$

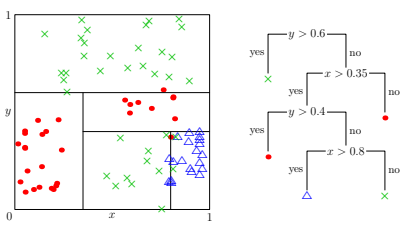
- As $n \to \infty$ the second term vanishes, but we are left with the first term
- If we want to do well we need our learning machines to be unbiased and decorrelated

Random Forest

- In random forests we average much less correlated trees
- We do this for each tree we choose a subset of $m \ll p$ of the features on which to split the tree
- Typically m 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

Outline

1. Decision Trees
2. Bagging
3. **Boosting**



Boosting

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

$$\hat{f}(\boldsymbol{x}) = \sum_{i=1}^{n} w_i \, \hat{h}_i(\boldsymbol{x})$$

- Weak learners, $\hat{h}_i(\boldsymbol{x})$, are learning machine that do a little better than chance
- The trick is to choose the weights, w_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 problems
  - ⋆ gradient boosting—used for regression, trains a classifier on the residual errors
- XGBoost is a very efficient library for gradient boosting on large data sets and often wins competition

Boosting a Binary Classifier

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

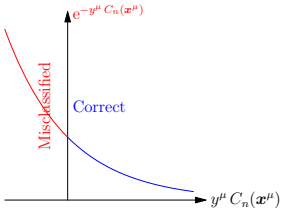
$$C_n(\boldsymbol{x}) = \alpha_1 \, \hat{h}_1(\boldsymbol{x}) + \alpha_2 \, \hat{h}_2(\boldsymbol{x}) + \cdots + \alpha_n \, \hat{h}_n(\boldsymbol{x})$$

- So that $\mathrm{sgn}\big(C_n(\boldsymbol{x})\big)$ is a strong learner?

AdaBoost

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

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



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

Iterative Learning

- We build up a strong learner iteratively (greedily)

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

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

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

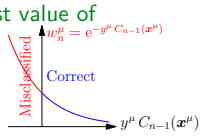
Choosing a Weak Classifier

- To minimise the error

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

- We choose the weak learner with the lowest value of

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



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

Choosing Weights

- We now choose the weight α_n to minimise the error $E_n(\alpha_n)$

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

- That is

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

Algorithm

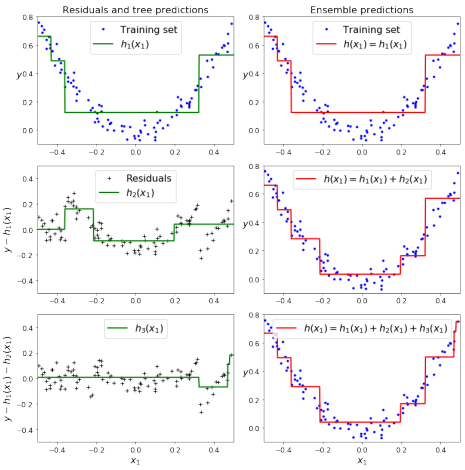
- Start with a set of weak learners $\mathcal{W}$
- Associate a weight, w_n^μ, with every data point $(\boldsymbol{x}^\mu, y^\mu)$, $\mu = 1, 2, \ldots, m$
- Initially $w_0^\mu = 1$ (large weight, w_n^μ, means $(\boldsymbol{x}^\mu, y^\mu)$ is poorly classified)
- Choose the weak learning, $\hat{h}_n(\boldsymbol{x}) \in \mathcal{W}$, that minimises $\displaystyle\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu$
- Update predictor $C_n(\boldsymbol{x}) = C_{m-1}(\boldsymbol{x}) + \alpha_n \, \hat{h}_n(\boldsymbol{x})$ where $\alpha_n = \frac{1}{2} \log \left(\frac{\sum_{\mu : y^\mu \neq \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu}{\sum_{\mu : y^\mu = \hat{h}_n(\boldsymbol{x}^\mu)} w_n^\mu} \right)$
- Update $w_n^\mu = w_{n-1}^\mu \, \mathrm{e}^{-y^\mu \, \alpha_n \, \hat{h}_n(\boldsymbol{x}^\mu)}$

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 work)
- It has fallen from fashion
- Unlike **gradient boosting**, although this is used for regression

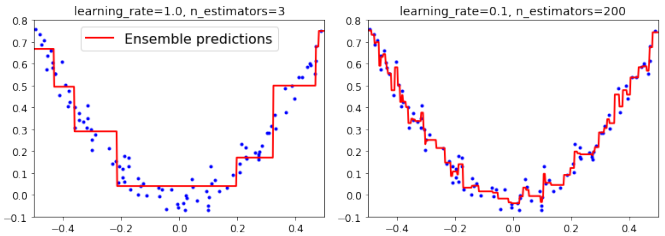
Gradient Boosting

- In gradient boosting we again build a strong learner as a linear combination of weak learners
- At each step we learn the residual error (i.e. the target minus the current prediction)
- (This difference looks a bit like a gradient hence the rather confusing name)
- We can use gradient boosting also to build regression trees



Keep On Going

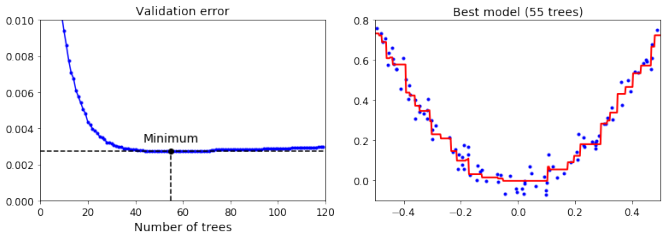
- We can keep on going



- But we will over-fit eventually

Early Stopping

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



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

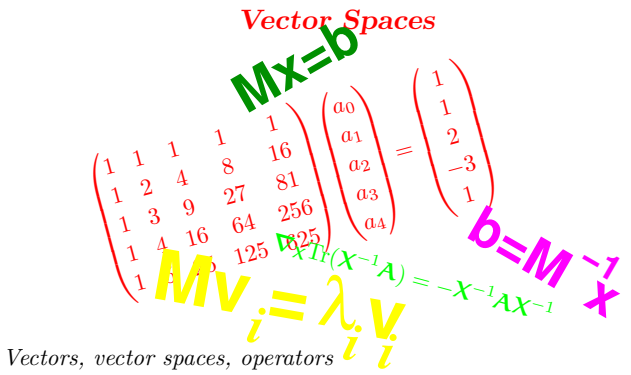
XGBoost

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

Conclusion

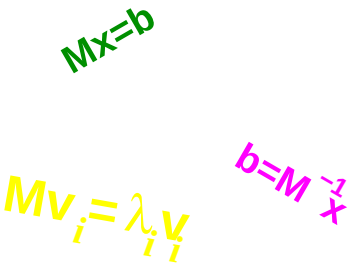
- Ensemble methods have proved themselves to be very powerful
- Tend to work best with very simple models (true of random forest and boosting)—seems to reduce over-fitting
- XGBoost is currently the best method for tabular data (particular for large training sets)—probably
- For images, signal and speech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

Advanced Machine Learning



Vectors, vector spaces, operators

Outline



Matrices, Vectors and All That

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

Vectors

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

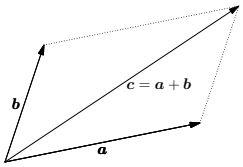
$$\boldsymbol{x} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}$$

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

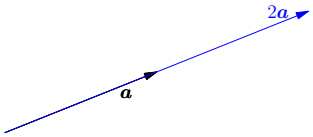
$$\boldsymbol{y}^\mathsf{T} = (y_1, y_2, \ldots, y_n)$$

Basic Vector Operations

- The basic vector operations are adding



- multiplying by a scalar (a number)



Vector Space

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

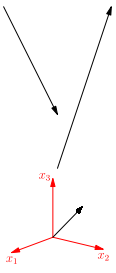
1. if $\boldsymbol{v}, \boldsymbol{w} \in \mathcal{V}$ then $a\,\boldsymbol{v} \in \mathcal{V}$ and $\boldsymbol{v} + \boldsymbol{w} \in \mathcal{V}$ (closure)
2. $\boldsymbol{v} + \boldsymbol{w} = \boldsymbol{w} + \boldsymbol{v}$ (commutativity of addition)
3. $(\boldsymbol{u} + \boldsymbol{v}) + \boldsymbol{w}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

(You don't need to remember these)

- Just from these properties we can deduce other properties

$\mathbb{R}^n$

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



Other Vector Spaces

- Vectors 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 $a\,f(x) \in C(a, b)$ and $f(x) + g(x) \in C(a, b)$
- Bounded vectors **don't** form a vector space

Metrics

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

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

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

Norms

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

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

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

Vector Norms

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

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

- Other norms exist, such as the p-norm

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

- Other special cases include the 1-norm and the infinite norm

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

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

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

Matrix Norms

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

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

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

$$\|\boldsymbol{A}\,\boldsymbol{B}\| \leq \|\boldsymbol{A}\| \times \|\boldsymbol{B}\|$$

Compatible Norms

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

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

(Frobenius 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 $\|\boldsymbol{M}\| \times \|\boldsymbol{M}^{-1}\|$

Function Norms

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

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

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

Normed Vector Spaces

- 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

Inner Products

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

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

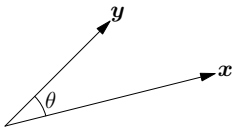
- For functions

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

- With matrices we don't usually think about inner products

Angles Between Vectors

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



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

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

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

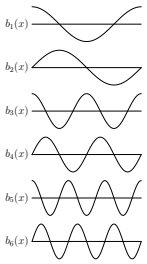
- Note that $\sin(x)$ and $\cos(x)$ are orthogonal in the interval $[0, 2\,\pi]$

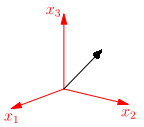
Basis Functions

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

Orthogonal Functions

- For functions we can use any ortho-normal set of functions as a basis
- The most familiar are the Fourier functions $\sin(n\,\theta)$ and $\cos(n\,\theta)$
- Any function in $C(0, 2\pi)$ can be represented by a point $\boldsymbol{f} = \begin{pmatrix} \langle f(x), b_1(x) \rangle \\ \langle f(x), b_2(x) \rangle \\ \vdots \end{pmatrix}$
- There might be an infinite number of components
- This is analogous to points in $\mathbb{R}^n$ (for large n)

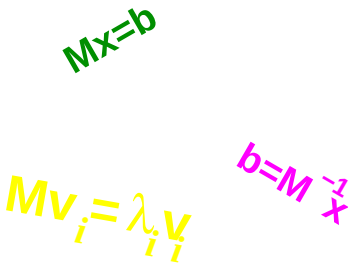




Algebraic Structure

- We have gone to these lengths as we want to show that many properties of vectors are shared by other objects (matrices, functions, etc.)
- Mathematicians study *algebraic structures* such as vector spaces, metric spaces, Hilbert spaces (infinite dimensional spaces with a norm and an inner product)
- Many of the concepts (distance/metric, norms, inner products) will re-occur many times within the course

Outline



Operators

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

Linear Operators

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

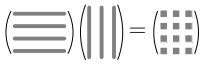
$$\mathcal{T}[\boldsymbol{x}] = \mathbf{M}\,\boldsymbol{x}$$

where $\mathbf{M}$ is a matrix

Matrix multiplication

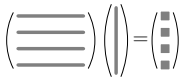
- For an $\ell \times m$ matrix $\mathbf{A}$ and an $m \times n$ matrix $\mathbf{B}$ we can compute the $(\ell \times n)$ product, $\mathbf{C} = \mathbf{A}\,\mathbf{B}$, such that

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



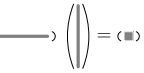
- Treating the vector $\boldsymbol{x}$ as a $n \times 1$ matrix then

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



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

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



Kernels

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

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

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

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

- We shall soon see examples of high-dimensional kernels

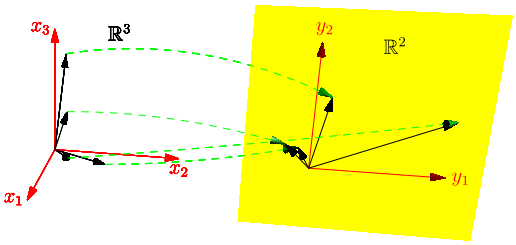
Kernels in Machine Learning

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

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

General Linear Mappings

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



Square Matrices

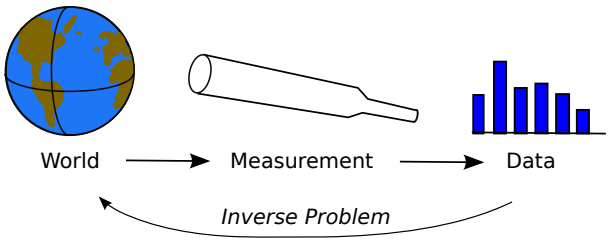
- We will spend a lot of time on operators that map from a vector space onto itself $\mathcal{T} : \mathcal{V} \to \mathcal{V}$
- For vectors in $\mathbb{R}^n$ such linear operators are represented by square matrices
- When there is a one-to-one mapping then we have a unique inverse
- We will study such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

Summary

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

Advanced Machine Learning

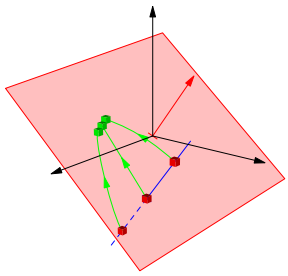
Understand Mappings



Mappings, Eigenvectors

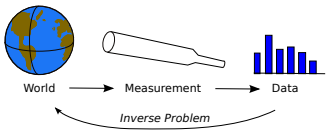
Outline

1. **Mappings**
2. Linear Maps
3. Eigenvectors



Transforming Data

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



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

Inverse Problems

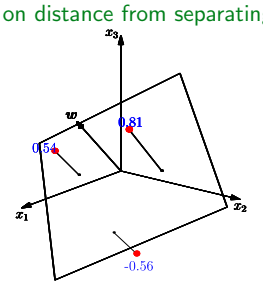
- Given P observations $\{(\boldsymbol{x}_k, y_k)|k = 1, \ldots, m\}$ and n unknown $\boldsymbol{w} = (w_1, w_2, \ldots w_n)$ such that $\boldsymbol{x}_k^\mathsf{T}\boldsymbol{w} = y_k$ then to find $\boldsymbol{w}$
- Define the *design matrix* as the matrix of feature vectors

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

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

Linear Regression

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



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

Linear Least Squares

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

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

- The squared error

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

- We can define the error vector

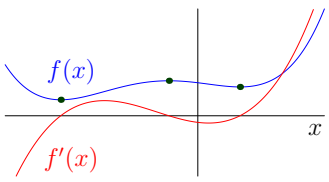
$$\boldsymbol{\epsilon} = \mathbf{X}\boldsymbol{w} - \boldsymbol{y}$$

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

- Minimising this error is known as the least squares problem

Finding a Minimum

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



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

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

Gradients

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

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

- The partial derivatives (curly d's)

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

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

Least Squares Solution

- The least squared solution is give by

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

- Or

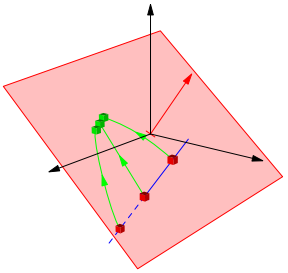
$$\boldsymbol{w} = \left(\mathbf{X}^\mathsf{T}\mathbf{X}\right)^{-1}\mathbf{X}^\mathsf{T}\boldsymbol{y} = \mathbf{X}^{+}\boldsymbol{y}$$

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

```
w = X\y
```


Outline

1. Mappings
2. **Linear Maps**
3. Eigenvectors

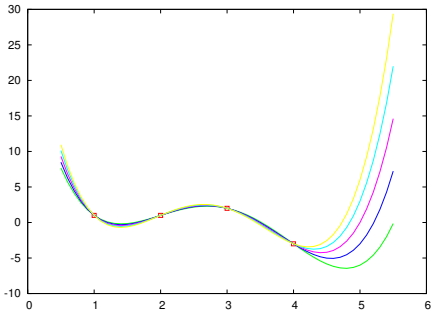


Solving Inverse Problems

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

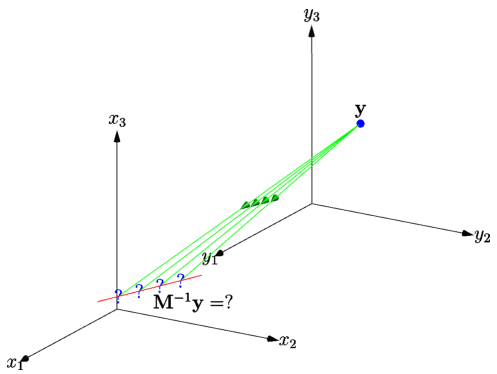
Under Constrained Systems

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



What is the Inverse?

- Many points can map to the same points



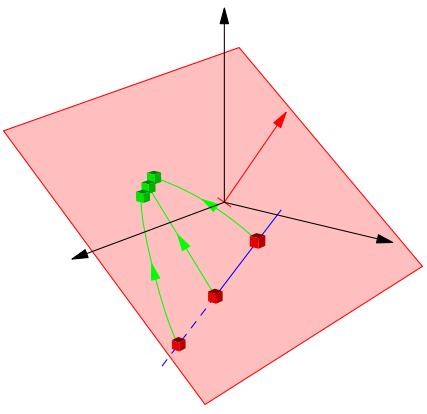
Under-constrained Systems

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

Ill-Conditions

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

Ill-Conditioned Matrices

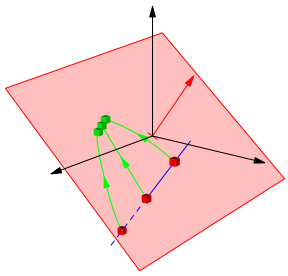


Ill-Conditioning in ML

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

Outline

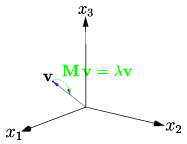
1. Mappings
2. Linear Maps
3. **Eigenvectors**



Eigenvalue equation

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

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



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

Symmetric Matrices

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

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

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

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

Proof of Orthogonality

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

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

- So either $\lambda_i = \lambda_j$ or $\boldsymbol{v}_i^\mathsf{T}\boldsymbol{v}_j = 0$
- If $\lambda_i = \lambda_j$ then any vector in the plane spanned by $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ are eigenvectors and we can always choose two orthogonal vectors in the plane

Orthogonal Matrices

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

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

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

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

The Other Way Around

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

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

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

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

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

Invertible Matrices

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

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

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

$$\begin{aligned}\mathbf{V}^\mathsf{T}\mathbf{V}\,\boldsymbol{x} &= \mathbf{V}\,\mathbf{0}\\ \boldsymbol{x} &= \mathbf{0}\end{aligned}$$

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

- Thus $\boldsymbol{V}$ is invertible

Rotations

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

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

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

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

- Rotations preserve lengths and angles

Matrix Decomposition

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

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

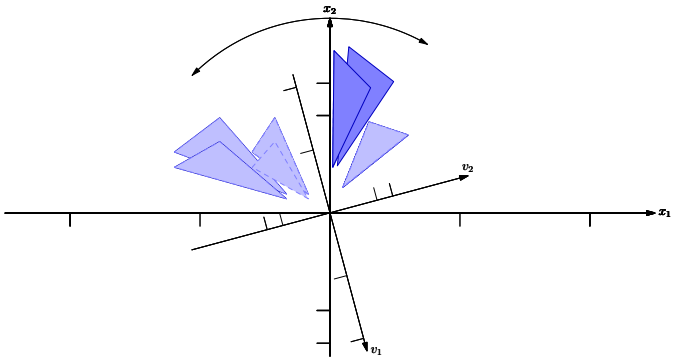
- where $\boldsymbol{\Lambda} = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n) = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{pmatrix}$
- Now

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

- Very important *similarity transform*

Mappings by Symmetric Matrices

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



Inverses

- For any square matrix

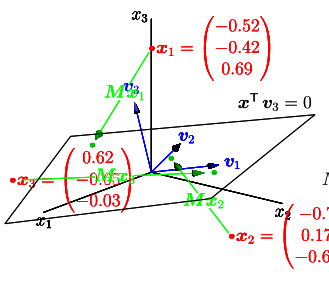
$$\boldsymbol{M} = \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^\mathsf{T} \qquad \boldsymbol{M}^{-1} = \boldsymbol{V}\boldsymbol{\Lambda}^{-1}\boldsymbol{V}^\mathsf{T}$$

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

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

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

Ill-Conditioning Again



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

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

Condition Number

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

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

- Large condition number implies very ill-conditioned

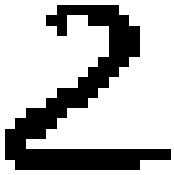
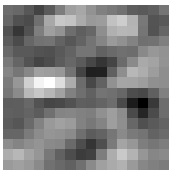
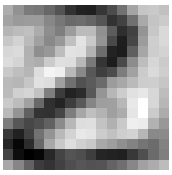
Summary

- Linear mappings are commonly used in machine learning algorithms such as regression
- We will often meet the pseudo-inverse involving inverting $\mathbf{X}^\mathsf{T}\mathbf{X}$
- They can be inherently unstable to noise in the inputs
- We can understand symmetric operators by looking at their eigenvectors
- Function spaces can similarly be understood in terms of eigenfunctions

Advanced Machine Learning

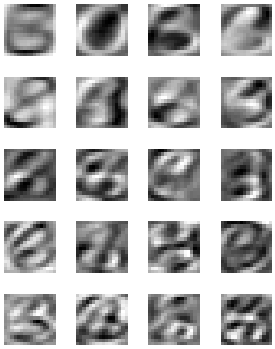
Principal Component Analysis (PCA)

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

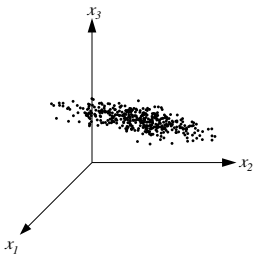
Covariance matrices, dimensionality reduction, PCA, Duality

Outline



Spread of Data

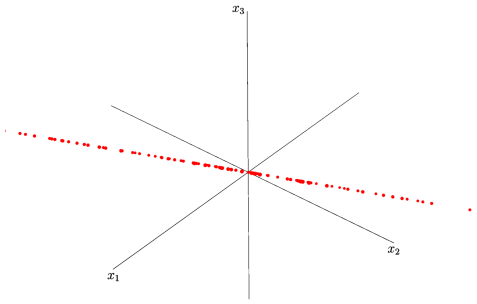
- Often data varies significantly in only some directions



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

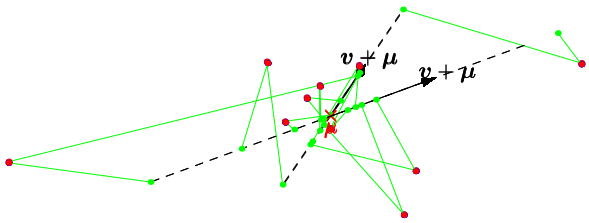
Looking is not Enough

Can't spot low dimensional data by looking at numbers



Dimensionality Reduction

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



Direction of Maximum Variation

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

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

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

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

- λ is a Lagrange multiplier

Direction of Maximum Variation

- Expanding the Lagrangian

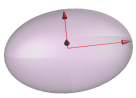
$$\begin{aligned}\mathcal{L} &= \frac{1}{m-1}\sum_{k=1}^{m}\left(\boldsymbol{v}^\mathsf{T}(\boldsymbol{x}_k - \boldsymbol{\mu})\right)^2 - \lambda\left(\|\boldsymbol{v}\|^2 - 1\right)\\ &= \frac{1}{m-1}\sum_{k=1}^{m}\left(\boldsymbol{v}^\mathsf{T}(\boldsymbol{x}_k - \boldsymbol{\mu})(\boldsymbol{x}_k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

- Extrema of the Lagrangian

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

Direction of Maximum Variation

- The eigenvectors are directions that are extrema of the variance



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

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

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

Covariance Matrix

- The **covariance matrix** is defined as

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

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

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

- C.f. covariance of random variables

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

Outer Product

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

$$\boldsymbol{x}\,\boldsymbol{y}^{\mathsf{T}} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix}\begin{pmatrix} y_1 & y_2 & \cdots & y_n \end{pmatrix} = \begin{pmatrix} x_1\,y_1 & x_1\,y_2 & \cdots & x_1\,y_n \\ x_2\,y_1 & x_2\,y_2 & \cdots & x_2\,y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_n\,y_1 & x_n\,y_2 & \cdots & x_n\,y_n \end{pmatrix}$$

- C.f. Inner product

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

Matrix Form

- The covariance matrix is

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

- Define the matrix

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

- We can write the covariance matrix as

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

Properties of Covariance Matrix

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

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

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

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

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

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

Eigenvalue Decomposition

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

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

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

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

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

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

Surface Defined by Matrix

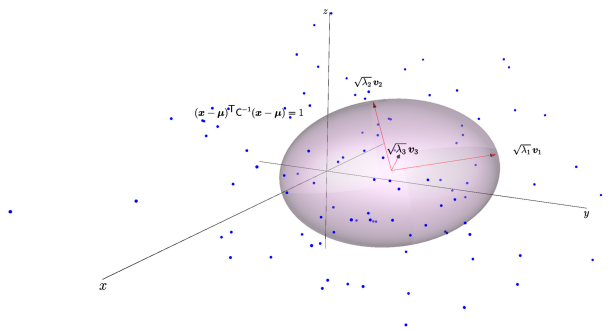
- The vectors $\boldsymbol{x}$ such that

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

defines a surface

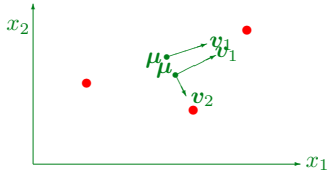
- The surface is an ellipsoid
- The eigenvectors point in the direction of the principal axes of the ellipsoid
- The radii of the principal axes are equal to the square root of the eigenvalues

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 $m+1$ patterns that are linearly independent (i.e. you cannot form any one out of m of the other patterns) will have no zero eigenvalues

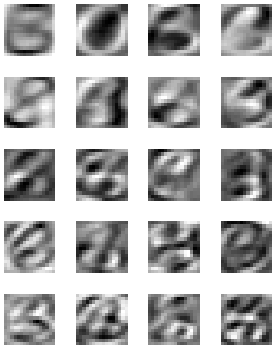


Positive Definite

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

Outline

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



Principal Component Analysis

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

Projection Matrix

- To project the inputs construct the projection matrix

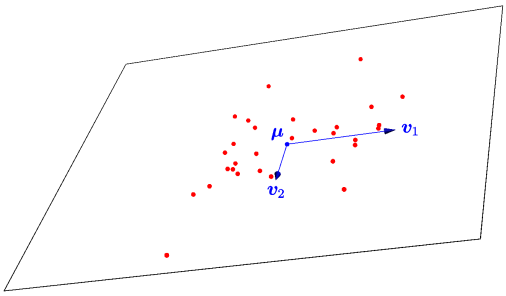
$$\mathbf{P} = \begin{pmatrix} \boldsymbol{v}_1^\mathsf{T} \\ \boldsymbol{v}_2^\mathsf{T} \\ \vdots \\ \boldsymbol{v}_k^\mathsf{T} \end{pmatrix}$$

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

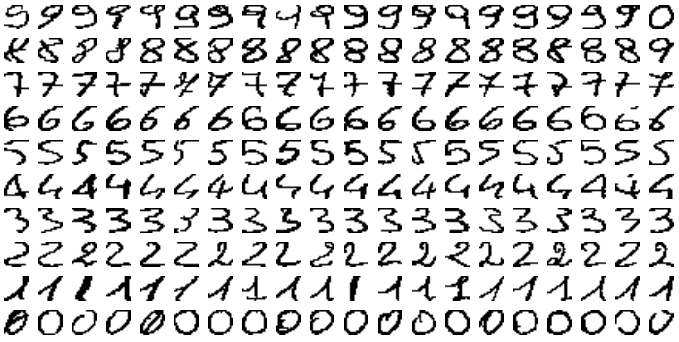
$$\boldsymbol{z} = \mathbf{P}\,(\boldsymbol{x} - \boldsymbol{\mu})$$

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

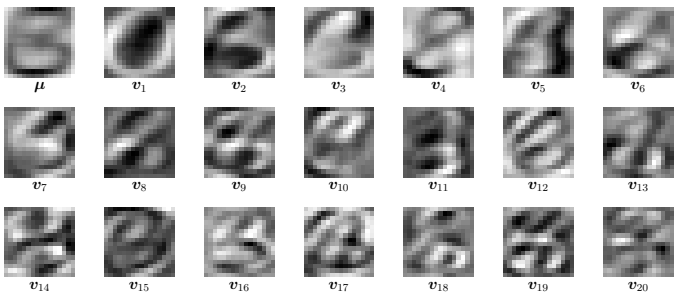
Subspace Projection



Hand Written Digits



Eigenvectors



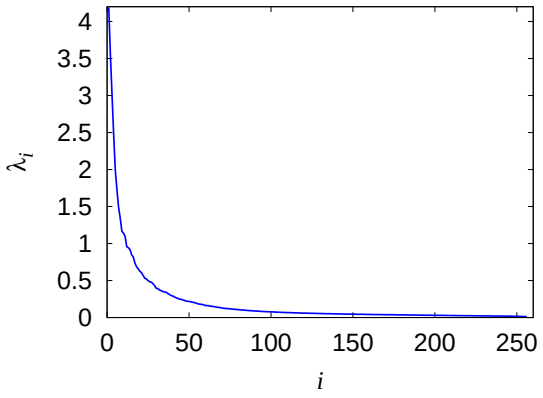
Reconstruction

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

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

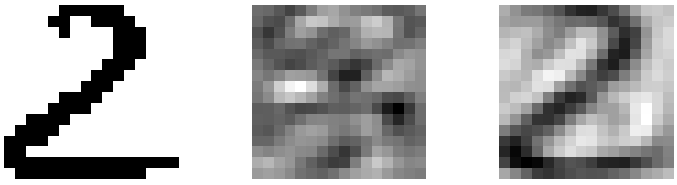
- Principle component analysis projects the data into a subspace of size m with the minimal approximation error $\mathbb{E}\left[\|\hat{\boldsymbol{x}}_i - \boldsymbol{x}_i\|^2\right]$
- The loss of "energy" is equal to the sum of the eigenvalues in the directions that are ignored

Eigenvalues for Digits



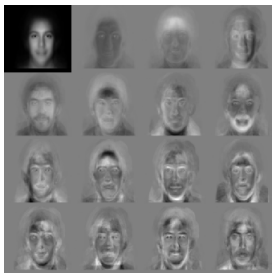
Reconstruction from Eigenvectors

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



Outline

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



PCA for Images

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

Dual Matrix

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

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

- $\boldsymbol{u} = \mathbf{X}\,\boldsymbol{v}$

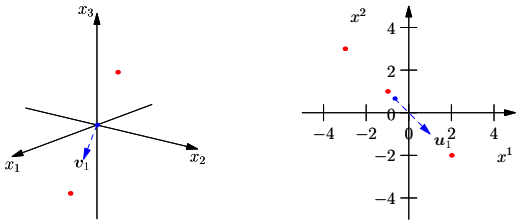
Dual Matrix

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

What Does a Subspace Look Like?

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

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

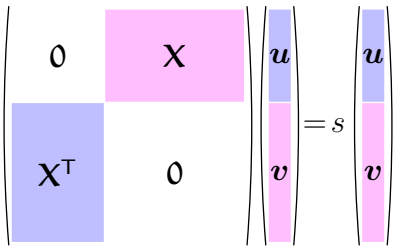


Summary

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

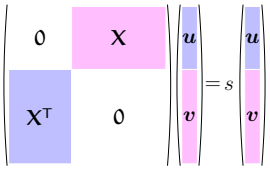
Advanced Machine Learning

Singular Value Decomposition (SVD)



Singular Valued Decomposition, SVD, general linear maps

Outline



Recap

- Given a set of data $\mathcal{D} = \{\boldsymbol{x}_i\}$ we can construct the matrix

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

- where $\boldsymbol{\mu} = \frac{1}{m}\sum_{\mu+1}^{m} \boldsymbol{x}_\mu$
- We can construct the covariance matrix

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

- This describes the variation of the data in feature space

Eigenvectors of Covariance

- The eigenvectors, $\boldsymbol{v}_i$, describe the direction of maximum variance
- The eigenvalues, λ_i, is equal to the variance of the data in the directions $\boldsymbol{v}_i$
- Thus directions $\boldsymbol{v}_i$ with large λ_i are directions where there is a lot of variation
- Directions $\boldsymbol{v}_i$ with small λ_i are directions with very small variation
- Usually these directions with small λ_i are dominated by noise and we can throw them out

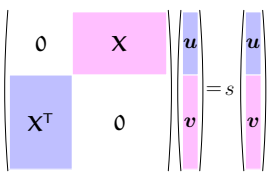
Zero Eigenvalues

- If $\lambda_i = 0$ this mean there is no fluctuations in this direction
- This will inevitably happen when we have m examples in an p dimensional feature space and $m \leq p$
- In some problems $m \ll p$ and most of the eigenvalues are zero
- Typically the data points will only span an $(m-1)$-dimensional space

Dual Matrix

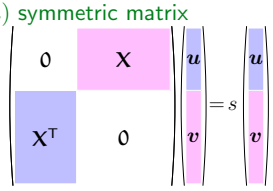
- We can define a dual matrix to $\mathbf{C} = \mathbf{X}\mathbf{X}^\mathsf{T}$ namely $\mathbf{D} = \mathbf{X}^\mathsf{T}\mathbf{X}$
- This will have the same non-zero eigenvalues at $\mathbf{C}$
- If $\boldsymbol{u}_i$ is an eigenvector or $\mathbf{D}$ then $\boldsymbol{v}_i = \mathbf{X}\boldsymbol{u}_i$ will be an eigenvector of $\mathbf{C}$
- When $m \ll p$ it is more useful to working in this dual space spanned by the data rather than the feature space

Outline



Singular Valued Decomposition

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



s is an eigenvalue

- We observe that

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

Eigenvectors

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

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

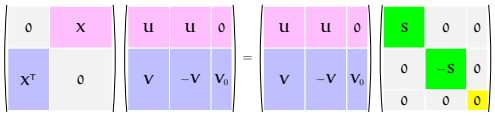
  so $\begin{pmatrix}\boldsymbol{u}\\-\boldsymbol{v}\end{pmatrix}$ is an eigenvector of the full matrix with eigenvalue $-s$

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

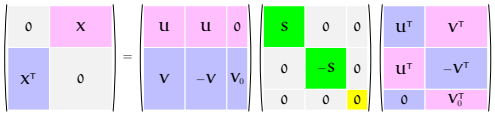
$$n \times \left(s_i, \begin{pmatrix}\boldsymbol{u}_i\\\boldsymbol{v}_i\end{pmatrix}\right) \quad n \times \left(-s_i, \begin{pmatrix}\boldsymbol{u}_i\\-\boldsymbol{v}_i\end{pmatrix}\right) \quad m - n \times \left(0, \begin{pmatrix}0\\\boldsymbol{v}_k\end{pmatrix}\right)$$

Matrix Decomposition

- Stacking the eigenvectors into a matrix



- Multiply on the left by the transpose of the orthogonal matrix



- An important consequence of this

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

SVD

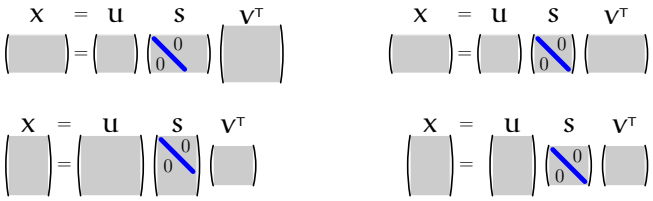
- Any matrix, $\mathbf{X}$, can be written as $\mathbf{X} = \mathbf{U}\,\mathbf{S}\,\mathbf{V}^\mathsf{T}$
  - $\mathbf{U}$, $\mathbf{V}$ are orthogonal matrices
  - $\mathbf{S} = \mathrm{diag}(s_1, s_2, \ldots, s_n)$
- s_i can always be chosen to be positive and are known as **singular values**
- To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

Finding SVD

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

Economical Forms of SVD

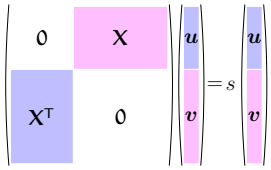
- Often the rows or columns of the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ that are not associated with a singular value are ignored



- In Matlab these are obtained using

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


Outline



General Matrix

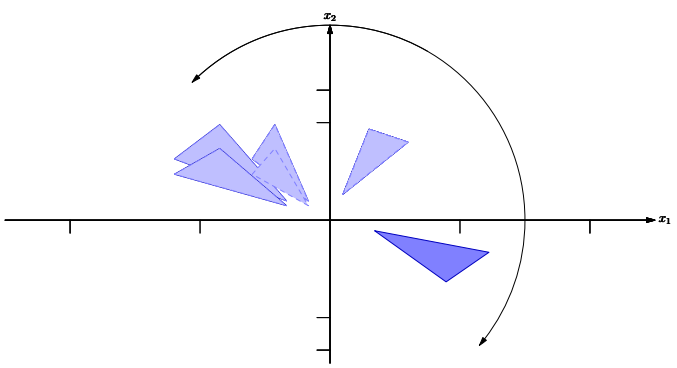
- Recall that we can compute the SVD for any matrix, $\mathbf{X}$
- As matrices describe the most general linear mapping

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

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

Matrices

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



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

Duality Revisited

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

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

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

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

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

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

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

Having A Go

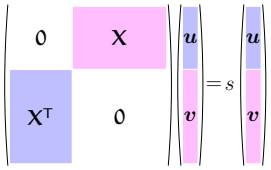
- It's really easy to verify this in MATLAB or OCTAVE

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

- Test yourself!

Outline

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



Linear Regression

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

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

- Which we fit by minimising the squared error

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

Matrix Form

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

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

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

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

- This is known as the pseudo-inverse

Using SVD

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

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

- If $m > p$

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

Pseudo-Inverse of S

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

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

Ill-Conditioned Data Matrix

- Recall that

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

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

Regularisation

- Consider linear regression with a regulariser

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

- Thus

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

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

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

Regularisation Continued

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

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

- where

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

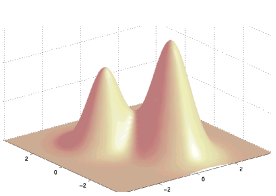
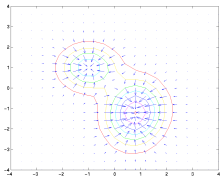
Effect of Regularisation

- If $s_i = 0$ the problem would be ill-posed (even S^+ does not exist since s_i^{-1} would be ill defined)
- Using a regularisation term if $s_i = 0$ then $s_i/(s_i^2 + \eta) = 0$ and the "inverse" is defined
- If $s_i \ll \eta$ then $s_i/(s_i^2 + \eta)$ is small and we are no longer ill-conditioned
- If $s_i \gg \eta$ then $s_i/(s_i^2 + \eta) \approx s_i^{-1}$
- Regularisation makes the machine much more stable (reduces the variance)

Summary

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

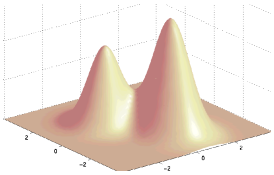
Advanced Machine Learning
Optimisation

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

Gradient descent, quadratic minima, differing length scales

Outline

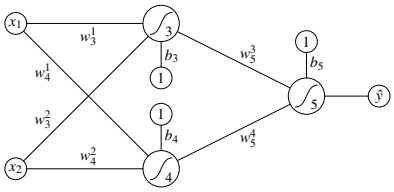


ML = Optimisation

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

MLP

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



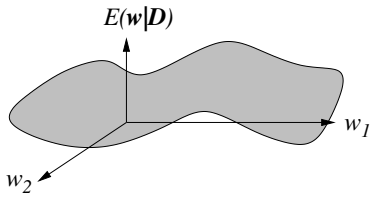
- Stands for the function ($\hat{y} = f(\boldsymbol{x}|\boldsymbol{w})$)
$$\hat{y} = g\big(w_5^3\, g(w_3^1\, x_1 + w_3^2\, x_2 + b_3) + w_5^4\, g(w_4^1\, x_1 + w_4^2\, x_2 + b_4) + b_5\big)$$
where, for example, $g(V) = \frac{1}{1+\mathrm{e}^{-V}}$

Training

- Given a (labelled) training dataset
$$\mathcal{D} = \big\{(\boldsymbol{x}_k, y_k)\big|k = 1, \ldots, P\big\}$$
- We choose our parameters $\boldsymbol{w}$ to minimise the learning error
$$E(\boldsymbol{w}|\boldsymbol{\mathcal{D}}) = \frac{1}{P}\sum_{k=1}^{P}\big(f(\boldsymbol{x}_k|\boldsymbol{w}) - y_k\big)^2$$
- We then use the machine with the weights $\boldsymbol{w}^*$ which minimise $E(\boldsymbol{w}|\mathcal{D})$

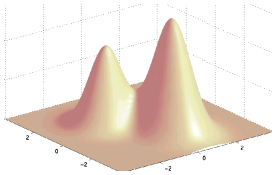
Computing Gradients

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



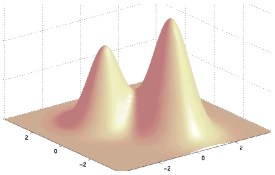
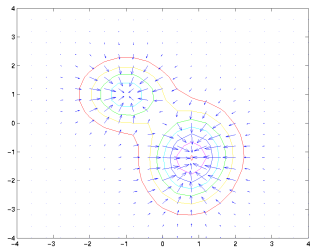
- To minimise we $E(\boldsymbol{w}|\mathcal{D})$ we compute the gradient $\boldsymbol{\nabla} E(\boldsymbol{w}|\mathcal{D})$
- In MLP an efficient algorithm for computing the gradient is known as back-prop

Outline



Gradient Optimisation

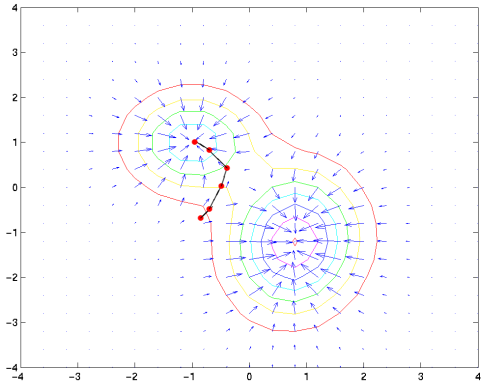
- A maximum or minimum occurs when $\boldsymbol{\nabla} E(\boldsymbol{w}|\mathcal{D}) = \boldsymbol{0}$
- E.g.
$$z = \mathrm{e}^{-(x+1)^2-(y-1)^2} + 0.6\,\mathrm{e}^{-(x-1)^2-0.5(y+1)^2+0.1(x-3)(y-3)}$$

Gradient Descent

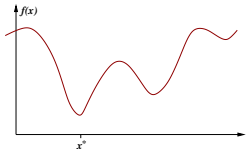
- For a simple function $E(\boldsymbol{w}|\mathcal{D})$ we can solve $\boldsymbol{\nabla} E(\boldsymbol{w}|\mathcal{D}) = \mathbf{0}$ explicitly. E.g. the linear perceptron
- For a non-linear functions we usually can't solve this set of simultaneous equations
- We can find a maximum or minimum **iteratively**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

Hill-Climbing



What Goes Right

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



- Taylor expanding around a minimum x^*

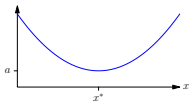
$$\begin{aligned} f(x) &= f(x^*) + (x-x^*)\, f'(x^*) + \frac{1}{2}(x-x^*)^2\, f''(x^*) + \cdots \\ &= f(x^*) + \frac{1}{2}(x-x^*)^2\, f''(x^*) + \frac{1}{3!}(x-x^*)^3\, f'''(x^*) + \cdots \end{aligned}$$

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

Newton's Method

- If we were in a quadratic minimum

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



- then

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

- so

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

- or

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

Newton's Method

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

Taylor's Expansion in High Dimensions

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

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

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

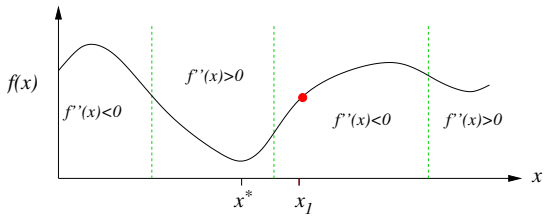
$$H_{ij} = \frac{\partial^2 f(\boldsymbol{x}_0)}{\partial x_i\, \partial x_j}$$

- Newton's method in high dimension is

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

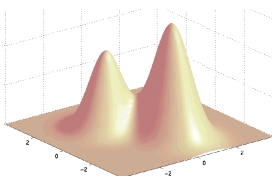
Using the Second Derivative

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



Outline

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



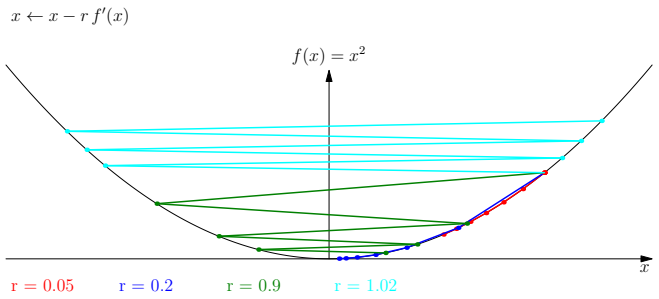
Step Size

- Gradient descent

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

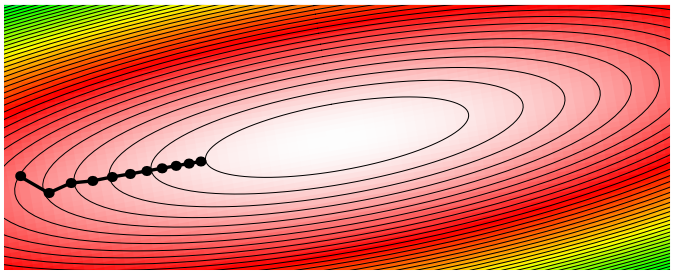
- Need to choose the learning rate of step size, r
- Too small steps takes lots of time
- Too large steps takes you away from minima

Step Size



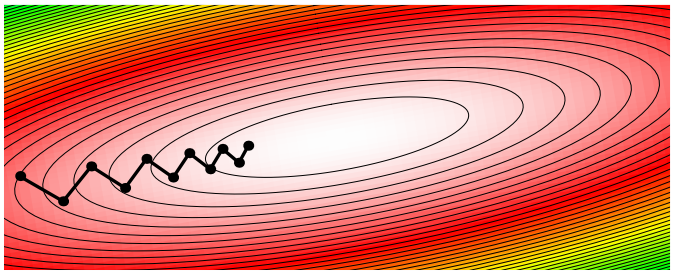
Higher Dimensions

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



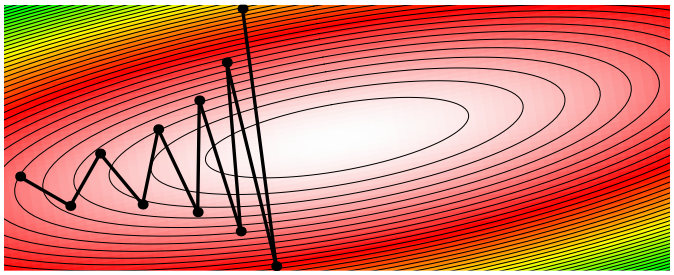
Getting There Quicker

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



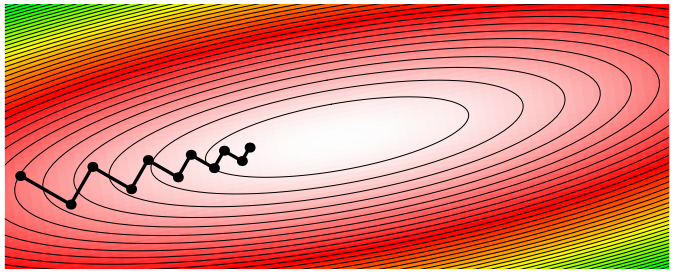
More Haste Less Speed

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



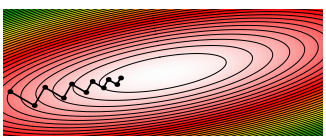
Line Minimisation

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



Zig-Zag

- Note that in high dimensions gradient descent tends to zigzag



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

Best Optimisation Algorithms

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

Levenberg-Marquardt

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

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

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

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

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

Trust Region

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

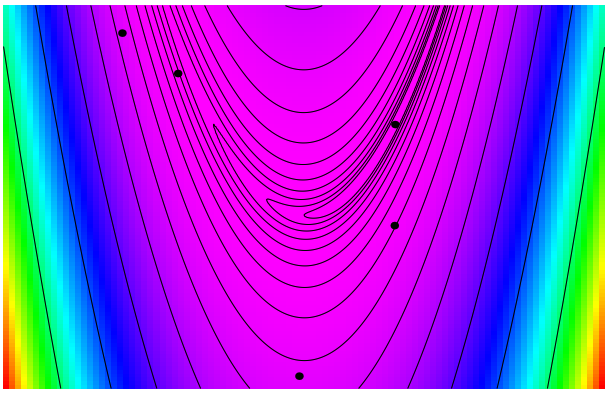
$$\boldsymbol{w}^{(k+1)} = \boldsymbol{w}^{(k)} - \left(\mathbf{J}^{\mathsf{T}}\mathbf{J}\right)^{-1}\mathbf{J}^{\mathsf{T}}\boldsymbol{\epsilon}(\boldsymbol{w})$$

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

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

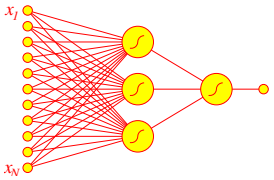
- ν limits the step size
- If predicted reduction in error is accurate reduce ν, if predicted reduction in error is very poor increase ν

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



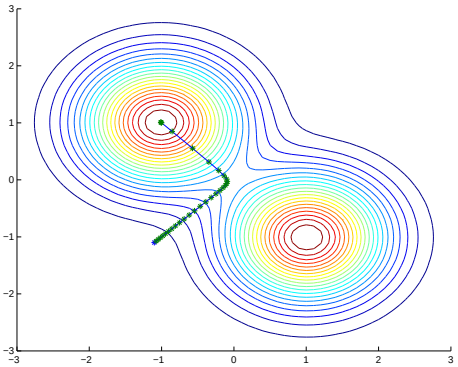
Landscape Problems

- For sigmoid response function such as the logistic or tanh function there are very flat regions away from the bias
- Very little gradient information (can use momentum)
- Many local minima—not guaranteed to find global minimum
- Weight space is symmetric because of permutation symmetry



Saddle-Points

- Many saddle-points due to permutation symmetry of hidden nodes



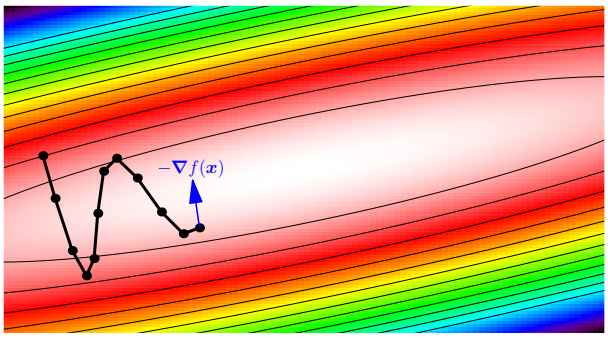
Modern Machine Learning

- Modern deep learning machines usually have so many parameters that it would be impractical to even estimate the Hessian
- We now commonly use ReLU's rather than sigmoids so the cost landscape is non-analytic
- Making assumptions about the minima being quadratic is no longer true
- In deep learning it is usually important to ensure that loss is differentiable almost everywhere otherwise we can't use gradient descent

Summary

- Minimising the error gives a general purpose learning algorithm
  - ⋆ E.g. Logistic Perceptron
  - ⋆ E.g. Multi-Layer Perceptron
- The minimisation is in a high dimensional space
- Minimisation can be very time consuming
- Nasty things can happen, but there are many standard algorithms which work well
- In complex models there may be many minima—you are not guaranteed to find the best one

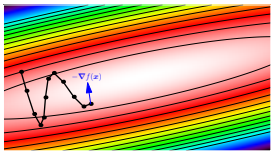
Advanced Machine Learning
Stochastic Gradient Descent



SGD, momentum, step size, ADAM

Outline

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



The Next Step

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

Mini-Batch Learning

- Rather than computing the gradient of the error, e.g.

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

- We often compute an approximation

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

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

Stochastic Gradient Descent

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

Automatic Differentiation

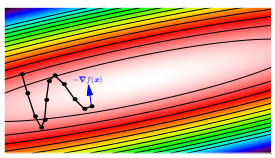
- For gradient descent we need to compute the gradient
- For most of my life this was just a pain
- You guys have it easy, modern deep learning frameworks do this for you automatically
- This has changed the game! We are no longer afraid of large models
- We can differentiate through algorithms allowing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

Convergence

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

Outline

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

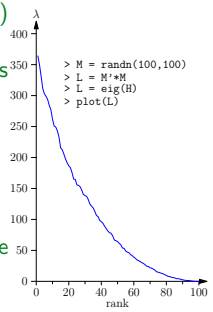


Step Size

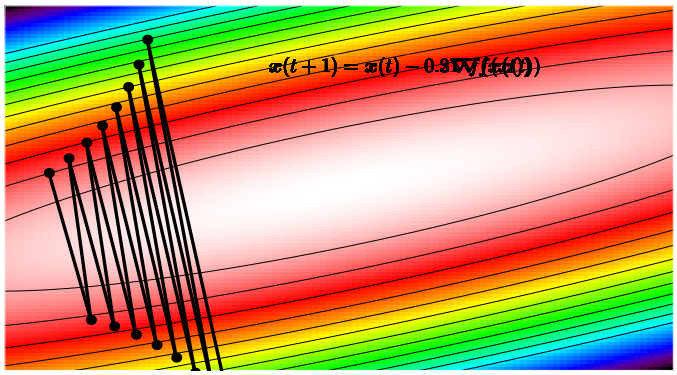
- The optimal step size depends on the gradient and the curvature (second derivative)
- For a quadratic minima the minimum is given by

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

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



Gradient Descent



Avoiding Divergence

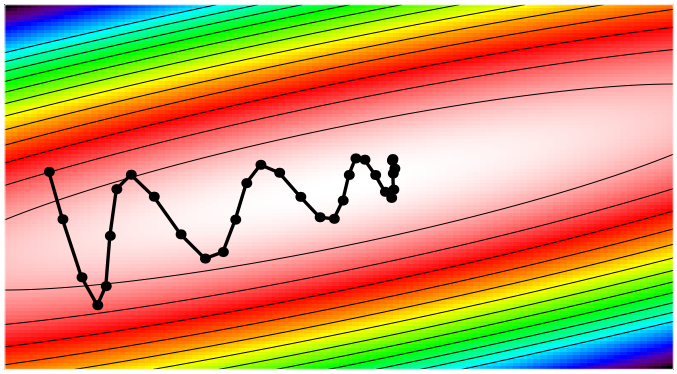
- We saw in the last lecture that gradient descent can actually diverge from a local optimum if you have a too big step size
- If we use too large a step size we quickly get NaN errors
- If we only use SGD the step size is determined by the smallest eigenvalue of the Hessian
- This seems impossibly slow,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

Momentum

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

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

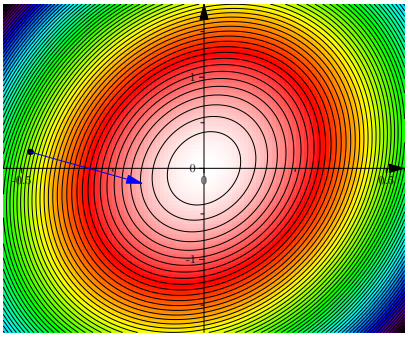
Gradient Descent with Momentum



Adaptive Methods

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

Rescaling Co-ordinates



AdaDelta

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

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

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

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

AdaDelta

- The AdaDelta algorithm uses the update

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

- Note that if we can rescale the weights or rescale the gradients and the update would be the same
- Because we are adaptively changing our step size we can use a single step size throughout the optimisation

Adding in Momentum

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

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

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

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

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

ADAM

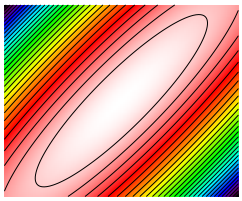
- Adaptive Moment Estimation (Adam) adapts the scale of the parameters, but also uses momentum
- Update weights

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

- ADAM and its variants are very successful in deep learning

Covariance

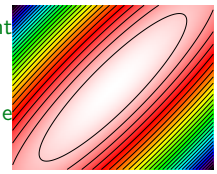
- Vector arithmetic, (matrix multiplication, addition, multiplying by a scale) has a covariance property that it is invariant to the coordinate system
- That is we can translate and rotate our coordinates and get the same result
- That is not true of element-wise multiplication of vectors
- Many ML algorithms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



Can't rescale coordinates

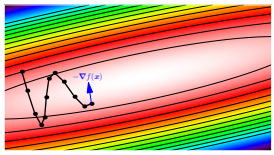
Correlated Weights

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



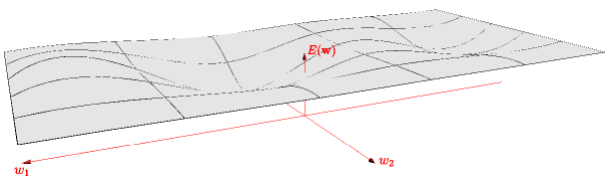
Outline

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



Loss Landscape

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



Global and Local Minima

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

Noisy Optimisation

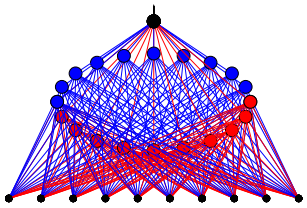
- Because we are using mini-batches we aren't actually following the correct gradients
- If we reduce our step size then our average gradient is closer to the true gradient
- Thus our optimisation is inherently noisy
- At least for deep learning there is no evidence that the weights stop changing even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

Valleys in Valleys in Valleys

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

Symmetries

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



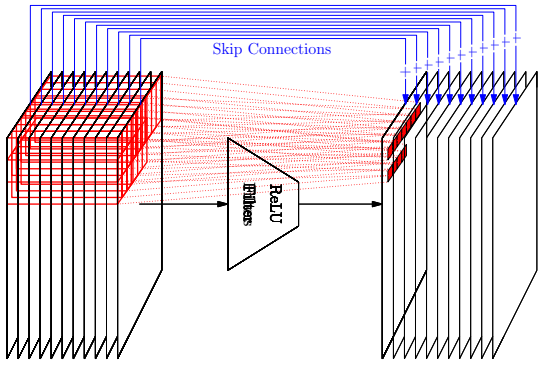
More Symmetries

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

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

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

Skip Connections



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

Neutral Network

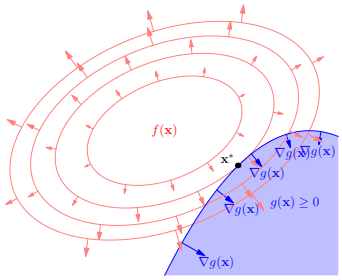
- The landscape potentially has a huge manifold with the same losses (neutral network)
- We should think of optimisation more as a process of travelling than a process of arriving
- Of course even if we do minimise our loss we are not guaranteed to minimise our generalisation performance
- On the other hand, by using a lot of regularisation we can reduce the tendency to overfit the training set

Lessons

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

Advanced Machine Learning

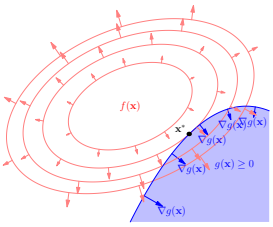
Constrained Optimisation



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

Outline

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

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

- A standard procedure is to define the Lagrangian

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

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

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

Conditions on Optimum

- The optimisation problem is

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

- Assuming differentiability

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

- The second condition is just the constraint

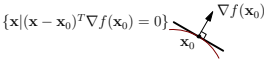
Note on Gradients

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

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

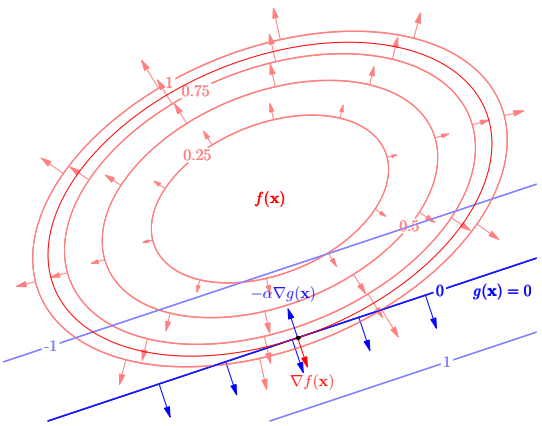
  where $\mathbf{H}$ is a matrix of second derivative known as the Hessian
- If we consider the set of points perpendicular to $\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}_0)$ which go through $\boldsymbol{x}_0$ (the tangent plane), these will have values

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



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

Constrained Optima



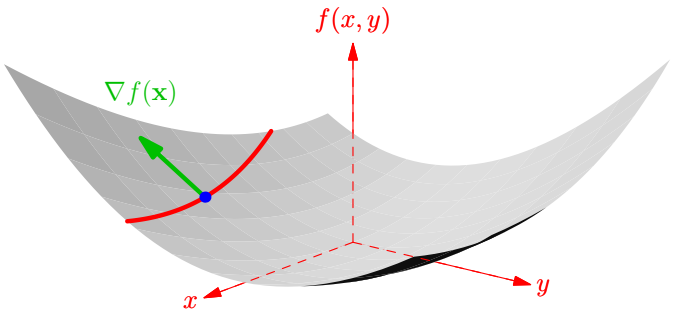
Example

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

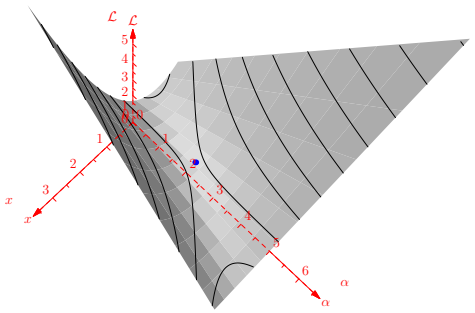
$$\boldsymbol{\nabla}_{\boldsymbol{x}} f(\boldsymbol{x}) = \begin{pmatrix} 2\, x - y \\ -x + 4\, y \end{pmatrix} = \alpha \boldsymbol{\nabla}_{\boldsymbol{x}} g(\boldsymbol{x}) = \alpha \begin{pmatrix} 1 \\ -2 \end{pmatrix}$$

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

Surface



Saddle-Point $y = -9/8$



Multiple Constraints

- Given an optimisation problem with multiple constraints

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

- We introduce multiple Lagrange multipliers

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

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

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

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

Example

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

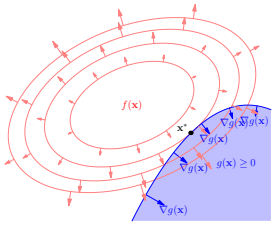
$$\begin{pmatrix} 2\,x - y - z \\ -x + 4\,y \\ 10\,z - x \end{pmatrix} = \alpha_1 \begin{pmatrix} 1 \\ -2 \\ -1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ 3 \\ 1 \end{pmatrix}$$

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

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

Outline

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



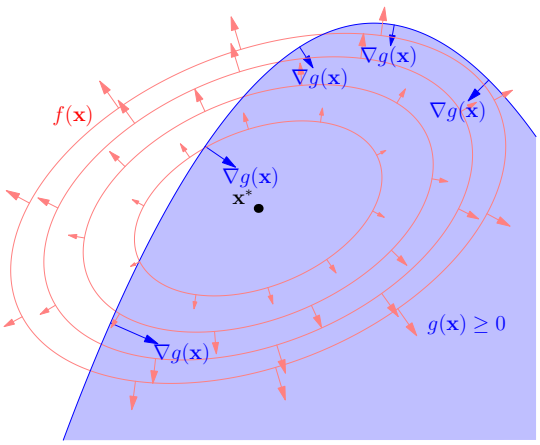
Inequality Constraints

- Suppose we have the problem

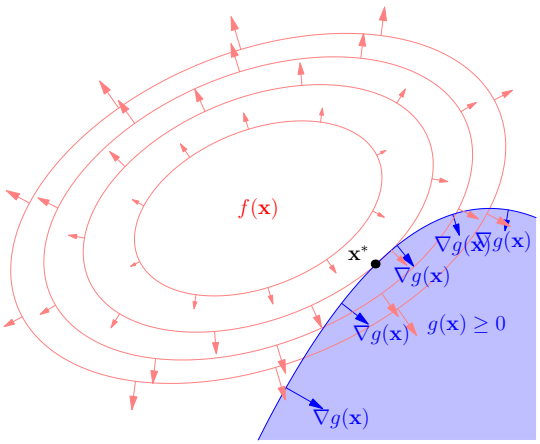
$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) \quad \text{subject to } g(\boldsymbol{x}) \geq 0$$

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

Inside Region



On the Boundary



KKT Conditions

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

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

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

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

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

Many Inequalities

- Given the problem

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

- We introduce multiple Lagrange multipliers

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

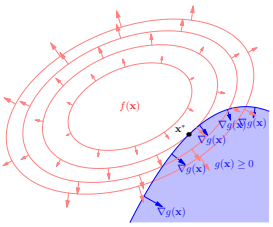
- The condition for an optima is

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

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

Outline

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



Solving the Lagrangian for $\boldsymbol{x}$

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

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

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

Dual Algorithm

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

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

- This leaves us with the dual problem

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

Linear Programming

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

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

- An equivalent way of writing the Lagrangian is

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

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

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	

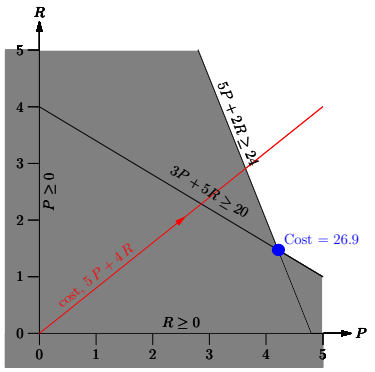
- We want to buy P kg potatoes and R kg of rice as cheaply as possible subject to fulfilling our vitamin requirement

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

$$\text{subject to} \quad 3P + 5R \geq 20 \quad \text{and} \quad 5P + 2R \geq 24$$

Linear Programme

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



Lagrangian

- This is equivalent to

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

- 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

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

Dual Problem

- The Lagrangian

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

leads to the dual problem

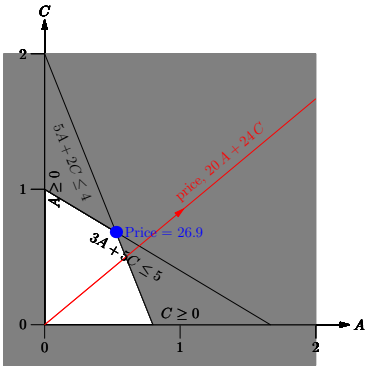
$$\max_{A,C} \; 20\,A + 24\,C$$

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

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

Dual Linear Programme

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



Why?

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

Quadratic Programme

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

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

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

Solution to Quadratic Programming Problem

- Using

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

- Then

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

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

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

Dual Quadratic Programming Problem

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

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

- We get the dual problem

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

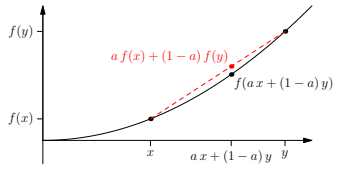
- If the constraints were inequality constraints then we have $\alpha_i > 0$
- We have exchanged one quadratic programme for another, but sometimes that very useful (e.g. SVMs)

Lessons

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

Advanced Machine Learning

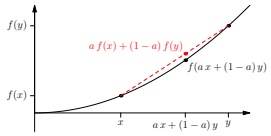
Convexity



Convex sets, convex functions, Jensen's inequality

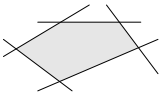
Outline

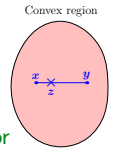
1. **Convex sets**
2. Convex functions
3. Jensen's inequality



Convex Regions

- Convex regions are familiar

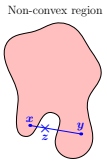




- For any two points $\boldsymbol{x}$ and $\boldsymbol{y}$ in a region $\mathcal{R}$ then for any $a \in [0,1]$ if

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

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



Convex Sets

- For any set of points, $\mathcal{S}$, where addition and scalar multiplication is defined then:

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

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

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

Positive Semi-Definite Matrices

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

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

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

- (We showed this also implies that all the eigenvalues are non-negative)
- We denote the fact that $\mathbf{M}$ is positive semi-definite by $\mathbf{M} \succeq 0$, and $\mathbf{M} \succ 0$ if it is positive definite
- The set of positive semi-definite (PSD) matrices (or kernels) form a convex set

Proof

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

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

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

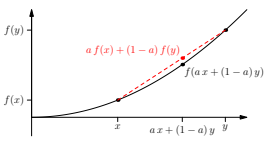
$$\begin{aligned}\boldsymbol{v}^\mathsf{T}\mathbf{M}_3\boldsymbol{v} &= \boldsymbol{v}^\mathsf{T}\left(a\,\mathbf{M}_1 + (1-a)\,\mathbf{M}_2\right)\boldsymbol{v}\\ &= a\,\boldsymbol{v}^\mathsf{T}\mathbf{M}_1\boldsymbol{v} + (1-a)\boldsymbol{v}^\mathsf{T}\mathbf{M}_2\boldsymbol{v}\\ &= a\,m_1 + (1-a)\,m_2\end{aligned}$$

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

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

Outline

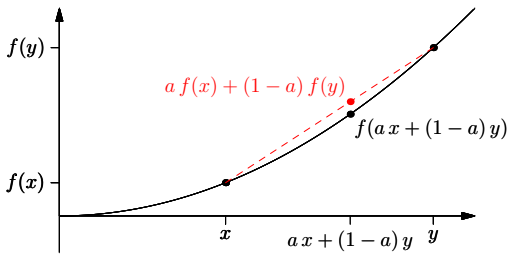
1. Convex sets
2. **Convex functions**
3. Jensen's inequality



Convex Functions

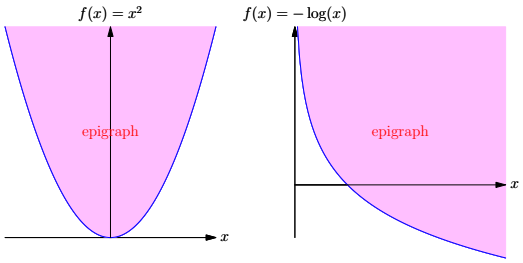
- Any function $f(x)$ is said to be a **convex function** if for any two points x and y and any $a \in [0,1]$

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



Epigraph

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



Convex-Down or Concave Functions

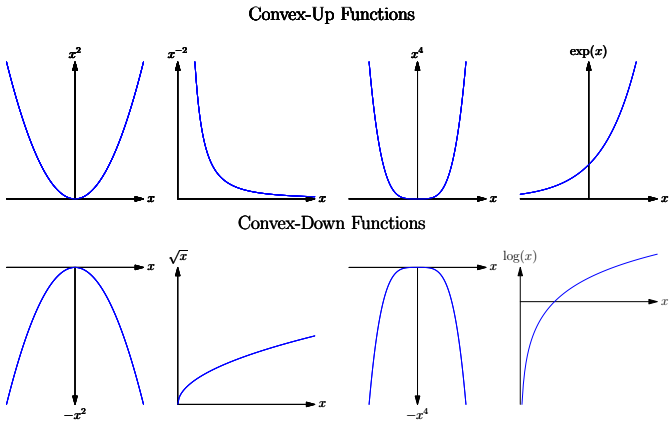
- Any function, $f(x)$, that satisfies the inverse inequality

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

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

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

Examples



Linear Functions

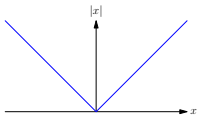
- Linear functions are given by

$$f(x) = m\,x + c$$

- They satisfy the **equality**

$$f(a\,x + (1-a)\,y) = a f(x) + (1-a) f(y)$$

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



Strictly Convex Function

- Functions that satisfy the strict inequality

$$f(a\,x + (1-a)\,y) < a f(x) + (1-a) f(y)$$

are said to be **strictly convex functions**

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

Convexity in High Dimensions

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

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

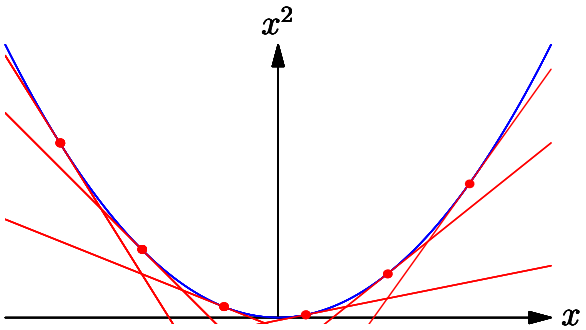
for any $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ and any $a \in [0,1]$ then $f(\boldsymbol{x})$ is a convex function

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

Properties of Convex Functions

- Convex functions lie on or above any tangent line

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



Second Derivatives

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

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

therefore $f''(x) \geq 0$ at all points x

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

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

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

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

Proving Convexity

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

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

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

Sums of Convex Functions

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

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

is convex

- Proof

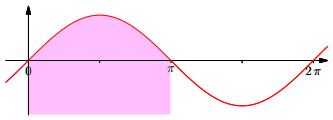
$$g''(x) = \sum_i a_i\,f_i''(x)$$

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

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

Convex Functions Defined on Convex Sets

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



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

Constraints

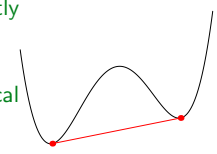
- Often we impose constraints on the set of points, e.g.

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

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

Unique Minimum

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



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

Convex Set of Minima

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

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

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

Linear Regression

- For linear regression the error function

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

is convex

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

Regularised Linear Regression

- In ridge regression we minimise a loss

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

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

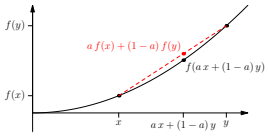
$$L(\boldsymbol{w}) = \|\mathbf{X}\,\boldsymbol{w} - \boldsymbol{y}\|^2 + \eta\,\|\boldsymbol{w}\|_1$$

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

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

Outline

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



Jensen's Inequality

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

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

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

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

Proof

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

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

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

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

- Taking expectations of both sides

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

Simple Proofs with Jensen's Inequality

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

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

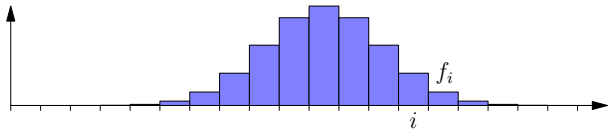
(i.e. variance are non-negative)

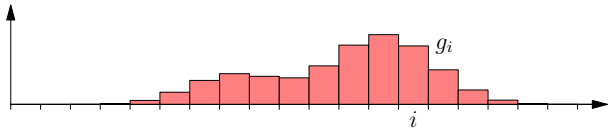
- The KL-divergence $\mathrm{KL}(f\|g)$ between two categorical probability distributions $(f_1, f_2, \ldots)$ and $(g_1, g_2, \ldots)$ is define as

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

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

Kullback-Leibler Divergence





$$\mathrm{KL}(\boldsymbol{f}\|\boldsymbol{g}) = -\sum_{i=1}^{n} f_i \log\left(\frac{g_i}{f_i}\right) = 2.21$$

Proof of Gibb's Condition

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

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

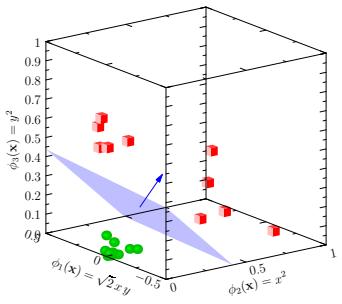
- We will meet KL-divergences later on

Lessons

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

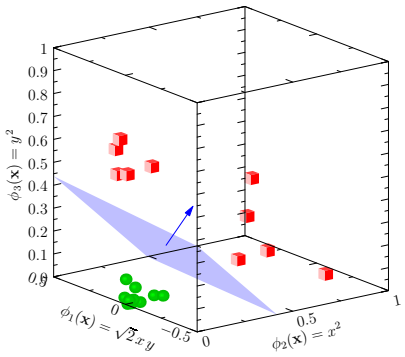
Advanced Machine Learning

Support Vector Machines



Support Vector Machines, maximum margins

Outline

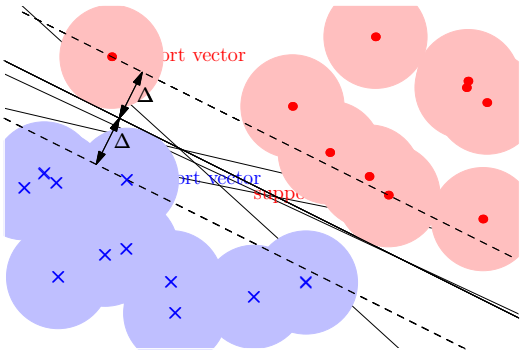


Support Vector Machines

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

Linear Separation of Data

- SVMs classify linearly separable data



- Finds maximum-margin separating plane

Extended Feature Space

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

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

$r \gg p$

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

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

Kernel Trick

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

$$K(\boldsymbol{x}_i, \boldsymbol{x}_j) = \vec{\phi}(\boldsymbol{x}_i)^{\mathsf{T}} \vec{\phi}(\boldsymbol{x}_j)$$

(like an eigenvector decomposition of a matrix)

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

Kernel Functions

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

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

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

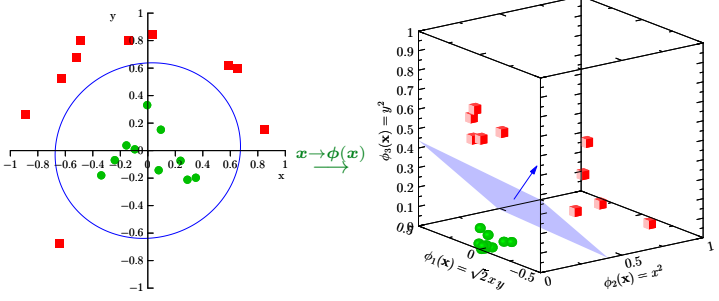
- Consider the mapping

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

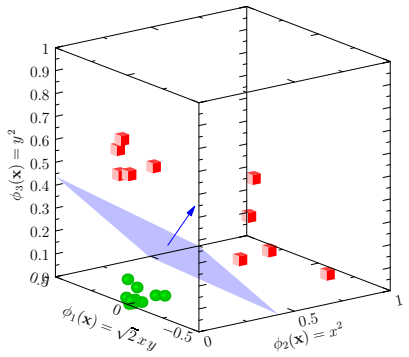
Non-linearly Separation of Data

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

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



Outline

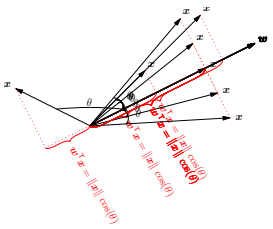


Inner Product

- Recall the inner or dot product

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

- If $\|\boldsymbol{w}\| = 1$ then $\boldsymbol{x}^\mathsf{T}\boldsymbol{w} = \|\boldsymbol{x}\|\, \cos(\theta)$



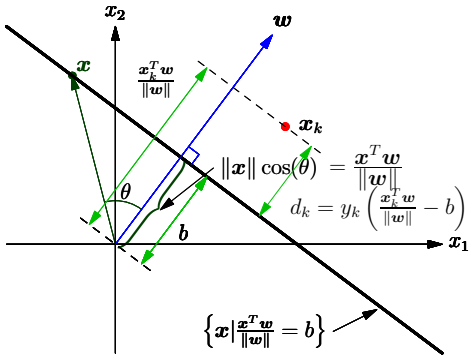
Maximise Margin

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

$$y_k \left(\frac{\boldsymbol{w}^\mathsf{T}\boldsymbol{x}_k}{\|\boldsymbol{w}\|} - b \right) \geq \Delta$$

where Δ is the margin

Distance to hyperplanes



Constrained Optimisation

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

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

- If we divide through by Δ

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

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

$$y_k \left(\hat{\boldsymbol{w}}^\mathsf{T}\boldsymbol{x}_k - \hat{b} \right) \geq 1$$

Quadratic Programming Problem

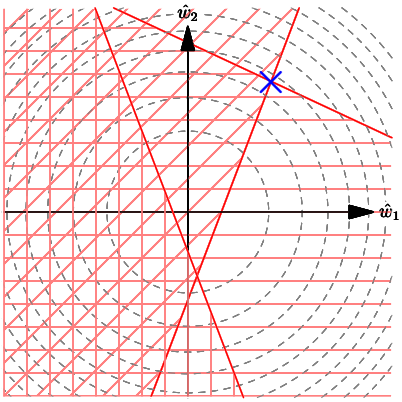
- Note that as $\hat{\boldsymbol{w}} = \boldsymbol{w}/(\Delta\|\boldsymbol{w}\|)$

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

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

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

Quadratic Programming in SVMs

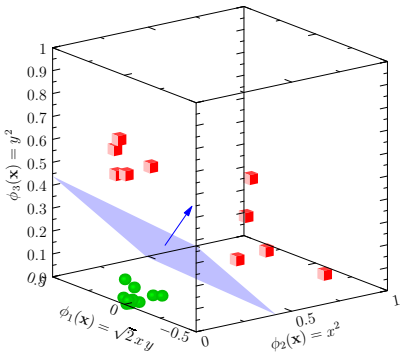


Quadratic Programming

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

Outline

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



Extend Feature Space

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

$$\boldsymbol{x} \rightarrow \vec{\phi}(\boldsymbol{x})$$

- The components of $\vec{\phi}(\boldsymbol{x})$ will typically be (non-linear) functions of $\boldsymbol{x}$ (e.g. $\phi_1(\boldsymbol{x}) = x_1^2$, $\phi_1(\boldsymbol{x}) = x_2^2$, $\phi_1(\boldsymbol{x}) = \sqrt{2}\,x_1\,x_2$)
- We are free to choose whatever mappings we like
- There may be many more components of $\vec{\phi}(\boldsymbol{x})$ than of $\boldsymbol{x}$ making it easier to find a linear separation of the two classes

Lagrangian

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

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

- We can write this as a Lagrange problem

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

where

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

subject to $\alpha_k \geq 0$

Obtaining the Dual Form of the Problem

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

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

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

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

The Dual Problem

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

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

- subject to constraints

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

- The Hessian of $\mathcal{L}(\boldsymbol{\alpha})$ is a matrix of the form $-\frac{1}{2}\mathbf{X}^{\mathsf{T}}\mathbf{X}$ where $X_{ik} = y_k\,\phi_i(\boldsymbol{x}_k)$ (note this is negative semi-definite so there is a unique maximum)

Kernel Trick

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

$$K(\boldsymbol{x}, \boldsymbol{y}) = \vec{\phi}(\boldsymbol{x})^{\mathsf{T}}\vec{\phi}(\boldsymbol{y})$$

- As $\vec{\phi}(\boldsymbol{x}_k)^{\mathsf{T}}\vec{\phi}(\boldsymbol{x}_l)$ appears in the dual problem we can express the dual problem as finding α_k's that maximise

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

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

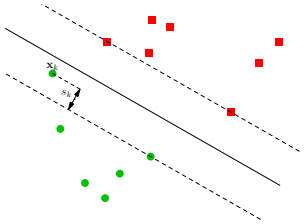
Sequential Minimal Optimisation

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

Soft Margins

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

$$y_k(\boldsymbol{x}_k^{\mathsf{T}}\boldsymbol{w} - b) \geq 1 - s_k$$



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

Dual Problem with Slack Variables

- The Lagrangian with slack variables is

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

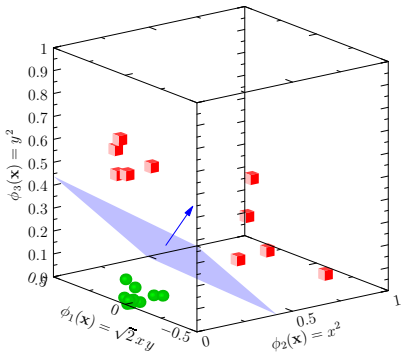
  where β_i are Lagrange multipliers that ensure $s_i \geq 0$ (note that $\beta_i \geq 0$—this is the KKT condition)

- Now minimising with respect to s_i

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

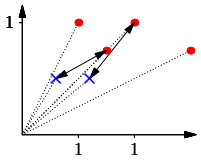
- Or $\alpha_i = C - \beta_i$. Since $\beta_i \geq 0$ the constraint is $\alpha_i \leq C$ (recall also $\alpha_i \geq 0$)

Outline



Getting SVMs to Work Well

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



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

Optimising C

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

Choosing the Right Kernel Function

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

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

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

Multi-Class Problems

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

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

- In both cases choose the class which the classifier is most certain about

SVM Libraries

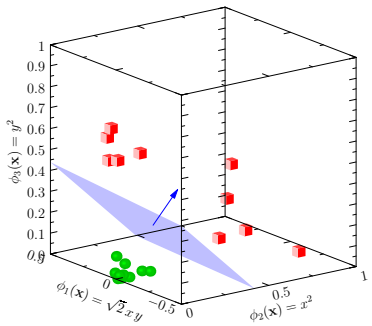
- Although SVMs have unique solutions, they require very well written optimisers
- If you have a large data set they can be very slow
- There are good libraries out there, svmlib, svm-lite, etc.
- These will often automate normalisation of data and grid search for parameters

Conclusions

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

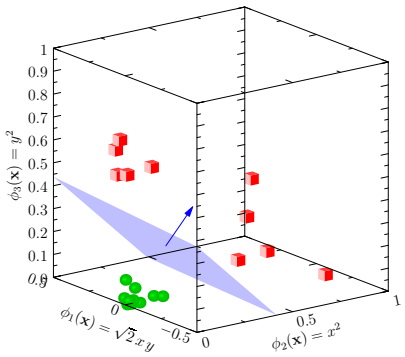
Advanced Machine Learning

Kernel Trick



The Kernel Trick, SVMs, Regression

Outline



Kernels

- Kernels are functions of two variables which can be factorised

$$K(\boldsymbol{x}, \boldsymbol{y}) = \vec{\phi}(\boldsymbol{x})^\mathsf{T} \vec{\phi}(\boldsymbol{y})$$

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

Dual Form

- Recall that the dual problem for an SVM is

$$\max_{\boldsymbol{\alpha}} \sum_{k=1}^{P} \alpha_k - \frac{1}{2} \sum_{k,l=1}^{P} \alpha_k \, \alpha_l \, y_k \, y_l \, \vec{\phi}(\boldsymbol{x}_k)^\mathsf{T} \vec{\phi}(\boldsymbol{x}_l)$$

- subject to $0 \leq \alpha_k (\leq C)$
- But since $K(\boldsymbol{x}_k, \boldsymbol{x}_l) = \vec{\phi}(\boldsymbol{x}_k)^\mathsf{T} \vec{\phi}(\boldsymbol{x}_l)$ the dual problem becomes

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

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

Classifying New Data

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

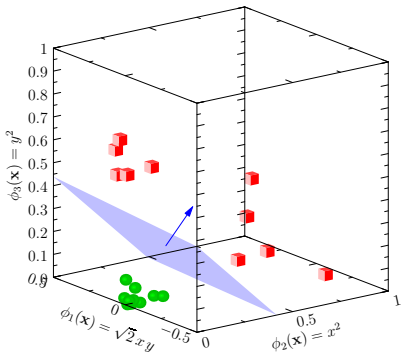
$$y = \operatorname{sgn}\left(\boldsymbol{w}^\mathsf{T} \vec{\phi}(\boldsymbol{x}) - b\right) \qquad \text{but} \qquad \boldsymbol{w} = \sum_{k=1}^{m} \alpha_k \, y_k \, \vec{\phi}(\boldsymbol{x}_k)$$

- In the dual representation this becomes

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

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

Outline



Recap on Eigen Systems

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

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

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

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

- Such matrices are said to be orthogonal

Eigen Decomposition

- From the eigenvalue equation $\mathbf{M}\,\boldsymbol{v}^{(i)} = \lambda_i \boldsymbol{v}^{(i)}$

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

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

$$\mathbf{M} = \mathbf{V}\boldsymbol{\Lambda}\mathbf{V}^\mathsf{T} = \sum_{k=1}^{n} \lambda_k \, \boldsymbol{v}^{(k)} \, \boldsymbol{v}^{(k)\mathsf{T}}$$

Or

$$M_{ij} = \sum_{k=1}^{n} \lambda_k \, v_i^{(k)} \, v_j^{(k)}$$

Eigenfunctions

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

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

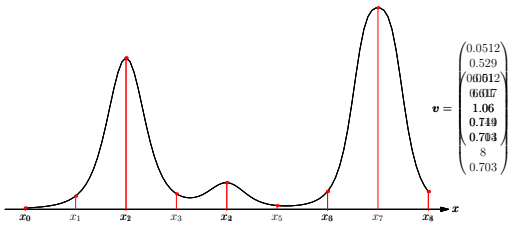
- In general there will be a denumerable set of eigenfunctions $\psi_i(\boldsymbol{x})$ where

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_i \lambda_i\, \psi_i(\boldsymbol{x})\, \psi_i(\boldsymbol{y})$$

- This is known as Mercer's theorem

Limit Process

- Consider sampling a function at a set of points

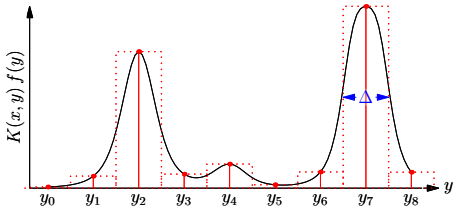


- In the limit where the number of sample points goes to infinity the vector more closely approximates a function
- Instead of the indices being numbers we use $i \leftarrow x_i$

Linear Operators

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

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



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

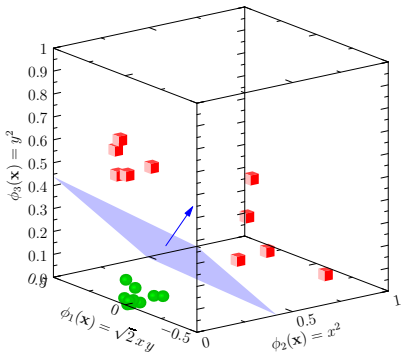
SVM Kernels

- If we define $\phi_i(\boldsymbol{x}) = \sqrt{\lambda_i}\psi_i(\boldsymbol{x})$ then

$$K(\boldsymbol{x}, \boldsymbol{y}) = \sum_i \lambda_i\, \psi_i(\boldsymbol{x})\, \psi_i(\boldsymbol{y}) = \sum_i \phi_i(\boldsymbol{x})\, \phi_i(\boldsymbol{y})$$

- This is the definition of a SVM kernel we started with
- Note that for $\phi_i(\boldsymbol{x})$ to be real $\lambda_i \geq 0$ for all i
- If $\lambda_i < 0$ the "distance" between points in the extended feature space can be negative!
- If we use a kernel that isn't positive semi-definite then the Hessian of the dual objective function will not be negative semi-definite and there will be a maximum where $\boldsymbol{\alpha}$ diverges

Outline



Positive Semi-Definite Kernels

- Kernels (or matrices) that have eigenvalues $\lambda_i \geq 0$ are called positive semi-definite
- (If the eigenvalues are strictly positive $\lambda_i > 0$ the kernels or matrices are called positive definite)
- Positive semi-definite kernels can always be decomposed into a sum of real functions

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

Properties of Positive Semi-Definiteness

- Since

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

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

$$\int f(\boldsymbol{x})\, K(\boldsymbol{x}, \boldsymbol{y})\, f(\boldsymbol{y})\, \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{y} = \int f(\boldsymbol{x}) \sum_i \phi_i(\boldsymbol{x})\, \phi_i(\boldsymbol{y})\, f(\boldsymbol{y})\, \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{y}$$

$$= \sum_i \left(\int f(\boldsymbol{x})\, \phi_i(\boldsymbol{x})\, \mathrm{d}\boldsymbol{x} \right)^2 \geq 0$$

Positive Semi-Definiteness

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

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

    where $\phi(\boldsymbol{x})$ are real functions
  - For any real function $f(x)$

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

Adding Kernels

- We can construct SVM kernels from other kernels
- If $K_1(\boldsymbol{x}, \boldsymbol{y})$ and $K_2(\boldsymbol{x}, \boldsymbol{y})$ are valid kernels then so is $K_3(\boldsymbol{x}, \boldsymbol{y}) = K_1(\boldsymbol{x}, \boldsymbol{y}) + K_2(\boldsymbol{x}, \boldsymbol{y})$

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

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

Product of Kernels

- If $K_1(\boldsymbol{x}, \boldsymbol{y})$ and $K_2(\boldsymbol{x}, \boldsymbol{y})$ are valid kernels then so is $K_3(\boldsymbol{x}, \boldsymbol{y}) = K_1(\boldsymbol{x}, \boldsymbol{y})\, K_2(\boldsymbol{x}, \boldsymbol{y})$
- Writing $K_1(\boldsymbol{x}, \boldsymbol{y}) = \sum_i \phi_i^1(\boldsymbol{x})\, \phi_i^1(\boldsymbol{y})$ and $K_2(\boldsymbol{x}, \boldsymbol{y}) = \sum_j \phi_j^2(\boldsymbol{x})\, \phi_j^2(\boldsymbol{y})$ then

$$K_3(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i,j} \phi_{ij}^3(\boldsymbol{x})\, \phi_{ij}^3(\boldsymbol{y})$$

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

Exponentiating Kernels

- If $K(\boldsymbol{x}, \boldsymbol{y})$ is a valid kernel so is $K^n(\boldsymbol{x}, \boldsymbol{y})$ (by induction)
  - ⋆ Assume $K(\boldsymbol{x}, \boldsymbol{y}) \succeq 0$ this satisfies base case
  - ⋆ If $K(\boldsymbol{x}, \boldsymbol{y})^{n-1} \succeq 0$ then

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

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

$$\mathrm{e}^{K(\boldsymbol{x}, \boldsymbol{y})} = \sum_i \frac{1}{i!} K^i(\boldsymbol{x}, \boldsymbol{y}) = 1 + K(\boldsymbol{x}, \boldsymbol{y}) + \frac{1}{2} K^2(\boldsymbol{x}, \boldsymbol{y}) + \cdots$$

but each term in the sum is a kernel

Gaussian Kernel

- Now $\boldsymbol{x}^\mathsf{T}\boldsymbol{y}$ is a valid kernel because it is of the form $\sum_i \phi_i(\boldsymbol{x})\, \phi_i(\boldsymbol{y})$ where $\phi_i(\boldsymbol{x}) = x_i$
- For $\gamma > 0$ we have $2\boldsymbol{x}^\mathsf{T}\boldsymbol{y}/\gamma \succeq 0$
- Thus $\exp(2\, \boldsymbol{x}^\mathsf{T}\boldsymbol{y}/\gamma) \succeq 0$
- Since $\exp(\boldsymbol{x}^\mathsf{T}\boldsymbol{x}/\gamma)$ and $\exp(\boldsymbol{y}^\mathsf{T}\boldsymbol{y}/\gamma)$ are positive numbers

$$\begin{aligned} \frac{\mathrm{e}^{2\, \boldsymbol{x}^\mathsf{T}\boldsymbol{y}/\gamma}}{\mathrm{e}^{\boldsymbol{x}^\mathsf{T}\boldsymbol{x}/\gamma}\, \mathrm{e}^{\boldsymbol{y}^\mathsf{T}\boldsymbol{y}/\gamma}} &= \mathrm{e}^{-\boldsymbol{x}^\mathsf{T}\boldsymbol{x}/\gamma + 2\, \boldsymbol{x}^\mathsf{T}\boldsymbol{y}/\gamma - \boldsymbol{y}^\mathsf{T}\boldsymbol{y}/\gamma} \\ &= \mathrm{e}^{-\|\boldsymbol{x}-\boldsymbol{y}\|^2/\gamma} \succeq 0 \end{aligned}$$

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 an industry of research 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 have become very important is in document classification
- This requires comparing strings
- There are a large number of kernels developed to do this

Spectrum Kernel

- A simple way to compare documents is to collect a histogram of all occurrences of substrings of length p
- This is known as a p-spectrum
- A p-spectrum kernel counts the number of common substrings

$s = \texttt{statistics}$ $\quad \mathcal{S}_3(s) = \{\texttt{sta}, \texttt{tat}, \texttt{ati}, \texttt{tis}, \texttt{ist}, \texttt{sti}, \texttt{tic}, \texttt{ics}\}$

$t = \texttt{computation}$ $\quad \mathcal{S}_3(t) = \{\texttt{com}, \texttt{omp}, \texttt{mpu}, \texttt{put}, \texttt{uta}, \texttt{tat}, \texttt{ati}, \texttt{tio}, \texttt{ion}\}$

- $K(s, t) = 2$ ("tat" and "ati")

All Subsequences Kernel

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

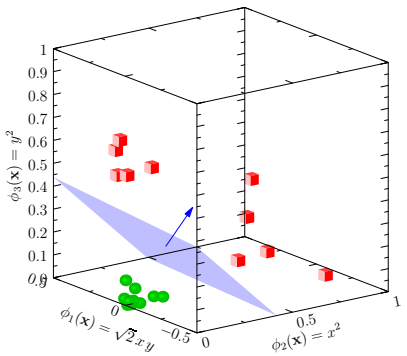
Other Kernel Applications

- String kernels for comparing subsequences are used in bioinformatics
- Kernels have been developed for comparing trees (e.g. for computer program evaluation, XML, etc.)
- Kernels have also been developed for comparing graphs (e.g. for comparing chemicals based on their molecular graph)

Fisher Kernels

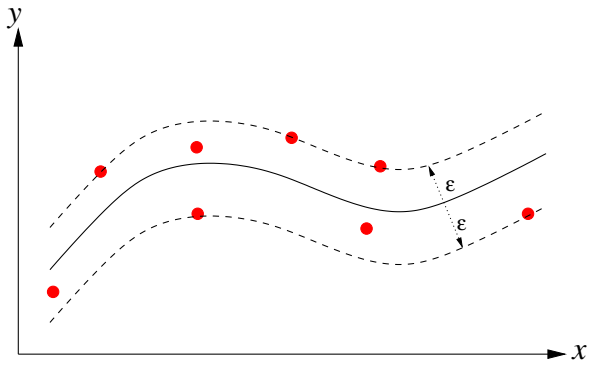
- In an attempt to build kernels that capture more domain knowledge, kernels are constructed from other learning machines
- An example of this are "Fisher kernels" whose features come from an Hidden Markov Model (HMM) trained on the data
- These tend to have better discriminative power than the underlying model (HMM), and has a better feature set than a SVM using a generic kernel

Outline



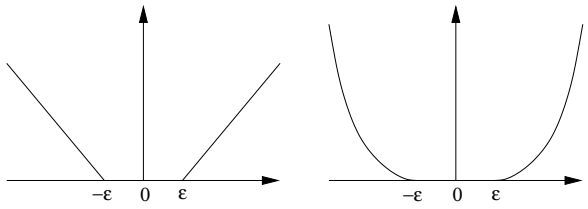
Regression with Margins

- SVMs can be modified to perform regression



Error Functions

- Can introduce slack variables with different errors



- This can be transformed to a quadratic programming problem

Ridge Regression Using Kernels

- We can also solve regression problems without using margins
- To solve a regression problem once again the problem is set up as a quadratic programming problem

$$\min_{\boldsymbol{w}} \lambda \|\boldsymbol{w}\|^2 + \sum_i \left(y_i - \boldsymbol{w}^\mathsf{T}\boldsymbol{\phi}(\boldsymbol{x_i})\right)^2$$

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

Kernel Methods

- Kernel methods where we project into an extended feature space is also used with algorithms
  - ⋆ Fisher discriminant analysis
  - ⋆ Principle component analysis
  - ⋆ Canonical correlation analysis
  - ⋆ Gaussian Processes
- These are also extremely power machine learning algorithms

Summary

- SVMs require a positive definite kernel function
- These can be built from simpler function
- There is an important industry of people creating new kernels for different application
- SVMs are just one example of a host of machine that
  - ⋆ use the kernel trick
  - ⋆ often use linear constraints
  - ⋆ tend to be convex optimisation problems