Machine Learning And Neural Networks (CM3015)

Course Notes

Felipe Balbi

July 27, 2021

Contents

Week 1

Key Concepts

- explain the concepts of clustering and dimensionality reduction
- Describe various types of machine learning problem
- Describe various applications of machine learning

1.101 Applications of Machine Learning

Machine Learning is a branch of artificial intelligence that enables machines to learn by example. Carried by the increase in data availability and computational power, we can already experiences applications of machine learning in our everyday lives: mobile phones, personal assistants, language translators, etc.

One application of machine learning are the e-passport gates at some airports which rely on face recognition to identify passengers with high probability.

Computer Vision systems can also be used to detect and classify human posture and facial expressions. Machine Learning can also be applied to other types of data such as text (handwriting recognition) or audio (speech recognition).

These systems collect and process vast amounts of data and the issue of privacy arises. We must be conscious about what data has been recorded, who has access to it, and how it can be used.

Autonomous Vehicles are a focus in machine learning research. They pose interesting and complex challenges both technically and ethically. Vehicles need to be able to detect and avoid pedestrians and other objets on the road. In the case of the accident, who's to blame? The owner of the vehicle? The company who made the car? The software engineers who built the system?

Another common system in our daily lives are recommender systems. We encounter them in streaming services, online shopping experiences, MOOC education providers, and many more. The main goal of these systems is to recommend other items similar to what we have already *consumed*. Because these systems are also used to suggest similar content to what we already watch, they may end up skewing our view of the world.

Generative Machine Learning are models that can generate new data based on a sample, for example given a sample of someone's handwriting, we want to produce more text in the same style.

Another application is related to Sensor-based Activity Recognition. Here the goal is to detect what activity the user is executing (sitting, walking, running, playing footbal) based on the data from sensors the user's wearing.

1.102 Types of ML

Machine Learning is used when we want to learn from data rather than hardcode a solution. There are two types of machine learning

Supervised Learning in supervised learning, the label y is associated with every sample x. We're trying to learning mapping from x to y

Unsupervices Learning here the goal is usually about clustering data in subgroups. For example, given a dataset containing pictures of animals, separate the images by animal.

We can use the decision tree depicted in figure 1 to decide which type of Machine Learning application to apply:

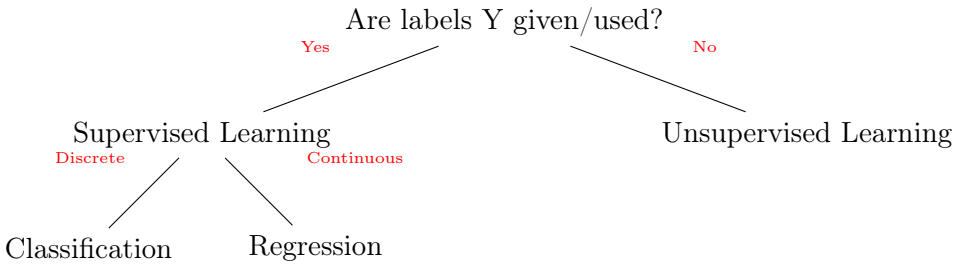

Figure 1: Decision Tree

One final type is Reinforcement Learning where the goal is to learn a sequence of actions that entail some reward. This can be used to teach a machine how to play a specific video game.

Week 2

Key Concepts

- explain the concepts of clustering and dimensionality reduction
- Describe various types of machine learning problem
- Describe various applications of machine learning

1.202 Further reading

- Chapter 1, sections 1.1 to 1.2 of the course textbook (Chollet).
- Chapter 1 of Alpaydin, E. Introduction to machine learning. (Cambridge, MA: MIT Press, 2014) 3rd edition [ISBN 9780262028189].
- Chapter 1, Introduction, up to and including section 1.3, of the following textbook gives a good introduction to the topic of ML: Murphy, K. Machine learning: a probabilistic perspective. (Cambridge, MA: MIT Press, 2012) [ISBN 9780262018029]

Week 3

Key Concepts

- Explain how a simple nearest neighbour algorithm works
- Describe the Decision Tree Classifier
- Evaluate a supervised classification algorithm on a dataset

2.101 Introduction to supervised Learning

Classification is a type of supervised learning where the labels on a data are discrete and categorical.

2.201 K-Nearest Neighbours Classification

The k-NN algorithm works on the premise that things are similar if they are closer together. Essentially, we measure the distance from a *test* sample X^* to every sample in the *training* set X, in other words, we compute $(X^* - X_i)^2$.

We can employ Euclidean distance ($\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$) or the Manhattan distance ($|(x_2 - x_1)| + |(y_2 - y_1)|$) for this computation.

The two parameters for the algorithm are the number k and the distance algorithm used.

K-Nearest Neighbours is known as a lazy learning algorithm, which means that we don't generalize on the training dataset until we want to make a query.

2.301 Decision tree

Decision Tree is a very versatile machine learning algorithm, because they are capable of handling both classification and regressions tasks. They can also handle non-linear datasets.

This algorithm can be used a the basic classifier in Random Forests, which is among the most powerufl class of machine learning algorithm.

To illustrate how Decision Tree algorithm, we will look at how we can classify Hares vs Rabbits. Figure 1 shows and example of how the decision tree could look like. When applying the algorithm to a new data to be classified, we descend through the tree until we get to a leaf node.

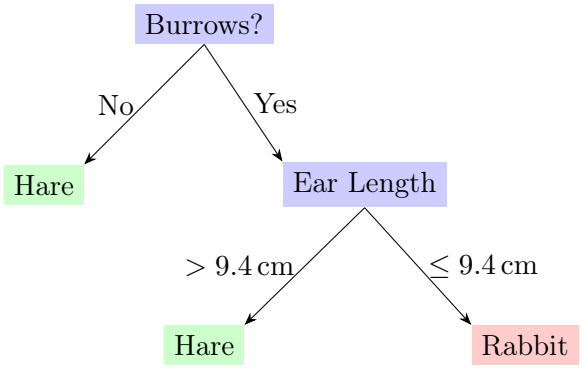

Figure 2: Decision Tree

Decision Trees are referred to as *White Box* models, this means they are easy to interpret, because of the hierarchical nature of the classification rules which is easy to visualize. Conversely, *Black Box* models make decisions in a more opaque process.

There are several types of Decision Tree algorithms, one of which is known as Classification And Regression Tree, or CART. It's a binary tree which can be used, as the name suggests, for both classification and regression.

A decision tree is prone to overfitting, which is when the model fits the training dataset perfectly; this makes it harder to generalize the trained model to other sets of data. Finding the optimal split and feature combination is an NP-complete problem.

Week 4

Key Concepts

- Explain how a simple nearest neighbour algorithm works
- Describe the Decision Tree Classifier
- Evaluate a supervised classification algorithm on a dataset

2.501 Classifier evaluation

One way of evaluating the performance of a classifier is to measure its accuracy, this, however, is not always a good measure of the quality of a classifier.

Another approach to measuring quality of the classifier is to employ a Confusion Matrix. This matrix lets us compare a true condition vs a predicted condition, Like shown in table 1.

Table 1: Confusion Matrix

	Condition Positive	**Condition Negative**
Predicted Positive	True Positive (TP)	False Positive (FP)
Predicted Negative	False Negative (FN)	True Negative (TN)

The *True Positive Rate*, also known as Recall or Sensitivity, tells us how likely the model is to predict the correct value. It's computed as shown below:

$$TPR = \frac{TP}{TP + FN}$$

The *Precision* or *Positive Predictive Value* tells us how likely the prediction is to be correct, given a positive prediction. It's computed as shown below:

$$PPV = \frac{TP}{TP + FP}$$

The *True Negative Rate*, also known as Specificity or Selectivity, is computed as:

$$TNR = \frac{TN}{TN + FP}$$

The *False Negative Rate*, also known as Miss Rate, is computed as:

$$FNR = \frac{FN}{FN + TP}$$

The *False Positive Rate*, also known as Fall-out is computed as:

$$FPR = \frac{FP}{FP + TN}$$

2.602 Further reading

- `http://scikit-learn.org/stable/modules/neighbors.html`
- `https://scikit-learn.org/stable/modules/tree.html`
- Chapter 1, section 1.2 of the course textbook (Chollet), also briefly mentions decision tree classifiers.
- Sections 2.1 and 2.5 of Ethem Alpaydin's book provide a good overview of supervised classification: Alpaydin, E. Introduction to machine learning. (Cambridge, MA: MIT Press, 2014) 3rd edition [ISBN 9780262028189].
- Alpaydin's book also discusses decision trees in depth in Chapter 9, sections 9.1 to 9.3.

Week 5

Key Concepts

- Explain the idea behind gradient descent
- Apply linear regression on a dataset.
- Explain the concept of linear regression and interpret results.

3.102 Linear regression

Linear Regression is a method for predicting output based on a linear combination of the input. Figure 3 shows an example of this.

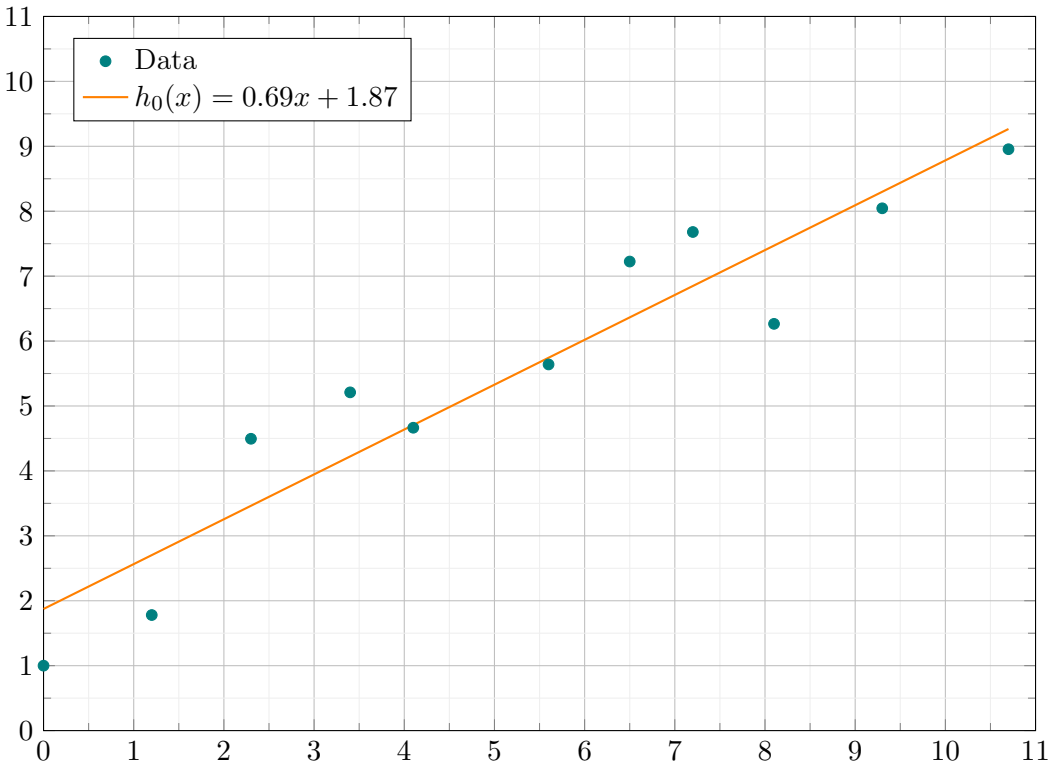

Figure 3: Linear Regression Example

The idea of Linear Regression is exactly that: try to fit a line through the data to predict new y values based on x input. There are two important parameters in the regression line: θ_0 is the y intercept point and θ_1 determines the gradient. With these two parameters we can construct the line's equation $h_0(x) = \theta_1 x_1 + \theta_0$. We call it $h_0(x)$ because it represents our *hypothesis*.

The hypothesis can also be represented as a summation:

$$h_0(x) = \sum_{j=0}^{1} \theta_j x_j$$

Where $x_0 = 1$. The same equation can be represented in vector notation:

$$h_0(x) = \begin{bmatrix} 1 & x \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

When modelling a linear regression, what we can do is plot the line anywhere and measure the *error* from the line to each of the points in the plot. The function shown below is the L2 loss or *Mean Squared Error*, it's a common function for computing error between regression line and x inputs.

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)} - y^{(i)} \right)^2$$

The goal is to minimize the function $J(\theta)$, thus minimizing the error of the regression line.

3.202 Further reading

- Linear Algebra with Python
- Goodfellow, I., Y. Bengio and A. Courville Deep learning. (Cambridge, MA: MIT Press, 2017) [ISBN 9780262035613] Chapter 2 Linear Algebra

Week 6

Key Concepts

- Explain the idea behind gradient descent
- Apply linear regression on a dataset.
- Explain the concept of linear regression and interpret results.

3.301 Gradient descent in 1D

Revisiting previous lectures, we learned that we can approximate a solution for a linear regression problem by starting a random regression and trying to iteratively minimize a Loss function given by:

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^{(i)})^2$$

As an example, we can use a simple 3-point data as shown in figure 4. Initial θ is 0.25.

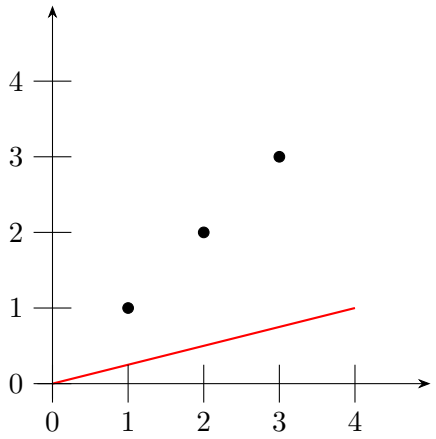

Figure 4: Input Data

We measure the distance from each point to the random regression line as shown in figure 5.

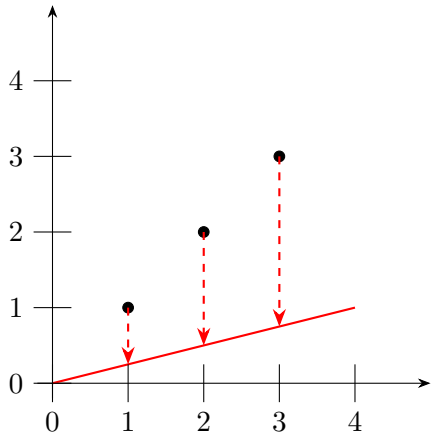

Figure 5: Input Data

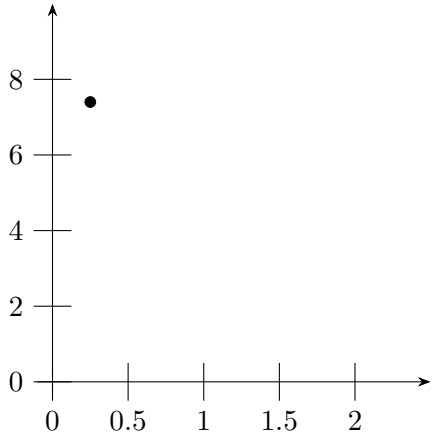

Figure 6: Loss $J(\theta)$

From this, we can compute $J(\theta_1) = 7.4$. As we compute the losses, we can plot the result in another graph, shown in figure 6.

Increasing our θ to 0.5, we get a new error, shown in figure 7. And that results in a new error $J(\theta_2) = 3.3$, which we update in our plot as shown in figure 8.

We repeat the process again with $\theta = 0.75$, which gives an error of $\theta_3 = 0.82$. The results are shown in figures 9 and 10.

We repeat this process until we find a *Global Minimum* and this is work of *Gradient Descent* algorithm. Given a convex loss function, we **know** there has to be a minimum value and gradient descent tries to find it by modifying the parameters of the loss function.

At each step of the Gradient Descent algorithm, the gradient or slope of the function is computed in order for the algorithm to make a decision of where to go next. In order to do that, we must calculate the derivative of the Loss Function.

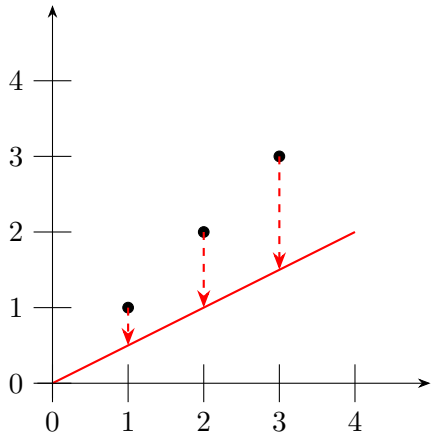

Figure 7: Input Data

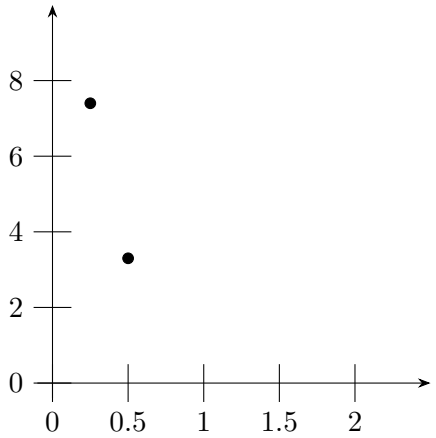

Figure 8: Loss $J(\theta)$

$$
\begin{aligned}
J_1'(\theta) &= \frac{\partial J(\theta)}{\partial \theta_1} \\
&= 2 \cdot \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)}) \cdot x_1^{(i)} \\
&= \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)}) \cdot x_1^{(i)}
\end{aligned}
$$

Gradient Descent applies a convergence rate α to the slope computed by the derivative of the loss function and uses that to compute the new value for θ_1, i.e. $\theta_1^{(2)} = \theta_1^{(1)} - \alpha J_1'(\theta_1^{(1)})$

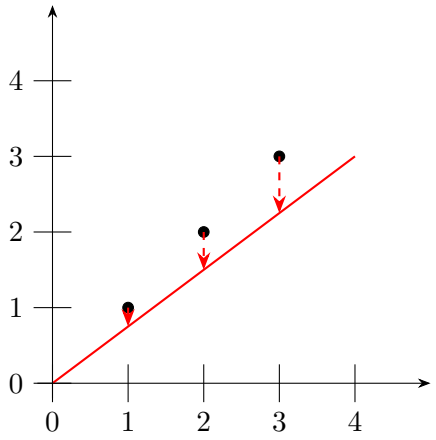

Figure 9: Input Data

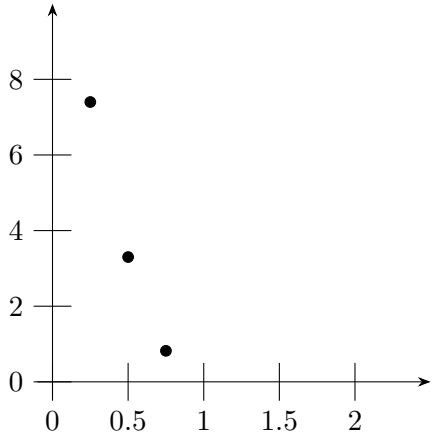

Figure 10: Loss $J(\theta)$

3.303 Gradient descent in 2D

We skipped θ_0 in previous video. The equation is shown below.

$$\begin{aligned} J_0'(\theta) &= \frac{\partial J(\theta)}{\partial \theta_0} \\ &= 2 \cdot \frac{1}{2m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)}) \\ &= \frac{1}{m} \sum_{i=1}^{m} (\theta_0 + \theta_1 x_1^{(i)} - y^{(i)}) \end{aligned}$$

When running gradient in higher dimensions we just compute more partial derivatives. For 3D we must compute partial derivatives with respect to θ_0, θ_1, and θ_2. For more dimensions, just add more θ_n parameters.

To summarise, the multivariate linear model is given by:

$$\begin{aligned} h_\theta(x) &= \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n \\ &= \sum_{j=0}^{n} \theta_j x_j && \text{with } x_0 = 1 \\ &= \theta^\intercal x && \text{with } x_0 = 1 \end{aligned}$$

3.305 Data scaling

When using multivariate data we can run into situations where the scale for each of the features in the data can be vastly different, as shown in table 2 below.

Table 2: Data Scale Can be Different

x_1	x_2	x_3	x_4	y
0	0	10000	1	13.1
16	12000	7000	0	17.8
8	500	7000	0	10.3
45	10000	5000	1	23.0
65	1000	12000	1	30.8

We can sort this out by scaling each feature so they all sit in a similar range. This is referred to as *Feature Scaling*. A common way of achieving this is *min-max normalisation*, which can be achieved with the equation below:

$$x_j^s = \frac{x_j - min(x_j)}{max(x_j) - min(x_j)}$$

Applying this to the previous table 2 results in the scaled table 3 shown below.

What this means in practice is that Gradient Descent runs faster when every dimension is of comparable range.

There are other normalisation techniques, such as Range Normalisation, which achieve the same thing by slightly different method. With Range Normalisation we *center* the data around the mean.

$$x_j^s = \frac{x_j - mean(x_j)}{max(x_j) - min(x_j)}$$

Table 3: Data Scale Corrected With Min-Max Normalisation

x_1^s	x_2^s	x_3^s	x_4^s	y
0	0	0.7	1	13.1
0.24	1	0.3	0	17.8
0.12	0.04	0.3	0	10.3
0.69	0.83	0	1	23.0
1	0.08	1	1	30.8

A third approach is called Standardization, or z-score.

$$x_j^s = \frac{x_j - mean(x_j)}{std(x_j}$$

3.306 Polynomial regression

If a linear regression doesn't fit the data very well, we can try increasing the number of θ terms to try to better fit the data.

3.402 Further reading

- Chapter 2, section 2.4 of the course textbook (Chollet)

Week 7

Key Concepts

- Explain how regularisation works.
- Explain the concept of cross-validation.
- Explain the effect of overfitting.

4.101 Overfitting and underfitting

A regression that's too simple to fit the data is said to *underfit* the data, or has a High Bias. An example of which is shown in figure 11.

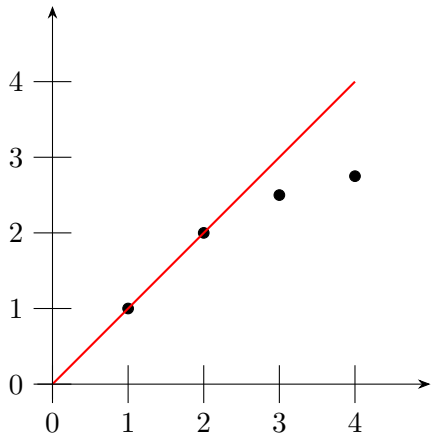

Figure 11: Underfit

A regression with a high degree that fits the data too perfectly is said to overfit the data[1], or has a high variance.

We can evaluate a model with the Bias-variance curve, as shown in figure 12. At the left side of the graph, we have underfitting (high bias), at the right side we have overfitting (high variance). We want to find a model that sits in the middle of the Bias-variance curve.

To ensure a model is generalisable, we can employ a number of techniques:

1. Reduce the number of features

[1]Not drawing that

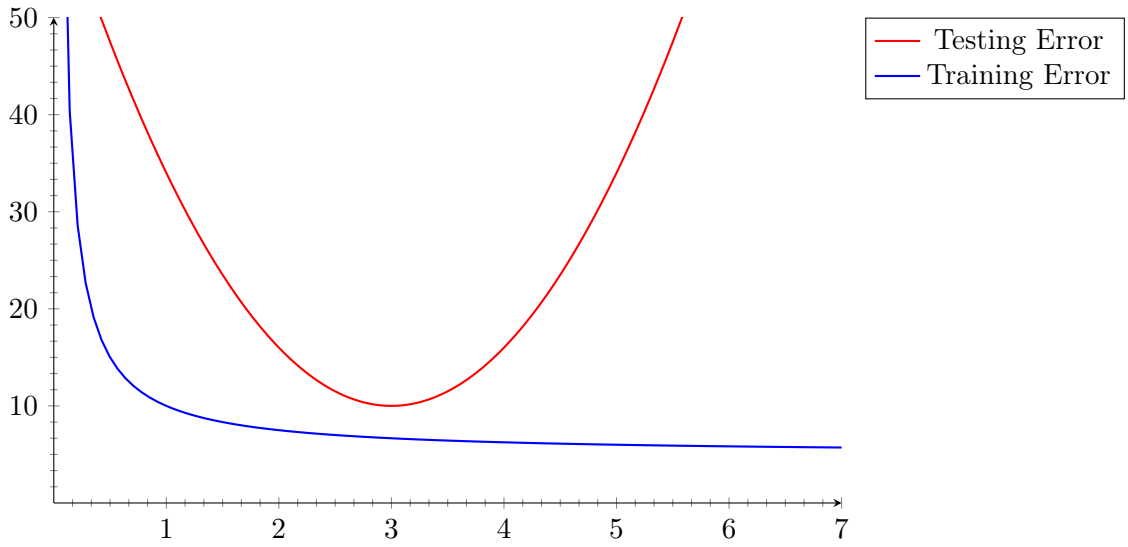

Figure 12: Bias-variance Curve

- Manually select which features to keep
- Use model selection algorithm (e.g. cross-validation)

2. Regularisation
   - Keep all features, but reduce the values of θ_j
   - Works well when we have a lot of features

4.201 Regularisation

Regularisation is a method of penalizing complexity in a Machine Learning Model. There are several regularisation methods, one of them, called, L2 regularisation, involves squaring and summing up all the θ parameters. In other words

$$\sum \theta^2$$

computes the *penalty* in question, the goal being reducing the penalty.

Alternatively, L1 regularisation

$$\sum |\theta|$$

involves adding up the absolute values of the θ parameters. We will rely on L2 for the time being.

Looking back at our linear regression Loss function

$$J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

to regularize this loss function, we add a new regularisation term, thus

$$J(\theta) = \frac{1}{2m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)^2 + \lambda\sum_{j=1}^{n}\theta_j^2$$

Note that the regularisation hyperparamter λ must be tuned:

λ **too big** results in **underfitting**

λ **too small** results in **overfitting**

4.301 Cross-validation

In order to avoid overfitting, it's important that the data used to train the algorithm is **not** the same the data used to test the algorithm. In practice, from our input data we create two disjoint sets called *Training Data* and *Test Data*; one dedicated to training the model and the other dedicated to evaluating the performance of the model.

Usually, we don't have enough data to carry out this process, an approach that maximizes the use of our daata is called *N-fold Cross-Validation*. In summary, we will run the process of splitting the data into test and training sets, train and evaluate the model multiple times, an example of this is depicted in figure 13.

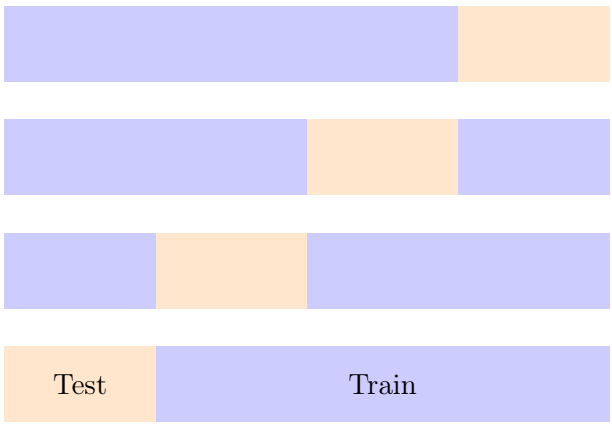

Figure 13: N-fold Cross Valication

The total error in this case is the sum of each of the errors for each of the N splits. In other words

$$e_{total} = \frac{1}{n}\sum_{i=1}^{n} e_i$$

is the total error.

Week 8

Key Concepts

- Discuss the difference between generative and discriminative models
- Describe the Naive Bayes classifier
- Explain Bayes' rule

5.101 Bayesian classification

Bayesian modelling is centered around the concept that new evidence can change decisions. The basic components of a base model are:

Prior initial degree of belief in some preposition

Posterior degree of belief after seeing some evidence

Ultimately, we want to calculate the posterior given evidence. This is, however, difficult to compute directly. We can use an indirect approach using a generative model of the likelihood that some outcome leads to a particular observation. Bayes Theorem helps in this case.

Bayes Theorem is given by:

$$Posterior = \frac{Likelihood \cdot Prior}{MarginalProbability}$$

Some probability rules:

Inverse $P(\bar{A}) = 1 - P(A)$

Conditional $P(B \mid A)$

Product Rule $P(B, A) = P(B \mid A)P(A) = P(A \mid B)P(B)$

Sum Rule $P(B) = P(B, A) + P(B, \bar{A}) = P(B \mid A)P(A) + P(B \mid \bar{A})P(\bar{A})$

Bayes Theorem $P(B \mid A) = \frac{P(A \mid B)P(B)}{P(A)}$

5.103 The Naive Bayes Classifier

Given by:

$$P(Y \mid X_1, X_2, \ldots, X_F)\alpha P(Y) \cdot \prod_{i=1}^{F} P(X_i \mid Y)$$

This results in very small numbers which are susceptible to underflow. A solution is use a logarithm scale:

$$\log(P(Y \mid X_1, X_2, \ldots, X_F))\alpha \log(P(Y)) + \sum_{i=1}^{F} \log(P(X_i \mid Y))$$

The final formulation for the Naive Bayes Classifier is:

$$NB = argmax_Y(\log(P(Y)) + \sum_{i=1}^{F} \log(P(X_i \mid Y)))$$

5.202 Further reading

- Probabilisitic modelling is covered in sections 3.1 and 3.2 of Ethem Alpaydin's book: Alpaydin, E. Introduction to machine learning. (Cambridge, MA: MIT Press, 2014) 3rd edition [ISBN 9780262028189].

Week 9

Key Concepts

- explain the concepts of clustering and dimensionality reduction
- implement the k-means algorithm
- explain principal component analysis (PCA) and its properties.

6.102 Clustering

With Unsupervised Learning, the goal is to learn patterns from the data without having any labels assigned. In applications of clustering, the goal is to separate our data points into groups based on some sort of similarit index. The K-Means algorithm is one implementation of this basic concept.

Given figure 14 below, we want separate the points into disjoint sets similarly to he one shown in 15 that follows.

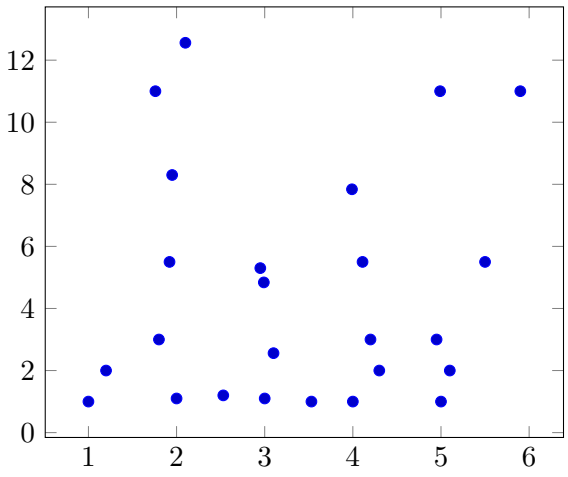

Figure 14: Scatter plot

6.103 K-Mean

The K-Means algorithm works similarly to the steps below:

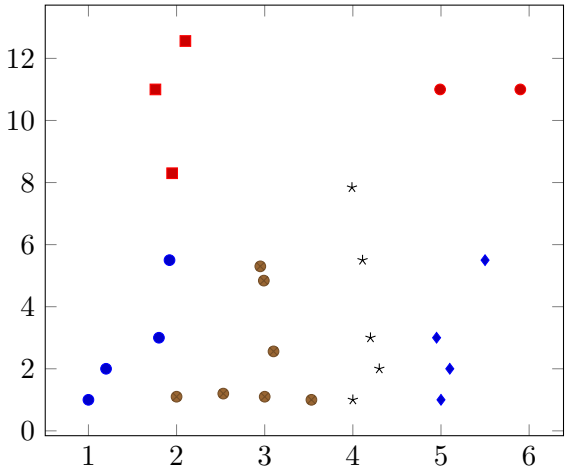

Figure 15: Scatter plot groupped

1. Initialize k centroids randomly
2. Calculate the distance from every data point to each of the k centroids
3. Assign data points to the nearest k_i centroid
4. Re-assign centroids as the mean of each cluster's data
5. Repeat 2–4 until convergence

Scaling the data with a min-max scaling algorithm is usually required to avoid certain phenomena that depend on scale of each of the axis. Min-max is given by

$$x = \frac{x - min(x)}{max(x) - min(x)}$$

which will make sure that every axis falls within the interval $[0, 1]$.

Week 10

Key Concepts

- explain the concepts of clustering and dimensionality reduction
- implement the k-means algorithm
- explain principal component analysis (PCA) and its properties.

6.301 Dimensionality reduction

Dimensionality reduction is particularly useful when dealing with images, because they take a lot of space. Principal Component Analysis is a common algorithm for dimensionality reduction. In summary, PCA is the process of computing the main components of the input data and working with the first few of them, rather than the full data input.

6.302 PCA

Given data X with F features and N samples

$$\mathbf{X} = \left[x^{(1)}, x^{(2)}, \ldots, x^{(N)}\right],$$

where $\mathbf{X} \in \mathbb{R}^{F \times N}$, the sample mean is then

$$\bar{x} = \frac{1}{N} \sum_{i=1}^{N} x^{(i)},$$

where $x^{(i)} \in \mathbb{R}^{F \times 1}$. The covariance is

$$cov(\mathbf{X}) = \mathbf{S} = \frac{1}{N} \sum_{i=1}^{N} \left(x^{(i)} - \bar{x}\right)\left(x^{(i)} - \bar{x}\right)^{\intercal}$$

The basic idea of PCA is that we're trying to linearly transform a set of data from one dimension into another.

$$\mathbf{Y} = \mathbf{W}^{\intercal}\mathbf{X}^{2}$$

For example, if we're mapping from 3 dimensions to 2 dimensions, could train our algorithm to produce the matrix

$$\mathbf{W} = \begin{bmatrix} 1 & 0 \\ 0 & 2 \\ 1 & 0 \end{bmatrix}$$

and the data

$$\mathbf{X} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

, then

$$\begin{aligned} \mathbf{Y} &= \mathbf{W}^\mathsf{T}\mathbf{X} \\ &= \begin{bmatrix} 1 & 0 & 1 \\ 0 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \\ &= \begin{bmatrix} 1 \cdot 1 + 0 \cdot 2 + 1 \cdot 3 \\ 0 \cdot 1 + 2 \cdot 2 + 0 \cdot 3 \end{bmatrix} \\ &= \begin{bmatrix} 4 \\ 4 \end{bmatrix} \end{aligned}$$

We have the relationship that

$$cov(\mathbf{Y}) = cov(\mathbf{W}^\mathsf{T}\mathbf{X}) = \mathbf{W}^\mathsf{T} cov(\mathbf{X})\mathbf{W}$$

We can let $\mathbf{S} = cov(\mathbf{X})$ which simplifies our notation to $cov(\mathbf{Y}) = \mathbf{W}^\mathsf{T}\mathbf{S}\mathbf{W}$. The goal of our model to train PCA is the maximize the covariance of $\mathbf{Y}$, in order words

$$\arg\max_{\mathbf{W}} \left(\mathbf{W}^\mathsf{T}\mathbf{S}\mathbf{W}\right)$$

is our goal, such that $\mathbf{W}^\mathsf{T}\mathbf{W} = \mathbf{I}$. One one to solve this problem is to use *eigenanalysis*, which will give us the optimal values for $\mathbf{W}$.

$$[\mathbf{W}, \mathbf{\Lambda}] = eig(\mathbf{S}),$$

where $\mathbf{S} = cov(\mathbf{X})$.

6.402 Further reading

- Sections 6.1 and 6.3 from Chapter 6 of Ethem Alpaydin's book discuss dimensionality reduction and PCA. Sections 7.1 to 7.3 from Chapter 7 discuss clustering (including k-means): Alpaydin, E. Introduction to machine learning. (Cambridge, MA: MIT Press, 2014) 3rd edition [ISBN 9780262028189].

Week 11

Key Concepts

- Understand the relationships between deep learning, neural networks and artificial intelligence
- Describe multi-layer neural networks, backpropagation and deep networks
- Talk about the history and assess the future of deep learning
- Explain machine learning workflow

7.001 This weeks reading: Chollet, Chapter 1 - What is deep learning?

- Chollet, F. Deep Learning with Python, 1st edn (Manning Publications, 2017). Chapter 1.1 What is deep learning?

7.101 Artificial Intelligence, Machine Learning and Deep Learning

Artificial Intelligence is mechanical thought. AI has come a long way. In the 1950s–1980s, AI was focussed on manipulation of symbols such as rearranging and simplifying equations algebraically. The idea works well for tightly defined tasks but crumbles under the pressure of looser tasks such as image classification and speech recognition.

The traditional method for solving problems computationally, is by encoding the problem in *rules*. The computer will, then, follow these rules to come up with a solution.

In the 1990s a new paradigm emerged: rules are to be learned (or approximated) not encoded in the program. The machine is trained by comparing input with desired outcome, a parameterized map from input to output is gradually tweaked until and acceptable approximation is reached. This is what we refer to as *Machine Learning*.

In the 2010s, a *supercharged* form of Machine Learning came into fruition employing Neural Networks as a form of learning. We call this *Deep Learning*.

7.103 Learning representations

In order to learn a representation, our Machine Learning models require three distinct inputs:

Input data Some training data for the algorithm to learn

Output labels expected labels for each sample from the Input data

Cost function some sort of error measure, or distance from expected label and model's output.

Our model is training by progressively tweaking the parameters of the model in order to get closer to the expected outcomes. For example, say we have a set of features x and y containing coordinates to points. Each (x, y) pair is labeled *red* or *black*. The goal is to learn a mapping from coordinates to a *red* or *black* label.

The input data may look like so:

x	y	color
-0.001	0.0032	red
0.01	-0.123	black
0.8432	0.843	black
-0.0135	-0.455	red

The machine seeks a transformation

$$x' = w_{11}x + w_{12}y$$
$$y' = w_{21}x + w_{22}y$$

Such that

$$x' > 0 \implies black$$
$$x' \geq 0 \implies red$$

The w_{ij} components are the model parameters, or weights. The task is to find weights which minimize the error of the predicted labels *red* or *black*. The learning process is kicked off with random weights, for each iteration of the learning process the weights are tweaked and the error in the prediction always informs the next iteration.

Learning is the automatic search for meaning representations. Possible transformations include translations, rotations, linear projections, and non-linear operations.

7.105 The deep in deep learning

In the previous example of *red* and *black* vectors, the data was processed with a linear transformation and a non-linear projection.

Each transformation is performed by a computational *Layer*. The number of layers in a model is its *Depth*. Models with three or more layers are called *Deep*. Therefore, *Deep Learning* models are simply models with or more layers of transformations. Deep models are almost always Neural Networks.

If we want to train a Neural Network to recognize handwritten digits. Each digit sample is composed of a single 28×28 pixels image. Before training, each sample is vectorized, i.e. we convert the 28×28 square into a 784×1 vector which will be the input to the neural network. This is fed to the first layer of the network which transforms the data into a 512 element vector which is subsequently fed into the second layer. The output from the second layer is 10 element vector containing floating point values that encode the probability that the digit is one of 0–9, see figure 16 for a simple representation of the network.

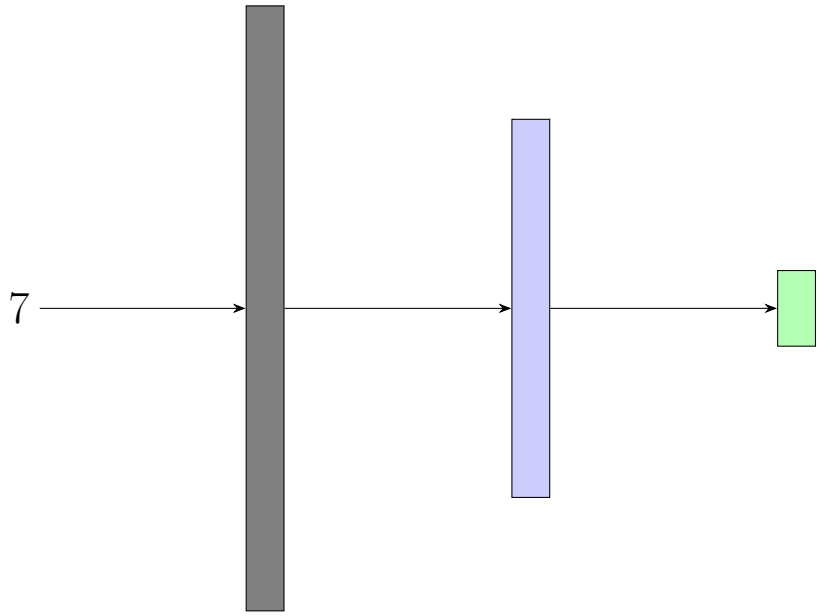

Figure 16: Handwritten Digits Neural Network

7.107 Understanding how DL works in three figures

The transformation of a given layer is controlled by a number of parameters: weights and biases. Each of these parameters can be thought of as sliders which can be moved and tweaked until we're happy with the result.

Neural Networks are a sequence of simple transformations parameterized by weights and biases mapping input **samples** to output **predictions**.

The performance of the model is evaluated by a *Loss Function* that compares prediction to expected value. Weights and Biases are initially set at random and the loss is

potentially very high. An *Optimizer* makes small changes to the parameters in order to reduce the loss.

As we iterate over the training data, the loss slowly drops and the network becomes a better predictor. The optimizer implements an algorithm known as *Back propagation*. After training the network with this process, we test the performance of the model using unseen data. See figure 17 for a depiction of this fact.

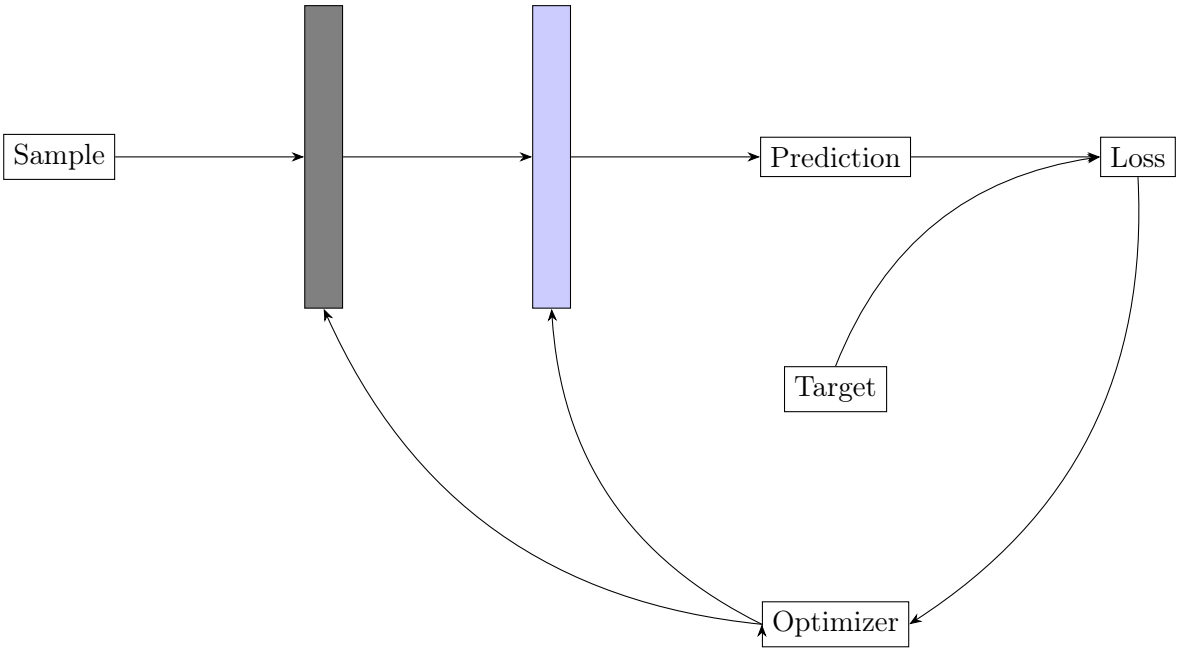

Figure 17: Neural Network Diagram

7.109 Achievements. Short-term hype. The promise of AI.

Deep Learning can produce results at near-human performance in many complex tasks:

- Image Classification
- Speech Recognition
- Handwriting Transcription
- Autonomous Driving

Others tasks are improved:

- Machine Translation

- Text-to-speech
- Ad Targetting
- Search Results

There have been some major breakthroughs:

- Digital Assistance
- Go and Chess
- Medical Diagnosis

And some tasks remain out of reach:

- Believable Dialogue
- Human-level translation accross several languages
- Complete natural language understanding
- General Intelligence

Summary of the History of AI:

1960s Symbolic AI

1970s AI Winter

1980s Expert Systems

1990s Second AI Winter

2010s Deep Learning

There are reasons to doubt a third winter:

- Many DL developments await deployment
- Continued diffusion into everyday life even if research stalls
- Many new applications: Assisting human scientists, climatology, drug discovery, and many more.

Week 12

Key Concepts

- Understand the MNIST dataset
- Understand how a simple neural network is built and trained with Tensorflow
- Understand neural network data representations

8.100 This weeks reading - Chollet, Chapter 2.1 and 2.2

- Read Chollet (2017), Chapters 2.1 and 2.2.

8.101 MNIST. Loading MNIST in Keras. Network architecture.

The MINIST dataset consists of 70,000 small grayscale images of handwritten digits. There are 60,000 **training** samples and 10,000 **test** samples.

Our workflow consists of a few steps:

1. Load data
2. Preprocess data
3. Build network
4. Train
5. Test

To load MNIST in Keras, we can run the following python code:

```
from tensorflow.keras.datasets import mnist
(train_images, train_labels),(test_images, test_labels) = mnist.load_data()
```

The data is stored in special multidimensional arrays called *Tensors*.

8.103 Compilation (pre-process)

The input data has to be *reshaped* to a format that's accepted by the neural network. Networks (usually) accept vectors — i.e. 1D arrays — of floating point values, therefore we have to convert the image representation into a long 784×1 array.

In the example that follows, we can see how to inspect the first element from the MNIST dataset.

```
import matplotlib.pyplot as plt

image = test_images[0]
label = test_labels[0]

print("sample:")

plt.imshow(image, cmap=plt.cm.binary)
plt.show()

print("label:", label)
```

Preprocessing the data has three steps:

1. Flatten 2D pixel maps `reshape()`
2. Convert to floating points
3. Rescale to $[0, 1]$

All three steps are shown below with relevant comments:

```
# Flatten 2D pixel maps
train_images = train_images.reshape((60000, 28 * 28))
test_images = test_images.reshape((10000, 28 * 28))

# Cast to floating points
train_images = train_images.astype('float32')
test_images = test_images.astype('float32')

# Rescale to [0, 1]
train_images = train_images / 255
test_images = test_images / 255
```

The network also expects categorically encoded labels, that is a vector where a single element is non-zero (sometimes referred to as *one-hot encoding*). This can be achieved with the code below.

```
from tensorflow.keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

8.105 Building the network

When it comes to building our network, we start by importing `models` and `layers` from `keras`.

```
from tensorflow.keras import models, layers
```

Then, we create an empty network.

```
network = models.Sequential()
```

Next, we add two layers to the network.

```
# Hidden layer
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))

# Output layer
network.add(layers.Dense(10, activation='softmax'))
```

With the `softmax` activation on the output layer, we get a vector of non-negative floats within the interval $[0, 1]$ which encode the probability that the sample is each of the 10 digits.

The network architecture is complete. In order to move on to training, we must define a Loss Function, an optimizer, and more training metrics.

The Loss Functions quantifies the error of the prediction, the optimizer adjusts parameters during the training loop, and metrics report on progress.

```
network.compile(optimizer='rmsprop',
                loss='categorical_crossentropy',
                metrics=['accuracy'])
```

8.107 Train network

Looking back at figure 17, we can match our chosen *pieces* to the diagram. Now we must decide on the mini-batch size — number of samples processed in a single pass of the algorithm —, and the number of epochs — the number of complete passes through the entire training set.

```
network.train(train_images, train_labels, epochs=5, batch_size=128)
```

After training, the network has to be tested.

8.109 Test network

To evaluate our trained network, we must provide some unseen data and compute the resulting accuracy.

```
test_loss, test_acc = network.evaluate(test_images, test_labels)
```

We can also request for the prediction of a single sample.

```
network.predict(test_images[:1])
```

The network prediction is the maximum of this prediction vector.

```
import numpy as np
np.argmax(network.predict(test_images[:1]))
```

Week 13

Key Concepts

- Understand tensors operations

8.300 This weeks reading - Chollet, Chapter 2.3

- Read Chollet (2017), Chapter 2.3.

8.301 Tensor operations

There are several tensor operations, some of which are:

- Slicing
- Arithmetic
- Broadcasting
- Element-wise operations
- Tensor dot

A layer performs

$$f(w \cdot x + b),$$

mapping rank 2 tensors to rank 2 tensors, where w is the tensor of weight parameters, b is the tensor of bias parameters and x is the input tensor, $\cdot$ operation is similar to matrix multiplication, $+$ is the element-wise addition, and $f()$ is the activation function (relu, sigmoid, etc).

8.303 Element-wise operations

Any element-wise operation works on pairs of elements as shown below:

$$\begin{pmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{pmatrix} \star \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} = \begin{pmatrix} a_1 & \star & b_1 \\ a_2 & \star & b_2 \\ \vdots & \star & \vdots \\ a_n & \star & b_n \end{pmatrix}$$

where $\star$ can be any regular arithmetic operation. For example:

$$\begin{pmatrix}3\\4\end{pmatrix} + \begin{pmatrix}8\\9\end{pmatrix} = \begin{pmatrix}3 & + & 8\\4 & + & 9\end{pmatrix} = \begin{pmatrix}11\\13\end{pmatrix}$$

This same idea also applies for element-wise function application, as shown below:

$$f\left(\begin{pmatrix}a_1\\a_2\\\vdots\\a_n\end{pmatrix}\right) = \begin{pmatrix}f(a_1)\\f(a_2)\\\vdots\\f(a_n)\end{pmatrix}$$

8.305 Broadcasting

Broadcasting enables element-wise operations on tensors of different shapes. It's achieved by duplicating a tensor along a given broadcasting axis in order to match the shapes of both tensors. Then, the operation can be carried out. In general, tensor shapes are compared right to left, dimensions are compatible if they are equal or one of them is 1. The axis with dimension 1 is copied to match the other.

8.307 Tensor dot

The dot operation is the tensor analogue of the dot product. For example, we can compute a dot product between two vectors:

$$\begin{pmatrix}2 & 3\end{pmatrix} \cdot \begin{pmatrix}-1\\2\end{pmatrix} = 2 \times -1 + 3 \times 2 = 4$$

Between a matrix and a vector:

$$\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix} \cdot \begin{pmatrix}-1\\2\end{pmatrix} = \begin{pmatrix}2 \times -1 + 3 \times 2\\4 \times -1 + 5 \times 2\end{pmatrix} = \begin{pmatrix}4\\6\end{pmatrix}$$

Between two matrices:

$$\begin{pmatrix}2 & 3\\4 & 5\end{pmatrix} \cdot \begin{pmatrix}-1 & 2\\2 & -3\end{pmatrix} = \begin{pmatrix}2 \times -1 + 3 \times 2 & 2 \times 2 + 3 \times -3\\4 \times -1 + 5 \times 2 & 4 \times 2 + 5 \times -3\end{pmatrix} = \begin{pmatrix}4 & -5\\6 & -7\end{pmatrix}$$

We can generalize the value of every i^{th} element in the tensor dot of w and x as

$$(w \cdot x)_i = \sum_{j=0}^{m} \sum_{i=0}^{n} w_{ij} x_j,$$

therefore the layer transformation $relu(w \cdot x + b)$ is simply the composition of a few elementary tensor operations.

8.309 Tensor reshaping

Tensors can be reshaped to fit different needs. For example, if we have the tensor

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix},$$

we can reshape it to a 6×1 column vector, which results in

$$\begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{pmatrix}.$$

We can also transpose a tensor, essentially turning rows into columns. For example, given the tensor

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix},$$

its transpose is

$$\begin{pmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{pmatrix}.$$

8.311 Geometric interpretation of tensor operations

Numerical tensors can be interpreted in as coordinates in a (often high-dimensional) real space. Say we have a tensor t holding four 2D points

$$\begin{pmatrix} 1 & 0 \\ 1 & 1 \\ -1 & 0 \\ -1 & -1 \end{pmatrix},$$

we can plot this a plane as shown in figure 18 below.

The operation $w \cdot x$ is a linear transformation. The operation $w \cdot x + b$ is an affine transformation, i.e. a linear transformation followed by a translation. *Relu* is a non-linear transformation, this means that the resulting plot will have distortions. This is easy to conclude when we consider that *relu* will clamp any negative values to 0.

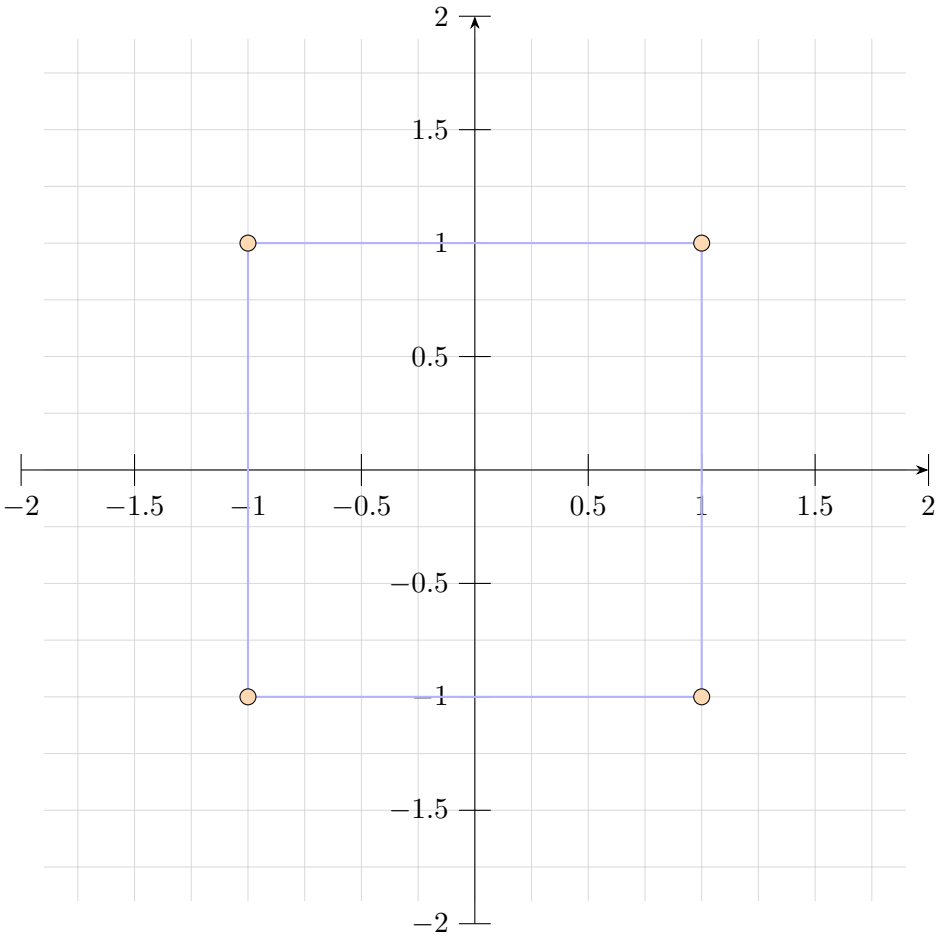

Figure 18: Tensor of 4 2D points

Week 14

Key Concepts

- Understand gradient-based optimisation

8.400 This weeks reading - Chollet, Chapter 2.4, 2.5 and 2.6

- Read Chollet (2017), Chapter 2.4, 2.5 and 2.6.

8.401 Gradient based optimisation

To find the optimal values for w and b for a given problem, we employ the use of Gradient Descent, which is a first-order iterative optimization method for finding a local minimum of a function that can be differentiated.

The training loop of a model consists of a few steps:

1. Randomly initialize weights and biases
2. Draw mini-batch of samples and labels
3. Networks makes a prediction `y_pred` (forward propagation)
4. Loss computation
5. Update parameters to lower loss
6. Exit or go to step 1

Step 6 above guarantees the termination of the training loop. Steps 1–4 are simple tensor operations. While updating the model's parameters, it's wise to do so towards a direction we know will decrease the loss. This is achieved by moving in a direction oposite to the gradient at current point in the function. This is where the Gradient Descent Algorithm comes into play.

8.403 What's a derivative?

Let f be a smooth function, i.e. without abrupt changes. In the vicinity of a point x, $f(x)$ can be approximated as

$$f(x + \delta x) \approx f(x) + m\delta x,$$

where δx is an infinitesimally small step and m is the gradient at point x.

Tensor flow computes gradients automatically with `tf.GradientTape()`

8.405 Derivative of a tensor operation

Differentiation Rules

f	$\frac{df}{dx}$	Rule
x^n	nx^{n-1}	Power
$g(x) + h(x)$	$\frac{dg}{dx} + \frac{dh}{dx}$	Linearity
$af(x)$	$a\frac{df}{dx}$	Linearity
$f(g(x))$	$\frac{df}{dg}\frac{dg}{dx}$	Chain

When differentiating multivariate functions, we differentiate with respect to each variable. These are called partial derivatives and are written with a different notation

$$f(x, y)' = \frac{\partial f}{\partial x} + \frac{\partial f}{\partial y}.$$

This is not different from regular differentiation, any variable not being differentiated is treated as a constant. All other differentiation rules still apply.

8.407 Stochastic gradient descent

The gradient tensor of f is written ∇f. The minimum loss is achieved when w and b are solutions to $\nabla f = 0$. The computation required is far too expensive, therefore we employ an iterative approach where get closer and closer to the solution in small steps.

The w and b parameters are updated using the gradient tensor with respected to w and b scaled by a value η known as the learning rate, as shown below.

$$w = w - \eta \nabla_w f$$
$$b = b - \eta \nabla_b f$$

When we compute gradients using randomly selected subsets of the training set, this is referred to as **Stochastic Gradient Descent**.

8.409 Backpropagation

Backpropagation updates the weights in each of the layers with a backwards pass through the model. This is because the gradients of Layer n depends on the gradients of layer $n + 1$.

Week 15

Key Concepts

- Understand the anatomy of a neural network

9.100 This weeks reading - Chollet, Chapters 3.1, 3.2 and 3.3.

- Read Chollet (2017), Chapter 3.1, 3.2 and 3.3.

9.103 Layers

The *Layer* is the fundamental building block of a neural network. They are responsible for mapping tensors to other tensors and are parameterized by weights and biases which update the SGD (Stochastic Gradient Descent) feedback loop.

In simple terms, a **Sample** is fed through the layers of a network during a **forward pass**, data is transformed by the layers and an **output** emerges.

A *Dense Layer* – i.e. fully connected layer – has a $n \times m$ weight matrix. The incoming vector – i.e. the input sample – has length m while the output vector has length m.

A *Recurrent Neural Network* is one which contains internal loops.

A *Convolutional Neural Network* slides a small window accross a 2D image using a process called *Convolution*. The response is mapped to layer weights.

Table 4 summarizes which Neural Network is likely to work well for which application:

Table 4: Summary of Neural Networks

Data	**Network**
Vector	Dense
Sequence	Recurrent
Image	Convolutional

A layer performs a tensor operation on input x, namely an affine transformation followed by an element-wise function application, i.e.

$$z = \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1m} \\ w_{21} & w_{22} & \cdots & w_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ w_{n1} & w_{m2} & \cdots & w_{nm} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}.$$

An affine transformation includes rotations, reflections, shears, and dilations. Straight lines are preserved and parallel lines remain parallel.

By themselves, affine transforms are insufficient for data uncrumpling. A non-linear component is required, the activation function.

$$f(z) = f\left(\begin{pmatrix} z_1 \\ z_2 \\ \vdots \\ z_m \end{pmatrix}\right) = \begin{pmatrix} f(z_1) \\ f(z_2) \\ \vdots \\ f(z_m) \end{pmatrix}$$

Common activiation functions are:

Sigmoid $\text{sigmoid}(x) = \frac{1}{1+e^{-z}}$

Rectified Linear $\text{relu}(x) = \max(0, x)$

Hyperbolic Tangent $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

9.105 Models: networks of layers

A feedforward network is a directed, acyclic graph. While a linear stack of layers is common, there are also two-branch layers, multihead layers, etc.

The network's topology and the layer functions define a space of possible transformations.

The transformation

$$y_l = f_l(w_l \cdot x_l + b_l) := L_l(x_l)$$

depends on the choice of activation f_l and the parameters values (w_l, b_l).

9.107 Loss functions and optimizers: keys to configuring the learning process

Neural networks have arbitrary architecture and activation function. The loss function and optimizer is also arbitrary. Hyperparameters are tuned, not trained.

Table 5 summarizes the common loss functions for a given application.

The optimizer is always a variant of SGD. Bespoke gradient-free methods are also possible.

Table 5: Common Loss Functions

Application	**Loss Function**
Many-class classification	Categorical Cross-Entropy
Regression	Mean Square Error

Week 16

Key Concepts

- Apply neural networks to binary classification tasks

9.200 This weeks reading - Chollet, Chapter 3.4

- Read Chollet (2017), Chapter 3.4.

9.201 The IMDB dataset

The International Movie Database problem is an example of binary classification. The dataset is composed of 50,000 textual movie reviews half of which are labeled positive and the other half labeled negative.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import imdb

((train_data, train_labels),
 (test_data, test_labels)) = imdb.load_data(num_words = 10000)

word_index = imdb.get_word_index()
reverse_word_index = dict([(value,key) for (key,value)
                           in word_index.items()])
decoded_review = ' '.join([reverse_word_index.get(i - 3, '?')
                           for i in train_data[100]])
```

9.202 Preparing the data

The network requires tensor inputs, not lists. We can either pad lists to have equal length, for integer tensors and use an `Embedding` layer or encode the input to *one-hot* representation and feed it into a `Dense` layer.

```
import numpy as np

def vectorize_seqs(seqs, dim = 10000):
```

```
    results = np.zeros((len(seqs), dim))
    for i, seq in enumerate(seqs):
        results[i, seq] = 1
    return results

x_train = vectorize_seqs(train_data)
x_test = vectorize_seqs(test_data)

y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

9.204 Building your model

Our model is shown in listing below.

```
from tensorflow.keras import models
from tensorflow.keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activatation = 'relu', input_shape = (10000,)))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))
```

We choose RMSprop as our optimizer and binary cross-entry as our loss function. The cross-entry loss function is the sum of

$$f_{sample} = -y \log y_{pred} - (1 - y) log(1 - y_{pred})$$

over the minibatch. The freely available book by Michael Nielsen Neural Networks And Deep Learning provides a great overview of why cross-entropy loss is used and how does it counteract learning slowdown in chapter 3.

We can compile the model with the listing shown below:

```
model.compile(optimizer = 'rmsprop',
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])
```

9.206 Validating your approach

A validation dataset are required for tuning hyperparameters.

```
model.fit(x_train, y_train,
          epochs = 4,
          batch_size = 512)
results = model.evaluate(x_test, y_test)
results
```

9.208 Prediction with a pretrained network

```
model.predict(x_test[:10])
```

Week 17

Key Concepts

- Apply neural networks to multi-class classification tasks

9.300 This weeks reading - Chollet, Chapter 3.5

- Read Chollet (2017), Chapter 3.5.

9.301 The Reuters dataset

The Reuters dataset contains several short textual reports, each fit into one of 46 possible classes. This is a single-label, multiclass classification problem.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.datasets import reuters

(train_data, train_labels),
(test_data, test_labels) = routers.load_data(num_words=10000)
```

9.302 Preparing the data

To preprocess the data, we follow the same recipe from IMDB example.

```
import numpy as np

def vectorize_seqs(seqs, dimension=10000):
    results = np.zeros((len(seqs), dimension))
    for i, seq in enumerate(seqs):
        results[i, seq] = 1.
    return results

x_train = vectorize_seqs(train_data)
x_test = vectorize_seqs(test_data)

y_train = np.asarray(train_labels).astype('float32')
y_test = np.asarray(test_labels).astype('float32')
```

And encode the labels in one-hot encoding.

```
from tensorflow.keras.utils import to_categorical

one_hot_train_labels = to_categorical(train_labels)
one_hot_test_labels = to_categorical(test_labels)
```

9.304 Building your network

There are 46 possible classes, which makes this problem more complex than IMDB example. Instead of 16 units for the hidden layers as in IMDB, we will try with 64 units.

We require that the network outputs a 46-dimensional vector encoding the probability of each class. A softmax activation

$$f(x_i) = \frac{e^{x_i}}{\sum_i e^{x_i}}$$

produces an output that we can interpret as a probability distribution.

```
from tensorflow.keras import models
from tensorflow.keras import layers

def softmax(x):
    return np.exp(x) / np.sum(np.exp(x))

model = model.Sequential()
model.add(layers.Dense(64, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(46, activation='softmax'))
```

We can get the shape of weight and bias tensors for each layer with the code below:

```
def print_layer_tensor_shape(layer):
    params = model.layers[layer].get_weights()
    weights = params[0]
    bias = params[1]

    print(layer, '\t', weights.shape, '\t', bias.shape)

print_layer_tensor_shape(0)
print_layer_tensor_shape(1)
print_layer_tensor_shape(2)
```

Compiling the model

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

9.305 Validating your approach

```
history = model.fit(x_train,
                    one_hot_train_labels,
                    epochs=7,
                    batch_size=512)

results = model.evaluate(x_test, one_hot_test_labels)
```

9.307 Generating predictions on new data

```
model.predict(x_test)
```

Week 18

Key Concepts

- Apply neural networks to regression tasks

9.400 This weeks reading - Chollet, Chapter 3.6

- Read Chollet (2017), Chapter 3.6.

9.401 The Boston house price dataset

The IMDB and Reuters dataset are examples of discrete problems, i.e. the model must choose a particular class from a finite set of options. Regression problems, however, are contiuous; that is, there are inifitely many options to choose from.

The Boston Housing dataset is an example of a continuous problem because houses may have any value. Our goal is to predict house prices given property tax, crime rate, rooms per dwelling, accessibility to nearby highways, and 9 other features.

There are 506 total samples, split into 404 training samples and 102 test samples. The scale of each of the features vary wildly.

```
1 from tensorflow.keras.datasets import boston_housing
2
3 (train_data, train_targets),
4 (test_data, test_targets) = boston_housing.load_data()
```

Because the scales on the features vary, the data must be *normalized*, otherwise we will run difficulties when training the model. Normalization is the process to make sure the scale of the features is the same.

9.402 Preparing the data

We normalize by shifting and rescaling to a mean of 0 and standard deviation of 1. Test data is normalized using the mean and standard deviation of the training set.

In essence we compute

$$x_n = \frac{x_n - \mu_{train}}{\sigma_{train}},$$

where x_n is one training sample, μ_{train} is the mean of the traning set, and σ_{train} is the standard deviation of training set.

```
mean = train_data.mean(axis = 0)
train_data -= mean
std = train_data.std(axis = 0)
train_data /= std

test_data -= mean
test_data /= std
```

9.404 Building your network

Because the dataset is so small, we try to avoid overfitting with a small network consisting of 2 hidden layers only.

```
from tensorflow.keras import models
from tensorflow.keras import layers

def build_model():
    model = models.Sequential()

    model.add(layers.Dense(64,
                           activation='relu',
                           input_shape=(train_data.shape[1],)))
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(1))
    model.compile(optimizer='rmsprop',
                  loss='mse',
                  metrics=['mae'])

    return model
```

`mse` is the Mean-Squared-Error, defined as

$$MSE = \frac{1}{N}\sum(y_{pred} - y)^2,$$

where y is the target, y_{pred} is the network output and N is the mini-batch size.

`mae` is the Mean-Absolute-Error, defined as

$$MAE = \frac{1}{N}\sum|y_{pred} - y|,$$

where y is the target, y_{pred} is the network output and N is the mini-batch size.

9.405 Validating - k-fold

```
import numpy as np

```

```
K = 4
num_val_samples = len(train_data) // K
num_epocs = 100
all_scores = []

for i in range(K):
    print("Processing fold ", i)

    a, b = i * num_val_samples, (i+1) * num_val_samples
    val_data = train_data[a:b]
    val_targets = train_targets[a:b]

    partial_train_data = np.concatenate([train_data[:a],
                                         train_data[b:]],
                                        axis = 0)
    partial_train_targets = np.concatenate([train_targets[:a],
                                            train_targets[b:]],
                                           axis = 0)

    model = build_model()

    model.fit(partial_train_data,
              partial_train_targets,
              epochs=num_epochs,
              batch_size=16,
              verbose=0)

    val_mse, val_mae = model.evaluate(val_data,
                                      val_targets,
                                      verbose=0)

    all_scores.append(val_mae)
```

Week 19

Key Concepts

- Know how and when to preprocess data
- Know when a neural network is underfitting or overfitting
- Know how to address overfitting with network capacity reduction, weight regularisation and dropout

10.100 This weeks reading - Chollet, Chapters 4.1, 4.2, 4.3 and 4.4.

- Read Chollet (2017), Chapter 4.1, 4.2, 4.3 and 4.4.

10.200 Training, validation and test sets

Machine Learning models are **not** evaluated on the training set because our end goal is that of **generalization**, i.e. we want the model to perform well on new, **unseen** data.

As we validate our model with our validation set, the model absorbs information from the validation set and tends to overfit for the seen data.

10.202 Simple hold-out validation

Split part of the training set and use exclusively for validation. This technique is simple and easy to implement, but if suffers from statistical fluctuation if the validation and test set are small.

10.204 k-fold validation

Conceptually very straight forward: split the training set into k equal partitions, then iterate over each of the k blocks, train with all but one of the blocks, and validate with the block that was held out. It's essentially several iterations of hold-out validation.

10.206 Things to keep in mind

Training and test data should be representative of the population. That is to say that if we have a dataset containing images of dogs, cats, and parrots, both training and test sets should have a similar proportion of all three animals. This is easy to guarantee with data shuffling, however time-sensitive data (e.g. weather and stock prediction) shouldn't be shuffled.

Care must be taken to guarantee that training and test sets are disjoint.

10.300 Data preprocessing for neural networks

- All inputs must be floating point tensors
- Data should be normalized
- Gradient Descent works best if data homogenous and constrained
- Missing data could be filled with zeroes
  - If test data has missing data, don't touch it. Duplicate some training data and randomly add zeroes so the network learns to ignore missing values

10.302 Feature engineering

Feature Engineering is data preprocessing in the light of human knowledge. Before Deep Learning, Feature Enginering with critical. Deep Learning networks are less dependent on FE.

10.401 Overfitting and underfitting

There is a balance between optimization (of training data) and generalization (to test data). The best solution to overfitting is to find more training data, however more data is often hard to find. Another solution is regularization:

- Network size reduction
- Weight regularization
- Drop-out

10.403 Reducing the network's size

Network capacity is the number of weights and bias parameters. A low capacity model is likely to underfit training data while a high capacity model risks overfitting to the training model, which would cause it to fail generalization.

We can't predict the optimal number of layers in advance, therefore it's wise to start with a low capacity model and increase capacity until improvement gain flattens.

10.405 Adding weight regularisation

A model with smaller weights, i.e. weights within a small interval around zero, is more robust.

L1 Regularization cost proportional to the absolute value of the weight parameters is added to the loss function

L2 Regularization like L1, but the cost is the square of the weight parameters

10.407 Adding Dropout

A fraction of the output from a drop-out layer is zeroed, this fraction is referred as the drop-out rate.