AINT252

Gabryel Mason-Williams

February 2019

Abstract

This document contains my lectures notes based on the series of lectures in the AINT252 module given by lectures at The University of Plymouth

Contents

1 Introduction Lecture

This lecture covered a quick introduction to the module about AI ending with a formal definition of a simple neural network and types of learning. This lecture was presented by Prof Roman Borisyuk and some of the information is directly copied from the lecture slides.

1.1 Introduction to AI

These days, AI is becoming a more and more popular approach for solving many different real-life problems. Since the time of the first computers (1950^{th}), AI has been an active scientific discipline.

The field of AI draws upon many different interdisciplinary fields such as; computer science, mathematics, information theory, engineering, psychology, cognitive science and many more.

AI is a challenging goal in which we want computer code (an agent) to mimic/ have natural intelligence, i.e. a human brain. Alan Turing in his paper "Computing Machinery and Intelligence. Mind 49:433-460 (1050) formulated a question: Can a machine think?. Turing suggested a way to test this would be to use the "imitation game." [1]

1.2 Natural AI

A simple but attractive idea for designing AI is to look at the human brain as this is the most complex and only thing we know of as General Intelligence which is ultimately our goal with AI, because of this a lot of our models are based on our understanding of how the brain works, i.e. the Biological Neuron.

1.2.1 Biological Neuron

A neuron consists of the cell body (soma), input (dendrite) and an output (axon). To communicate between neurons is via the synapse, and the synaptic weight defines the efficacy of this transmission [2]. A neuronal network is a set of interconnected neurons, and a neuron integrates all the inputs signals and compares the inputs with the threshold: if it exceeds the threshold, then the neuron generates an action potential.

We try and mimic this structure in neural networks as you will see in the next section.

[1] If you do not want to read the whole paper this overview is very detailed and give the gist of the paper (link)

[2] This is also known as the connection strength and is adjustable

1.3 Formal Neural Network

A basic Neural Network can be written like so, and an example of what this network would look like is shown in figure **??** [3]

- Inputs: $x_1, x_2, .., x_n$; binary (0/1)
- Connection Strengths: $w_1, w_2, ..., w_n$
- Summation $h = \sum_{i=1}^{n} w_i x_i$
- Output $y = f(h)$; binary: if $h > T$ then $y = 0$ else $y = 1$

Where $f(h)$ is the activation function with a threshold of T

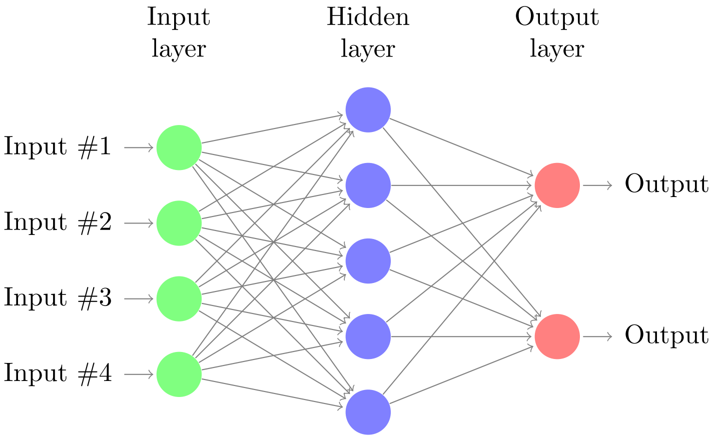


Figure 1: Neural network:4-5-2

Neural Network is a set of coupled neurons Similar to real neurons; formal neurons are interconnected: the output of one neuron is an input to another neuron. The figure above demonstrates this.

1.4 Learning

Similar to a real neural network, the Artificial Neural Network (ANN) can learn by adjusting the connection strength. The current most popular types of learning in AI: supervised, unsupervised and reinforcement learning

For example, Deep Learning (Deep Neural Network) is a method to learn from the data; Deep Learning can use supervised learning: for a particular input vector, a ”teacher” defines a desirable output. Thus, the deep neural network can be trained to find a correspondence between inputs and desirable outputs.

[3] There is a series of lectures on youtube from 3Blue1Brown covering this topic in great detail (link)

2 Theoretical basis of AI

2.1 AI algorithms for classification

From a theoretical point of view many AI algorithms are formulated in terms of classification, i.e. a pattern recognition problem can be formulated as a classification problem.

A data set consists from cases (objects) and each case is characterised by some features, each case can be represented by a vector of features such that $\overrightarrow{x} = (x_1, x_2, ..., x_p)$ each vector can be considered as a dot in p-dimensional space and all cases together form a cloud. If we Assume that each vector belongs to some class, the number of classes is M. The goal is to find a classifier that assigns to each case a label of some class.

2.1.1 What a computer can see

An image classifier takes a single image and assigns probabilities to labels. For an image of size 248×400 pixels and three colour channels, the image will consist of 297,600 numbers where each number is an an integer in a range from 0 to 255, our task if turn these numbers into a label.

2.2 Data pre-processing

There are four common forms of data pre-processing: centering, normalisation, dimensionality reduction, noise cancelling. To understand these procedure we assume that our data has a random nature and we use probability theory and statistics to deal with random variable and data analysis.

A Data matrix X, where we will assume that X is of size $[n, p]$ (where n is the number of data (cases), p is the dimensionality of the feature vector).

2.3 Normal distribution

A continuous random variable X which is bell-shaped and has mean (expectation) μ and standard deviation σ is said to follow a Normal Distribution with parameters μ and σ.

In shorthand, $X \sim \mathcal{N}(\mu,\, \sigma^2)$. This may be given in *standardised* form by using the transformation

$$z = \frac{z - \mu}{\sigma} \to x = \sigma z + \mu, \text{ where } Z \sim \mathcal{N}(0,\, 1)\,. \tag{1}$$

The normal distribution is important because of the Central Limit Theorem which states that a sum of arbitrary random variables will have a distribution that is approximately normal.

2.4 Bivariate normal distribution

The multivariate normal distribution is a generalisation of the univariate normal to two or more variables, Each element of a normally distributed random vector has a univariate normal distribution. In the simplest case there is no correlation among variables, and elements of the vectors are independent univariate normal random variables.

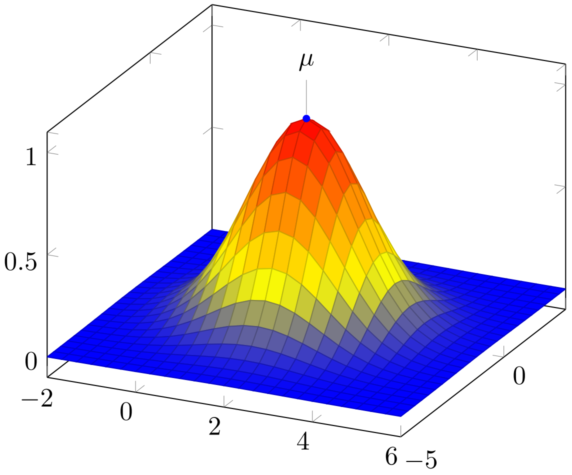


Figure 2: Bivariate normal distribution 3D perceptive

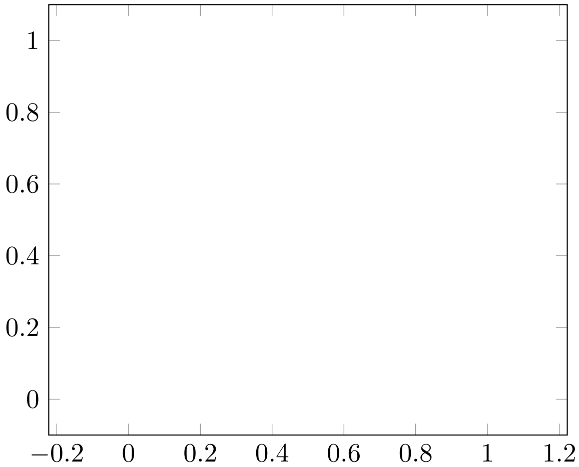


Figure 3: Bivariate normal distribution 2D perceptive

p is the correlation $-1 \leq p \leq 1$

- $p = 0$ means that the varables are independent

- $p = +1$ means that the variables are linearly dependent
- $p = -1$ means also the linear dependence but the increase of one variable is associated with the decrease of an other.

2.5 Statistics: Random sample

Statistics is an interface between the probability theory and data, it provides multiple procedures for data analysis. In statistics the data set is a sample, this sample should be chosen in a proper way and correctly represent a random variable which we would like to estimate.

2.6 Parameter estimators

Data analysis is based of statistical estimator of the random variable X, i.e. for sample $X = (x_1, x_2, ..., x_n)$,

The estimator of the mean is

$$\bar{X} = \frac{1}{n}\sum_{i=1}^{n} x_i. \tag{2}$$

The estimator of the variance is

$$S_x^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{X})^2. \tag{3}$$

The estimator of the standard deviation is

$$S_x = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{X})^2} \tag{4}$$

Now consider the bivariate random variable $(X, Y) = \{(x_i, y_i)\}, i = 1, 2, ..., n$

The estimator of the covariance is

$$cov(X, Y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{X})(y_i - \bar{Y}) \tag{5}$$

The estimator of the correlation is

$$p = \frac{cov(X, Y)}{S_x S_y} \tag{6}$$

2.7 Data Centering

Mean subtraction is the most common form of pre-processing, it involves subtracting μ across every individual feature in the data and has the geometric interpretation of centering the cloud of data around the origin along every dimension.

2.8 Data normalisation

Normalisation referees to normalising the data dimensions so that they are approximately the same scale. There are two common ways of achieving this normalisation

1. To divide each dimension by its standard deviation, once it has been zero centered
2. normalised each dimension so that the min and max along the dimension is -1 and 1 respectively

It only makes sense to apply this pre-processing if you have a reason to believe that different input features have different scales, but thy should be approximately equal importance to the learning algorithm. In the cases of images the relative scales of pixels are already approximately equal, so it is not strictly necessary to perform this addition pre-processing step.

2.9 Dimensionality reduction: Principal Component Analysis (PCA)

Principal components analysis is a quantitatively rigorous method for achieving data simplification and dimensionality reduction. The method generates a new set of variables, called principal components. Each principal component is a linear combination of the original variables. All the principal components are orthogonal to each other, so there is no redundant information. The principal components as a whole form an orthogonal basis for the space of the data.

1. The first principal component is a single axis in space. When you project each observation on that axis, the resulting values form a new variable. The variance of this variable is the maximum among all possible choices of the first axis.
2. The second principal component is another axis in space, perpendicular to the first. Projecting the observations on this axis generates another new variable. The variance of this variable is the maximum among all possible choices of this second axis.
3. The full set of principal components is as large as the original set of variables. But it is a common place for the sum of the variances of the first few principal components to exceed 80% of the total variance of the original data.

2.10 Classify images

The problem is very simple for natural intelligence, but very challenging for the AI. Recent Google developments of Convolutional Neural Networks (CNN) allow to the solve this problem and classify images with a small amount of error

2.11 Binary (two classes) classification

If we simplify the image problem down to dots on a 2-dimensional plane and would like to classify them, the idea would be to find the separator between the two classes the simplest separator is a straight line.

2.12 Linear Regression

Regression is the task of predicting real valued quantities. For this type of task, it is common to compute the error function between the predicted quantity and the true answer, the error function includes both data and parameters. Minimization of the error function provides optimal parameter values.

2.12.1 Least squares

Linear regression line is the line of best fit: the sum of squared distances from the data point to the line is the smallest (least squares).

This is niavie and simple approach however it shows a typical approach to the construction of an AI algorithm.

- Define an error function, which includes data and parameters
- Find min/max of this function according to the parameter variation (a training data set is given and fixed)
- Optimization procedure provides the optimal parameter values which define the best classifier for the given training set.

The solution of this optimization problem (i.e. the parameter values corresponding to the global minimum or some point nearby) is the best set of parameter values for the classifier

2.13 Numerical optimization

There two main numerical methods to solve the optimization problem

2.13.1 Gradient descent

The vector gradient is defined as

$$\nabla f = \Big(\frac{\delta f}{\delta x}, \frac{\delta f}{\delta y}\Big)^T \tag{7}$$

Gradient descent uses just the gradient of the function to navigate to the minimum. From calculus, the direction of ∇ f is that of maximum change of the function. The gradient descent uses - ∇ f (the down-gradient direction) as an estimate of the direction of the minimum

2.13.1.1 Multivariable calculus: gradient

The gradient is a vector having direction equal to the greatest rate of increase, and magnitude equal to the slope of the function in that direction.

- The Jacobian Matrix is defined for scalar-valued functions (functions with a single output), whereas for vector-valued functions it is necessary to use the Jacobian matrix.

2.13.2 Higher dimensional Newton-Raphson

In cases where the first derivative $f'(x)$ is well behaved in the region of the root, we can used the Newton-Raphson method to compute the root.

$$X_{n+1} = X_n - J^{-1}(X_n)F(X_n) \tag{8}$$

2.13.2.1 Example

Consider finding the roots of the two functions

$$f(x, y) = 9 - x^4 - y^4$$

$$g(x, y) = x^2 - 2y + 2$$

Starting at

$$x_0 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}$$

The Jacobian is

$$\begin{pmatrix} \frac{\delta f}{\delta x} & \frac{\delta f}{\delta y} \\ \frac{\delta g}{\delta x} & \frac{\delta g}{\delta y} \end{pmatrix} = \begin{pmatrix} -4x^3 & -4y^3 \\ 2x & -2 \end{pmatrix}$$

The first step of the 2D Newton-Raphson method is:

$$X_1 = X_0 - J^{-1}(X_0)F(X_0)$$

$$\begin{pmatrix} x_1 \\ y_1 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} - \frac{1}{8} \begin{pmatrix} -1 & 2 \\ -1 & -2 \end{pmatrix} \begin{pmatrix} 7 \\ 1 \end{pmatrix} = \begin{pmatrix} 1.625 \\ 2.125 \end{pmatrix}$$

3 Clustering

3.1 Data Clustering

Cluster analysis is a way of categorizing a collection of objects in groups. Suppose we have a collection of objects and each object is described by some number of features. The features may be physical measurements, subjective ratings, or indications of the presence or absence of features. We want to organise the objects into groups so that those object within each group are relatively similar and those in different groups are relatively dissimilar.

We hope that the groups will be 'natural' and 'compelling', and will reveal some structure actually presented in the data. Group members will share certain properties in common and it is hoped that the resultant classification will help to understand general features of objects based on group membership

3.2 Hierarchical Clustering

The working example for this algorithm in this document will be

- Objects: Several houses in a town
- Features: coordinates of a house on a town map
- Goal: Produce the hierarchical cluster using these two features (coordinate variables)

In Hierarchical clustering we need to work out how we are going to define 'similar' and 'dissimilar'. What dissimilarity measure (distance among objects) should be used? Using the example above we can simply measure the distance between two houses on a map.

A clustering algorithm measures distances between all pairs of houses and selects two of the closest houses. This will be the first cluster, we then select the next house which is nearest to the cluster, to do this we need to choose the linkage between clusters and houses. There are many different linkages we could use however the most popular linkages are

- Nearest-neighbour(distance between closest objects from two clusters)
- Furthest-neighbour (distance between furthest objects)
- Average distance (average of distances between objects from two classes)

3.2.1 Linkage

3.2.1.1 Nearest-neighbour

This is a single linkage clustering or minimum of distance method, the dissimilarity between 2 clusters is the minimum dissimilarity between members of the two clusters. This method produces long chains which form loose, straggly clusters.

3.2.1.2 Furthest-neighbour

This is a complete linkage clustering method, the dissimilarity between 2 groups is equal to the greatest dissimilarity between members of the two clusters. This method tends to produces very tight clusters of similar cases.

3.2.1.3 Average linkage

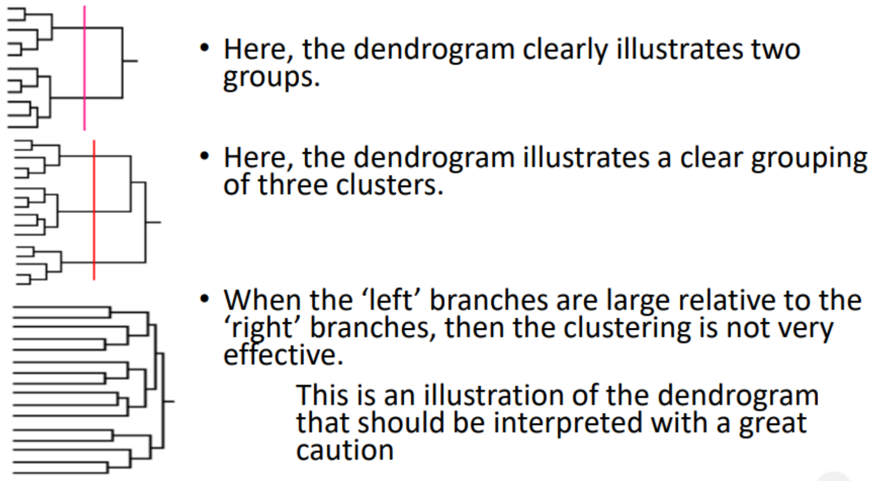
The dissimilarity between clusters is calculated using average values, there are many ways of calculating an average! The most common one is between groups average method (the average distance is calculated from the distance between each point in a cluster and all other points in another cluster)

3.2.2 Dendrogram

Eventually, more and more object will be linked together and larger clusters of increasingly dissimilar elements will appear, Once all object are joined together, the results of a cluster analysis is often represented by a dendrogram.

3.2.2.1 Where should we cut a tree

A dendrogram that clearly differentiates groups of objects will have small 'left' branches of the tree (in the beginning of the clustering) and large 'right' branches. You should cut a tree when the gaps between the clusters is the largest.



- Here, the dendrogram clearly illustrates two groups.
- Here, the dendrogram illustrates a clear grouping of three clusters.
- When the 'left' branches are large relative to the 'right' branches, then the clustering is not very effective.

  This is an illustration of the dendrogram that should be interpreted with a great caution

Figure 4: Interpreting a dendrogram

Interpretation of the clustering depends on the linkage method selected.

3.2.3 Similarity

There are many ways of measuring similarity between clusters the most common way is using Euclidean distance

3.2.3.1 Euclidean Distance

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{9}$$

Euclidean distance is the standard measure for clustering.

3.2.3.2 Squared Euclidean Distance

$$d(x, y) = \sum_{i=1}^{n} (x_i - y_i)^2 \tag{10}$$

This helps to get dense clusters, if we were to plot the cluster in a multidimensional space, we would like them to be relatively dense aggregations of points, with small distance between points within clusters and big distances between clusters

3.2.3.3 Cosine similarity

$$sim(x, y) = \cos(\theta) = \frac{x \cdot y}{\|x\|\|y\|} \tag{11}$$

The more similar object the closer the points, Cosine is maximal for similar objects.

3.2.3.4 Manhattan Block

$$d(x, y) = \|x - y\| = \sum_{i=1}^{n} \|x_i - y_i\| \tag{12}$$

The geometry has been used in regression analysis since the 18th century, and today is often referred to as LASSO.

3.3 Cluster Analysis

Possible problems with Euclidean and squared Euclidean distances is different scales for different variables, one way to get around this problem is to standardise the variables using the Z-score.

Cluster analysis methods will always produce a grouping the groupings produceds by cluster analysis may ore may not be useful for classifying objects. There are many options for cluster analysis therefore we can mine the data trying different methods combining different distances and linkages to discover the structure in the data.

There are many example of successful application of cluster analysis to real problems a famous example, PRIZM system: In 1970s the company Claritas clustered neighbourhoods (U.S. zip codes) into forty different groups based on census information, such as population density, income, age, etc. This classification scheme, call PRIZM (Potential Rating Index for Zip Markets), has proven to be very useful in direct mail advertising, radio station formats, and decisions about where to locate stores.

3.4 K-means: Non-hierarchical clustering

This method differs from hierarchical clustering in many ways, the most obvious ways being

- There is no hierarchy, the data is partitioned instead
- You have to supply the number of clusters (k) into which you want to data to be grouped

This method is conceptually simple but computationally intensive

- The cases are initially assigned randomly to the k clusters
- Cases are then moved around between clusters using an iterative method so that a classification is produced such that the clusters must be internally similar but externally dissimilar to the other clusters
- We stop when moving any more cases between clusters would make the clusters become more variable.

Cluster variability is measured with respect to their means for the classifying variables, hence the name k-means clustering. If more than one variable is used to define the clusters then the distances between clusters are measured in multidimensional space

K-means is a relatively simple procedure, and consists of choosing random k points that represent the distinct centres of the k subsets, which are called centroids.

- We then select, for each centroid, all the points closest to it
- This will create k different subsets.
- At this point, for each subset, we will re-calculate the centre
- We have again, k new centroids, and we repeat the steps above, selecting for each centroid, the new subsets of points closest to the centroids.
- We continue this process until the centroids stop moving

For this technique to work, we need to be able to identify a metric that allows us to calculate the distance between points. This summarised as follows.

1. Choose initial k-points, called centroids
2. To each point in the data set, associate the closest centroid
3. Calculate the new centre for the sets of points associated to a particular centroid
4. Define the new centres to the new centroids
5. Repeat steps 3 and 4 until the centroids stop moving

3.4.1 Silhouette

Clustering results can be represented in a graphical form using silhouettes.

The silhouette value for each point is a measure of how similar that point is to points in its own cluster, when compared to points in other clusters. The silhouette value for the *i th* point, S_i, is defined as

$$S_i = \frac{b_i - a_i}{max(a_i, b_i)} \tag{13}$$

Where a_i, is the average distance from the *i th* point to the other points in the same cluster as i, and b_i is the minimum average distance from the *i th* point to points in a different cluster, minimised over clusters.

The silhouette value ranges from -1 to +1, a high silhouette value for point i indicates that this point is well-matched to its own cluster, and poorly-matched to neighboring clusters. If most points have a high silhouette value, then the clustering solution is appropriate. If many points have a low or negative silhouette value, then the clustering solution may have either too many or too few clusters. The silhouette clustering evaluation criterion can be used with any distance metric. The mean silhouette value can be used to estimate a quality of clustering.

3.4.2 What to bear in mind

It is important to notice that this method is extremely sensitive to the initial choice of random centroids, and that it may be a good idea to repeat the process for different initial choices. Also it is possible for some of the centroids not to be closest to any of the points ion the data-set reducing the number of subsets down from k.

3.5 K Nearest Neighbour (KNN) Classifier

This is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The KNN algorithm is among the simplest of all machine learning algorithms. The training examples are vectors in a multidimensional feature space each with a class label. The training phase of the algorithm consists only of storing the feature vectors and class labels of the training samples. In the classification phase, k is a user-defined constant, and an unlabelled vector (a query or test point) is classified by assigning the label which is most frequent among the k training samples nearest to that query point.

3.5.1 Algorithm summary

- A positive integer k is specified, along with a new sample
- We select the k entries in our database which are closest to the new sample
- We find the most common classification of these entries
- This is the classification we give the new sample

3.5.2 Features of KNN

KNN stores the entire training data-set which it uses as its representation, KNN does not learn any model and it makes predictions just-in-time by calculating the similarity between an input sample and each training instance.

3.5.3 Pros and cons of KNN

Pros

- No assumptions about data — useful, for example, for nonlinear data
- Simple algorithm — to explain and understand/interpret
- High accuracy (relatively) — it is pretty high but not competitive in comparison to better supervised learning models

Cons

- Computationally expensive —because the algorithm stores all of the training data

- High memory requirement
- Stores all (or almost all) of the training data
- Testing stage might be slow for large N
- Sensitive to irrelevant features and the scale of the data

4 Supervised Learning

A Neural networks are AI computational systems that are 'inspired' by biological neural systems. Their main feature is the abiblity to learn from the expericnece with using examples of classification.

A single neuron is a linear classifier, classification can be thought as a way of separating the input data.

4.1 Perceptron

This is a machine learning algorithm that helps provide classified outcomes for computing. It dates back to the 1950s and represents a fundamental example of how machine learning algorithms work to develop data structure. This algorithms was invented by Frank Rosenblatt, it has an input and an output layer, a binary activation function. Update rule based on Hebbian learning: connections that are active at the same time are strengthen.

The basic of the algorithm is four parts.

- Setup: Weights are randomly assigned typically between -0.5 and 0.5
- Inputs: $x_1, x_2, .., x_n$; binary (0/1)
- Connection Strengths: $w_1, w_2, ..., w_n$ and bias θ
- Summation $h = \sum_{i=1}^{n} w_i x_i$
- Output $y = f(h - \theta)$; binary: if $h > T$ then $y = 0$ else $y = 1$

A Perceptron is a linear classification. Therefore the linear separator between two classes on a 2d plane is $x_2 = kx_1 + b$ where $k = \frac{w_1}{w_2}$ as x_2 can be written as $x_2 = -\frac{w_1}{w_2} x_1 + \frac{\theta}{w_2}$

4.1.1 Learning rule: training perceptron

In a general case, to adjust weights w and the bias θ of the perceptron, the learning (training) procedure is used. The training set includes many pairs, the input vector x and the targeted output $t(x, t)$ We send one input vector x to the perceptron adn the the actual output is O. The learning rule to adjust weights and bias is based on the difference (error) between the actual output O and the targeted value $t : error = t - O$

- Error: $e = t - O$
- $W_{new} = W_{old} + qx(t - O)$
- $\theta_{new}\theta_{old} + q(t - O)$

- Where $0 < q < 1$is the learning rate

This is also know as the Hebbian learning algorithms and mathematical equation is as follows.

$$W_{ij}[n+1] = W_{ij}[n] + \eta x_i[n]x_j[n] \tag{14}$$

Where η is the learning rate coefficient and x are the outputs of the ith and jth elements.

4.1.2 Example

Lets examine a simple example to help see how a perceptron can learn to represent the logical-OR function for two inputs, using a threshold of zero $t = 0$ and a learning rate η of 0.2.

- First we need set the associated weights which for this example will be $w_1 = -0.2$ and $w_2 = 0.4$
- Now, the first epoch is run through, the training data will consist of the four combinations of 1's and 0's possible with two inputs. Hence, our first piece of training data is $x_1 = 0$ and $x_2 = 0$ and our expected outcome is 0
- So we apply the formula $h = \sum_{i=1}^{n} w_i x_i$
- resulting in $h = 0$
- since h is as expected, and the error, e, is therefore 0, the weights do not change.
- Now $x_1 = 0$ and $x_2 = 1$ and our expected outcome is 1
- Apply the formula and we get $h = 0.4$
- So the output is 1, so the weights don't need to change
- Now $x_1 = 1$ and $x_2 = 0$ and our expected outcome is 1
- Apply the formula and we get $h = -0.2$
- So the output is 0, so the weights need to change as the expected output was 1
- So we apply the perceptron training rule to assign new values to the weights, using the predefined η and this case e is 1.
- $w_1 = -0.2(0.2 \cdot 1 \cdot 1)$, $w_1 = 0$
- $w_2 = 0.4(0.2 \cdot 0 \cdot 1)$, $w_1 = 0.4$ as this did not contribute to this error it is not adjusted.
- Now $x_1 = 1$ and $x_2 = 1$ and our expected outcome is 1
- Apply the formula and we get $h = 0.4$

- So the output is 1, so the weights don't need to change
- This is the end of the first epoch, and at this point the method runs again and continues to repeat until all four piece of training data are classified correctly.

4.1.3 Limitations

The exclusive OR function cannot be modeled using a perceptron. This is because the perceptrons can only learn to model functions that are linearly separable.

4.2 Multilayer Perceptron

An artificial neural network (ANN) consists of a set of interconnected nodes. ANN is able to receive inputs from the environment and to learn new behaviours (responses)

4.2.1 ANN: forward pass

A ANN: forward pass is similar to a perceptron, the unit output is given by the weighted sum of the inputs, passed through an activation function.

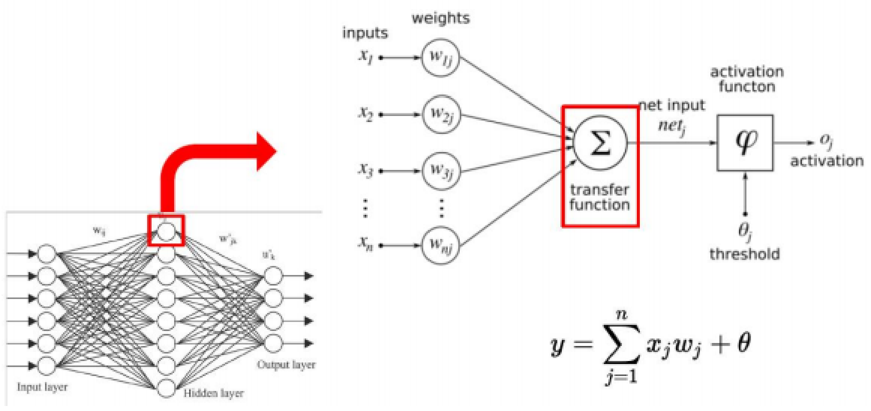


Figure 5: Hidden node

4.2.2 ANN: backward pass (error estimation)

The performance of the network is evaluated using a loss function which must be differentiable. Generally an L2 norm is used the output is then compared with the expected value. The expected value is given by the label associated with the input and is stored in the data set.

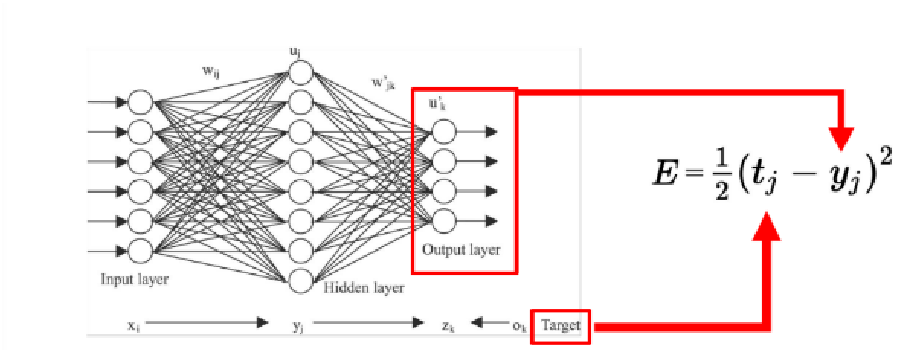


Figure 6: Output node

4.2.3 ANN: backward pass (update rule)

The standard approach for training an ANN is back-propagation this is based on gradient descent, at each iteration the weights are adjusted in order to obtain an output that is more similar to the target. It is important to select the right hyper-parameters (e.g. learning rate).

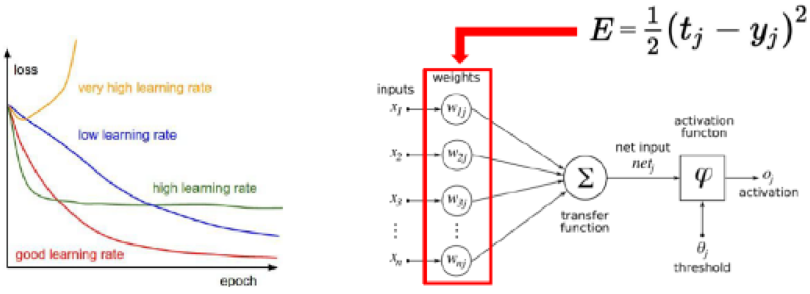


Figure 7: Update weights

4.2.4 Activation functions

An activation function of a node defines the output of that node. It maps the resulting values into the desired range such as between 0 to 1 or -1 to 1 etc. (depending upon the choice of activation function). For example, the use of the logistic activation function would map all inputs in the real number domain into the range of 0 to 1, depending upon the choice of activation function.

4.2.4.1 Sigmoid activation function

This is a mathematical function that has the characteristic of an "S"-shaped curve. Often the sigmoid function refers to a special case of the logistic function.

The definition of a sigmoid functions is as follows.

A sigmoid function is bounded, differential, real function that is defined for all real input values and has a non-negative derivative at each point.

In general, a sigmoid function is monotonic, and has a first derivative which is bell shaped. A sigmoid function is constrained by a pair of horizontal asymptotes as $x \to \pm\infty$

4.2.5 ANN with hidden layers

The training data (input array and target output) should be used to adjust parameters of ANN classifier,the learning procedure is based on minimization of the total (for all training data) error function which depends on training data and many parameters (weights and biases. Minimization is based on the idea of the gradient descend method. To minimize computational time, the idea of backpropagation is used: moving backward (from output to input though several hidden layers) pick up already calculated components of the gradient and use them to adjust ANN parameters.

4.3 Support Vector Machine Binary Classifier

A support vector machine constructs a hyper-plane or set of hyper planes in high or infinite-dimensional space known as the decision boundary in (2 dimensional space this is a line), the distance between hyper-planes is know as the margin. The goal is to find the decision boundary with maximal margin! A good hyperplane is equidistant from both classes. A support Vector are data points that the margin pushes up against.

4.3.0.1 SVM as Nonlinear classifier's

SMV can be applied to data of complex non linear structure, for this type of classification data transformation (kernel) is used, the SMV can be used to classify data to several classes, for that a set of binary SVM is combined to produce the best solution for a multi class problem.

5 Problem Solving and Game Playing

Problem solving is the study of AI algorithms that try and find the sequence of actions that lead to problem solution. There are several benchmark problems/games

- Travelling Salesperson Problem
- Missionaries and cannibals
- Crypto-arithmetic
- Games: Hanabi ,Starcraft2, Go, Chess and Tic-Tac-Toe
- Robot navigation

Some of these problems have been define as complex problems our have a layer of information hiding so the agent does not know all of the information.

Key terms

- Search space: The initail set of all possible conditions that satisfy the problem constraints
- Problem definition: A search tree which illustrates the initial, intermediate and goal states
- Search Strategy: A way to decide the next state to expand

5.1 Graphs and Search Trees

A graph is made of a set of nodes connected by links that may be directed or undirected, A successor is a neighbouring node that can be reached via a link, with a path being a sequence of nodes that connect two nodes together via links. A acyclic graph is a graph where no node has a path linking to itself, no cycles. Finally a tree is a type of graph where each node is connected by only one path.

5.2 Problem types

There are two main problem types Single-state problem and Multiple-state problem. A Single-state problem is where all action consequences are known, whereas, a Multiple-state problem is when the environment is not fully accessible and only partial states are known to the agent.

5.2.1 To solve problems

To solve problems you need a set of initial conditions that will help define what a optimal and valid solution to the problem is.

- Initial state and State space: Collection of all possible world states

- Operators: actions that lead to state change
- Goal test: A way to identify the goal state(s)
- Path cost: A function that is assigns a cost to a particular path
- Search trees: A Pathway between search states
- Search Strategy: To decide the next state to expand to.

5.3 Search Algorithms

There a two main types of search algorithms Blind search and Heuristic search. Blind search is and uninformed method and used where there is no information about the number of steps of the total path cost, whereas, Heuristic search is informed meaning that it is used when it is possible to know the relative cost of nodes.

5.3.1 Blind Search Algorithms

General Search algorithms are distinguished by the order in which nodes are expanded. A list that certainly is not exhaustive can be seen below.

- Breadth-First Search: expands the shallowest node first
- Depth-First Search: expands the deepest node first
- Depth-Limited Search: puts a limit in the depth-first search
- Uniform Cost Search: expands the least operator-cost node first

5.3.2 Heuristic Search

For general AI a heuristic is any technique that improves the average performance on a problem solving task. A list of heuristic search algorithms that certainly is not exhaustive can be seen below.

- Greedy Search: Best-first algorithm where the heuristic function defines the node cost.
- A* Search: Greedy search and the Uniform cost algorithm.

5.4 Minimax Algorithm

The minimax algorithm determines the optimal strategy for the *Max* player by maximising its utility and assuming the *Min* player will play perfectly to minimise it.

In tic-tac-toe this can be used using the two evaluating functions

$$3X_2 + X_1 - (3O_2 + O_1) \tag{15}$$

Where X_n is the number of rows with n X and no O.

$$Open_{MAX} - Open_{MIN} \tag{16}$$

Where $Open_{MAX}$ is the number of complete rows/columns/diagonals that are still open for MAX.

6 Evolutionary Computation

Evolutionary Computation is the area of computer science interested in optimising solutions to problems that are often NP-hard, by using nature as the inspiration to for the techniques.

6.1 Formal definition

Given an optimisation problem $f(x)$, find a solution x that optimises the function f. When the search space is simple this is not a very hard problem see figure **??**, whereas for search space is often not that nice!

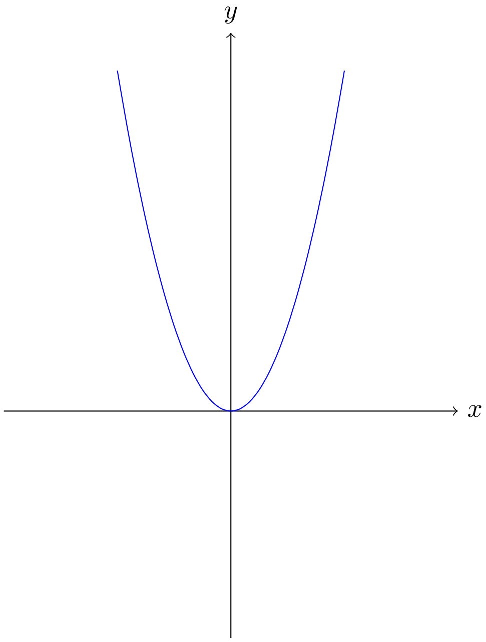


Figure 8: $f(x) = x^2$

To optimise a problem we must consider a few very important things about the solution

- representation
- evaluation of quality
- illegal solutions are not generated
- they are generated from scratch
- modify

- a way to identify the best one

7 Computer Vision

Computer Vision is a field that deals with how computers can be made to gain a high level of understanding from images.

7.1 Image Processing

There are many different ways to process an image as shown below

- Spatial filtering using Difference of Gaussians
  - Low-pass
  - Band-pass
  - High-pass
- Illumination gradients
  - Image differentiation -edges
  - Sobel, Canny filters
- Filtering in time
  - Motion analysis

All of these techniques use convolution

7.1.1 Convolution

Convolution is the a mathematical operations on two functions f and g to produce a third function that expresses how the shape of one is modified by the other.

Definition

The convolution of w and v is written as $w * v$. It is defined as the integral of the product of the two functions are one is reversed and shifted. As such, it is a particular kind of integral transform.[4]

$$(w * v)(x) \triangleq \int_{-\infty}^{\infty} w(\tau)v(x-\tau)d\tau$$

[4] For more information about the definition with a visual explanation (link)

7.1.1.1 Discrete Convolution

Definition

For complex-valued functions, f, g are defined on the set Z, the discrete convolution of f and g is given by

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[m]g[n-m]$$

or equivalently by

$$(f * g)[n] = \sum_{m=-\infty}^{\infty} f[n-m]g[m]$$

So in images this method is used as images are represented in the Z set.[5]

7.1.2 Spatial filtering

Spatial filtering is an technique for changing the intensities of a pixel according to the intensities of the neighboring pixels.

7.1.2.1 Difference of two Gaussians (DoG)

This is a symmetric band-pass filter, which removes high frequency components representing noise, and some low frequency components representing the homogeneous areas of the image. The frequency componets in the passing band are assumed to be associated to the edges in the images.

The first step is to smooth the image by convolution with the Gaussian kernel of certain width σ_1

$$G_{\sigma_1}(x,y) = \frac{1}{\sqrt{2\pi\sigma_1}} e^{-\frac{x^2+y^2}{2{\sigma_1}^2}}$$

to get

$$g_1(x,y) = G_{\sigma_1}(x,y) * f(x,y)$$

with a different width σ_2, a second smoothed image can be obtained

$$g_2(x,y) = G_{\sigma_2}(x,y) * f(x,y)$$

We can now show that difference in these two Gaussian smoothed images can be used to detect edges in an image.

$$g_1(x,y) - g_2(x,y) = G_{\sigma_1} * f(x,y) - G_{\sigma_2} * f(x,y) = (G_{\sigma_1} - G_{\sigma_2} * f(x,y) = DoG * f(x,y).$$

[5] To see a visual example of this working and a walk through (link)

The DoG as an operator is defined as

Definition

$$DoG \triangleq (G_{\sigma_1} - G_{\sigma_2}) = \frac{1}{\sqrt{2\pi}}(\frac{1}{\sigma_1}e^{-\frac{x^2+y^2}{2\sigma_1^2}} - \frac{1}{\sigma_2}e^{-\frac{x^2+y^2}{2\sigma_2^2}})$$

7.1.2.2 Sobel gradient processing

This operator uses two 3 by 3 kernels which are convoled with the original image to calculate approximations of the derivatives. If we define A as the source image, and G_x and G_y are the two images which at each point contain the vertical and horizontal derivative approximations respectively.

$$G_x = \begin{bmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{bmatrix} * A$$

and

$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} * A$$

Since the Sobel kernels ca nbe decomposed as the products of an averaging and differentiation kernel, they compute the gradient with smoothing. At each point in the image the resulting gradient approximations can be combined to give the gradient magnitude.[6]

$$G = \sqrt{G_x^2 + G_y^2}$$

Using this we can calculate the gradients direction

$$\theta = \arctan(\frac{G_y}{G_x})$$

7.2 Pattern Recognition

Pattern Recognition is the art of automated identification of objects.

7.2.1 Scale Invariant Feature Transform (SIFT)

This is a feature detection algorithm that is used to detect and describe local features in images. In this algorithm keypoints of the objects are first extracted to form a set of reference images and stored. An object is then recognised in a new image by individually comparing each feature from the new image to this database and finding a candidate matching the features based on the euclidean distance of their feature vectors. This algorithm composes of 5 steps.

[6] For more information on this filtering technique visit (link)

1. Gaussian Blurring
2. DoG extraction
3. Local Gradient Histogram
4. Extract Maxima, mark as key point
5. Key points showing orientation

7.3 Robot Vision

This is an area of robotics intended to give robots the ability to sense of seeing, thus making them more human like. Many algorithms can be used to achieve these attributes, however, Monte Carlo Localisation (MCL) and Simultaneous Location And Mapping (SLAM) are only discussed in this module.

7.3.1 Monte Carlo Localisation (MCL)

This algorithm is also known as particle filter localisation, and used to help robots localise there position by using a particle filter. Given a map of the environment, the algorithm estimates the position and orientation of the robot. The algorithm starts with a uniform random distribution of particles over the space, Whenever the robot moves, it shifts the particles to predict its new state after the movement. When the robot senses something, the particles are re sampled based on a recursive Bayesian estimation. After a period of time the particles should converge towards the robots actual position. This type of algorithm is computationally expensive because it has to deal with a lot of information.

7.3.2 Simultaneous localization and mapping (SLAM)

SLAM is used when there is an unknown environment and vehicle pose, meaning that a estimate of the robot pose and map of environmental features needs to be generated simultaneously.

8 Formal Languages and Automata

Pattern matching is central to many applications and supported by regular expressions in programming languages. The Chomsky hierarchy of Formal Languages provides a theory of programming languages of increasing expressively. Formal Automata or Machines are models of computation, that implement languages of the Chomsky hierarchy.

8.1 Regular Expressions

A regular expression describes a pattern, it has very basic operators however with these operations it can specify a usually infinite set of strings see table **??**. [7]

Operation	**Regular Expression**	**Yes**	**No**
Concatenation	aabbaab	aabaab	Every Other String
Wildcard	.u.u.u.	cumulus jugulum	succubus tumultuous
Union	aa — baab	aa babb	Every Other String
Closure	ab*a	aa abbba	ab ababa
Parentheses	a(a—b)aab	aaaab abaab	Every Other String
	(ab)*a	a ababababa	ε abbbaa

Table 1: Regex

8.1.1 Generalised Regular Expressions

These have all of the rules previously mentioned however they also have more as seen in the table **??**

[7] RegExp cheat sheet (link)

Operation	**Regular Expression**	**Yes**	**No**
One or more	a(bc)+de	abcde abcbcde	ade bcde
Character classes	[A-Za-z][a-z]*	capitalised Word	camelCase 4illegal
Exactly k	[0-9]{5}-[0-9]{4}	08540-1321 19072-5541	111111111111 166-54-111
Negations	[ˆaeiob]{6}	rhythm	decade

Table 2: Generalised Regex

8.2 Deterministic Finite State Automata (DFA)

A DFA M is a 5-tuple $M = \{Q, \Sigma, q_0, F, \delta\}$ where

- Q is a finite set of states
- Σ is finite set of input symbols (”alphabet”)
- q_0 is a start state from Q
- F is a set of accepting states from Q
- δ is a transition function, i.e. a total mapping from $Q \times \Sigma$ to Q

8.2.1 Transition Function

The transition function δ takes a state q and an input symbol a and maps them to a state q'. The function is 'total' for each pair (q, a) exactly one target state q must exist so $\delta(q, a) = q'$ this function is often written as $q \xrightarrow{a} q'$. δ can be represented by a matrix T in code.

8.2.2 Example of what the symbols mean

Using the figure **??** and calling it M so $M = \{Q, \Sigma, q_0, F, \delta\}$ the symobls represent

- Q is q_0, q_1
- Σ is a,b
- q_0 is q_0
- F is q_0
- $\delta(q_0, a) = q_1$, $\delta(q_0, b) = q_0$, $\delta(q_1, a) = q_1$, $\delta(q_1, b) = q_1$

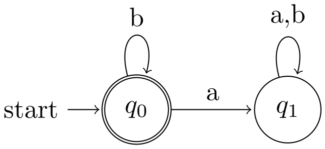


Figure 9: DFA of regex b*

8.2.3 How a DFA Works

A run of a DFA M on $s = a_0, a_1, .., a_{n-1}$ is a sequence of states $q_0, q_1, q_2, .., q_n$ such that $q_i \overset{a}{\rightarrow} q_{i+1} \; \forall \; 0 \leq i \leq n$. The determinism part means for a given input word a DFA has a unique run, because the transition function is a total function. A DFA accepts a word if q_n is in the set of final state F; otherwise it rejects the word.

8.2.4 Informal definition

In figure **??** the DFA has three states q_0,q_1 and q_2 with q_1 being the goal state and q_0 being the start state.

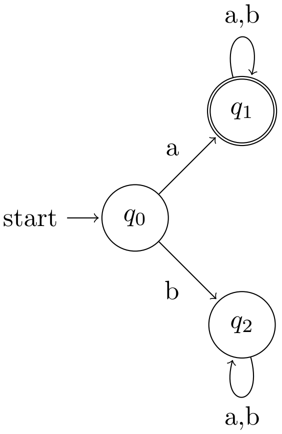


Figure 10: DFA

The DFA in figure **??** reads inputs symbols as a's or b's and transits between states as indicated by the labelled arrows.

In a DFA there can only be one starting state however there can be multiple goal states which is represented as by the double circle. Once an all inputs have been read if the end state is a goal state then the string is correct/accepted otherwise it is wrong/rejected.

8.2.5 Pseudo code

Algorithm 1 DFA Algorithms

```
Result: True or False
T = [1,0;1,1]
q = $q_0$
a = $a_0, a_1, .., a_{n-1}$
while $i \leq N - 1$ do
    q = T[q,a[i]]
end
if $q$ is in $F$ then
    return True
else
    return False
end
```

8.3 Non-Deterministic Finite State Automata (NFA)

In a DFA, each pair of state and input symbol uniquely defines a next state, however, in Non-determinism allows for missing and non-unique transitions (and possibly for more than 1 start state). If multiple targets exist (non-determinism!) the machine is "cloned" and all alternatives run further. There may be "epsilon-transitions" (without an input). If a target state cannot be determined the machine dies A word is accepted if at least one derivation exists, i.e. if at least one machine survives in an accepting state.

8.3.1 NFA example

The NFA figure **??** can have multiple runs with the same word for example *abbb* can end up in either s_1 or s_2. An NFA can have 0,1 or more than one runs on a given word.

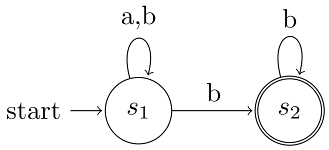


Figure 11: NFA of words ending in b

8.4 Formal Languages

Finite state machines accept or reject words, the set of all words an FSA called A accepts is called the Language it accepts

$$L(A) = \{u \in \Sigma^* \mid A \text{ has an accepting run on } u\} \tag{17}$$

Regular Expressions characterise words as well, thereby they also define Languages.

8.4.1 Equivalence (without proofs)

For each DFA exists a regular expression that defines the same Language as the DFA.For each regular expression exists a DFA that accepts the same language as the RegExp For each NFA exists a DFA that accepts the same language (i.e. NFAs are not stronger than DFAs) DFAs, NFAs, and regular expressions all compute the same class of languages. These are the regular languages

8.4.2 Grammars and Languages

A Formal Grammar is a 4 tuple $G = (V, \Sigma, P, S)$ where

- V is a finite set of (Syntactic) Variables
- Σ is a finite set of Terminal Symbols disjoint with V
- S is the Start Symbol
- P is a finite set of Production Rules

The Formal Language defined by a grammar is the set of all strings that can be derived from S

8.4.2.1 Regular Grammars

For regular grammars all productions are of one of these two forms

- A::= "a";
  A produces a terminal
- A::="a"B;
  A produces a terminal followed by a syntactic variable

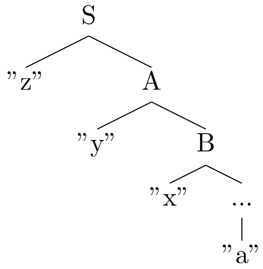


Figure 12: Syntax tree

The syntax tree of productions is always degenerated because terminals are "printed" from the left to the right one by one. Production proceeds until a single terminal is derived.

The languages generated by Regular Grammars are exactly the same as those defined by regular expressions. RegExps, DFAs, NFAs, and regular Grammars all generate or recognise exactly the Regular Languages.

8.4.3 Limits of Regular Languages

$$\text{Consider } L = \{a^n b^n \mid n \geq 0\} \tag{18}$$

This can not be recognised by an DFA for any finite n one could have a change of $2n$ states, first for n a's and then n b's. However, there is always a larger n. This can also not be expressed by a Regular Expression.

8.5 Context-Free Grammars

For Context-Free Grammars productions can have an arbitrary but finite sequence of variables on the right hand side of any rule.

A::=B ”c” D ”e”;

However, the left hand side must always be a single variable (without ”context” nearby)

8.5.1 Example Toy Language

This is a toy language because it always has single variable that get expanded, making it a tree.

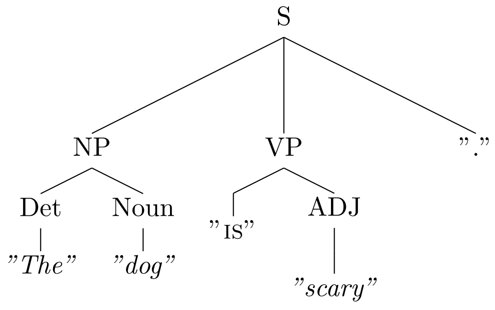


Figure 13: Toy Language

8.5.2 Example $a^n b^N$

If we set a grammer of G as $G = (\{S\}, \{a, b\}, \{S \rightarrow ab, S \rightarrow aSb\}, S)$ This grammar will generate $\{a^n b^n \mid N \geq 0\}$ This means that $S \rightarrow aSb \rightarrow aaSbb \rightarrow aaaSbbb \rightarrow aaaabbbbb$

8.5.3 Pushdown Automaton

A pushdown automaton is basically a finite state automaton with an additional stack to store some information about the ongoing computation. A PDA would use an NFA.

A NPDA M is a 7-tuple $M = \{Q, \Sigma, \Gamma, \delta, q_0, z, F, \}$ where

- Q is a finite set of states
- Σ is finite set of input symbols (”alphabet”)
- Γ is the stack alphabet of symbols that can be pushed onto the stack.
- δ: $Q \times \Sigma_\varepsilon \times \Gamma_\varepsilon \to \varphi(Q \times \Gamma^*)$ is the transition function where no tuple is mapped to an infinite set
- q_0 is a start state from Q
- z is the Stack start symbol
- F is a set of accepting states from Q

The automaton accepts if it ends in an accepting state with no input remaining.

8.5.4 Transition Function

The transition function is no longer a simple as it is $\delta(q_i, a_X) = \{(q_j, Y)\}$, non determinate as is possible.

- q_i is the current state
- a is the next input symbol
- X is the current stack top
- Y is the stack stop replacement
- q_j is the next state

8.5.5 NPDA for $a^n b^n$

A PDA for $L = \{a^n b^n \mid n \geq 0\}$ where $\Sigma = \{a, b\}$, $\Gamma = \{0, \$\}A$ and $ keeps tack of the ”bottom” of the stack.

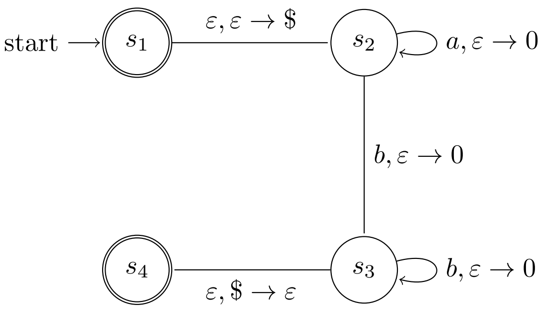


Figure 14: NFA of words ending in b

8.5.6 NPDA for Arithmetic

If we let $\Sigma = \{int, +, *, (,)\}$ and consider the language, ARITH $= \{w \in \Sigma^* \mid w$ is a legal arithmetic expression. So you could have expressions as such

- $int + int \times int$
- $((int + int) \times (int + int)) + (int)$

The NPDA for ARITHM would look DIAGRAM

8.5.7 The Language of a PDA

The language of a PDA is the set of string that the PDA accepts

$$\mathscr{L}(P) = \{w \in \Sigma * \mid P \text{ accepts } w\}$$

If P is a PDA where $\mathscr{L}(P) = L$, we say P recognises L.[8]

8.6 What does this all mean for Computing

- A Program is a "word" of a context free language (CFL) (C/C++/Java...)
- A Parser uses a CFL to construct a syntax tree
- A compiler uses the syntax tree (plus context-sensitive constraints) to generate code for the Program that can be directly executed on a machine.
- Execution may use a (Deterministic) PDA, but usually virtual or real machines that execute code are more complicated (Random Access Machine, von Neumann Architecture, Harvard Architecture ...)

[8] It can be shown that None-Deterministic PDAs compute (recognise) exactly the context free languages

8.7 Beyond Context-Free Grammars

In regular and context-free languages the left hand side of a production is always a single syntactic variable, more generally this could indeed be any finite, non empty string of variables and terminals e.g. $aBc \to uVwXy$. Leading to context-sensitive languages if the strings never get shorter under a rule and to Phrase-Structure Grammars if they can shrink e.g. $aBc \to uV$

8.7.1 Chomsky Hierarchy

Regular, context-free, context-sensitive and phrase structure languages form a hierarchy by proper set inclusion, the Chomsky Hierarchy.

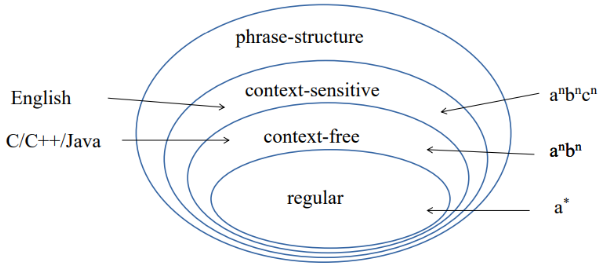


Figure 15: Chomsky Hierarchy

8.7.2 Turing Machines

Context-sensitive and phrase-structure languages can be recognised by Turing machines. A Turing machine is basically an Non-Deterministic Finite Automaton with two sided tape, where more tapes do not increase the power. The Turing machine can compute ”everything” that is computable with discrete states.

8.8 Summary

- RegExp are truly useful in programming
- Formal languages and grammars are truly useful to understand what computers can do
- RegExp, DFAs, NFAs regular grammars all exactly correspond with regular languages
- Context-free grammars exactly correspond with Non-Deterministic Pushdown Automata
- Formal Languages form the Chomsky Hierarchy

9 Turing Machines and Computability