Contents

1 2012 Exam - Part A 2
1.1 Oppgave 1 - Optimization: Constrained vs. Unconstrained . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2 Oppgave 2 - Simulated Annealing . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Oppgave 3 - EA: Long Runs . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.4 Oppgave 4 - EA: parent -and survivor selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.4.1 parent selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.4.2 survivor selection . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
  1.4.3 Effect of different implementations . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.5 Oppgave 5 - EA: one-point crossover . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.6 Oppgave 6 - EA: Termination Conditions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.7 Oppgave 7 - EA: Self Adaption . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.8 Oppgave 8 - Optimization: Evaporation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.9 Oppgave 9 - EA: Domain Knowledge . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4

2 2012 Exam - Part B 5
2.1 Oppgave 1 - Perceptron: XOR . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.2 Oppgave 2 - Multi-Layer Perceptron: Decision Boundary . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.3 Oppgave 3 - Multi-Layer Perceptron: Overfitting . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.4 Oppgave 4 - Early Stopping . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.5 Oppgave 5 - SVM: Feature Mapping . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.6 Oppgave 6 - Naïve Bayes’ Classifier . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.7 Oppgave 7 - k-NN Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
2.8 Oppgave 8 - k-NN Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.9 Oppgave 9 - Action Space . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.10 Oppgave 10 - Reward Function . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.11 Oppgave 11 - $\epsilon$-greedy policy . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.12 Oppgave 12 - Self-Organising Maps . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
2.13 Oppgave 13 - Supervised- and Unsupervised Learning . . . . . . . . . . . . . . . . . . . . . . . . . . . 6

3 2011 Exam - Part A 7
3.1 Oppgave 1 - GA . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
3.2 Oppgave 2 - Crossover . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
3.3 Oppgave 3 - No Free Lunch Theorem . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
3.4 Oppgave 4 - Genetic Programming: Roulette Wheel selection . . . . . . . . . . . . . . . . . . . . . . . . 7
3.5 Oppgave 5 - GA vs. ES . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
3.6 Oppgave 6 - MOOP: Domination . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 7
3.7 Oppgave 7 - Evolutionary Programming (EP): parent selection . . . . . . . . . . . . . . . . . . . . . . . . . 8

4 Common Evolutionary Algorithms 8
4.1 Ant-Colony Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
  4.1.1 Evaporation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
4.2 Gradient Descent Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
4.3 Greedy Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
4.4 Hill-Climbing Algorithm . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8
5 Common Terms
5.1 Domain Knowledge .......................................................... 8
5.2 Crossover ........................................................................ 9
5.3 Termination Conditions ...................................................... 9
5.4 Overfitting ....................................................................... 9
5.5 Soft Margins .................................................................... 9
5.6 Early Stopping .................................................................. 9
5.7 Feature Mapping ................................................................ 10
5.8 The Naïve Bayes’ Classifier ............................................. 10
5.9 Nearest Neighbour Classifiers ......................................... 10
5.10 Reward Function ............................................................. 10
5.11 Supervised- and Unsupervised-Learning ......................... 11
5.12 The No Free Lunch Theorem .......................................... 11
5.13 Roulette Wheel Selection ................................................. 11
5.14 Multi-Objective Optimization Problems (MOOP) ............ 11

6 Common Machine Learning Algorithms .............................. 12
6.1 Perceptron ...................................................................... 12
6.2 The Multi-Layer Perceptron (MLP) ................................. 12
6.3 Support Vector Machine (SVM) ........................................ 12
6.4 Reinforced Learning ......................................................... 12
6.5 Self-Organising Maps ....................................................... 12

Index .................................................................................... 13

List of Figures
1 Illustration of why heuristic initialisation might not be worth additional effort .............................................. 3
2 An example of how the one-point crossover may result in doubled bits ....................................................... 4
3 Data for the logic XOR function and plot of the four data-points .............................................................. 5
4 ............................................................................................. 8
5 The effects of overfitting is that rather than finding the generating function (as shown on the left), the neural network matches the inputs perfectly, including the noise in them (on the right). This reduces the generalisation capabilities of the network ......................................................... 9
6 The effect of overfitting on the training and validation error curves, with the point at which early stopping will stop the learning marked. ................................................................. 10
7 A set of 5 bit-strings with different fitness value evaluated by roulette wheel selection ............................ 11

1 2012 Exam - Part A

1.1 Oppgave 1 - Optimization: Constrained vs. Unconstrained

From WikiPedia: Optimization is the selection of a best element from some set of available alternatives.

In general we wish to either maximize or minimize a function by choosing different input values (x,y,...). Constrained Optimization: given a function of f(x₁,...,xₙ) we want to maximize or minimize it by giving constraints to the inputs (x₁,...,xₙ). Examples:

\[ \min f(x, y) = x^2 + 2y^2, \quad x > 0 \]
\[ \min f(x, y) = x^2 + 2y^2, \quad -2 < x < 5; y \geq 1 \]
\[ \min f(x, y) = x^2 + 2y^2, \quad x^2 + y^2 = 2 \]

Unconstrained Optimization: in unconstrained optimization there are no limitations on the values of the parameters:

\[ \min f(x, y) = x^2 + 2y^2 \]

Constrained functions are usually better, but requires good constraints.
1.2 Oppgave 2 - Simulated Annealing
Simulated Annealing is stochastic algorithm (randomness), where the outcome may be different at different trials. It is a variant of hill-climbing, in the start of the process, some downhill moves may be made. It is often used when the search space is discrete.
- At each step the algorithm considers some neighbour states, and then decides whether to move to one of the states or to stay in the current state.
- It does exploration of the whole space early on, so it’s relatively insensitive to the starting state.
- It allows occasional ascent in the search process, that makes it possible to escape from local minima/maxima.

1.3 Oppgave 3 - EA: Long Runs
In general, the longer you run your evolutionary algorithm, the closer you get to the best value, or maxima. But after a certain amount of runs, the different solutions, or the population, becomes so similar to each other, that there is no point of continuing. Basically, if you increase the number of generations by a notable amount, you increase the computation-time a lot more then the you increase the accuracy of your search.

From “Introduction to Evolutionary Computing”:

![Figure 1: Illustration of why heuristic initialisation might not be worth additional effort](image)

1.4 Oppgave 4 - EA: parent -and survivor selection

1.4.1 parent selection
In Evolutionary Algorithms, parent selection is to distinguish among individuals based on their quality. In general, the individuals with the best value are allowed to be parents and to generate the new offsprings, and by extension the parents selection mechanism are responsible for pushing quality improvements.

1.4.2 survivor selection
The role of survivor selection is to distinguish the individuals based on their quality. In this sense it is equal to the parent selection, but the survivor selection is used in a different stage of the evolution. Since, in many cases, we want to keep the population size constant, the survivor selection decides which individuals will be allowed in (survive to) the next generation. This decision is usually based on their fitness value, although the concept of age has been frequently used.

1.4.3 Effect of different implementations
In concerns of parent-selection, we are basically just adjusting the chance for bad solutions to be allowed to reproduce. The choice of bad solutions chance of reproducing varies the bandwidth of solutions we get from the search, varies the chance of getting stuck at a local optima, and variates how greedy our algorithm are considering the computational time it take to achieve a “good-enough” solution.

1.5 Oppgave 5 - EA: one-point crossover
The one-point Crossover is not suitable for use with permutation representation because you would get doubled up of some bits during the mutation:
Figure 2: An example of how the one-point crossover may result in doubled bits

More suitable crossovers:

- Cycle Crossover
- PMX
- Order Crossover

1.6 Oppgave 6 - EA: Termination Conditions

Termination Conditions is a term used in evolutionary algorithms that describes the situation where we want to stop the search/algorithm from going any further. There could be multiple reasons for this, and some commonly used termination conditions are:

- The maximal allowed CPU time has elapsed
- The total number of fitness evaluations has reached a given limit
- For a given number of time (i.e. for a number of generations or fitness evaluations) the fitness improvement remains under a given threshold value.
- The population diversity drops under a given threshold.

1.7 Oppgave 7 - EA: Self Adaption

Self Adaptation is the concepts of how an evolutionary algorithm would adjust its parameters while running in according to some reference on how well the evolution is going. Typically, in the beginning of a search process, a large part of the search space has to be sampled in order to locate promising regions. Therefore, large mutations are appropriate in this phase. As the search proceeds, and the algorithms is getting closer to an optimal solutions, only fine tuning of the given individuals is need: thus smaller mutations are required.

1.8 Oppgave 8 - Optimization: Evaporation

Evaporation is a term used in, amongst others, the Ant-Colony Algorithm. It builds on the concept of pheromones and the fact, that in nature, these pheromonos evaporates. For ants, this makes long routes non applicable because the trails rate of evaporation is larger than the ants rate of reinforcement. And this makes for a good analogy for the algorithms as well. If the solution found is to expensive with respect to computational time, it isn’t worth following up.

1.9 Oppgave 9 - EA: Domain Knowledge

Weither insertion of Domain Knowledge is beneficial or not depends on the problem at hand. If we have a small search space, Domain Knowledge could increase the diversity of the population. Contrary, if we have a large search space, there are so many solutions that the computational time becomes so expensive that Domain Knowledge becomes unnecessary because the difference in the best solution found wouldn’t be of any admirable degree.
2 2012 Exam - Part B

2.1 Oppgave 1 - Perceptron: XOR

As illustrated in (3), it is impossible to draw a straight line to separate true from false. This is because the XOR function is not linearly separable. One can solve this problem by adding more neurons.

2.2 Oppgave 2 - Multi-Layer Perceptron: Decision Boundary

In the Multi-Layer Perceptron (MLP), to represent an increasingly complex decision boundary, we need to have at least two hidden layers. We can use back-propagation to determine the weights of the different nodes. There are no theory to guide how many nodes you need for any given algorithm, one will just have to try, and chose the one that gives the best results, though it is sensible to assume that with more complex computations, one would need more nodes and weights.

2.3 Oppgave 3 - Multi-Layer Perceptron: Overfitting

In the Multi-Layer Perceptron (MLP), Overfitting can occur if we train to long on a specific dataset. In the Support Vector Machine (SVM), Overfitting can occur if we use to strict margins. If we use Soft Margins, we could avoid this problem.

2.4 Oppgave 4 - Early Stopping

see Early Stopping.

2.5 Oppgave 5 - SVM: Feature Mapping

see Feature Mapping.

2.6 Oppgave 6 - Naïve Bayes’ Classifier

see The Naïve Bayes’ Classifier

2.7 Oppgave 7 - k-NN Algorithm

The k-NN Algorithm is a version of the Nearest Neighbour Classifiers. If we have a test-point, we can identify the k nearest neighbours, and then set the class of the test-point to be the most common amongst the nearest neighbours. We have two important functions (or kernels) that decides how much emphasis (weight) to put onto the contribution from each datapoint according to its distance from the input.

Both of these kernels are designed to give more weight to point that are closer to the current input, with the weights decreasing smoothly to zero as they pass out of the range of the current input, with the range specified by λ. First, we have the Epanechnikov quadratic kernel:

\[ K_{E,\lambda}(x_0, x) = \begin{cases} 
0.75(1 - \frac{(x_0 - x)^2}{\lambda^2}) & \text{if } |x - x_0| < \lambda \\
0 & \text{otherwise}
\end{cases} \]

Then there’s the tricube kernel:

\[ K_{T,\lambda}(x_0, x) = \begin{cases} 
(1 - |\frac{x_0 - x}{\lambda}|^3)^3 & \text{if } |x - x_0| < \lambda \\
0 & \text{otherwise}
\end{cases} \]
2.8 Oppgave 8 - k-NN Algorithm

The choice of $k$ is not trivial. Make it too small and the nearest neighbour methods are sensitive to noise, too large and the accuracy reduces as points that are too far away are considered.

2.9 Oppgave 9 - Action Space

see In the game of chess we have a board with the size of 64 cells. This defines the state space for each of our pieces. The action space for each piece depends on the class of the piece. The pawn, for example, can only move forwards, except for when its an opponent piece diagonal from our piece, when the queen can move in 8 different directions. We can thus represent it this way:

State Space
= 64 (All cells on the board)

Action Space
- Pawn : 2
- Knight: 4
- Bishop: $4\times x$
- Rook : $4\times x$
- Queen : $8\times x$
- King : 8

where 'x' is the number of available cells on the straight/diagonal line from the piece to a given point.

2.10 Oppgave 10 - Reward Function

see Reward Function.

In the task we compare the following two reward functions on learning to play the game of chess in a reinforced learning context:
i. reward = +1 if the agent wins; reward = -1 if the agent loses; reward = 0 otherwise
ii. reward = same as i, but additionally reward = +1 when the agent captures an opponent’s piece.

In order to win the game of chess you would have to make the king unable to move without getting captured. In the first reward function, i, the algorithm has only the motivation of winning the game, and no concept of how it should be done. It would therefore move its pieces randomly around until it either loses or wins, while the other algorithm would try to capture as many pieces as possible. ii is probably a better solution, because in order to win, one should try to strip the opponent of as many pieces as possible, although in chess it isn’t always a good idea to capture an opponent’s piece just because you can. So this algorithm would probably loose against a more experienced chess player aswell. An optimization would be to implement a reward = -1 when one of the agents piece’s are capture. Then the algorithm would try to capture as many pieces as possible without loosing one of it’s own.

2.11 Oppgave 11 - $\epsilon$-greedy policy

In the $\epsilon$-greedy policy we typically decrease the value of $\epsilon$ over time, because in the start we favour exploration, in other words we want to explore large parts of the state space. As the algorithm starts to find good action sequences, we lower $\epsilon$ to favour exploitation, in other words we want the algorithm to continue in that part of the state space that gives the best results.

2.12 Oppgave 12 - Self-Organising Maps

When we are working with Self-Organising Maps, it is often helpful reduce the dimensionality of data. One can imagine a room full of coordinates that stretches from the lower left corner, to the upper right. These points have intrinsic dimensionality of 2 since they are all on the same plane, but they are represented in an 3-dimensional space. In general, noise and other inaccuracies in data often leads to it being represented in more dimensions than are actually required, and so finding the intrinsic dimensionality can help reduce the noise.

2.13 Oppgave 13 - Supervised- and Unsupervised Learning

see Supervised- and Unsupervised-Learning
3 2011 Exam - Part A

3.1 Oppgave 1 - GA

- Genetic Algorithms (GA) is a randomized parallel search algorithm, based on the principles of natural selection, the process of evolution \( \rightarrow \) true
- GAs are exhaustive, giving out all the optimal solutions to a given problem \( \rightarrow \) false
- GAs are used for solving optimization problems and modeling evolutionary phenomena in the natural world. \( \rightarrow \) true
- Under any circumstances, GAs always outperform other algorithms \( \rightarrow \) false

3.2 Oppgave 2 - Crossover

If all of the crossover operations between chromosomes in a generation do not produce significantly different offspring (given that each offspring consist of one half of each parent), what does it imply?

- The crossover operation is not successful \( \rightarrow \) false
- The population is converging to a solution \( \rightarrow \) true
- Diversity is so poor that the parents involved in the crossover operation are similar \( \rightarrow \) true
- The search space of the problem is not ideal for GAs to operate \( \rightarrow \) false

3.3 Oppgave 3 - No Free Lunch Theorem

see The No Free Lunch Theorem

3.4 Oppgave 4 - Genetic Programming: Roulette Wheel selection

see Roulette Wheel Selection.
For our problem, we would have to estimate the selection probability for each bit-string.

\[
TotalFitness = 3 + 6 + 9 + 12 + 15 = 45
\]

We get the selection probabilities: \( \{ \frac{3}{45}, \frac{6}{45}, \frac{9}{45}, \frac{12}{45}, \frac{15}{45} \} \)

3.5 Oppgave 5 - GA vs. ES

Similarities

in GA, the choice of “Representation”, “Parent-Selection Algorithm”, “Recombination Algorithm” “Mutation” and “Survival Selection” is up to the architect, and the problem at hand. So when it comes to these parameters, the GA and the ES could be alike.

Differences

GA usually generates lesser or equally many children as parents, whereas ES usually generates a much more children than there are parents

The meaning of \((\mu,\lambda)\)-ES and \((\mu+\lambda)\)-ES (\(\mu\) is the number of parents within a population. \(\lambda\) is the number of children within a population):
Both are survivor selection methods: - \((\mu,\lambda)\): Survivors are selected from the offsprings only. - \((\mu + \lambda)\): Survivors are selected from both parents and offsprings.

3.6 Oppgave 6 - MOOP: Domination

see Multi-Objective Optimization Problems (MOOP).
The concept of “domination for two solutions (x and y)” of a multi-objective optimization problem:
The solution 'x' dominates 'y' if 'x' is as good in every way as 'y' and better for at least one objective.
3.7 Oppgave 7 - Evolutionary Programming (EP): parent selection

4 Common Evolutionary Algorithms

4.1 Ant-Colony Algorithm

The Ant-Collony Algorithm is based on the phenomenon found in ant behavior that searches for food. At first the ants move in random patterns, until one of the ants find a source of food. He then leaves behind a pheromone trail. If other ants find such a trail, they are likely not to keep travelling at random, but instead follow the trail, and as these other ants find the food source, they return and reinforce the pheromone trail in the process. This concept can be implemented in combinatorical optimization problems, for instance The Traveling Salesman Problem. If we generate a population that at random search for the best route, we could add the functionality of releasing pheromones if an individuals route has a certain fitness value, and thus more individuals will follow this route with slight variations, and thus reinforcing the routes fitness value.

4.1.1 Evaporation

Evaporation builds on the concept of pheromones and the fact, that in nature, these pheromones evaporates. For ants, this makes long routes non applicable because the trails rate of evaporation is larger than the ants rate of reinforcement. And this makes for a good analogy for the algorithms as well. If the solution found is to expensive with respect to computational time, it isn’t worth following up.

4.2 Gradient Descent Algorithm

The “Gradient Descent” algorithm forms the basis of many machine learning algorithms. It is mainly used in continious optimization problems. Gradient is defined as the vector of partial derivatives:

\[ \nabla f(x) = \left( \frac{\delta f}{\delta x_1}, \frac{\delta f}{\delta x_2}, ..., \frac{\delta f}{\delta x_n} \right)\]

where:

\[ f(x) = (x_1, x_2, ..., x_n) \]

To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient of the function at the current point.

![Figure 4:](image)

4.3 Greedy Algorithm

4.4 Hill-Climbing Algorithm

5 Common Terms

5.1 Domain Knowledge

The term “Domain Knowledge” explains the concept of how, in an algorithm, one can early on explore the whole search space to find promising areas. It could be a useful tool when there is a small search space, and computational
time isn’t that big an issue. The reasons for this is that we increase the diversity of individuals in the population early on.

5.2 Crossover

5.3 Termination Conditions

5.4 Overfitting

Overfitting is the incident where an system has trained to much on one dataset that it matches its inputs so perfectly (even the noise), that the algorithm isn’t generalisationable any more.

Figure 5: The effects of overfitting is that rather than finding the generating function (as shown on the left), the neural network matches the inputs perfectly, including the noise in them (on the right). This reduces the generalisation capabilities of the network

5.5 Soft Margins

Instead of perfectly separating all data, allow some misclassifications.

Introduce slack variables:
- Optimize tradeoff between maximum margin and misclassification penalty.
- Tradeoff is balanced by penalty factor “C”.

Usefull when some error is tolerated, or when there are chances of mislabeled training data.

5.6 Early Stopping

Early Stopping is a technique to avoid Overfitting when working on Machine Learning networks. Generally we use the validation set to monitor the generalisation ability of the network at its current stage of learning. We train the network for some predetermined amount of time, and then use the validation set to estimate how well the network is generalising. We then carry on training for a few more iterations, and repeat the process. At some point, the error of the validation set will start increasing again, because the network has stopped learning about the function, and started to learn about the noise that is in the data itself. At this stage we stop the learning.
Figure 6: The effect of overfitting on the training and validation error curves, with the point at which early stopping will stop the learning marked.

5.7 Feature Mapping

Feature Mapping is a functionality in Machine Learning that describes the transition from one Feature Space to another. The Feature Space is a description of all the variables as features. Example:

Target: 
\[ Y = \text{Thickness of car tires after some testing period} \]

Variables:
1. \( X_1 \) = distance travelled in test.
2. \( X_2 \) = time duration of test.
3. \( X_3 \) = amount of chemicals C in tires.

The Feature Space is now \( \mathbb{R}^3 \). We could generate another variable, lets say:
\[ X_4 = \frac{X_1}{X_2} \] = the speed of the vehicle during testing

This extends our old feature space into a new one, \( \mathbb{R}^4 \). Therefore we say Feature Mapping is a function, \( \phi \), from \( \mathbb{R}^3 \) to \( \mathbb{R}^4 \):
\[ \phi(x_1, x_2, x_3) = (x_1, x_2, x_3, \frac{x_1}{x_2}) \]

5.8 The Naïve Bayes’ Classifier

When we use a Naïve Bayes’ Classifier, we assume that the vector of features has many elements, so that there is lots of different features to measure. We also assume that the elements of the feature vector are conditionally independent of each other (often doesn’t make much sense, thus naïveté).

5.9 Nearest Neighbour Classifiers

Analogy: - Say you’re in a nightclub and decide to dance. It is unlikely that you know the moves for the particular song that is playing, so you would try to work out what to do by looking at person closest to you and copy that person. However, since most of the people who are in the nightclub are also unlikely to know the moves, you might decide to look at a few people and do what most of them are doing.

This is pretty much exactly the idea behind nearest neighbour methods: If we don’t have a model that describes the data, then the best thing to do is to look at similar data and choose to be in the same class as them.

5.10 Reward Function

The “reward function” is a function often applied to reinforced learning. It is a function that defines how well any given action are with respect to achieving the goal. The reward function tells the algorithm how well it is doing in according to achieving its goal, but it doesn’t tell it how it should achieve its goal. The choice of a suitable reward function is absolutely critical, with very different behaviours resulting from varying the reward function.
5.11 Supervised- and Unsupervised-Learning

Supervised Learning
A training set of examples with the corresponding responses (tags) are provided and, based on this training set, the algorithm generalises to respond correctly to all possible inputs. This is called learning from examples.

Unsupervised Learning
Correct responses are not provided, instead the algorithm tries to identify similarities between the inputs so that inputs that have something in common are categorised together.

5.12 The No Free Lunch Theorem
The “No Free Lunch” theorem is important to keep in mind when working with algorithms, because it means that there is no algorithm that is guaranteed to perform better than any other algorithm for any given problem, i.e. there is no ideal solution to any problem. Depending on the problem, one should always give some consideration to what algorithm would be the best choice for this specific problem.
“Every algorithm has its price”.

5.13 Roulette Wheel Selection
“Roulette Wheel Selection” is a way of selecting individuals (or bit-strings) from a set of individuals (or set of bit-strings) with respect to their fitness values. If you imagine a one-armed spinning wheel, where the size of the different areas reflect the selection probabilities (usually their fitness value divided by the total fitness of the population) of the different individuals.

![Figure 7: A set of 5 bit-strings with different fitness value evaluated by roulette wheel selection](image)

5.14 Multi-Objective Optimization Problems (MOOP)
A “Multi-Objective Optimization Problem” is an Optimization Problem where we want satisfy more than one objective function simultaneously with respect to optimization. Usually denoted:

\[
\text{optimize}(f_1(x), f_2(x), \ldots, f_k(x))
\]

In most cases we would have to find a pareto-optimal, a point where none of the objectives could get it better without anyone getting it worse. Example:
Say we’re a car manufacturer, and we want to optimize our product with respect to fuel consumption, engine performance, air resistance, etc. We would then have to write an optimization function with respect to all these parameters.
6 Common Machine Learning Algorithms

6.1 Perceptron

6.2 The Multi-Layer Perceptron (MLP)

6.3 Support Vector Machine (SVM)

6.4 Reinforced Learning

Reinforced learning is often implemented in unsupervised learning, where an algorithm maps states to actions in order to maximize some numerical reward. There is a clear distinction between the agent, that is doing the learning, and the environment, which is where the agent acts. The easiest way to think about reinforcement learning is by thinking about a robot. The current sensor readings of the robot, or processed versions of them, could define the state. They are a representation of the environment around the robot. The possible ways that the robots can drive its motors are the actions, which move the robot in the environment, and the reward could be how well it does its task without crashing into things.

The basic idea is that the learning algorithm choose the action that gets the maximum expected reward. The reward is calculated by the reward function, which is pretty similar to the fitness function for evolutionary algorithms. The reward-function tells the learner what the goal is, not how it should be achieved.

6.5 Self-Organising Maps

The Self-Organising Map (or SOM) is alot similar to the Feature Mapping, with some additional functions. The neurons in the network should be located near neurons that corresponds to similar input patterns. In addition, the neurons are arranged in a grid with connections between them, rather than in a layer with connections only between different layers.
Index

Action Space, 6

Early Stopping, 5, 9
Epanechnikov quadratic kernel, 5
Evolutionary Algorithm, 3
  one-point crossover, 3
  parent selection, 3
  run length, 3
  survivor selection, 3
  termination conditions, 4
Evolutionary Programming
  Parent Selection, 8

Feature Mapping, 10
Feature Space, 10

GA vs. ES, 7
Genetic Algorithm
  Roulette Wheel Selection, 7, 11
Gradient Descent, 8

MLP
  decision boundary, 5
  overfitting, 5
Multi-Objective Optimization Problems, 7, 11

Naïve Bayes’ Classifier, 10
Nearest Neighbour Classifiers, 10
  k-NN Algorithm, 5, 6
No Free Lunch Theorem, 11

Overfitting, 9

Perceptron, 5
Policies
  $\epsilon$-greedy policy, 6

Reinforced Learning, 12
Reward Function, 6, 10

Self-Organising Maps, 6, 12
Simulated Annealing, 3
Soft Margins, 9
Supervised Learning, 11

tricube kernel, 5

Unsupervised Learning, 11