Master's Thesis

A Gamification approach for the experimental proof of the interactive Machine Learning concept

Andrej Müller

Institute of Interactive Systems and Data Science,
Graz, University of Technology

Supervisor: Assoc.Prof. Dr. Andreas Holzinger

Graz, September 16, 2018

This page intentionally left blank

Masterarbeit

(Diese Arbeit ist in englischer Sprache verfasst)

Ein Gamifizierungs-Ansatz zur experimentellen Wirksamkeit des interaktiven Maschinellen Lernens

Andrej Müller

Institute of Interactive Systems and Data Science,
Technische Universität Graz

Betreuer: Holzinger, Andreas, Mag.phil. Mag.rer.nat. Dr.phil. Univ.-Doz. Ing.

Graz, 16. September 2018

This page intentionally left blank

Abstract

Todays' life is driven by Artificial Intelligence(AI) in form of Machine Learning(ML) algorithms. They recommend products, help in the household and some of them even solve complex problems, like helping us cure diseases. These algorithms usually don't include humans in the process of computing an outcome.
This work is about interactive Machine Leaning (iML). It discusses the basic concept and introduces some self-made applications. The main application is a playable game with an iML algorithm running in the background. The algorithm is called interactive Ant Algorithm, which is based on the swarm behavior of ants. The work starts with an introduction into all the basic fields of Artificial Intelligence, Swarm Algorithms, Probability and Gamification. Later on it leads to my practical work and shows the outcome.

Keywords
Interactive Machine Learning, Gamification, Ant Algorithm

ÖSTAT classification
ÖSTAT CLASSIFICATION

ACM classification
ACM CLASSIFICATION

This page intentionally left blank

Kurzfassung

Das moderne Leben ist gesteuert durch Künstliche Intelligenz (KI) oder besser gesagt durch Algorithmen im Bereich Maschinelles Lernen (ML). Sie schlagen uns Produkte vor, helfen uns im Haushalt und manche von ihnen lösen sogar komplexe Probleme, wie z.B. Krankheiten. Die Algorithmen, die dazu verwendet werden, binden den Menschen gewöhnlich nicht in die Entscheidungsfindung ein.
Diese Arbeit behandelt das Thema interaktives Maschinelles Lernen (iML). In der Arbeit wird das Grundkonzept von iML behandelt. Weiters werden auch eigene Computerprogramme vorgestellt. Das Kernprojekt der Arbeit ist ein Spiel mit einem interaktiven Algorithmus, welcher im Hintergrund agiert. Der Algorithmus wird interaktiver Ameisenalgorithmus genannt und ahmt das Schwarmverhalten von Ameisen nach. Die Arbeit startet mit einer Einführung in das Thema Künstliche Intelligenz, Schwarmalgorithmen, Wahrscheinlichkeiten und Gamifizierung. Danach wird meine praktische Arbeit präsentiert und zum Schluss die Ergebnisse diskutiert.

Schlüsselwörter
Künstliche Intelligenz, interaktives Maschinelles Lernen, Ameisenalgorithmen, Gamifizierung

ÖSTAT Klassifikation
ÖSTAT CLASSIFICATION

ACM Klassifikation
ACM CLASSIFICATION

This page intentionally left blank

STATUTORY DECLARATION

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Graz, September 16, 2018

Andrej Müller

This page intentionally left blank

Acknowledgements

Andrej Müller

Graz, September 16, 2018

This page intentionally left blank

Table of Contents

1. Introduction and Motivation for Research

In this first chapter I want to introduce the basic concepts of my work: Gamification, Probability and Machine Learning.

1.1 Gamification

1.1.1 Introduction

Nowadays hundreds of millions are playing video games in their everyday life. The number is increasing rapidly. People playing video games are called *Gamers*. For a few minutes or hours a day, over the whole weekend or sometimes even more than 40 hours a week they abandon their reality for a virtual one. They are lawyers, who come home to play *MMOs*[1], where they coordinate up to 40 people to solve complex tasks. (e.g. defeating in-game bosses) They are teachers, who are playing ego shooters, where they try to survive in a war, by killing other players in the game. They are mothers, who are playing Candy Crush [2] on their smartphones, swiping candy, to climb up the highscore. But most of all they are teenagers and young people, who rather spend their spare time in a virtual world, than in the real one.

It's highly likely, that computer/virtual games in the twenty-first century will be a primary platform to enable the future! Games like World of Warcraft give players the means to save worlds, and incentive to learn the habits of heroes. (McGonigal (2011b)) Until the year 2011 players spent 5.93 million(now it's much more) years

[1] Massive Multiplayer Online Games (e.g. World of Warcraft)
[2] https://candycrushsaga.com/en/ visited on 14.08.2018

playing this specific game. So why are we spending so much time tackling unnecessary obstacles? A quote from Brian Sutton-Smith says, that *the opposite of play isn't work - it's depression.* Now take a look at the definition of depression: *A mental condition characterized by feelings of severe despondency and dejection, typically also with feelings of inadequacy and guilt, often accompanied by lack of energy and disturbance of appetite and sleep.*[3]. The opposite seems to be, what McGonigal said at the Game Developer Conference 2011: *An optimistic sense of our own capabilities and a rush of invigorating activity.* She sums up, that this would be a perfect description of the emotional state of gameplay. (McGonigal (2011a))

So the aspect of Gaming is getting more and more important. What if this enjoyment and energy can be used to solve complex problems, in other words, to make boring scientific work fun?

This is why the term *Gamification* came up. In the Oxford dictionary [4] "Gamification" is described as follows:

"The application of typical elements of game playing (e.g. point scoring, competition with others, rules of play) to other areas of activity, typically as an online marketing technique to encourage engagement with a product or service: *gamification is exciting because it promises to make the hard stuff in life fun*"

In 2010 game designer Jesse Shell predicted, that in the future our life will be strongly influenced by *Gamification* aspects. (Lee and Hammer (2011)) Now, in 2018, *Gamification* has a strong foothold in marketing, health, fitness, politics and many other fields. Even some everyday routines, like brushing your teeth [5] are *gamified* today. On http://www.thefuntheory.com/ [6] you can also see some examples of Gamification in everyday life, e.g. how people can be convinced to use the stair instead of the escalator. The strong impact of Games, especially in form of Gamification, hit us already, so now is the turn to take advantage out of it.

[3]https://en.oxforddictionaries.com/definition/depression visited on 11.01.2018
[4]https://en.oxforddictionaries.com/definition/gamification visited on 11.01.2018
[5]http://www.playbrush.com visited on 08.07.2018
[6]http://www.thefuntheory.com/ visited on 08.07.2018

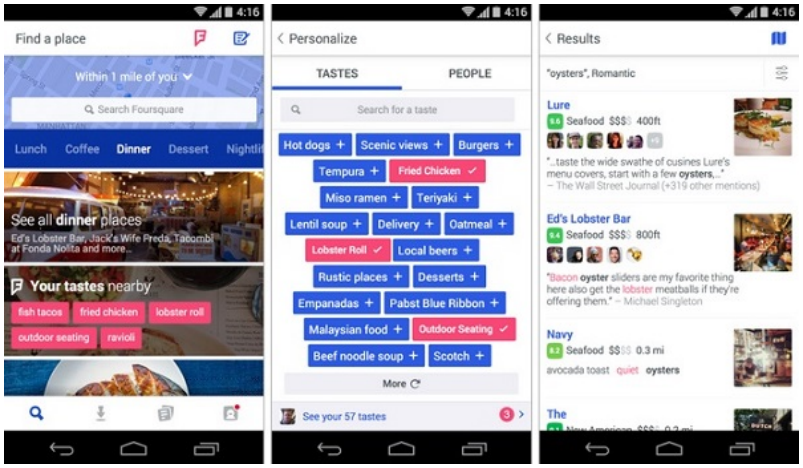


Figure 1.1: Foursquare app on Android
(from https://www.fonearena.com/blog/111463/revamped-foursquare-app-goes-live-on-android-and-iphone.html visited on 08.07.2018)

1.1.2 Examples

Gamification is a really strong tool to attract a large amount of people. That's why quite a lot of software solutions implement it in certain ways. In this section I will give you some examples of gamified software.

Foursquare - Explore

Foursquare (see figure 1.1) is a "local-search and discovery" mobile app which launched in 2009. Today it has quite a lot of features: local search & recommendations, tips & expertise, tastes, location detection, ratings and lists. [7] Gamification takes place in awarding users who write recommendations, seeing what their friends are doing, awarding users sharing the location, by checking in at places of interest.

Mira Rehab - Health

Imagine a system that might help those with conditions such as Parkinson's, Multiple Sclerosis, or Alzheimer's. - with this sentence Cosmin Mihaiu started a TED presentation in 2015, about a gamified software called Mira. Mira, which stands for Medical Interactive Recovery Assistant, makes boring physical exercises after a serious injury fun. This software provides a set of simple video games. By playing

[7] https://en.wikipedia.org/wiki/Foursquare visited on 08.07.2018

Figure 1.2: Child and Physiotherapist using the MIRA Rehab system (from http://blog.mirarehab.com/2016/05/31/case-study-15-year-old-syndrome-uses-mira-rehabilitation/ visited on 08.07.2018)

a game with a precise physical movement sequence, captured by a Microsoft Kinect input device, you can gain points or clear a level. (see figure 1.2) According to the website all games can be personalized by parameters and needs.

Pokémon Go - Fun

Another interesting example of Gamification is the mobile game Pokémon Go. In this game you use the real world as a playground. The game consists of a map and uses the GPS signal to track your position. You have to walk around in the city and catch Pokémons, gather items and compete with other players. The so called *non-gaming context* in this game is the fitness aspect and the real world itself. The game is also linked to the users social media profiles. There is a possibility to use the GPS data for commercial goals. (e.g. advertising) There are a lot of advantages, if you need to physically work to reach a goal in the game. It can help overcome depression, solve the obesity problem, which leads to less type 2 diabetes diseases and even make people socially interact with each other. (McCartney (2016)) But it has also negative aspects. You could be hit by a car, due to lack of attention to the traffic or you could enter some restricted areas. Nevertheless it is a social game experience which has never existed before. Sure, there are already health applications on the market, but the big difference is, that Pokémon Go is not marketed as health application. It is a game and therefore has a quite different target group.

Figure 1.3: Pokémon Go mobile app
(from https://www.xyztimes.com/13127/download-pokemon-go-apk-update-android-latest-version.html visited on 08.07.2018)

According to these three examples we can conclude, that Gamification is important and people are using it for years already. It helps us to make life easier, more interesting and also more challenging.

In the next sections I want to introduce the basics of probability and Machine Learning, which is the foundation of my practical work.

1.2 Probability

The serious study of risk began during the Renaissance. 1654 a French nobleman challenged the mathematicians Blaise and Pascal to solve a puzzle: Suppose we have an unfinished game between two players, where one of them is ahead, what are the odds? The solution of this puzzle from Pascal and his friend Fermat meant, that for the first time people can make predictions or decisions using numbers. In the eighteenth century Bernoulli introduced the law of large numbers, inspired by Leibniz. Moivre suggested the structure of normal distribution. 100 years after the Pascal and Fermat breakthrough, Bayes demonstrated, how to make better decisions, by blending new information into old one. In the nineteenth century Galton discovered regression to mean, an important probability fundamental, which explains why clouds tend to have silver linings. (Bernstein and Bernstein (1996)) Thanks to the research of these mathematicians and progress in technology we were able to introduce fields like Information Theory and Machine Learning.

1.2.1 Probability Theory

In this subsection I will breefly introduce some basics of the probability theory. It is strongly based on the chapter 2 of Murphy (2012) and chapter 3 of Goodfellow et al. (2016). Both books are very good introductions to probability theory. I will also discuss Information Theory by Shannon, to get a basic understanding behind data and its‘ compression, since we are now living in a time of exponential grow of data and computational power.

Discrete random variables

Consider we have the expression $p(E)$, where E is an event and $p(E)$ is the probability that the event will occur. E is for example "coin lands tail", if we flip a coin. It is required that $0 \leq p(E) \leq 1$, where for $p(E)$ a value of 1 means, that the event will definitely happen and 0, that it will not happen at all. There is also the possibility to negate a probability, with the notation $p(\overline{E})$, which means exactly the opposite.

There is also the possibility to extent the notation of binary events by defining a **discrete random variable (RV)** X, which takes values from a finite or countable

infinite set $\mathcal{X}$. We can write that the probability that X is set to a particular event x is $p(X = x)$ or simply $p(x)$. Hereby the $p()$ is called **probability mass function (pmf)**. It satisfies, as above, the property $0 \leq p(x) \leq 1$ and additionally also $\sum_{x \in \mathcal{X}} p(x) = 1$.

Marginal Probability

In some cases it is necessary to know the probability distribution of a subset of a known set. This is called *marginal probability distribution*(1.1)

$$\forall x \in X, p(X = x) = \sum_{y} p(X = x, Y = y). \tag{1.1}$$

Conditional Probability

Most of the time it is necessary to know the probability of an event, given that some other event happened before. This is called *conditional probability distribution*(1.2).

$$p(Y = y \mid X = x) = \frac{p(Y = y, X = x)}{p(X = x)}. \tag{1.2}$$

The Chain Rule for Conditional Probabilities

If we want to decompose a joint probability distribution over many RVs into a conditional distribution over only one RV we need to use the *Chain Rule of Probabilities*(1.3).

$$p(X_1, ..., X_n) = p(X_1) \prod_{i=2}^{n} p(X_i \mid X_1, ..., X_{i-1}). \tag{1.3}$$

Independence

Two RVs are *independent*, if the equation 1.4 holds.

$$\forall x \in X, y \in Y,$$
$$p(X = x, Y = y) = p(X = x)p(Y = y) \iff X \perp Y. \tag{1.4}$$

There is also a conditional independence, where equation 1.5 holds.

$$\begin{aligned}&\forall x \in X, y \in Y, z \in Z,\\ &p(X = x, Y = y \mid Z = z) = p(X = x \mid Z = z)p(Y = y \mid Z = z) \iff X \perp Y \mid Z.\end{aligned} \tag{1.5}$$

Expected Value and Variance

An expected value of a RV is the theoretical mean of the RV, shown in equation 1.6 and for functions in 1.7

$$E(X) = \sum_x xp(x) = \mu. \tag{1.6}$$

$$E(g(X)) = \sum_x g(x)p(x). \tag{1.7}$$

The average squared distance from the mean is called Variance, shown in 1.8.

$$Var(X) = E\left[(X - \mu)^2\right] = \sum_x (x - \mu)^2 p(x) = E(X^2) - [E(X)]^2 = \sigma^2. \tag{1.8}$$

Bayes' Rule

One of the most important rules in probability is the Bayes' Rule in equation 1.9. We have a lot of situation, where $p(Y \mid X)$ and $p(X)$ is known and we need to know $p(X \mid Y)$.

$$p(X \mid Y) = \frac{p(X)p(Y \mid X)}{p(Y)}. \tag{1.9}$$

The calculation of $p(Y)$ is shown in equation 1.10.

$$p(Y) = \sum_x p(Y \mid x)p(x). \tag{1.10}$$

1.2.2 Common Distributions

A very useful tool are common distributions, since they are used very often in Machine Learning algorithms.

Figure 1.4: Binomial Distribution

The green plot shows the binomial distribution with $n = 20$ and $\theta = 0.9$ and the red one shows it with $\theta = 0.5$.

Bernoulli Distribution

If we observe only one binary RV the distribution is called *Bernoulli Distribution* or Experiment. Suppose we have a probability θ, which is for example "coin lands on head" in the previous experiment. We can describe the Bernoulli Distribution as seen in equation 1.11 or in the equation 1.12 .

$$\text{Ber}(x \mid \theta) = \begin{cases} \theta, & \text{if } x = 1 \\ 1 - \theta, & \text{if } x = 0 \end{cases} \tag{1.11}$$

$$\text{Ber}(x \mid \theta) = \theta^x (1 - \theta)^{1-x}. \tag{1.12}$$

From this it follows, that $E(X) = \theta$ and $Var(X) = \theta(1 - \theta)$.

Binomial Distribution

Now suppose, that we toss the coin n times. If we assume, that the number of heads is $X \in \{0, ..., n\}$, then we can say $X \sim \text{Bin}(n, \theta)$, which means that X has a *Binomial Distribution*. The definition of the pmf is given in equation 1.13 and a plot in figure 1.4. Hereby $E(X) = \theta$ and $Var(X) = n\theta(1 - \theta)$.

$$\text{Bin}(k \mid n, \theta) = \binom{n}{k} \theta^k (1 - \theta)^{n-k}. \tag{1.13}$$

Multinomial Distributions

If we toss a coin, we have only two possible outcomes. What if we want to model a K-sided coin, or better, a dice? In this case we use the *Multinomial Distribution*. (see equation 1.14) A special case, where the number of dices is one, is called *Multinoulli Distribution*.

Name	n	K	x
Bernoulli	1	1	$x \in \{0, 1\}$
Binomial	-	1	$x \in \{0, 1..., n\}$
Multinoulli	1	-	$x \in \{0, 1\}^K, \sum_{k=1}^K x_k = 1$
Multinomial	-	-	$x \in \{0, 1, ..., n\}^K, \sum_{k=1}^K x_k = n$

Table 1.1: Summary of the binomial and related distributions

$$\text{Mu}(\text{x} \mid n, \theta) = \binom{n}{x_1 ... x_K} \prod_{j=1}^{K} \theta_j^{x_j}. \tag{1.14}$$

An overview of the distributions so far is shown in table 1.1

Gaussian Distribution

The *Gaussian Distribution*, also known as *Normal Distribution* is the most commonly used distribution. One of the reasons is the *central limit theorem*. It says, that the sum of many independent RVs is approximately normally distributed. Due to this fact many complex systems can be modeled as normally distributed noise. A second reason is, that out of all probability distributions with the same variance, the normal distribution encodes the *maximum amount of uncertainty* over the real numbers. So if there is no prior knowledge about the model, the normal distribution is the default choice. In figure 1.5 you can see two normal distributions. The calculation of the normal distribution is shown in equation 1.15.

$$\mathcal{N}(x; \mu, \sigma^2) = \sqrt{\frac{1}{2\pi\sigma^2}} \exp\left(-\frac{1}{2\sigma^2}(x - \mu)^2\right). \tag{1.15}$$

Exponential and Laplace Distribution

Especially in machine learning it is useful to have a sharp point at $x = 0$. To achieve this, the *Exponential Distribution* (see figure 1.6 and equation 1.16) or *Laplace Distribution* (see figure 1.7 and equation 1.17) is used very often.

$$f(x; \lambda) = \begin{cases} \lambda e^{-\lambda x} & x \geq 0, \\ 0 & x < 0. \end{cases} \tag{1.16}$$

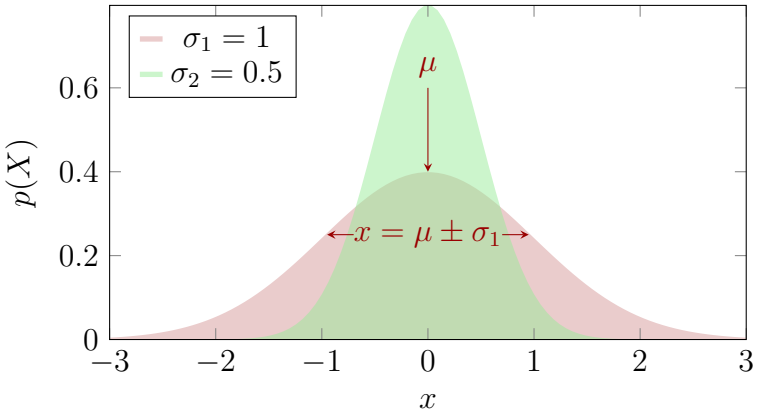


Figure 1.5: Normal Distribution

Two normal distributions, the red one with $\mu = 0$ and $\sigma = 1$ and the green one with $\mu = 0$ and $sigma = 0.5$

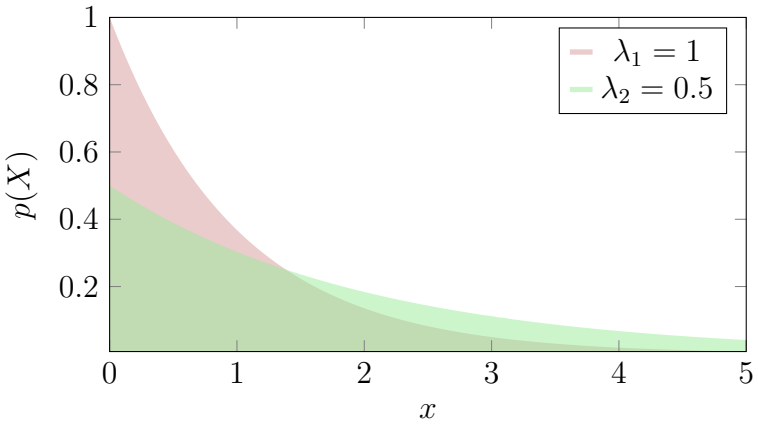


Figure 1.6: Exponential Distribution

Two Exponential distributions, the red one with $\lambda = 1$ and the green one with $\lambda = 0.5$

$$\text{Laplace}(x; \mu, \gamma) = \frac{1}{2\gamma} exp\left(-\frac{|\ x - \mu\ |}{\gamma}\right). \tag{1.17}$$

The Dirac Distribution

There are also cases, where you have to cluster all mass around a single point. In equation 1.18 you can see the definition of a probability density function using the Dirac delta function $\delta(x)$ and the Dirac delta function is defined in equation 1.19

$$p(x) = \delta(x - \mu) \tag{1.18}$$

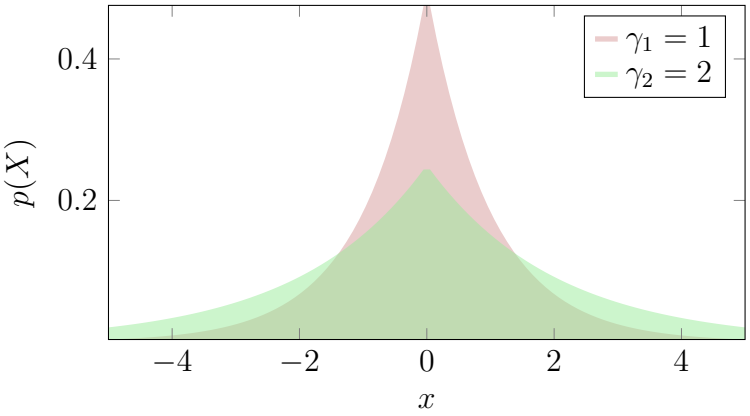


Figure 1.7: Laplace Distribution

Two Laplace distributions, the red one with $\mu = 0$ and $\sigma = 1$ and the green one with $\mu = 0$ and $sigma = 2$

$$\int_{-\infty}^{\infty} \delta(x)\mathrm{d}x = 1,\ \delta(x) = 0 \text{ if } x \neq 0 \tag{1.19}$$

1.2.3 Information Theory

Information theory is the theory of storage, communication and quantification of information, or more general, it deals with the questions of compression and transmission of data. Information Theory is based on the work of Shannon (1948). To give you a short overview, I will briefly introduce you to all the main concepts of information theory, structured like in Cover and Thomas (2006).

Entropy

The most famous concept of Shannon is the *Shannon Entropy* or short Entropy. It is a measure of the average uncertainty in the RV. The best way to understand the entropy is by explaining it with an example.

Example 1.2.1. Let

$$X = \begin{cases} a & \text{with probability } \frac{1}{2}, \\ b & \text{with probability } \frac{1}{4}, \\ c & \text{with probability } \frac{1}{4}. \end{cases} \tag{1.20}$$

Just suppose we want to determine the value of the RV X with the minimum amount of binary questions. If we ask "Is $X = a$?", it splits the probability in half. Now we

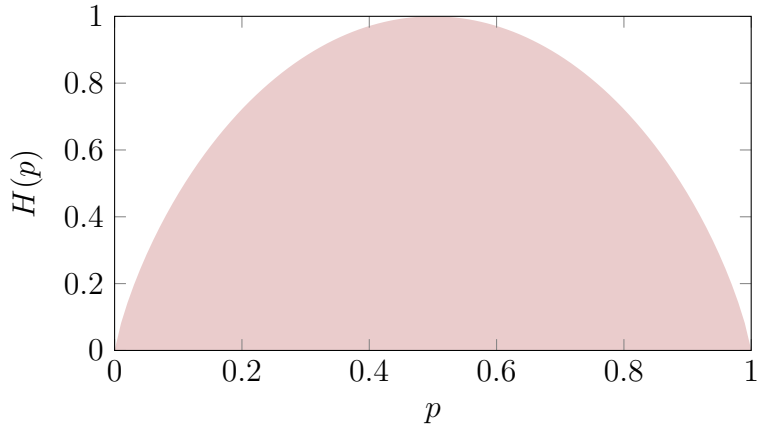


Figure 1.8: Entropy

Entropy of a RV with 2 values p and $p-1$

ask "Is $X = b$?". So the The resulting expected number of binary questions is 1.5. (equation 1.21) You can interpret the number of binary questions as uncertainty of the RV or in other words, the amount of information required in average to describe the RV.

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x). \tag{1.21}$$

Now consider, that you have only 2 values for a RV. If one value has the probability p, the other has the probability $1-p$. So you can write $H(X)$ like in equation 1.22. A diagram is shown in figure 1.8. As you can see the maximum Entropy in this Bernoulli experiment is, if the two values of the RV have the same probability to occur.

$$H(X) = -p \log(p) - (1-p) \log(1-p) \overset{\text{def}}{=\!=} H(p). \tag{1.22}$$

We can also compute the *Joint Entropy* (equation 1.23) and the *Conditional Entropy* (equation 1.24)

$$H(X,Y) = -\sum_{x \in \mathcal{X}} \sum_{y \in \mathcal{Y}} p(x,y) \log p(x,y). \tag{1.23}$$

$$H(Y \mid X) = \sum_{x \in \mathcal{X}} p(x) H(Y \mid X = x). \tag{1.24}$$

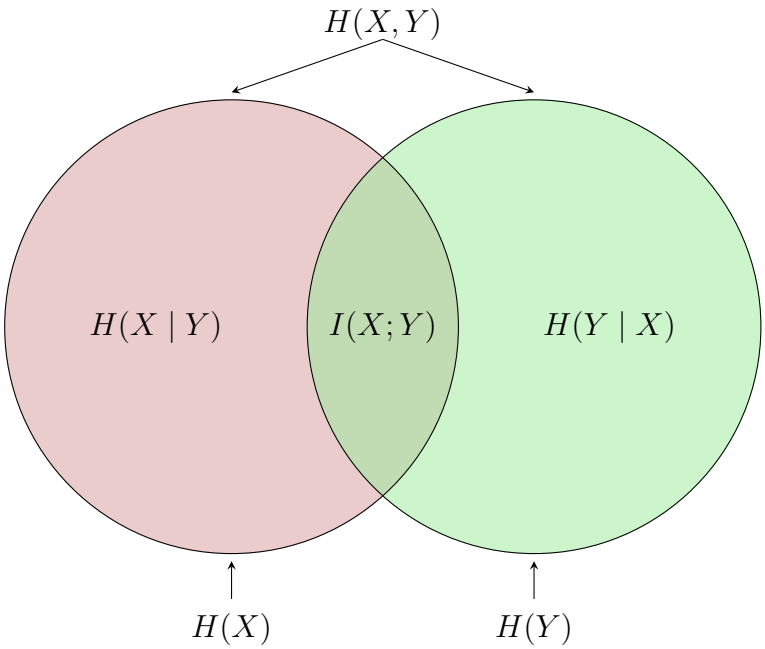


Figure 1.9: Mutual Information vs. Entropy

Relation between Mutual Information $I(X;Y)$ and Entropies $H(X), H(Y), H(X \mid Y), H(Y \mid X), H(X,Y)$

Relative Entropy & Mutual Information

If you want to compute the divergence between two distributions on the same RV, there is a concept called *Kullback-Leibler divergence* or *Relative Entropy*. The divergence can be explained as a measure of inefficiency, assuming the distribution is q, when the distribution is actually p. (see equation 1.25)

$$D(p \mid\mid q) = \sum_{x \in \mathcal{X}} p(x) \log \frac{p(x)}{q(x)}. \tag{1.25}$$

The *Mutual Information* $I(X;Y)$ is the Relative Entropy between the joint distribution $p(x,y)$ and the product distribution $p(x)p(y)$. You can see the relationship between Entropy and Mutual Information in figure 1.9 and the definition in equation 1.26.

$$I(X;Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} = H(X) - H(X \mid Y). \tag{1.26}$$

Markov Chains & Entropy Rate

If we have a sequence of RVs $\{X_i\}$, which is called a *stochastic process*, there can be a sort of dependence among those RVs. This process is characterized by the joint probability mass function seen in equation 1.27.

$$\Pr\{(X_1, ..., X_n) = (x_1, ..., x_n)\} = p(x_1, ..., x_n), (x_1, ..., x_n) \in \mathcal{X}^n \text{for} n = 1, 2, ... \tag{1.27}$$

A stochastic process is called *Markov Chain* for $n = 1, 2, ...$, if equation 1.28 holds.

$$\Pr(X_{n+1} = x_{n+1} \mid X_n = x_n, ..., X_1 = x_1) = \Pr(X_{n+1} = x_{n+1} \mid X_n = x_n) \forall x_1, ..., x_{n+1} \in \mathcal{X} \tag{1.28}$$

So we can say, that a Markov Chain only depends on the previous state, not on all other previous states.

If we now want to compute the Entropy for a stochastic process, we use equation 1.29. This is called *Entropy Rate*.

$$H(\mathcal{X}) = \lim_{n \to \infty} \frac{1}{n} H(X_1, ..., X_n). \tag{1.29}$$

Data Compression

Data Compression basically deals with assigning short descriptions (codewords) to frequent outcomes of the data and longer descriptions to less frequent ones. Nowadays this is a really important topic, since it is used mostly everywhere we deal with data. Let me now give a short overview of the theory behind Data Compression. Firstly we need to define the concept of *Source Code*. It is the mapping from one RV X to $\mathcal{D}^*$, where $\mathcal{D}$ is the alphabet of the code. The codeword is written as $C(x)$ and the length of the codeword as $l(x)$. Furthermore we need to define the term *prefix code*, which means, that no codeword is a prefix of an other codeword. But if the code is a *prefix code*, what are the lengths of the codewords? There exists an inequality, which holds, if the code is a prefix code and the lengths in this inequality are the lengths of the prefix code, which is called *Kraft Inequality* shown in equation , where D is the size of $mathcal D$. Now we want to find an *optimal code*. We do that, by finding minimizing the length $L = \sum p_i l_i$ over all lengths satisfying the

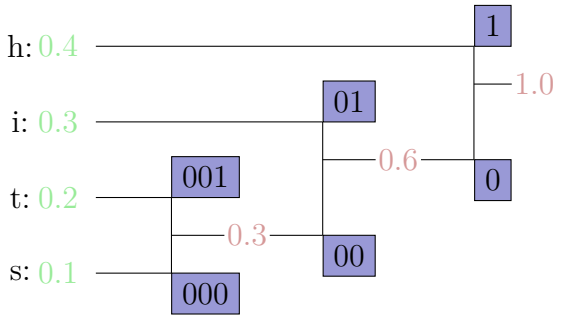


Figure 1.10: Huffman Coding Example

The green numbers on the left side are the probabilities how often each character in the input alphabet X occurs. Each step we add the lowest two probabilities. The first step is to add 0.2 and 0.1. After that we have three probabilities left: 0.4, 0.3 and 0.3. The next step is to add 0.3 and 0.3. Now we have 0.4 and 0.6 left. The solution is a sort of binary tree. To get the Huffman Code we just need to label each step in the tree as like by the blue numbers and we get the solution that h is 1, i is 01, t is 001, and s is 000.

Kraft Inequality. Now, how we construct *optimalcodes*? It can be constructed by an algorithm, discovered by Huffman (1952). To fully understand Huffman Codes, I will give a short example

Example 1.2.2. We want to compress the message "h hi hit hits" (without the whitespace). The code alphabet $\mathcal{D} = \{0, 1\}$ and the source alphabet is obviously $X = \{h, i, t, s\}$. The encoding of this message with the Huffman code is shown in figure 1.10. The solution of this encoding is $L = 0.4*1+0.3*2+0.2*3+0.1*3 = 1.9$ bits. If we have just used naive binary encoding for these 4 character the solution would be 2 bits. So we have saved 0.1 bits of space, just by using the Huffman Encoding!

1.3 Machine learning

1.3.1 Introduction

In 1959 Arthur Samuel published an article, which describes Machine Learning(ML) as the ability to make computers learn. He describes it by the game of checkers. The computer knows only the rules of the game, a sense of direction and a list of parameters. There is no explicit programmed solution of the problem. Samuel (1959) The basic problem in learning theory is to assign a given sample to one of at least two classes. Scholkopf and Smola (2001) Basically we can divide Machine

Learning in two big fields. *Unsupervised Learning* and *Supervised Learning*. There is also a third group, known as *reinforcement learning*. In the following subsections I will discuss the basics of machine learning (like in Goodfellow et al. (2016), Murphy (2012) and Bishop (2006)), show some examples, introduce you to the Deep Learning method and finally to interactive Machine Learning (iML).

Supervised Learning (SL)

SL is allocating samples, with the help of labeled training data. Simply speaking you use an algorithm to learn a mapping function from the input to the output.

Example 1.3.1. Lets demonstrate this on an example of *classification*:

Suppose we have dogs and cats. We know that a dog is barking(lower sound) and usually bigger as a cat. We also know that a the cat is meowing(higher sound) and usually smaller as a dog. Let this be our two parameters *height* and *sound*. In this example we demonstrate the two as vectors in 2 dimensional space. The idea of this simple binary classification algorithm is, that data points and their behavior are already known.(training data) If a new point is added, it is possible to classify him by its parameters, due to the prior processed training data. 1.11

In a mathematical description we can display this as follows:

$(x_1, y_1), ..., (x_n, y_n) \in \chi \times \{+1, -1\}$

The x_i in the formula are called patterns (also inputs) and the y_i are called labels, targets or outputs. In our example the x_i are the vector representation of an animal (cat or dog) and the y_i, if the representation is a dog or cat (usually -1 or $+1$ in binary classification). The χ is a collection of training data.

The second method besides *Classification* (example 1.3.1) is called *Regression*. Regression is basically the same as Classification, except that the output variable is continuous.

Unsupervised Learning (UL)

The big difference between UL and SL is, that the data points are not labeled. You have only input data and no corresponding output data. The goal of an unsupervised learning algorithm is to model the underlying structure in the data and consequently

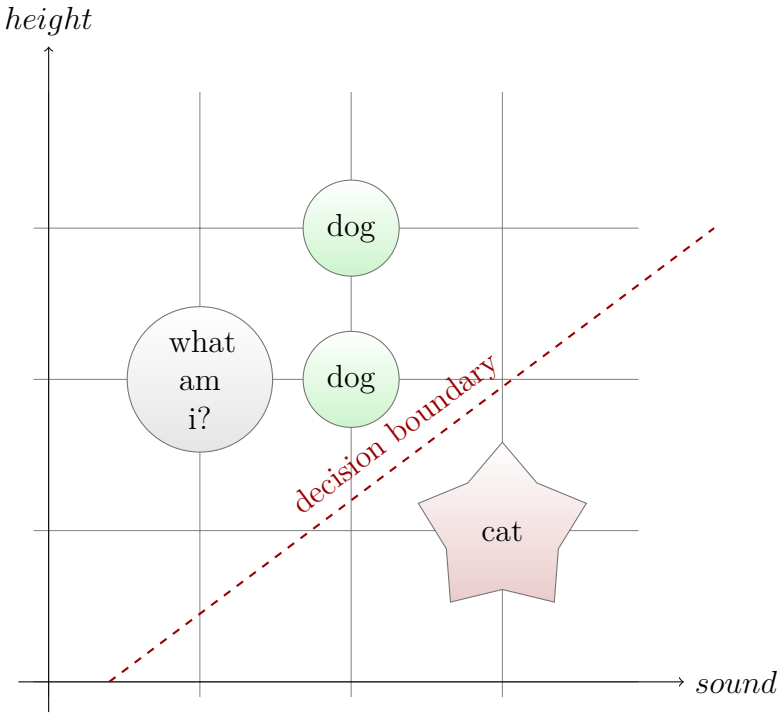


Figure 1.11: Supervised Learning

Idea of a binary classification algorithm with training data (red and green nodes) and test data (gray node). A dog is big and has a lower voice(green nodes). A cat is small and has a higher voice(red node). The decision boundary (dashed red line) divides the data into 2 classes

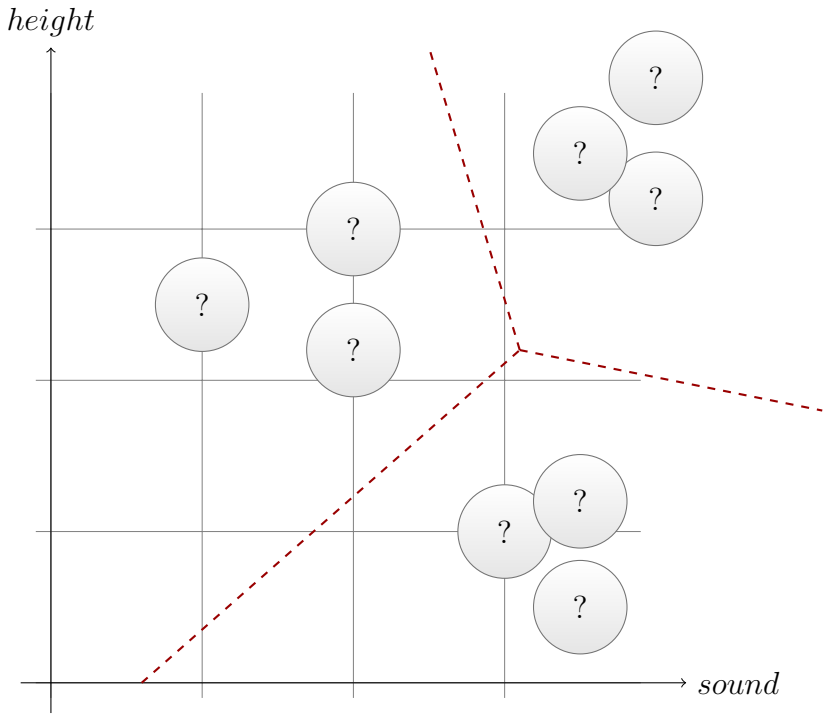


Figure 1.12: Unsupervised learning

Unlabeled data points (gray nodes) are separating into classes without prior knowledge.

learn more about it. As seen in figure 1.12 an unsupervised learning algorithm can separate data points into three classes. This learning process is very similar to the human one. Holzinger (2016)

$$(x_1), ..., (x_n) \in \chi$$

Reinforcement Learning (RL)

RL deals with the question, how to act or behave, by giving rewards or punishment. This is how humans learn. For example a baby learns to walk, if it falls, it can be seen as a sort of punishment and if it reaches a toy it can be seen as a sort of reward. The acting and work to accomplish the positive goal can be seen as learning output. In Silver et al. (2016) Deep Neural Networks(next section) trained with SL and RL mastered the game GO (with a percentage of 99.8%), which is viewed as the most classic challenging game for AI.

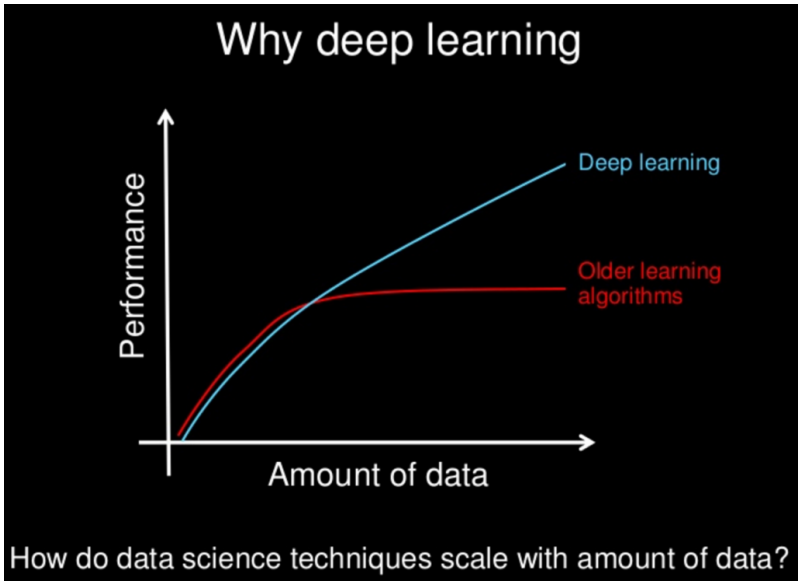


Figure 1.13: Basic concept of Deep Learning
(from https://www.slideshare.net/ExtractConf visited on 20.11.2017)

Deep Learning (DL)

DL is a very young field in Machine Learning, but already very wide spread in the industry. It is used in speech recognition, object detection, object recognition and many other domains. I am very sure, that everyone will use or is currently using DL in some way and thats the reason I want to describe it shortly.

> Deep Learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstractions. LeCun et al. (2015)

Andrew Ng (from Coursera, Chief Scientist at Baidu Research, founded Google Brain) provided a good first picture in his slides, which summarizes the basic concept Deep Learning. (see figure 1.13) He also mentioned, that almost all the value of deep learning today is through SL.

One of the quintessential DL models are the Deep Feedforward Networks, also called Multi Layer Perceptrons (MLP). MLPs can solve problems, which are not linearly separable (One Perceptron is only able to solve a linear separable problem. Minsky and Papert (1969)). This means, that the model can understand interactions between two input variables(Goodfellow et al. (2016) p.168)

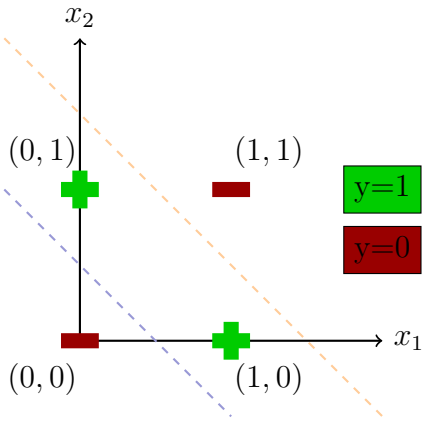


Figure 1.14: XOR nonlinear separable

You need 2 hyperplanes(blue and orange) to separate in a XOR. You can also see the output y for the inputs x_1 and x_2. (For example, if $x_1 = 1$ and $x_2 = 0$ the $y = 0$). If the point $(0, 0)$ would be a $+$, there would be only the orange hyperplane and the behavior would be like a negative AND. If the point $(1, 1)$ would be a $+$, there would be only the blue hyperplane and the behavior would be like an OR. So you have basically two linearly separable problems in one.

		x_1	
		0	1
x_2	0	$h_1 = \sigma(-2*0-2*0+3) = 1$ $h_2 = \sigma(2*0+2*0-1) = 0$ $y = \sigma(2*1-2*0-3) = \mathbf{0}$	$h_1 = \sigma(-2*1-2*0+3) = 1$ $h_2 = \sigma(2*1+2*0-1) = 1$ $y = \sigma(2*1+2*1-3) = \mathbf{1}$
	1	$h_1 = \sigma(-2*0-2*1+3) = 1$ $h_2 = \sigma(2*0+2*1-1) = 1$ $y = \sigma(2*1+2*1-3) = \mathbf{1}$	$h_1 = \sigma(-2*1-2*1+3) = 0$ $h_2 = \sigma(2*1+2*1-1) = 1$ $y = \sigma(2*0-2*1-3) = \mathbf{0}$

Table 1.2: Output of the perceptrons

Example 1.3.2. A classic non linear separable problem is learning the XOR function. This is a classic example to demonstrate a MLP. Figure 1.14 shows the classical XOR and why it is not linearly separable. In figure 1.15 you can see the MLP to solve this problem (with already trained weights). If we combine the behaviors of each single perceptron, we get the desired XOR (shown in table).

1.3.2 Interactive Machine Learning

The unsupervised and supervised learning approach is usually used in context of automatic machine learning (aML). AML is an approach, where algorithms automatically learn from data, without any human intervention. Roughly speaking, the

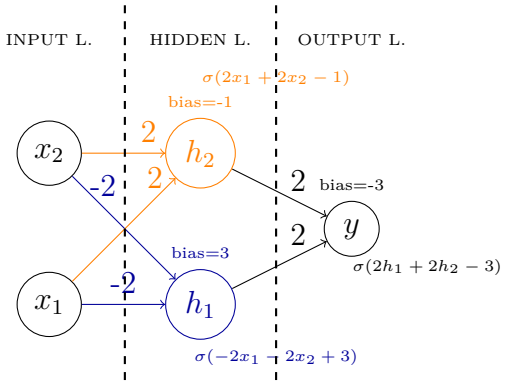


Figure 1.15: MLP for XOR function

This figure is inspired by Lavrenko (2015).
The layer on the left is called *input layer* with the inputs x_1 and x_2, both with possible values 0 and 1. The layer in the middle is the *hidden layer* with 2 perceptrons h_1 and h_2. h_1 simulates the behavior of a logical negative AND and h_2 the behavior of a logical OR. The left layer is called *output layer*. It has one perceptron with the behavior of a logical AND. The perceptrons are connected with trained weights and have a trained bias. Each perceptron output is a summation of the input weights (multiplied by the input value) and the bias, normalized by a sigmoid function (to get a suited output).

learning phase of a such algorithms can be seen as a *"black box"*, because you can't see what's happening in there. Examples for aML approaches are speech recognition, recommender systems and autonomous vehicles. In the previous section I have also mentioned the playing algorithm for the board game GO, which is also an example for aML.

But as soon the complexity of a problem increases, combined with rare events and a small amount of training data, the aML approach is hard to implement. (Holzinger et al. (2017)) Especially in health care (eg. protein folding, subspace clustering) aML will most likely deliver unsatisfying results. Another big issue is the lack of transparency. There is usually no way to tell, *why* a certain decision has been made, which can cause privacy and also legal issues. Under the new European General Data Protection Regulations(GDPR) taking effect on June, 1st, 2018, customers are given a right e.g. to delete their user-specific data on request. (Malle et al. (2017)) When there is no way to tell why decisions has been made and how they are influenced by user-specific data, it can cause major issues with todays' black box aML algorithms.

By taking the human in the loop, such that he/she can influence decisions during the learning phase of a ML algorithm, the previous mentioned black box turns into a

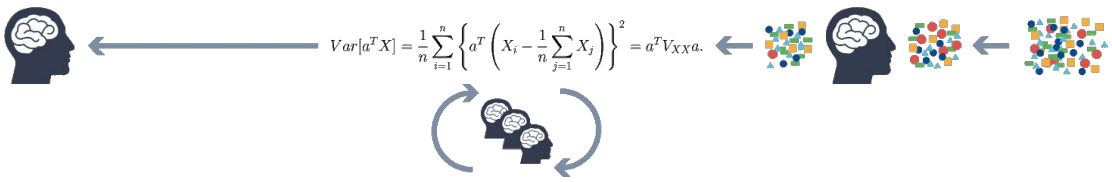


Figure 1.16: The iML approach
(from Holzinger et al. (2017))
Humans are involved in preprocessing of the data and in the algorithm learning phase, making the classical *black box* approach *glass-box*.

so called glass box. If you include a human in the decision finding process, you need to know at any time the state of current computations, to add human interactions, when needed. That makes the algorithm usually fully transparent at any time. Another big advantage is, that interactive approaches can still perform, when there are very few examples or when we deal with complex data. The key to this behavior is human intuition. Humans can outperform current ML algorithms, because they have the ability to interpret complex patterns in a fast way. Akgül et al. (2011) suggested in the context of Content-Based Image Retrieval an expert-in-the-loop approach to consider the experts high-level knowledge and judgment. All in all it would be a pity not to value humans expertise in complex decision making, so the term *interactive Machine Learning*(iML) was introduced:

Algorithms that can interact with agents and can optimize their learning behavior through these interactions, where the agents can also be human. (see figure 1.16) (Holzinger (2016))

2. Theoretical Background

In the previous chapter I mentioned, that there is no possibility in aML to interact directly with the ML algorithm at runtime. In this chapter I will take a look on some ML algorithms. Especially I will point out *Ant Colony Optimization* (ACO). I will explain, how these algorithms can be modified to transform them from an aML to an iML approach. The focus hereby is to get a good approximated solution for the the *Traveling Salesman Problem* (TSP). I will also discuss the importance of *Gamification* and how it's capable to reach a high amount of participation.

2.1 Traveling Salesman Problem (TSP)

The TSP is in the set of **NP**-hard problems, which means, that there is no polynomial algorithm to solve this problem to optimality. The problem is explained by a tour of a Salesman, who needs to visit a certain amount of cities and wants to minimize the route through all these, ending with the starting city.

2.1.1 Mathematical description

The TSP can be represented by a graph $G = (N, A)$ with N as the set of $n = |N|$ nodes and A, the set of arcs(edges) fully connecting all nodes. Each edge $(i, j) \in A$ has a cost measure d_{ij} which represents the distance between the cities i and j.

The problem hereby is to find the minimum length of a Hamilton circuit.[1] A Hamilton circuit is a tour through all nodes of G, visiting each node exactly one time. There are two sorts of TSPs: symmetric and asymmetric TSPs. A symmetric TSP the cost measure is not directed. This means $d_{ij} = d_{ji}$. In an asymmetric TSP

[1] named by the Irish scientist Sir William Rowan Hamilton (1806-1865)

$d_{ij} \neq d_{ji}$. I represented solutions of TSP instances in this work as a permutation of city indices, where the first index is added at the end, to simulate a cyclic tour. In a graph with four nodes a possible solution representation is e.g. $2, 3, 1, 4, 2$

2.1.2 Solution finding

Since TSP is a **NP**-hard problem, finding an exact or good solution is a very big field in the current research. Over the years a lot of exact and "suboptimal" algorithms for solution finding were published and some of them I will introduce you shortly.

Exact Algorithms

The output of an exact algorithm is always the best possible solution for the problem the algorithm is made for.

The simplest algorithm for solving the TSP is to try all permutations and see which one is the best one. It's very trivial, that the runtime of this approach is $\mathcal{O}(n!)$.

One of the early algorithms is the dynamic programming approach by Held and Karp (1962) and also Bellman (1962), which is known as **Bellman-Held-Karp algorithm**. This algorithm uses a property of TSP, which says, that *every sub-path of a path of minimum distance is itself of minimum distance.* Basically it says, that you are able to reuse previous computations of subproblems in combination with new computations, to construct the solution. This algorithm has a runtime of $\mathcal{O}(n^2 2^n)$.

Also mentionable is the Concorde TSP Solver[2] written by David Applegate, Robert E. Bixby, Vašek Chvátal, and William J. Cook. It uses a *Branch and Cut* approach (published by Padberg and Rinaldi (1991)) and the currently largest TSP instance, solved by Concorde counts *85,900* cities. (Con (2016))

Heuristic and Approximation Algorithms

Sometimes an exact solution for a TSP is not needed and therefore it is enough to find an approximated solution. Following you can see a short introduction to some

[2]http://www.math.uwaterloo.ca/tsp/concorde/index.html visited on 07.06.2017

Approximation and Heuristic algorithms, structured like in Nilsson (2003). First we will take a look at algorithms, which are used for *tour construction*.

Nearest Neighbor (NN) algorithm: The most trivial approximation algorithm is the NN algorithm, where you iterate through all the points, always choosing the nearest, unvisited point and add him to the tour. The constructed tour by the algorithm is a *Hamiltonian tour* with N nodes. Note, that the path changes, if you choose a different starting point. The average solution, using the NN algorithm is approximately 25% higher, than the optimal solution. You can also say, that the NN algorithm keep its' tours within 25% of the *Held-Karp lower bound(HKLB)* (Johnson and McGeoch (1997)), which is a measurement of the performance of a TSP heuristic. (averages about 0.8% below the optimal tour length) (Johnson (1996)) The runtime of the algorithm is $\mathcal{O}(n^2)$.

Greedy algorithm: The Greedy algorithm is actually very similar to the NN algorithm. It repeatedly adds the shortest edge to the tour, if the edge does not create a cycle with less than N edges or increases the degree of a node to more than 2. The runtime of the algorithm is $\mathcal{O}(n^2 \log_2(n))$ and it is usually in a 15-20% above the HKLB.

Christofides' algorithm: Another approximation is the Christofides' algorithm, which combines the minimum spanning tree with minimum-weight perfect matching. It gained popularity, because it has a constant worst-case performance of 3/2 and is named after his inventor Professor Nicos Christofides. Christofides (1976) The runtime is $\mathcal{O}(n^3)$ and its in average within 10% above the HKLB.

The following algorithms are used for *tour improvement*.

2-opt, 3-opt and k-opt: *2-opt* and *3-opt* algorithms (Lin (1965)) are very often used in tour improvement of a TSP. A 2-opt algorithm removes two edges from the tour and reconstructs it in the only one other possible way. A 3-opt algorithm removes 3 edges and can be seen as a combination of two or three 2-opt moves. A 3-opt move can construct two other valid tours. The algorithm stops, if no permutation can improve the tour. A 2-opt algorithm results in

a tour 5% above the HKLB in average, where a 3-opt algorithm results in a tour 3% above the HKLB in average. There are also algorithms, that removes more than 3 edges, which are called *k-opt* algorithms.

Lin-Kernighan: Lin-Kernighan (Lin and Kernighan (1973)) is a dynamic k-opt algorithm, which makes it very complex. Each iteration the most suitable k is selected. Lin-Kernighan is within 2% of the HKLB.

Tabu-Search: A Tabu-Search (Glover (1989)) is a neighborhood search (searches among neighbors, to find a better one; in a TSP usually 2-opt move), which allows moves with negative gain, to avoid stucking in a local optimum. To prevent ending up in circles, it uses tabu-lists, where illegal moves are saved. Unfortunately the runtime with $\mathcal{O}(n^3)$ is very high. However, it performs better than the default 2-opt search.

Genetic Alorithms: Genetic algorithms simulate the evolutionary process of nature. Basically they start with a initial population. In every iteration some or all candidates of the population produce offspring or go through a mutation process. By selecting the fittest candidates in an iteration (best candidates) to reproduce or mutate, the overall fitness will raise. Genetic algorithms can produce better solutions than LK algorithms, but unfortunately they are very slow in comparison to other algorithm families.

Now, we have discussed algorithms for constructing and improving tours in a TSP, I want to emphasize another solution for TSP: Ant Colony Optimization (ACO). I used and experimented with it in my practical part of this work.

2.2 Ant Colony Optimization

2.2.1 Ants

To get a clear understanding of Ant Colony Optimization (ACO) it is important to take a look at some basics of the ant social behavior. In the book "The insect societies" by WILSON (1971) behaviors of ants has been studied extensively. Ants have rudimentary sights, a limited visual and auditory communication and are not

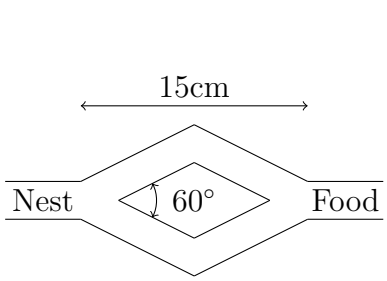
capable to achieving complex tasks on their own. Although they are capable of reaching impressive group results, like nest building and defense, nest temperature regulation, forming living bridges and cooperatively carrying large items, sorting brood and food items and *searching for food sources*. Especially the searching for food or getting it, with the lowest possible amount of energy can be seen as natural optimization. Ants will always take the shortest path to a food source. The key to this behavior is *Stigmergy*, which was first defined by Pierre-Paul Grassé in Grassé (1959). He said, that individuals can interact with one and another through modifications of their environment. Stigmergy in reference to ants is accomplished by chemicals which are called pheromones. If an ant moves from point A to point B it deposits a certain amount of pheromones on the trail. By sensing the pheromones on the trail other ants can follow the path and reach certain points of interest. (e.g. a food source) They mark the path between the food source and the nest with this pheromones. This collective behavior, of depositing and following pheromone trails, is the main inspiration for ACO.

2.2.2 Double Bridge Experiments

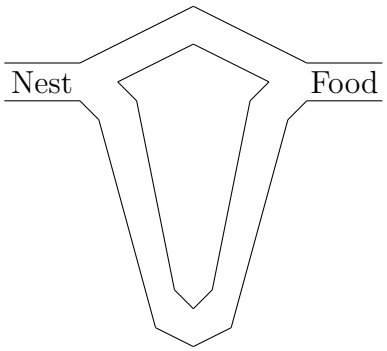
The trail-laying and follow behavior of ants was investigated by many researchers in the past. To give a short insight into their work I will explain two popular experiments.

Experiment 1

The first experiment from Deneubourg et al. (1990) shows a bridge with two branches of equal length. (see figure 2.1a) Ants have to choose either the upper or the lower path to the food source. After they reach their goal, they return to the nest. Initially the percentage of ants, who choose the upper branch is nearly the same to ants who choose the lower one. After a specific time nearly all ants prefer a certain branch. This behavior can be explained by pheromones. Initially there are no pheromones on the branches. Ants have no preference, which branch to choose. Because of random fluctuations, some ants select one branch over the other and deposit their pheromones. This leads to higher pheromone concentration on one branch and following ants will choose this particular one with a higher chance. Finally the



(a) Branches of equal length



(b) Branches of different length

Figure 2.1: Double Bridge Experiments
based on Dorigo and Stützle (2004)

ants converge to a branch. This behavior of ants is a an example for the previous mentioned *Stigmergy*.

Experiment 2

The second experiment from Goss et al. (1989) shows the double bridge with different branch length. (see figure 2.1b) One branch is twice as long as the other. Like in the experiment before, the ants initially choose the branch randomly - approximately half of the ants select the shorter one and the other half the longer one. However, there is a big difference to the first experiment. Due to the shorter length of one branch, the ants much faster reach the food source and start to returning to the nest. There will be a much higher pheromone level on the shorter branch, because the initial ants of the shorter branch are returning long before the ants of the longer branch. After some time, nearly all ants will choose the shorter path. Compared to the first experiment, the random fluctuations have a lower impact. But there are still ants, which choose the longer branch. This behavior is described as *path exploration*. But what happens if the shorter branch is added after the ants converge to the longer one? The authors also tested this behavior. They added the shorter path after 30 minutes. But in this case, the ants won't change their decision, because of the slow evaporation rate of pheromones. This rate prevents the ant colony to "forget" the suboptimal branch.

2.2.3 ACO Algorithms for TSP

The Ant System was first introduced by Marco Dorigo in 1992 and was called Ant System (AS) by Dorigo (1992). In this subsection I will explain the Ant System at it was in 1992 and modifications of this algorithm - the Ant Colony System (ACS) by Dorigo and Gambardella (1997) and the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System by Stützle and Hoos (2000). I will also compare these algorithms and introduce some changes to add interactiveness, to reach the iML character. (Holzinger et al. (2016))

Ant System (AS) by Dorigo (1992)

Consider, that there is a graph with several nodes and edges. Every edge has a cost measure $\delta(r, s)$ and also a *pheromone* measure $\tau(r, s)$. The $\tau(r, s)$ is updating during the runtime by the previous mentioned artificial ants. Each ant generates a complete tour, including all nodes. The nodes are selected by a *probabilistic state transition rule*. This rule says, that an ant will most likely choose a path with a high pheromone measure $\tau(r, s)$ and a low cost measure $\delta(r, s)$. Once all ants have finished the tour (iteration) a *global pheromone updating rule* will be applied. This rule says, that a fraction of pheromones evaporates on all edges and each ant deposits an amount of pheromones on their edges. (see algorithm 1)

Algorithm 1 AS algorithm pseudocode

```
1: Initialize trail
2: while stopping criteria not satisfied do
3:     position each ant in starting position
4:     repeat
5:         for each ant do
6:             choose next node by state transition rule
7:         end for
8:     until every ant completed the tour
9:     perform global pheromone updating rule
10: end while
```

In equation 2.1 we see the **state transition rule** (probability with which an ant **k** in point **r** chooses to move to point **s**)

$$p_k(r,s) = \begin{cases} \frac{[\tau(r,s)]^{\alpha} \cdot [\eta(r,s)]^{\beta}}{\sum_{u \in J_k(r)} [\tau(r,u)]^{\alpha} \cdot [\eta(r,u)]^{\beta}}, & \text{if } s \in J_k(r) \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

where

- $\boldsymbol{\tau}$ is the pheromone measure,
- $\boldsymbol{\eta = 1/\delta}$ is the inverse of the cost measure,
- $\boldsymbol{J_k(r)}$ is the remained set of nodes, needed to be visited by the current ant $\boldsymbol{k}$ positioned on the node $\boldsymbol{r}$,
- $\boldsymbol{\beta}$ and $\boldsymbol{\alpha}$ indicate the importance between pheromone measure and cost measure.

In general this state transition rule says, that we prefer edges with lower costs and a high amount of pheromones. After defining the *state transition rule*, we can move on to the **global pheromone updating rule**, illustrated in equations 2.2 and 2.3.

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \sum_{k=1}^{m} \Delta\tau_k(r,s) \tag{2.2}$$

where

$$\Delta\tau_k(r,s) = \begin{cases} \frac{1}{L_k}, & \text{if } (r,s) \in \text{tour done by ant } k \\ 0, & \text{otherwise} \end{cases} \tag{2.3}$$

and

- $\boldsymbol{0 < \rho < 1}$ is a pheromone decay parameter,
- $\boldsymbol{L_k}$ is the length of a tour, performed by an ant k,
- $\boldsymbol{m}$ is the number of ants

Basically the equation in 2.2 says, that there is a greater allocation of pheromones on a shorter tours. This behavior can be seen as a type of reinforcement learning schemes.

Ant Colony System (ACS) by Dorigo and Gambardella (1997)

Now we know, how the basic AS algorithm works, we can take a look at an extension of AS, the *ACS*. There are three improvements or changes between the AS and the ACS. The AS algorithm is explained above. There are changes in *state transitioning* and *global updating of pheromones*. Furthermore there is also a *local pheromone updating rule* introduced after each "movement" of an ant. (see algorithm 2)

Algorithm 2 ACS algorithm pseudocode

```
1:  Initialize trail
2:  while stopping criteria not satisfied do
3:      position each ant in starting position
4:      repeat
5:          for each ant do
6:              choose next node by state transition rule
7:              perform local pheromone updating rule
8:          end for
9:      until every ant completed the tour
10:     perform global pheromone updating rule
11: end while
```

Let us take a closer look into the changes:

1. **Tour construction**: The state transition rule now provides a *way of balance*. The balancing is accomplished between the exploration of new edges and the exploitation of *a priori*/accumulated knowledge about the route. In other word, as seen in equation 2.4, the tuning of the parameter q_0 allows the balance between concentrating on the best-so-far solution or exploring other tours.

$$s = \begin{cases} \arg\max_{u \in J_k(r)} \left\{ [\tau(r,u)] \cdot [\eta(r,u)]^{\beta} \right\}, & \text{if } q \leq q_0 \\ S, & \text{otherwise} \end{cases} \tag{2.4}$$

   where

   - $\boldsymbol{q}$ is a random variable uniformly distributed in [0,1],
   - $\boldsymbol{q_0}$ $(0 \leq q_0 \leq 1)$ is a parameter,
   - $\boldsymbol{S}$ is a random variable selected accordingly to the probability distribution in 2.1.

2. **Global Pheromone Trail Update**: Another big difference is, that only the one ant, with the best tour is allowed to add pheromones after each iteration, which makes the search more directed. It has also the advantage, that the complexity of the pheromone update is reduced from $\mathcal{O}(n^2)$ to $\mathcal{O}(n)$. (see equations 2.5 and 2.6)

$$\tau(r,s) \leftarrow (1-\rho) \cdot \tau(r,s) + \Delta\tau(r,s) \tag{2.5}$$

where

$$\Delta\tau(r,s) = \begin{cases} \frac{1}{L_{gb}}, & \text{if } (r,s) \in \text{global best tour} \\ 0, & \text{otherwise} \end{cases} \tag{2.6}$$

and

- $\mathbf{0 < \rho < 1}$ is the pheromone decay parameter,
- $\boldsymbol{L_{gb}}$ is the length of the globally best tour.

3. **Local Pheromone Trail Update**: In addition to the global pheromone update rule, which is described above, ACS is also using a local pheromone update rule, that is applied after an ant is traveling one city ahead. (see equation 2.7) The purpose of this rule is, that it dynamically changes the desirability of edges. If an ant visits an edge, it becomes less interesting for following ants and this causes a better spreading of ants. There is a lower probability that ants are searching in narrow neighborhood of the best previous tour.

$$\tau(r,s) \leftarrow (1-\xi) \cdot \tau(r,s) + \xi \cdot \Delta\tau(r,s) \tag{2.7}$$

where

- $\mathbf{0 < \xi < 1}$ is the pheromone parameter for evaporation and deposition,
- $\boldsymbol{\Delta\tau(r,s)}$ is a parameter.

Dorigo and Gambardella (1997) experimented with different values for the parameter $\Delta\tau(r,s)$. The outcome was, that a good choice is $1/(n \cdot L_{nn})$,

where n is the number of cities and L_{nn} is the length produced by the nearest neighbor heuristic. (see section 2.1)

$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) by Stützle and Hoos (2000)

Probably one of the most studied ACO algorithms is the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. There are four changes regarding the initial AS algorithm. First, it exploits the best tour of an ant, like the ACO. Second, it limits the excessive growth of the pheromones on good tours (which in some cases is suboptimal), by adding upper and lower pheromone limits τ_{min} and τ_{max}. Third, it initializes the pheromone amount of each edge to the upper pheromone limit τ_{max}, which increases the exploration of new tours at the start of the search. Finally each time, if there is a stagnation in some way or no improvement of the best tour for a particular amount of time, it reinitialize the pheromone trails.

1. **Update of Pheromone Trails**: The update of pheromones is nearly identical to the global updating rule of ACS. (see equation 2.5 and 2.5) A small difference is, that the $\mathcal{MM}$AS was mainly tested with the iteration-best tour.

2. **Pheromone Trail Limits**: $\mathcal{MM}$AS uses lower and upper pheromone trail limits τ_{min} and τ_{max} to avoid search stagnation. The reason for this is the limitation of selection probabilities. The possibility p_{ij} for choosing a city j, if the ant is currently in city i should be 1.0, if there is only one city left to choose. There is a proof, that τ_{max} is bounded to $1/(\rho L^{opt})$, where L^{opt} is the length of the optimal route. This implies, that every time a new best-so-far tour is found, the value of τ_{max} needs to be updated. The value of τ_{min} is set to a value related to τ_{max}. (see equation 2.8)

$$\tau_{min} = \frac{\tau_{max}\left(1 - \sqrt[n]{p_{best}}\right)}{(avg - 1)\sqrt[n]{p_{best}}} \tag{2.8}$$

   where

   - $\boldsymbol{p_{best}}$ is the probability, that if the algorithm converges, the best solution found is constructed.
   - $\boldsymbol{avg}$ is $n/2$ (n is the number of cities).

3. **Pheromone Trail Initialization**: The pheromones are initially set to τ_{max}, which is usually accomplished by setting it to an arbitrary high value. After the first iteration of the algorithm the value will be set to τ_{max}, due to the behavior described in point 2. Tests with setting the value to τ_{min} led to unsatisfactory results.

4. **Pheromone Trail Reinitialization**: It is also called smoothing of pheromone trails (PTS). After convergence or very close to it, the mechanism increases the amount of pheromones on paths. This has the great advantage, that trails with low probability will be considered again. PTS is especially interesting for runs with a high iteration count and makes the $\mathcal{MM}$AS less sensitive to the choice of τ_{min}.

$$\tau_{ij}(t) = \tau_{ij}(t) + \delta(\tau_{max}(t) - \tau_{ij}(t)) \tag{2.9}$$

where

- $\mathbf{0 < \delta < 1}$ is a parameter for the degree of smoothing

Increasing performance of ACO

It should be clear, that increasing performance of ACO is the most important aspect, since these optimizations are dealing with **NP**-hard problems. In ACS, a local search routine is used after each ant has completed their tour. This most likely increase the quality of the outcome, because ACO solution construction uses a different neighborhood, than local search routines. Maye its also important to mention, that for speeding up local search routines nearest neighbor lists are used.

2.2.4 Parameter settings

An Ant algorithm only delivers satisfying solutions in a good amount of time, if the algorithm parameters are selected properly. To give a short insight in the parameter settings of the previous described algorithms AS, ACS and $\mathcal{MM}$AS take a look at the table 2.1. (from Dorigo and Stützle (2004) p.71) These settings are found over a significant set of TSP instance. There are also individual instances, where other parameters may be lead to better performance.

ACO algorithm	α	β	ρ	m	τ_0	p_{best}	ξ	q_0
AS	1	2 to 5	0.5	n	m/L^{nn}	-	-	-
MMAS	1	2 to 5	0.02	n	$1/\rho L^{nn}$	0.05	-	-
ACS	1	2 to 5	0.1	10	$1/nL^{nn}$	-	0.1	0.9

Table 2.1: Parameter settings for ACO algorithms
Please keep in mind, that for individual instances, different settings may be better!

Remark: Most scientific papers are using the TSPLIB. (see Reinelt (1991)) It is a collection of TSP problems, with best solutions and is mainly used for testing algorithms.

2.3 Gamification

2.3.1 Definitions

There are several definitions of Gamification. I will give you a short overview of the - in my opinion - most popular ones.

Definition by Deterding et al.

One of the first and most cited definitions of Gamification is from Deterding et al. (2011) and describes it as the *"use of elements of game design in non-game contexts"*. Following I will take a closer look to the definition and define the terms "game", "design", "elements" and "non-game context", which are a part of the definition.
This part is strongly related to the work of Deterding et al. (2011). The classification overview is given in figure 2.2.

1. **Game** There is a difference between the terms "game" and "play". If we see it as sets, you can say, that game is a subset of play and vice versa. It depends on the framing of these words. We stick to the definition, that "play" is a broader, looser category, containing "game".

   Let me demonstrate this on an example:

   If a dog is catching a ball, playing with other dogs or if a child is playing with toys, we can consider this behavior as "play". If we stick to certain rules and

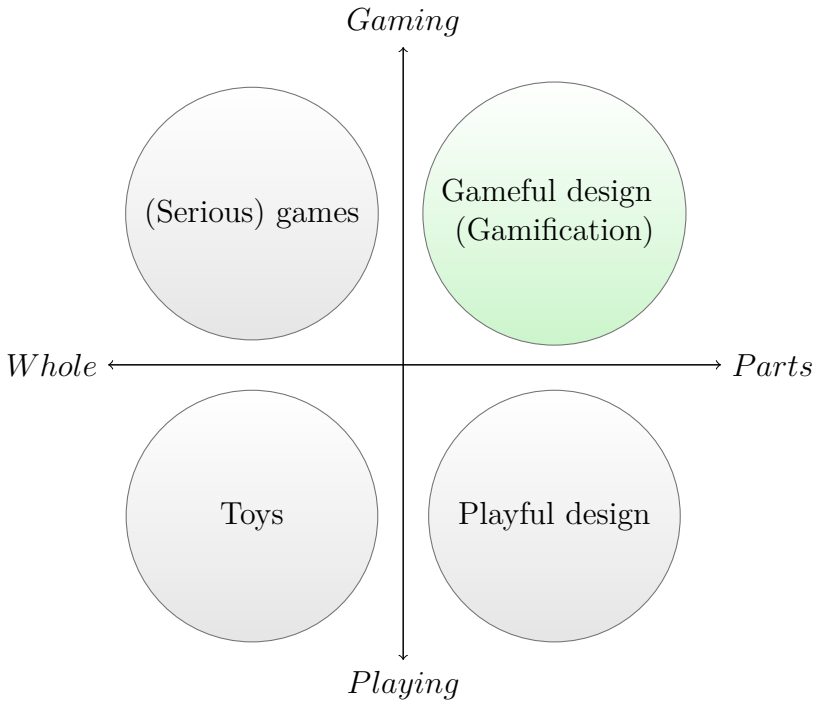


Figure 2.2: Classification of Gamification by Deterding et al.
Contrast between Serious games, gameful design, Toys and playful design in two dimensions of playing/gaming and parts/whole.

compete to win, as f.e. in soccer, we can consider this as "game".

In the book "Rules of Play" (Salen and Zimmerman (2003)) "game" is defined as following:

A game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome.

Now as we know the definition of "game" and we can continue with the definitions of the new term "gamefulness" (term by McGonigal (2011a)). It describes the qualities of gaming. Due to this definition, "gameful design" is "designing for gamefulness" by using "game design elements". Gamification is usually consistent with "gameful design".

Often we can't differentiate between a gamified application and a serious game. It depends on the point of view. Deterding et al. conclude the characteristics of gamified applications, that they afford a more fragile, unstable "flicker" of experiences and enactments between playful, gameful, and other, more-instrumental-functionalist modes, compared to games. This sounds very complicated, but basically it says, that a gamified application is not yet grown to

Level	Description	Example
Game interface design patterns	Common, successful interaction design components for a known problem in a context, including prototypical implementations	Highscore, badge, level
Game design patterns and mechanics	Commonly reoccurring parts of the design of a game that concern gameplay	Limited resources, time, turns
Game design principles and heuristics	Evaluative guidelines to approach a design problem or analyze a given design solution	Enduring play, clear goals, variety of game styles
Game models	Conceptional models of the components of games or experience	MDA (), GEGE
Game design methods	Game design-specific practices and processes	Playtesting, playcentric design, value conscious game design

Table 2.2: Levels of Game Design Elements
from Deterding et al. (2011)

a game.

2. **Element**

   Deterding et al. point out, that no typical element on its' own make up a game. You need a set of game elements to build one. They suggest, that in the context of Gamification, *game Elements can be defined as elements, that are characteristic to games (elements, that are found in most games, readily associated with games, and found to play a significant role in gameplay).*

3. **Non-Game Context** Deterding et al. simply sum up *Non-Gaming context* as context, that is not normally expected for entertainment. They won't give any further limitations, since there is no clear advantage to give some.

4. **Design** In table 2.2 you can see the different levels of abstraction of game design elements by intensive study from previous researchers. They suggest to include all of these levels in the definition of Gamification.

Definition by Huotari and Hamari

Huotari and Hamari (2012) define Gamification as *a process of enhancing a service with affordances for gameful experiences in order to support user's overall value creation.* It is a much more general definition than the one by Deterding et al. (2011), with some differences:

1. **There are no game elements, or if there are, they are not unique to games as we understand them.** They argue, that Gamification is a technique to provide "gameful experiences" rather than the usage of game-like elements. That's because a large number of systems (e.g. stock exchange dashboards, decision support systems and other services and systems, that have i.e. levels, points and progression metrics) can be categorized as games or gamified systems and according to previous definition, they are "non-gamifiable". This leads them also to a second difference:

2. **There are no non-game contexts.** They argue, that the dichotomy between games and non-games does not necessarily exist (and if it does, it's rather subjective). That makes the non-game and game contexts indistinguishable.

3. **One can't create "gameful experiences".** They argue, that there is no guarantee, that a game designer accomplishes the experience as intended and therefor the gamifier can merely provide *affordances*, such as game mechanics, for gameful experiences.

4. **The goal of gamification is first and foremost to afford gameful experiences.** Because there are several definitions for Gamification with a limited set of goals, they suggest, that a broader set of goals is in order to make the definition domain-independent.

Definition by Zichermann et al.

In Zichermann and Cunningham (2011) they stick with the definition for Gamification as *the process of game-thinking and game mechanics to engage users and solve problems.* It is an rather easy and good understandable definition with the "limited" goal (regarding to Huotari and Hamari (2012)) of solving problems. Nevertheless

they think, that this is a really powerful framework for any problem, that can be solved through influencing human behavior and motivation.
In this work I will stick with this definition, because it is the most fitting one for the practical part of my master thesis.

2.3.2 Gamification by Design

This structure of this part is strongly based on the book "Gamification by Design" - Zichermann and Cunningham (2011).

In the previous section we mainly defined the term Gamification. Now I would like to emphasize, why there is such an enjoyment in playing video games and what kind of people are playing these.

But first I would like to mention a study by a data analytics company named "Nielsen Entertainment". The study [3] shows that in the year 2017 64% of the U.S. population (older than 13 years) are gamers. (2012 it was at 58%) I remember back in the day, when I started with gaming, I felt like an exception. I am very sure, that in future everyone will be in contact with some sort of Gaming, simply because we will need it to learn for or do things in our 21st century lives!

Motivation

The theory of motivation by Lazzaro (2004) says, that fun is divided into 4 big categories: *Hard Fun, Easy Fun, Altered state fun and Social Fun.* In Hard fun the player is searching for a challenging situation; in Easy fun he is exploring the system; in Altered state fun the game changes by player emotions and in Social fun the communication with other players is the main motivator.

Player Types

Bartle (1996) distinguishes between 4 types of players (expendable up to 16).

1. **Explorers**: An explorer likes to figure things out, explore a level up to its' most secret corners and find enjoyment in it. An example would be playing a

[3] http://www.nielsen.com/us/en/insights/reports/2017/us-games-360-report-2017.html visited on 03.01.2018

level several times to explore hidden treasures.

2. **Achievers**: An achiever likes to simply "achieve". Loosing a game most probably causes loosing interest in the game.

3. **Socializer**: A socializer plays games to meet up with other gamers. To him, the game needs to have a background for long-term social interactions.

4. **Killers**: A killer is basically an achiever with the difference, that winning isn't enough. If he wins, others must loose and his victims needs to express some sort of admiration or respect.

Bartle also points out, that player types are mutually inclusive. An average person would be 80% socializer, 50% explorer, 40% achiever and 20% killer.

Seeking mastery

A very interesting study by Dreyfus and Dreyfus (1980) points out five stages of mastery, when engaging with systems.

1. **Novice**: Someone, who is just started to the experience.

2. **Problem Solver**: has some information about the experience or knows where to find it.

3. **Expert**: Has started to learn about the system. knows something what is not obvious to the casual player.

4. **Master**: Understands the whole system. Has a long year experience in that field.

5. **Visionary**: A special kind of master, who is willing to improve the system.

Zichermann and Cunningham (2011) suggests, that no one should be forced to improve - everyone can stops at any level.

Game Mechanics

The most known framework for game design is called the *MDA Framework* (where game mechanics is the *M*). "It is a postmortem analysis of the elements of a game." - Zichermann and Cunningham (2011)

1. **Mechanics**: functioning components of the game
2. **Dynamics**: Are the interactions with mechanics. What is each player doing with the game mechanics?
3. **Aesthetics**: How does the game feel during the interaction?

Now as we know some basic concepts for player motivation, types and mastery we can continue with a short introduction to Game Mechanics.

Points

The most basic and probably most important game mechanic are points. Examples for points are video game score(player progress measurement) or social networking score (popularity in social media). In Gamification you will be probably dealing with following point system:

1. **Experience points (XP)**: everything a player is doing is measured with experience points. There is no maxing out in XP.
2. **Redeemable points (RP)**: points for exchanging or purchasing.
3. **Skill points**: points gained by performing certain activities in the system. More specific than XP and RP.
4. **Karma points**: points shared by players to measure e.g. player behavior.
5. **Reputation points**: points for measuring trust between two or more parties. Usually complex systems.

Levels

To have a sort of progress indication, levels are a good concept to fulfill these requirements. In my practical work Levels are expressed by the background image the distribution of things to collect and the background music. Players use levels

as a marker, to know, where they are located in the gaming experience over time. One of the most known examples for very good level design is the game "Angry Birds". In this game the player gains confidence and experience after nearly each level. Levels are probably the most important factor of the huge success of this game.

Leaderboards

To have a sort of competition against others, leaderboards were introduced. They are also known as Highscores. The concept of leaderboards is easy to understand - the higher ranked the better. Leaderboards are an important game mechanic since the 80s. Nowadays every competitive game has them implemented. There are two kinds of leaderboards: The no-disincentive leaderboards and the infinite leaderboard. In the no-distinctive leaderboard the focus is to not discourage players, if they are not performing too well. The player will be displayed in the middle of the scoreboard (as a center of attention). If the score is well, it should be properly displayed with the players in the performing range. The infinite leaderboard is the classic leaderboard, where a player tries to beat the score and "kick others from the throne". Nowadays leaderboards are in a very advanced state. They are connected to social profiles, also to daily, weekly, local, global, all time, section scores and many other kinds of leaderboard scenarios. A scoring system is also very useful for the developer - he can see, if the level difficulty is as expected, the statistics match as expected and it can also indicate faulty game behavior.

Badges

Badges reward players for reaching certain milestones or progress in the game. In some players they awaken the desire for collecting. For game designers it's a excellent way to encourage social promotion of their products and services. In some games badges can replace the classical progress system of levels.

Onboarding

Onboarding is a game mechanic, which is used to bring a new player (novice) into the game. Nowadays it is important to get a player into a game as fast as possible, because lessons of casual game market has shown, that a player usually decides in the first minute, if a game is worth playing. A game designer should consider this

mechanics and present the core features of a game as early as possible.

3. Current System

In this section I will describe my work in the field of iML. I will give you a short overview and describe some parts of my implementation in detail. First I will describe the heart of my work - the algorithm itself ant its' visualization. After that, I will introduce and describe a game, where my algorithm is integrated.

3.1 Algorithm implementation

I decided to implement three ant algorithms: the AS, ACS and the MMAS. These are written in C♯ and published on GitHub. The main purpose of this project is to use it in Unity Games, which is described in the next sections. The basic structure of the work shown by an UML diagram is given in 3.1.

3.1.1 Entry point

The purpose *AntAlgorithmChooser.cs* is to choose the proper algorithm for your needs. You or the class itself chooses between one of the three implemented algorithms. The algorithm itself is a member of the class, which can be accessed easily.

3.1.2 The algorithm skeleton

The abstract class *AntAlgorithm.cs* represents the algorithm structure. The main methods of this class are Init(), Iteration() and Step(). Init() initializes the algorithm, Iteration() simulates one iteration of the algorithm and Step() moves all ants one step(one city) ahead. Step() was introduced to make the algorithm even more interactive. You can even split the algorithm up on the step level, to better observe

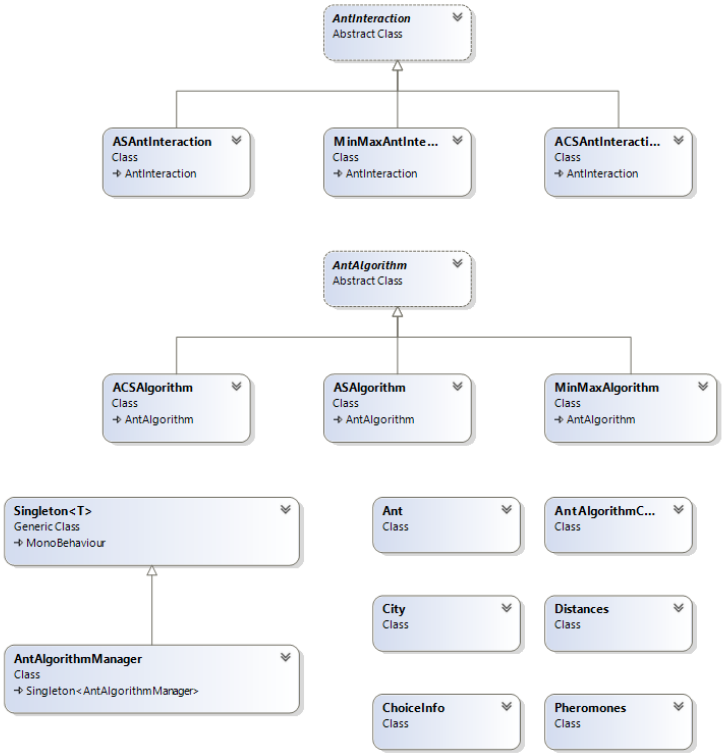


Figure 3.1: UML diagram of the the algorithm implementation

and interact with ant movement and pheromone deposition. Furthermore *AntAlgorithm.cs* can give you pheromones, ants and the cities. It also saves and checks the best solutions so far. Since *AntAlgorith.cs* is an abstract class the implementations of these methods is done in the classes *ASAlgorithm.cs*, *ACSAlgorithm.cs* and *MMASAlgorithm.cs*. These three classes implement the methods Init(), Iteration() and Step() respective to the algorithm. A code snippet of the implementation of the AS algorithm in *ASAlgorithm.cs* is shown in listing 3.1. As you can see in line 3, I use a class called *ASAntInteraction.cs* for the basic functionalities of the AS algorithm. Basically the same is also used in the other two classes *ACSAlgorithm.cs* and *MMASAlgorithm.cs*.

Listing 3.1: C♯ code from ASAlgorithm.cs

```
1 public override void Init()
2 {
3 antin = new ASAntInteraction(a, b, q, numOfAnts, Cities);
4 CheckBestTour();
5 algStep = 1;
6 }
```

```
7
8  public override void Iteration()
9  {
10 antin.UpdateAnts();
11 antin.UpdatePheromones();
12 CheckBestTour();
13 }
```

3.1.3 The algorithm data structures

As you can see in figure 3.1 I used several classes for the representation of objects of the algorithm. The class *Ant.cs* represents an Ant in the algorithm. It initializes and handles the tour (path of the ant), since every ant has a own tour. It also uses the *Distance.cs* class for the distance information. The class *City.cs* represents a point in 2D space. It is called that way, because the points in the TSP are called cities. This class was implemented to grant expandability (so we can also easily add 3D points) and a sort of transparency. *Distances.cs* is calculating and saving the 2D distances between cities. *ChoiceInfo.cs* is handling the choices of ants. It calculates the choice regarding to the pheromone and distance value on the trail. (See listing 3.2 line 7)

Listing 3.2: C♯ code from ChoiceInfo.cs

```
1  public void UpdateChoiceInfo(Pheromones pheromones, Distances
       distances, int alpha, int beta)
2  {
3  for (int i = 0; i < size; i++)
4  {
5  for (int j = i + 1; j < size; j++)
6  {
7  choiceInfo[i][j] = Math.Pow(pheromones.GetPheromone(i, j), alpha)
       *
8  Math.Pow((1.0 / distances.GetDistance(i, j)), beta);
9  choiceInfo[j][i] = choiceInfo[i][j];
10 }
11 }
12 }
```

Another important class is *Pheromones.cs*. This class has methods for initialization, updating, reinitialization, increasing, decreasing and checking the trail pheromone limits. The reinitialization and smoothing of the pheromones is a part of the MMAS algorithm. You can find the source code for this in listing 3.3. In line 3 and 4 you can see the computation for the trail limits as described in the subsection 2.2.3. From line 7 to 14 I have also implemented the pheromone trail smoothing (PTS) (2.2.3), which is used, if the algorithm converges.

Listing 3.3: C♯ code from Pheromones.cs

```
1 public void UpdateTrailLimits(double optimalLength, double rho,
      double pBest)
2 {
3 trailMax = 1.0 / (rho * optimalLength);
4 trailMin = (trailMax * (1.0 - Math.Pow(pBest, 1.0 / numOfCities))
      ) / (((numOfCities / 2) - 1.0) * Math.Pow(pBest, 1.0 /
      numOfCities));
5 }
6
7 public void reinitTrails(double smoothingFactor)
8 {
9 for (int i = 0; i < _pheromones.GetLength(0); i++)
10 {
11 for (int j = 0; j < _pheromones.GetLength(1); j++)
12 {
13 _pheromones[i, j] = _pheromones[i, j] + (smoothingFactor * (
      trailMax - pheromone));
14 }
15 }
16 }
```

3.1.4 The algorithm core

The core functions of the algorithm are controlled and implemented in the abstract class *AntInteraction.cs* and all its child classes *ASAntInteraction.cs*, *ACSAntInteraction.cs* and *MMASAntInteraction.cs*. Basically the main purpose of this classes is to

handle the transition probabilities over all cities for all the ants and pheromone evaporation. Choosing which city an ant is visiting next is located in *AntInteraction.cs* and is performed by the 2 methods CalculateProbs() and ExplorationDecision(). CalculateProbs() simply does what the method name says. (see 3.4) It uses the function GetChoice() from the class *ChoiceInfo.cs* for getting values, representing the choices of ants. In 3.2 you can see, how choices are computed.

Listing 3.4: C♯ code from AntInteraction.cs

```
protected void CalculateProbs(int currCityIndex, int antIndex)
{
// variable initializations

for (int i = 0; i < selectionProbability.Length; i++)
{
if (Ants[antIndex].IsCityVisited(i))
{
selectionProbability[i] = 0.0;
}
else
{
selectionProbability[i] = choiceInfo.GetChoice(currentCity, i);
sumProbabilities += selectionProbability[i];
}
}

// some normalizations
}
```

The decision, which city is the next one with the precomputed probabilities is made by firstly calculating the cumulative probabilities. Then a value between 0 and 1 is chosen by a random function. The city assigned to the computed random value in the cumulative probability list is then chosen as next city in the tour. (see 3.5)

Listing 3.5: descision of the next city

```
double p = random.NextDouble();

```

```
for (int i = 0; i < cumulativeProbs.Length - 1; i++)
{
 if (p >= cumulativeProbs[i] && p <= cumulativeProbs[i + 1])
 {
  return cities[i].Id;
 }
}
```

Another very important part of an ant algorithm is the evaporation and increasing function of pheromones on trails. Evaporation is happening after every iteration. It simply decrease all pheromones by a factor. (see 3.6)

Listing 3.6: pheromone evaporation

```
protected void EvaporatePheromones(int antIndex)
{
double decreaseFactor = 1.0 - rho;

for (int i = 0; i < cities.Count; i++)
{
 int j = Ants[antIndex].GetCityOfTour(i);
 int l = Ants[antIndex].GetCityOfTour(i + 1);

 Pheromones.DecreasePheromoneAs(j, l, decreaseFactor);
 Pheromones.DecreasePheromoneAs(l, j, decreaseFactor);
}
}
```

Increasing pheromones is happening after an iteration. What is increased depends on the algorithm. In the simple AS all used trails are increased. In MMAS only the best ant of the iteration is able to deposit pheromones. The deposit function looks like 3.6 with the difference, that the DepositPheromones() function of the pheromone object is used.

3.2 Algorithm visualization: Ant Algorithm Solver

To visualize the algorithm and make it explainable, I decided to built an Graphical User Interface (GUI), where you can see what the algorithm is doing at any time. I called the implementation with the GUI "Ant Algorithm Simulator". It took me a huge amount of time to implement all features for the visualization and therefor this is a big part of my master thesis.

3.2.1 Current status

The current version of the GUI is capable of:

- visualize MMAS, ACS and AS
- adjust parameters
- perform iterations and steps
- display pheromone connections
- display ant tours
- animate tour construction of all current ants
- change pheromones at any time (interactive)

3.2.2 Explanation

The visualization of the algorithm has several states of the User Interface.

Initial screen

The basic scheme of the GUI is shown in 3.2. After you start the visualization you will see a well structured user interface with several controls to navigate through the different states of the algorithm.

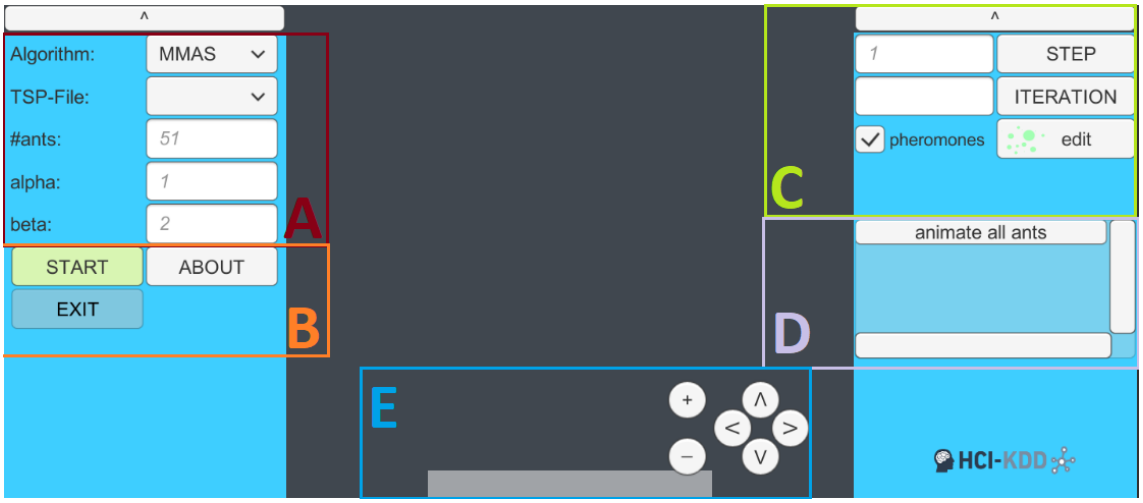


Figure 3.2: Graphical User Interface before initialization

- In region **A** you can switch between algorithms, set the input file and adjust some previous explained parameter.
- In region **B** you can start the initialization of the algorithm, open a window with additional information and exit the program. (the "EXIT" button is only available for Standalone and Android versions)
- In region **C** you can perform algorithm steps or iterations, toggle the visualization of pheromones and edit the values of the pheromones.
- In region **D** you can start the route animation of all ants or a particular one.
- In region **E** you can navigate through the point area, which is available after the initialization.

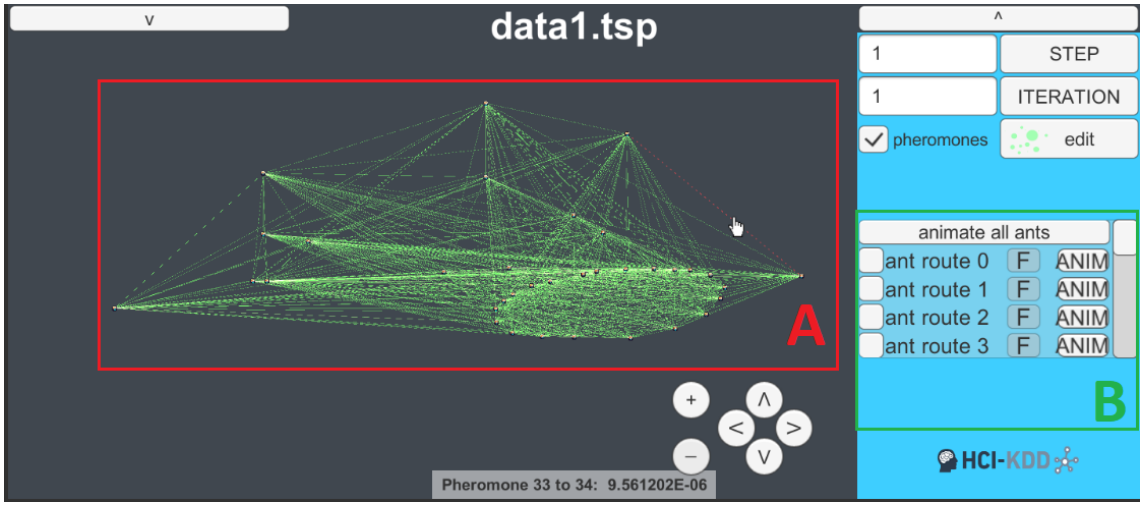


Figure 3.3: Graphical User Interface after initialization

- In region **A** you can see the point set of the algorithm with the initial connections in green. The width of the connections is the value of the pheromones (the thicker, the more pheromones).
- In region **B** you can see the animation controls. In this visualization yo have the opportunity to animate and see the traveling route of any ants at any iteration. At the animation you can change between the "main camera" and a "ant camera" viewpoint. The "main camera" viewpoint shows the default overview and the "ant camera" viewpoint shows the traveling ant while moving. (see 3.5)

After the initialization

After you entered the initial parameter(default vales already pre-entered) and hit the "start" button you will see the visualization displayed in 3.3.

Advanced state

If you enter 200 iterations in the iteration field and hit the "iteration" button, your screen will look like 3.4.

Source code & software description

In figure 3.6 you can see the basic scripts of the visualization part of the ant algorithm. There were no complex task to solve the visualization and analyzing the scripts would not be inside the scope of this work. To make the animations of ants as

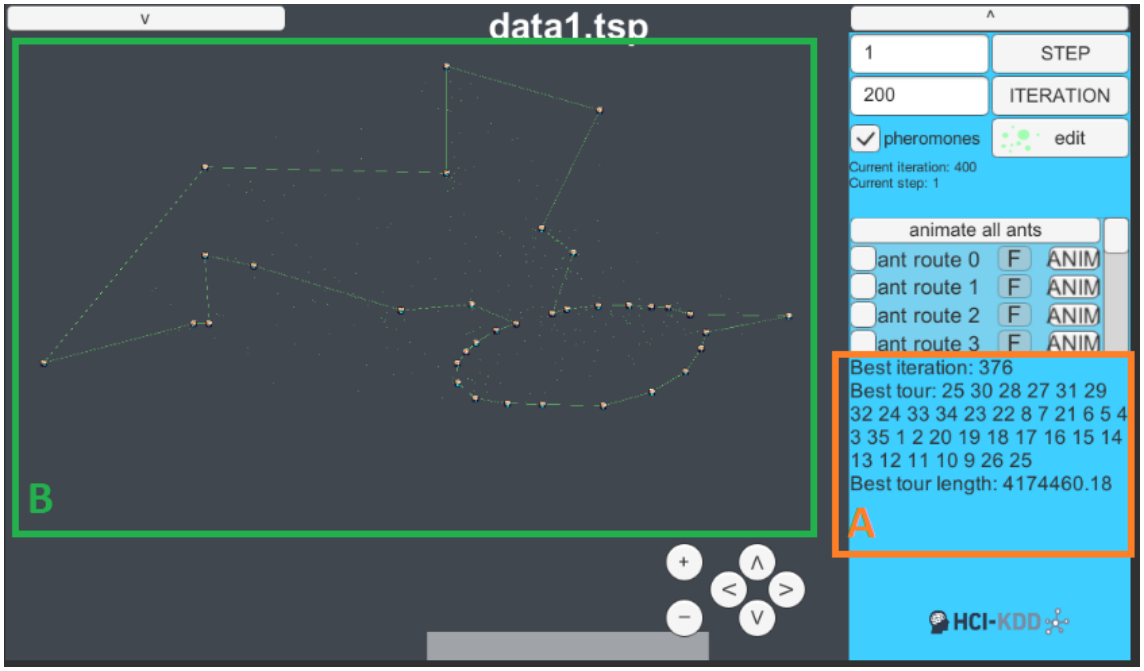


Figure 3.4: Graphical User Interface after 200 iterations

- In region **A** the best route so far (city sequence) and the overall route length.
- In region **B** after 200 iteration you will see, that pheromones on some edges will form a route.

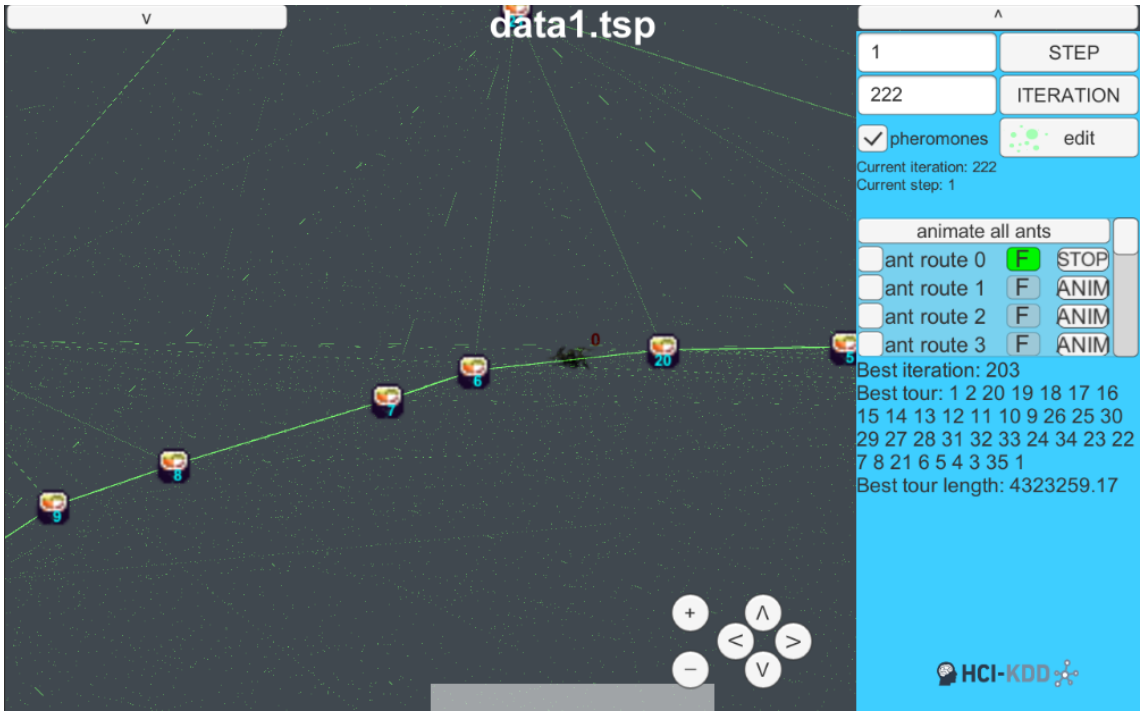


Figure 3.5: Ant animation
You can animate every iteration to get an idea how the algorithm constructed the solution. The visualization has a lot of animation options.

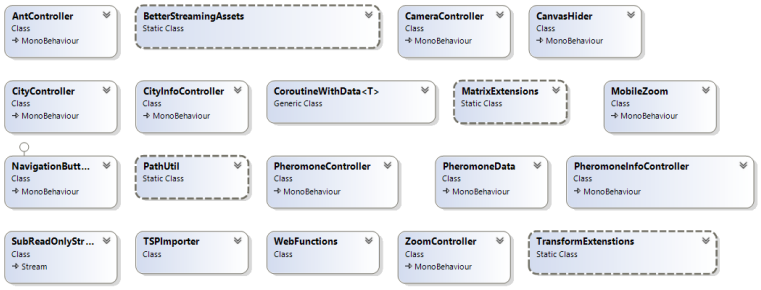


Figure 3.6: UML diagram of the visualization scripts

smooth as possible I used the animation system *iTween* [1] for the Unity game engine. To have a thread safe and low overhead loading of sample point-sets over the web i used the Unity plug-in *Better Streaming assets*[2]. Because of a lack of time, because I wanted to add as much features as possible, I decided to test the user interface manually and did not implement UI or unit tests. The source code of the visualization and the core algorithm is available under https://github.com/AndrejMueller01/IML-TSP-Solver[3]. The whole code is well commented to explain the different parts of the implementation.

Download

Since this part of my project is also implemented with the Unity engine, we have the big advantage to built it for multiple platforms. The current supported platforms are Android and WebGl (Browser). All of this version are available for all and free to use!

1. **Android:** https://goo.gl/SM7cTG

2. **Browser:** https://iml.hci-kdd.org/imlTspSolver/

[1] http://www.pixelplacement.com/itween/index.php visited on 15.08.2018
[2] https://assetstore.unity.com/packages/tools/input-management/better-streaming-assets-103788 visited on 15.08.2018
[3] visited on 22.05.2018

3.3 Travelling Snakesman

3.3.1 Description

The idea behind the game Travelling Snakesman was to design a game, where the previous described iML algorithm runs in the background. The initial version of the game was developed with the help of students from the course "AK Human Computer Interfaces". In this section I will give you a brief overview of the current state of the game, explain some details and refer to possible future work.

Gameplay

As soon as you start the game you will see the main menu, where you can open the leaderboard, open a help window, quit the game and of course, play the game. Before you can hit the "play" button you need to enter a name and select the level, with which one you want to start. After you have started the game you can control a snake with simply clicking or tapping on the screen. The goal is to eat all apples as fast as possible! The time you needed for eating all the apples in a certain level shows up in the leaderboard! The white markers indicate the apples off the screen. On the top left you can see a red, yellow or green square. This square notifies, if your contribution is actually better than the algorithm computation itself. (see next section) On the top ou can see also the time you needed so far and the number of remaining apples. If you finish a level (eat all the apples), you can start the next one or go back to the main menu! With the back or "ESC" button you have also the opportunity to pause the game. The leaderboard is showing the best scores of all time, the week and the day.

Technical background

Immediately after the gameplay starts the algorithm (MinMaxAntSystem algorithm) runs "number of apples * 5" iterations. After the computation you have full control over a snake. Every time you eat an apple the previous mentioned algorithm runs 5 iterations, with a higher amount of pheromones on the traveled edge.(based on the version. Usually x5) So if you travel from one apple to another, your choice

will be considered for the algorithm computations, we can say, the player interacts with the algorithm in the background and has an influence of the outcome. There is also a suggestion indicated by brighter apples. This suggestion is based on the pheromones from the current position. The more pheromones on the edge, the stronger the suggestion (apples are brighter). After you have eaten the third apple the algorithm computed 15 iterations with a higher chance to choose edges you have traveled on.

The scientific goal of this game is solving the question: *"Do we have a better performance by interacting with the algorithm?"* (see chapter 5)

Main menu

The main menu of the game is very simple. (see figure 3.10a) You can switch to the leaderboards and help screen. Furthermore you can quit the game, change the player name and select a level. (currently three)

Leaderboards

The leaderboards are showing the scores of all the players.(see figure 3.10b) You have the opportunity to switch between the levels and to scroll down. Since the version 1.1 there are also daily, weekly and all-time leaderboards, with an improved design.

Database

The SQL database consists of a table with following columns:

- the first algorithm computation (without human help)
- the best route of the algorithm computation
- the best iteration of the algorithm computation
- the second algorithm computation (algorithm with human help)
- the best route of the algorithm + human computation
- the best iteration of the algorithm + human help computation

#	Name	Typ	Kollation	Attribute	Null	Standard	Extra
1	ID	int(11)			Nein	*kein(e)*	AUTO_INCREMENT
2	scoreID	varchar(50)	latin1_swedish_ci		Nein	*kein(e)*	
3	localpheromonevalue	text	latin1_swedish_ci		Nein	*kein(e)*	
4	iterationssofar	int(11)			Nein	*kein(e)*	
5	bestsuggested	int(11)			Nein	*kein(e)*	
6	previousechosen	int(11)			Nein	*kein(e)*	

Figure 3.7: Pheromone database table

#	Name	Typ	Kollation	Attribute	Null	Standard	Extra
1	ID	varchar(50)	latin1_swedish_ci		Nein	*kein(e)*	
2	timestamp	timestamp		on update CURRENT_TIMESTAMP	Nein	CURRENT_TIMESTAMP	ON UPDATE CURRENT_TIMESTAMP
3	name	varchar(50)	latin1_swedish_ci		Nein	*kein(e)*	
4	tspname	varchar(50)	latin1_swedish_ci		Nein	*kein(e)*	
5	algoscore	double			Nein	*kein(e)*	
6	algobestiteration	int(11)			Nein	*kein(e)*	
7	algotour	text	latin1_swedish_ci		Nein	*kein(e)*	
8	userscore	double			Nein	*kein(e)*	
9	userbestiteration	int(11)			Nein	*kein(e)*	
10	usertour	text	latin1_swedish_ci		Nein	*kein(e)*	
11	time	int(11)			Nein	*kein(e)*	
12	platform	varchar(10)	latin1_swedish_ci		Nein	*kein(e)*	
13	collectedapplessequence	text	latin1_swedish_ci		Nein	*kein(e)*	
14	followedSuggestion	text	latin1_swedish_ci		Nein	*kein(e)*	

Figure 3.8: Scores database table

- the time of the gameplay (for the leaderboard)
- the name of the player (for the leaderboard)
- all pheromones at any time (in a separate table)
- all suggestions made for the next apple (in a separate table)

The actual version of the pheromone database table you can see in figure 3.7 and the scores table in figure 3.8.

Parameter settings

The algorithm is running with the default parameter for MMAS as shown in table 2.1. We also decided to adjust the amount of pheromones increasing by using an edge in different versions.

Human intuition

Because of the different platforms the screen size will vary. We decided to add a web version, where you can see all the apples at once (without the white markers). In the result section we will discuss if there are differences in the results, if the human is able to see the whole point-set at once or just a part of it.

Figure 3.9: UML diagram of the game scripts

Cross platform & Download

The game is created with the Unity engine [4] in C♯. This engine runs on nearly every platform. We decided that the game should run on Android, in the web browser and standalone on Windows.
We decided also to add some versions without affecting human behavior/intuition too much. (Version from far (no zooming); version without suggestions) Following some download links for Travelling Snakesman:[5]

1. **Android:** https://goo.gl/nYgVjQ
2. **Browser:** https://iml.hci-kdd.org/TravellingSnakesman/
3. **Browser from far:** https://iml.hci-kdd.org/TravellingSnakesmanV2/
4. **Browser without suggestions:** https://iml.hci-kdd.org/TravellingSnakesmanWS/

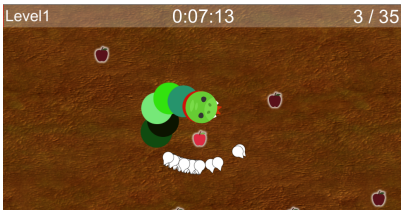
Source code & software description

The UML structure is shown in figure 3.9. I programed different scripts to control the flow and interaction of the game. Because of the simplicity of the game and a lack of time, I decided to test the user interface manually and did not implement UI tests. The source code of the Travelling Snakesman is available under

[4] https://unity3d.com/de visited on 20.01.2018
[5] visited on 20.07.2018

(a) Main menu

(b) Leaderboard



(c) Gameplay

Figure 3.10: Main scenes of Travelling Snakesman
Screenshots from the Android version

https://github.com/AndrejMueller01/AK_HCI_Travelling_Snakesman[6]. The development isn't finished yet. Parts of the code will be updated in the future to maintain and ensure the scientific value of this project.

3.3.2 Development tools

I programmed Travelling Snakesman in C#, because it is designed for developing apps for Microsoft, which is certainly a goal for a game to fulfill. C# is also part of the .NET framework, which is steady growing and brings a lot of reliefs in form of libraries. C# is also used to develop web applications and increasingly gain popularity for mobile development too. The main purpose of using C# was, because it is supported by the game engine Unity, which is the most used Game Engine today. I decided to use Unity, because it is free, provides a lot of tutorials, it is very simple, platform independent, and the fact, that more than a third of top games today are made in Unity. So it can't be bad. For version control I used GIT, because it is fast, there are a lot of visualization tools for it and because I was already experienced using it. (see table 3.1)

[6]visited on 22.05.2018

Development tool	Reason for Decision	Alternatives
Programming language C#	commonly used programming language for games, supported by .Net framework, personal skills	Java, Python
Game Engine Unity	easy to use, free, **platform independent**, most used	Unreal engine 4, CryEngine
Version control system GIT	easy to use, mostly used today, fast	SVN

Table 3.1: Used tools

3.3.3 Future work

There is a lot of future work possible to improve the Travelling Snakesman game. To make the gaming experience better we could add some the badge game mechanics. To make the algorithm more adjustable we could add config files for the parameters, to change them without compiling the whole game. To make the debugging easier we could add logging into files. To make it more readable and preventing errors we could add unit tests and even UI tests. To make it accessible for everyone we could deploy it for Apple iOS. To make it safer we could improve the database connection and structure.

I learned in the past years as software developer, that you are never finished with a project. All in all I think, we made a stable first version for a basis of future improvements.

4. Materials and Methods

Our approach to implement iML and the main part of my Master Thesis was to design a Game, playable on many platforms, where an iML algorithm has a major role. We've decided to make an iML Challenge, with the goal, that we have a fancy game with integrated iML elements at the end. The Challenge was a part of the course AK Human Computer Interaction on the TU Graz.

4.1 iML Challenge

We decided to split our iML Gamification project and provide a part of it as a challenge in Machine Learning.

4.1.1 Description

The idea is, that we provide the preprocessed data and the iML algorithm and you make us an user interface. We will also make sure, that the interfaces to the GUI are well designed. It should be a minimal and easy understandable game, programmed with the Unity Game Engine [1] In figure 4.1 you can see the three parts of our project.

[1] https://unity3d.com visited on 02.08.2018

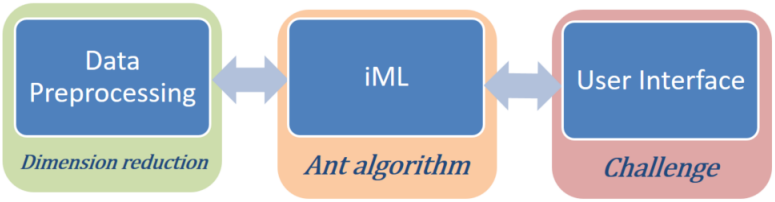


Figure 4.1: The three part of the iML project
the challenge is the right part, designing a user interface

This challenge is used in a course at the TU Graz. This has the great advantage to attract or include a higher number of people. There is also a challenge website under the institute domain, with all updated information and guidelines. [2]

4.1.2 Goal and Awards

The goal is to create an user interface of the game, which is simple, minimal and has a professional look. It is also very important to consider current game design guidelines to make the player play and enjoy the game for a longer period of time.

[2] http://hci-kdd.org/iui-where-hci-meets-ai-challenge-2017/ visited on 12.12.2016

5. Results

5.1 Travelling Snakesman

We asked students to play our game. The following results are from the 25th of June 2018. We tested it with the version without the suggestion feature, default parameter settings (see 2.2.4) and increasing the pheromones on chosen edges by 5 times the actual value. This was an early test. We evaluated some problems during the testing and we will consider them in future tests.

5.1.1 Level 1 - 39 games played

- **Median Computer**: 4528781.47186654
- **Median Human + Computer**: 4367300.9983592
- **Median Human**: 4841033.06461351

5.1.2 Level 2 - 28 games played

- **Median Computer**: 36436306.85666895
- **Median Human + Computer**: 35914005.21526084
- **Median Human**: 41549697.67231192

5.1.3 Level 3 - 28 games played

- **Median Computer**: 44338616.8737223

- **Median Human + Computer**: 43412582.05347122
- **Median Human**: 46713316.36265121

6. Discussion and Lessons Learned

In the results from chapter 5 you may see significant improvement in performance, if humans and a computer are working together. Nevertheless there are too few test samples and the test environment is too small. There are also some other things to take into consideration. The first problem was, that we used a fixed number of iterations. In future tests we should investigate, how long it takes to reach the nearly optimal solution. The second problem was, that we only had 3 levels. The levels were designed in a way, that you perform better, if you are a human. In future tests we should use randomized levels as well. The third problem was, that we need to emphasize and analyze the causes of improvements. The last problem was, that the algorithm isn't in a approved state. Others should review my code and confirm, that it is working properly. There are no excessive tests to ensure that the algorithm performs as good as possible. *In the future work we should eliminate these problems and take more focus on software quality. To do that, we should arouse interest for iML, get more helping hands and finally achieve bigger goals.*

7. Conclusion

There are a lot of opportunities open in the field of interactive Machine Learning, especially in the medical domain. If you want to solve complex tasks, it should be often of great interest to involve humans directly into the core calculations of an algorithm. In my work I challenged a lot of difficult tasks. It was often not clear, how the influence of a human would affect the performance. A great achievement was to implement my self made interactive Ant Algorithm, use it in a game and save the behavior and the performance of each player for a later evaluation. It was a lot of work. In the first phase I studied different Ant Algorithms and decided to implement some of these on my own. After that phase I got help from students, which started to develop a game for my algorithm. Later on I customized and improved the game and I released a first version for mobile phones and browsers. I even programmed a visualization for the algorithms itself, to get a feeling, how such an interactive Ant Algorithm works. Interactiveness combined with complex Machine Learning algorithms is in my opinion a really necessary topic to improve. I am sure, that there are many opportunities for this in the future!

"Science is to test crazy ideas, Engineering is to put these ideas into Business"
- Holzinger Group

List of Figures

List of Tables

References

[2016]. *Concorde TSP Solver Project Website*. http://www.math.uwaterloo.ca/tsp/concorde/index.html. Accessed: 2017-06-07.

Akgül, Ceyhun Burak, Daniel L. Rubin, Sandy Napel, Christopher F. Beaulieu, Hayit Greenspan, and Burak Acar [2011]. *Content-Based Image Retrieval in Radiology: Current Status and Future Directions*. *Journal of Digital Imaging*, 24(2), pages 208–222. ISSN 1618-727X. doi:10.1007/s10278-010-9290-9. https://doi.org/10.1007/s10278-010-9290-9.

Bartle, Richard [1996]. *Hearts, clubs, diamonds, spades: Players who suit MUDs*. *Journal of MUD research*, 1(1), page 19.

Bellman, Richard [1962]. *Dynamic programming treatment of the travelling salesman problem*. *Journal of the ACM (JACM)*, 9(1), pages 61–63.

Bernstein, Peter L and Peter L Bernstein [1996]. *Against the gods: The remarkable story of risk*. Wiley New York.

Bishop, Christopher M [2006]. *Pattern recognition and machine learning*. springer.

Christofides, Nicos [1976]. *Worst-case analysis of a new heuristic for the travelling salesman problem*. Technical Report, Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group.

Cover, Thomas M. and Joy A. Thomas [2006]. *Elements of Information Theory 2nd Edition*. 2 Edition. Wiley Series in Telecommunications and Signal Processing, Wiley-Interscience. ISBN 0471241954. http://www3.interscience.wiley.com/cgi-bin/bookhome/110438582?CRETRY=1&SRETRY=0.

Deneubourg, J. L., S. Aron, S. Goss, and J. M. Pasteels [1990]. *The self-organizing exploratory pattern of the argentine ant. Journal of Insect Behavior*, 3(2), pages 159–168. ISSN 1572-8889. doi:10.1007/BF01417909. http://dx.doi.org/10.1007/BF01417909.

Deterding, Sebastian, Dan Dixon, Rilla Khaled, and Lennart Nacke [2011]. *From Game Design Elements to Gamefulness: Defining "Gamification"*. In *Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments*, pages 9–15. MindTrek '11, ACM, New York, NY, USA. ISBN 978-1-4503-0816-8. doi:10.1145/2181037.2181040. http://doi.acm.org/10.1145/2181037.2181040.

Dorigo, M. and L. M. Gambardella [1997]. *Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation*, 1(1), pages 53–66. ISSN 1089-778X. doi:10.1109/4235.585892.

Dorigo, Marco [1992]. *Optimization, learning and natural algorithms. Ph. D. Thesis, Politecnico di Milano, Italy.*

Dorigo, Marco and Thomas Stützle [2004]. *Ant Colony Optimization.*

Dreyfus, Stuart E and Hubert L Dreyfus [1980]. *A five-stage model of the mental activities involved in directed skill acquisition.* Technical Report, California Univ Berkeley Operations Research Center.

Glover, Fred [1989]. *Tabu Search—Part I. ORSA Journal on Computing*, 1(3), pages 190–206. doi:10.1287/ijoc.1.3.190. https://doi.org/10.1287/ijoc.1.3.190.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville [2016]. *Deep Learning.* MIT Press. http://www.deeplearningbook.org.

Goss, S., S. Aron, J. L. Deneubourg, and J. M. Pasteels [1989]. *Self-organized shortcuts in the Argentine ant. Naturwissenschaften*, 76(12), pages 579–581. ISSN 1432-1904. doi:10.1007/BF00462870. http://dx.doi.org/10.1007/BF00462870.

Grassé, Pierre-Paul [1959]. *La reconstruction du nid et les coordinations interindividuelles chez Bellicositermes natalensis et Cubitermes sp. La théorie de la stigmergie:essai d'interprétation du comportement des termites constructeurs. Insectes Sociaux*, 6, pages 41–83.

Held, Michael and Richard M Karp [1962]. *A dynamic programming approach to sequencing problems. Journal of the Society for Industrial and Applied Mathematics*, 10(1), pages 196–210.

Holzinger, Andreas [2016]. *Interactive machine learning for health informatics: when do we need the human-in-the-loop? Brain Informatics*, 3(2), pages 119–131. ISSN 2198-4026. doi:10.1007/s40708-016-0042-6. `http://dx.doi.org/10.1007/s40708-016-0042-6`.

Holzinger, Andreas, Markus Plass, Katharina Holzinger, Gloria Cerasela Crişan, Camelia-M Pintea, and Vasile Palade [2016]. *Towards interactive Machine Learning (iML): applying ant colony algorithms to solve the traveling salesman problem with the human-in-the-loop approach.* In *International Conference on Availability, Reliability, and Security*, pages 81–95. Springer.

Holzinger, Andreas, Markus Plass, Katharina Holzinger, Gloria Cerasela Crisan, Camelia-M Pintea, and Vasile Palade [2017]. *A glass-box interactive machine learning approach for solving np-hard problems with the human-in-the-loop. arXiv preprint arXiv:1708.01104.*

Huffman, David A [1952]. *A method for the construction of minimum-redundancy codes. Proceedings of the IRE*, 40(9), pages 1098–1101.

Huotari, Kai and Juho Hamari [2012]. *Defining gamification: a service marketing perspective.* In *Proceeding of the 16th International Academic MindTrek Conference*, pages 17–22. ACM.

Johnson, D [1996]. *Asymptotic experimental analysis for the Held–Karp traveling salesman bound DS Johnson* LA McGeochf EE Rothberg.* In *Proceedings of the seventh annual ACM-SIAM symposium on Discrete algorithms*, volume 81, page 341.

Johnson, David S and Lyle A McGeoch [1997]. *The traveling salesman problem: A case study in local optimization. Local search in combinatorial optimization*, 1, pages 215–310.

Lavrenko, Victor [2015]. *Neural Networks 6: solving XOR with a hidden layer.* youtube. `https://www.youtube.com/watch?v=kNPGXgzxoHw`.

Lazzaro, Nicole [2004]. *Why We Play Games: Four Keys to More Emotion Without Story.* In *Game Developers Conference.* http://xeodesign.com/xeodesign_whyweplaygames.pdf.

LeCun, Yann, Yoshua Bengio, and Geoffrey Hinton [2015]. *Deep learning. Nature*, 521(7553), pages 436–444.

Lee, Joey J and Jessica Hammer [2011]. *Gamification in Education: What, How, Why Bother? Academic Exchange Quarterly*, 15(2), page 2. http://www.gamifyingeducation.org/files/Lee-Hammer-AEQ-2011.pdf.

Lin, S. [1965]. *Computer solutions of the traveling salesman problem. The Bell System Technical Journal*, 44(10), pages 2245–2269. ISSN 0005-8580. doi:10.1002/j.1538-7305.1965.tb04146.x.

Lin, S. and B. W. Kernighan [1973]. *An Effective Heuristic Algorithm for the Traveling-Salesman Problem. Oper. Res.*, 21(2), pages 498–516. ISSN 0030-364X. doi:10.1287/opre.21.2.498. http://dx.doi.org/10.1287/opre.21.2.498.

Malle, Bernd, Peter Kieseberg, and Andreas Holzinger [2017]. *Do not disturb? classifier behavior on perturbed datasets.* In *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pages 155–173. Springer.

McCartney, Margaret [2016]. *Margaret McCartney: Game on for Pokémon Go. BMJ*, 354. doi:10.1136/bmj.i4306. http://www.bmj.com/content/354/bmj.i4306.

McGonigal, J. [2011a]. *We don't need no stinkin' badges: How to re-invent reality without gamification.*

McGonigal, Jane [2011b]. *Reality Is Broken: Why Games Make Us Better and How They Can Change the World.* Penguin Group , The. ISBN 1594202850, 9781594202858.

Minsky, M. and S. Papert [1969]. *Perceptrons.* MIT Press, Cambridge, MA.

Murphy, K [2012]. *Machine learning: a probabilistic approach.*

Nilsson, Christian [2003]. *Heuristics for the traveling salesman problem. Linkoping University*, pages 1–6.

Padberg, Manfred and Giovanni Rinaldi [1991]. *A Branch-and-Cut Algorithm for the Resolution of Large-Scale Symmetric Traveling Salesman Problems. SIAM Review*, 33(1), pages 60–100. doi:10.1137/1033004. `https://doi.org/10.1137/1033004`.

Reinelt, Gerhard [1991]. *TSPLIB—A traveling salesman problem library. ORSA journal on computing*, 3(4), pages 376–384.

Salen, Katie and Eric Zimmerman [2003]. *Rules of Play: Game Design Fundamentals*. The MIT Press. ISBN 0262240459, 9780262240451.

Samuel, A. L. [1959]. *Some Studies in Machine Learning Using the Game of Checkers. IBM Journal of Research and Development*, 3(3), pages 210–229. ISSN 0018-8646. doi:10.1147/rd.33.0210.

Scholkopf, Bernhard and Alexander J. Smola [2001]. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, Cambridge, MA, USA. ISBN 0262194759.

Shannon, CE [1948]. *A mathematical theory of communication. The Bell System Technical Journal*.

Silver, David, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. [2016]. *Mastering the game of Go with deep neural networks and tree search. Nature*, 529(7587), pages 484–489.

Stützle, Thomas and Holger H. Hoos [2000]. *MAX-MIN Ant System. Future Gener. Comput. Syst.*, 16(9), pages 889–914. ISSN 0167-739X. `http://dl.acm.org/citation.cfm?id=348599.348603`.

WILSON, E. O. [1971]. *The insect societies*. Cambridge, Massachusetts, USA, Harvard University Press [Distributed by Oxford University Press.

Zichermann, Gabe and Christopher Cunningham [2011]. *Gamification by design: Implementing game mechanics in web and mobile apps*. " O'Reilly Media, Inc.".