KLEX
Klusemann Extern
Marschallgasse 19-21
A-8020 Graz

Artificial Intelligence

a categorization

Vorwissenschaftliche Arbeit
vorgelegt von Adriel Ondas

BetreuerIn: Manuel Menzinger

Schuljahr 2020-2022

Abstract

This thesis will start by describing and defining artificial intelligence, and what it makes possible.

It will then move on to differentiate between the different families of AI, and demonstrate these differences with appropriate examples.

Preface

Thank-yous here

Contents

1 Introduction

=Abstract, aber genauer, kann referenzieren

This thesis was written as a VWA.

The Author chose this topic, as he is very interested in the topic of artificial intelligence, and wanted to use the possibility of investing more time into a topic to learn more about it.

2 What is artificial intelligence?

First it is necessary defining what is meant by artificial intelligence in this thesis, as there is a broad range of definitions.

To get an overview it is helpful to view an artificial intelligence as an entity, a sort of black box. This construct can be called an "Agent". This thesis will shed some light into these black boxes.

2.1 Defining Artificial intelligence

2.1.1 Agents

First, an agent has to get an input, like sound from a microphone, or a click position on a website. Then, it has to process this input, and lastly it has to have some sort of output, like driving somewhere, or showing different things on a screen.

We call the input "Percept", and the output "Action". As depicted in **??**

Agents vary widely in their implementation and function, and can range from bots on the web, to the microchip in your smart stove or Roomba. There are many kinds of agents, because they have to act in different environments, for example a computer, a room where a robot is driving around, a map, a game, ...

This thesis however, will mostly focus on the internal structure of the agents, which can be similar, even through the corresponding agents are acting in very different environments. The goal of this thesis is an overview of these different internal structures.

As the concept of an agent is so openly defined, it helps to narrow it down further. For example a card shuffler has all the prerequisites for an agent, but this thesis will focus only on intelligent agents.

For this, we must define intelligence first.

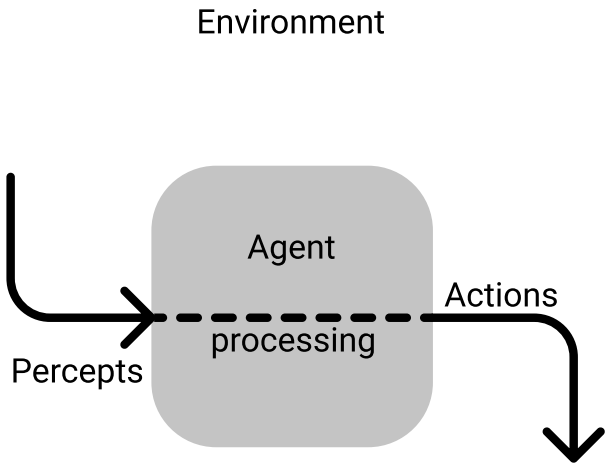

Figure 2.1: Interaction of an agent and its environment

2.1.2 Intelligence

Intelligence can be viewed as ”the ability to learn, understand and think logically about things; the ability to do this well”

Important to stress here are two separate concepts: Learning, and thinking logically. In artificial intelligence, these concepts can be separated, as to say there is a learning, and an applied phase, where the logical thinking comes into play. There are also models combining these two phases, where learning takes place while also applying the previously learned.

2.1.3 Artificial intelligence

Artificial Intelligence can therefore be defined as an intelligent agent. This restriction of only intelligent agents is still not very narrow, as intelligent agents can still be a lot of things: A calculator has input, processes it intelligently, and shows the result.

So this work will focus on only the more complicated intelligent agents.

2.2 Further definitions

2.2.1 Combination

But there are very different models for artificial intelligence, as will be explained in more detail in the next chapters, they all have their advantages, but of course their own little problems as well. So in practical use, different models are often chained together, to leverage the advantages of each sort of agent, to create an agent better than the individual agents. A GAN - a generative adverserial network would be one example: In such a setup there are two sub-agents (Generator and Discriminator) working against eachother, as depicted in **??**.

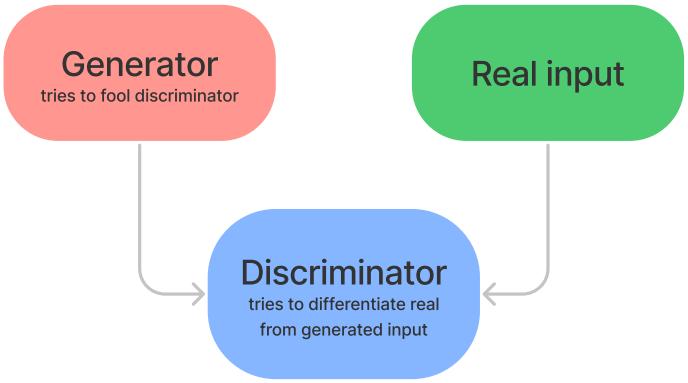

Figure 2.2: a generative adversarial network

2.2.2 Overfitting

When a model performs good on a trainingset, but bad on the testing data, this is called overfitting. It happens when the model is ”memorizing” the answers, which results in less general knowledge.

2.2.3 Cross validation

To combat overfitting cross validation can be used. Cross validation is a process where the border between training and test data is defined differently a few times, and the

same learning algorithm is applied to all of these cases. Then the best one is used.

3 Example architectures

In this thesis four examples of models will be used to illustrate the categorization. These were chosen, as they are relatively simple examples to explain, and are fitting to the categorization.

3.1 Decision trees

A decision tree is a tree consisting of nodes, in which each node answers a yes/no question.

A decision tree can be made symbolically, but to create one subsymbolically, data is needed, which is used to shape the tree accordingly. The hard part is figuring out what question the next node will split the data by.

One method of measuring the usefulness of a question is the Gini method. The Gini method is a function where the inputs are: the number of samples this node as to sort (n), the amount of samples sorted to yes(yes), and the amount sorted to no(no), which are combined to give the relative percentage of true and false outcomes. It gives out an output between 0 and 1, showing the mixedness of the output. It is calculated as follows: Gini impurity $= 1 - (yes/n)^2 - (no/n)^2$ This means that, for example, the data 11100000 has a gini impurity of $1 - (3/8)^2 - (5/8)^2 = 0.47$, the data 11111110 has a lower impurity (0.22). The smaller this impurity, the less mixed the output is. Which is what we want, as less mixed means the tree is surer of its output. So to choose the next node to add to a tree, we measure the GINI impurity of all possible parameters, and choose the one with the smallest impurity. We repeat this process until we only have few samples left in each leaf. An example for a decision tree and its building process is viewable here:

$https://www.w3schools.com/python/python_ml_decision_tree.asp$

Once created, to find an answer, one must walk down the tree, following the path the nodes lead you, until an end (a leaf) is met.

3.1.1 Usage

There are two advantages of decision trees in comparison to other forms of AI. One is the ease of understanding it, as we can retrace decisions with relative ease, which is very important in applications where trust in an algorithm is an issue. So for example in an environment where job applications or prison sentences are guided by algorithms, it might be required to be able to check the algorithm for, for example, discriminatory behavior.

Another benefit decision trees offer is the low computational complexity, in both training and application. Of course, they can grow arbitrarily large, but with the right techniques they can be brought back to reasonable sizes, which just capture the essentials.

There are two methods for cutting the size of a tree: Early stopping and pruning.

In early stopping we stop a branch as soon as we cannot split the data with sufficiently low Gini impurity. This can be very efficient, but there are cases where one question does not help at all, but more do. One example would be an XOR-Gate, as any one input alone does not provide enough information, so the learning algorithm would stop, although the question could be answered by two questions.

featureA	featureB	output
0	0	0
0	1	1
1	0	1
1	1	0

FeatureA and featureB both have a gini impurity of 0.5, which is quite high, but together they can accurately describe the output.

So to avoid stopping too early, one can first generate the whole tree, and then prune useless branches away.

Of course, decision trees are not perfect: For one they require a lot of data to be conclusive, and as they are one of the simpler forms of AI, they need to get quite big to capture complex concepts.

3.2 Neural nets

MA A neural net is a structure where values are passed on through layers of nodes. Each node performs only a relatively simple operation with parameters inherit to each specific node.

First the amount of layers and nodes has to be defined, and to train the neural net, these parameters can be tuned.

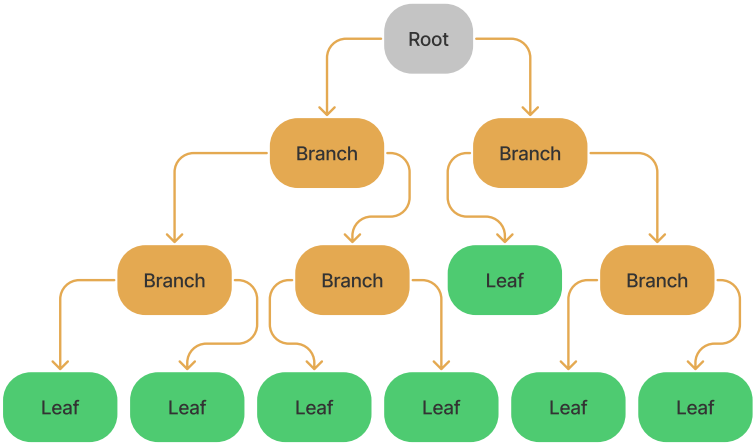

Figure 3.1: This Image depicts a decision tree

As there are normally too many possible values for the parameters not every possible combination can be brute forced. There are different methods for tuning the parameters. Of course these parameters could, in theory, also be filled in by hand, but this is normally not feasible. Two methods to fill them in are back propagation and evolutionary learning.

3.2.1 Backpropagation

In backpropagation **??**, a supervised method, an input is given, and the output is compared to the known correct output. Now the nodes responsible for the biggest error are slightly nudged in the right direction.

3.3 Support vector machines

3.3.1 K-nearest-Neighbours

As a parameterless approach, k-nearest-neighbours works by saving the whole of the training data, representing it in an n-dimensional space, and letting the k-nearest neighbors decide how to act. There are only few parameters to control now, and the amount of parameters is no longer strictly bound by the complexity of the problem at hand. Mainly, there remain: the k, the weight of the different dimensions, and the way the

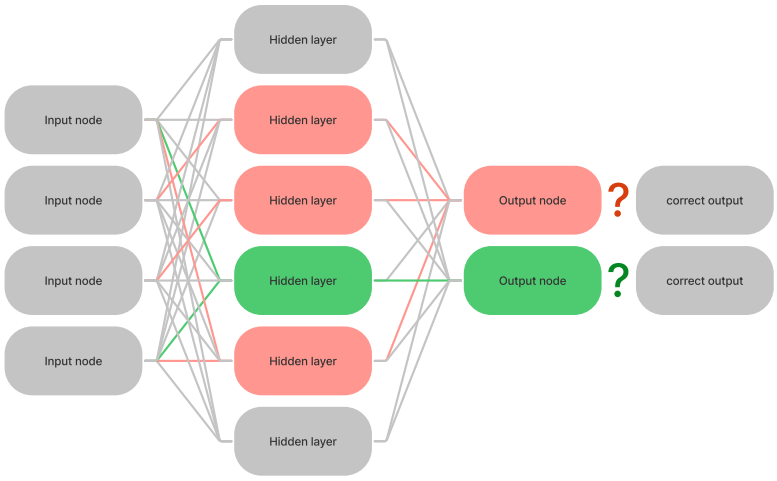

Figure 3.2: Neural network and backpropagation

neighbors are combined. The weight depends on the real-live weight of the dimensions (for example, color could have a lower significance than the size of an object to categorize). As for the ways in which the neighbors are combined: There are many options here, like, for example, Majority vote, average, weighed average, ...

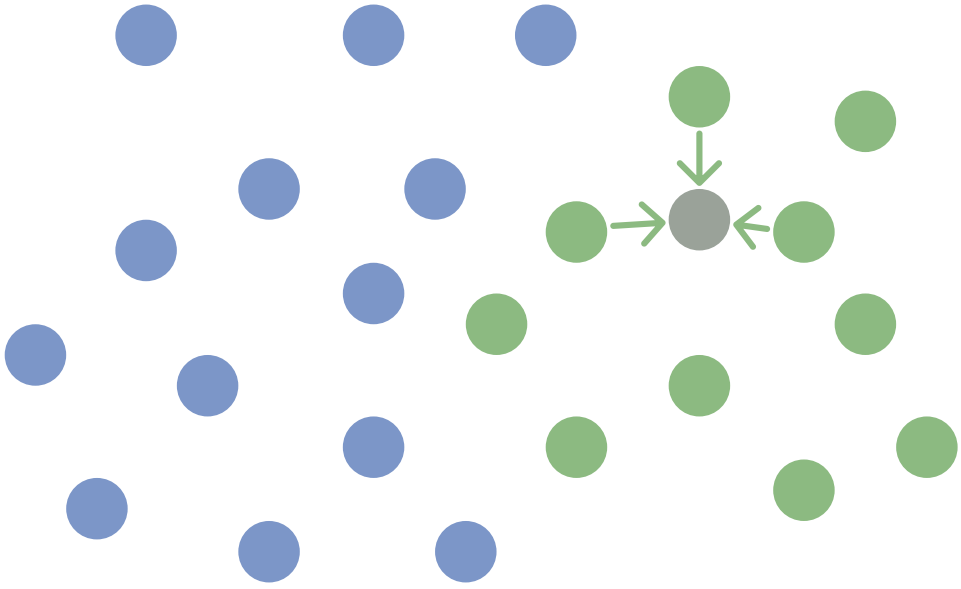

Figure 3.3: K-Nearest-Neighbours

3.3.2 Support vector machines

A problem with k-nearest-neighbours type approaches is high memory usage, as all the training data has to be stored. To combat this, one can use the fact that in most problems the entries close to the border are more helpful than entries in the middle of a decisive cluster.

So as an extension of k-nearest-neighbours, support vector machines were developed. They come with most of the strengths of k-nearest-neighbours, while eliminating some weaknesses.

In a support vector machine only the few entries along the border are stored, these are called support vectors, as they ”hold up” the border. When a decision needs to be made, only the side of the border has to be checked to come to a conclusion. This can also mean a significant increase in efficiency, as not all distances have to be checked like in k-nearest-neighbours.

This reduces the problem to one of fitting a curve to some points, which is already well-researched, and the parameters to choose here are on how detailed the curve can be while still not overfitting.

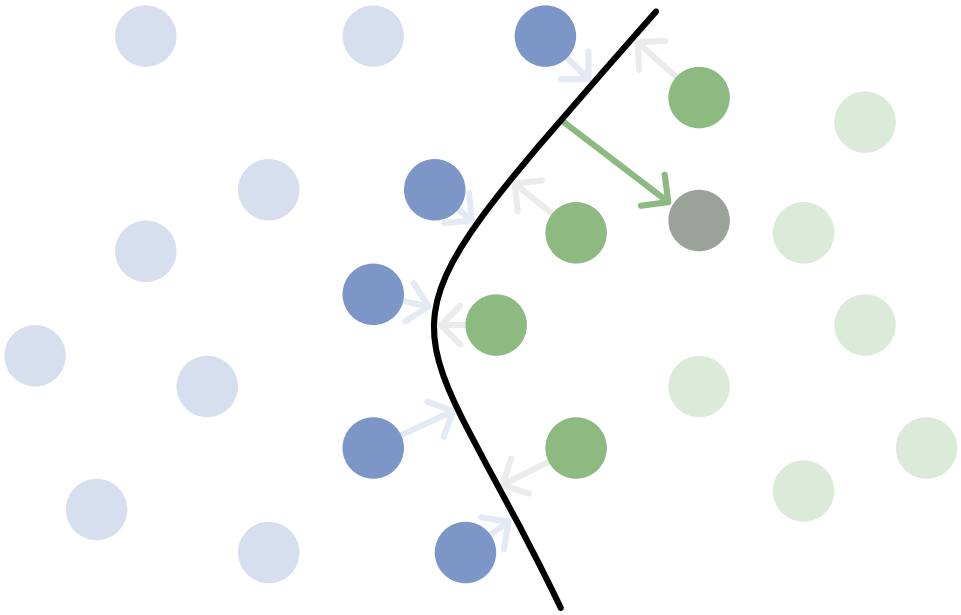

Figure 3.4: Support Vector Machine

3.3.3 Usage

Support vector machines are often the first approach tried, as they are relatively simple, and yield good results in a wide variety of use cases.

They are relatively simple to view and understand, should there only be few important dimensions.

3.4 Q-Tables

A Q-Table is a very literal interpretation of the definition of AI: As an AI maps inputs to outputs, a very simple method without a function in the background would just be a table. To implement such a table is only possible in discrete space, as every state has to have an action assigned to it.

To train a Q-Table is relatively straightforward: There is a score assigned to every State-Action pair. Then the agent can learn with reinforcement learning: Should an action lead to a positive reward the score of the previous state-action pair is updated up slightly and vice versa.

To update the values often the function $Q_{new} = (1 - \alpha) * Q_{old} + \alpha * reward$ is used.(α = learning rate)

4 Categorization

As there are many models of artificial intelligence, all working differently, there are also a few distinctions that can be made between them. This thesis will focus on three fairly important distinctions that can be drawn between different models.

4.1 Symbolic vs subsymbolic

SymbolicVsSubsymbolic A first big distinction is to make between symbolic and subsymbolic artificial intelligence. The difference is in how knowledge about the environment is stored and used. In symbolic artificial intelligence, knowledge is coded into an agent, while in subsymbolic artificial intelligence knowledge is generally learned by an agent through observations (=data). These observations can either be pre-recorded or acquired by trial and error. (The structure is still hard coded)

Which sort of artificial intelligence is better to use really depends on the circumstances: Symbolic artificial intelligence for a complex subject normally requires more time to program in comparison, and needs a programmer well versed in the subject. Subsymbolic artificial intelligence on the other hand depends on Data being available. Of course, it still is complex to program such a system, but the programmer does not have to know the subject at hand. Another aspect to keep in mind is the accuracy: Symbolic artificial intelligence can have a 100 percent accuracy rate when programmed correctly, as it is just logical statements and numbers chained together by some operators, while with a subsymbolic artificial intelligence it is normally hard to get a 100 percent accurate model, and most models are capped at some (high) percentage of accuracy. This is in part because of incorrect/incomplete data, but also because of considerations like time and resource constrains while training. Subsymbolic artificial intelligence can still be more accurate in certain domains: When there is no one correct answer but good data for example. As it is often the case, there is a cost-accuracy payoff, and whether subsymbolic or symbolic artificial intelligence is better suited really depends on the use case. Often they are combined to achieve better results. This work will focus on subsymbolic artificial intelligence.

The requirement for data in subsymbolic artificial intelligence is also the reason why data collection is getting increasingly important in our time.

4.2 Supervised vs unsupervised learning

MA In supervised learning, labeled data is used. In contrast to this, in unsupervised learning, unlabeled data is used instead. Labeled data is easier to use, but harder to get. Labeled data can be a lot of things, and it does not necessarily have to be human-labeled. A label can also be the output of an action, like, for example, the view time of a video on YouTube presented in different ways, or when the data is the weather yesterday, the label could be the weather today.

In supervised learning, algorithms like backpropagation can be used to model a function, where some input-output pairs are given.

In unsupervised learning the artificial intelligence has to find connections and the structure of the data on its own, as no clear input-output pairs are given. Here algorithms like support vector machines can be used to cluster data. Unsupervised learning can be used for, for example, serving ads based on interaction with an agent like Facebook. User data can be clustered and analyzed without the need for labeling.

Of course these approaches can also be combined: In so called semi-supervised learning only parts of the data are labeled. This approach brings with it the convenience of not having to label all the data, while still having the greater accuracy and ease of supervised learning. **semisupervised**

4.3 Parameter(less)

MA In many algorithms, like, for example, neural nets one hurdle is finding a number of parameters big enough to accurately describe a concept, while still not overfitting **??**. The problem here is that this number of parameters is not universal, and has to be newly reconsidered for each problem. To eliminate this need for a decision, there are also parameterless options.

So called parameterless approaches do not base their complexity on the problem, but rather on the training data available. Rather than representing the problem with parameters, often a part of the training data/the whole of it is used to come to a decision. This can be, for example, clustering. One example for a parameterless approach is K-nearest neighbors/Support vector machines **??**

4.4 Classification

Decision trees can be categorized as subsymbolical (more symbolical than other algorithms through), are trained supervised, and are a sort of parameterless algorithm.

Neural nets can be categorized as subsymbolical, are trained supervised, and are a sort of parametered algorithm.

Q-Tables can be categorized as subsymbolical, are trained supervised, and are a sort of parameterless algorithm.

SVMs can be categorized as subsymbolical, can be trained unsupervised, and are a parameterless algorithm.

List of Figures