A short introduction to artificial intelligence

Daniel Schruhl
ThoughtWorks

A short introduction into the general topic of artificial intelligence. This should help address some aspects of artificial intelligence and define them on a shallow level to give a starting point into this topic.

Motivation

There are a lot of definitions for Artificial Intelligence (AI). Most of them are based on a definition of intelligence which in itself is already difficult to define. A simple approach would be to say that humans cognitive functions (e.g. solving complex problems or learning) is considered intelligent. So making programs or machines pursue the way humans might solve problems or learn can be considered as AI. These complex problems can be natural language processing, visual perception, motor control, planning or game theory.

AI programs are sometimes also considered intelligent because from the outside their outcome appear to be magic. When actually developing the program itself and deep diving into the implementation, it leads to the realization of the often trivial nature of the implementation. This illustrates the possible paradox in AI.

AI can be used for image recognition, for controlling machines, to play games or to detect fraud. It can be used in nearly any domain and has already found great usage in medicine or commercial scenarios. Using AI can give a company a major advantage against competitors. Companies like Netflix have embraced AI (Gomez-Uribe & Hunt, 2015) and have established a market leading position also backed by AI.

Types of AI

Approaches to implement AI have resulted into two major paradigms: **symbolic** and **sub-symbolic**. Symbolic approaches model a problem space with tokens or symbols that are humanly readable. This problem space is then processed by the AI programs. The symbols themselves are therefore manipulated and processed. Because of the symbolic nature the AI programs can be completely understood by humans. These programs are often called Expert Systems, Rules engine, Knowledge based AI or Knowledge graph. They were the first paradigms that found usage in the past and are therefore also called Good, Old Fashioned Artificial Intelligence (GOFAI). In theory they try to solve modeled problems in the same abstract way humans think and would solve problems.

The sub-symbolic paradigm also consists of symbols but they are not really human interpretable. These symbols represent more abstract components that can accomplish tasks suitable for AI related tasks. The whole idea about this paradigm is to build the parts that make human cognitive functions possible on a more low level abstraction. It is highly inspired by biology (neurobiology, genetics, evolution) and psychology. This paradigm has found a lot of usage and popularity in the recent years.

Symbolic AI

As stated before symbolic AI solves complex problems by using strategies researchers thought humans would also use. This is done by transforming a problem space into symbols and then applying functions on these symbols. The key here is coming up with an abstract model of the problem space and transforming it into symbols. Because of the nature of this paradigm, expert systems are easier to debug and understand and to control. A big disadvantage is that you need expert domain knowledge in order to solve the problem the way humans would do. Another advantage is though that big data is not needed.

Due to the symbolic nature **logic** can be used to solve complex problems. Logic consists of **semantic** and **syntax**. Syntax defines the representation of logics (symbols). It consists of an alphabet which forms words which in turn form language (**formulas**). This syntax is then interpreted by semantics. Semantics give meaning and interpretation to syntax and can be used to derive new formulas (**calculus**). This kind of system is called **formal system**. For more information about logics and formal systems see (Richardson, 2006).

Formal systems have rules. These rules can be used to infer new rules or answer questions. This is often done by calculus (e.g. resolution calculus). Such a system of AI can be built with PROLOG. An example for this would be the task of parsing natural language into semantic compositions. Theoretical computer science can be used to define natural language with grammar. Inside that grammar are symbols like terminal and non-terminal symbols and productions which form rules. These rules can be used to parse natural language into semantic compositions like verbs, nouns or adjectives. These can in turn be used to be processed. This can be done with **Definite Clause Grammar (DCG)** (Wood,

1990).

Another important aspect of symbolic AI are **searches**. In symbolic AI searches are used in finding solutions in a modeled problem space. The solution can be a path leading to a goal or the goal itself. The key aspect here is again modeling the problem space so a search is usable. This could be for example trying to find the perfect schedule for people with limited amount of time. Searches can be simple algorithms (breadth-first, depth-first) or **informed algorithms** (e.g. A^*). Informed algorithms use **heuristics** that try to estimate the best possible path inside a graph. This should lead to a faster search because some paths can be **pruned**. The intelligent part here lies in defining the heuristic. An example of an AI using pruning and informed search is the Deep Blue chess computer (Campbell, Hoane Jr., & Hsu, 2002).

Another aspect in searches is the **Constraint Satisfaction Problem (CSP)**. In a CSP are **variables** which have value spaces (**domains**). Each variable is attached to **constraints**. These constrains can have multiple arity and define limitations for the domains. This way a net of variables and constraints is formed (see figure 1). The problem of solving the net and narrowing down the domains to a solution is the actual CSP. To solve these problems there are multiple algorithms (e.g. AC-3). CSP can be used to solve riddles or puzzles like Sudoku.

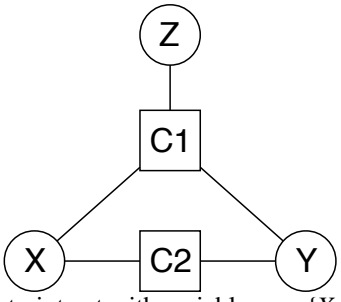

Figure 1. Constraint net with variables $v = \{X, Y, Z\}$ and constraints $c = \{C1, C2\}$. $C1 \mapsto X < Y$ and $C2 \mapsto X + Y = Z$. The domains for the variables are $D_X = \{1, 2\}$, $D_Y = \{2, 3, 4\}$, $D_Z = \{4, 7\}$. The solution would be $D_X = \{1\}$, $D_Y = \{3\}$, $D_Z = \{4\}$.

Searches can also be used to find paths from a start to a goal. This is called **planning**. In planning you have a state which describes a current situation. For the state there are possible moves that can be taken. Each move has a **condition**, **delete** and **add list**. Those lists can be used to test and execute the moves to change the state in order to pass the start state into the goal state. This kind of procedure can be used to control for example robotic arms. For more information see Stanford Research Institute Problem Solver (STRIPS) and (Nilsson, 1982).

Sub-symbolic AI

Sub-symbolic AI uses symbols in a more abstract way. The symbols become part of a low level system that is capable of cognitive functions. It is no longer possible to map concrete high level tasks like identifying objects in images directly to the symbols in the system. The output generated by sub-symbolic AI is therefore often probabilistic or approximated. This has led to more robust models that often showed better performance and scalability. Unfortunately most of these models need big data to get to that point. But these models are no longer bound to expert domain knowledge and have shown way better performance in tasks like visual perception. Sub-symbolic AI consists of machine learning, reinforcement learning and evolutionary computing.

Machine learning

Learning itself plays a huge part in cognitive functions. Learning essentially means incrementally improving performance on a given task. This implies that knowledge is represented in any form and incrementally more knowledge is generated or the already existing knowledge is hardened and diversified. Ways of achieving this would be by adding new data to that knowledge base or by adding time to extract more knowledge of the given knowledge base. Learning not only consists of extracting knowledge and retaining that knowledge of already learned data, but also means handling new data and transferring already learned knowledge (by **generalizing**).

Machine learning accomplishes learning by doing **pattern recognition** in (big) data and generalizing these patterns. Programs that actually perform machine learning are called **models**. Models use **input data** and **output data** (see figure 2). The input data contains **features**. Input data is used to input into the model. Output data is data that should or is output by the model. Machine learning models generate knowledge implicitly by the features of the input data. Machine learning tasks can be grouped into **supervised** and **unsupervised learning**.

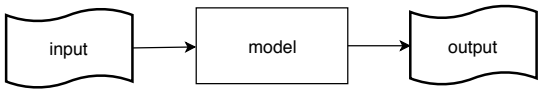

Figure 2. Models use input data to produce output data. The model needs to be trained in order to generate knowledge from given data and generalize from it. The model can then be used to produce output on unseen input data (prediction).

In order to make models work they have to be trained on data. That data is separated into test and train data. The training data is used to feed the model with during training. The test data is used after training to see how well the model is able to generalize on unseen data. The most suggested train and test split is 80:20.

When training the model on the training data a metric called loss or error is calculated. This metric determines how far off the models output is from the original output (high loss or error). The goal of training is to minimize this metric. Depending on the error change rate (derivation) the parameters of the models are tuned to that error derivation. The model itself is therefore optimized through training.

After the training is done the test data can be used to determine if a model is underfitting or overfitting. These terms describe the generalizing capabilities of the model. Generalizing means that general abstract knowledge was generated from the trained on data (learning) and new unseen data can be processed by applying that general abstract knowledge on that data. This means that decisions and assumptions can be made on unseen data. Overfitting means, that the model started to memorize patterns in the data instead of generalizing from it. This can happen when the training is too long or the variance in the training data is not high enough. Underfitting means that the training error or loss has not been well enough optimized yet. This can happen when training is too short or not enough data has been seen.

Supervised learning

Supervised learning contains solutions for **classification** and **regression** tasks. For that input data with given output labels are needed. This kind of data is also called **labeled data**. For classification tasks a **decision boundary** is searched within the data. That decision boundary separates the input data into classes that should classify fish into sea bass or salmons (see figure 3). To train this model labeled data of fish with the features length and skin brightness is used. For each data entry (one fish) the class sea bass or salmon (label) is already existent. The model then trains on this data and forms the decision boundary. If the model gets data of a new fish and its length and skin brightness, it will be able to identify the class this fish would be. It will predict if a fish is a salmon or a sea bass based on its length and skin brightness.

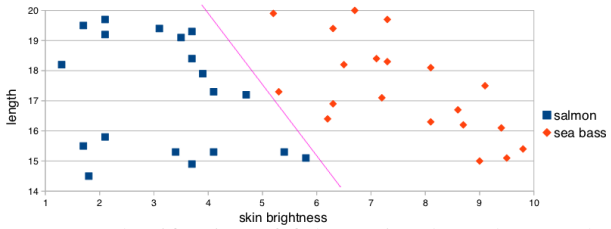

Figure 3. Classification of fish species dependent on their length and skin brightness. The classification model should learn the decision boundary (pink) to successfully distinguish between the two fish species.

For regression tasks the features within the input data are used to approximate other variables for the output. The output itself is unlike to classification tasks **numerical values** and not classes. *Numerical values (input) are therefore used to approximate the values of one or more other numerical values (output)*. An example for this would be the approximated prediction of mileage of cars dependent on their motor power and weight. The regression model would learn a function that approximates the mileage (output) dependent on the input data, which consists of motor power and weight. This model then learned to generalize by the given data and can be used to predict the mileage of new cars it has not seen before dependent on the knowledge it gained before (given data).

Unsupervised learning

Unsupervised learning is used to find patterns in data without labels (**unlabeled data**). The input data does not have labeled output data. This means that no interpretation can be concluded of the data. So using unsupervised learning is often also called **knowledge discovery**. In real life most existent data is unlabeled or partially labeled. This makes unsupervised learning very useful. Unsupervised learning can then be used to **cluster** data. This will group similar data input clusters. These clusters can then be further processed and put into classes for example.

Connectionism

Connectionism tries to solve complex problems by modeling what we use to solve problems - our brain. Neurobiologically the brain consists on a low level of **neurons** that are highly connected. These neurons are structured in **layers**. Subgroups of these layers can be mapped to individual tasks like hearing, controlling motor functions and more. The actual connections and the resulting topology highly specializes the actual function of each of these nets and their performance on different tasks. This can be modeled into a computer science abstraction.

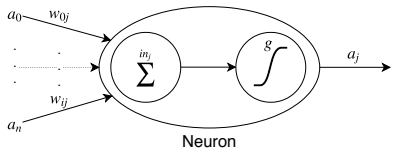

Figure 4. Computer science model of a neuron. A neuron has incoming connections $a_0 ... a_i$. Each of those connections is weighted by $w_{0j} .. w_{ij}$. These connections and weights are multiplied and summed up in $\sum^{in_j}$. The output a_j of this neuron is then calculated by applying the activation function g on the sum: $a_j = g(\sum^{in_j})$.

A neural network is a **graph**. The nodes in this graph are called neurons. Each neuron has incoming **weighted connections**. These are used to calculate the **output** of each neuron (see figure 4). The neurons are arranged in layers. Layers are just collections of neurons. A neural network

trains by adjusting the weights of each connection with an algorithm called **backpropagation**. To speed up that process a optimization technique called **stochastic gradient descent** is used. The way the neurons are connected determines the topology and specification of that network. Different topologies are better at solving different tasks.

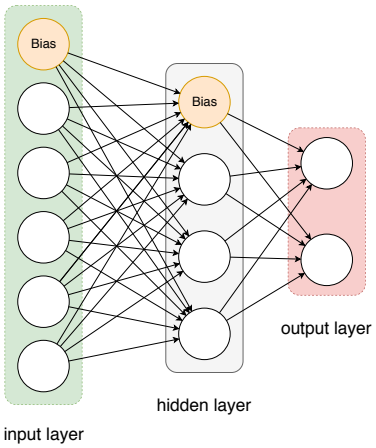

Figure 5. Feed forward neural net with input layer, one hidden layer and a output layer. The input layer and the hidden layer have a Bias neuron that can be used to enhance activation function usage.

The simplest topology is a **feedforward network** (see figure 5). Feedforward networks are networks with **input**, **hidden** and **output layers**. The neurons of each layer are connected to each neuron of the following network. The connections are only in a feedforward direction. This is also called **fully connected**. These networks are good for not so complex tasks.

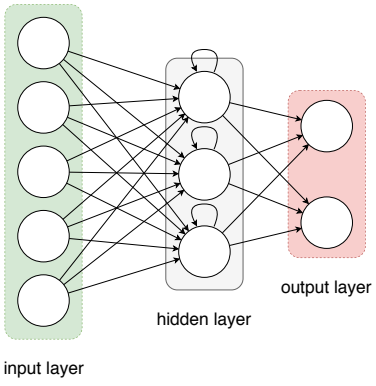

Figure 6. Recurrent neural net with input layer, one hidden layer and a output layer. The hidden layer has neurons that have connections to themselves (recurrent).

Another topology are **recurrent neural nets** (see figure 6). Recurrent neural nets have neurons in layers, that have connections from neurons from later layers or from themselves. This way it is possible to create a memory with the internal states of these neurons. This kind of topology has been shown promising when trying to process sequential data or time series data.

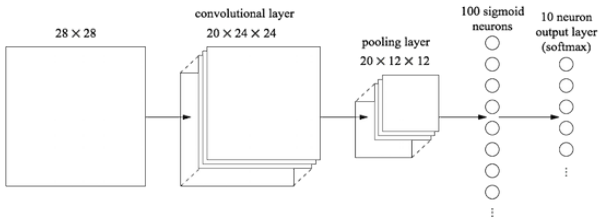

Figure 7. Convolutional neural net with input layer, one convolutional layer, one pooling layer, one feedforward layer and a output layer. Taken from (Nielsen, 2015, Ch. 6)

A similar topology are **convolutional neural nets** (see figure 7). Convolutional neural nets have multidimensional input data (2D, e.g. pictures). They feed through that data with a **receptive field**. This field is a smaller subset that is fed into parallel hidden layers that process that input data. Those hidden layers can then be connected to **pooling layers** that condense the parallel hidden layers and can in the end be fed into fully connected layers. This kind or architecture is great for learning from images or sequential data where neighboring data helds critical information.

Deep learning

Machine learning algorithms perform bad on complex task with a **high dimensionality space**. This is due to the fact that with rising dimensionality the amount of possible combinations explode and the amount of available data to train on diminishes. *The data density shrinks*. This is in contrast to the desired generalization. For generalization it is hard to find answers for data that is not at all represented in any way in the training data. To solve this problem deep learning was introduced. By adding more layers to a model and making the model deeper this way, the model is able to learn more **abstract meta features** in the underlying training data. In this way **feature extraction** is being done automatically.

Reinforcement learning

Reinforcement learning is based on human psychology. A possible way humans might learn is by **rewards** or **penalization**. So learning is done by **reinforcement** (positive or negative). The model for this contains an **agent** that lives in an **environment** (see figure 8). The agent can take possible **actions** inside that environment. The actions then change the **state** of the environment. The result of that action is also a **reward**. This reward and the changed state are then perceived or processed by the agent. The agent tries to maximize the reward. Rewards are given by for example finding a

goal. The agent has to run multiple **episodes**. Each episode contains a start and an end. On each start the state of the environment is reset. Eventually the agent should then learn the most optimal way to achieve a goal and solve a complex task.

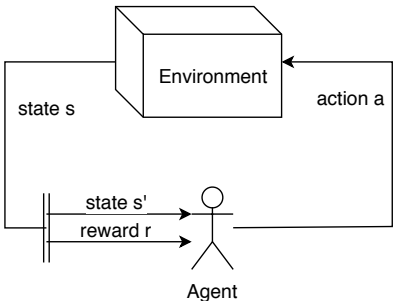

Figure 8. Model of reinforcement learning. An agent performs actions in an environment and receives rewards. The state of the environment gets changed as soon as the agent performed an action. The agent tries to optimize the reward.

In doing so an agent can either **exploit** or **explore**. Exploitation means the agent finds a way to get to the goal but starts to only replay this way of achieving the goal and might not find a more optimal path. Exploration means the agent tries out different paths to get to the goal and might not be able to find it. It is a trade off the agent has to do to be optimal. A way to overcome this problem is **Thompson Sampling** (Chapelle & Li, 2011).

Some popular algorithms for reinforcement learning are **Q-Learning** (Watkins & Dayan, 1992) or **Sarsa** (Berges & Pryzant, 1995). Reinforcement learning was used in AlphaGo and AlphaZero (Silver et al., 2017).

Big Data in context

As stated before a lot of AI applications need a lot of data. When it comes to machine learning the more data the better. The more data with high variance you can get the better your model will be able to generalize. That often means handling big amounts of data. This brings difficulties regarding extracting, transforming and loading data (**ETL**). Big data describes programming practices and techniques in dealing with big amounts of data when doing for example ETL jobs.

Often machine learning applications need data from a series of different sources. The data in that case is also often in formats that can not be directly processed by machine learning applications. Extra efforts have to be made to make the data ingestible for machine learning applications. This includes loading the data into one single format, often on something like a **datalake** in a format that is also called **datamart**. A **datalake** is nothing but a massive store that is easily accessible and flexible in regards to storing data. A **datamart** is a subset of aggregated data from different sources. It is often also transformed in a way that it can be further ingested by machine learning applications. Because we are dealing with big amounts of data this would take a lot of time with conventional programming methods. Big data often incorporates the use of a programming model called **MapReduce**. This allows operations on data on a highly parallel and concurrent matter which significantly speeds up the process. The resulting data from all those processing steps that is usable by machine learning applications is also often called **dataset**.

Recent advances in machine learning and further topics

Recent advancements in neural networks include **capsule networks**. Capsule networks are neural nets that can be used for image recognition. They are inspired by neurobiology and what our brain structures for image recognition look like (Sabour, Nov, & Hinton, 2017). Capsule networks are similar to convolutional neural nets but instead of convolutional layers have capsules. These capsules are groups of neurons that are trained to recognize objects (with their instantation) and orientations of those objects.

Another field of neural networks that is getting more attention in the last years are **generative models**. Generative models consist of neural nets that generate data. Generative models can be **autoencoders** or **Generial Adversial Networks (GANs)**. They are trained to generate output based on the abstract features they learned from given input. This in itself represents a unsupervised problem. You can make it a supervised and trainable problem though by training input data and expecting the same input data again (autoencoders) or training a different neural network to become a **discriminator** and tell whether the created data was artificial or not (GANs).

When it comes to handling a high dimensional input space, the data often comes with a lot of noise. A way to address this problem are **attention mechanisms** that make a neural net focus on certain parts of the input data at a certain time (Bahdanau, Cho, & Bengio, 2014). In that way attention is introduced to the neural net. With this mechanism it is possible to get more meaningful features from input chunks that make more sense in an isolated environment before taking everything else into consideration.

Other approaches are combining connectionists models with other paradigms of artificial intelligence. That includes combining reinforcement learning with neural nets, where you make a neural net optimize against a reinforcement learning error (Q-learning). This is called **Deep Q-network (DQN)** (Mnih et al., 2015). Also combining evolutionary computing with neural nets has led to better overall performances. This can be done by **neuroevolution** (Sher, 2013). Neuroevolution describes taking neural nets and considering them as **replicants** or **survival machines**. These replicants have a phenotype and a genotype. They also multiply and can be manipulated, by manipulating the phenotype. When they multiply random mutation can occur that changes the

phenotype. This will lead to new generations of those replicants. At the end of each generation the replicants are measured against a **fitness function** that determines how well the replicants can survive. This means solving a given task in an environment. These approaches can go so far to not only change the hyperparameters of the neural net but also change the whole topology. This kind of neuroevolution algorithm is called **NeuroEvolution of Augmenting Topologies (NEAT)** (Stanley & Miikkulainen, 2002).

Further readings

To get into the basics deeper and to learn more about some neural network architectures refer to (Goodfellow, Bengio, & Courville, 2016). This book is available online and contains nearly everything you need to know about neural nets and deep learning.

If you want to get a practical approach in the field of neuroevolution, (Sher, 2013) is a good start.

Glossary

AI Artificial Intelligence.

Connectionism is using graph theory to mimic neurobiological features. Each node in the graph is called neuron. Neurons are connected in specific ways identifying the overall type of network..

CSP Constraint Satisfaction Problem.

DCG Definite Clause Grammar.

DQN Deep Q-network.

Expert Systems Synonym of symbolic AI.

GANs Generial Adversial Networks.

GOFAI Good, Old Fashioned Artificial Intelligence.

Knowledge based AI Synonym of symbolic AI.

Knowledge graph Synonym of symbolic AI.

NEAT NeuroEvolution of Augmenting Topologies.

PROLOG Logical pogramming language. For more information consider: `http://www.learnprolognow.org/`.

Rules engine Synonym of symbolic AI.

STRIPS Stanford Research Institute Problem Solver.

Sub-symbolic AI is a AI paradigm solving complex problems in a more low level system where symbols represent abstract structures that enable cognitive functions.

Symbolic AI is a AI paradigm solving complex problems the way humans would. Modeling the problem space with symbols and use computations to transform them into a solution.

References

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural Machine Translation by Jointly Learning to Align and Translate. , 1–15. Retrieved from `http://arxiv.org/abs/1409.0473` doi: 10.1146/annurev.neuro.26.041002.131047

Berges, V.-p., & Pryzant, R. (1995). Reinforcement Learning for Atari Breakout. , 1–8.

Campbell, M., Hoane Jr., a. J., & Hsu, F.-h. (2002). Deep Blue. *Artificial Intelligence*, *134*(1-2), 57–83. Retrieved from `http://www.sciencedirect.com/science/article/pii/S0004370201`

Chapelle, O., & Li, L. (2011). An Empirical Evaluation of Thompson Sampling. *Advances in Neural Information Processing Systems*, 2249—-2257. Retrieved from `http://explo.cs.ucl.ac.uk/wp-content/uploads/2011/05/An-Empir`

Gomez-Uribe, C. A., & Hunt, N. (2015). The Netflix Recommender System. *ACM Transactions on Management Information Systems*, *6*(4), 1–19.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... Hassabis, D. (2015, feb). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529–533. Retrieved from `https://doi.org/10.1038/nature14236` doi: 10.1038/nature14236

Nielsen, M. A. (2015). *Neural networks and deep learning*. Determination Press.

Nilsson, N. J. (1982). *Principles of artificial intelligence (symbolic computation)*. Springer.

Richardson, D. (2006). *Formal systems , logic and semantics.* Department of Computer Science, University of Bath. Retrieved from `http://www.cs.bath.ac.uk/pb/EMCL/DS/DS-Ref-2011/c19.pdf`

Sabour, S., Nov, C. V., & Hinton, G. E. (2017). Dynamic Routing Between Capsules. *Nips*.

Sher, G. I. (2013). *Handbook of neuroevolution through erlang*. Springer New York. Retrieved from `https://doi.org/10.1007/978-1-4614-4463-3` doi: 10.1007/978-1-4614-4463-3

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Hassabis, D. (2017, 10). Mastering the game of go without human knowledge. *Nature*, *550*, 354-359.

Stanley, K. O., & Miikkulainen, R. (2002). Evolving neural networks through augmenting topologies. *Evo-*

lutionary Computation, *10*(2), 99-127. Retrieved from http://nn.cs.utexas.edu/?stanley:ec02

Watkins, C. J. C. H., & Dayan, P. (1992, May 01). Q-learning. *Machine Learning*, *8*(3), 279–292. Retrieved from https://doi.org/10.1007/BF00992698 doi: 10.1007/BF00992698

Wood, M. (1990). Prolog and natural-language analysis. *Science of Computer Programming*, *14*(1), 110.