Sighted MicroPsi Agents in Minecraft:

Object Recognition Using Neural Transfer Learning and Automated Dataset Collection

by

Dirk Meulenbelt

A document submitted in partial fulfillment of the requirements for the degree of
MSc Artificial Intelligence
at
Utrecht University

Abstract

We need intelligent agents that interact with an open world through perception and action. Although previous research has investigated how cognitive architectures could be interfaced with virtual worlds, it is not yet known if and how we could build an interface with a virtual world that meets the characteristics for suitable Artificial General Intelligence (AGI) research environments. Vision is arguably our primary source of information. It is though unclear, how we can provide an agent with vision without world-specific annoted data. To address these challenges, the Malmo API for Minecraft is connected to the MicroPsi cognitive architecture, which I have tested against AGI virtual world requirements laid out by AI researchers. In order to bootstrap vision, transfer learning is applied on automatically extracted data from an experimental setup in Malmo. Agents can use this network to recognise objects in-game in real time, with high accuracy. This shows that networks trained outside of virtual worlds can generalise to in-game objects, when they are provided with a few in-game examples. This provides a basis for research on intelligent agents in virtual worlds.

Contents

1 Introduction

Artificial Intelligence (AI) has, as a field, evolved from an ambitious attempt to (reverse) engineer intelligence, into a more constrained project whereby well defined problems are tackled with approaches that are catered to a specific task, or tasks alike, so much so that the original goals of AI have now been re-branded into a sub-branch called Artificial General Intelligence (AGI). These original goals have been that of understanding, recreating and superseding human intelligence. These goals are still very much unresolved and their attendant questions are still very much unanswered. Could we make this human level intelligence, whereby the world is explored, understood and manipulated, in a way that a human does this, or at least with the same broad capacities? Human intelligence may be and often is taken as a benchmark for AGI, as it is currently our only real example of general intelligence.

It's becoming increasingly clear, though, that specifically human performance, in the sense of doing at least as well as a human, has been surpassed on a variety of tasks already, such as the game of Go [74], the quiz-show Jeopardy [25], image classification [28], and recently even No Limit Texas Hold/Em [51] which I've enjoyed playing myself and had always imagined to be such a 'human intuition' game, especially when playing other humans. Additionally, there are various tasks that humans were never capable of to begin with, or at least not within reasonable time limits. The simplest example (though arguably not AI) is a calculator, but more recent examples may be inferring high-quality images from blurry pictures 'CSI style' [62], and creating lots and lots of photo-realistic images of non-existing celebrities [34]. Currently, the field of artificial intelligence is rife with bullish prophecy; machine learning scholar and entrepreneur Andrew Ng recently said: "If a typical person can do a mental task with less than one second of thought, we can probably automate it using AI either now or in the near future." [55].

However impressive the headlines, the bulk of AI's current success stems from improvements in an approach called deep learning. Deep learning is a subset of the field of machine learning, which comprises all techniques that allow computers to learn from data. Deep learning stems back decades though saw its resurgence a few years ago when researchers managed killer results on the ImageNet classification challenge [37]. The reason that deep learning wasn't popular before but is popular now is mostly because recent improvements in hardware and a recent increase in available data have made it possible to apply this technique fruitfully but it was functional though slow before. Deep learning is a statistical technique characterised by the use of multi-layer perceptrons, also known as neural networks, with multiple hidden states that build models based on sample data. Neural networks take a certain input, such as the pixels of an image, or words in a sentence, on various input nodes. These input nodes are connected to nodes in 'hidden states', often many (hence, deep), which are themselves connected to the following hidden state, and ultimately the nodes of the output layer.

Mathematically speaking, matrices of input activations are multiplied by matrices of weights, and its subsequent multiplications are squeezed by activation functions for every node in the network, for it to be able to learn non-linear functions. The neural network builds a model of

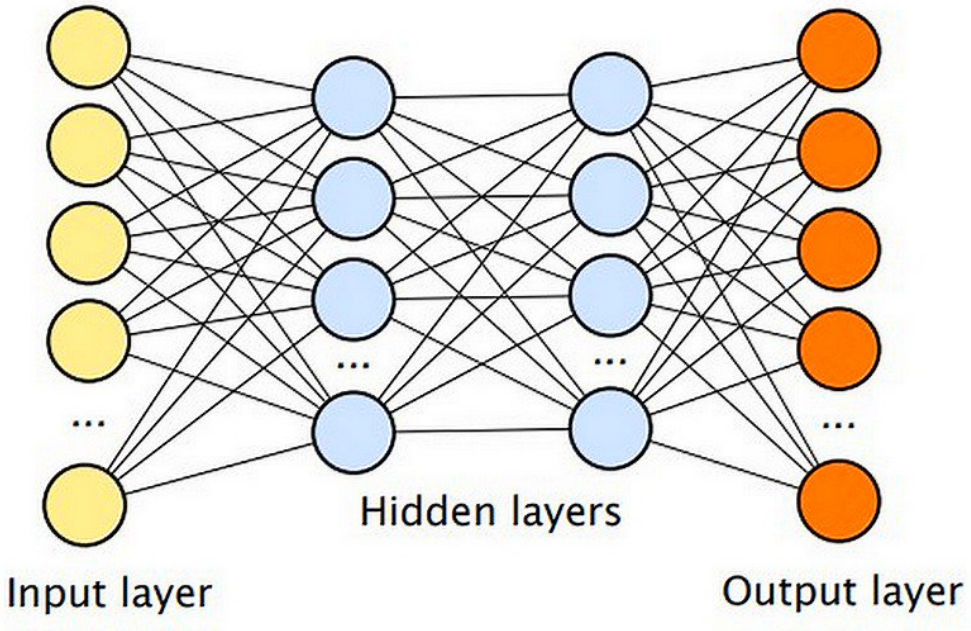


Figure 1.1: A neural net with two hidden layers.

its labeled input data, such that when it is confronted with an example it has not seen before, it can apply this model to make a prediction of a label.

Contrary to what nomenclature suggests, the technique is only *inspired* by its biological counterpart, and could hardly be called neural at all. The name stems from the fact that the nodes in the network can be thought of as neurons, and the activation function can be likened to an action potential. It's however doubtful that neural networks are even biologically plausible [45], because backpropagation, the technique that allows neural networks to learn, requires the 'neurons' in the network to communicate in a backward direction, which biological neurons do not do. The technique's very own champions don't even pretend otherwise; Geoffrey Hinton, the scientist that originated, or at least popularised backpropagation recently stated that we have to go back to the drawing board because it is definitely not "what the brain is doing" [43]. Andrew Ng agrees, and in his ever popular course on deep learning, tells students that although neural networks were inspired by the brain indeed, that's about as much as anyone should expect from it as far as biology is concerned.

Of course, the success of deep learning and its derivatives may prompt one to ask "what gives?" when confronted with its biological infidelity. Results are results: AI could just employ a strategy of 'whatever works'. Of course, planes don't flap their wings, but this kind of reasoning would only be permissible, from a practical point of view, if indeed the current approaches need only be extended and improved to produce the real deal, and to produce actual general intelligence of the kind that does not need any reprogramming to tackle a new task. As the current rhetoric goes: we need more data, more computer power, and ever more layers and layers and layers. There are reasons to doubt this would actually boost results as it has before. While there is no reason to dismiss deep learning as an approach, it is crucial that

we look for more techniques, as there is still much to learn from Cognitive Science. As I see it, there are two main reasons to draw from Cognitive Science and keep the mind in mind:

Reason one: current approaches to AI perform poorly still at various aspects of human intelligence, such as language, common sense reasoning, representing causal relationships, abstract ideas, instability, and logical inferences [45]. Simply adding more data, or computing power, may not be enough. To give an example, Google Translate saw staggering diminishing returns on its language data. A corpus of 75 million words in Arabic to English translation produced a sentence accuracy of 48,5 per cent. Increasing this corpus to 5 *billion* words, yielded only a slight improvement with 51,5 per cent sentence accuracy [26]. Another example of a current shortcoming in AI deals with common sense. AI researcher Hector Levesque [42] proposes that we come up with better tests for AI, better than the Turing test at least. He proposes that we use, among other thigns, Winograd schemes. Consider the following two sentences: "The city council denied the demonstrators a permit because they feared violence", versus "the city council denied the demonstrators a permit because they advocated violence". It's immediately clear to anyone what "they" refers to in either sentence. There is however no syntactical rule of language that decides this, and it is simple knowledge gathered through living on earth, about city councils, and demonstrators, and human aversion to violence, so basic that nobody even bothers to discuss them, make for the deciding principles in this anaphoric resolution. AI as it functions does equally poor at understanding causal relationships because it only looks at correlations between features of its input data, unless explicitly otherwise instructed. A deep learning application will find a correlation between shoe size and mathematical ability, and predict one from the other. It however has no understanding on how that correlation came about [45]. And even if adding lots of data could improve the system or even level it towards a human level or extra-human performance on the abovementioned tasks, then it might still be countered that such data is by no means necessary to train a human, a human child for example, on such matters. It only takes a single instance of pointing out "hey look, a rabbit" to a child, before it can from then on classify rabbits. Imagine the drudgery of having to go through slide shows full of rabbit pictures before children can finally understand what they're looking at. Plus, it is a lot of work to gather all this data. The tremendous successes that have been achieved on ImageNet classifications still only involve a fairly small set of categories (mostly dogs, apparently). To do this for every category in the world would be practically impossible, if not just incredibly tedious. The main issue with deep learning is that it learns correlations between sets of features without any structural assumptions and understanding of supporting causes. Simply increasing computing power, which has helped before as it was very computationally expensive to train deep models, isn't going to help when there are algorithmic limitations. To reach Artificial General Intelligence, we are going to need more than just a deep learning approach. If all of human intelligence's capacities can't be solved by deep learning, with or without increased data and computing power, then what are we to do? Well, instead of mindlessly going ahead and figuring out intelligence from first principles, we could look to cognitive science. Our very own minds have plenty to teach us about intelligence and how to bring it about artificially. Furthermore, we should not neglect symbolic approaches [45] to AI, as many topics with discrete elements are still poorly understood. By no means should we discard deep learning and return to AI's symbolic predecessors altogether (the present thesis doesn't even do this) but we should at the very least attempt to take from cognitive science that which we can take to improve current approaches.

Bringing about intelligence artificially could also teach us about our own minds, which leads me to reason number two: even if deep learning or its offspring manage to solve any or all of its current problems—which is indeed unlikely—it would still fail at providing insight into the human mind specifically, which would be interesting if only to satisfy our burning curiousity. And also to cure various ailments, and/or improve human intelligence using cool implants, such as those investigated by Elon Musk's latest pet project Neuralink [71]. The failure to create insight into the nature of intelligence is due to the fact that deep learning approaches are notoriously opaque, and its creators often have no idea why and how it arrives at a certain conclusion over another [45]. When you are multiplying matrices with many millions of parameters it can be hard—or downright impossible—to locate the faulty or to pinpoint the successful. The lack of understanding in AI sometimes leads to completely unintelligible mistakes, such as IBM's Watson, in the Jeopardy quiz-game, answering "Toronto" to a question in the category of U.S. Cities [12]. In this case, even the creators of the system didn't have understanding, for they were clueless as to how such a farce could have been permitted. Deep learning though can show us interesting new behaviour that wasn't explicitly programmed, even to the bafflement of professionals in the problem or field it attempts to pursue. The recent victory over the world champion Go player Lee Sedol by AI system AlphaGo [74] saw Sedol dumbfounded by the apparent creativity behind some of the moves. Apparent creativity, with emphasis because while of course these insights are interesting for the studious Go player, who'll attempt to improve their own grasp of the game, it is hard to derive any sort of philosophical insight into thinking so far as the human mind is concerned, or intelligence at all.

A way to take the human brain into account when conducting AI research is through cognitive modeling. Cognitive modeling is an approach to understanding and producing theories in cognitive psychology whereby computer models are made of cognitive processes. Cognitive models can teach us about the functioning of the brain, because building these simulations forces us to formalise the processes and allows us to inspect the formalisation: the working code. What is it doing? The goal here is not to re-create the brain with its every neuron and its every lobe. Given the multiple realizability of computation, which states that mental states can in principle be run on any kind of hardware [14], we don't have to recreate a biological brain, or even a synthetic brain, to run the programs that our brains run. We need merely look at what the brain is doing, not at exactly what substrate it is using to achieve all of this. The scope of the models are typically limited to a specific task or process. Cognitive models can then exist within a cognitive architectures. In other words, the architecture is the greater whole and defines the constraints within which the models are created, such that the models "listen to the architecture" [59].

The cognitive architecture the current paper focuses on is MicroPsi. MicroPsi is an interesting architecture because it takes the motivational system as first principles for cognition [10]. Another interesting feature of MicroPsi is that it combines both symbolic and sub-symbolic approaches to intelligence, which I will explain in due course. Such allows us to take from deep learning that which works, and integrate it into an architecture that could combine these approaches with symbolic AI (e.g. planning, language). MicroPsi creates agents that are situated within a world. Worlds are the best way to expose agents to the variety of tasks and situations that gave rise to human intelligence in our world. Given the constraints on what is feasible (building full on robots is out of scope, certainly for this thesis) these worlds are primarily virtual worlds, though MicroPsi is not limited to application in virtual worlds only as it is

currently used for robot arms in the real world as well [46]. As it stands, worlds connected to MicroPsi or other cognitive architectures lack in scope and opportunity when it comes to demands made on such environments, which relate to arguments made towards embodied cognition, MicroPsi's instantiation of the motivational system, as well as demands previously laid out by AGI researchers [39]. In any case, an excellent instance of a virtual world provides a low-weight analogue to the real world such that it provides opportunities aplenty for agent and multi-agent experiments. One of this papers contributions is to single out Malmo [31], which is a platform based on top of Minecraft, as a stellar candidate for AGI experimentation that well answers to the demands.

Robot arms notwithstanding, the most important way that humans interface with the world is through vision. Currently, the most successful way to interact with a virtual world with visual information is using (perhaps ironically) neural networks, most notably the convolutional variety, which I'll explain in due course. Using convolutional networks within the MicroPsi context provides an excellent way to integrate cognitive architectures and deep learning such that we integrate symbolic and sub-symbolic approaches, and draw from both approaches their respective strengths. Training neural networks to recognise image data typically requires a lot of example images labeled with the class they pertain to. Object recognition in a virtual world, built from scratch, is going to require a lot of labeled visual data from that particular virtual world, which is likely unavailable or tediously gathered. This thesis contributes to this problem by implementing a solution to create a preliminary way of semi-automatic data extraction from a virtual world, in this case Malmo. Then, a convolutional neural network that is pre-trained on ImageNet is used to cope with the small amount of data such that it can be used to recognise objects using the extracted data from Malmo.

2 Cognitive Modeling and Cognitive Architectures

2.1 Cognitive Modeling

The idea that we can build strong AI on computers, at least in principle, is based upon two assumptions: *the computational theory of mind*, and *multiple realisability* [14]. The computational theory of mind states that the brain is an information processor. The cognitive psychologist Steven Pinker [58] writes:

> Mental life consists of information-processing or computation. Beliefs are a kind of information, thinking a kind of computation, and emotions, motives, and desires are a kind of feedback mechanism in which an agent senses the difference between a current state and goal state and executes operations designed to reduce the difference.

This view, that the mind is not *like* a computer, but *is* a computer that engages in computation, is the most basic assumption made by AI researchers that believe that we can create minds using our computers.

The second assumption, that of multiple realisability, assumes that there are a lot of different physical architectures that can give rise to a mind. Futhermore, a Turing-complete machine could in principle simulate any other architecture, such that, for example, a non-binary mind or system may run on a binary architecture, and vice versa. The corollary is that minds don't have to run on meaty brains, and we could build minds out of different computer architectures.

Arguments for the computational theory of mind and substrate independence are made in the literature, and I will assume them here, though they are not entirely uncontroversial. I will only add that if minds are not computers, then we could potentially find out by cognitive modeling, as we are doomed to run into the proverbial wall at some point.

If consciousness and love and everything in between are computations, and these computations do not have to take place in a brain made out of meat, then this opens up the possibility to build computational models of the mind. Cognitive modeling involves building computer models of mental behaviour to understand how the mind works by building it. The models can serve for comprehension, but also prediction, and in the case of artificial intelligence: replication.

To show how various perspectives can give rise to different types of insight, the philosopher Daniel Dennett [19] distinguishes between the *physical stance*, the *design stance* and the *intentional stance*.

The physical stance is taken on when one looks at the physical constitution of systems. This is the de facto position of the physicist, and this is how we learn the physical make-up of, say, a brain, or a flower, or a chair. We could in principle predict a person's action knowing their complete physical make up and the forces impacting this at some point on time. This is however not practically possible.

What we do then, is take an intentional stance, where we are concerned with beliefs, desires and thinking; explanations are conceived in terms of mental states. This is what we do when we try to understand the people around us—and even ourselves.

Lastly, between the physical stance and the intentional stance, there is the design stance, which doesn't involve looking at the physical constitution but at how things are constructed: how it functions and what its purposes are. Cognitive modeling allows us to take a design stance to study the workings of the mind. This is a worthwhile pursuit because this allows us to learn by doing: our code will serve to give us insight into what is going on within the mind.

Often, in psychology, only the output of behaviour is observed, as participants of some experiment may be confronted with some stimulus, and their response is measured. What mental processes have caused that behaviour is not clear from this information. In the field of neuroscience a similar experiment may be conducted, and instead of measuring the response, a brain-scan may be made of the participant, and from this it can be seen which part of the brain was involved in the computation. This approach also gives scarce insight into the workings of the mind, for this approach is akin to opening up a computer to figure out how Microsoft Word works. In fact, neuroscience doesn't even seem to be able to figure out the working of systems of which we already know how they work [32].

Clearly, neither psychology nor neuroscience approaches alone give very good insight into what exact functions are implemented within the mind. Cognitive modeling serves to bridge this gap. By building models of the mind we get to look at our code, and we get to test it against psychological research: it's testable. The input/output of the model can be compared to the input/output of that which it models: the brain. Psychological research may thus work in a feedback loop with the cognitive modeling approach. Likewise, neuroscience can give hints as to the kind of computation that is taking place. If we see that a particular task makes heavy use of the visual parts of the brain, then this could mean that the tasks makes use of mental imagery, as may be the case with many language tasks. Doing this, we can base our understanding of intelligence around psychological and neuroscientific research, without having to rely on one-sided explanations.

This 'learn by doing' approach not only helps to serve our understanding of the intricacies of the mind, but can also help us to build artificial intelligence that is ultimately more general, as the human mind is our best example of general intelligence.

2.2 Cognitive Architectures

Cognitive models pertain to specific tasks and events and are typically contained and constrained within cognitive architectures, though the distinction is somewhat blurry as cognitive architectures can be seen simply as larger models. The architecture explains what the brain can do and can't do and might set limits on memory, bandwidth and so forth. Cognitive architectures are then not 'Turing-complete', as limits are set on what they can and can't compute, such as a limit on available memory, bandwith, and so forth. Cognitive architectures are, to quote one of its pioneers, "a specification of the structure of the brain at the level

of abstraction that explains how it achieves the function of the mind" [4]. This means that the overarching structure of an intelligent system, with which models can be constructed, is specified by the cognitive architecture. Typically, an architecture defines parameters that remain constant over time, such as long- and short-term memory, the representations of elements within memory, and the functional processes that operate on them, such as learning processes [40].

While cognitive architectures set constraints on what can and can not be computed, they are unlike narrow AI in the sense that they do not define what has to be computed within these parameters. The architecture may set a limit on memory, but does not have to say anything about what can go into that memory slot, and the architecture can be generally applicable to a variety of tasks, without having to specify anything relating to any particular task, unless this is called for. Assumptions about language, for example, may be innate and built into the architecture.

Cognitive architectures are based on the idea that we need a more unified theory of how the mind works. Back in 1973 Allen Newell argued that we need more systems-level research into cognitive science and AI [54]. This means that we take an overview instead of looking into all sorts of isolated experiments in psychology without any way to connect these into a coherent framework. Since Newell's call to arms, much more research has been conducted in this domain [40]. However, this type of research still leaves wanting: there are still far too many experiments in psychology that are not connected to each other in some common coordinate system, and the field of psychology seems rather uninterested in building overarching theories of how the mind works [40]. Cognitive architectures can serve as a common denominator for cognitive science research. We need complete and integrated systems, instead of islands [9]. Mental processes are intertwined with each other, such that emotion affects perception and motivation accepts emotion and twice versa. Such processes really should not be studied in isolation as reality and intelligence simply aren't divided up to perfectly match departments in a university or sections in a library.

Currently we can distinguish between two major categories of cognitive architectures, that come from different angles [9]. On the one hand there are the Fodorian Architectures, which are based on a kind of mentalese, or 'language of thought' [21], which is typically rule-based. Examples such as these are ACT-R [5] and Soar [38], and they are built step by step by adding more functionality when this is called for, essentially coming from nothing and being as parsimonious as can be until a new task may not be tackled by current machinery.

On the other side are connectionist approaches, such as PDP [60], and the Harmonic Mind [63]. These architectures impose constraints on a system until we have a kind of mind. This may make more sense from an evolutionary point of view, as brains have likely not evolved in small functional increments, but mostly through scaling and local tuning [9]. The problem is that such connectionist approaches are still notoriously bad at typically symbolic processes such as language and planning.

Given the strength of connectionist approaches for many task such as image recognition and robotics, taking a purely symbolic approach is likely going to be insufficient. Likewise, given the need for symbolic approaches to language and planning, taking purely connectionist approaches may never converge on a proper model. For this reason, the best approach is to incorporate both symbolic and sub-symbolic approaches into one architecture, such that we can profit from their respective strengths. This has been done in for example Clarion [66], Lida [24], and Mirrorbot [15], and is aptly named neuro-symbolism. This paper focuses on

the cognitive architecture MicroPsi [6], which distinguishes itself from other architectures not only through its neuro-symbolic approach but also because it conceives of agents not as goal-oriented, but goal-setting first, taking motivation as the first principles of cognition and as a why to behaviour.

3 The MicroPsi Cognitive Architecture

MicroPsi is a cognitive architecture created by Joscha Bach [8]. MicroPsi is based on Psi-theory, which is an overarching theory of cognition [20]. Most of psychology to date does not involve itself with such grander theories [53]. The 'Micro' in MicroPsi is there to say that the architecture does not encompass all of Psi-theory, but is a start at least. As stated, MicroPsi distinguishes itself from other, more established architectures, by focusing more on motivation, neuro-symbolism, and (embodied) interaction with an environment. The result is a whole, testable framework that simulates (intelligent) agents.

Motivation is paramount in Psi-theory because cognition is unlike a chess-computer in the sense that it doesn't care about arbitrary goals, unless those goals are there to serve a need. In other words, we don't do things for no reason, and reasons can be grounded to the demands of the system. I will highlight the import of this focus as it is one of the reasons why I believe MicroPsi stands out among its peers.

Another aspect that sets MicroPsi apart from the rest is that it combines both symbolic and sub-symbolic representation. Most of current successes in deep learning are a prime example of sub-symbolic activation, but some have recently argued that symbolic representations should get a bit more attention, most notably because sub-symbolism is still seriously lacking in some domains [45]. The neuro-symbolic approach aims at universal mental representations that incorporate the strength of both compositional and distributed representations. Psi-theory conceives of AGI or general problem solving as operations over these neuro-symbolic representations.

The reason that MicroPsi's agents are embodied, or rather situated, in a world, and what we want from such a world, will be addressed in the next chapter.

An interesting practical feature of MicroPsi is that it easily allows setting up a *request confirmation network* [11], which is a general paradigm for neuro-symbolic plan execution devised by MicroPsi's creator.

Only those aspects of MicroPsi relevant to the current thesis are outlined here. For a comprehensive overview of the MicroPsi architecture please see [8]. General information is drawn from that book.

3.1 Motivation and Emotion

3.1.1 Needs

General intelligence in humans did not evolve only to reach some goal or another, but to come up with that goal in the first place. Humans are organisms with many conflicting demands that have to be met in an ever-changing complex world that isn't always so hospitable to these demands. One part of a sound strategy to achieve human-level intelligence could be to give a

virtual agent a similar impetus. The Psi-theory way to look at this problem is to not conceive of any pre-set goals, but to resort to a minimal set of needs that the cognitive system has to meet [7]. The needs, or demands, are signaled to the agent through an *urge*, or *urge indicator*. The urge can have varying strengths leading to a varied urgency for the corresponding demand. An example of a need is nutrition, and an urge indicator is hunger, which can have varying degrees of intensity, as we all know, as it can be felt in varying degrees of intensity. The urge may swing into both directions, and often uses different signals for each end, such that feeling hungry and feeling starved feel very different.

The intensity of urges govern arousal, execution speed, and cognitive processing [7]. Satisfying an urge (rapidly) results in a pleasure signal (e.g. joy), which comes down to a message that says: 'do more of this'. Not satisfying or even aggravating an urge leads to a displeasure signal (e.g. pain), that serves to tell you: 'do less of this'. These signals function as reinforcements for motivational learning, where the current situation can be associated with the urges and their satisfaction or aggravation. Motivational learning can strengthen this association.

Goals are actions or events or situations that are associated with the satisfaction of a need in an environment. Cognition sets goals to meet demands, and can set and change these goals, but cannot change the needs directly (unless perhaps you are a trained monk). Since needs change all the time, an agent may sometimes be hungry during some task, sometimes not. The system will have to consider its goals and behaviors in real time, which results in a dynamic evaluation of activities and opportunities.

Motives are urges that have a particular goal associated with them. For example, the satisfaction of the need for romantic affection, which signals itself through a kind of loneliness, may be associated with going to a pub, or a dating website, and this urge may thence serve as a motive for these events. When there are no motives, the agent is left to explore (which is itself a need).

3.1.2 Types of needs

I will give a full list of all the needs that are currently put forward by Bach, because it is important that any world that is connected to MicroPsi contains elements that a need can be satisfied with. Needs come in three groups: *physiological* needs, *social* needs, and *cognitive* needs. Needs are not structured in a hierarchy, and are all active at the same time, at the same level. Needs are associated with a weight, which modifies the need relative to other needs; a decay, which sets how quickly a need decays and needs fulfilment; a gain, which sets how quickly a need fulfills; and a loss, which reacts to how the needs respond to frustration or failure.

Physiological needs relate to the basic survival and integrity of the organism and will include all that is necessary for this organism to stay in physical order. Physiological needs are not exhaustively listed by Bach [7], but are said to include *food and water* (nutrition), *pain avoidance*, *rest*, avoidance of *hypothermia* (being too cold), or *hyperthermia* (being too hot), and *libido*, which is the physical need for sex and a bridge from the physical needs to the social needs, as it takes two to tango.

Social needs involve other people. That is why they are called social needs. Because needs are not situated within a (dominance) hierarchy, humans may jeopardise their physiological well-being in order to satisfy social needs (like a hero). Social needs are *affiliation, internal legitimacy, nurturing, romantic affection, and dominance*.

Affiliation is the need to be recognised and accepted by other individuals. Smiles and praises and texts that are replied to can be seen as *legitimacy signals*, whereas a frown, a demotion, and unrequited blue check-marks can be seen as *anti-legitimacy signals*.

Internal legitimacy or "honour" is a kind of affiliation with internalised social rules, and what drives someone to return a wallet found while alone.

Nurturing is the need to care for other people. Bonding may occur after repeated reciprocal nurturing [7].

Romantic affection is the need to form a bond with someone that results in courtship. This is related to the physical need of libido, but is rather more concerned with building a relationship with a single person. (Or possibly a few.) Love, in the romantic sense, is not a need but an emergent factor from social needs compared with libido satisfied by some particular individual.

Dominance is the need to struggle for a high position within a social hierarchy, or to maintain such a position.

Cognitive needs are related to skills, play, creativity and exploration. Cognitive needs are *task-related competence, effect-related competence, general competence, exploration, stimulus-oriented aesthetics, and abstract aesthetics*.

Task-related competence is the need to have skills on particular tasks, such as cooking, or tennis.

Effect-related competence is the need to have the ability to change the surroundings, and related to a kind of mastery of the surroundings that the agent exists in.

General competence is the need to be able to address your general needs, and may result from a general satisfaction of needs. This is essentially how you feel when things are working out for you, but with the consistent failure to address some need can come the call to feel competent again, urging a person to try another task at which they are better. This could be the reason I have a strong urge to clean my house right now.

Exploration is the need to understand the environment and the objects and processes therein. The more uncertainty there is about an environment, the larger the need to explore it.

Stimulus-oriented aesthetics is the need for stimuli that are in and of themselves pleasant, likely as a side effect of sensory processing of sounds and visual stimuli. Examples include harmonious sounds but also a painting of a landscape that has water, shelter, and sunny weather.

Abstract aesthetics is the need to play with mental representations, and increase their efficiency. This can give rise to mathematical elegance or minimalist music, where structure becomes more apparent.

The needs of a person can give rise to their personality, as needs can differ between people. Some people may have a stronger decay of some need, or a higher reward associated with the satisfaction of some need. We can for example postulate that a high need for exploration may give rise to a personality that is open to experiences, and that high needs for competence and internal legitimacy may give rise to a conscientious personality. High needs for affiliation and romantic affection combined with low needs for competence may give rise to an agreeable character. Modulators can affect this too, as a low decay of affiliation could give rise to introversion, and a high decay of dominance can make for one aggressive human. From my personal experience, I can say that although I really like eating and drinking a lot, maintaining my weight is not very difficult as I don't suffer that much from fasting for extended periods of time. This is anecdotally very different from some of my acquaintances.

It is important for motivated MicroPsi agents that they are connected to a world in which there is ample opportunity to address the range of their needs. Simply put, the world needs food, water, other agents, and so forth. The reasoning behind the motivational learning approach is that an agent has to deal with conflicting demands and has to make choices in the face of changing needs and a changing environment. It may then not be necessary that the full range of human needs are modeled and that the environment simply allows for a multitude of conflicting needs, such that agents can learn to deal with the emergent uncertainty.

Emotions are beyond the scope of this thesis, but they can be understood as configurations that create a focus for certain behaviour with respect to the needs of the person. To give some examples: being sad may function as a focus on a negative aspect that needs to be resolved, because it is negatively impacting some need. Some emotions, like anger, or fear, can shut down long-term thinking and provide a fighting or fleeing mechanism with regards to some stimulus. And love, as stated, directs someone to a particular individual that serves their needs very well.

3.2 Node Nets

Agents are the embodiment of the architecture: they are what the human is to the mind. Node nets are the brains of these agents. A node net is connected to an environment through data targets, which correspond to actions, and data sources, which correspond to sensory information. Consider node nets as acting between perception and action.

3.2.1 Node Types and Link Types

Node nets combine both neural network principles and symbolic operations through a recurrent spreading activation network structure, with directed links. Nodes in the network process their information by summing the activation that enters the nodes through slots, and calculating a function for each of their gates, which send out links to other nodes [11]. A node net is characterised by a set of such node types, and set of states, a starting state (s0) and a network function that advances to the next state by checking the data sources of the current state, and then writing to the data targets to advance to the next state, and so forth. The state of a node net is given by a set of nodes, a set of links, and a set of node spaces, and the current time step t. Node spaces provide structure to the node net, and each node is only a member of one node space.

What defines a node is the number of gates and slots that they contain and what functions and parameters their gates encode. MicroPsi's most basic building block is a concept node:

Nodes have an *id*, which is its name, and a type *nt*, in this case 'concept'. Nodes may also have a set of parameters *params*, which can make a node stateful, because these parameters can be changed during the course of the node's functioning, i.e. changing their state.

Slots are where activation 'come in', and gates are where the activation 'goes out'. These slots and gates give rise to link types. The types of these links are expressed by what gate they come from. These links can be weighted: symbolism or rules may be expressed in the network using discrete links. The following slot or gate types can be distinguished:

- gen: General activation
- por: Is-followed-by / causes

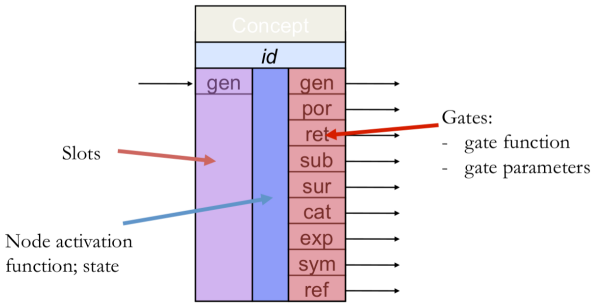


Figure 3.1: Basic building block: a Concept Node.

- ret: Follows-after / is-caused-by
- sub: Has-part / has-attribute
- sur: Is-part-of / attributes
- cat: Class / Is-a, e.g. 'chair' leads to 'furniture'
- exp: Member / Has-subcategory, e.g. 'furniture' points to 'chair'
- sym: Symbol / This-is-my-symbol, e.g. the concept chair leads to the word 'chair'
- ref: Refers-to / This-is-what-I-refer-to, e.g. 'chair' points to a chair.

Node functions typically do no more than pass on activation from slots to gates. But, there are no strict rules on what happens in a node, and a node that performs any function can be defined through native modules, which are nodes that can be implemented in Python by the MicroPsi user. One may for example build a node that fires randomly as it receives input, or a node that only fires once it's activated three times in a row.

Gates have an activation (the result of the node function), and a gate function, the default of which assumes a treshold parameter and sets the activation to zero if it is under this treshold. As such, gate calculations are split up into two different functions, where the gate function can function as a non-linear activation function on the node's activation, such that neural networks may be built out of MicroPsi nodes.

Slots sum up incoming activation. Most nodes simply have a gen slot, but nodes with multiple slots are possible (e.g. script nodes). Slot functions can be defined freely to square, or root, or do anything you would like on incoming activation.

In addition to the concept node, there are some other basic node types which can be used to set up node nets. Firstly, the node net is grounded to the world through its sensors and actuators, which are constantly updated per time step t. Sensors and actuators talk to data sources and data targets, respectively, and will function as terminal nodes of the node nets, where either information comes into the senses or flows out into the world as action.

Another important built-in node type is the neuron. These are the individual elements with which you can build a neural network. Neurons can have an activation function, such as a sigmoid, or simply send activation through to a weighted link. Neurons can be used to 'turn on' a script, by keeping itself turned on as it loops to itself, and then sending information on.

There are more complex nodes called flow modules, which are a special kind of native module. They have a ‘sub‘ slot and gate, but no other configurable slots or gates. Flow modules exchange more complex information via their named inputs and outputs. Connected flow-modules construct a flow. If you create a link to a flowmodule's sub-slot, the flow-graph that leads to this flowmodule will be completely calculated in every nodenet step. I had originally intended to use flow modules to parse visual information, as flow nodes can deal with arrays of information rather than single scalar figures. However, this is currently not feasible because flow modules have no way to 'talk' to 'normal' modules, making it impossible to integrate this information with the scalar actuators and sensors. Future research may look into building this connection.

Lastly, script nodes can be linked to order script nodes in either sub/sur, or por/ret fashion, allowing a script or schema.

3.3 Request Confirmation Networks

By stringing together such script nodes, one can build a 'script', or 'schema'. This can be used to encode for sequences of things, where one action is performed after another, in a sequence, whereby certain conditions may satisfy a certain sub-plan. Doing the dishes, for example, can involve a lot of sub-plans, where each plate is scrubbed until some condition arises, like a washed plate, or finally an empty sink. Such scripts may also be called request confirmation networks, or ReCoNs for short [11]. ReCoNs are a general approach to neuro-symbolic script execution, but arise from MicroPsi's representation style, as the elements of such a network are typically made out of script nodes, sensors, and actuators, through sub/sur and por/ret connections.

Building such scripts may allow for a general top-down and bottom-up spreading activation network. Sensors deliver cues bottom-up to activate higher-level features (like circle, diagonal line, red, nose, etc), while perceptual hypotheses produce top-down predictions of features (the face also has eyes, the foot has toes: inverse rendering), that have to be verified by looking. When using a neural network paradigm, the combination of bottom-up and top-down processing requires recurrency: bottom-up inferences are fed by top-down inferences and vice versa. ReCoNs combine constrained recurrency with the execution of action sequences: they are a neural solution for the implementation of sensorimotor (the body and senses: actuators and sensors) scripts—so basically a node net that has sensors and actuators, and devises a step-by-step to go about moving and sensing based on its hypotheses. ReCoNs work to implement any plan, such as visually scanning an environment as well as executing a sequence of behaviours (which is essentially what you do with your eyes as you scan an environment).

ReCoNs are implemented in MicroPsi but are a general solution rather than necessarily MicroPsi-specific. They can be defined as follows (U = unit/node, E = edge/link):

U = script nodes OR terminal nodes
E = sub, sur, por, ret == child, parent, successor, predecessor

Script nodes in ReCoNs are all the nodes inside the network (neither root nor leaf) which can be:

1. inactive
2. requested
3. active
4. suppressed
5. waiting
6. true
7. confirmed
8. failed

They also have an activation, which can be used to store an additional state. (In MicroPsi, the activation is used to store the results of combining feature activations along the sub links: a summation.)

All script nodes need to have at least one sub link (child), that is either another script link or a terminal node. Terminal nodes are the end, and perform a measurement, which is what sensors and actuators do in MicroPsi—gather information, or make a move. The script functions to put these into hierarchical action plans.

Terminal nodes make a measurement (e.g. 'retrieve luminance'), or write an action (e.g. 'move') and can be:

1. inactive
2. active
3. confirmed

Terminal nodes also have an activation, which represents the value that is obtained by that measurement, if a measurement is made. All of these networks necessarily end in terminal nodes, as script nodes would require another child. Terminal nodes can only be the recipient of a sub link.

ReCoNs ensure that you can have a hierarchical network script without needing a central interpreter that sees what is happening and without any of the nodes in the network having any ‚clue' of what is happening around it: it just gets a message. The units function as (finite) state machines. This simply means that the node can be in a finite number of discrete states, from which it can go to a predefined set of other states (state transitions), based on predefined conditions.

The state-machine units are connected to other units in a way that could look like Figure 3.2. What is shown is an example of a script execution: how such state machines are connected with other state machines. The states of the units are dependent on the information they get through their links.

Initially, all units are inactive. A request is sent to the root node (1). The root node represents a hypothesis that is spelled out in the script. And we want to test that hypothesis (by running the script). After we have executed the script, and thus tested the hypothesis, the root node will either be confirmed or failed. After that, the request signal is turned off and the root

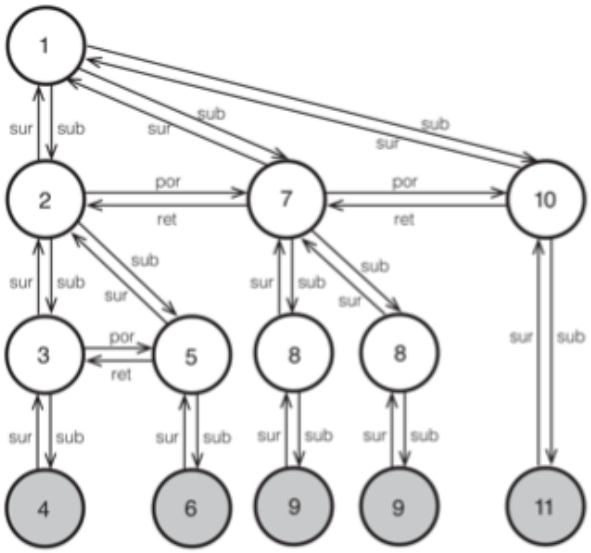


Figure 3.2: An example of script execution.

node returns to inactive. What we request when we request a root node, is the validation of that node.

During this validation process, the root node will have to validate all of its children. It does this by sending a request signal to all of its sub links (children). These children can either be alternatives or sequences: alternatives are validated in parallel, and sequences are validated in succession. Successive execution means that you suppress your successors from becoming active, by sending „inhibit request“ along your por links, which go into the „suppressed“ state: so everything that has a member to the right in the picture (2, 7, 3, 5) will send this to their neighbour, at which point this neighbour will have to wait for the predecessor becoming true. So when 5 gets an inhibition request from 3, it will have to wait until 3 becomes true, and then sends activation further on (to terminal node 6 in this case). If a unit is active, then it tells its parents (sur links) to wait, so the parents go into the waiting state. Note that units acting as features will usually be used by multiple parents: a „wheel“, for example, can be „part-of“ any object with wheels. Parents will then become true once they get a confirmation message from their children, and go into failed when they do not get another wait message from another child. Parents can only get a confirm message from its rightmost child, so 10 in this case for the root node, where 11 has already confirmed 10. The way this is done it by sending an inhibit confirm along the ret links. So 5 says to 3 "I know that you're done but please wait confirming 2 because I have stuff to do and 2 wants me to finish doing my stuff too“. The last unit in the sequence won't get this request because there are no successors so it doesn't have any ret links going into itself (remember all links are directed, and typed by their function). If you move the request from the root node, you can interrupt and reset the script any time you like.

The ReCoN can be used to execute a script with discrete activations, but it can also perform additional operations along the way. These additional operations can be achieved by calculating additional activation values (recall the „additional state“?) during the request and confirmation steps. During the confirmation step, so when a node goes into the „confirmed“

or „true“ state, the activation of that node is calculated based on the activations of all its children, and the weights of the sur links coming from their children. So all of the children have a particular value on a certain point (which is in turn calculated from the values of their children), and these values are multiplied by the sur link values and which then determines the activation of the parent node. During the requesting step, children can receive parameters from their parents, which are calculated using the parent activation and the weights of the sub links from these parents.

This paper uses a ReCoN to implement a script of actions whereby actions are performed until some condition is met, which will be explained in detail later.

3.4 The MicroPsi Framework

MicroPsi is written in Python3 [1], with minimal dependencies to keep the program light [6]. The GUI is found in the browser, and MicroPsi can thus be used as a web application. MicroPsi consists of a server, a runtime component, a set of node nets, a set of simulation worlds, a user manager, and a configuration manager. On starting the web application, the server invokes the runtime component, which communicates with the server using the API. The runtime works independently of the server. It is possible to manage MicroPsi's node nets from this runtime component, and it is possible to manage simulated worlds from within this as well, though the current paper uses a separate Minecraft server. From within the web application, environments can be created, and agents can be put within environments, and the agents' node nets can be created, altered and saved using the runtime component. The runtime can be started, forwarded and reset using some playtime operators that interact with the simulated world.

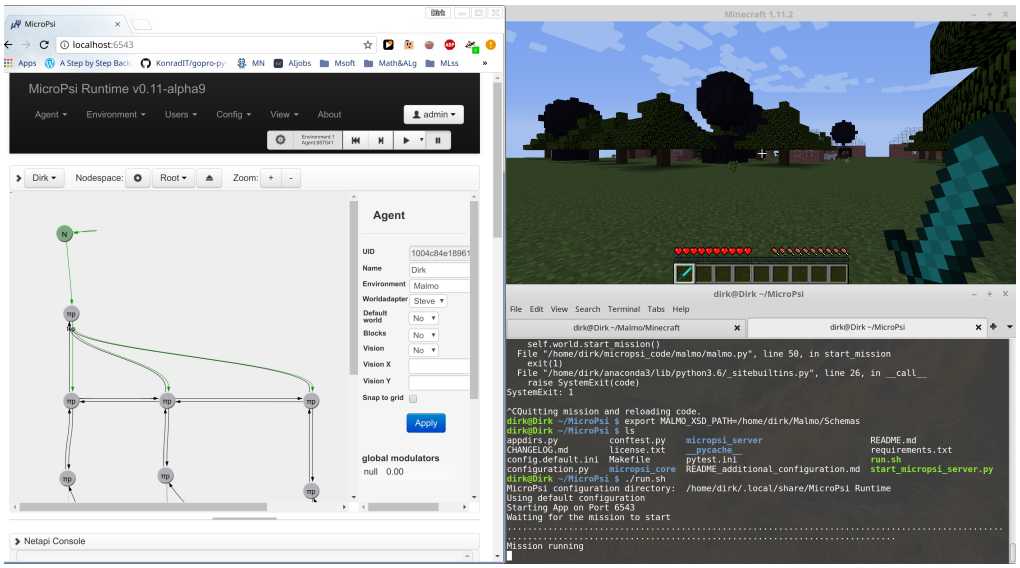


Figure 3.3: A visual overview of running MicroPsi and the Malmo world side by side.

[1] https://github.com/joschabach/micropsi2

4 Virtual Environments

4.1 Embodied Cognition

General intelligence can solve a large array of problems without explicit programming or restructuring, whereas 'narrow' AI focuses by definition on a more constraint effort to beat well-defined tasks or limited set of tasks, such as Go or Chess, practically allowing it to discard anything task-irrelevant, and thus requiring explicit programming or restructuring for novel tasks. To achieve a more human-like general intelligence, an agent has to be exposed to a large range of tasks, whereby it will experience a multitude of evolving and conflicting demands, which is, after all, the reason general intelligence evolved in the first place. Discarding anything task-irrelevant compromises then the ability to beat other challenges. In other words, general intelligence exists for and in relationship with the range of problems it can solve.

It's very hard to come up with a range of symbols that represents and refers to the real world. This problem is named the *symbol grounding problem* [27]. This problem became clear to many of the early AI experimenters, who attempted to isolate to small domains that could be described using logic, math, or simple finite sets of symbols. They promptly found out that this does not generalise to the real world, where a large set of problems and challenges ranging from language to abstract reasoning can not be easily captured by isolated programs with a small set of abstracted symbols without a clear real-world referent. To deal with the richness of the real world, AI systems are likely going to need symbol systems that can express that richness in full, e.g. through mental imagery, and refer to the world in some way, such that the symbols can be meaningful.

A most strongest interpretation of the symbol grounding problem is the position of embodied cognition: only a (human) body in a physical world can bring about cognition and thereby intelligent behaviour. Mere symbol manipulation can not be enough, and we need to be in real reality. This, on the surface, makes some sense: human beings are not brains isolated from bodies and its surroundings, so let's not isolate bodies and surroundings from our study of the mind. General intelligence evolved to address the needs of a body in a complex and ever-changing environment, in order for it to survive and replicate successfully; the brain does not operate in abstraction from the world but is assisted in its function by a physical body in a physical world. As such, symbols can acquire their meaning through physical interaction with the real world.

Concrete arguments for embodied cognition are plentiful [70]. Cognition is situated in a world with many ongoing processes, and under time-pressure (I can't keep writing this thesis for very much longer..). We off-load work to the environment through our phones, calendars, and by counting on our fingers. As such, there is no studying the human mind without considering the environment in which it evolved. It simply wouldn't make sense. Cognition is there for action: humans and animals seek out the use of the things in the world, and aren't designed to sit around and do mathematics in their heads without interaction with the outside. And a lot of human cognition is body-based. Mental imagery is there to imagine

that which is not in front, short term memory is there to store that which is present in time, episodic memory remembers events that took place in the world, and implicit memory, or muscle memory, is there to remember what physical body movements lead to what result, in the world.

To be true to real embodiment in artificial intelligence research, however, one must create actual bodies in actual worlds for anything short of an actual body is going to leave questions open about the integrity of the embodiment. We are left then to figure out the field of robotics alongside artificial cognition; the first can not come without the second and the second, per strict interpretation of embodiment, can certainly not come without the first. (And will this be enough? Would the field of AI not merge with parenthood, i.e. drop the A leave the I.)

Luckily for the eager AGI experimenter there is no good evidence that we should take the long and winding road through bodybuilding to artificial intelligence. Embodied cognition may be crucial to understand particularly human intelligence in context. But for human-like (general) intelligence it is probably unnecessary, so long as the system is exposed to the richness of presentation that the world offers, and the symbols in the system can indeed represent that world and its richness in full. Practically bodiless, the late Stephen Hawking did well from an intelligence point of view, because for all intents and purposes, the world projects upon a mind a recurrent set of patterns in which regularity can be found and manipulated [9]. The symbols may refer to these patterns, and these patterns can be generated by any pattern generator, such as a virtual world [9].

If not for bodybuilding, how else may we address the aforementioned views on embodiment? A milder interpretation of the symbol grounding problem can be found in situated cognition. Situated cognition does not put forth a physical body as a requirement but does view intelligence to be indistinguishable from action: situated cognition and situated perception emphasize the influence of the environment on cognitive processes [65]. Hawking did of course spend time in the world. A world that provided him with incoming information. Each of the arguments in favour of embodiment could seemingly be addressed in a virtual world. I'm sure we could count on virtual fingers (and practically speaking AI doesn't need to do that anyway), and more generally come up with a virtual setting wherein an agent is confronted with a large variety of tasks that require planning and deliberation. Full embodiment is a physical step too far, and we may address the grounding problem through virtual agents in virtual environments. Once the roboticists advance far enough we may always turn back to the issue.

A virtual environment is currently the most feasible and comprehensive way for an agent to be exposed to a large variety of tasks, patterns and situations. Without an environment an agent would be left to study pure mathematics by itself (in the dark). For this reason, MicroPsi agents live in an open environment that they explore and learn to navigate [10].

4.2 World Requirements

Even though we do not need embodied agents in the strictest sense, having something that comes close to it, at least within feasibility requirements, isn't going to hurt. The same is true for any virtual world in which situated agents are tested. We may not need a physical planet but we do want the relevant characteristics of that planet that helped bring about intelligence: as far as we know, general intelligence only evolved in humans on planet earth (animal cognition may be someone else's thesis).

The point of the environment is not to provide a way to make the best AI for any specific task (narrow-AI), but to expose the agent to the breadth of tasks and challenges for which general intelligence was necessary in humans. Considering the difficulty of simulating the entire planet, a worthwhile question to ask at this point is what aspects of the real world we wish to keep and what aspects we can confidently discard. Laird and Wray [39], slightly modified by Adams et al. [2] (Bach being one of the authors), came up with eight aspects of a virtual environment that are necessary and possibly even sufficient for AGI. I will address each one shortly, in my own words. I will not dispute these characteristics, but assume them as guidelines. I will though, in the conext of this thesis, add a characteristic of my own.
C1. The environment is complex, with diverse, interacting and richly structured objects. The objects in the world can not be easily classified and come in multitudes of shapes and forms. Objects need to have an internal structure that requires complex, flexible representations.
C2. The environment is dynamic and open. The environment has to have the ability to change the agent as well as be changed by the agent. Openness is added on because otherwise an agent could rely on a fixed library of objects, relations and events.
C3. Task-relevant regularities exist at multiple time scales. Despite its variety and complexity, the environment is learnable and has coherent physics. Otherwise no sense could be made of the world.
C4. Other agents impact performance. Intelligence evolved in relation to other intelligences. Like the real world, other agents provide opportunity to conflict and cooperate.
C5. Tasks can be complex, diverse and novel. To avoid narrow AI, tasks have to be both complex and changing.
C6. Interactions between agent, environment and tasks are complex and limited. Although the agent's interaction with the environment encompasses many options, these options are limited by learnable limitations, which in part are observable by the agent. The agent therefore needs to have plentiful ways of sensory input and actions available for interaction with the environment.
C7. Computational resources of the agent are limited. Agents do not have godlike capacities and like humans have to learn what they can and can't do.
C8. Agent existence is long-term and continual. The agent does not get a break from its environment and needs to address its goals on a long-term basis, and will have to think towards these long-term goals.

Given the context of this thesis, I'd like to add one of my own to the abovementioned criteria: *C9: We wish there to be a way to for each distinct urge type in MicroPsi to have an analogue to its real world intention.* E.g., there needs to be some kind of 'food', as well as other agents.

An option is to use video games as worlds for AI experimentation. Video games can provide a way for AI to learn various approaches to differing tasks. Video games often require a varied strategy and it is relatively easy to tell the agent when it's doing well, or when it's doing badly, as this is often directly expressed in the score or a sore game over. An example of this is the Arcade Learning Environment [13], which is a platform for Atari 2600 games, such as Breakout, wherein Google Deepmind managed to build an AI that played extremely well and even came up with its own unique strategies [48]. Another example is Torchcraft [67], a Facebook platform for AI experimentation in Starcraft, a strategy game. Also examples are the successes of AI in Go and Chess.

To date, research in these games has not given rise to any notion of general intelligence. Each new game is tackled using a new system. If the systems that beat these games are indeed general, then the same algorithms could be used on new games. This is often not true, because new systems are constantly built, even though the 'old' system got super-human performance on another game. In the cases where a single algorithm manages high performance on multiple games, such as with Go and Chess, it is because these games are very similar in the relevant ways, such as that all of the information is visible (in contrast with e.g. poker), steps are discrete, intermittent scores are easily determined, and of course: the symbols are scarce and have clear 'meaning'. (Effectively avoiding the grounding problem.)

A way to combat this is by building AI systems that can beat more games. An organised attempt is through the General Video Game AI Competition [57], where participants are invited to come up with a single algorithm that would then go on to beat multiple games. Initiatives like this may be hard to synchronise on a larger scale, because companies and scientists are looking for lucrative or prestigious results and something that does OK on a bunch of things might not be as interesting as something that does incredibly well on something. Another criticism is that there is unlikely to be a general purpose algorithm that beats every problem in humans, so setting up a quest to tackle challenges with a single system may miss the point by overshooting the mark and not discriminating correctly.

This is not a fail for games as a category, which is so extensive that Wittgenstein famously failed to define it [72]. A better option would be to use so called sand-box games, called so because they are interactive open worlds as sand-boxes are to children. An option here is to use the Open AI universe software platform [17], which allows experimentation in any software by allowing the AI to take charge of the keyboard and therefore play any game. Grand Theft Auto becomes an option as well as the aforementioned Minecraft. The downside still is that both come without good tools to set up an experimental environment.

MicroPsi agents have been embedded in virtual worlds from the outset, and have recently also found application in real world robotics. As far as the richness of these worlds goes, the options have been lacking. An autonomous Braitenberg vehicle [8] is instantiated that reacts to light, and MicroPsi currently works on robot arms for manufacturing, but any real-world analogues that match the 9 characteristics have been absent thus far. With notable exception of an interface with Minecraft through the Spock API [35]. Minecraft is an excellent world for AGI experimentation, which as a meta-game scores perfectly on the characteristics laid out by Laird and Wray [39]. The Spock API however does not allow for full experimental control. For this reason I connected MicroPsi to the Malmo API, which is built on top of Minecraft, and which does give full experimental control.

4.3 Malmo

Malmo is a platform for AI experimentation built on top of Minecraft. It is an API that is designed to easily set up tasks, agents and experiments [31]. Minecraft is an excellent game for AGI experimentation because it is very much a meta-game. This means that you can set up challenges within the game, much as the real world has challenges within it. Minecraft is a large open world made out of blocks or voxels wherein you can play, build structures, survive, explore and discover. It looks somewhat trite on first glance, but on further inspection, the world is clearly rich, versatile, and very much like the real world in many ways. I'll test it against the the AGI world characteristics, previously done by Johnson et al. [31] but in my

own words.
C1: Minecraft has many items and block-types that can be combined together in ever more complex objects. On the small scale, though, the richness leaves wanting, but texture packs are available [49].
C2: Malmo allows for infinite possibilities for missions and environments.
C3: Tasks are like real world tasks, varied, time-limited, and involving planning, navigation and so forth.
C4: There is support for multi-AI as well as human-played agents interacting with AI.
C5: It is easy to create tasks in Malmo so this is up to the experimenter.
C6: Some abstraction levels allow the user to make ever more complex perception/action feedback loops.
C7: The world naturally constrains and more constrains may be added.
C8: Agents can live forever in the Minecraft world and survive within it.
C9: The world allows for expression of a large subset of needs. To review: Physiological needs may be addressed in part. Food is present in the Minecraft world and there is a hunger bar. The agent can be damaged and may need time to heal. Rest may be introduced by introducing timely negative rewards to movement. There is no danger of hypothermia of hyperthermia, as there is no temperature in Minecraft. Nonetheless, there still a fair bit of physical needs that can find expression in Malmo. Social needs are relating to other agents, which can be part of the Malmo world. Affiliation can be expressed in various ways, by measuring time interacted with agents, for example. Legitimacy signals may be expressed through the chat, or through other actions. (Other than facial expressions.) Nurturing may be addressed by helping other agents reach rewards, which may introduce a kind of friendship that needs even not be encoded through affiliation, and could be a result of reinforcement learning whereby other agents are associated with reaching goals. Which is, if we are honest, a large part of friendship in humans. Romantic affection may be difficult, as there is no such thing as sex in Minecraft, but this may be in the future be addressed by mods that allow agents to have children, whose quality could be based on some mate value. Dominance can be expressed through agent interactions whereby one agent limits the freedom of the other agent with some (dis-)incentive structure. Cognitive needs can each be expressed in Minecraft, for competence may be the ability of an agent to reach its goals, change the environment, or finish certain subtasks of varying difficulty. Exploration is a given, and agents may come up with some cognitive map of the environment. Stimulus oriented aesthetics are more difficult, as there are no musical compositions or tactile sensations in Minecraft, but those Minecraft landscapes that do so reliably satisfy an agent's goals may over time receive some aesthetic appreciation. Abstract aesthetics may be satisfied by any world of sufficient complexity and per C1, Minecraft qualifies excellently in this regard, though with some reservation on the rich internal structure of Minecraft objects, which are, admittedly, a bit blocky. But to address this, texture packs are already available [49], which could potentially be integrated with Malmo.

Malmo, though young, already has a stellar track record of working as an experimental ground for A(G)I research. It has been for example used to automatically extract data from Minecraft [50], learn how children try to teach agents [3], and to integrate data from different sources in one agent [22]. To conclude, Malmo is a great platform to AI experimentation and a good match for MicroPsi.

5 Connecting Malmo to MicroPsi

Malmo works as a layer of abstraction on top of Minecraft [31]. The high-level components of Malmo are agents that interact with an environment through a continuous loop of perception and action. Malmo sets up a Minecraft host server in which experiments with agents can be run whereby the user supplies the code that works as the 'brain' of the agent. Even multiple agents and humans can interact in this Minecraft environment at once. AI researchers using the Malmo platform are provided with the following concepts, which are set up as objects through the API:

The *MissionSpec* allows the user to set up a mission for the user. This is what distinguishes Malmo from the Spock API. The researcher can set up any environmental world, and decide what is located within that world, and whether there are rewards attached to this (for reinforcement learning), as well as equip the agent with the necessary items. The MissionSpec takes an XML file for its specification, and the MissionSpec can be further altered for a specific agent through the API as well.

The *AgentHost* instantiates a mission, and sets up an agent within that world. The AgentHost then starts the mission. During the mission, commands can be sent through *SendCommand* to the AgentHost, such as "jump 1", or "walk 0.5". The AgentHost can also request a *WorldState*. The WorldState can provide an agent with game information, such as its coordinates, or its life. The AgentHost may also set a videopolicy such that it can receive frames and even video from the WorldState such that the agent can only receive sensory information rather than direct information from the environment. Finally, there is a *HumanActionComponent*, which allows for human-AI interaction.

MicroPsi agents are situated in a virtual world and are instantiated as node nets [8]. For practical purposes, Malmo works as a buffer between the user's code and the Minecraft world. The user's code shall in this case be supplied by MicroPsi's node nets. The node nets talk to their environments through sensors and actuators (data sources, and data targets). Data sources feed into sensor nodes, and data targets feed into actuator nodes. That is, MicroPsi takes in perceptual information from a world through its sensors, and writes activation to its data targets, to effect an action within that world. This works as shown in Figure 5.1.

In the MicroPsi code, a World class is set up to specify an environment and a World Adapter class is set up to feed the sensors and actuators by linking them to a world.

To connect Malmo to MicroPsi, the following was done: In the initialisation of the Malmo World class, a Malmo AgentHost is set up. Then a mission is initialised using the MissionSpec, and a *MissionRecordSpec* is set up here as well. The World class contains two more methods, one that attempts to start a mission through the AgentHost and another that attempts to receive a WorldState from this mission that would signify that the mission has started. When one creates a new environment within the MicroPsi editor, this world class is set up, and the necessary objects for Malmo experimentation are set up, including the mission file.

The World Adapter class takes care of certain options that can specify things such as whether we wish to receive video data or not, and in its initialisation method the data targets are fed

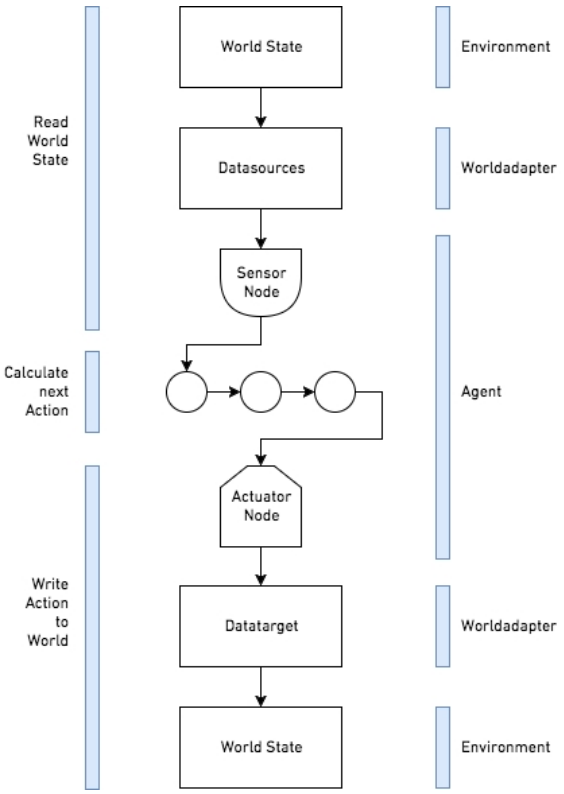


Figure 5.1: From perception to action.

with possible moves that can be taken through the AgentHost in Malmo, such as “move”, “strafe”, and “crouch”. The data sources are fed with information that Malmo may receive from its WorldState, such as “TotalTime”, measuring the total time passed, and “Xpos”, denoting the X-coordinate. The main function that is looped in a MicroPsi agent is the *update data sources and targets* method that is specified in the World adapter class. In my code this function first attempts to start a mission, when no mission is ongoing, by calling these methods from the Malmo World class. If there is a mission running, this method receives a WorldState from the AgentHost. Then the actuators are updated, by taking the information fed to actuator nodes and sending the activation on to the AgentHost in Malmo. Then sensors are updated by taking the information from the WorldState and feeding to data sources. While MicroPsi is able to work with nodes that process tensors, the interaction between these nodse and scalar figure types of nodes is still work in progress in MicroPsi, as I’ve been told through private communication. For this reason, the information from the vision classifications are fed to single scalar figure data sources. We now have a working interaction between MicroPsi’s node net agents and the AGI experimentation platform Malmo [1]. This lays a foundation for AGI research on motivated agents with MicroPsi.

[1] https://github.com/Brigadirk/MalmoPsi

6 Computer vision

Being able to see is arguably the first step for agents to interact with any world—real, or virtual. Of course, many virtual worlds can provide ground truths about the entire environment, but that would be missing the point for AI research, because real world problems do not provide such luxuries. Even though humans or other animals may use sound and smell in rather impressive ways to navigate the planet [36], neither of these options are currently very feasible when it comes to virtual worlds, especially not Minecraft. If then we are to create an intelligent agent and test its abilities within Minecraft, our best bet is to begin implementing vision within this agent, such that it can 'see' what it is doing. 'See' within quotation marks because this process should not be interpreted as a venture into how human vision works: the point here is to bootstrap the agent to maneuver within the virtual world, such that a variety of experiments can be run, either to improve that vision or to use the current visual capacities for other experiments, such as planning routes—experiments which can, might, give a view into human cognition. If the goal is to have computers process their surroundings on a high level then it is not clear that we have to build something that is biologically plausible so long as we hit our goals. Of course this may be put forward as a defense for all of narrow AI, but computer vision does yield rather impressive results, such as actually outperforming humans on ImageNet, getting a 3.6 per cent error rate [28], where humans get a 5-10 per cent error rate (some were so crazy to try) [33]. So, notwithstanding the limitations of deep learning outlined in the introduction, deep learning, and especially its convolutional variant, is clearly an excellent tool for visual processing. For this reason, the current study ventures to apply a convolutional neural network to process vision within Malmo. In fact, while some of the limitations of neural networks are addressed in recent work by Geoffrey Hinton on *Capsule Networks* [61], his solution still makes important use of convolutions, adding to the credibility of this approach to computer vision—even when the computer vision techniques are being revised to address limitations.

Computer vision tasks can be categorised into a few subsets: localisation, detection, segmentation and classification. Localisation attempts to find where in an image an object is located. Detection is localisation for multiple objects. Segmentation attempts to find the exact outline of an object in the environment. And classification is what ImageNet contest are about, and deals with classification of objects into a finite set of objects. Because classification is currently the most straightforward to achieve, the present study focuses on this as a way to bootstrap vision in MicroPsi agents, though other techniques can later be added to this pipeline to enhance the agent's visual capacities. The current paper bootstraps vision by using a MobileNet convolutional neural network, which is optimised for efficiency. Before I explain how this works, I have to explain how convolutional neural networks work. General information about convolutional neural networks has been gathered from [56], and [41].

6.1 Convolutional Neural Networks

Convolution and convolutional neural networks are arguably the most important concepts in deep learning at the moment. They can be most succinctly characterised by the creation of feature maps: where are what features in an image, and how do they combine to form objects? The neural network used by Krizhevsky and friends in 2012 [37], that revived the field of deep learning, made use of a deep convolutional architecture. It won the yearly ImageNet competition, dropping the error rate from 26 per cent to 15 per cent, a major achievement at the time. Convolutional neural networks are widely used for a variety of applications, but are most notably good at processing images—as is clear from the fact that it outperforms humans on this feat, at least on the ImageNet test.

Convolutional neural networks take their inspiration from biology—though inspiration again being the operative word. Hubel and Wiesel [30] showed in an experiment that specialised neurons fired only in response to edges of a certain orientation in an image or video. Some neurons only respond to diagonally and some only to horizontally directed lines. Hubel and Wiesel postulated that these neurons were organised in a columnar architecture which, meshed together, would form the brain's visual perception. The gist of their findings and the basis of the convolutional inspiration is that the brain carries specialised feature detectors that are organised structurally to be used in visual perception.

Convolution is a mathematical operation that serves to mix information by taking two sources and applying rules with which the mix is effected. We can apply this to images in two dimensions, the width and the height. The first source of information that we have is the pixel information of the image. Images are typically stored as width x height x 3, where the 3 stands for the RGB values, or colour channels. Each pixel consists of a value between 0 and 255 for every colour channel (red, green and blue). This means that for each image we get three matrices of size width * height, one for each colour channel. Channels are also referred to as dimensions.

The second source of information is a convolutional kernel, or filter, which is a single matrix of numbers (also known as the weights) of a certain predefined width and height. The numbers in the kernel are organised in a way that forms a recipe that is applied to the input image. Through convolution we can then go on to mix the information of the input image with the information in the kernel. You do this by applying the kernel, which is for example 3x3, to a similar sized spot on the image, so a spot of 3x3 pixels in this case, and then performing element wise multiplication with that part of the image and the convolution kernel. You then take the sum and get 1 datum in the feature map. After this you slide, or rather convolve the kernel over the complete image, by beginning, say, one to the right, and repeat the process until you get a full feature map, going one down and repeating the process one step down.

The stride is the step size with which you convolve over the image, meaning that you could for example set the stride to two where it would take two steps to the right and down. This means that there will be less overlap in the feature map. This could keep the input to the next layer smaller, which reduces computational costs.

Padding adds a few pixels (generally put to 0) at the outsides of the image, such that the corner pixels get featured as much as any of the other pixels. Take for example the upper left pixel. At any stride, and without padding, this pixel will only be part of one computation, in contrast to many other of the pixels. To give them even weight, you add the stride based on

the size of the filter such that it has a larger vote. This can be important if there is important information in the outsides of the image and considered good practice.

The feature map basically lays out where a feature is located within an image. This is further illustrated by Figure 6.1 and Figure 6.2.

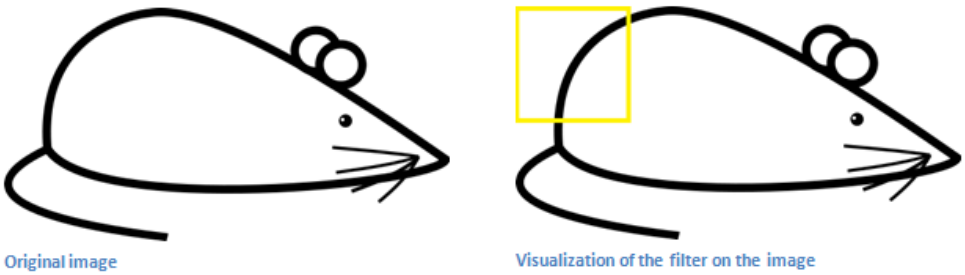


Figure 6.1: Edge detection through convolutional filters.

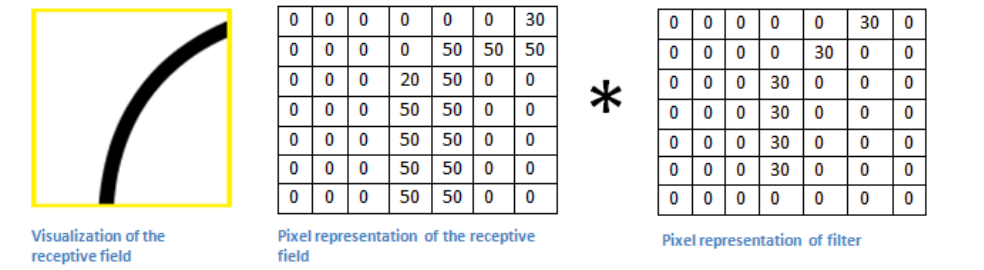


Figure 6.2: An edge is detected by the filter.

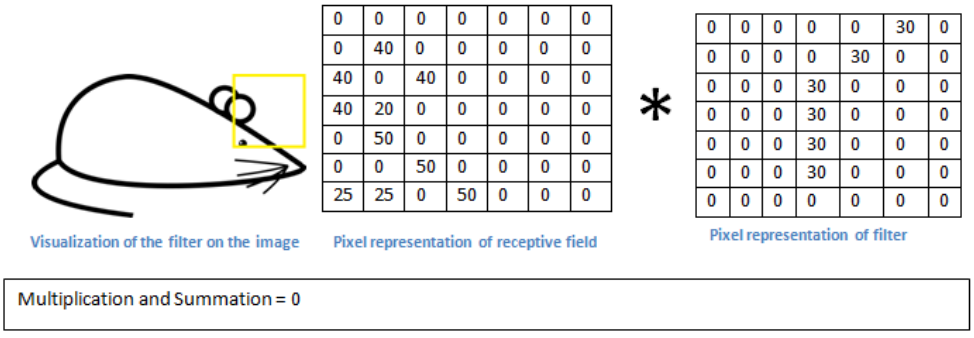


Figure 6.3: This particular filter did not detect anything at this point.

Consider the segment of the drawing of the mouse in Figure 6.1. Imagine that sliding a kernel over this part of the image at that moment. The filter "matches" very well with the

input, and it will output a large number on its feature map for this location. This means that the CNN has detected that feature at that point in the picture. This is in contrast to other places, as can be seen in Figure 6.3. The curve was not detected in this part of the image, and thus the output was 0—or generally low.

We can now understand the convolutional kernel as a feature detector similar to an edge orientation oriented neuron. Similar to how the brain does it—do take such claims with reservation—you can organise these kernels structurally in a neural network to integrate many feature detectors built upon more feature detectors upon even more feature detectors so to recognise more complex features and in the end full objects or scenes. The beauty of machine learning is that the feature detectors need not be pre-defined, and the numbers within the kernel can be learned through a technique called backpropagation. In a typical pass through a CNN an image goes through a series of convolutional, non-linear, pooling and fully connected layers. A classical layout would look something like:

1. Convolutional layer
2. ReLU activation function
3. Another Convolutional layer
4. ReLU activation function
5. Pooling layer
6. ReLU activation function
7. Convolutional layer
8. ReLU activation function
9. Pooling layer
10. Fully connected layer

For the first layer, we could take as our nodes, for example, three 5x5x3 filters, each taking 5 by 5 pixels at the time for each colour channel, and each looking for different features. The dimensions of the filter have to match the dimensions of the input, so an RGB image will have filters with a dimension of three. The output of such a layer will have as many dimensions as there are filters in the layer. A 3x3x3 filter on a 6x6x3 image will end up as a 4x4x1 feature-map. The output of such a layer, to which an extra bias weight is added, goes through a ReLU activation function. The ReLU activation function introduces nonlinearity to a network that thus far has only computed linear operations in the convolutional layers (multiplication and summation). Historically, sigmoid or tanh functions were used, but researchers found out that the ReLU works a lot better because it's computationally efficient without performing any worse than other options [52]. What a ReLU basically does is change all negative activation to 0. The input to the second layer are then the low-level feature maps that the first convolutional layer took from the picture, allowing the second layer to pick up on higher order features convolving over this output. These features will typically be circles, squares, half-circles, and half-squares and all that can be likened to it.

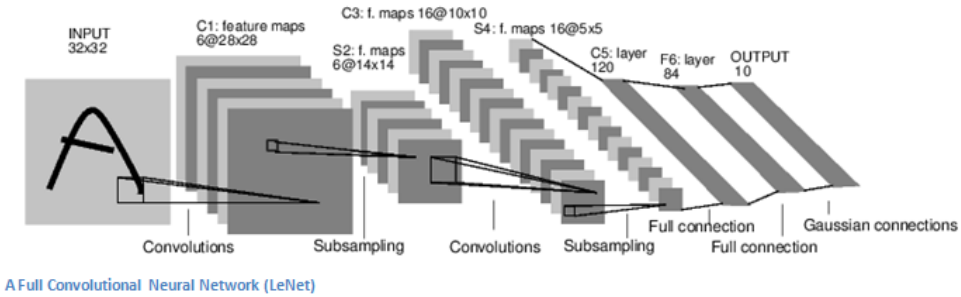


Figure 6.4: A typical convolutional neural network.

Pooling layers are there to reduce the size of the output by for example only taking the largest number in each 'block', which is a segment of the input. Once we know that a particular feature is present in the input, its exact location is not as important as its relative position with regards to other features. This pooling allows us to reduce the size of the output and save on computing costs.

At the end of the convolutional neural network is a fully connected layer, a la 'normal' neural networks. The image size, or feature map, has by then gotten so small, that the contents are squeezed into a one-dimensional vector, which is fed into a fully connected layer. This fully connected layer outputs an N dimensional vector with N being the amount of discrete, mutually exclusive alternatives (classes) the network has to choose from. It then proceeds to output probabilities for each of these classes, adding up to 1. This is called a softmax function. This means that the fully connected layer is there to see which category correlates most strongly with the high level features that the network presents it. An overview of a typical architecture can be found in Figure 6.4.

Another tool to use in convolutional neural networks is 1x1 convolutions, which are sometimes also referred to as 'network in network' [44]. These function not to reduce the width and height, but the dimensionality of its input, and are often used to compute reductions before other more expensive convolutions. The way this works is that if you use a 1x1 filter of, say, the number 1, you just recreate the input with every feature kept the same. But it can do this over all dimensions of the input, collapsing these into 1, resulting in an output that is exactly as many dimensions deep as there are filters that you use. So, if you have an input of, say, 28x28x192, on which you apply 16 1x1x192 filters, then this results in a 28x28x16 output. You could also reduce the channels by using a similar amount of 3x3 or 5x5 filters, but these are much more expensive, so using 1x1 filters do this before you apply the larger filters, can greatly reduce the computational cost and allows creation of deeper networks without too much expense. By applying an activation function to a 1x1 kernel's output, you can also introduce more nonlinearity in the network, allowing it to learn more complex functions, without too much extra expense.

A CNN forward propagates to produce some output, and we test the quality of that output by comparing it to the ground truth on a labeled dataset. The way this works is by first taking the loss function, such as a mean squared error, which measures the distance between the network's output and the ground truth. This cost function can be seen graphically as a landscape full of hills, where the cost function can be at any spot within that landscape. What we wish to do, is reach the lowest point in that landscape, where the distance between

the output of the network and the ground truth is the smallest. This can be done through backpropagation, which performs a backward pass through the network whereby weights on the connections between the nodes are changed according to their respective bearing on the output value. These weights of the network are, in our case, the numbers in our filters. This means that this algorithm actually constructs feature detectors automatically. This agnosticism is useful because it doesn't require the programmer to build in any innate structure or feature detectors. Backpropagation finds this minimum using gradient descent. Gradient descent means that we can move down a hill in the landscape in incremental steps. The size of the steps we take is expressed in the learning rate, which if set too small, the network converges too slowly, but if set too large, the process may overshoot its target and can't get into the valley, as it keeps stepping over its lowest point back and forth.

We start off by setting random weights for the network, for example through Xavier initialization [23]. After taking a forward pass on all of our training examples, we compare the output of the network to the labels, and sum the loss, to get the loss of the network. We want to compute the contributions to the error of each weight in the network. A neural network is essentially a large composite function where some function, some layer, multiplies a weight matrix using the activation of the previous function, the previous layer. Because of this, we can use the chain rule to compute gradients for the whole network. The chain rule works by taking the derivative of the outside function, leaving the inner function alone, and then multiplying by the derivative of that inner function. You do this for each layer in the network. So to perform backpropagation, we can iteratively apply the chain rule to calculate error derivatives for every weight and bias in the network and update each weight and b in the opposite direction of the gradient. We update the weights slightly (somewhat depending on our learning rate) and then do the complete forward pass through every training example again. This can be very slow, as we have to go to the entire training set each time. A solution to this is to use Stochastic Gradient Descent [16] where only one training example is taken at a time, and the weights adjusted according to the error on that training example. This works just as well as batch gradient descent, which takes batches of multiple examples, and even full gradient descent, on most occasions. But it is much quicker.

The goal is to reach a minimal loss, after which the network has learned a model of the input data. One problem is that the model could have overfit the training data, meaning that it has modeled the noise present in the data. A way to combat that is to use regularisation, which puts a penalty added to the loss on large weights such that the network is incentivised to keep its weights small and thus is less able to fit outlandish distributions. Another way to combat overfit is through dropout, whereby some neurons in the network are sometimes turned off during training such that the representation gets spread over the entire network [64]. Pooling layers also serve to counter overfitting, because we do away with some precise information making our model more general. Lastly, we might prevent overfitting by gathering more data, which is an important requirement for many applications of neural networks.

6.2 Data requirements for convolutional neural networks

CNNs require a lot of training data. This is problematic for our current purposes because we have little sample data to deal with. A way to deal with this problem is through data augmen-

tation, whereby images are mirrored, tilted or otherwise changed slightly in order to create variation in the dataset which can result in a more generally applicable model. This technique only gets you so far, though, because if you only have a few dozen examples you might be able to scale it up 4x, but that still results in a relatively small dataset. Another technique to circumvent needing a lot of input for a given domain, is by using transfer learning.

6.3 Transfer learning

The power of general intelligence is that it can solve a large number of tasks, instead of having to be re-purposed. If not for this quality, then every time humans encountered a new task, they would have to learn this from scratch. This is not what happens. Evolution doesn't go back in time to rebuild the system to allow for a new feature. It uses what is already there. As one learns, intelligence crystallises and learned information and skills can be carried over to novel tasks and applications, allowing for much faster learning, as only task-specific skills and information have to be learned. This is the idea behind transfer learning. It would save a lot of time and energy if we could use already trained models and slightly alter these models to fit the task at hand.

There is a good reason why transfer learning can work in image recognition with neural networks. The first layers of neural networks tend to encode low-level features such as edges and curves. These edges are then in subsequent layers combined to form more complex structures, such as circles, squares, or half-cireles, or half-squares, and so forth. Unless you have a problem set that contains objects that look much unlike 'normal' objects, then your network is going to need to learn how to detect edges and curves all the same. This learning takes a lot of time and computational power, but once the weights are learned, a single forward pass does not require much. As such, we can use pre-learned weights of neural networks that are pre-trained on a different dataset, and only train the last, or some of the later, layers on our specific dataset [73] This last layer is often called the bottleneck. Only training the bottleneck of a network requires a lot less data, and even a few examples may be enough, which makes it suitable for present purposes.

CNNs were popularised by Kizhevsky [37] with the 2012 ImageNet victory. ImageNet is a depository of 14 million images for more than 1000 categories (though still mostly dogs), with clear and nicely curated examples of all. There are various models available as a result of various ImageNet competitions, the weights of which are available for download, such that they can be applied to novel approaches. It would be interesting to test whether or not networks trained on real world images generalise to the Minecraft world. On the one hand one might say that if children, who are also trained on the real world, are able to recognise trees and other objects within Minecraft, then so could a network. A tree within Minecraft is nothing more than a few edges and curves all the same. However, it could also be conjectured that the understanding that the Minecraft world is made out of blocks, may alter the prediction and extend the tree category, for example, to include blocklike structures. Based on what we know about the workings of CNNs, that is that they work by building up larger structures from basic features, I hypothesized that this should be able to generalise perfectly well to a Minecraft voxel-type world. There are a few examples of pre-trained weights on ImageNet competitions to apply transfer learning on a smaller dataset. These networks can be easily downloaded from the web. For present purposes I chose to use Google's MobileNet archi-

tecture, because it is optimised for efficiency which makes it suitable to use in real-time in the Minecraft world.

6.4 MobileNet

MobileNets are a family of mobile-first models for vision built in TensorFlow [29]. The unique selling point of these MobileNets is that they are mindful of restricted resources while still powerful and accurate for most applications. This makes using a MobileNet very practical for current purposes, as a visual agent has to process each frame one by one. Moreover, it would be nice to be able to experiment with visual agents on any device, including my laptop without a proper GPU, as most of MicroPsi's implementation does not require a GPU and is designed to run on a CPU.

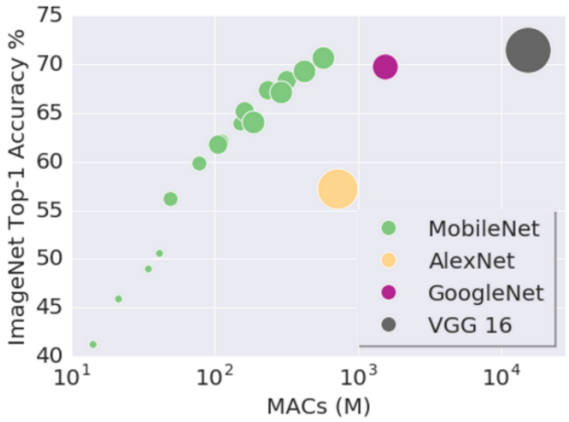


Figure 6.5: MobileNet performance.

The Y-axis of Figure 7.2 shows the ImageNet Top-1 Accuracy, which measures how many pictures the network classified correctly by having the actual class of the image as its highest probability output. The X-axis measures the complexity of the algorithm using Multiply-Accumulates (MACs), which measures the number of fused multiplication and addition operations, which is a good measure of the computational requirements of the network. As can be seen from this graph, MobileNets score very well versus larger networks, despite a significantly smaller size. Google's flagship model, Inception V3 [68], has a Top-1 accuracy of 78 per cent on ImageNet, but the model is 85MB to download, and requires significantly more processing power than the MobileNet in even the larges size, which gives 70.5 per cent accuracy and counts a mere 19MB for download [18].

MobileNets are based on a streamlined architecture that uses depth-wise separable convolutions to build light-weight deep neural networks. Normal convolution filters combine the values of all the input dimensions into one dimension. So an input of 3 channels becomes 1 channel, and an input of a 1000 channels becomes 1 channel all the same. MobileNets also make use of such a construction, but only in their first layer. Other layers use depthwise separable convolutions, which are a combination of sequential depthwise and pointwise convolutions. Depthwise convolutions filter the input channels, but keep the dimensions, such that a 6x6x3 input image, as exampled before, will end up not as a 4x4x1 result after applying a 3x3x3

kernel, but as a 4x4x3 result. This depthwise convolution is then followed by a pointwise convolution, which is essentially the application of a 1x1 filter, which then functions to collapse those dimensions into one. 1x1 convolutions take up 95 per cent of the computational time of MobileNets [29]. These processes together are called a depthwise separable convolution, which works to effect that which a regular convolution achieves in one go. The end results of regular convolutions and depthwise separable convolutions are roughly similar, but regular convolutions expend more effort to get there. For 3x3 kernels, the depthwise separable convolution is 9 times as fast and achieves almost the same results. An added benefit is that we can also apply a ReLU activation function twice instead of once, allowing for more complex functions without adding extra computational cost. A full MobileNet network involves 30 layers. The design of the network is as follows:

1. Convolutional layer, stride of 2
2. Depthwise layer
3. Pointwise layer, doubling the number of channels
4. Depthwise layer, stride of 2
5. Pointwise layer, doubling the number of channels
6. Depthwise layer
7. Pointwise layer
8. Depthwise layer, stride of 2
9. Pointwise layer, doubling the number of channels

And so on and so forth, with ReLU activation functions in between. A stride of 2 is sometimes used to reduce the width and height of the data, and pointwise layers sometimes double the number of channels. In the end the input image is filtered down to 7x7px with a dimension of 1024, on which an average pooling is applied that ends up in a vector of 1x1x1024. This will function as the input to the final layer, which will output a softmax function.

In the paper the authors introduce two simple global hyperparameters that effficiently trade off between latency and accuracy. First there is the width multiplier (*alpha*), which can shrink the number of dimensions. If alpha is set to 1, the standard, then the network starts with 32 channels and ends up with 1024. The second is the resolution multiplier (*rho*), which can shrink the dimensions of the input image. A rho of 1 results in a 224x244px input size. Another option a user is granted is to include or leave out a group of 5 layers in the middle of the network. MobileNets are trained in TensorFlow, which is Google's machine learning framework [1]. MobileNets are trained using asynchronous stochastic gradient descent, a variant on stochastic gradient descent, and using RMSprop [69] for optimisation.

7 Sighted MicroPsi agents

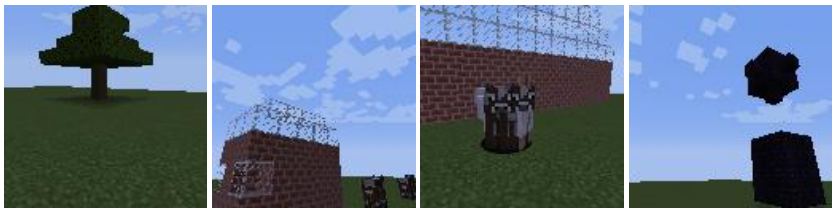
7.1 Experimental setup

The Minecraft world that is generated under 'normal' circumstances is largely unpredictable in its setup, and does not lend itself to reliably repeatable experiments due to its fluid nature. Malmo allows a user to build their own world through XML code that is loaded by the MissionSpec. I have set up an experimental ground, wherein I can run experiments in controlled fashion. I discuss the relevant settings of this experimental world here.

Malmo distinguishes between a server section and an agent section in its MissionSpec XML, which allows the user to set various options. In the server section I have set the following parameters: in *ServerInitialConditions*, under the *Time* header I have set *StartTime* to '1200' and *AllowPassageOfTime* to 'false'. This means that it is always 'day' within the Minecraft world, to improve recognition of objects, because they are more constant over time. Outside of the time settings, I have set the *Weather* to 'clear', to further enhance learnability, as rain might occlude the objects. For more variation, future experiments may also incorporate nighttime and varying weather conditions.

Under the *ServerHandlers*, using the *FlatWorldGenerator* option in the XML and a flat world generator [47], I created a flat world with grass only. In this flat world I draw a fence around my experimental ground, using *DrawCuboid* under the *DrawingDectorator* header. This concludes my barebone setup: a flat ground with a fence around it. The reason for the fence is so that I can keep the agent from running outside of my experimental area when it moves about.

On this flat ground I use Python scripts to insert objects within that world. These Python scripts code recipes, or 'object constructors' for four different types of objects: trees, houses, mansions, and art spheres. All scripts take coordinates for where they are supposed to stand within the experimental world. Trees are built out of wood and foliage, where the wood is a random height between 3 and 6 blocks, and the foliage covers the stem in pyramidal descent. The result is something that looks a standard oak or dark oak tree within the regular Minecraft world.



(a) Tree. (b) House. (c) Mansion. (d) Art sphere.

Figure 7.1: Example snapshots of constructed objects.

Then there is a script that generates a house or a mansion. If the 'small' argument is set to true the script creates a small house, and otherwise a larger house. A small house has a width of 1-3 blocks, a length of 2-3 blocks, and a height of 2-3 blocks. A mansion has a width of 4-10 blocks, a length of 5-10 blocks, and a height of 3-7 blocks. A randomiser chooses between a fence or no fence around the house, which is then placed there (or not). The type of wall is a brick block, the roof is made of glass, and the door is randomised over a few pre-defined choices. Future research may attempt to incorporate more varying structure in the objects.

Art spheres are generated by code that creates a big block with a big ball floating on top of it, for no reason other than to create some variation in the playground and the objects therein. Otherwise the classification between houses and trees may be too 'easy' because they are so dissimilar. Art spheres provide a kind of middle ground. The width and height of the art sphere are between 1-3 blocks, and the art spheres are made out of the obsidian blocktype.

In the *AgentSection*, under *AgentStart*, an agent is spawned in the middle of the playground, with a diamond sword in its hand (inventory slot 0). Under *AgentHandlers* the agent has a few options turned on through the XML. *ObservationFromRay* is added because I currently do not have a mechanism that allows me to know whether an object is centered in the visual field (object localisation). Using the ray we can request what block object is directly in the line of sight of the agent. By asking the ray whether the block in line of sight is a wood block, we can know whether the tree is in the middle of view. This is, of course, a hack and future work using object localisation should seek to do away with this and determine whether an object is directly in front based on object localisation network outputs. The rest of the XML is unmeddled with and has the 'default' settings—taken from Malmo's included Python example number 2. I now have an experimental flat ground, wherein I can spawn an agent and various objects using object constructors that are hand-coded.

7.2 Dataset extraction

Currently there is no available dataset for Minecraft objects together with convenient labeling, or at least not to my knowledge. For this reason, I have built a small dataset generator within the experimental world setup. This takes the normal flat ground setup as described above in the experimental setup, and spawns one object in the middle of the field. Then, by allowing for *AbsoluteMovementCommands* in the *AgentHandlers* section of the world's XML, the agent can spawn around the object and take snapshots from each position. I take a total of 4 distances from the object. Previously, I took 3 distances, but this resulted in poor object recognition from up close in-game. I added in a distance of just 3 blocks, so now I have 3,6,8, and 10 blocks as distances from the center of the object. Per distance 8 positions are taken in around the object, in every corner and half-way between each corner.

The agent is always facing the object using *setYaw*, pre-set for each position. Per position 9 orientation points are taken. Three different Yaws are set with *setYaw* at -20,0 and 20 degrees horizontally, which are each combined with a Pitch, using *setPitch* of 10, 0 and -25 vertically (taking a -30 or +30 takes the object too much out of vision). The result is 4 distances * 8 positions * 9 orientations = 288 images. The images are taken by requesting a video frame from the WorldState through Malmo and saved in a folder using the Python Imaging Library, with the name of the corresponding object as folder name. The pixel size of the saved images is 128*128px for each image, as this has been found to be sufficient for classification by the

MobileNet, and the lowest possible settings on the downloaded version of the MobileNet used in training.

The objects spawned are those specified above in the experimental ground section, plus an option to spawn 'nothing', which results in just that flat ground with different orientations on this flat ground and towards the sun (as is the case with the objects). This is in order to avoid the agent always trying to classify an object even when there are none. The resulting classes are: houses, mansions, trees, art spheres, and 'nothings'. Some resulting pictures are manually removed as they don't have the actual object in them.

7.3 MobileNet Training

The MobileNet model can be downloaded and easily instantiated in *TensorFlow*, through the *TensorFlow for Poets 2* repository [18]. Using this pre-downloaded network for transfer learning is very straightforward. A few settings are requested. The image size can be set to 128,160,192 or 224px, and we get to choose from the relative size of the MobileNet, 1.0, 0.75, 0.5 and 0.25, which are fractions of the largest MobileNet architecture. I have found that using a 0.25 MobileNet with a px size of 128 works well enough for present purposes.

The MobileNet is then trained (4000 iterations, which was already overkill: the network converged after a few hundred iterations) on the images that are collected by the automated dataset collector with a result of no less than 100.0 per cent accuracy (N=119) on test. The resulting fully trained network is saved in retrained graph for use in other applications. This is great because this model is very low in resources, and is the smallest option of both pixelsize and modelsize that MobileNet allows. Future research may look for even smaller networks, or test whether a larger set of classes affects the low requirements of the network.

7.4 Integration

I have built a rudimentary mission where a simple MicroPsi agent (a 'lumberjack') is placed in the center of the experimental setup, where it is surrounded by houses, trees, mansions, and art spheres (and sometimes nothing), which is done by a little script that takes object constructors and spreads the objects over the experimental ground. The agent is equipped with a diamond sword and turns around its axis and when it spots a tree, it moves forwards towards the tree, and smashes the block in front of it. This script, or plan, is implemented using a request confirmation network.

A neuron node is set up with an activation of 1. The neuron sends activation through a gen link to itself to keep the script perennially running. The neuron sends activation down to a script node with another gen link. This script node is sub/sur connected to three further script nodes, setting up a request confirmation network, because these script nodes are por/ret connected in sequence to each other. The first script node in this sequence takes care of the turning of the agent. Activation is sent down sub/sur to another script node which sends sub activation to a 'turn' actuator which maps onto the turn command in Malmo. There is a block-type sensor manually set up to check the block-type in the line of sight of the agent. If this block-type is equal to 'wood1' or 'wood2', then this sensor is turned on, and sends a negative signal to the turn actuator, because I set the weight to -1. It also sends a satisfying signal to the script node. This part of the plan is now satisfied and the agent can move on to the next part of the plan.

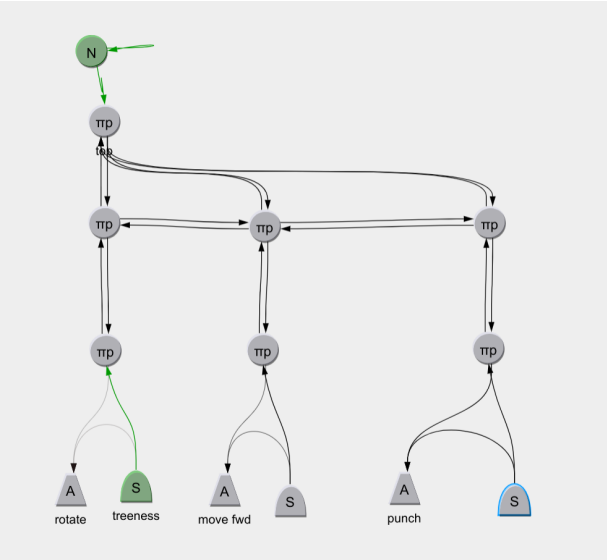


Figure 7.2: A request confirmation network that smashes trees.

A second script node in the sequence from the top most script node sends activation down sub/sur to another script node and then a sub signal to a movement actuator, that sends the agent forward. This movement signal is stopped when the block that is in line of sight is within hitting range (this is information that can be requested from the line of sight ray), by a sensor, that also sends a satisfying sur signal to its parent script node.

In the last part of the plan, a third script node in the sequence sends down sur activation to another script node that sends sub activation to a 'punch' actuator, that propels the agent to smash up the block in front of it. Then the sensor that measures whether there is a block in front of the agent is used to stop this action too with a negatively weighted link, and sends satisfying sur response to its parent script node, and then the action sequence in this request confirmation network is satisfied. After this last action, the script reloads, and the agent starts turning again, repeating the plan. The result is an agent that smashes trees and turns to find new trees to smash once it has completed smashing one tree.

While the agent turns, the 'update data sources and targets' method in MicroPsi's World Adapter class calls upon a method that classifies each incoming frame using the retrained graph. The softmax function at the end of the MobileNet network outputs a probability for each class, and essentially reports on its vision. Vision can also be integrated by sending activation to sensor nodes, based on certain thresholds of probability output. As such, visual information is already usable in this current setup, but my implementation is meant only to show that it works.

The network reports on vision with generally high accuraccy, especially on trees, which, if put in front of the agent, will result in an output of close to 1. The network is sometimes undecided between houses and mansions, as they have the same substrate, and may be indistinguishable from upclose, even to a human.

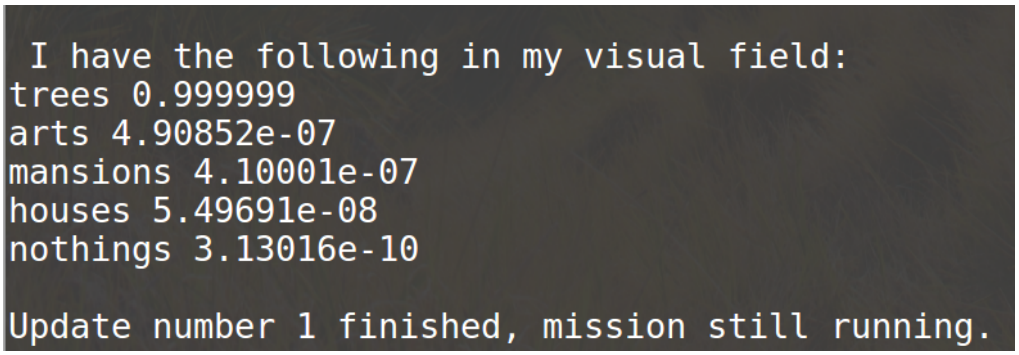


Figure 7.3: Vision report: a tree is being recognised.

8 Conclusion and Discussion

To explore the possibility of testing the MicroPsi cognitive architecture, I considered various options for virtual worlds, and singled out Malmo because it fits the AGI characteristics laid out by AI researchers as well as most of MicroPsi's motivational requirements. I made a succesful connection between MicroPsi and the Malmo API for experimentation in Minecraft. This means that we can interface a cognitive architecture with a satisfactory virtual world in which there is opportunity to run a large variety of experiments.

Furthermore, I have set up an experimental world within Minecraft-a kind of photo booth-wherein I spawn objects that I then take snapshots of using an automated script. I trained a pre-trained MobileNet neural network on the extracted images. In another experimental world, I spawned the same class of objects that I generated within said photo booth. I have built a neuro-symbolic agent that uses a request confirmation network within MicroPsi, and which scans the world for trees that are then hacked down by this 'lumberjack'. While this happens, the MobileNet scans the environment and reports probabilities on what it sees, and correctly recognises the objects that are in its view, and does so well. This shows that we can build neuro-symbolic agents that are endowed with some rudimentary form of vision, and can recognise objects in the game world without a lot of world-specific data with transfer learning and neural networks.

MicroPsi agents can now live within a virtual world wherein they are exposed to a large variety of tasks, challenges, and situations. Within this world, agents are equipped with a way to recognise objects on which they are trained, functioning as a proof of concept for further research in this area. Experimentation in virtual worlds for AGI research has until now been limited by worlds that do not generalise to the real world, as well as by a lack of experimental control over such virtual worlds.

An interesting find was that a neural network trained on real-world images transfers very well to the Minecraft world, meaning that Minecraft object recognition needs no special re-calibration of what it means to be an object. The transfer works so well, that there is little need for a complex model, and the MobileNet, which is already small, may be used in its smallest settings. This is good news for experimentation in Minecraft, because it means that computational resources may be used elsewhere, in collaboration with the use of vision.

There are however some limitations to my thesis, providing opportunity for further research. Firstly, my dataset is small, and hard-won. If we were to extend the approach of this study, we would have to build an experimental setup for every item or object within Minecraft, which will take a while, since these objects don't exist as-is in Minecraft, and have to be coded up block-by-block (though individual blocksized spawnable items are easy). An idea for further extensive research would be filming the Minecraft world with the built-in video producer and extracting the video frames to images which are then (still manually) labeled, and trained upon. Having more objects also provides more variety to test the model, as objects are now so dissimilar that it's easy to differentiate between them.

I have attempted to infer object categories from voxel/box information retrieved from Minecraft. There is a way to request what types of voxels are around the agent, such that you can receive a list that contains, for example, ['air', 'air', 'wood', ..]. Currently, Malmo allows the agent to receive information about what boxes are around it. This is available through adding ObservationsFromGrid to the XML and requesting these observations from the API. At the the time I ran the experiment, the problem was that there was no way of only retrieving the information of boxes that are actually visible to the agent, because every image in the box is logged. This makes it hard to correlate box information with objects 'seen' by the agent, because all objects are logged, even those that are occluded to the normal eye, as you can not see the back of the tree, and what is behind this. Through interaction with Malmo engineers through their chat, I read that they are working on such functionality, and this line of research may be revisited.

Secondly, my agent currently does not make good use of the visual information. It just blurts out what it sees into the void. In subsequent research, present findings can be extended with techniques such as YOLO [redmon2016you], such that we may localise objects, so the agent can move towards a tree or another object without having to resort to hacks such as the line of sight ray that was implemented to determine whether the tree was in front of the agent. A downside of using localisation techniques is that it takes a lot of work and data, and has for this reason been ommitted from the present proof of concept.

Lastly, my current approach makes no actual use of motivation in agents. This is not so much a limitation of functionality, rather than time and focus: this is already perfectly possible. I have it on good authority that my class mate Reinier Tromp is currently working to implement such an agent using the Malmo connection.

The current paradigm of using deep learning on every problem is not enough to reach artifical general intelligence. We have to integrate cognitive science into AI, in a formal and comprehensive way. Cognitive architectures provide such a way, by building frameworks wherein models of cognitive processes can be built and tested. MicroPsi is a strong architecture in the space of cognitive architectures because it puts agents in a world, wherein these agents are embodied, motivated, and can deal with both symbolic and sub-symbolic information. I have connected MicroPsi to Malmo, which means that we now have MicroPsi agents within the Minecraft world, with full experimental control, such that MicroPsi can be tested. MicroPsi agents are now situated within an excellent world for AGI experimentation, wherein they can learn to make autonomous decisions based on a variety of demands. MicroPsi agents are now equipped with a primary, visual way to interact with Malmo: through transfer learning on automatically gathered data. This provides a proof of concept for sighted, autonomous agents in Minecraft, and gives way to more experiments in virtual worlds.

9 Acknowledgements

Thanks to Joscha Bach, Jan Broersen and Chris Janssen for supervision and advice. Thanks to Reinier Tromp for collaboration and discussion. And of course thank you Jesus for being there for me always.

Bibliography

1. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard et al. "TensorFlow: A System for Large-Scale Machine Learning." In: *OSDI*. Vol. 16. 2016, pp. 265–283.
2. S. Adams, I. Arel, J. Bach, R. Coop, R. Furlan, B. Goertzel, J. S. Hall, A. Samsonovich, M. Scheutz, M. Schlesinger et al. "Mapping the landscape of human-level artificial general intelligence". *AI magazine* 33:1, 2012, pp. 25–42.
3. F. Allison, E. Luger, and K. Hofmann. "Spontaneous Interactions with a Virtually Embodied Intelligent Assistant in Minecraft". In: *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM. 2017, pp. 2337–2344.
4. J. R. Anderson. *How can the human mind occur in the physical universe?* Oxford University Press, 2009.
5. J. Anderson. "The adaptive character of thought. 1990". *Hillsdale, NJ: Earlbaum*.
6. J. Bach. "MicroPsi 2: the next generation of the MicroPsi framework". In: *International Conference on Artificial General Intelligence*. Springer. 2012, pp. 11–20.
7. J. Bach. "Modeling motivation in MicroPsi 2". In: *International Conference on Artificial General Intelligence*. Springer. 2015, pp. 3–13.
8. J. Bach. *Principles of synthetic intelligence PSI: an architecture of motivated cognition*. Vol. 4. Oxford University Press, 2009.
9. J. Bach. "Seven principles of synthetic intelligence". *Frontiers in Artificial Intelligence and Applications* 171, 2008, p. 63.
10. J. Bach. "The micropsi agent architecture". In: *Proceedings of ICCM-5, international conference on cognitive modeling, Bamberg, Germany*. Citeseer. 2003, pp. 15–20.
11. J. Bach and P. Herger. "Request confirmation networks for neuro-symbolic script execution". In: *Proceedings of the 2015th International Conference on Cognitive Computation: Integrating Neural and Symbolic Approaches-Volume 1583*. CEUR-WS. org. 2015, pp. 43–51.
12. S. Baker. *How could IBM Watson think that Toronto is a US city?* http://www.huffingtonpost.com/stephen-baker/how-could-ibms-watson-thi_b_823867.html. [Online; accessed 14-January-2018]. 2011.
13. M. G. Bellemare, Y. Naddaf, J. Veness, and M. Bowling. "The Arcade Learning Environment: An evaluation platform for general agents." *J. Artif. Intell. Res.(JAIR)* 47, 2013, pp. 253–279.
14. J. Bickle. "Multiple realizability". *Encyclopedia of cognitive science*, 2006.

15. M. Borst, G. Langner, and G. Palm. "A biologically motivated neural network for phase extraction from complex sounds". *Biological cybernetics* 90:2, 2004, pp. 98–104.
16. L. Bottou. "Large-scale machine learning with stochastic gradient descent". In: *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.
17. G Brockman, V Cheung, L Pettersson, J Schneider, J Schulman, and J Tang. *andW. Zaremba, "Openai universe,"* 2016.
18. M. Daoust. *TensorFlow for Poets 2*. https://github.com/googlecodelabs/tensorflow-for-poets-2. [Online; accessed 18-March-2018]. 2017.
19. D. C. Dennett. *The intentional stance*. MIT press, 1989.
20. D. Dörner. "Bauplan für eine Seele.", 2001.
21. J. A. Fodor. *The language of thought*. Vol. 5. Harvard University Press, 1975.
22. P. Geiger, K. Hofmann, and B. Schölkopf. "Experimental and causal view on information integration in autonomous agents". *arXiv preprint arXiv:1606.04250*, 2016.
23. X. Glorot and Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.
24. B Goertzel and P Wang. "A foundational architecture for artificial general intelligence". *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms* 6, 2007, p. 36.
25. E. Guizzo. "IBM's Watson Jeopardy computer shuts down humans in final game". *IEEE Spectrum* 17, 2011.
26. L. S. Hadla, T. M. Hailat, and M. N. Al-Kabi. "Evaluating Arabic to English machine translation". *Editorial Preface* 5:11, 2014.
27. S. Harnad. "The symbol grounding problem". *Physica D: Nonlinear Phenomena* 42:1-3, 1990, pp. 335–346.
28. K. He, X. Zhang, S. Ren, and J. Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
29. A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". *arXiv preprint arXiv:1704.04861*, 2017.
30. D. H. Hubel and T. N. Wiesel. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". *The Journal of physiology* 160:1, 1962, pp. 106–154.
31. M. Johnson, K. Hofmann, T. Hutton, and D. Bignell. "The Malmo Platform for Artificial Intelligence Experimentation." In: *IJCAI*. 2016, pp. 4246–4247.
32. E. Jonas and K. P. Kording. "Could a neuroscientist understand a microprocessor?" *PLoS computational biology* 13:1, 2017, e1005268.
33. A. Karpathy. "Lessons learned from manually classifying CIFAR-10". *Published online at http://karpathy. github. io/2011/04/27/manually-classifying-cifar10*, 2011.

34. T. Karras, T. Aila, S. Laine, and J. Lehtinen. "Progressive growing of gans for improved quality, stability, and variation". *arXiv preprint arXiv:1710.10196*, 2017.
35. J. Kemper. "MicroPsi and Minecraft".
36. A. J. Kolarik, S. Cirstea, S. Pardhan, and B. C. Moore. "A summary of research investigating echolocation abilities of blind and sighted humans". *Hearing research* 310, 2014, pp. 60–68.
37. A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.
38. J. E. Laird, A. Newell, and P. S. Rosenbloom. "Soar: An architecture for general intelligence". *Artificial intelligence* 33:1, 1987, pp. 1–64.
39. J. E. Laird and R. E. Wray III. "Cognitive architecture requirements for achieving AGI". In: *Proc. of the Third Conference on Artificial General Intelligence*. 2010, pp. 79–84.
40. P. Langley, J. E. Laird, and S. Rogers. "Cognitive architectures: Research issues and challenges". *Cognitive Systems Research* 10:2, 2009, pp. 141–160.
41. Y. LeCun, L. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, P. Simard et al. "Learning algorithms for classification: A comparison on handwritten digit recognition". *Neural networks: the statistical mechanics perspective* 261, 1995, p. 276.
42. H. J. Levesque. "On our best behaviour". *Artificial Intelligence* 212, 2014, pp. 27–35.
43. S. LeVine. *AI pioneer says we need to start over*. https://www.axios.com/artificial-intelligence-pioneer-says-we-need-to-start-over-1513305524-f619efbd-9db0-4947-a9b2-7a4c310a28fe.html. [Online; accessed 14-January-2018]. 2017.
44. M. Lin, Q. Chen, and S. Yan. "Network in network". *arXiv preprint arXiv:1312.4400*, 2013.
45. G. Marcus. "Deep Learning: A Critical Appraisal". *arXiv preprint arXiv:1801.00631*, 2018.
46. MicroPsi. *MicroPsi Industries*. http://www.micropsi-industries.com. [Online; accessed 11-April-2018]. 2018.
47. *Minecraft Superflat Creator*. http://www.minecraft101.net/superflat/. [Online; accessed September-2017].
48. V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. "Playing atari with deep reinforcement learning". *arXiv preprint arXiv:1312.5602*, 2013.
49. C. Mods. *Soartex Fanfare Vanilla*. https://www.curseforge.com/minecraft/texture-packs/soartex-fanver-vanilla. [Online; accessed 16-April-2018]. 2018.
50. M. Monfort, M. Johnson, A. Oliva, and K. Hofmann. "Asynchronous data aggregation for training end to end visual control networks". In: *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2017, pp. 530–537.

51. M. Moravcık, M. Schmid, N. Burch, V. Lisy, D. Morrill, N. Bard, T. Davis, K. Waugh, M. Johanson, and M. Bowling. "DeepStack: Expert-Level Artificial Intelligence in No-Limit Poker." arXiv preprint". *arXiv preprint arXiv:1701.01724*, 2017.
52. V. Nair and G. E. Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Proceedings of the 27th international conference on machine learning (ICML-10)*. 2010, pp. 807–814.
53. A. Newell. *Unified theories of cognition*. Harvard University Press, 1994.
54. A. Newell. "You can't play 20 questions with nature and win: Projective comments on the papers of this symposium", 1973.
55. A. Ng. "What Artificial Intelligence Can and Can't Do Right Now". *Harvard Business Review* 9, 2016.
56. M. A. Nielsen. *Neural networks and deep learning*. Determination Press, 2015.
57. D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul. "General video game ai: Competition, challenges and opportunities". In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016, pp. 4335–4337.
58. S. Pinker. "So how does the mind work?" *Mind & Language* 20:1, 2005, pp. 1–24.
59. P. S. Rosenbloom, J. Laird, and A. Newell. "The SOAR papers: Research on integrated intelligence", 1993.
60. D. E. Rumelhart, J. L. McClelland, P. R. Group et al. *Parallel distributed processing*. Vol. 1. MIT press Cambridge, MA, 1987.
61. S. Sabour, N. Frosst, and G. E. Hinton. "Dynamic routing between capsules". In: *Advances in Neural Information Processing Systems*. 2017, pp. 3859–3869.
62. M. S. Sajjadi, B. Schölkopf, and M. Hirsch. "Enhancenet: Single image super-resolution through automated texture synthesis". *arXiv preprint arXiv:1612.07919*, 2016.
63. P. Smolensky and G. Legendre. *The harmonic mind: From neural computation to optimality-theoretic grammar (Cognitive architecture), Vol. 1*. MIT press, 2006.
64. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A simple way to prevent neural networks from overfitting". *The Journal of Machine Learning Research* 15:1, 2014, pp. 1929–1958.
65. L. A. Suchman. *Plans and situated actions: The problem of human-machine communication*. Cambridge university press, 1987.
66. R Sun. *A detailed specification of CLARION 5.0*. Technical report. Technical report, 2003.
67. G. Synnaeve, N. Nardelli, A. Auvolat, S. Chintala, T. Lacroix, Z. Lin, F. Richoux, and N. Usunier. "Torchcraft: a library for machine learning research on real-time strategy games". *arXiv preprint arXiv:1611.00625*, 2016.
68. C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. "Rethinking the inception architecture for computer vision". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2818–2826.

69. T. Tieleman and G. Hinton. "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude". *COURSERA: Neural networks for machine learning* 4:2, 2012, pp. 26–31.

70. M. Wilson. "Six views of embodied cognition". *Psychonomic bulletin & review* 9:4, 2002, pp. 625–636.

71. R. Winkler. "Elon Musk Launches Neuralink to Connect Brains With Computers". *Wall Street Journal. https://www. wsj. com*, 2017.

72. L. Wittgenstein. *Philosophical investigations*. John Wiley & Sons, 1965.

73. J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. "How transferable are features in deep neural networks?" In: *Advances in neural information processing systems*. 2014, pp. 3320–3328.

74. M. Zastrow. *Machine outsmarts man in battle of the decade*. 2016.