Imitation Learning in Artificial Intelligence

by

Alexandros Gkiokas

Thesis

Submitted to the University of Warwick

for the degree of Computer Science.

Supervisor: Alexandra I. Cristea

Doctor of Philosophy

Department of Computer Science

September 2016

Contents

Acknowledgments

I would like to express my sincere gratitude to my supervisor Alexandra Cristea, without whom this thesis and research would never have taken place. My parents George and Maria, for never giving up on me and always believing. My current employer Stratos Arampatzis and Ortelio Ltd, as well as my previous employer Matthew Sewell and Athium Ltd, for their support, funding and understanding. Karen Stepanyan and Panagiotis Petridis for putting up with my questions, offering their support and advice, and guiding me through this adventure. Finally, Mina Makridi for putting up with me and for the past four years, and entertaining the notion that one day we'll witness true AI.

Declarations

This thesis is submitted to the University of Warwick in support of my application for the degree of Doctor of Philosophy. All experimental data presented and simulations were carried out by the author, except in the following cases: Daedalus experiments were done on-line after being approved by the BSREC with reference REGO-2015-1529. All experiments for Daedalus were carried out on the web and they represent anonymous data. Chapter 3 contains theoretical formulations [Gkiokas et al., 2014] carried out in cooperation with Matthew Thorpe from the Mathematics Institute in Warwick. Parts of this thesis have been published by the author, including submitted papers:

- Training a Cognitive Agent to Acquire and Represent Knowledge from RSS feeds onto Conceptual Graphs [Gkiokas and Cristea, 2014a].

- Unsupervised neural controller for Reinforcement Learning action-selection: Learning to represent knowledge [Gkiokas and Cristea, 2014b].

- Self-reinforced meta learning for belief generation [Gkiokas et al., 2014].

- Cognitive Agents and Machine Learning by Example: Representation with Conceptual Graphs [Gkiokas and Cristea, 2016a].

- Deep Learning and Encoding in Natural Language Understanding: Sparse and Dense Encoding Schemes for Neural-based Parsing [Gkiokas and Cristea, 2016b]

Abstract

Acquiring new knowledge often requires an agent or a system to explore, search and discover. Yet us humans build upon the knowledge of our forefathers, as did they, using previous knowledge; there does exist a mechanism which allows transference of knowledge without searching, exploration or discovery. That mechanism is known as imitation and it exists everywhere in nature; in animals, insects, primates, and humans. Enabling artificial, cognitive and software agents to learn by imitation could potentially be crucial to the emergence of the field of autonomous systems, robotics, cyber-physical and software agents. Imitation in AI implies that agents can learn from their human users, other AI agents, through observation or using physical interaction in robotics, and therefore learn a lot faster and easier.

Describing an imitation learning framework in AI which uses the Internet as the source of knowledge requires a rather unconventional approach: the procedure is a temporal-sequential process which uses reinforcement based on behaviouristic Psychology, deep learning and a plethora of other Algorithms. Ergo an agent using a hybrid simulating-emulating strategy is formulated, implemented and experimented with. That agent learns from RSS feeds using examples provided by the user; it adheres to previous research work and theoretical foundations and demonstrates that not only is imitation learning in AI possible, but it compares and in some cases outperforms traditional approaches.

Abbreviations

ADABOOST	Adaptive Boosting
AGI	Artificial General Intelligence
ANN	Artificial Neural Networks
AI	Artificial Intelligence
BFGS	Broyden-Fletcher-Goldfarb-Shanno
BPROP	Back Propagtion
CA	Cognitive Agent
CE	Cross Entropy
CG	Conceptual Graph
CNN	Convolutional Neural Networks
CRF	Conditional Random Field
DARPA	Defense Advanced Research Projects Agency
DNN	Deep Neural Networks
FOL	First Order Logic
FSM	Finite State Machine
GP^2U	General Purpose Compute on Graphics Processing Units

GOFAI	Good Old Fashioned Artificial Intelligence
HOL	Higher Order Logic
KR	Knowledge Representation
LBFGS	Limited storage Broyden-Fletcher-Goldfarb-Shanno
LMA	Levenberg-Marquardt Algorithm
LSTM	Long Short Term Memory
MBSGD	Mini Batch Stochastic Gradient Descent
MDP	Markov Decision Process
ML	Machine Learning
MRL	Meaning Representation Language
MSE	Mean Squared Error
NLP	Natural Language Processing
NLU	Natural Language Understanding
PBD	Programming by Demonstration
PBE	Programming by Example
POS	Part Of Speech
RBM	Restricted Boltzmann Machine
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
RPROP	Resilient Propagation
SMT	Statistical Machine Translation

SVM	Support Vector Machine
UI	User Interface

List of Tables

List of Figures

Chapter 1

Introduction

In the never-ending quest for Artificial Intelligence (AI), we take example from ourselves and our own intellect, since we are what we believe to be the most intelligent species on the planet. Intelligence however, did not spontaneously come into existence, but was the result of a painstaking process of evolution [Bjorklund, 2006; Wynn, 1985; Sternberg, 1982]; even more interestingly, scientists argue that there exist multiple intelligences and not just one [Gardner, 2011]. How those intelligences arose is a topic biologists, geneticists and psychologists researching human intelligence have been working on for more than a century; yet their beliefs and theories directly affect computer scientists working on AI. Our focus are machines: robots, software and hardware, artificial artifacts, upon which humanity is trying to instill intelligence and make them *as smart as humans*. Yet we cannot dismiss *how human intelligence* arose as outside the scope of AI, not only because it may be very relevant to the actual processes we're trying to recreate, in that there may be crucial information in the emerge of intelligence in Homininae, information that could make *Artificial General Intelligence* (AGI) a reality (AGI as in, an irrefutably intelligent, sentient and self-aware technological singularity [Goertzel and Pennachin, 2007; Kurzweil, 2005]).

1.1 Human Intelligence

Nowadays we know that one of the pieces of the puzzle that aided the emergence of intelligence is **imitation** [Dautenhahn and Nehaniv, 2002b]. Not only did imitation aid human intelligence, but there is overwhelming evidence in nature which suggests that imitation is one of the core mechanisms behind intelligence in animals, insects and Homininae [Fritz and Kotrschal, 2002; Herman, 2002; Visalberghi and Fragaszy, 2002; Galef Jr, 1988]. It is human intelligence that interests us the most, and there have been speculations that the "*great leap forward*", a period 200,000 years ago when our intellect exploded and we started creating tools, is mostly attributed to our ability to imitate [Ramachandran, 2000] in combination with our tendency to congregate and socialise. Thus it appears that the two driving factors of the *leap forward* were the development of communities and the ability to learn from others. By doing so, the *knowledge* of previous generations was passed down to the next generations, and thus individual knowledge, and by extension the collective knowledge began accumulating [Jones, 2009]. Through imitation, human societies taught their offspring how to create tools, how to farm, and most importantly how to survive. It thus becomes apparent that *learning* and *imitation* are closely knit together [Heyes, 2002]; we can't have imitation without the ability to learn, and, vice-versa, being able to learn is of questionable use when there exists no mechanism through which to acquire learning material.

1.2 Imitation in Humans

Imitation is a very broad term but appears to be *a low-level to mid-level ability of identifying examples, and learning from peers either via direct demonstration or from observation* [Dautenhahn and Nehaniv, 2002a]. Contemporary researchers argue that imitation is a unitary competency, a behavioural process that could have evolved as a unit and can be inherited as well as shared across a species [Myowa-Yamakoshi et al., 2004; Ferrari et al., 2006]. The implication of this argument is that in AI an imitation process is *something that can be learnt*; an algorithm, a heuristic or a cybernetic system.

Other researchers add that a neurological mechanism enables imitation in humans [Iacoboni and Dapretto, 2006; Iacoboni, 2005; Grèzes et al., 2003; Decety et al., 2002; Iacoboni et al., 1999] known as the "*mirror system*". Whilst the neuron-based imitation in humans is not fully understood, such a hypothesis could imply that an *artificial* neural-based imitation system could in theory be implemented. However, it is not currently known if the *mirror system* enables higher cognitive functions, or only sensor-motor functions [Bonini and Ferrari, 2011], although there exists evidence to suggest that social and higher level functions are indeed *partially* attributed to the mirror-system.

There exist many types of imitation: high-level, physical, hierarchical, structured and more [Dautenhahn and Nehaniv, 2002a]. Imitation may be *supervised* when a tutor or teacher provides paradigms and rewards, or it can be *unsupervised* when observation is the only means of acquiring learning material.

1.3 Imitation in Artificial Intelligence

Artificial imitation research has mostly focused on Robotics, in order to achieve a similar process to how infants learn movement and sensormotor abilities [Suleiman et al., 2008; Nakaoka et al., 2007; Breazeal et al., 2005; Breazeal and Scassellati, 2002]. Artificial imitation in applications not related to robotics deals mostly with programming by example (PBE) also known as programming by demonstration (PBD), the only imitation-related topic in non-robotic AI [Lieberman, 2001, 2000; Halbert, 1984]. Research in PBE/PBD is more than 15 years old, and was mostly concerned with programming and focused on user interfaces (UI). Regardless of the advent and subsequent sunset of PBE, imitation as a *learning* mechanism for AI, and more specifically for software *agents*, has been ignored and is to this very day an esoteric and perplexing topic.

The transition from PBE to AI agents is not an easy one; whereas PBE was concerned only with programming, AI agents are focusing on autonomy, learning, self-organisation, knowledge representation, logic and reasoning. An *imitating* AI agent is an even more peculiar entity: it does utilise the aforementioned topics, but it revolves around the combination of learning via imitation.

The notion that imitation is not learning is often perplexing, the reason being that learning is the main focal point of an agent without describing *how* the training material or samples have been acquired and used. Imitation thus focuses on acquiring learning material, training samples, data or information which is of use to the agent, and serves the purpose of acquiring new behaviours, performing new tasks or procedures [Dautenhahn and Nehaniv, 2002a]

Whilst learning and imitation are sometimes used interchangeably in AI and machine learning (ML), fundamentally they are different. The way in which the model, agent or algorithm acquires information and translates it into knowledge, is what differentiates learning from imitation [Dautenhahn and Nehaniv, 2002a].

A neural network being trained and evaluated by a user displays no form of imitation; yet an autonomous, self-trained and self-evaluating agent requires that it is able to identify paradigms from which it can extract samples, pre-process them, and then use them appropriately so that learning may occur. Before acquiring learning material an imitating agent must be able to extract or decode some kind of a paradigm which relates to what is being learnt. Post learning the agent should be able to reuse newly acquired knowledge, re-organise it, and transmit that knowledge to other agents [Lawniczak and Di Stefano, 2010].

1.4 Cognitive Artificial Intelligence

Due to the cognitive nature of such agents this research further explores cognitive AI and AI architectures. The term *cognitive agent* (CA) interchangeably used with the term "cognitive AI", requires that the agent must meet certain criteria [Lawniczak and Di Stefano, 2010].

- perceive information in the environment provided by other agents
- reason about this information using existing knowledge
- judge the obtained information using existing knowledge
- respond to other agents
- learn and augment current knowledge if newly acquired information allows it.

The above basic criteria set the bar for a *cognitive agent*, but an *imitating* agent has additional requirements, which are discussed in detail in Chapter 2.

This thesis sets the imitation requirements by taking into consideration the cognitive nature of such systems; the core premise of the imitative ability being the acquisition of knowledge by the agent and by drawing parallelisms from the observable, ostensible and discernible processes of the human cognitive system, thereby recreating the outcome of that process. Whilst the goal is not to produce a biologically-plausible system, the agent is driven by biomimicry since it demonstrates how AI might mimic human intelligence.

From the background research and work carried out in the last two decades, it can be asserted that imitation is not a unitary model, a finite-state machine (FSM) or an algorithm, nor is it a theoretical abstract; it is in fact a group of models, algorithms, a fusion or cascade of existing and new models into a software middle-ware, an agent. However imitation in AI and in robotics has not delivered its promises; PBE has eclipsed as a field, and robotics to this very day still depend on heuristic controllers. High level cognitive functions are usually *programmed* rather than learnt and few state-of-the-art experimental research have so far focused on learning by imitation. The main research question is therefore:

How can cognitive agents learn by imitation?

Due to recent developments in deep learning and cognitive AI and because of the complexity of such agents, this thesis implements an agent using an AI existing architecture: the *Icarus* cognitive model developed by Stanford University [Langley et al., 2003]. Icarus is a hybrid cognitive AI architecture, funded by Defence Advanced Research Projects Agency (DARPA) Information Processing Techniques Office, United States Office of Naval Research, and the Unites States National Science Foundation. It incorporates various models from across computer science, and it mainly focuses on *action* and *perception* over cognition.

Furthermore, Icarus separates categories from skills, uses a hierarchical structure for long-term memory [Langley et al., 2004] and uses *correspondence* between short-term and long-term memory [Langley et al., 2009]. Those four fundamental notions of Icarus are the basis upon which we draw comparison with the biological counterpart and the human imitation mechanisms and implement the set of those mechanisms in software. The advantages of implementing Icarus as a software cognitive agent are that it allows to examine the abilities, algorithms, processes and qualities that an *artificially imitating* learning agent *should or may* poses, formulate a theoretical model, examine the hypotheses via experimentation, and consolidate our conclusions through evaluation.

1.5 Icarus Engine

The Icarus implementation (called hereinafter *Icarus engine*) is greatly inspired by PBE which has its roots in Henry Lieberman's work [Lieberman, 2001]. However in stark comparison to Lieberman's PBE (described in Chapter 2.6), in this thesis Icarus is deployed as a stand-alone autonomous agent with the sole purpose of acquiring knowledge from the Internet.

The reason for aiming at acquiring knowledge from the Internet is its ubiquitous nature. According to the United Nations Telecommunications development Sector (ITU-D) around 40% of the global population has Internet access [Peña-López et al., 2009], and most of those users generate *content*, information, news, knowledge and data. Most of the human knowledge is being accumulated on the Internet, either in open and public sites such as *Wikipedia* or in specialist platforms, such *Quora* or *StackExchange*. Other knowledge engines (such as *Wolfram Alpha* or *DBpedia*) offer tailored meta-data, and last but not least, the blogs, new-sites, RSS feeds and social networks all provide *free* information and knowledge. Hence, the core research question is rephrased as:

How can an AI agent acquire knowledge from the Internet via imitation?

Using the largest knowledge pool in the history of the human civilisation is a promising source from which future AI agents can mature and reach higher-levels of intelligence. Thus the Icarus engine aims to acquire knowledge extracted from widely and freely available information found on the Internet. The Icarus engine is the first step towards an agent which learns by being taught *"how to read and understand"* the Internet data thereby transforming information into *knowledge.* Albeit the domain is natural language, it is not constrained by algorithms or models tailored for natural language processing (NLP) and should be able to parse and acquire knowledge from other domains. Its main purpose is to project textual information found on the Internet onto a knowledge representation (KR) structure. It does that by satisfying all the *cognitive agent* (CA) criteria set by [Lawniczak and Di Stefano, 2010], but it is not limited by finite-states or heuristics. Furthermore, the way the *memory* is organised adheres to the Icarus specifications of a hierarchical and structured knowledge index, with corresponding short and long term memory. The Icarus engine is in a sense a *parser*, which instead of being *programmed* how to parse, *learns how to parse* by example. By doing so, this thesis researches and experiments into the specifics of *imitation learning* and extracts conclusions about the suitability of such agents and systems for cognitive AI functionality.

1.6 Research Scope & Biological Plausibility

Choosing to implement such a CA as a parsing agent is due to the fact that such an ability is considered to be one of the high-level developmental steps in humans: learning to read and understand information [Stuart and Coltheart, 1988]. The approach taken is that of *simulation*: the process of mimicking the outwardly observable behaviour of children who learn how to read by being shown repeatedly text inputs of various (usually increasing) sizes. This thesis focuses on the third and fourth state of reading development [Seymour, 1999] due to the fact that those are the stages where decoding and hierarchical structuring develops. However, the work reported in this thesis draws no parallelism to the human brain, nor does it claim to *simulate* the same processes. Yet the choices of machine learning (ML) are all biologically-inspired, some based on behaviourist psychology, others use *artificial* neural networks, and only a handful are mathematical or heuristic components. This approach is thus indirectly based on the *human information processing models* [Berger et al., 2013], but does not implement them or as a whole; it only *appears* to be functioning in a similar way, in order to achieve similar goals, however as a cognitive *hybrid model.*

The advantage of taking this approach outweighs the effort of designing and implementing what can seem to be a complex agent: first and foremost the work is focused on an artificial agent which does not rely on *pre-programmed logic* but on *learnt logic*. This agent's learning is not constrained by the logic embedded in the program, but is adaptive and flexible. The implementation and experimentation of the Icarus engine theory is based upon the following rationale: *not programming an agent, only teaching it.* Whereas the actual implementation does indeed require to be programmed, it is done via a neuro-dynamic agent [Bertsekas and Tsitsiklis, 1995] using a *behaviourist* approach, similar to how humans learn from reinforcement Thorndike [1901]; Galef Jr [1988].

The advantages of *imitation* in AI are the same as the premise of imitation in nature; allowing agents to acquire knowledge and information from their peers, their social structure and our society, as well as surviving and evolving into capable entities. The implied novelty of imitating agents (cognitive or not) is promising: AI software which can seamlessly and effortlessly acquire and manipulate knowledge and information from humans directly or indirectly (through the internet), robots which can learn how to reason and use logic by example and through observing human interactions, and much more. A comparison between traditional software systems and imitating agents can thus provide the incentive to further explore imitation and support the usage of such agents in real-life applications. The advantages of enabling autonomous agents and systems to acquire and evolve their knowledge base only recently have been explored as corporations are gearing towards AI assistants, such as Microsoft's Cortana, Apple's Sirii, and Amazon's Alexa. It is a fact that such AI agents require imitation because it is the only known mechanism through which passive observation and proactive teaching enables information and knowledge to be acquired and manipulated. Therefore less central but still important research questions are:

- *What are the differences between learning by imitation and programming by example [Lieberman, 2001]?*

- *What are the advantages of agents which learn by imitation?* [Dautenhahn and Nehaniv, 2002a]

- *How do artificially imitating agents compare to traditional software systems?*

Other questions related to the imitation learning literature [Dautenhahn and Nehaniv, 2002a], such as "*what makes a good teacher?*" are still inherently relevant, but not as central to the work presented in this thesis. The field of *imitation learning* in AI has numerous applications and can be applied in a variety of ways with the potential to change high-level cognitive functions, such as *learning, reasoning, logic, decision-making*, etc. Although all those areas are relevant and applicable to the work described hereinafter, it would be impossible to include them all, experiment with a broad array of applications, or address all the entailing issues from each of those fields. Therefore, our only scope is *learning* and not *logic, reasoning* or other cognitive abilities. However, the Icarus engine *sets a basis* upon which logic and reasoning can take place in addition to learning using our theoretical model and software engine.

1.7 Contributions

- The main and foremost contribution to the field of AI is the *formulation and combination* of a Markov decision process (MDP) in a *temporal-spatial fashion* through which learning of symbolic KR structures (conceptual graphs) takes place [Gkiokas and Cristea, 2014a].
- This novel approach enables reinforcement learning [Sutton and Barto, 1998] *to manipulate as an episodic process the creation and representation* of a KR structure learnt by example.
- The importance of this contribution is explained in detail in Chapter 3 and challenges the way in which symbolic and connectionistic AI deal with data and knowledge, due to it demonstrating *how those two foundational approaches in AI can be bridged.*
- Furthermore, I address "learning by imitation" at the highest possible level in AI, that of symbolism, but learn it via reinforcement and deep learning.
- The imitation paradigm is given by a human user and is decomposed based on observations, similar to visual decomposition in the brain [Biederman and Gerhardstein, 1993], and inspired by the decoding process of the 3rd and 4th reading developmental stages in infants [Seymour, 1999].

Furthermore I created new algorithms and used them in experiments; decomposition heuristics, relational and attribute semantics and statistical inference. Those algorithms were implemented as parts of the Icarus CA, and used in order to examine both accuracy and suitability of such agents in AI [Gkiokas and Cristea, 2016a] and simulate the imitation process in humans, thereby formulating, evaluating and providing results and conclusions to the research questions aforementioned earlier. The cascade of various learning models used within the Icarus CA included artificial neural networks (ANN) and restricted Boltzmann machines (RBM) in combination with the reinforcement learning algorithm, as an action-selection mechanism for KR construction [Gkiokas and Cristea, 2014b], thus addressing observational qualities of the agent and *off-policy* exploration as well as *inference.* I also employed sparse and dense encoding with deep learning in Icarus in order to examine how it compares to more traditional shallow networks, drawing conclusions on the advantages and the complexity involved when using sparse non-processed encoding, whilst formulating alternatives to dealing with unknown input [Gkiokas and Cristea, 2016b]. In addition to experimenting with natural language understanding (NLU) in the Icarus CA, this work expanded into the field of *meta-learning* by formulating a new model based upon the same principle of MDP knowledge graph construction. We used *abstraction* of existing KR graphs and theorised it is possible to *compress* and *generalise* knowledge into *beliefs*; autonomously generated meta-KR constructs which represent a group or cluster of highly related KR instances [Gkiokas et al., 2014].

The contributions therefore are numerous and address imitation as a mechanism in AI and cognitive agents, all the processes involved, such as *decomposition* or *decoding* of paradigms, the main learning mechanism and models used to both *learn* and *associate* paradigms with *understanding* of the input (both semantically and syntactically) as well as a variety of learning models, algorithms and sub-processes required by the ad-hoc Icarus engine. I have expanded all research questions and mapped the characteristics and attributes that govern them and describe possible solutions to previous questions raised in the imitation learning literature [Dautenhahn and Nehaniv, 2002a].

1.8 Thesis Overview

This thesis is organised in the following manner: in Chapter 2 is described what has been researched in the past, all related fields, models and systems. The theoretical agent model is formulated and analysed in Chapter 3, and correlated to both the biological mechanisms and the Icarus CA design. Following the theoretical description, the data-set created to evaluate the Icarus engine is presented and analysed in Chapter 4. Chapter 5 analyses experiments carried out using the Icarus engine, describes in detail the components and algorithms, and reports on results and findings. The last Chapter 6 discuss in detail various findings, conclusions and future work.

Chapter 2

Background and Literature Review

In this chapter the background literature is detailed and an analysis and presentation on the work on which this thesis is based is examined. This aids in justifying the arguably unconventional - but highly utilitarian - approach taken.

2.1 Alan Turing and the intelligent machines

One of the forefathers of Artificial Intelligence was Alan Turing, amongst others such as Allen Newell, J.C. Shaw, Herbert Simon, John McCarthy and Marvin Minsky. Turing envisioned AI not just as an an intelligent machine, but as an artificial child, a synthetic entity which has to go through a developmental process to achieve intelligence [Turing, 1950]. How Turing had imagined that specific progression and development of AI is matter of speculation or scientific debate [Muggleton, 2014], but we can deduce from his paper that he believed AI would have to follow a developmental phase similar to that of infants. And as aforementioned, two of the most important developmental phases in infants revolve around imitation learning, symbol decoding and hierarchical representation [Seymour, 1999].

The brief history of computer science showcases that modern AI has not indulged Turing's original thoughts and ideas. Two different schools of thought have existed since the birth of AI: *Symbolic* AI, also know as *good old fashioned AI* (GOFAI), and *Connectionism* or *Connectionistic* AI[1]. Whereas Symbolism takes a *modelling* or a *programmatic* approach, *Connectionism* takes a pseudo-biological or

[1]Also known as *sub-symbolic AI*, albeit that term may have been coined by the symbolic school on purpose.

network-centric approach [Smolensky, 1987]. It is not clear if Turing had intended to take any of the two approaches or combine them; in fact although it was in the that the first artificial neural networks were theorised, actual implementations and models were published later, in the late 1950s, after his death. Therefore, Turing's approach was mostly theoretical and albeit based on biomimicry, it did not explicitly specify *how* those intelligent machines would be created, since when he wrote *Computing Machinery and Intelligence* [Turing, 1950], Connectionism was still in its infancy.

Around the same time, the field of *Cybernetics* [Wiener, 1948] was described as a interdisciplinary field for examining systems, their structures and organisation. Whereas Norbert Wiener differentiated Cybernetics from AI, there was a clear tendency of describing AI systems and agents in a cybernetic fashion: as well defined and modelled systems. The late 1940s and 1950s therefore saw the genesis of modern AI, which was for the largest part based on symbolic approaches: models, theorems, well defined processes and programs.

2.2 Imitation in Nature

2.2.1 What is imitation?

The exact nature of imitation has been studied only in modern sciences; it wasn't until the early 20th century that Edward Thorndike begun studying imitation in animals [Galef Jr, 1988; Thorndike, 1901]. The study of imitation in humans was belated; only after the modern field of Psychology started advancing (e.g., Jean Piaget and developmental theory, Raymond Cattell and crystallised intelligence, Burrhus Skinner's behaviourism and reinforcement, Erik Erikson and developmental psychology, and Albert Bandura and social cognitive theory) was imitation given some attention. The field of developmental psychology and in specific the development of children is what mostly interests us. As discussed in Chapter 1, Sections 1.1 and 1.2, one of the core mechanisms which enable human intelligence to develop and progress into what Cattell refers to as *crystallised intelligence* [Cattell, 1963], is the ability to learn from others, our environment, our parents and peers.

That development enables us to acquire knowledge through subjective experience, what Haikonen refers to as *qualia* [Haikonen, 2009]. However, that experience is in effect a *knowledge transference*, either sensorimotor, or more general and abstract. Verbalisation and learning by description plays a role (such as a teacher explaining things), but even higher level learning (such as learning how to write and read) are in fact imitation. The imitative counterparts are both physical and non-physical: the hand learns how to draw and write, yet the brain is conditioned into learning representations of letters, and then words. Psychologists suggest that imitative learning is a mechanism inherited and shared across humans [Myowa-Yamakoshi et al., 2004; Ferrari et al., 2006] and one which is the product of evolution; a hypothesis supported by the fact that not only humans are capable of imitation, but so are homininae, primates, animals and insects [Fritz and Kotrschal, 2002; Herman, 2002; Visalberghi and Fragaszy, 2002; Galef Jr, 1988], thus making imitation a cognitive function shared across multiple species (but not necessarily of the same competence level). Imitation therefore is *the ability to acquire knowledge from peers or others* either via active demonstration, or passive observation.

2.2.2 How does imitation work?

Not taking into account the neurological and morphological properties of the brain, and how those have evolved or how they enable imitation, Albert Bandura explicitly states (direct quotation):

> "*Attentional processes regulate exploration and perception of modelled activities; through retentional processes, transitory experiences are converted into symbolic conceptions that serve as internal models for response production and standard for response correction; production processes govern the organisation of constituent sub-skills into new response patterns; and motivational processes determine whether or not observationally acquired competencies will be put to use.*" [Bandura, 1986]

Analysing the above quote we can deduce a few key components and properties of imitation and its overall structure:

- Perception produces a model (or a modelled activity)
- A process retains a symbolic conception (through experience)
- The internal models serve the purpose of providing responses (or correcting responses)
- Motivation determines if a competency (skill) will be re-used.

Thus, and similar to how Thorndike [Galef Jr, 1988] and others have demonstrated, imitation uses rewarding (either via motivation or reinforcement) in order to *learn* a competency. The subjective experience (*qualia*) must perceive a temporal sequence (physical or non-physical) and then *model* it. That model is stored and uses some form of symbolism or conceptualisation and serves the purpose of being reused, either so that the agent can provide responses, or correct its responses. The key components are a model or structure which uses symbolism to represent an action, a sequence or behaviour, an episode or sequence, the related reinforcement, and the re-usability of the model.

The perceptive and cognitive abilities which relate to imitation have been studied in human infants [Seymour, 1999] and involve a *decomposition* (or decoding) of structures, objects or symbols and the internal *hierarchical* modelling (or representation) of those within the short-term, and if rewarded or reinforced, into the long-term memory. Visual *understanding* also uses decomposition [Biederman and Gerhardstein, 1993], thus it appears that the process of breaking down stimuli or information into primitives or archetypes, prior to internally representing them or modelling them, is a commonly occurring phenomenon.

2.3 Symbolic Artificial Intelligence

The symbolic school of thought has its roots in philosophical and centuries old beliefs about what intelligence and cognition is, and many of those beliefs stem from Julien Offray de La Mettrie and the *L'homme machine* (Man - Machine) [de La Mettrie, 1912]. The philosophical beliefs that followed suit of de La Mettrie's medical experiments, although revolutionary for his time[2] set the path for future research and development in Medicine, Psychology and eventually Artificial Intelligence. Those beliefs described the human mind as a complicated machine, a notion which supports Artificial Intelligence; it did however indoctrinate later research in Psychology, Cognitive sciences and AI to the effect where mental and cognitive abilities were believed to be definable as a process or a model, such as in the case of Allen Newell and the *Logic Theorist* [Newell and Simon, 1956] or the *General Problem Solver* [Newell and Shaw, 1959].

This belief still echoes in modern AI: many designs for cognitive systems take a purely *programmatic* approach, identical to a finite state machine (FSM) [Gill et al., 1962; Minsky, 1967] and thereby limited by its very *finite* nature and the logic enabling it. Admittedly, Symbolism and FSM are a necessity and part of what is today the modern field of AI and *Von Neumann* computers operate on a symbolic level. There are realistically many advantages to modern computing and Symbolism which are not easily dismissed and a plethora of algorithms and models

[2]He was forced to quit his position with the French Guards, due to his materialistic and quasi-atheistic conclusion that man is in fact a machine [Wellman, 1992].

with decades of research and a proven track record.

2.3.1 Knowledge Representation

One of the topics most important and relevant to imitation learning of Symbolic AI is *Knowledge Representation* or KR [Sowa, 1999]. Representing knowledge is a major field under AI, with close ties to Philosophy and Psychology. Whilst the philosophical aspects of KR are outside the scope of artificial imitation (e.g., "what is knowledge") the psychological attributes aren't; since the research in this thesis is using the developmental cycle of human intelligence as a reference point, employing KR requires that this work is at least partially based on models which are perceived to be plausible mental representation models.

It is not by chance that the issue of KR first arose during the development of the *General Problem Solver* [Newell and Shaw, 1959], that was also the moment when the symbolic school of AI underpinned the basis of KR. In the past three decades, multiple KR models and schemes have been devised, ranging from the family of *KL-ONE* [Woods and Schmolze, 1992], to Sowa's *conceptual graphs* (CG) [Chein and Mugnier, 2008; Sowa, 1999, 1984], and to more modern schemes such as the *resource description framework* or RDF [Klyne and Carroll, 2005] and the *web ontology language* or OWL [Bechhofer, 2009]. Most KR schemes feature similar designs: an ontology and the relations required to describe the hierarchy; it is intended to be used by computers (although it can be readable) and serves the purpose of representation but may also enable reasoning [Sowa, 1999; Levesque and Brachman, 1984]. KR can be used by *first order logic* or FOL [Fitting, 1990] which operates on the actual KR structure, and may be combined with or represent *semantics* [Fellbaum, 1998]. KR uses primitives (e.g., *domain archetypes*) which, depending on the application domain, may change. Meta-representation (or meta-data) is also used in most modern KR schemes such as RDF and OWL, which are applied *on top of* or as extensions to the primitives.

The most important aspects or topics related to KR are:

- *incompleteness* or *completeness* (e.g., *semantic, functional, refutation or syntactic* completeness associated with statement in the structure) [Lipschutz and Judith, 1916; Duffy, 1991]. Fuzzy logic is one of the sub-fields of AI which deals with a certain degree with incompleteness [Novák et al., 2012; Zadeh, 1996].

- *definitions, universals, facts and defaults* are general rules and patterns which relate to *specificity* and offer *quantification* and *generality* employed by logic operators [Leivant, 1994; Van Benthem and Doets, 1983].

- *non-monotonic reasoning* or hypothetical reasoning, asserts new hypotheses based on rules or facts [Dung, 1995].

- *expressiveness* or functional completeness relates to the ability of *adequately* expressing all truth tables when using first order logic (FOL) [Fitting, 1990] expressions such as *AND, OR* as well as *NAND, NOR.*

- *reasoning* in general terms relates to the ability of the agent or system to be updated, develop new inferences, and operate within reasonable time constraints.

From the above list it is possible to identify related material which overlaps KR, logic, learning and imitation. Whereas KR serves the purpose of *representation* and *description*, it is used by *logic.* For the intents and purposes of this thesis, I am mostly concerned with KR and how it is to be used and manipulated by imitating agents, rather than the logic enabled or applicable, which albeit relevant is outside the scope of the work described hereinafter. There is a clear connection between KR and logic, how KR enables or allows inference, hypotheses and propositions, and more important how the agent may *learn* to form such hypotheses, assertions and propositions. Yet my scope is *focusing on creating KR* and not logically manipulating it.

Another form of KR is the *meaning representation* (MR) or the most commonly used term *meaning representation language* (MRL). Those structures have been widely used in *parsing* (see Section 2.7.5) and are mostly related to *natural language processing* (NLP). The phrase "*quick brown fox jumps over lazy dog*" is shown as a simple MRL in Figure 2.1, whereas the same phrase with directed *edges* is shown in Figure 2.2.

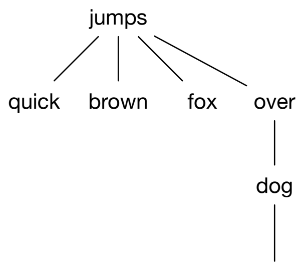


Figure 2.1: Simple MRL structure.

Whereas the first Figure 2.1 is overly simplistic, the addition of directed edges in Figure 2.2 demonstrates the importance of *directionality* in relations within a KR.

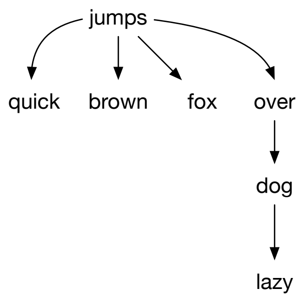


Figure 2.2: MRL with Edges.

In certain KR such as a Penn treebank tree[Marcus et al., 1993], direction is often implied originating from the root label/node of the tree graph. Such structures can often become esoteric or obscure, since they may include syntactic attributes mixed with words and labels, making them hard to understand. A modern MRL form of the same phrase which includes propositional and syntactic meta-data is shown in Figure 2.3; in this example, the blue labels in capitals are syntactic attributes (*part of speech tags*, see Section 2.7.4) with extra information including a rudimentary propositional logic (edge labels in pink).

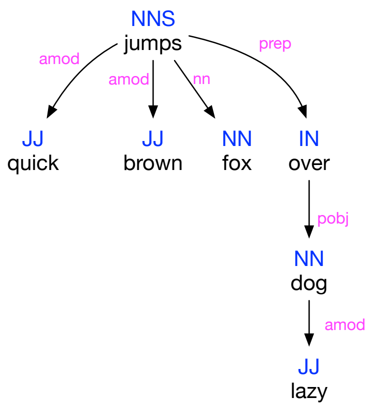


Figure 2.3: Annotated MRL with Edges and Meta-data.

The equivalent of that same phrase as a conceptual graph (CG) is shown in Figure 2.4. In this instance and according to Sowa's publications [Sowa, 1999, 1984], *rectangles* depict *concepts* and *circles* depict *relations*. Sowa's CG contain *implied* relations and not just relations *extracted* from the text such as relations between concepts. That approach might be confusing; whereas the *edges* describe the relations between the nodes within the graph, the graph is bipartite: concepts and relations are of different types. The edge label (number) indicates the order of the edges and some of the labels in the concepts are implied and not contained within the phrase, such as the relations *attr* (attribute), *act* (actions), *loc* (location) and *rcpt* (recipient).

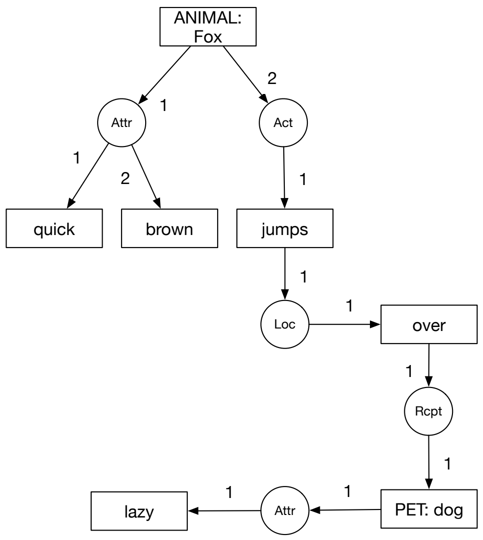


Figure 2.4: Conceptual Graph example.

Furthermore, Sowa in his original publications provided *contradicting* examples such as the one shown in Figure 2.5. Those examples are called a *display form*, which is a visualised conceptual graph. Figure 2.5 demonstrates the phrase "*a cat on a mat*"; the relation *on* is contained within the phrase, and is used to link the concepts.

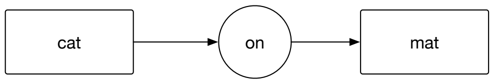


Figure 2.5: Conceptual Graph Directed Edges.

Part of the reason why Sowa treats CG (in their display form) in such a way may *possibly* be because he derives them from a logic form. The logic formula of Figure 2.4 may be as shown in (2.1) when *ignoring the attributes.* The root is the action *jump* which in the Figure is shown as a concept and based on a *Pierce* formula from which Sowa derived the CG in the display form.

$$(\exists_x \exists_y)\Big(Jump(x,y) \wedge Animal(fox) \wedge Pet(dog)\Big). \tag{2.1}$$

The *linear form* of (2.1) would be as shown in (2.2).

$$[Fox]- > (Jump)- > [Dog]. \tag{2.2}$$

Similarly, Sowa describes Figure 2.5 as having a *linear form*, as shown in (2.3).

$$[Cat]- > (On)- > [Mat]. \tag{2.3}$$

The *conceptual graphs interchange format* (CGIF), which was developed similar to the ISO Common Logic Project, would represent the linear form of (2.2) as $(Jumps[Fox][Dog])$ and the linear form of (2.3) as $(On[Cat][Mat])$. Therefore, Sowa did not attribute importance to either the *order* of appearance, nor to the direction of edges. He hypothesised that, albeit those forms may appear different, their semantic foundations translate to the same predicate calculus. Later, researchers in CG have used a better defined approach [Obitko, 2007; Amati and Ounis, 2000], in which the order of the graph is *drawn* from the first appearing node (concept or entity), and the direction of the edges is important, as it demonstrates the logic continuation of the phrase being represented.

Furthermore, whereas Sowa used *relations* in an implicit manner (e.g., extracting roles of concepts), there is no rule or limitation as to why relations (the nodes within the graph) must be *implied entities*; Sowa in his examples also used relations *explicitly obtained* from the original phrase.

2.3.2 Criticism and Limitations of Symbolic Artificial Intelligence

When researching imitating agents, we expect a form of KR structure to be produced which is identical or *highly similar* to the one the teacher, paradigm or demonstrator provided to the agent. That process, when described using symbolic AI, is a well-defined model, an algorithm, a program or a heuristic process, as that is the nature of Symbolism. Therefore, that process is in fact a *finite state machine* or a combination of FSM, described by a program or agent operating on the information on a symbolic level. In the history of AI the two cornerstones supporting the suitability of symbolic AI for our intents and purposes are the *Church-Turing thesis* [Searle, 2001] and the *Myhill-Nerode theorem* [Ignjatović et al., 2010]. The Church-Turing thesis states that a function is computable by a human following an algorithm, if it is computable by a Turing machine. Church-Turing thesis implies what can physically be computed by a computer [Piccinini, 2011] or what could realistically be computed, however multiple researchers in the past have argued for or against it [Goldin and Wegner, 2005; Cleland, 1993; Kalmár, 1957].

The Myhill-Nerode theorem offers insight into what can and cannot be done when using FSM: it suggests that any language can be recognised by a model, by mapping strings in the language to unique accepting states, and strings not in the language to unique non-accepting states. However, not all languages are regular, that is they do not correspond to the language accepted by any FSM, and equivalently, there may be no regular expression to represent that language. Furthermore, natural languages contain ambiguity and contradictions [Gorrell, 2006], which create exceptions to rules and the *logic* enabled by the FSM. Finally, the FSM may suffer from the *halting problem* [Stannett, 1990], and therefore the use of FSM, albeit advantageous, due to decades of research and the existence of multiple models, theories and algorithms, does have the aforementioned drawbacks.

2.4 Connectionistic Artificial Intelligence

Connectionism takes a *black-box* approach and is based on Neuroscience and the observation that our own intelligence *emerges* from neural networks in the human brain. Neuroscience and modern medical imaging have allowed us to observe and analyse the neuronal *network* mechanisms which enable cognitive functions and intelligent behaviour. It is from Neuroscience that *artificial neural networks* (ANN) were inspired, and in 1943 McCulloch described the first artificial neuronal model [McCulloch and Pitts, 1943]. In the 1949 Donald Hebb first describes Hebbian networks [Hebb, 1949], but it isn't until 1958 that the *Perceptron* is published [Rosenblatt, 1958], the precursor of modern ANN. In 1969 Minsky criticises Connectionism and neural networks [Minski and Papert, 1969] due to their computational limitations in training and deploying them, but most important due the fact that they couldn't learn to perform an *exclusive* (XOR) on the input. Those issues were addressed in the 1980ies and 1990ies with the advent of the personal computer, and *multi-layered* networks in combination with back-propagation [Werbos, 1974] showcased that they could process complex input. At the same time GOFAI lost interest from the research community, yet connectionism and the related ANN did not make any considerable breakthroughs until the late first decade of the third millennium with *deep learning.*

2.4.1 Artificial Neural Networks

An artificial neural network is a non-biologically plausible, yet bio-inspired network. It works upon the premise of associative memory and learns to associate input with output, or classify it, approximate it, perform regression, predictions, etc.

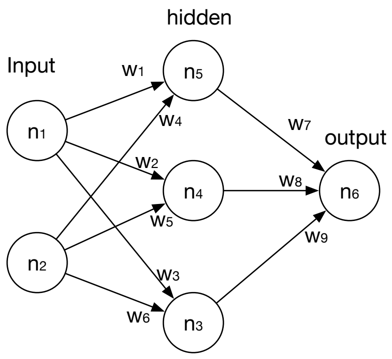


Figure 2.6: Example of a Neural Network.

[Picton, 1994; Hertz et al., 1991; Hopfield, 1988]. There are multiple types of neural networks, but we'll only examine and present the most typically used, the *fully connected feed-forward neural network*, also known as the single-layer or multi-layer *perceptron*. From the Figure 2.6 we observe what a neural network is: layers of nodes (or neurons) which are connected by *synapses*, or weights. In that figure above, there are three layers: the input layer with nodes n_1 and n_2, one hidden layer with nodes n_3, n_4 and n_5 and an output layer with a single node n_6. The premise upon which the feed-forward perceptron relies is forward propagation: an input vector of two values is fed into the input nodes, and is the activated, using an *activation function*, most often a *logistic* function, with the most popular shown below.

$$f(x) = \frac{1}{1+e^{-x}}. \tag{2.4}$$

$$f(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}. \tag{2.5}$$

The above functions are the *sigmoid* shown in (2.4) and *hyperbolic tangent* shown in (2.5); others less popular are the *sigmoid bipolar* in (2.6), the *scaled hyperbolic tangent* [LeCun et al., 2012] in (2.7), and the *soft-sign* in (2.8).

$$f(x) = \frac{-1+2}{1+e^{-x}}. \tag{2.6}$$

$$f(x) = 1.7159 * tanh(\frac{2}{3}x). \tag{2.7}$$

$$f(x) = \frac{x}{1+|x|}. \tag{2.8}$$

A characteristic of the logistic functions is that they produce a S shaped output. Non-logistic functions such as the *Gaussian* have been used but are not as popular. Common criticism for such networks, is that logistic activation functions cannot be used in *deep* networks; often there is no need for such networks, and one hidden layer may suffice to learn the task at hand. However, logistic activation functions often lead to *saturation* [Heaton, 2015], an issue which has been addressed by using *linear* activation functions.

The forward propagation in neural networks is a task of accumulating the input I_j multiplied by the weights W_i for each node and activating it via $f(I_i)$. The most commonly used approach is to do vector and matrix arithmetic: we obtain previous layer node output denoted as O_j where subscript j is the previous layer. For input layers, O_j is the input value I_j whereas for hidden layers we substitute O_j with I_i, e.g., the output of the activation function $f(x)$ from the previous layer j. Multiplying each node output $O_j \cdot Wi$ with every corresponding weight W_i is a *vector-matrix* product, we then *sum* each input vector for the nodes in layer i: $\sum (O_j \cdot W_i)$, e.g. reducing each produced matrix into a row vector of values, each row corresponding to node input I_i in layer i. Finally, each value in that vector is *activated* using one of the functions $f(I_i)$ as shown earlier in (2.4,2.5,2.6 or 2.7). The above step is repeated for all layers; that is in essence *forward propagation.*

Training neural networks is considerably more complex; one of the early critiques on neural networks was the computational requirements, and the biggest issue was that training was inefficient [Minski and Papert, 1969]. Nowadays, there are many training algorithms, the most notable of which are the *back propagation* (or BPROP); other methods also have been used, such as *reinforcement learning*, the *Levenberg-Marquardt* [Moré, 1978], and the *Broyden-Fletcher-Goldfarb-Shanno algorithm* [Shanno, 1985] abbreviated as (BFGS), amongst the most famous. The most commonly used training approach for ANN and shallow networks throughout the last two decades, has been the BPROP as described by Seppo Linnainmaa [Linnainmaa, 1970], and was later demonstrated by application [Werbos, 1982]. The basic mechanism behind BPROP, is that the observable error at the output layer, is back-propagated to account for each weight adjustment, until the desirable, or highly similar to the desirable output is produced by the network. We define the *error* as E shown in (2.9), e.g., the squared difference for each output node, where y_i is the *ideal node output* and $\hat{y}_i$ is the *actual node output.*

$$E = (y_i - \hat{y}_i)^2. \tag{2.9}$$

We then proceed to reversely iterate all the layers, first we calculate the output delta error, shown in (2.10).

$$\delta_i = -E \cdot f'\big(\sum (O_i)\big). \tag{2.10}$$

In (2.10) f' is the prime or derivative of the *activation function* and O_i is the node input. The sigmoid and tanh derivatives are shown in (2.11) and (2.12).

$$f'(x) = f(x) \cdot (1 - f(x)). \tag{2.11}$$

$$f'(x) = 1 - tanh^{2(x)}. \tag{2.12}$$

For hidden layers we calculate the value of the derivative on the node input $f'(\sum (O_j \cdot W_{ji}))$, e.g., the vector-matrix multiplication. Then those values are multiplied by the *next* weights W_{ik} and the next layer's delta error[3] δ_k, which are called *node deltas*, shown in (2.13).

$$\delta_i = f'\Bigg(\sum (O_j \cdot W_{ji}) \cdot \sum (W_{ik} \cdot \delta_k)\Bigg). \tag{2.13}$$

Finally, we calculate the weight gradient for layer i to k, as shown in (2.14), with each gradient multiplying the next layer's *node delta* and the *observed* node output O_i.

$$\frac{\partial E}{\partial Wik} = \delta_k \cdot O_i. \tag{2.14}$$

That gradient is then used in the *update* of the weight values. There are different ways of updating weights; in *batch* training, the gradients are *summed* $\sum \big(\frac{\partial E}{\partial W_{ik}}\big)$ and then used to adjust the individual weights at the end of an *epoch* (a training sample iteration), whereas in *on-line* training, the weights are updated after propagating a single training sample. The update rule in BPROP for batch training is shown in (2.15), the time-step t defines the index in time, and hence $t-1$ is the previous update, the *learning rate* α defines how large adjustments are made and the *momentum* μ affects current adjustment using previous adjustments.

[3]Please note we swap δ_k with δ_i from formula (2.10). At every backwards iteration, we replace δ_k in (2.13) with the next layer δ_i. Because this is a reverse iteration, we start at the output layer, and proceed towards the first hidden layer.

$$\Delta W_{ik}^{(t)} = \alpha \cdot \left(\frac{\partial E}{\partial W_{ik}}\right) + \mu * \left(\Delta W_{ik}^{(t-1)}\right). \tag{2.15}$$

Other training algorithms have been based on BPROP, most notably *resilient back-propagation* (RPROP) [Riedmiller and Braun, 1993], as well as many of its derivatives. The RPROP uses incremental small weight adjustments, and is assumed to be faster than BRPOP, as shown in (2.16).

$$\Delta W_{ik}^{(t)} = \begin{cases} -\Delta W_{ik}^{(t)}, & \text{if} \frac{\partial E}{\partial W_{ik}}^{(t)} > 0 \\ +\Delta W_{ik}^{(t)}, & \text{if} \frac{\partial E}{\partial W_{ik}}^{(t)} < 0 \\ 0, & \text{otherwise} \end{cases} . \tag{2.16}$$

The increments and decrements taking place are constant (or range bound) and $\Delta W_{ik}^{(t)}$ increases or decreases by η, where $0 < \eta^{-} < 1 < \eta^{+}$. Redmiller and Braun published a *kernel* of the algorithm, and reported significantly better results than BPROP [Riedmiller and Braun, 1993], since then newer versions have been published.

Neural networks have various ways of calculating the *error*; the most notable being the *mean square error* (MSE): $\frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$ for n samples. Nowadays for classification the *cross entropy/log loss* (CE) [Heaton, 2015, 104,120] is also used as shown in (2.17) where y_i is the ideal node output and $\hat{y}_i$ is the actual node output[4].

$$CE = -\frac{1}{N}\sum_{i=1}^{N}\left(y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i})\right). \tag{2.17}$$

All the above networks are trained in a *supervised* manner where training samples are obtained and associate input to specific output. Non-supervised learning before the advent of deep learning was a peculiar topic; the few models able to learn unsupervised were the *self-organising maps* (SOM) [Kohonen, 1990], the Hebbian-inspired *gas networks* [Fritzke et al., 1995] and the *K-means* family of clustering kernels [MacQueen et al., 1967].

Evidently, the field of neural networks had its ups and downs; in its infancy it was heavily criticised, it appeared to not live up to the expectations of revolutionising AI, and even after the 1980 developments they hadn't been widely adopted.

[4]Some papers or authors sometimes replace the *ideal* with y or t and the *actual* with a.

2.4.2 General Purpose Computing on Graphic Processing Units

Part of the reason of the advent of deep learning has to do with advances and recent changes from the traditional ANN to deep networks and activation functions, but it is also attributed to the new GPU hardware. Most of the graphics processing units (GPUs) on modern computers have hundreds of simple cores and even consumer-grade GPUs nowadays have thousands of cores. At the early third millennium researchers started using GPUs for *general purpose* computing [Thompson et al., 2002; Pharr and Fernando, 2005], coined *general-purpose computing on graphics processing units* (GP^2U), once it was realised that GPUs favoured parallel algorithms [Nickolls et al., 2008; Che et al., 2008; Owens et al., 2007] such as the training algorithms which are used for neural networks.

The nature of the matrix form operations on the input I_j and weights W_i, the vector-matrix multiplications during forward propagation or the back propagation such as (2.13, 2.14) as well as the training update rules such as (2.15, 2.16) can execute a lot faster when using GP^2U. Because GP^2U kernels are able to run asynchronous parallel operations on multiple training samples, weights and input vectors, a resurgence in the research of ANN took place before and during the rebirth of the field, now called *deep learning.*

2.4.3 Deep Learning

The pragmatic approach of using GP^2U from the research community enabled deep learning to advance into *realistic* applications dealing with computer vision, speech recognition, pattern recognition, classification, prediction, regression, approximation and auto-encoding.

Whereas the early ANN were mostly used for *toy problems* and trained using small data-sets, deep learning is able to crunch *big data* and has proven usable in real-life sceanrios and most important on par with human performance [He et al., 2015; Taigman et al., 2014] thus having a profound effect to the adaptation of deep learning in multiple fields and domains. Perhaps the most important breakthrough was recently in 2016 by Google [Silver et al., 2016] when *DeepMind* beat the world champion in the game *Go* by 5 - 0, thereby proving that deep learning and Connectionism are capable of *super-human* performance. Modern ANN (hereinafter *deep networks* or *deep learning*) differ from the traditional ANN in five distinct ways:

- Traditional ANN use *logistic* activation functions [Zadeh et al., 2010], e.g., sigmoid, tanh, arctan. Deep learning most often use linear functions such as the *rectified linear unit* (ReLU) [Nair and Hinton, 2010] or soft-sign [Glorot and Bengio, 2010].

- ANN normally use no hidden layers, or one to two hidden layers. In comparison deep neural networks use multiple hidden layers, hence the term *deep*. In some cases it is possible to stack different types of networks (e.g., as in the case of *auto-encoders*).
- Although not limited to deep networks, *node dropout*, *L1* and *L2* regularisation are new optimisation techniques which evolved as optimisation techniques for deep learning [Dahl et al., 2013; Ngiam et al., 2011b,a; Bengio, 2009].
- Other types of deep networks such as *convolutional neural networks* (CNN) are profoundly different from traditional ANN. They use *max pooling* and *convolution layers* (often in alternating multi-layered fashion) and perform classification at the final fully connected layer [Krizhevsky et al., 2012; Ciresan et al., 2011].
- Development of deep learning saw the use of soft-max as the output layer activation, instead of using traditional logistic functions. Whilst soft-max is applicable in shallow networks, its use in deep learning proved very successful [Glorot and Bengio, 2010].

The Figure 2.7 below showcases the difference between traditional and deep networks. More often than not deep networks are harder to train [Glorot and Bengio, 2010], they have more hyper-parameters which do not always warrant better accuracy or performance, such as in the case of increasing hidden layers and amount of nodes and quite often require more training iterations or epochs.

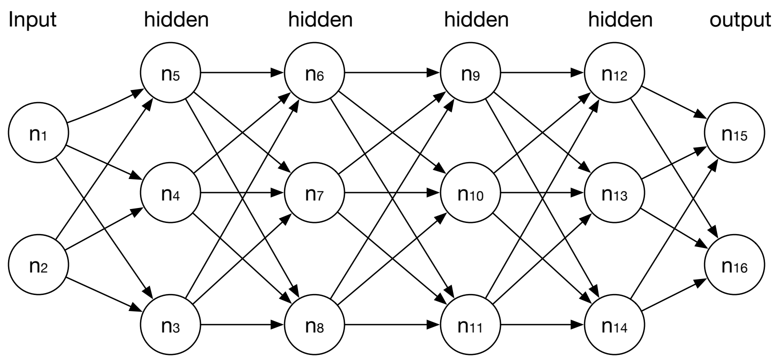


Figure 2.7: Example of a deep fully connected neural network.

Other types of networks used for deep learning are the *deep Boltzmann machines* (DBM) [Salakhutdinov and Hinton, 2009] which are stacked restricted Boltzmann machines (RBM), deep recurrent neural networks (deep-RNN) and different variations or combinations of the aforementioned networks.

The advent of deep learning was not restricted to *supervised* training but also included *unsupervised* training [LeCun et al., 2015; Le, 2013; Lee et al., 2009], which allowed discovery of structures from the network itself; coupled with the *unlabelled big data* found on the Internet and the progress of GP^2U it was easy to accept the use of deep learning as the cutting edge tool of Connectionism.

Using a *linear* activation function such as ReLU shown in (2.18) is very common in deep learning. This is usually paired with a *soft-max* activation at the output layer, shown in (2.19). Soft-max *squashes* the output within $0 \geq f(x) \geq 1$, and the sum of all outputs is 1, e.g., a probabilistic *likelihood* of output node O_x belonging to a specific class or group of the K groups or classes.

$$f(x) = \begin{cases} 0, & \text{if}(x < 0) \\ x, & \text{if}(x \geq 0) \end{cases}. \tag{2.18}$$

$$f(x) = \frac{e^{O_x}}{\sum_{k=1}^{K} e^{O_k}}. \tag{2.19}$$

The reason why that combination and usage is quite common, is because it has been shown to be very effective and fast at learning [Tomczak, 2015; Dahl et al., 2013; Nair and Hinton, 2010] and avoids saturation during training. This may be explained by the fact that ReLU has a simple derivative shown in (2.20) with range $[0, \infty]$.

$$f'(x) = \begin{cases} 0, & \text{if}(x < 0) \\ 1, & \text{if}(x \geq 0) \end{cases}. \tag{2.20}$$

Calculating the output delta error and gradient is shown in (2.21), with the derivative of soft-max f on output layer i being $f_i(1 - f_i)$, however the Jaccobian takes the form $y_i - \hat{y}_i$, e.g., the difference of ideal and actual output.

$$\frac{\partial E}{\partial Wik} = f(1 - f) = y_i - \hat{y}_i. \tag{2.21}$$

Another interesting update in deep learning is *regularisation* and its two forms: L1 and L2. Regularisation deals mostly with over-fitting which becomes a problem with modern large networks which are considerably larger than traditional ANN. In theory it is accepted that over-fitting *may be solved* by increasing the number of training data [Cawley and Talbot, 2007] but that may not always be possible. Another way of dealing with over-fitting is reducing the size of the network, which is done empirically, is time consuming and not always a good solution since a large network may normally perform better than a smaller network.

The L1 and L2 regularisation (also known as l1 and l2) prevent the coefficients from fitting perfectly to the training samples [Ng, 2004]. The L1 is shown in (2.22) and is based on the *least squares* using the sum of weights. The parameter λ is the *regularisation parameter* and N is the number of training samples, with $0 > \lambda \geq 1$.

$$\lambda \sum_{i=1} |W_i|. \tag{2.22}$$

L2 regularisation is slightly different and uses the sum of the square of weights, shown in (2.23).

$$\lambda \sum_{i=1} W_i^2. \tag{2.23}$$

Some authors and researchers prefer to *scale* the λ parameter via the use of the number of training samples, as shown in (2.24).

$$\frac{\lambda}{2N} \sum_{i=1} W_i^2. \tag{2.24}$$

The manner in which L1 and L2 are used is in combination with the *cross-entropy* function (see formula 2.17), for example the use of L2 and CE is shown in (2.25).

$$CE = -\frac{1}{N} \sum_{i=1}^{N} \Big(y_i log(\hat{y_i}) + (1 - y_i) log(1 - \hat{y_i}) \Big) + \frac{\lambda}{2N} \sum_{i=1} W_i^2. \tag{2.25}$$

The benefits of using L2 are finding small weights and minimising the cost function; L2 tends to be computationally efficient due to having analytical solutions, yet it does not perform *feature selection*(which may or may not be an advantage) [Ng, 2004]. In comparison the L1 is not efficient on *non-sparse* use cases but does tend to perform feature selection. Both L1 and L2 may be used in tandem, albeit anecdotal evidence suggests that L1 is more robust, but usage depends on the type of network and training data.

In addition to L1 and L2, *node dropout* as described by Heaton et al [Srivastava et al., 2014; Dahl et al., 2013], removes nodes from a network and its associated weights, thus making the network smaller. Dropout randomly chooses a node to dropout using a probability, and thus changes the network architecture and design.

Newer training algorithms, such as the Adaptive Boosting [Schapire and Singer, 1999; Freund et al., 1999] (ADABOOST) which won Yoav Freund and Robert Schapire the Gödel prize in 2003, or the mini-batch stochastic gradient descent (MBSGD) [Li et al., 2014] promise faster training.

2.4.4 Reinforcement Learning

A rare exception to the otherwise homogeneous field of neural networks is *reinforcement learning* (RL) [Bertsekas and Tsitsiklis, 1995; Sutton and Barto, 1998]. In general it is considered *dynamic programming* [Bellman, 1957; Weiss, 1960; Sniedovich, 2010] but it its not limited by it: it uses neuronal principles inspired from behaviouristic Psychology, simulating how agents learn by associating a cumulative reward with their actions [Watkins, 1989; Sutton, 1984; Galef Jr, 1988]. The learning uses *temporal differences* and is described by a *Markovian decision process* (MDP) [Howard, 1970; Bellman, 1957] such as the one shown in Figure 2.8.

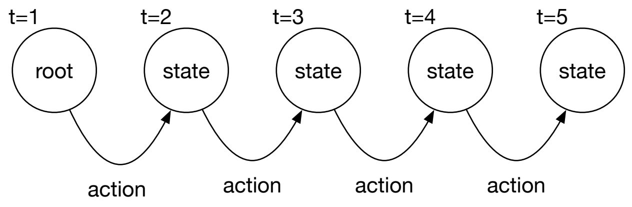


Figure 2.8: Reinforcement Learning Episode.

The basic foundation of an MDP as used in reinforcement learning, is the *episode*[5]. In the episode $e = (s_{t=1}, s_{t=2}, \dots, s_{t=n})$ we describe a temporal instance using time step t. The state s or when indexed by the time-step denoted as s_t is the core structure of reinforcement learning.

The true potential of RL can be achieved when we shift our preconception of what a *Markov* state is; the common perception is that a state uses arithmetic information but it can also be fuzzy, symbolic and use abstraction. The *state is traditionally* considered a *descriptor*; it may be fully or partially observable or hidden and can represent internal or external state of the agent, or a combination of both.

States are connected (or chained) by *actions*, actions performed by the agent, and which lead to the next state the agent experiences (often denoted s_{t+1}). The tuple s_t, a_t is the notation used which indicates that the agent has taken action a_t

[5]Episodes are used in episodic learning; there exists continuous learning which is a different topic outside our scope.

when in state s_t. In some literature a transition matrix $P = \big(p(s|s,a)\big)$ is often used; that transition defines the probability of transition from s_t to s_{t+1} in non-deterministic systems.

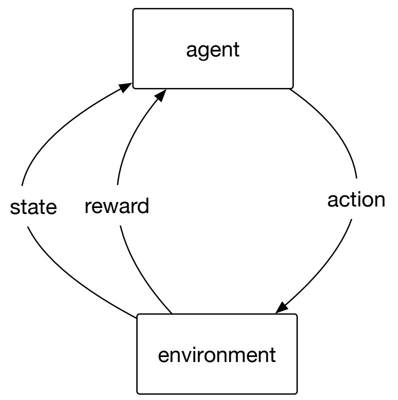


Figure 2.9: Reinforcement Learning Agent Interaction with Environment.

An example of the interactive cycle with the environment is shown in Figure 2.9; however there is another component which is used: the reward. Rewards are not obtained for all states but only for the *terminal* (or final) state which we denote with r, so that $r_t \in \mathbb{R}$ and $r_t = r(s_t, a_t)$. How the reward is obtained is subject to each specific use case: it may be internally calculated (e.g., self-rewarding), it may be observable in the environment, provided from another agent or entity, or the product of a fitness function.

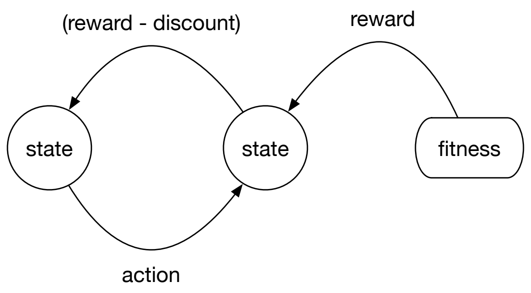


Figure 2.10: Rewarding the Terminal State

The reward is discounted using constant γ so that $\gamma \cdot r_t$ (or $\gamma \cdot r(s_t, a_t)$) back-propagates a smaller value to previous states[6]. The reward is used to implicitly calculate the value of a *policy* V^π or in deterministic scenarios $Q(s_t, a_t)$. Figure 2.11 demonstrates that a policy π is the *decision* to perform action a_t from state s_t.

[6]Please note, this is not neural network back-propagation.

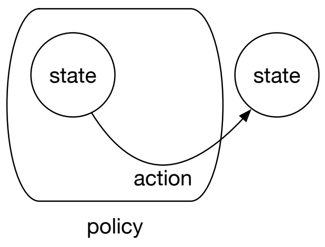


Figure 2.11: Example of a Policy.

There are two general types of learning in RL: *indirect* which is based on estimates (and probabilities) and *direct* where the optimal episode is learnt without first obtaining an explicit model. In direct learning the agent first has to *experience* and *observe* the episode, and then learn it via rewarding. We focus on the *direct* learning approach which is what is used in Temporal Dynamic learning or TD(λ); it offers the advantage of allowing full observability of states and the episode, and thus allows meta-information extraction. When the agent explores new policies, new episodes are experienced. In the case of starting from the same epicentre (the same root state) a tree graph is created and every time the transition matrix is populated with new or different states and actions, shown in Figure 2.12.

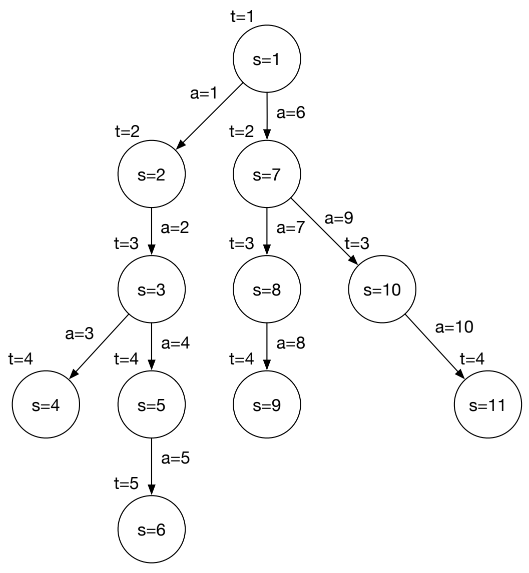


Figure 2.12: Epicentre of Multiple Episodes.

When an episode is experienced the agent uses the best or max valued policy $V^{*}(s) = \underset{\pi}{max} V^{\pi}(s)$ by looking forward to the next state's *policies* V^{π}_{t+1}. In the case of TD(λ) and deterministic systems we substitute $V^*(s) = \underset{a}{max} Q(s_t, a)$.

There are a variety of algorithms in TD(λ), one of the most widely used being the Q-Learning algorithm [Sutton and Barto, 1998]. The update rule for Q-Learning is shown in (2.26) and defines how values are calculated *after* having been rewarded at the terminal state, with $0 \geq r \geq 1$.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \cdot \Big(r_{t+1} + \gamma \cdot \underset{a}{max} Q(s_{t+1}, a) - Q(s_t, a_t)\Big). \tag{2.26}$$

In the above formula shown in (2.26), there are the following components: the previous value $Q(s_t, a_t)$ which is also updated, the learning rate α, the best policy calculated using $\underset{a}{max} Q(s_t, a)$, the reward r_{t+1} and its discount γ. Another algorithm in RL takes a similar approach: S.A.R.S.A the online TD [Sutton and Barto, 1998, 145], and with Q-Learning the offline TD [Sutton and Barto, 1998, 148] those two are the most famous and arguably the best to use.

2.4.5 Deep Reinforcement Learning

The emerging field of *deep reinforcement learning* [Mnih et al., 2015, 2013] from Google has showcased *better-than-human* performance when playing old video-games (Atari 2600). The idea behind deep-RL as described by Mnih et al is a fusion of traditional RL and CNN; the CNN act as pre-processors of the raw pixels crafting *states* and the RL learns the temporal sequence and the related policies; by doing so, complex information, objects and information is learnt by the *deep agent.* The overall idea is demonstrated in Figure (2.13) below [Mnih et al., 2015, 530].

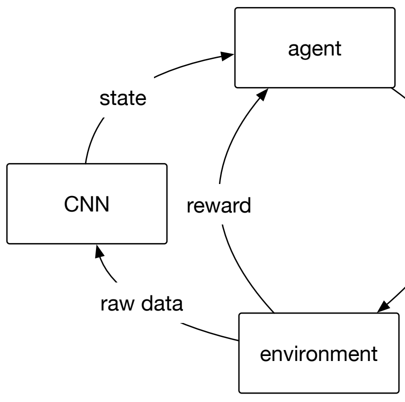


Figure 2.13: Deep Reinforcement Learning Agent & Environment.

This approach strongly demonstrates the multitude of uses of RL in combination with other models, algorithms or networks; furthermore it supports the notion that RL can act as the bridge between connectionism and symbolic AI. In the case of Minh et al, the *Q-learning* based policy formula is shown in (2.27).

$$Q^*(s,a) = \underset{\pi}{max}\mathbb{E}\Big[r_t + \gamma r_{t+1}\gamma^2 r_{t+2} + \ldots \mid s_t = s, a_t = a, \pi\Big]. \tag{2.27}$$

Minh et al empasise the use of ANN for action and policy approximation in RL, and albeit they support their choice based on the instability or divergence of RL when using non-linear approximators [Mnih et al., 2015, 529], they set the basis and notion of mini-batch random stochastic training of neural networks by sampling *action-policies*. They describe their deep learning algorithm as using Q-value updating periodically and from random mini-batches, thus smoothing over the changes in data distribution.

It should be noted that the deep Q-Network (DQN) did perform better than humans at specific games; in other old video-games it performed as good as humans and at a few it did worse than humans [Mnih et al., 2015, 531]. Some of the linear learners also performed better than DQN.

Regardless of the criticism of DQN and deep reinforcement learning, Google showcased and supported a notion we mentioned earlier and which as shown later throughout this thesis serves as the cornerstone of the Icarus engine. That notion is that RL can serve as the back-bone, the template to which other AI algorithms, symbolic or non-symbolic, are anchored. This allows for the *fusion* of machine learning, neural networks and other approaches, with symbolic Algorithms.

2.4.6 Connectionism Criticism and Limitations

One of the known limitations and criticism for connectionism and neural networks is their inability to operate on a symbolic level. Neural networks *usually* operate on real valued input, most often *scaled* and *normalised*. Other networks such as RBM are unable to operate on real values and use only binary values unless enabled by a *Gaussian-Bernoulli* visible node layer [Krizhevsky and Hinton, 2009]. Whilst a lot of uses of ANN are not confined by this limitation when exploring uses in NLP, NLU or other domains where the information is encoded in text, symbols or tokens, the inherent ability of ANN to work on *numbers* (prime or real) is a limitation. Another of those limitations is that Semantics are *lost* once text is used as input, unless the network is trained specifically to learn semantics on word observations [Dahl et al., 2012] or semantic similarity between words [Mikolov et al., 2013a,b,c]. However, doing so implies that multiple neural-networks must be used, each one tailored for a specific task; whereas that may not be an issue it gives rise to a plethora of disadvantages such as training and optimising multiple ANN, combining them in order to effectively achieve the final goal, pre-processing the input, etc. Furthermore, the researcher or developer faces various options: different models and architectures, hyper-parameter optimisation, architectural design and parameters, a variety or pre-processing choices when manipulating the information or data, each one affecting the final outcome, often in an unpredictable way. Whereas the aforementioned options are arguably not necessarily a disadvantage, they complicate agents and system design when compared to the clear-cut and well defined nature of symbolic AI.

2.5 Cognitive or Synthetic Artificial Intelligence

Quite recent sub-fields of AI are the *cognitive* and *synthetic* AI fields, which focus on the same topic: autonomous intelligent agents, systems or applications. A metaphor which may be used for cognitive and synthetic AI is that of the building blocks and the overall system; if symbolic AI and connectionism contain various types of building blocks (Algorithms, models, neural networks, etc.) then the fields of cognitive and synthetic AI describe the blueprint of how we may use those blocks to build complex agents, systems or software.

The difference between *synthetic* and *cognitive* AI is the following: the field of *synthetic AI* appears to cover a broad range of abilities [Bach, 2009] such as cognition, thought, perception, emotion, experience (just to name a few). In comparison, *cognitive AI* seems to focus a smaller range of abilities [Kinsner, 2006], although that is a supposition since current literature appears to cover similar abilities as those described in *synthetic AI*. Albeit the terms cognitive and synthetic are used interchangeably, in this thesis the preference is to use the word *cognitive* rather than synthetic.

2.5.1 Bach and Synthetic Intelligence

A large amount of research setting the modern foundations of cognitive AI was done by Joscha Bach who not only sets the basis for the transition from Psychology to Computational modelling [Bach, 2009], but also avoids picking sides in the Symbolic vs Connectionism debate. Bach appears to be mostly interested in Artificial General Intelligence (AGI) but sets a philosophical precedent when designing cognitive systems: to avoid specifics and focus on the greater picture. One could say that Bach indulges the readers to use a top-down (or stepwise design) approach which should result in the exhibition of intelligent behaviour. In general, Bach's advocacy and the *seven principles* [Bach, 2008] are:

1. Create holistic architectures which are functional rather than focusing on particular aspects.

2. Avoid methodologism, e.g.: not to assert that intelligence must be thoroughly explained by every single individual component.

3. Aim for the larger picture and avoid focusing on particular components or experiments. This statement holds true for many modern systems and is often a pitfall which prohibits the overall display of intelligence in favour of heavy optimisation or description of sub-components.

4. Build systems which are not too narrow or which do not focus only specific domains, also know as the *symbol grounding problem* [Harnad, 1990].

5. Robotic embodiment is not panacea. This statement is debatable and is likely to not be found agreeable or acceptable by all readers, however the proposition here is that cognitive AI should not be restricted only to physical systems, but could be explored and developed for non-physical systems.

6. Focus on autonomous systems, e.g.: systems which may not be the *best* at achieving their goal, but achieve that goal autonomously nonetheless when compared to expert systems or software which are designed for a sole purpose and highly optimised.

7. Emergence of intelligent agents won't take place on its own. This is an opinionated statement, yet it holds a certain gravity: AGI, AI or cognitive/synthetic AI is currently an *unknown* and we don't know *how or when* it will happen, and furthermore it *probably* won't happen in a day, but will be a long and tedious process, similar to how Alan Turing envisaged it decades ago.

From the aforementioned *seven principles* it is clear that Bach sets the guidelines for the design of intelligent systems and agents. Whilst that list is opinionated and perhaps slightly biased, it is useful because it serves as a starting point when deciding on how to create *cognitive agents*.

2.5.2 Haikonen and Cognitive Intelligence

Bach is not the only researcher to have carried out work in the field, Pentti Haikonen has done a tremendous amount of work on *consciousness*, robot brains and the components that could (or should) enable AGI. Haikonen's focal point appears to be *consciousness* and describes the components which he surmises to be crucial [Haikonen, 2007]:

1. Meaning and representation and their relation to information
2. Perception and recognition (visual, auditory, haptic, motor, objects, etc)
3. Association (and associative memory)
4. Motor actions (related to robotics and embodiment)
5. Cognition, understanding, memory (short-term and long-term) and models
6. Imagination and planning, deduction and reasoning

Moreover, Haikonen also analyses more challenging topics [Haikonen, 2012], such as:

1. Property dualism
2. The identity theory
3. Subjective experience
4. Externalisation
5. Attention and its relation to consciousness
6. Feelings and emotions

The most important topic which Haikonen has researched and is directly related to imitation learning, is *qualia* [Haikonen, 2009] the subjective experience and how it relates to objects (and therefore *observation*). Whereas the aforementioned topics are related to cognitive AI, not all of them are within our scope; for the Icarus engine the topics of most interest are the *subjective experience, emotions and rewards, meaning and representation, perception, recognition, decision making and associative memory.* Those topics directly relate to Imitation and are key components as identified by Psychologists (see Section 2.2.1).

The insight and arguments Haikonen provides in his book "*The cognitive approach to Conscious Machines*" [Haikonen, 2003], albeit sometimes a bit general and abstract, serve as one of the foundations upon which the presented research has been based. Haikonen describes the *same process*, e.g., an emulation (rather than a simulation) where the same process and the same output is expected. It is debatable as to why one approach or another may be better suited for artificial imitation. Furthermore, Haikonen summarises Gödel's theorem [Haikonen, 2003, 24–25], and the proposition that some arithmetic truths are not provable using the arithmetic system, as an argument against AI. He proceeds by asserting that the human mind (e.g., a mathematician) is capable of understanding that statement and theorem, and therefore Gödel's theorem does not disprove AI. Another argument against AI is that of determinism [Haikonen, 2003, 25–26]: since *thinking* is non-deterministic, GOFAI and similar symbolic systems could potentially suffer. Non-feasibility [Haikonen, 2003, 27–28], yet another argument against AI relates to the inability of programming to deal with every single possible scenario, whereupon Haikonen concludes that if programming is not a solution, then a solution would be to employ machines which learn.

All the arguments against AI aforementioned indirectly imply that non-symbolic approaches may in fact render that criticism mute: Connectionist designs are non-deterministic, non-symbolic, and learn instead of being programmed. That advocacy is found within all of Haikonen's books, he implies that consciousness and intelligence cannot be the byproduct of symbolic systems. Haikonen's analysis and persistence with *associative learning* and conditioning, as well as correlative learning make a strong argument for the use of neural models in cognitive systems. He also mentions *imitation learning* albeit briefly [Haikonen, 2003, 79–81], in support of its use for learning and knowledge acquisition. Although he doesn't expand he argues that other types of learning, such as trial and error, or by verbal description, are not as *profitable* to the system (or agent) as is imitation. Last but not least, Haikonen analyses emotion and its significance; in fact much of his analysis [Haikonen, 2003, 116-118] supports the use of learning systems based on emotion or rewards.

2.5.3 Five Cognitive Agent Criteria

Another recent research is that of Anna Lawniczaka and Bruno Di Stefano, who discuss in great detail about the specifics of designing cognitive agents (CA) and the need for standardisation [Lawniczak and Di Stefano, 2010]. Lawniczaka and Di Stefano mention the five CA criteria (mentioned in Section 1.4) which are deemed necessary for the deployment of CA. Furthermore, Lawniczaka and Di Stefano support the notion that CA should use hybrid models (symbolic and connectionistic), but propose to model CA similar to the OSI model [Stallings, 1987]. An argument against their approach is that existing meta operating systems, such as the *robot operating system* (ROS) [Quigley et al., 2009] take a computational graph approach which appears to be better suited and has been extensively tested, in comparison to their approach which is theoretical.

2.5.4 AI Architectures

A plethora of AI architectures exist yet some of them are outdated or were never implemented but remained a theoretical framework. Following is a brief analysis of the most significant and widely known AI systems.

4CAPS

This is a cognitive architecture by Just and Varma [Just and Varma, 2002] and it is the successor of CAPS and 3CAPS. It aims to be a biologically plausible cognitive AI and it relies mostly on Symbolic and less on Connectionism models. It has been implemented in software.

SOAR

The forefather of modern AI architectures it was created and described by Newell and Laird and aimed to model human behaviour and be a generic problem solver [Laird et al., 1987]. It uses Symbolism, representations, procedural, episodic and is declarative. It also uses reinforcement learning, imagery and emotional modelling. It uses explicit production rules to govern its behaviour (if-then-else). It has been used widelly and is still experimented with.

ACT-R

It was designed by John Robert Anderson and has evolved since then [Anderson, 1996]. Aims to reproduce the cognitive system and irreducible perceptual operations that enable the human mind. It uses individual processing modules that produce cognition inspired by Allen Newell and has perceptual-motor modules as well as memory modules (declarative, procedural). All modules can be accessed by their buffers; the contents of the buffers represent the state of the agent. Procedural knowledge is represented in the form of productions (the informational flow from the buffers to the cortex).

CLARION

This acronym stands for *Connectionist Learning with Adaptive Rule Induction - Online* and is a recent addition to the family of AI architectures [Sun and Zhang, 2006]. It makes a distinction between implicit and explicit processes and focuses on the interaction of the two. It has distinct subsystems each with implicit and explicit representations such as (a) Action-centred subsystem, (b) Non-action subsystem for general knowledge, (c) Motivational subsystem and (d) Meta-cognitive subsystem (monitor, direct, modify all other subsystems). It uses learning, inference, categorising, processing, reaction and creativity.

CHREST

This is a Symbolic architecture based on limited attention, limited short memories and chunking [Gobet et al., 2001]. It uses learning which is essential to the architecture, it is modelled as a network of chunks/nodes which are connected in various ways. Critics say that it has more similarities with Connectionist models than with Symbolic traditional models. It parametrises time which is an important key factor in its operation.

DUAL

This is a general cognitive architecture trying to implement both the Symbolic and the Connectionist approach at the micro level [Kokinov, 1994]. It is based on decentralised representation and emergent computation. Computations emerge from many micro-agents which are a hybrid Symbolic-Connectionist device. The agents exchange messages and information and activation via links that can be learnt and modified, thus forming coalitions which collectively represent concepts, episodes and facts.

R-CAST

This is a group decision support system [Fan et al., 2005] which uses multiagenct technology and a common shared knowledge space. It relies on the shared mental model about context of decision making, and is sased on naturalistic decision making.

LIDA

The acronym stands for *Learning Intelligent Distribution Agent* and is a model proven to work using experiments and empirically grounded [Franklin and Patterson Jr, 2006]. It is neither symbolic or connectivist but a hybrid model which attempts to cover a broad spectrum from low level to high level perception and reasoning. It uses cognition functions by iterating interactions (cognitive cycles); these cognitive cycles function as atoms for higher level cognition processes.

FORR

The acronym stands for *FOr the Right Reasons* [Langley et al., 2009; Epstein, 1992] and was inspired by Nobel laureate Herbert A. Simon's ideas on bounded rationality and satisficing [Bearden and Connolly, 2008], a decision-making system based on cognitive Heuristics. FORR Focuses on learning and problem solving, it is general enough for problem solving and has been tested thoroughly in robotics and software agents. It learns from experience how to solve problems and has 3 components: (a) Descriptives: they describe the state/problem, (b) Advisers: advise for the problem (rationales), and (c) Behavioural script: queries advisers and performs actions.

Icarus Architecture

The Icarus architecture [Langley et al., 2009; Langley and Choi, 2006; Langley et al., 2004, 2003] which serves as my starting point and CA implementation takes a compartmentalised approach rather than a layered approach. Although it is intended for *physical* agents, it uses an elegant and minimalistic design which simplifies the amount of components and processes. I make no claim that the Icarus engine is on par with the original intentions of the creators of Icarus; instead it is implemented by focusing on imitation only and is intended to be a node of a computational graph (ROS) offering NLU to the overall system or agent. The original goals of Icarus architecture [Langley et al., 2003] are:

- The integration of perception, cognition and actions
- To combine Symbolic structures and affective values, e.g., social affective learning and knowledge representation
- To behave re-actively in tandem with problem solving
- To learn from experience whilst also using domain knowledge

Icarus uses a long-term and short-term memory which is further compartmentalised into long-term conceptual and long-term skill memory and the respective short-term memories. It relies on *rewarding*, either past or expected rewards upon which it bases its decisions. The original design of Icarus used primitive concepts in the long-term conceptual memory: Boolean and quantitative values, yet it associates *objectives* to skills and concepts. I analyse the implementation of the Icarus engine in detail in Chapter 3, and explain how it differs from the original blueprint and how it adheres to the aforementioned goals.

Other Architectures

Others AI architectures have been designed and used in the recent past: PreAct engine[7] which now appears to be defunct, OpenCyc[8] a Semantic reasoning engine, OpenCog[9] an Artificial General Intelligence (AGI) framework [Hart and Goertzel, 2008], MicroPsi Project[10] based on PsiTheory [Bach, 2012], and PRS (procedural reasoning system) [Ingrand et al., 1992].

Choosing which AI architecture to use was a tedious task but ultimately I opted for the simplest, most minimalistic and easy to use. Whereas many of the aforementioned frameworks, engines and architectures offer a tremendous wealth of functions, my focus has been on learning, representation and imitation; as such the Icarus architecture is the best fit satisfying the criteria whilst remaining simple.

2.6 Programming by Example

The field most closely related to the research described in this thesis is *programming by example* (PBE), also known as *programming by demonstration* (PBD); the pioneers of the field being Henry Liebermann [Lieberman, 2000], Daniel Halbert Halbert [1984], Mathias Bauer [Bauer et al., 2001], Brad Myers [Myers et al., 1995], Francesmary Modugno [Modugno, 1996] and Richard McDaniel [McDaniel, 2001].

2.6.1 PBE: Theory and Models

In PBE *programming* is not done by the agent but by the user and the process is in fact a form of *inductive heuristic* where the user's paradigm contains *hidden states* [McDaniel, 2001]. The user *replicates* an internal mental process which is then transcribed and examples are extracted. Part of that process is *hidden*, implying a *partially observable Markov process* [Smallwood and Sondik, 1973] from which the agent is programmed. Smith et al describe it as the *creator approach* [Smith et al., 2001] where rules are inferred by observing changes from *before* to *after*. They describe the process, and assert that:

[7]http://artificial-intelligence.silk.co/page/PreAct
[8]http://sw.opencyc.org
[9]http://opencog.org/
[10]http://cognitive-ai.com/page2/page2.html

"*Each inferred rule represents an arbitrary number of primitive operations, or statements in other languages.*" [Smith et al., 2001, 10]

Smith et al declare the need and use of a representation, the process of observing changes in the user examples from which they infer rules. They expand on the need for a symbolic representation basing much of their work on the psychologist Jerome Bruner who focused on the cognitive development of children and its relation to education. According to Bruner there are three stages of development: *enactive, iconic and symbolic* [Bruner, 2009]. Inspired from Bruner Smith et al propose that the use of Symbolism governed by rules or laws, is a *Fregean* representation; they named it the *Runer's approach*, a *Symbolic-enactive* representation based on UI interactions and visual, iconic and symbolic representations.

McDaniel describes passive observation as *the passive watcher* in an argument against observation; he asserts that the *passive watcher* cannot request from the demonstrator the *hidden states* and that the *object* (e.g., paradigm or teaching material) may not be useful to the agent [McDaniel, 2001], without however proving it.

An *action-focused* approach is the one taken by Bauyer et al, who viewed the task of replicating the learnt process as a sequence of actions [Bauer et al., 2001, 100–102]. Although he did not expand much on the idea he named the processes "*recipes*" indexed by a *task library* in the agent's memory. Each recipe uses action estimation based on a monotonic probabilistic training approach [Bauer et al., 2001, 101], shown in (2.28).

$$EU(a, u, n) = [ass(w_a) - ass(w_{curr})] \cdot P_u(a) - annoy(u, n). \tag{2.28}$$

The formula describes the expected use of an action a, for user u with an already executed number of actions n. The descriptors $ass(w_a)$ and $ass(w_{curr})$ are the assessments of the best *policy*, before and after executing the action a. The probability $P_u(a)$ denotes user u carrying out action a and function *annoy* approximates the user's characteristics growing monotonically as the training increases. This is an *interactive* approach using a UI and it penalises the agent when requesting actions from the user.

Bauer et al further describe the issue of *sharing* knowledge between the user and the agent. This is a fundamental approach which relates to how agents are trained, the partial observability or hidden states, and how all aspects overlap (shown in Figure 2.14).

The depiction in Figure 2.14 demonstrates the issues arising from sharing

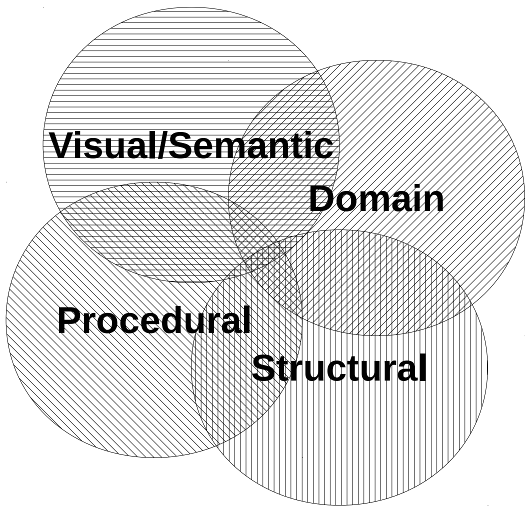


Figure 2.14: Knowledge sharing by agent and user

knowledge with the trainee (e.g., the agent). This dramatisation is quite accurate; since the agent *subjectively* experiences (e.g., qualia) the components shown above, those are often hidden, undeclared, invisible, implied or even unknown.

- Structural knowledge relates to the *internal* properties of the domain.
- Procedural knowledge relates to understanding part or the entirety of the processes involved.
- Visual and semantic knowledge relates to knowing and *understanding* relations and ontologies.
- Domain knowledge, is *specific* knowledge (e.g., in the case of NLP, *syntax and grammar*).

As evident from Figure 2.14, those categories often overlap; in some cases they can be acquired by other means (e.g., a lexicon or an external process), and in other cases they may be *inferred*.

Most of the PBE research uses simple inference, heuristics, rules and inductive logic [Lieberman, 2000; Myers and McDaniel, 2001; Modugno, 1996; Myers et al., 1995; Myers and Zanden, 1992; Zanden and Myers, 1990]. Machine learning was absent from PBE research, with the only exceptions being [King et al., 1992] and [Liebermann et al., 2001] albeit using rudimentary models. It wasn't until 2013 that researchers from UCLA, the Weizmann Institute and Microsoft publicised work [Menon et al., 2013] that combines the two fields; in this case, *rules, probabilities and associations* are *learnt* so that predictions could later be made by the system.

2.6.2 PBE: Application Domains and Criticism

The majority of PBE focused on user interfaces (UI); most of the PBE research used rule-based approaches in order to interact with the user and extract some form of a *program* [Lieberman, 2000; Myers and McDaniel, 2001; Modugno, 1996; Myers et al., 1995; Myers and Zanden, 1992; Zanden and Myers, 1990]. A few others were interested in automating information acquisition tasks [Bauer et al., 2001], mostly by transcribing interactions to queries. Bauer criticises PBE as having been overly simplistic and admits it was not designed with AI in mind (or by AI researchers) in what he calls *level of intelligence* [Bauer et al., 2001, 49]. Extracting rule definitions from multiple examples was considered a novelty [Liebermann et al., 2001] and it was applied in tasks such as *text categorisation.*

In hindsight, PBE took a minimalistic approach to enabling user-agent interaction, mapping of knowledge, action prediction (or suggestion); it was in fact focusing on *programming* and not learning. The early stage of UI and the graphical UI of the OSes at the time (Microsoft Windows 95, 98, 2000, NT) in combination with the fact that not as many people used computers back then as nowadays, may have contributed to the sunset of PBE. The web was just starting to appear, and personal computers were not as widespread as they are today.

PBE at that stage took a *toy-problem* approach and simplified the underlying mechanisms. Whereas complexity of an agent is a different topic, in the case of PBE it may have been detrimental to the abilities of the agents. No machine learning was used[11], few (if any) AI-related models or algorithms were employed; let us not forget that its focal point was extracting a sequence from the UI being replicated for the UI or operating system.

However, PBE set the foundations for much of what is *imitation learning* in AI today: it described the *example extraction* process as a MDP, it emphasised on the need for multiple examples, it showcased how visual generalisation (and visual UI interactions with agents) can be flexible and easy to use [Amant et al., 2001] and that PBE can be used for domain-independent tasks or use spatial representation for languages [Paynter and Witten, 2001]. Furthermore and perhaps most important it reinforced the notion of using Symbolic representation and requiring a *temporal* structure for procedures, and including the use of Semantics, and domain attributes.

[11]With the aforementioned exceptions of King [King et al., 1992] and Liebermann [Liebermann et al., 2001]

2.6.3 PBE: Differences from Imitation Learning

Evidently PBE was a subset of *learning by example* as it only focused on interfaces, automation and programming. There are many similarities with the work presented in this thesis, in fact we drew inspiration from Lieberman, McDaniel and Amant, however *imitation* in AI is more than merely programming. PBE sets the basis on using a visual interaction for creating examples, the need to use multiple examples, the use of inference in transcribing user paradigms to an MDP, and raises the issue of *partial observability* in the MDP/example.

Terminology issues aside (*programming by example* versus *learning by example*) the major difference is that in Icarus engine (a cognitive agent) I focus on *learning* which is enabled by *deep reinforcement learning* in combination with other techniques; there is no focus on the UI, and the user-UI interaction is of little interest to the field of AI.

2.7 Parsing and Understanding

The connection of NLP and NLU to CA and imitation is as follows. The *exact task* which the CA is performing (the Icarus engine) is in fact *parsing* an input sentence and projecting it to a KR, thereby creating a structured interpretation. The way semantic and syntactic parsing works in Icarus is complex and will be analysed in detail in subsequent Chapters. Notable research carried out in the fields of NLP and NLU is described, in order to be able to compare and explain performance, accuracy and design decisions.

2.7.1 Semantics

The word *Semantics* (from the Greek word "sěmantikós") is the field which studies *meaning* of words, symbols, tokens, signs and phrases [Ullmann, 1959]. Semantics are a closely related to the study of *Semiotics*, a *meaning-making* process [Carnap, 1948]. In general the topics related to semantics are:

- Semantics: the relation of symbols, words or tokens and their *meaning.*
- Syntactics: categorical or taxonomic relation between sets of symbols, words or tokens.
- Pragmatics: relation between symbols, tokens or words and agents using them or interpreting them.

From the above list it is evident that a cognitive agent deals with all three items as well as Semiotics. However in order to avoid being epigrammatic I herein refer to Pragmatics, Semiotics and Semantics as Semantics only. A distinction is made from Syntactics since syntax is been treated differently in NLP and NLU[12].

2.7.2 Distributional Semantics

The approach taken in distributional Semantics (also known as *bag-of-words)* offers a *point of view* regarding phrases; this field has been governed by what is known today as *vector space models* (VSM) and *feature vectors* [Turney et al., 2010]. The notion behind feature vectors and VSM is based on attributes; the presence or absence of words, tokens, symbols or pairs of those within a phrase or sentence. I denote a sentence or phrase as a *pattern* p from herein, whereas use the word *term* t to define tokens, words, symbols or signs. A VSM builds upon feature vectors; a feature vector is a vector *encoding* the presence or absence of a *term* t from an index. The index is a set of all *known* terms, words and symbols: $\mathbb{S} = \{\ldots\}, \mathbb{S} \neq \emptyset$. The *cardinality* of that set $|\mathbb{S}|$ defines the *lexicon* or index size m; feature vectors use that size to calculate the index or position of a term within the vector. The hypothetical index vector shown in (2.29) indexes some names and the binary feature vector shown in (2.30) denotes the presence or absence (in Boolean terms) of the respective name.

$$\mathbb{S}_m = \Big[Alex, Bob, Chris, John, \ldots, Xavier\Big]. \tag{2.29}$$

$$V_m = \Big[0, 0, 1, 1, \ldots, 0\Big]. \tag{2.30}$$

In the above example only the names *Chris* and *John* were present as encoded by the vector V_m. Therefore feature vectors represent presence or absence and not relation between terms such as words or symbols. The VSM builds on top of the notion of feature vectors by creating a matrix, a vector of vectors. The matrix $A_{m,n}$ where m is the lexicon size and n is the number of patterns p (phrases, sentences, etc.) represents which terms t are present and which are absent, as shown in (2.31).

$$A_{m,n} = \begin{bmatrix} p_1t_1 & p_1t_2 & p_1t_3 & \cdots & p_1t_m \\ p_2t_1 & p_2t_2 & p_2t_3 & \cdots & p_2t_m \\ p_3t_1 & p_3t_2 & p_3t_3 & \cdots & p_3t_m \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ p_nt_1 & p_nt_2 & p_nt_3 & \cdots & p_nt_m \end{bmatrix}. \tag{2.31}$$

[12]For example, Semantic and Syntactic parsing are not the same process.

In the above matrix $A_{m,n}$ columns are terms and rows are patterns; we can therefore retrieve *attribute* similarity between patterns, using the formula shown in (2.32) as described in the VSM literature [Turney et al., 2010].

$$Sim(V_i|A_{m,n}) = \frac{A_{m,n} \cdot V_i}{\|A_{m,n}\| \cdot \|V_i\|}. \tag{2.32}$$

The equation shown in (2.32) results in a vector of cosines: degrees Θ of similarity of the queried vector V_i respective to all other vectors (rows/patterns p) in matrix $A_{m,n}$ as shown in (2.33).

$$Sim(V_i|A_{m,n}) = \Big[cos(\Theta_{p_1}), cos(\Theta_{p_2}), cos(\Theta_{p_3}), \ldots, cos(\Theta_{p_n})\Big]. \tag{2.33}$$

From equation (2.32) and the produced vector of similarities shown in (2.33) it is evident that a VSM relies on the presence of terms, tokens, words, symbols in order to calculate *similarity.* Fine tuning of VSM often uses *term-frequency-inverse-document-frequency* (tf-idf) [Turney et al., 2010] by penalising frequently appearing tokens and biasing towards infrequent ones. Other optimisation techniques include *lemmatising* and *stemming* [Jivani et al., 2011], techniques which shift the morphology of terms t by grouping them or reducing them to their inflicted forms [Jurafsky and Martin, 2000].

2.7.3 Relational Semantics

Contrary to Semantics based on attributes (as discussed in 2.7.2), *relational* Semantics take a *taxonomic, categorical* or *structured* approach, and examine relations between *terms*, words, tokens or symbols. The pronounced tool used for such research is *Word-Net* [Fellbaum, 1998]; it is a lexical database which organises words into tree graphs. *Hypernyms* are trees which are produced when querying *specific* words and produce a tree going towards the most abstract or general term, the *root* of the graph, e.g., super-ordinates or super-classes. *Hyponyms* are tree branches, and are produced when querying words which are *more specific*, e.g., sub-ordinates or sub-classes. *Synonyms* as the term implies, are other *words* which are synonymous to the query.

In Figure 2.15 the query is shown in the centre of the graph. The graphs acquired by WordNet are *Hasse* graphs: mono-directional and ordered. Each hypernym, hyponym or synonym appear as blocks; in actuality WordNet groups those words as layers, however multiple layers may exist at the same level, creating *forks*

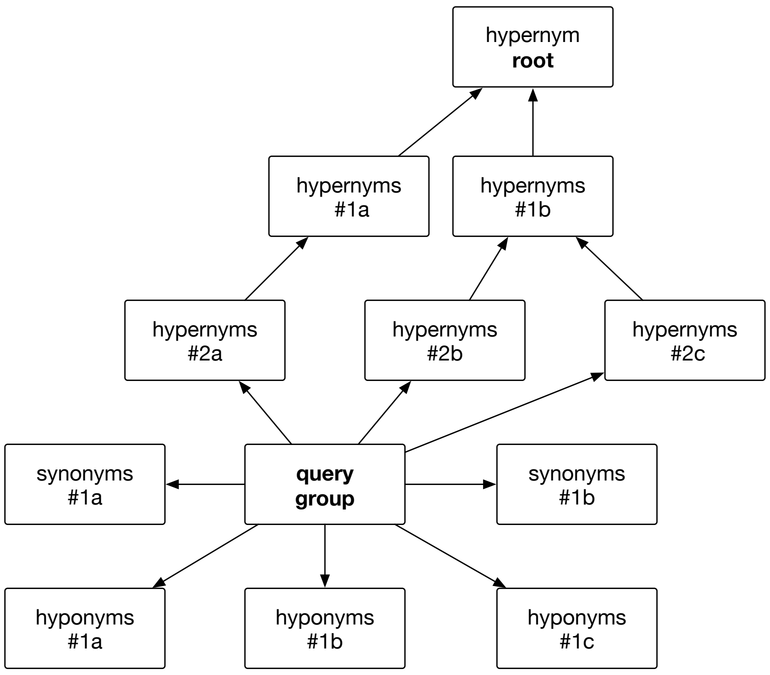


Figure 2.15: WordNet Semantic Tree Graph.

or branches semantically diverging from the same-level groups. The most abstract and general root layer is also shown; those are the taxonomies decided by the lexicographers who created WordNet [Fellbaum, 1998].

One of the limitations of this approach is that a *lexicographer* is needed; a human user who will organise, categorise and place the words in their correct taxonomies, in their respective positions. The implication which arises from the fact that although a cognitive agent may use semantic graphs such as the one one shown in Figure 2.15, is that the use of WordNet may be non-beneficial if it encounters unknown words, *particles* which are not indexed in WordNet, or other neoterisms, abbreviations, slang words or even words often used in Twitter or other online social media.

Google recently offered *word-2-vec* [Mikolov et al., 2013a,b] a solution to the aforementioned limitations, which uses CBOW [Mikolov et al., 2013c] (continuous bag of words) and/or Skip-gram [Guthrie et al., 2006]. Furthermore, *word-2-vec* assumes the role of the taxonomist; it observes and parses big data (extremely large datasets of text) and infers Semantic similarity between words as a vector. Google was not the first to offer this approach, earlier work [Dahl et al., 2012] has seen the use of RBM with *n-grams*.

A distinction which must be made is that *word-2-vec* is not a lexicographer;

albeit a lot more flexible than WordNet, it also is fallible to the limitations of the dataset used to train it, e.g., dealing with unknown words. The implied advantage is that *word-2-vec* may be trained on data, and thus this could potentially be an automated procedure which would not require a human user (the lexicographer) since it relies on observing samples.

A hypothetical scenario in this case would be to combine *WordNet* and *word-2-vec* in order to complement each-other, or even use multiple instances of *word-2-vec* which are trained on multiple data-sets in order to obtain a better mapping of Semantic relations in a domain.

The stark difference between them is that WordNet produces a graph which we can traverse and iterate in order co *calculate* the *semantic similarity* [Agirre et al., 2009; Jiang and Conrath, 1997; Rada et al., 1989] as we see fit; word-2-vec insted creates vectors. The fact that a graph *encodes* that information for us to explore allows the use of multiple approaches to quantifying the information of *similarity* [Lin, 1998]. On the contrary, *word-2-vec* provides a vector of values for the agent to consume; those vectors are classified by neural networks, obtained by models using the distributional semantics hypothesis (see 2.7.2). The issue with this approach is that a *word-2-vec* vector may encode or represent values in a different scale or manner than the one we intend to use with WordNet.

2.7.4 Part of Speech Tagging

Syntactic processing in NLP often uses *part-of-speech tagging* (POS or POS-tagging) which is a family of different algorithms and models with the aim of grouping words into *Tree-Bank* grammars [Charniak, 1996]. That is a set of 36 POS tags identifying specific categories of words and digits; it is a taxonomy based upon grammatical properties. The complete list of POS tags is shown in the Appendix (A).

Throughout this thesis, I use *laPOS* [Tsuruoka et al., 2011] (look-ahead POS tagger) which uses training models in order to predict correct POS tags. It is highly accurate (97.22% on the WSJ corpus) and the only one written in C++ and open-sourced.

2.7.5 Semantic and Syntactic Parsing

Parsing in NLP and NLU is a process; it *usually* processes text and produces some kind of output [Socher et al., 2013a]. More often than not the output has been a Penn Treebank tree [Charniak, 1996], and in most cases it is a KR structure such as an MRL. It is often partitioned into two distinct categories: *Semantic parsing* and *Syntactic parsing.* Often those two approaches are combined [Kambhatla, 2004] mostly due to the well known problem of *ambiguity* [Gorrell, 2006] an issue encountered by us human beings [MacDonald et al., 1994] as well as AI algorithms [Trueswell et al., 1994]. Parsing is often a task of *relation* extraction from the data [Mintz et al., 2009; Culotta and Sorensen, 2004]; it may also be a task of *concept* extraction [Villalon and Calvo, 2009] and in specific scenarios it can include both [Abebe and Tonella, 2010].

From concept and relation extraction, the task of *concept mining* [Looks et al., 2007; Bichindaritz and Akkineni, 2006] is closely related, whereupon algorithms or models attempt to extract or discover concepts, concept maps, or *knowledge* by using ontology [Ławrynowicz and Potoniec, 2011]. One of the most frequent application scenarios of parsing is *data-mining* [Cabena et al., 1998], which when targeting specific types of data becomes *knowledge extraction.* Parsing onto a KR, MRL or other type of representation is often called *dependency parsing*, *sentence modelling*, or *machine reading* but for our intents and purposes we use the umbrella term NLU.

Other application scenarios vary: parsing may be used for *sentiment analysis* [Liu, 2012; Wilson et al., 2005; Nasukawa and Yi, 2003], *text classification* [Bloehdorn and Moschitti, 2007; Zhang and Lee, 2003; Cavnar et al., 1994], *dialogue systems* [Vlachos and Clark, 2014], *machine translation* [Andreas et al., 2013; Wong and Mooney, 2006] and many more. Furthermore, there exists *shallow* parsing and *deep* parsing and the manner each one is implemented ranges by employing methods across the broad spectrum of modern AI. Regardless of the application domain, there appear to be four core elements to parsing:

- Parsing based on Syntax.
- Parsing based on Semantics.
- Parsing for information retrieval or knowledge extraction.
- Parsing for classification or prediction.

The first two items are self-explanatory; various researchers argue about their choices and often use one or the other, or even combine both approaches. The latter two items are somewhat hard to distinguish; research often focuses on *extracting* information or knowledge which later is re-used (such as when indexing books [Zhong et al., 2011], data-mining health-related data [Croitoru et al., 2007], parsing clinical data [Campbell and Musen, 1992], or simply text-mining [Montes-y

Gómez et al., 2002]). However, classification or prediction appears to aim towards associating a parsed input to an output, as for example in the case of sentiment analysis or dialogue systems, in which case the parsing mechanism serves the purpose of enabling classification or prediction. Those application domains have to use similar mechanisms and although the implementation model may vary, it is the process we're interested in, and not the domain itself.

2.7.6 Implementing Parsing and NLU

Current *state-of-the-art* only recently made significant breakthrough, some of it attributed to *deep learning.* The most recent breakthrough is from Google [Andor et al., 2016; Weiss et al., 2015; Zhang and McDonald, 2012] and the use of its *TensorFlow*[13] library as used by *SyntaxNet*[14]. Google relied mostly on POS tagging (the *Parsey McParseface* POS tagger) rather than Semantics, yet achieved the best F_1 scores to date (F_1 results range from 94.44% on news data, 95.40% on question-answers and 90.17% on web data).

Other *state-of-the-art* is the research and platform by the *spaCy* [Honnibal and Johnson, 2015] start-up in Germany. Current *software, tools and platforms* considered state-of-the-art are shown in Table 2.1; for a full list and analysis Choi et al [Choi et al., 2015] have gauged performance, accuracy, speed, etc., yet some platforms have been renamed, and some appear to be unmaintained or deprecated since then.

Name	**URL**
Deep-Syntactic Parser	`https://github.com/talnsoftware/deepsyntacticparsing`
MaltEval	`http://www.maltparser.org/malteval.html`
NLP4J	`https://github.com/emorynlp/nlp4j`
RedShift	`https://github.com/syllog1sm/Redshift`
RBGParser	`https://github.com/taolei87/RBGParser`
SNN	`http://nlp.stanford.edu/software/nndep.shtml`
SpaCy	`https://spacy.io`
SyntaxNet	`https://github.com/tensorflow/models`
TedEval	`http://www.tsarfaty.com/unipar`
TurboParser	`http://www.cs.cmu.edu/~ark/TurboParser`

Table 2.1: State of the Art NLU Software Tools.

Dependency parsing (hereinafter NLU) has seen some very innovative research using mechanisms which were tailored for computer vision such as CNN

[13]`https://www.tensorflow.org`

[14]`https://github.com/tensorflow/models/tree/master/syntaxnet`

[Kalchbrenner et al., 2014] or DNN [Grefenstette et al., 2014]. *TensorFlow* in *SyntaxNet* allows the use of both deep and shallow neural networks; Google's work used deep networks with sparse input [Weiss et al., 2015] in combination with *Graph-based parsing.* They also compared shallow networks [Andor et al., 2016] with one to two hidden layers to Long-Short-Term-Memory (LSTM) networks [Zhou and Xu, 2015], a special type of RNN often used in NLP. From Table 2.1 we've investigated much of the research work and implementation methods: most researchers also focus on *Syntactic* parsing rather than Semantic, a considerable amount of work relies on *tensors* [Clarke, 2015; Lei et al., 2014], *machine learning* [Ballesteros et al., 2014; Chen and Manning, 2014], whereas others use some form of *inference* [Zhang et al., 2014b,a; Martins et al., 2010] often probabilistic, *Heuristics* [Martins and Almeida, 2014; Martins et al., 2013; Tsarfaty et al., 2011; Bohnet, 2010], *Statistics* [Tsarfaty et al., 2012a], or a combination of the above [Rasooli and Tetreault, 2015]. Notable research has been carried out by Standford University [Berant and Liang, 2014] using *paraphrasing* and VSM on question-answer scenarios and recurrent neural networks (RNN) on Treebanks [Socher et al., 2013b], albeit not as well performing as the aforementioned work and tools.

More traditional approaches have used probabilistic or statistic approaches such as *statistical machine translation* (SMT) [Andreas et al., 2013; Wong and Mooney, 2006] often with some kind of Heuristic algorithm or kernel [David L. Chen, 2010; Chen and Mooney, 2008] or inductive logic [Tang and Mooney, 2001]. A large amount of research has focused on ML-models [Pradhan et al., 2004] such as *support vector machines* (SVM), RNN [Socher et al., 2011], neural-based *conditional random fields* (CRF) [Durrett and Klein, 2015] a hybrid approach of statistics and ML, whilst other hybrid approaches use neural networks, SVM and Semantics [Liang and Potts, 2015] or tensors [Clarke, 2015] (e.g., feature vectors from detected patterns).

Older research in parsing was mostly based on programming: Algorithms, Heuristics and logic [Shi and Mihalcea, 2005; Culotta and Sorensen, 2004], then mid early 20th century a shift towards *Statistics* took place as evident by the development of *WASP* [Wong and Mooney, 2006] and its later variants [David L. Chen, 2010] which combined statistical alignment with Heuristics, or other similar work based on Statistic-Heuristics [Och and Ney, 2003], such as *DAGGER* [Davies and Edwards, 2000], *TextRunner* [Yates et al., 2007; Etzioni et al., 2006], and *DIRT* [Lin and Pantel, 2001a,b]. Older systems and Algorithms relied on *alignment*, *inference* and *rule extraction* [Dinu and Wang, 2009; Szpektor and Dagan, 2008], and even the latter systems which combined Statistics and Heuristics (such as WASP) were prevalent until the end of the previous decade.

At the same time that research in NLU started focusing on ML and its models (late first decade of the second millennium) a plethora of new approaches started appearing such as the use of reinforcement learning based on *DAGGER* [Vlachos and Clark, 2014] on action-selection (predicting arguments, dialogues, etc), the use of *imitation learning* for prediction and classification of entities and relations [Vlachos, 2012], and unsupervised learning [Vlachos, 2011] which were not as accurate as current state of the art, but entirely outside the norm, thereby offering new approaches even as proof of concept.

2.7.7 Models and Algorithms in NLU

Apart from all the aforementioned approaches (ML, Heuristics, inference, Statistics and probabilities) there are four core items which are highly relevant to *imitation learning* and NLU:

- Decomposition into parts [Martins et al., 2011; Koller, 2014].
- Higher-order graph parsing [Zhang and Zhao, 2015].
- Structured inference and graph factoring [Martins et al., 2010].
- Distance-based metrics [Tsarfaty et al., 2012b] dealing with segmentation based on distance.

Decomposition in NLU usually refers to decomposing a MRL/KR into parts from which the model can be trained, or from which learning can occur. There have been proposed various techniques and approaches, such as *dual decomposition* [Martins et al., 2011] which transforms combinatorial problems into convex hulls [Martins et al., 2013].

Graph parsing is one of the two major approaches in NLU/dependency parsing, the other being *transition-based parsing.* In transition-based sentences are parsed using *shift-reduce* sequences [Sagae, 2009; Sagae and Lavie, 2006]; in graph-based parsing, tree graphs are decomposed into factors [Zhang and Zhao, 2015; Nivre, 2003; Yamada and Matsumoto, 2003], and recreated as tree graphs (MRL or KR). Dynamic programming has been used in combination with graph-based parsing [McDonald and Pereira, 2006; McDonald et al., 2005a] where the criticism was that inference in those approaches suffered from *sparsity* [Zhang and Zhao, 2015], in which case approximation via neural networks was considered preferable. *Distributed representation* (or encoding) of the information via feature vectors or *word* *embeddings*, has been widely used to process sentences via neural networks [Morin and Bengio, 2005] and deep learning [Collobert, 2011].

Graph factoring and structured inference is the process by which a *hidden* structure of a graph is inferred [Martins et al., 2010], with multiple choices and approaches possible, such as the *variational representations*, approximating inferences, and others. Distance metric as proposed and described [Tsarfaty et al., 2012b] deals with the distance in tree graphs, and how it may be used for more accurate parsing. Sub-problem solving [Martins et al., 2013] is the task of solving smaller problems, rather than treating a sentence as a big problem.

A major component in parsing is *shift-reduce* [Sagae, 2009; Shieber, 1983] which is deterministic, less accurate but usually a lot faster than statistic-based approaches. A probabilistic-based shift-reduce [Sagae and Lavie, 2006] has also been implemented, attempting to address accuracy. The shift reduce algorithm is an operation which moves forward onto the text or graph, without reversing; at any moment in time it creates trees or sub-trees of the graph. It may be using rules, probabilities, *precedence* or other control methods and may be used in combination with *look-ahead*, e.g., examination of future items, before making a decision at the current item.

In comparison to shift-reduce, a *beam search* [Zhang and Clark, 2008; Lafferty et al., 2001] is the exploration of a graph by expanding the most *probable* nodes and edges; it may be based on *best-first* search [Korf, 1993] and it can be Heuristic. Normally the parser stores a fixed amount β of best states as it explores the graph, and it may be used in combination with *shift-reduce* [Zhu et al., 2013].

2.7.8 NLU Performance and Issues

One of the most severe issues in parsing and NLU is *ambiguity* [Gorrell, 2006; Trueswell et al., 1994]. Ambiguity appears to be mostly encountered in syntactic parsing, due to the fact that categories or taxonomies may contain words which contradict other samples from the training data, and is not a phenomenon observed only in NLU; human development also learns to deal with ambiguity [MacDonald et al., 1994].

Other issues relate to the higher order of the sparse information encoded in a graph; e.g., the structure, the representational complexity which when reduced for parsing becomes more problematic at learning and mapping [Zhang and Zhao, 2015].

The performance of NLU is *often* measured using the F_1 score [Brodersen et al., 2010], although different variations of the formula shown in (2.34) do exist.

$$F_1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{2.34}$$

Actual F_1 scores reported have steadily increased the past 5 years; whereas at the end of the previous decade the *state-of-the-art* ranged within 80% to 90%, nowadays the scores range above 90% with SyntaxNet having achieved some of the highest scores [Andor et al., 2016], 94.44% on news data, 95.40% on question-answers, and 90.17% on web data as mentioned earlier in Section 2.7.6. NLU accuracy is approaching POS tagging accuracy: Google's *Parsey McParseface* achieved 97.77% on news data [Andor et al., 2016], whereas the laPOS tagger [Tsuruoka et al., 2011] was at 97.22%. There exists a correlation between NLU and POS tagging accuracy; since NLU uses the syntactic attributes (syntactic parsing) it is plausible that a propagation of errors from POS results in lower NLU scores. We're also observing a saturation of scores: more and more research work is pushing the scores upwards but the increments are becoming smaller, it is thus possible that the field has reached a plateau or a saturation point and there will be very little increase in accuracy in the near future.

Other metrics are also relevant, such as parsing speed and training time, Choi et al [Choi et al., 2015] offer a highly analytical review of the current tools and provide plethora of information. Perhaps the most undervalued and simultaneously most avoided topic in NLU is *sentence length*. It is reasonable that sentence length would affect both performance and accuracy, but Choi et al demonstrate that all parsers perform well with sentences under 10 words long, but accuracy declines for all parsers when given sentences with large word count. Whilst the group of examined parsers achieved F_1 ranging from 93.43% to 95.50% for less than 10 word sentences, for sentences with 20 words that range is 90% to 93%, for sentences with 30 words it is 87% to 92%, and for 50 words it is 81.66% to 86.61% [Choi et al., 2015, 392].

2.8 Background Conclusion

In this Chapter I have described the history and related research to my work. Furthermore, a logic continuation has been established with related sub-fields of AI: Neural Networks, Deep Learning, Reinforcement Learning, Semantic, Syntactic and Dependency parsing, Semantics and Cognitive AI.

The work presented hereinafter is both inter and intra-disciplinary. I have used research contributions from the human child development, from imitation in nature, as well as from across various AI fields and topics. Although such an approach comes with a certain risk, it was a necessity in order to address and recreate certain attributes occurring in nature and humans and then use various AI models forming pieces of a puzzle which simulates an imitating mechanism.

Other fields of AI which have not been described, such as Robotics or Artificial Life, are remotely related yet outside the scope of this thesis; whilst they may be related to imitation they deal with either physical agents (robots) which are focusing on sensorimotor imitation (learning of physical sequences) and Artificial Life focuses usually on the evolutionary processes which could enable imitation. As such, not much background research has been carried out in those fields.

The larger picture described in this Chapter follows a specific route, from human Psychological development and epigenetic traits, to programming by example, which is implemented using a cognitive agent through artificial neural networks and reinforcement learning. The outcome is a cognitive agent which performs NLU: Semantic, Syntactic and Dependency parsing. It should be noted that the field of NLU experienced tremendous progress whilst this PhD was underway; from 2012 to 2016, dependency parsing explored the use of deep learning by corporations such as Google, Yahoo, IBM and Microsoft, whereas performance of those systems steadily increased.

However, the reader should not confuse the work described here as research on NLU; it is research focusing on *artificial mechanisms which enable imitation*; it was a conscious choice to implement and compare an agent doing NLU parsing, because at the time it was a field considered extremely difficult and had the potential to showcase the real applications of imitating cognitive agents.

In the following Chapter the theoretical basis and the actual design of the Icarus agent and how each of the models, algorithms and technologies mentioned in this Chapter are being used, are described in detail.

Chapter 3

Theory and Agent Design

In this Chapter, I present the theoretical foundations upon which the Icarus engine was designed, and how each decision relates to either PBE, NLU, CA or human developmental psychology and imitation. Furthermore, I describe in detail every mechanism at its formulaic level and then refer to previous literature. Finally, I show how each piece of the puzzle fits together, and what algorithms enable those pieces to function in tandem. The Figure 3.1 shows the overall *agent design* in a process-flow, its components and processes, and how they are pieced together to enable the cognitive functionality.

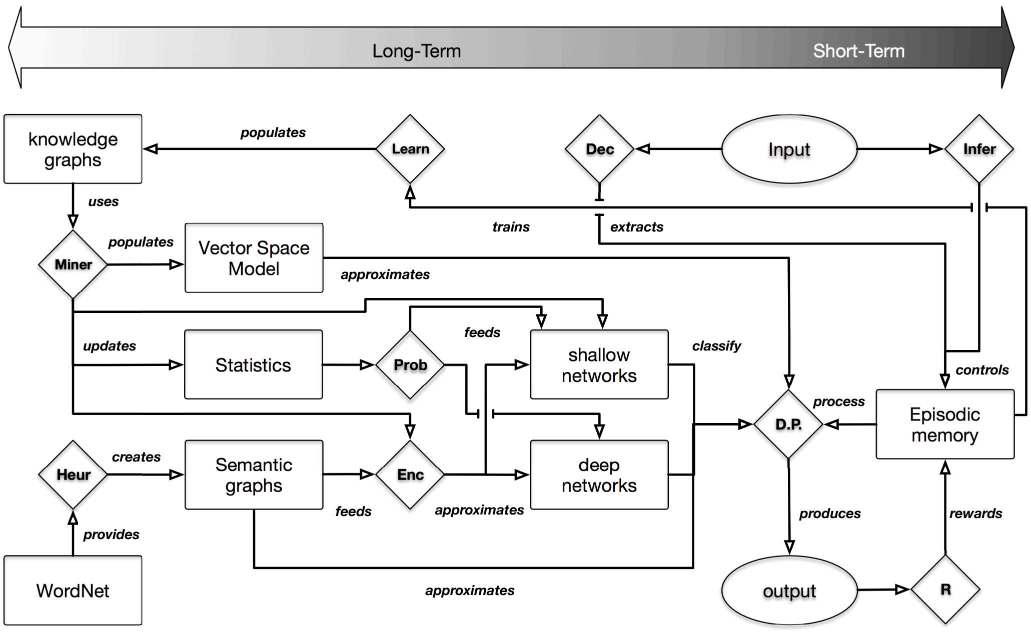


Figure 3.1: Icarus Engine Blueprint.

The left side of the figure demonstrates the *long-term* modules and processes whereas the right side of the figure contains the *short-term* modules and processes. Most of the modules, memory and processes deal with *long term* functionality, and only the entities operating on the input and output are residing in the *short term* partition of the agent. The short-term entities deal with *episodic memory* and the algorithms associated with it: the decomposition ("*Dec*"), inference ("*Infer*"), rewarding ("*R*") and decision processes ("*D.P*").

The learning process is shown as ("*Learn*") in Figure 3.1 and relies on episodic memory from which graphs are obtained; it uses the Q-learning update rule (see equation 2.26) and data mines them ("*Miner*") in order to extract samples. The mining process also encodes samples for the deep and shallow networks which are then used to classify actions. From the samples extracted the Statistics memory is updated, which in turn uses probabilities ("*Prob*") which are then given as input to neural networks. The mining process also populates the VSM with input patterns in order to enable approximation via attribute frequency. The agent also uses WordNet from which it heuristically ("*Heur*") extracts semantic graphs, those graphs are encoded ("*Enc*") for the neural networks, or used directly by the decision process ("*D.P*"). Last but not least, the neural networks are used to *classify* or *predict* actions for the ("*D.P*").

The core of the Icarus engine uses *reinforcement learning* which relies on MDP and other sub-processes; each one analysed in the Sections hereafter. The overall trend is to move *knowledge* acquired from information using *qualia*, e.g., from short-term to long-term memory, and then *re-use* that knowledge as information, in order to enable the *short-term* processes perform better and more accurately.

3.1 MDP as a Template for Learning

As already mentioned in Chapter 2, an MDP (a *Markov decision process*) is a chain of states linked together by actions. I use this fundamental structure as the *backbone* of the Icarus engine; the *qualia* (subjective experience) of the agent are stored using MDP. The temporal-sequential process used by Icarus serves as the short-term memory experiencing a decoded paradigm. It is implemented as an *episode* using Q-learning [Sutton and Barto, 1998, 148]. Decoding, learning and experience indexing are implemented using a MDP; the reasons being numerous:

- Development in human children is partially based on behaviourism and reinforcement [Skinner, 2014], adding the bonus of artificial biomimicking.

- An MDP episode allows both *internal* as well as *external* rewarding; in turn this enables the agent to both self-control reinforcement via observation or accept 3rd party rewarding.

- Qualia [Haikonen, 2009] the subjective experience, maps observations, meaning, knowledge via a *Markovian* state. It functions exactly as Haikonen described it by *encapsulating* perception through the agent's point of view.

- Meaning, representation, relation to information are also mapped to a *state* [Haikonen, 2007], thus enabling *perception* and recognition in a temporal-dynamic manner.

- Every criteria as described by Bandura [Bandura, 1986] is satisfied (discussed in Section 2.2.2). The agent through perception and imitation produces a model via the MDP, the process retains the symbolic concept, the internal models may provide responses, and reinforcement serves as the motivation to re-use models.

- PBE has already established the use of MDP even partially observable as a process which can learn from the demonstrator [McDaniel, 2001]. We expand on that notion and showcase that partial observability and hidden states can be infered accurately.

- Enactive representation becomes possible [Bruner, 2009] via both passive observation and external feedback.

- Google has demonstrated [Mnih et al., 2015] through deep RL/DQN that human-level performance is possible when combining RL with Deep Learning. Albeit this was not my original motivation, it supports the adoption of RL as the template mechanism.

3.2 Paradigm Decomposition and Training

The basic functionality of the high-level imitation mechanism is shown in Figures 3.2 and 3.3. In the first one the agent is *acquiring* or *observing* the example, and in the second the agent is recreating the example. This process forms the *qualia* as it is a subjective experience, constraint by the agent's long-term knowledge and perceptive abilities.

The *observation-recreation* via paradigm forms the training phase; a form of learning using *inferred* episodes from the observed example. This must be noted:

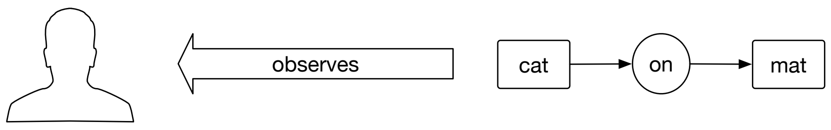


Figure 3.2: Agent Observes Example

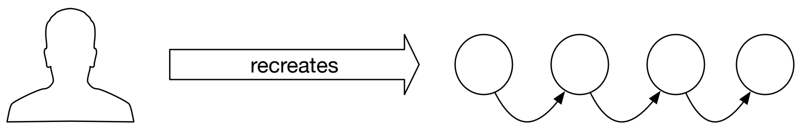


Figure 3.3: Agent Recreates Example

the actual example provided by the demonstrator may differ from the one the agent internally perceives, that is the key factor making the experience *subjective.*

As per the PEB and NLU literature, the example must be analysed, decomposed and factored into the constituent primitives of the domain (in this case, words or *terms* and their relations). The process of decomposition via inference has been explored in both PBE and NLU and is a logic-heuristic sub-process; it may be *taught* or it may be genetically enabled such as primitive decomposition in humans [Pirri, 2005; Biederman and Gerhardstein, 1993]. For my intents and purposes I did not invest in evolutionary computation to evolve a program that performs decomposition, so that it would have been evolved similar to the imitation mechanism in nature. Because decomposition may not rely solely on behavioural development we opted to heuristically program it; arguably a more elaborate approach would have been to *learn* to tokenize and decompose in an unsupervised manner [Wrenn et al., 2007], however such an approach would induce and propagate additional errors from tokenization to part-of-speech tagging and parsing. In Icarus engine, the decomposition process is shown in Figure 3.4 for the example phrase "*cat on mat*".

The current decomposition in Icarus is a *white-space tokenization* of the sentence [Jurafsky and Martin, 2000] and a decomposition of the observed graph G into its primitive constituents: nodes and edges. The observed graph is analysed into three sets: *concepts* C, *relations* R and *edges* E.

$$G_t = \begin{cases} C_t = \Big\{Cat, Mat\Big\} \\ R_t = \Big\{On\Big\} \\ E_t = \Big\{[Cat, On], [On, Mat]\Big\} \end{cases} . \tag{3.1}$$

We use the subscript/index t which is a time-step; because a state s_t is

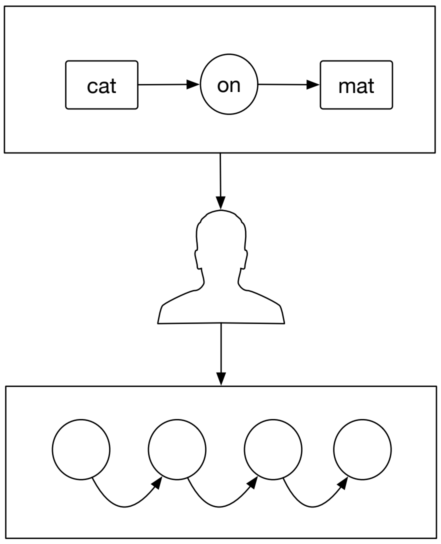


Figure 3.4: Agent Decomposes Example

described by a graph G, each graph instance within the episode is also indexed by t. The graph G_t is a *bipartite directed graph* [Sowa, 1999] $G = (N, E)$ where N are the nodes and E are the edges. I denote the graph as $G_t = (C_t, R_t, E_t)$ and therefore by extension the state $s_t = (C_t, R_t, E_t)$.

The order of the node pairs forming an edge e defines the *direction*, e.g., $e = [Cat, On]$ is different from edge $e = [On, Cat]$. The order of the edges in the Edge set is equally important; in this example the edge set order must be $E = \Big\{[Cat, On], [On, Mat]\Big\}$ and **not** $E = \Big\{[On, Mat], [Cat, On]\Big\}$. In comparison, the node sets are *unordered sets* and I do not attribute importance to their order of appearance, e.g., $C = \Big\{Cat, Mat\Big\}$ and $C = \Big\{Mat, Cat\Big\}$ are the same.

There are other benefits in using a *Markovian* state and the MDP as the foundations of a cognitive agent. The most important aspect is the fact that I may combine *Symbolism* such as the KR with *ML* seamlessly in a temporal manner. The manner in which we combine a symbolic concept, in this case a KR and in specific a conceptual graph (CG) is best demonstrated in Figure 3.5 where the phrase "*cat on mat*" is being parsed.

The process of *populating* each subsequent Markovian state s_t is done via *shift-reduce* [Zhu et al., 2013]. A list of tokenised words from the input sentence is reduced by the agent who is *deciding* if each word (hereinafter referred to as a *term*) is a *concept* or *relation* as per the conceptual graph literature [Sowa, 1999], thereby *populating* a graph with nodes.

Similarly, the process of creating *edges* (or arcs) between the graph nodes is

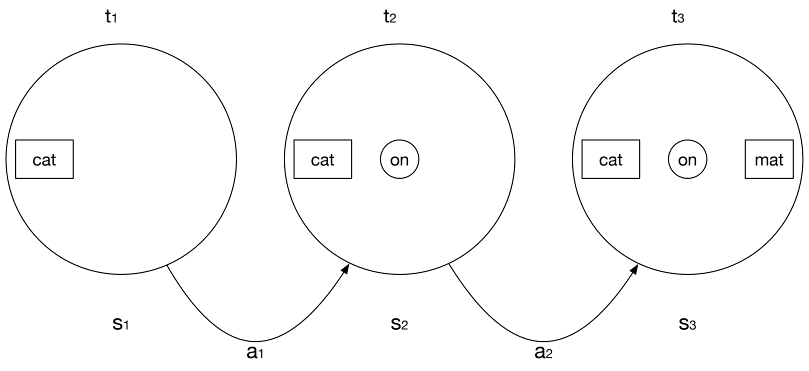


Figure 3.5: Shift-Reduce Graph Polulation

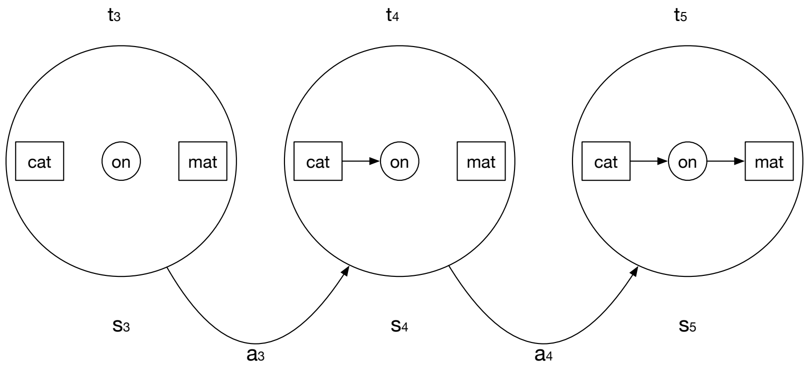


Figure 3.6: Shift-Reduce Edge Creation

a graph manipulation, whereby the agent *decides* which nodes in the graph should connect with each-other, shown in Figure 3.6. Both Figures 3.5 and 3.6 form an *episode*, and the *terminal* (last) state of that episode *contains* the product of that process. All previous graphs in various time-steps t are ignored as short-term memory manipulations and although stored they only serve the purpose of *subjective experience*. Long-term memory uses the terminal state's graph, the final graph which is stored in the *knowledge base*.

Therefore the MDP encapsulates a complex *shift-reduce* episode where the actions a_t manipulate the state's s_t graph G_t. In the event where the agent is being *trained*, the actions a_t are inferred by having reduced the graph into its primitive components; each component constitutes an action. Only during *testing* does the agent *evaluate* and *re-use* its experience and knowledge where actions must be decided by *approximation* and *classification* rather than inference. The following differences are what make this agent an *imitating* entity:

- The agent learns by observing examples provided by human users. Thereby, the agent requires *decomposition* of the paradigms, into a *qualia* MDP episode.

- The agent may *explore* graph creation trees similar to a *beam search.* In this scenario, the agent can explore diverging states and actions and end-up at a variable amount of terminal states.

- The agent rewards itself after training by observing if the re-created graph of an episode is the same as the example given. Thus the imitation is in fact *observation* and the agent must *infer hidden* or *partially observable* states.

- The agent, after being rewarded can follow existing policies, or approximate them. Reuse of experience and knowledge enables the agent to satisfy the *cognitive agent* criteria.

3.3 Rewarding and Evaluation

After the agent has re-created an MDP episode by decomposing an example, it examines if the output graph is identical or isomorphic to the one that was provided as the example. If those two graphs are indeed identical or isomorphic then the agent proceeds to self-reward itself positively: it updates the generated episode by reinforcing it with a reward $r_t = r(s_t, a_t) = 1$. It then proceeds to update the episode's policies using the Q-Learning update equation (2.26). Should the reproduced output graph not match the input example graph, then the implication would be that the decomposition process failed.

Contrary to the *training* procedure, evaluation *tests* the agent by providing a *sentence input* only and expecting a graph output which is isomorphic to the *correct* graph[1]. The methodological approach assigns a correct graph to each sentence, yet during evaluation the graph is not provided to the agent, until the agent has produced an output graph.

This evaluation process is a non-binary classification task: the agent is given an input pattern (sentence) p_i and produces an output graph $G_i^{\hat{y}}$ representing input p_i, with i indexing the pattern and respective graph. The assigned and assumed correct graph G_i^y is compared to the produced $G_i^{\hat{y}}$ and thus it is possible to quantify similarity between conceptual graphs.

The difference of the two graphs $\Delta_{G_i} = G_i^y - G_i^{\hat{y}}$ is used as a *precision* score; the agent however infers its own performance in *Boolean* terms and rewards itself

[1]Correct as ascertained and described by the trainer or demonstrator.

with a 1 or -1 depending on the outcome, as shown in the pseudo-code in Algorithm 1.

Algorithm 1: Self-Rewarding Behaviour.

```
   Input: $G_i^{\hat{y}}, G_i^{y}$
   Output: $Boolean$
1  $\Delta_{G_i} \leftarrow G_i^{y} - G_i^{\hat{y}}$ ;
2  if $\Delta_{G_i} = 0$ then
3      return $True$ ;
4  else if $\Delta_{G_i} \neq 0$ then
5      return $False$ ;
```

3.4 Episode Iteration and Inference

Creating an episode and then iterating it requires the use of *deterministically* infering the *next state* s_{t+1} when taking an action a_t for state s_t. Furthermore in this scenario the agent is being *evaluated*; should it have experienced this episode before it would already have been taught *which* policies to follow. Because the episode is new the agent has to find *similar* state(s) and action(s) and approximate which is the best; we refer to this function as the *decision-making* process. If the agent *decides* that no further actions are possible or should not be carried out, then no new states can be created, and thus the episode has ended.

Algorithm 2: Agent Episode Iteration.

```
   Input: $p_i$
   Output: $G_t^{\hat{y}}(p_i)$
1  $s_{t=0} \leftarrow Root(p_i)$ ;
2  while $\exists(a_t) \leftarrow Decide\Big(\underset{a}{max} Q(s_t, a)\Big)$ do
3      $s_{t+1} \leftarrow Infer(s_t, a_t)$ ;
4      $s_t \leftarrow s_{t+1}$ ;
5      $t \leftarrow t + 1$ ;
6  return $G_t$ ;
```

The episode *iteration* and *graph* generation process is shown in pseudo-code in Algorithm 2. In this case the agent keeps generating *states* after deciding on an action a_t for the current state s_t. It does not diverge into a *beam search*[Zhang and Clark, 2008] or following multiple graph exploration, but instead tries to maximise the best possible policy $\underset{a}{max}Q(s_t, a)$. Whereas the *result* of the agent's action a_t leading to s_{t+1} is deterministic, the decision to take an action *may not be*; that is one of the advantages of combining RL with other models: the manner in which decisions are made may be heuristic, semantic, statistics-based, machine learning-based, etc.

3.5 Decision Making and Policy Approximation

The part of the agent that performs *action-selection* is a process of the agent which can be implemented using different algorithms. Its sole purpose is to *estimate* or *approximate* the best policy $\underset{a}{max}Q(s_t, a)$ when the episode is new and has not been experienced before.

In cognitive agents and autonomous systems alike there is a need to *re-use* experience and background knowledge, whether it is *qualia* or information acquired through trial-and-error. Basing future action-selection on previous experience is crucial to autonomous agents [Lawniczak and Di Stefano, 2010; Haikonen, 2009, 2007].

For our intents and purposes we assert that approximation is used during exploration; during that phase the agent is processing unknown or partially unknown input and re-use of previous knowledge and experience is highly desirable. Exploring the entire state and action space is avoidable; previous Q-policies of *similar* episodes can indicate *future* actions, ergo the agent may infer those actions using a variety of models and algorithms.

Similar to Google's DQN and their combination of RL and CNN, the Icarus engine uses a sub-routine approach in addition to reinforcement learning. The core notion behind the decision-making sub-routine is as follows:

- A new episode is generated (if not previously experienced).
- States are created (as described in Section 3.1 using *shift-reduce*).
- Actions are approximated (if no known policies exist).

Therefore the *action creation* process is differentiated from the *state creation* MDP. Whereas the states s_t are manipulated on a symbolic level, actions are manipulated using Heuristics, probabilities, Statistics and ML. This scheme is depicted in Figure 3.7.

The relationships described in Figure 3.7 are:

- State s_t is given as input to the *decision making* mechanism.

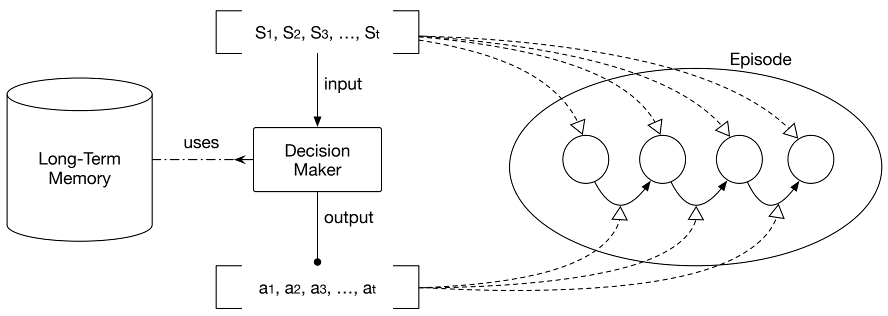


Figure 3.7: Agent as a Decision Maker.

- Decision making relies/uses part of the *long-term* memory.

- Actions are the output.

- Actions are indexed and associated with the respective state s_t in the episode.

The *long term* memory (also shown in Figure 3.7) consists of *statistical* information and *probabilities* obtained and inferred from observations based on the training samples. It also uses Semantic graphs to establish relations between *terms* and *nodes*, as well as a VSM to find similarity between episode sentences. Last but not least, the long term memory is made up of multiple deep networks used to control the decision-making process. The agent is able to analyse the actions taken during training and associate or index frequency of node conversions and edge creations.

Differentiating between *short-term* and *long-term* memory is described in both the Icarus architecture [Langley et al., 2009] as well by Haikonen [Haikonen, 2003]. Arguably modern computers are not *constrained* by memory as it has become cheap and abundant, therefore that differentiation is mostly done for *optimisation* reasons (e.g., faster lookup and indexing). For our intents and purposes, *short-term* is considered the MDP and the related episodes learnt or created, whereas Statistics, probabilities, Semantics, neural networks and such are all part of the *long-term* memory, as shown in Figure 3.8.

In the Icarus blueprint (Figure 3.1) a greater amount of detail was shown, whereas in Figure 3.8 the components related to the decision making mechanism are the only ones shown. The components involved within the decision making processes, are the Semantic, Statistics, Vector Space and Neural (associative) memory. Each one serves a distinct purpose, and is accompanied by a group of algorithms responsible for populating, updating, training, manipulating and propagating information and meta-data from the *long-term* towards the *short-term* entities and processes.

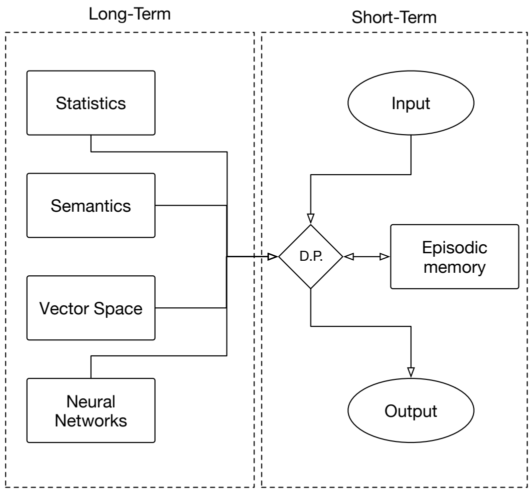


Figure 3.8: Decision Making Process.

Choosing to approximate the *best* candidate state at each moment in time t, transforms the *decision making process* into a *predictor*, a *classifier* or an *approximator*. Actions are generated by the agent without any external input; the agent controls what it learns, we only parametrise the learning mechanisms, this has as a result a faster parsing time and an autonomous agent.

Algorithm 3: Non-Greedy Behaviour.

```
    Input: $s_t$
    Output: $a_t$
 1  $Q(s_t, a_t) \leftarrow Find\Big(\max_a Q(s_t, a)\Big)$ ;
 2  if $\nexists\Big(Q(s_t, a_t)\Big)$ then
 3      $Q(s_t, a_t) \sim f\Big(\max_a Q(s_t, a)\Big)$ ;
 4      return $a_t$ ;
 5  else if $\exists\Big(Q(s_t, a_t)\Big)$ then
 6      if $Q(s_t, a_t) > 0$ then
 7          return $a_t$ ;
 8      else if $Q(s_t, a_t) \leq 0$ then
 9          $Q(s_t, a_t) \sim f\Big(\max_a Q(s_t, a)\Big)$ ;
10          return $a_t$ ;
```

The pseudo-code in Algorithm 3 demonstrates how the agent iterates and performs approximation; the term *non-greedy* implies that the agent will not greedily decide to ignore policies, but as shown in Algorithm 3 (line #2 to line #4) if there doesn't exist a policy it will opt to use an *approximation* function f. That function addresses the issue of *predicting* or *approximating* the most suitable action a_t based on the agent's previous experience; the state s_t or policies Q are unknown. Whilst the exact nature of the approximation function f is discussed in later Sections, it is important to note that the agent may also use the same approach if it knows that the policy values are *negative* (shown in Algorithm 3, lines #8 and #9).

3.6 Statistical and Probabilistic Approximation

A large portion of the agent's memory indexes *observations* made on *meta-data* mined from its knowledge base. Those observations are based on both the *term* label as well as the term's POS tag (e.g., Semantic and Syntactic observations). The probabilities are acquired by observing how frequently an action is taken and if an action is possible or not. We define two probabilities: $p(e)$ for edges and $p(n)$ for nodes. In both cases the same notion is implied, for $p(n)$ the probability of a *term* being a *concept* or *relation* node, and for $p(e)$ the probability of two nodes (concept to relation, or vice versa) being connected by an edge. The $p(n)$ is binomial; either a *term* is a concept or a relation. However the probability $p(e)$ relies on observations of frequency and presence: (a) the query of whether two nodes have *ever been observed* to be connected by an edge, and (b) how *frequently* is a specific edge observed. The *realisation* of $p(e)$ is shown in (3.2).

$$p(e) = \frac{E(n_i n_k)}{\sum n_i n_k}. \tag{3.2}$$

In formula (3.2) $n_i n_k$ defines an edge from *node* i to *node* k whilst taking into account the node types (concept/relation). Therefore $\sum n_i n_k$ is the sum of all the observations when such an edge *could have been created*, e.g., the conditions to create the edge were met: both nodes n_i and n_k were present in the graph. On the contrary $E(n_i n_k)$ is the frequency of observations when this edge *was indeed created* when the conditions were met. The implication of the aforementioned notion is that nodes may exist but edges do not always connect them.

Those probabilities further observe peculiar aspects of the nodes: (a) their POS tag, (b) the *term*, label or word, and (c) the distance between the nodes inside the sentence. The probability of a pair of nodes having an edge based on the *term* is a Semantic approach; it implies that the probability is calculated based on the *meaning* of the actual nodes, hence we define it as $p(e_{term}) = E(t_i t_k)$, where *term* is a word, label, symbol or sign. Similarly, when the probability is based on a POS tag it is in fact a *Syntactic* approach; ergo the probability is based on syntax alone hence $p(e_{tag}) = E(t_i t_k)$.

The last type of probability used depends on *distance*: we obtain the metric distance between two *terms* inside the input sentence (p_i) and using their respective position the Δ measures how *far apart* they are. Calculating the probability of an edge based on the distance of the *terms* t_i and t_k respectively labelling the nodes n_i and n_k, has a realisation similar to equation (3.2) as shown in equation (3.3) with $\Delta_{t_i t_k} = t_k - t_i$ defining the distance between the terms.

$$p(e) = \frac{E\Big(\Delta_{t_i t_k}\Big)}{\sum \Delta_{t_i t_k}}. \tag{3.3}$$

Therefore the decision making on *action creation* can use a *probabilistic approach*; a large look-up table is computed after the agent is trained. That probability look-up table enables a *frequency/Statistic* approximation based on prior observations; the observations being Semantic, Syntactic and distance metrics.

3.7 Semantic and Heuristic Approximation

As discussed in Chapter 2 and in Sections 2.7.2 - 2.7.3, Semantics have been used for approximation: (a) finding similar episodes based on a VSM, and (b) finding similar *terms* using WordNet [Fellbaum, 1998]. Indexing similar episodes uses the VSM as described by the matrix in formula (2.31) and formulae (2.32) and (2.33). The purpose of that VSM is to find *highly similar* episodes which *should be* considered for examination, thereby cutting down search time.

The *consideration* process relies on Semantic relations (discussed in Section 2.7.3) and iterates in a *breadth-first* search [Lee, 1961] each Semantic tree-graph layer accumulating the distance between graph layers. The iteration process aims to discover that *distance* within the Semantic graph or graphs if *intersecting* them, thus allowing the agent to quantify Semantic similarity between two terms t_i and t_k.

The *likelihood* that a WordNet query will contain both terms is *anecdotal and empirically small*, hence a solution is to use the *union* of two WordNet graphs, each one produced by querying the respective *terms/words.* In Figure 3.9 two graphs are queried, one for t_i and one for t_k.

Both terms have a common *super-class* node t_r and thus their union creates a new Semantic graph demonstrating how those two tree graphs are related. Iterating

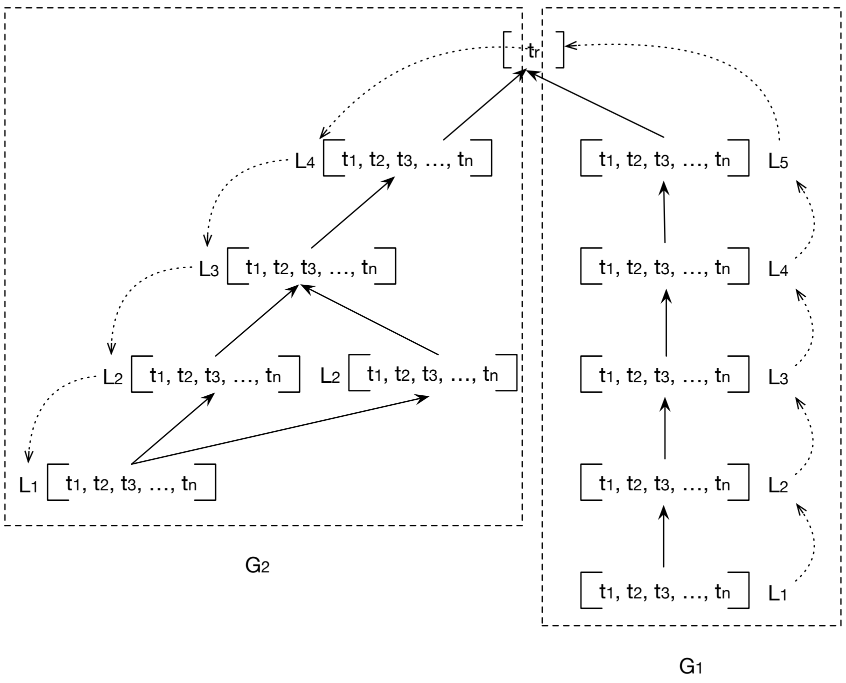


Figure 3.9: Semantic Graph Union.

upwards from graph G_1 and then downwards towards graph G_2 in what is shown as the dotted arrows, is the *Semantic distance* metric $v_{t_i t_k}$ shown in (3.4).

$$v_{t_i t_k} = v\langle L_1, L_2, \cdots, L_k \rangle = \sum \vec{L_i L_k}. \tag{3.4}$$

From equation (3.4) the vector/path $v_{t_i t_k} = v\langle \vec{t_i, t_k} \rangle$ originating from term t_i which is inside layer L_i and is directed towards L_k which contains destination term t_k; therefore the *size* of vector/path $v_{t_i t_k}$ defines how far apart those two terms are inside the graph union. The iteration search accumulates *layers* traversed (denoted with L) with $O(|N| + |E|)$ complexity [Lee, 1961]. The terms are searched inside one of the *hypernym* layers since that is the way in which WordNet organises the Semantic graphs; the principle upon which this Heuristic search is based is *information relation* and *Semantic relation* [Agirre et al., 2009; Lin, 1998; Jiang and Conrath, 1997; Rada et al., 1989].

Because WordNet returns *senses* ordered from *most frequent* to *least frequent* Semantic graphs, the use of the sense index allocates a probabilistic importance as to *which* Semantic relation path is most likely to be valid, as shown in (3.5). The actual *sense* is a Semantic graph out of an ordered set of Semantic graphs, therefore $s_i(v_{t_i t_k})$ is the sense graph indexed by the set subscript i.

$$\hat{v}_{t_i t_k} = \frac{s_i(v_{t_i t_k}) - min}{max(s_i(v_{t_i t_k})) - min} \cdot v_{t_i t_k}. \tag{3.5}$$

The scaling uses the index of the first Semantic graph as a *bias* or *weight* thereby resulting in a normalised and scaled measure which takes into account both sense order and graph union vector/path distance. The minimum values are indexed by setting *max* the size of the sense set, and *min* as the last sense graph, thereby biasing towards the first graphs which appear *more frequently* as Semantic graphs [Fellbaum, 1998].

The above approach enables the use of Semantic relations in a *taxonomic* method via Heuristics, so that *terms* in input sentence/patterns may be *swapped.* The proposition as in most Semantic/Heuristic algorithms was that by discovering similar episodes using a VSM and Semantic relations, the agent would be able to infer similar $Q(s_t, a_t)$ policies, based on substitution of *highly similar terms.* These algorithms are described in detail in Chapter 5, as is their accuracy and performance.

3.8 Neural Approximation and Distributed Encoding

Using raw values from probabilities or Semantic similarity requires some kind of filtering; simply put the agent *does not* a-priori know which values approximate an action. In this scenario the probability values $p(e)$ for edges (see 3.2) and distance (see 3.3) are *given as input* to a shallow neural network, a network with one hidden layer (Figure 2.6) two or three input nodes and two output nodes. The input nodes use $p(e_{term})$ based on term probability, $p(e_{pos})$ based on POS tag and the *normalised* and *scaled* distance (3.6). The actual distance is *normalised* using *min-max* normalisation, as shown in (3.6). The $max(\Delta)$ is the maximum possible distance dictated by the *size* of the sentence/pattern p_i, whereas *min* is the minimum distance between two terms (always set to 1 since term self-distance is zero).

$$\hat{\Delta}_{t_i t_k} = \frac{(k - i) - min}{max(\Delta) - min}. \tag{3.6}$$

The *miner* process iterates the knowledge base e.g., the correct conceptual graphs, and examines valid edges based on the aforementioned observations; thus it generates *samples* which associate a probability to an edge action $p(e) \leftrightarrow a_t$ to an network output y_i. The network input is thus a vector I of those values for sample i, as shown in (3.7).

$$I_i = \Big[p(e_{term}), p(e_{tag}), \hat{\Delta}_{t_i t_k}\Big]. \tag{3.7}$$

Using a shallow network has certain advantages: it is a simpler, smaller network, fast to train. For the agent's intents and purposes it is trained once and evaluated multiple times; yet in realistic scenarios the user decides on which network to use. An automated procedure can train and cross-validate the shallow network multiple times, before selecting on which one to use. The shallow networks used are two: (a) one which uses only $p(e_{tag})$ and $\hat{\Delta}_{t_i t_k}$ because the *terms* or one of the edge *terms* are *unknown* e.g., never encountered before, and (b) one which uses all three values as shown in (3.7). This form of encoding is *distributed* and non-categorical [Picton, 1994]; the input values do not represent the presence or absence of a feature, but distribute the probability value of an edge based on posterior observations.

The agent internally trains and uses those networks for the decision making process. During evaluation all candidate edges for a state s_t are examined and the networks filter each possible one at a time, resulting in a list of possible actions.

Algorithm 4: Neural-based Action Selection.

```
    Input: $s_t$
    Output: $a_t$
    Data: $candidates = []$
1   for $n_i \in concepts(s_t)$ do
2       for $n_k \in relations(s_t)$ do
3           $candidates[] = E(n_i n_k)$;
4   for $n_i \in relations(s_t)$ do
5       for $n_k \in concepts(s_t)$ do
6           $candidates[] = E(n_i n_k)$;
7   for $E_i(n_i n_k) \in candidates$ do
8       $E_i = n_i n_k$;
9       $p(e_i^{term}) = P(n_i n_k)$;
10      $p(e_i^{tag}) = P(n_i n_k)$;
11      $\hat{\Delta}_{t_i t_k} = normalise(i - k)$;
12      $I_i = \left[ p(e_{term}), p(e_{tag}), \hat{\Delta}_{t_i t_k} \right]$;
13      $\hat{y_i} = propagate(I_i)$;
14      if $\hat{y_i}[0] \geq \hat{y_i}[1]$ then
15          return $a_t \leftrightarrow E_i(n_i n_k)$;
```

What the pseudo-code in Algorithm 4 showcases is how the edge selection process (part of the *D.P.*) takes place; first all possible edge combinations are created (called *candidates*) and then the neural networks act as classifiers deciding on which actions to filter and which not. The result is a MDP using neural networks for the action selection; networks which are trained by the agent after it has acquired the examples from the user. The only actual external involvement is the hyper-paramaterisation of the networks. Therefore the neural network maps meta-data acquired after learning and from that mapping it learns to classify actions.

The reason why the shallow neural networks are used as filters is their ability to *approximate* and *classify* the given input probabilities as suitable and non-suitable. This approach is related to the meta-data obtained by the miner: it is non-linearly separable [Elizondo, 2006], and thus a multilayer neural network is an optimal model for indirectly detecting which input values to disregard and which ones to use.

3.9 Deep Neural Approximation and Sparse Encoding

Contrary to the shallow networks described in Section 2.4.1 and Section 3.8, the agent was also implemented using state of the art *deep feed-forward* neural networks. The background and characteristics of deep learning were discussed in Section 2.4.3, and hereinafter their use within the Icarus agent is analysed. Deep networks have been researched extensively in the past 6 years, and an overwhelming amount of publications suggest that they can be used for NLP, NLU and similar processes (see Section 2.4.3 for a thorough analysis and citations).

The deep networks in Icarus form a *cascade*, meaning that networks propagate their output to other networks. Whilst the outer networks perform an encoding and classification function, the succeeding inner networks learn to classify *likely-hood* output from previous networks.

The deep networks are capable of processing *probabilities* in a distributed-encoding scheme similar to how the shallow networks function (see Section 3.8) but have also been implemented by using a *sparse encoding* scheme, where a *feature vector* is extracted by each conceptual graph. That approach has become a standard in NLP and NLU, and as showcased by Google *word2vec* [Mikolov et al., 2013a] as well as the research in the human neurobiology [Olshausen and Field, 1997], human sensory processing [Olshausen and Field, 2004] and the human brain in general [Rolls and Treves, 1990], it is generally admitted that sparsity offers advantages, such as better pattern recognition and larger storage capacity. The field of deep learning was inspired from the visual cortex [Cadieu et al., 2014; Lee et al., 2008] where hidden layers are often more than 10 and some times up to 20, with millions of nodes and billions of weights. Linearly and sparsely encoded scheme has demonstrated that even shallow networks can perform a lot better [Montalto et al., 2015] instead of using distributed encoding.

The agent uses sparse binary feature vectors which are obtained by indexing all nodes (concepts and relations) into separate lexicons; those networks deal with classification of those node combinations and thus a network is used to classify concept to relation edges, and another to classify relation to concept edges, and yet another to classify POS tags to POS tags. The feature vector V is therefore as big as the lexicon is: V_t indexes the Boolean representation of a term t and is σ long where σ defines the magnitude/size of the vector. In order to encode both concepts and relations, $\sigma = \|C\| + \|R\|$ where $\|C\|$ is the cardinality of the set of concept nodes and respectively $\|R\|$ is the cardinality of the set of relation nodes. Because *order* of the terms defines the edge (see formula 3.1 and Section 3.2), inherently the order of the sets also defines the edge classifier; when V_t encodes using the sets $\|C\| + \|R\|$ this is a concept to relation edge, and vice versa when using the sets $\|R\| + \|C\|$ it is a relation to concept edge. A POS tag to POS tag classifier does not account for the Semantic meaning of the term t but only uses the tag set, therefore only one deep network is used and the size of the set of the PENN tags is multiplied by two. The cascade of deep networks is shown in Figure 3.10.

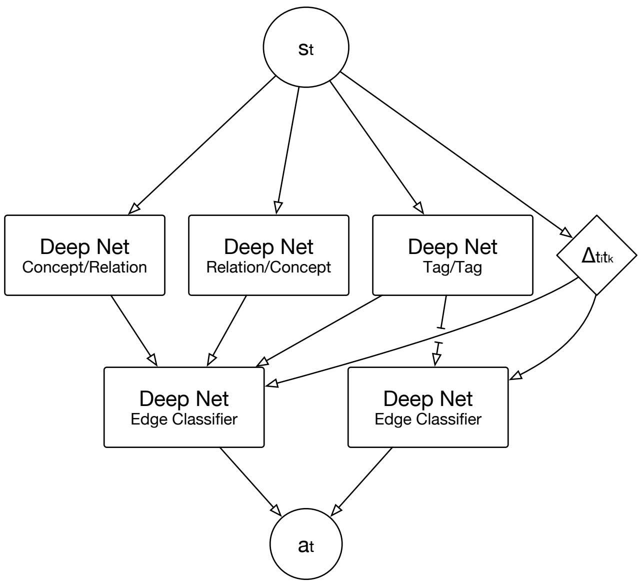


Figure 3.10: Deep Learning Cascade.

The advantages of using a cascade rather than a single network are the following:

- Any of the classifiers can be re-trained when the lexicon increases, without requiring to re-train the rest of the networks
- Each classifier can be further optimised or replaced, without significantly affecting the rest of them
- Outer classifiers learn to associate sparse encoded feature vectors with the likely-hood of an action, respective to a specific edge order (concept to relation, or relation to concept) ignoring distance and POS tags
- Inner classifiers are unaware of edge order or lexicon changes; instead they learn to classify network output as appropriate actions
- Inner classifiers process the normalised distance of nodes/terms irrespective of the lexicon size or edge order

Detailed analysis of how deep networks were implemented and how they performed, are discussed in Chapter 5.

3.10 Semantic Approximation and Sparse Encoding

One of the large problems in neural networks dealing with NLP is the fact that they need to be re-trained whenever the lexicon increases or changes. Accounting for unknown terms (words, tokens, symbols, signs, etc) is not a straightforward practice: either the network will not process unknown or new input, or it has to be re-trained in order to expand the input vector in order to accomondate for the newly indexed term or terms. A solution to this issue has been created; similar to how Heuristic algorithms work using WordNet and the theoretical basis described in Section 3.7, a Semantic distance metric kernel was developed in order to replace sparsely encoded terms within a feature vector.

Ordinarily and as described in Sections 2.7.2, 3.9 and shown in formulae (2.29) and (2.30), a feature vector is a vector of binary values. Reusing the notion of similarity and information [Agirre et al., 2009; Lin, 1998; Jiang and Conrath, 1997; Rada et al., 1989] via the formula (3.5) a similarity metric is obtained for an unknown to the network lexicon term t_i, and a known to WordNet term t_k. In order to squash the value of $\hat{v}_{t_i t_k}$ the largest paths from both Semantic graphs G_i and G_k are required. Using the maximum values $max(G_i)$ and $max(G_k)$ enables the squashing of the normalised and scaled value to be obtained, as shown in (3.8).

$$\tilde{v}_{t_i t_k} = \frac{\hat{v}_{t_i t_k} - min}{(max(G_i) + max(G_k)) - min}. \tag{3.8}$$

Whilst the scaled and normalised similarity metric $\hat{v}_{t_i t_k}$ accounts for *sense* bias, we transform the feature vector from a binary vector to a *real-valued* vector, with a value of 1 indicating absolute similarity between terms, and a zero value indicating no similarity at all, therefore the similarity value is min-max normalised and scaled in a range of zero to one so that $0 \geq v_{t_i t_k} \geq 1$ and $v_{t_i t_k} \in \mathbb{R}$. In order to convert the range of value so that it matches the directionality of non-similarity at 0, and full similarity at 1, we invert the squashed value from formula (3.8) as shown in (3.9).

$$\begin{aligned} v_{t_i t_k} &= 1 - \tilde{v}_{t_i t_k} \\ &= \{v_{t_i t_k} \in \mathbb{R} \| 0 \geq v_{t_i t_k} \geq 1\}. \end{aligned} \tag{3.9}$$

Thus the term swapping can be performed online after having trained a network; instead of providing as input to the network the binary feature vector, a Heuristic algorithm finds the most similar term t_k to the unknown term t_i and categorically represents it using the $v_{t_i t_k}$ in the deep network classifiers previously shown in Figure 3.10. Therefore this approach uses WordNet to alleviate the problem of dealing with unknown words without requiring the agent to train the deep network again.

3.11 Conceptual Graph Output

Throughout this thesis CG are used as the medium of KR because they are the most *visually* appealing; those graphs when demonstrated as a paradigm to the agent, are generated on a web-UI from a human user (a knowledge engineer) and as such one of the critical requirements was the use of a structure which was simple and easy to manipulate. Furthermore, CG are simplistic and minimal models without an excess of meta-data, such as RDF or OWL. Other advantages are the simple nature of representing relations through nodes rather than edges, their expressiveness which is similar to natural language and their accuracy and highly structural information [Rasli et al., 2014; Zhong et al., 2011]. Researchers state that conceptual graphs are intuitive and semantically sound means of knowledge representation [Croitoru et al., 2007]. Last but not least, conceptual graphs have been demonstrated to offer a computationally tractable and sound way of representing text and natural language [Montes-y Gómez et al., 2002].

3.12 Metalearning and Knowledge Compression

The Icarus agent was designed to do more than parsing and understanding of language onto KR. During the development my cooperation with Matthew Thorpe from the Warwick Mathematics Institute saw the development of a theoretical extension to Icarus which would allow it to abstract conceptual graphs, compress knowledge, offer summary of text and further enhance its performance. Unfortunately due to time constraints this module was never implemented and experimented with, although the blueprints were published [Gkiokas et al., 2014].

3.12.1 Metalearning on Learnt Knowledge

The hypothesis of metalearning in Icarus is that an agent after having acquired knowledge, can further manipulate that knowledge, alter it by compression, generalisation or abstraction (thus generating rules or extracting patterns) which may be more useful than the *specialisations*, e.g., the specific knowledge represented by a graph. This notion implies that an agent doesn't need to store all knowledge specialisations and instances, instead it can opt to group them by similarity, thereby saving space, decreasing time complexity (access, search and insertion) when accessing knowledge, but most important to optimise its own knowledge acquisition behaviour.

3.12.2 Grouping by Similarity

The metalearning process functions using VSM (Section 2.7.2) and by clustering the VSM matrix (shown in formula 2.31). It iterates every row (a pattern p_i) and comparing it to all the other matrix patterns using formula (2.32), thereby producing a new symmetric matrix $B_{m,n}$ which defines how similar is each pattern to all other patterns. All values in matrix B are *min-max* normalised in order to ensure the same scaled range of similarity between zero and one. The matrix $B_{m,n}$ is square, e.g. $m = n$ and is shown in (3.10), and the diagonal is set to one since each pattern is identical to itself.

$$B_{m,n} = \begin{bmatrix} (p_1 \sim p_1) & (p_1 \sim p_2) & (p_1 \sim p_3) & \cdots & (p_1 \sim p_m) \\ (p_2 \sim p_1) & (p_2 \sim p_2) & (p_2 \sim p_3) & \cdots & (p_2 \sim p_m) \\ (p_3 \sim p_1) & (p_3 \sim p_2) & (p_3 \sim p_3) & \cdots & (p_3 \sim p_m) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (p_n \sim p_1) & (p_n \sim p_2) & (p_n \sim p_3) & \cdots & (p_n \sim p_m) \end{bmatrix}. \tag{3.10}$$

We process matrix $B_{m,n}$ using K-Means clustering [Hartigan and Wong, 1979], so that we *minimise* the distance to the cluster centres μ_i as shown in (3.11) on derived dataset $\{x_j\}_{j=1}^{n}$ where x_i are the similarity values of pattern p_n to all other patterns.

$$\sum_{j=1}^{n} \min_{i=1,\dots,K} \|x_j - \mu_i\|^2 . \tag{3.11}$$

The similarity group of patterns is therefore defined by associating each x_j with the centroid closest to it, i.e., $k_i = \{x_j : \|x_j - \mu_i\| \leq \|x_j - \mu_m\| \text{ for all } m = 1, 2, \dots, K\}$. Estimating the number of clusters uses the number of dominant eigenvalues of the matrix B where $B(i,j) = \|x_i - x_j\|^2$. The result is a number of clusters of patterns p_i which group by *attribute* similarity and frequency, due to the fact that the original matrix $A_{m,n}$ was a VSM.

3.12.3 Generalising Cluster Graphs

From the aforementioned procedure the input patterns can be categorised into groups which are then generalised. Generalisation uses Semantic *relational* similarity, as explained in Section 2.7.3, and using a variation of the formula (3.4). In this instance the query is not from one specialisation to another (e.g., from one term t_i to t_k) but from two different terms t_i and t_k to their *common* super-classes. The algorithm generates new graphs which are based on two factors:

1. graphs must be isomorphic (e.g., have the same directed edges)

2. graphs must contain *similar* nodes

The premise is that by replacing the specialisation terms t_i and t_k labelling nodes n_i and n_k in two structurally identical (same edges) graphs, then the newly produced graph represents a *belief*, e.g., an inferred knowledge graph which has been produced by abstracting node labels of very similar graphs.

The algorithm may produce multiple belief graphs but their population should be smaller than the original number of graphs contained within the cluster. Adjusting the distance metric or the number of clusters can result into smaller groups within clusters which can be more similar, thus resulting in more homogeneous structural similarity (isomorphism) with more abstract beliefs.

3.12.4 Optimisation by Belief Evaluation

The purpose of the clustering and generalising procedure described in Sections 3.12.2 and 3.12.3 was that the Icarus engine would end up with graphs which would be supported by multiple knowledge instances acquired either by training or during evaluation. The amount of graph specialisations which supported a new graph belief, would strengthen the notion that this belief, an inferred graph pattern extracted via generalisation, would allow Icarus to acquire new knowledge more accurately. This assertion is based upon the hypothesis that if *multiple* graph specialisations can be isomorphically projected to a single graph belief, then that recurring pattern should be preferable over individual graphs which have a structure that is not encountered often. Due to time constraints and because the scope of the research work could widen a lot whilst introducing additional risk, this algorithm was not implemented.

3.13 Conclusion

Icarus architecture was chosen as it is one of the most well-defined yet generic cognitive architectures. Other CA architectures do exist, however some are theoretical constructs and the rest are simply too complex for my scope. As with cognitive or synthetic AI, the publications, research and work carried out is mostly theoretical and serves as the foundation upon which agents are to be implemented. Taking into consideration all the above, Icarus is an architecture which offers the long-term/short-term differentiation, and allows for the juxtaposition of neural-temporal short term entities and processes and long-term non-temporal entities and processes. Furthermore, it has been tried and tested in physical agents [Langley et al., 2009; Langley and Choi, 2006; Langley et al., 2004, 2003] in an event-driven approach, with a clear segmentation of the memories and modules.

In practice, Icarus serves as a theoretical approach, just as Haikonen's research (Section 2.5.2) and Bach's propositions (Section 2.5.1) serve as a theoretical guideline. What is most important is how it is implemented, the functions it performs, and the purpose the implementation may serve. Icarus engine is a standalone agent developed in C++ and running as a single process under a Linux environment. It accepts incoming paradigms (examples) provided by a knowledge engineer, pre-processes them, parses them, learns from them, and internally organises its memory, look-up tables, knowledge base and episodic memory. It further supports query from external agents and processes unknown input producing output graphs. Thus, it serves the basic functional requirements of imitation (as discussed in Section 2.2.1). Furthermore, and most important, the current Icarus design as described in this Chapter addresses the key components described by Bandura, Haikonen, Bach, Lawniczaka and Di Stefano, and Langley (Icarus). How each theoretical approach is addressed is described in the following Sections.

3.13.1 Bandura and Imitation in Humans

Bandura describes four key components [Bandura, 1986] (see Section 2.2.1) listed below:

- A perceptive process that produces a model

- A Process which retains symbolic conception through experience

- Internal model serving the purpose of providing responses

- Motivation determining if a skill is reused

The Icarus design uses an MDP as a perceptive process (Section 3.1), which produces a model of the input, a conceptual graph (Section 3.11) [Sowa, 1999]. That MDP process retains the symbolic conception (the conceptual graph) through a rewarded/reinforced experience (the MDP via Q-Learning [Sutton and Barto, 1998]). Its internal model serves many purposes, one of which is to provide responses to input queries such as during evaluation (Section 3.3). Finally, the actual motivation behind reusing an experience (which has since become a skill) is reinforcement and rewarding, based on reinforcement learning, which as stated before, is inspired by the human psychological rewarding processes [Watkins, 1989; Sutton, 1984; Galef Jr, 1988] discussed in this Chapter's first three Sections.

3.13.2 Haikonen and Cognitive AI

Haikonen has provided multiple criteria for a Cognitive AI (Section 2.5.2) of which the following are full-filled and addressed by the current design.

- A process which creates qualia (subjective experience)

- Meaning and representation and their relation to information

- Perception and recognition

- Association and associative memory

- Cognition, understanding, memory (short-term and long-term) and models

The first and foremost criteria is addressed by both the paradigm decomposition process (Section 3.2) but also from the MDP approach, evaluation and decision making processes (Sections 3.1, 3.3 and 3.5). Meaning and representation and their relativity to information are addressed by the Semantics, Statistics and probabilistic approaches, whereas the ultimate goal of the agent is to acquire new knowledge and re-use it. It does so through a (limited but highly accurate) perceptive recognition cascade of algorithms, which rely on associative memory (Sections 3.8 and 3.9) with the ultimate goal of cognition and understanding. Memory models are partitioned in short and long-term as Icarus dictates, and the only criteria set by Haikonen which is only partially addressed is "*Imagination and planning*". Whereas a rudimentary form of reasoning is used for decision making and simple inference for paradigm decomposition, Icarus does not deal with imagination or planning.

3.13.3 Bach and Synthetic Intelligence

Bach and his guidelines (Section 2.5.1) have been the basis upon which the Icarus implementation was developed. Those guidelines are:

- Create holistic architectures which are functional
- Avoid methodologism
- Aim for the larger picture
- Build systems which are not too narrow or which do not focus only specific domains (Symbol grounding problem)
- Robotic embodiment is not panacea
- Focus on autonomous systems
- Emergence of intelligent agents won't take place on its own

The decision to make Icarus an agent (rather than a system or a tool) addresses the *holistic* aspect; albeit it may not be a holistic agent design it is functional and takes into consideration many processes and memory requirements. It avoids methodologism and aims for the bigger picture since it is an agent which in principle is built to learn by imitation and is not constrained to a specific domain (in fact originally it was designed for learning how to understand computer programs). Whereas it is somewhat bound to the Syntactic nature of NLP (due to the POS tagger) it is not limited by it; it could in fact be used in other domains such as Mathematics, Physics, Chemistry or any kind of domain which uses written language. It doesn't aim for robotic embodiment although it could be used within Robotics, focuses on autonomy and relies on user-agent interaction as means to augmenting its knowledge and experience.

3.13.4 Five Cognitive Agent Criteria

The five criteria for cognitive agents as described by Lawniczaka and Di Stefano [Lawniczak and Di Stefano, 2010] are:

- Perceive information in the environment provided by other agents
- Reason about this information using existing knowledge
- Judge the obtained information using existing knowledge
- Respond to other agents
- Learn and augment current knowledge if newly acquired information allows it.

The Icarus agent perceives information in its (virtual) environment, information related to the NLP (Statistics, Syntax, Semantics) which is extracted from the imitative process (observation and decomposition). That information is reasoned with and reused, it is also judging it (using the decision making process) and furthermore, it is associated and stored. It is able to respond to other agents (via queries) and most important it focuses on learning and augmenting its current knowledge from others.

3.13.5 Icarus and Cognitive AI

The Icarus design principles [Langley et al., 2009; Langley and Choi, 2006; Langley et al., 2004, 2003] are listed below:

- Integration of perception, cognition and action
- Combination of symbolic structures, knowledge representation and affective values
- Behave re-actively in combination with problem-solving
- Learn from experience whilst using background knowledge

The implementation and theory described in this Chapter address the integration of perception and cognition, and how action management (decison making) relate to reinforcement learning and information acquired from the agent. Furthermore, the MDP template enables the combination of KR and affective values in a novel and unique manner; it combines Symbolism and Connectionism with Heuristics in order to simulate the imitative process. The agent behaves re-actively, but problem-solving is not programmed or embedded in the agent; instead the agent learns how to solve specific problems or tasks, through imitation. Thus another novelty is the way in which problem-solving capacities are acquired: through imitation. It expands on the learnt behaviours using approximation and classification through neural associative memory, and reuses its background knowledge (conceptual graphs, Semantics, Syntax and Statistics) in order to further augment its experience and knowledge.

3.13.6 Discussion on Icarus Implementation

The aforementioned functionality is enabled in order to gauge and establish if the theory and design described in this Chapter, can empirically and methodologically validate the hypothesis that cognitive agents and artificial imitation can use the simulated processes to acquire knowledge indefinitely. The exact process that is being learnt by imitation is dependency, Semantic and Syntactic parsing, as a unitary temporal sequence. In the next Chapter the dataset and its related complexity is discussed in order to describe *what* the agent learns and is evaluated on.

Chapter 4

Conceptual Graph Dataset

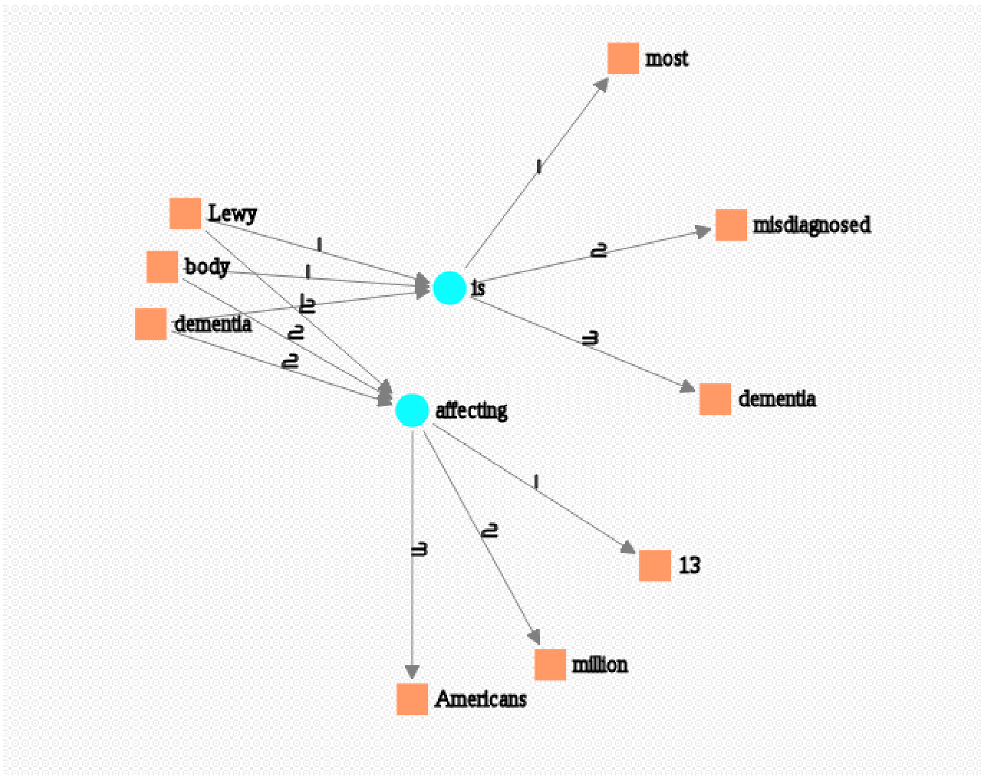


Figure 4.1: A CG example

The Icarus agent is evaluated using a dataset which is randomly partitioned into a training set and an evaluation set. The training set contains pairs of an input pattern (sentence) p_i and the associated conceptual graph G_i. Therefore the projection of text onto a KR (the conceptual graph) is the task or problem which the agent is required to resolve. The evaluation and experimentation process which is described in the next Chapter 5 uses the dataset described in this Chapter.

The dataset described in this Chapter is from RSS feed data, which is represented using Conceptual Graphs, presumed sound and complete, and using the approach of [Obitko, 2007; Amati and Ounis, 2000] as discussed in Section (2.3.1). Hereinafter I briefly examine the existing datasets commonly used, following by how this dataset was created and why as well as the option of translating other datasets into a CG-set, and then provide an overview of its characteristics via statistical analysis. Last but not least, an analysis of the complexity contained with the dataset is given and CG Figures (in Appendix B) are provided in order to demonstrate possible issues and key factors affecting the Icarus engine.

4.1 Datasets for NLU and NLP

Name	**Train Size**	**Test Size**	**Average Sample Size**
ATIS3	7,300	1,000	20.5
Penn-Treebank-3	2,499	unknown	25.6
BioNLP11ST	800	260	unknown
GeoQuery	600	280	6.87
RoboCup Data-set	300	unknown	22.52

Table 4.1: Common NLP Datasets

A variety of data-sets for NLU and NLP exist, some are new, created as part of a publication, whereas others are well established and have been used widely. Because a complete list would be too large, the most widely used data-sets are shown in Table 4.1. The ATIS3 dataset [Hemphill et al., 1990] has been used in numerous studies and research work, and as such has been continuously updated for decades. Similar to the ATIS3 the Penn-Treebank/Wall-Street Journal dataset [Marcus et al., 1993] has also been developed and used for decades and is partitioned in smaller sets [1]. Another dataset is BioNLP11ST, which has been created for the BioNLP shared task [Kim et al., 2011]. Similar datasets have been created from RoboCup [Kitano et al., 1997] and used in parsing for robotics [David L. Chen, 2010; Chen and Mooney, 2008]. New datasets have been introduced in the past years, some of which are not free, such as the English Web Treebank [Bies et al., 2012], or the OntoNotes [Hovy et al., 2006], and others which are open-source and free, such as the GUM corpus [Zeldes, 2016], or the data-sets hosted on Universal

[1]`https://catalog.ldc.upenn.edu/ldc99t42`

Dependencies[2]. To the best of my knowledge no conceptual graph dataset exists, albeit one has been proposed [Campbell and Musen, 1992] in the past.

4.1.1 Creating a New Dataset

Due to the fact that conceptual graphs were used throughout this thesis and the research carried out, I took the decision to create a CG dataset. The options presented were to either create a tool which would translate a dataset from the existing ones (see Table 4.1) into a CG dataset, or create a new one[3]. Translating existing ones required some form of heuristic and probability based inference, so that concepts and relations would be accurately extracted (similar to how the agent functions). The risk seemed to be too high when considering how accurate such a translation tool would have been, therefore the final decision was to create one manually.

The data used was from RSS feeds, since the secondary research question relates to artificial imitation acquiring data from the Internet (see Section 1.5). RSS articles are usually well formatted since they are written from news editors, and are as such of better *quality* when compared to Web data. That observation is reflected by the fact that all parsers perform better on news articles than they do on Web data or utterances.

I chose RSS sites (*BBC, Sky News, USA Today, Science Daily and Knox News*) in order to account for various topics, but mostly focused on health-related news, celebrity news, scientific discoveries and lifestyle articles. From empirical observation I deduced that those articles tend to be mostly factual rather than opinionated or subjective, and range from simple and easy sentences (lifestyle and celebrity news) to very complex ones (health articles and scientific discoveries).

The actual process of creating the dataset was implemented through a web-portal using JavaScript; an input sentence was given, which then had to be converted to a CG. This is the same process the agent has to learn and re-create, thus the interaction with the web-portal is the actual MDP that the knowledge engineer is providing as an example to the agent.

The MDP of creating the actual graphs *is not recorded* as a temporal-sequence; doing so would imply that the agent is **not decoding** or decomposing the example, but simply learning an existing MDP. Ergo, no *hidden* property of the MDP would need be inferred, no decomposition would be required, and the actual process would not be a qualia.

Instead, the only byproduct is the input pattern (sentence) and a graph

[2]`http://universaldependencies.org`

[3]For a summary on CG, please refer to Section 2.3.1

serialised in a JSON format. Each example is saved within a SQL database, which can then be queried by the agent. The dataset contains a total of 1,199 conceptual graphs and sentences and was created over a period of 9 months. The actual dataset is available on-line [4] saved in a JSON file, and a C++ library I created called cgpp[5] can parse them into memory. The patterns range from very small (4 to 5 words) up to 34 words, however the average pattern length is smaller than in other datasets, with 10.92 words average pattern length. Creating a larger dataset with bigger pattern lengths would have required a considerable amount of work with additional time spent on creating it.

4.1.2 Partitioning the Dataset

The new CG dataset created for the Icarus engine was randomly partitioned into smaller ones; the reason was that in order to gauge accuracy and the corresponding effect of pattern size, as well as to enable sub-sampling techniques (stochastic random sub-sampling) and faster training times. The pseudo-random generator, a Mersenne twister implementation [Matsumoto and Nishimura, 1998] first prompts the user to decide on the cut-off length of input (maximum sentence size) and then filters graphs and corresponding sentences. Then, it proceeds to randomly partition the dataset into 4/5 training set and 1/5 testing set as is usually the case with most datasets. Repeating this process a few times creates different similar-sized random datasets of a specific pattern size, thus the accuracy and performance reported in the next Chapter is uniform, average and consistent.

As shown in Figure 4.2 multiple random subsets were used obtained from the dataset. Those were used to train and evaluate the agent; the actual input size varies in each one; some are smaller, others are larger. The top left plot shows how minimum input size, maximum input size and average input size vary. The X Axis denotes the actual subset index ID, and the data plotted are the stats of the *testing* set (and not the training set). On average the input size for patterns p_i has a mean $\bar{x} = 7.9733$ terms and a standard deviation of $\sigma = 2.459$ term size, thus all testing subsets are consistent and biased towards smaller to medium-sized sentences, as shown in the top right plot. The minimum input size is very consistent with a mean $\bar{x} = 3.6667$ terms and a standard deviation of $\sigma = 0.4879$ terms, shown in the bottom left plot. Only the maximum input size varies, as shown in the bottom right plot, with a mean $\bar{x} = 15.20$ terms and a standard deviation of $\sigma = 10.571$ terms.

[4] `https://github.com/alexge233/conceptual_graph_set`
[5] `https://github.com/alexge233/cgpp`

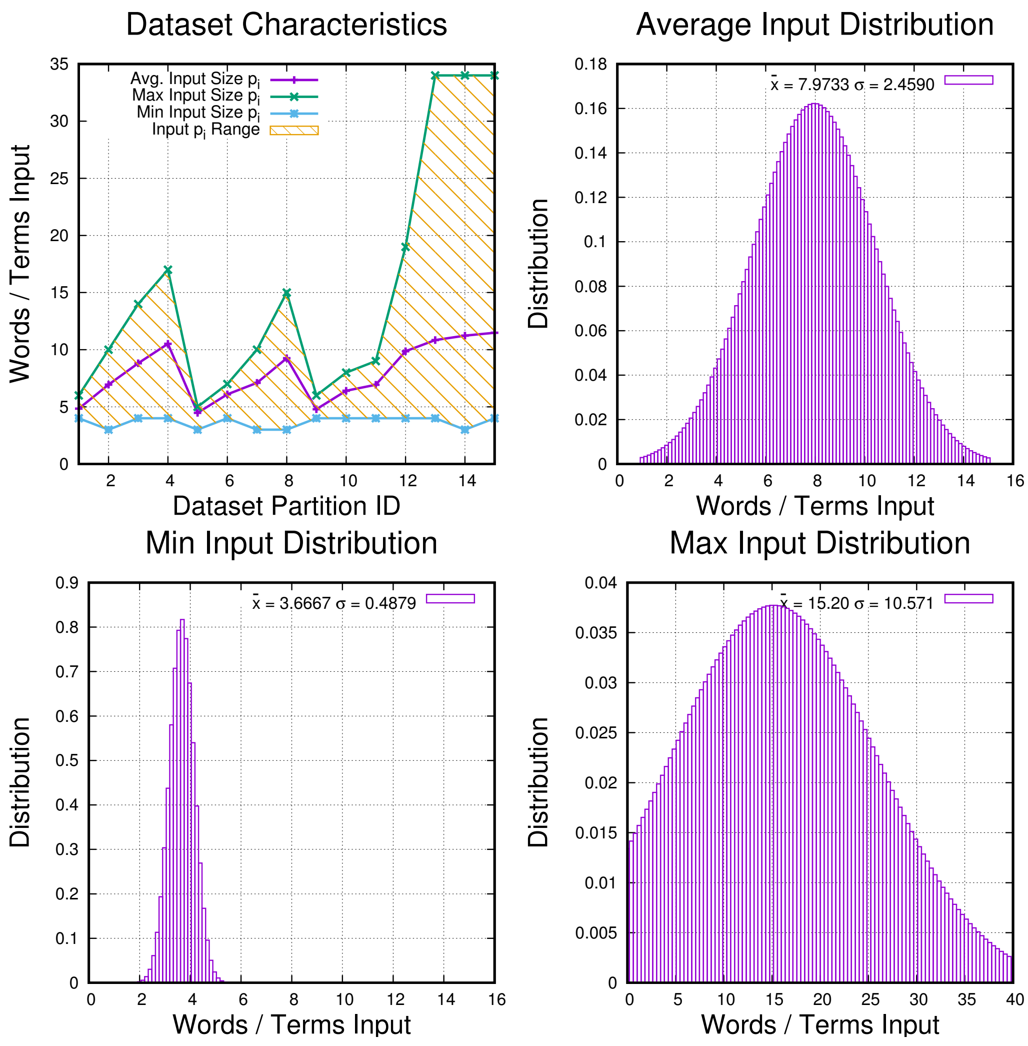


Figure 4.2: Dataset Info and Distribution.

4.1.3 Translating and Converting Datasets

Translating an existing dataset to a CG dataset would have required the partitioning of the existing dataset into the bipartite nature of CG: concepts and relations. Doing so manually would imply the transcribing of a non-CG dataset (e.g., MRL) to a CG one, therefore taking a considerable amount of time for transcription, and a questionable accuracy.

On the contrary, having created a CG dataset which in effect trained the agent, it is now possible to transcribe other datasets because the mechanisms which decide on concept or relation nodes have been implemented and trained. Furthermore, differences between datasets do exist regarding the edges and how those are implemented. For example, a fully trained Icarus engine could in theory partition ATIS3 or Penn Treebank WSJ into concepts and relations, and then learn the edges by observing the edges contained in the transcribed datasets.

This process would function exactly the same way as the heuristic decomposition works (see Section 3.2), with the only difference being that it would be done in a two-pass procedure: once to convert the dataset into a CG, and then once to parse each dataset sample into an MDP-qualia from which to learn. Implementing the aforementioned process would require pre-training of the Icarus engine, therefore it would add an additional overhead: pre-train the agent in order for it to be able to transcribe a dataset before being able to be trained. Therefore the decision to opt for a new CG dataset was justified during the research carried out and described in this thesis. However, as future work it is possible to use Icarus to transcribe existing MRL datasets into CG datasets.

4.2 Conceptual Graph Complexity

An issue often not addressed is the *complexity* of the underlying MRL, CG or knowledge in general. Generally it is assumed that larger input will produce more complex KR, the reason why all parsers' accuracy deteriorates with increased input size [Choi et al., 2015]. In the case of CG, complexity is measurable because it refers to *graph complexity* and we can therefore use existing approaches to quantify it and map it, and then deduce how input complexity relates to agent accuracy and performance.

Hereinafter the use of *node to edge ratio* $|V|/|E|$ (also known as *graph sparseness*) and *average path length* l_G of a graph are used as indicators of the structural complexity of a graph; other more complex measures such as *betweenness, radius, closeness, clusterization* [Barooah and Hespanha, 2007; Wright, 1977] were not used due to requiring more elaborate work on the CG without necessarily providing data that could aid the study of knowledge complexity.

In addition to the above, the *edge search* space was used (e.g., how many possible edge combinations exist in a graph) as a metric of defining the entire set of possible edges. What was discovered (and discussed in detail in the next Chapter) is that similar to other research in NLU, accuracy and performance deteriorates with increased complexity.

Other metrics such as *clustering coefficient* or *degree distribution* could have been implemented; however the focus of this Chapter is the dataset and not its complexity metrics and analysis, which was an empirical byproduct. Furthermore, the clustering coefficient for directed graphs uses *triplets* [Watts and Strogatz, 1998] as shown in (4.1), where C_i is the cluster coefficient, e_jk is an edge from v_j to v_k, and k_i is the number of neighbours to a node.

$$C_i = \frac{|\{e_jk : v_j, v_k \in N_i, e_jk \in \mathbb{E}\}|}{k_i(k_i - 1)}. \tag{4.1}$$

However, the notion of a triplet in a CG is ill-defined; instead it should be a quadruple since the use of a triplet is violated by the fact that concepts via relations would connect to another concept thereby forming a rhombus/quadruplet instead of a triplet. This difference is demonstrated in Figure 4.3, where the left-side rhombus is a CG cluster, whereas the right-side triangle is the traditional approach described by equation (4.1).

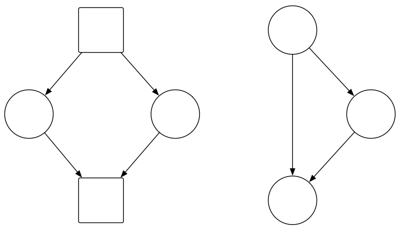


Figure 4.3: Clustering Coefficient: CG rhombus versus KR triplet

Through the creation process (Section 4.1.1) certain empirical observations were made, based on visual analysis of the data; recurring patterns and common rules were identified which have been recorded and are herein presented. In all graphs presented in Section 4.2.1 the root node is always at the left side.

4.2.1 Graph Columns and Linearity

Certain patterns are associated with a *column-like* appearance, related to the linear structure of the phrase. Such an example is shown in Appendix B.1. Those graphs are assumed to be easier to re-create and learn since their structure is linear; furthermore they produce simpler training material due to fewer contradicting nodes and edges. A column like graph will have a single average path length, and the specific graph in Appendix B.1 has $|V|/|E| = 0.54$ and $l_G = 8$. Another column-like graph is shown in Appendix B.2, with a $|V|/|E| = 1.14$ and $l_G = 4$.

Some of the linear graphs are less column-like, and resemble more of a butterfly or tree branch, with large concept clusters, such as the one shown in B.3. In this graph, the ratio is $|V|/|E| = 0.62$ and $l_G = 2$.

A more complex graph is shown in Appendix B.4; this one has a large mid centre cluster of concepts, which are connected by two relations before and after. In this graph, the ratio is $|V|/|E| = 0.78$ and $l_G = 4$. Other forms of column graphs are identified by clustering of concept nodes, which are all connected to the same group of relations. Relation grouping is also possible, but more rare than concept grouping.

4.2.2 Graph Branching and Grouping

Branches of graphs increase complexity, especially when a graph contains multiple branches. Appendix B.5 shows what a shallow graph with a high ratio of concepts to relations, and a high ratio of nodes to edges looks like, especially when combined with a very small path length. The actual ratio is $|V|/|E| = 0.91$ and $l_G = 2$. However, in this instance the graph has two groups, the first connecting the left-most concepts through relation "is", and the second relation "affecting" connecting to the second group on the right.

This type of causality is represented by branching, which adds a level of complexity. A measure which could potentially aid would be the clustering coefficient for directed graphs. Yet that would not suffice since the distinction between groups based on edges is more difficult to calculate and would thus need to be combined with degree distribution. The graph shown in Appendix B.6 which has a ratio $|V|/|E| = 0.51$ and $l_G = 4$.

Branching may include groups of column-like (linear) graphs, such as the one shown in Appendix B.7. This type of graph is a peculiar entity: there appear to exist two distinct column-like graphs, joined together by the relation "for", it has a $|V|/|E| = 0.63$ and $l_G = 8$. Those graphs are often hard to get right because they are specialisations; they define a new rule (or often an exception to a recurring pattern) which is an outlier.

Another column-like graph in Appendix B.8 shows how even small branches may affect accuracy; it has a $|V|/|E| = 0.93$ and $l_G = 6.2$. Similar to Appendix B.7 is Appendix B.9, only this one is more symmetric, has a smaller $l_G = 4$ and a ratio $|V|/|E| = 0.84$. Last but not least, there is Appendix B.10 which albeit having an $l_G = 5.3$ is characterised by a ratio $|V|/|E| = 0.92$, thus making it more sparse than dense.

4.2.3 Graphs and Operators

Analysing graph patterns during their creation (empirically) provided some interesting observations, most of which revolve around logic embedded in the KR and relations/operators, such as "*for*", "*and*", "*or*", "*if-else*". Some hint towards cause and effect, others hint state changes, and whilst there are exceptions they are very rare. A clear connection between relations (in CG-terms) and operators exists; operators are always relations and relations are always the node in a graph which diverges into branches and leafs. For example, in Appendix B.7 it is a "*for*" that creates the downside branch, and in Appendix B.9 it is an "*or*" and an "*in*" that act as the connecting branch nodes.

This form of reappearing patterns is what inspired the theoretical algorithm in Section 3.12. Much of the premise of Semantic parsing in the agent relies on the *meaning* of those terms which act as operators in the graph; therefore the hypothesis is that Semantic-based parsing is preferable to Syntactic parsing, since it can inherently provide greater accuracy; this hypothesis is examined later in Chapter 5.

4.3 Dataset Conclusion

In this Chapter the dataset being used and created was presented, as well as a description of the most commonly used datasets in NLP and NLU was given. Furthermore, certain attributes and characteristics of conceptual graphs have been empirically analysed and presented, relating those attributes to complexity in the underlying knowledge represented. The dataset described in this Chapter was created out of the necessity to use a CG-set for training and testing the Icarus engine. Albeit it is smaller than compared datasets often used (see Section 4.1.2) it sufficient to evaluate the agent and serve as a measure similar to other datasets and research.

Chapter 5

Experiments, Methodology and Results

In this Chapter I describe the experimental methodology and how in particular, the Icarus agent was used and validated. The task Icarus was evaluated on is processing unknown sentences and correctly projecting them on CG. Each Section in this Chapter contains the implementation of an algorithm (or group of models), and the corresponding results and is compared with other similar algorithms. As described in earlier chapters, the development of Icarus started by following simple implementations, and progressed towards more complex models (see Chapter 3). Therefore, the experiments described hereinafter analyse the performance of each model and algorithm implemented and tested, and not the best one or the overall accuracy. Icarus was tested using the dataset created by projecting sentences on CG (see Chapter 4), hence the input is a sentence and the output is a conceptual graph. The primary measure of accuracy used is the Dice-Sørensen coefficient, for reasons described in Section 5.1.3.

5.1 Methodology and Experiment Design

Careful consideration was put into the methodology; there is a clear need to provide consistent, reproducible and quantifiable results [Winsberg, 2003] and experimental data in order to assure the validity of the claims and conclusions made in this thesis. It is important to note that the experiments were carried out on **different versions** of Icarus using **different algorithms**, starting from the most simplistic and naive ones, and then through constant evaluation and optimisation, progressed towards more complex versions of Icarus.

5.1.1 Randomised Block Design

Because of the aforementioned criteria, I opted for a *stochastic* methodology, which uses random blocks of experiments which are repeated multiple times [Cavazzuti, 2012] as a means to ensure constant accuracy, by averaging the performance of randomised data samples. This approach is known as randomised block design (or RBD) [Winer et al., 1971, p. 240] and uses multiple iterations and repetitions of experiment blocks [Higgins, 2003]. Furthermore, because datasets are partitioned randomly (see Section 4.1.2) this approach makes the methodology described herein a *stochastic mini-batch* design, inspired from the mini-batch stochastic gradient descent [Li et al., 2014].

In RBD, experiments are organised in blocks: for my intents and purposes, every first block of experiments is organised using the preordained randomised datasets from Chapter 4, which is called L1 (or level 1). The second level of RBD refers to the Algorithm (or group of models) being used, in order to determine which one functions more accurately; that is the L2 (level 2). The last level (or L3) is the experiment itself: each experiment was repeated at least ten times, and then the agent accuracy was averaged, instead of using the best one. This approach ensures that results reported in this Chapter are reproducible, constant and robust, and not an outlier of good performance, or based on random/PRNG performance. Because many experiments were performed multiple times often the L3 is an average value of more than 10 experiments. Each individual experiment has multiple episodes (associated MDP with input and output graph), I denote this as L4. However, because L4 is the most granular level and always averaged, the L4 episode scores are simply ignored and averaged as L3.

5.1.2 Experiment Logs

Every experiment produces two log-files; one for the episodic memory data and one for the action data. The log-file for the MDP episode data is described in the list below.

1. **Graph ID**: a unique ID (UUID version 4)
2. **Episode reward**: a Boolean value obtained by examining if the output graph is identical or isomorphic to ideal graph. Averaged for all graphs in the L3/L4 block.
3. **Jaccard coefficient** [Real and Vargas, 1996]: the difference of ideal and actual graph output $J_{G_i G_k} \in \mathbb{R}, 0 \geq J_{G_i G_k} \leq 1$. Averaged for all graphs in the

L3/L4 block.

4. **Sørensen-Dice coefficient** [Rijsbergen, 1979]: the difference of ideal and actual graph output $S_{G_iG_k} \in \mathbb{R}, 0 \geq S_{G_iG_k} \leq 1$. Averaged for all graphs in the L3/L4 block.

5. **Node similarity**: min-max normalised percentage of the same nodes $S_{v_iv_k} \in \mathbb{R}, 0 \geq S_{v_iv_k} \leq 1$. Averaged for all nodes of all graphs in the L3/L4 block.

6. **Edge similarity**: min-max normalised percentage of same edges ($S_{e_ie_k} \in \mathbb{R}, 0 \geq S_{e_ie_k} \leq 1$. Averaged for all edges of all graphs in the L3/L4 block.

7. **Pattern Size**: how large is the input pattern/sentence $p_i \in \mathbb{N}^+_{>0}$. Averaged for all input samples in L3/L4 block.

8. **Graph Sparseness**: the ratio of nodes/vertices to edges/arcs $|V|/|E| \in \mathbb{R}$. Averaged for all graphs in the L3/L4 block.

9. **Average path length**: an average size of the continuous paths in a graph $l_G \in \mathbb{R}_{>0}$. Averaged for all graphs in the L3 block.

10. **Edge Space**: the cardinality of the possible edges for a graph, given its nodes $\|E_G\| \in \mathbb{N}^+_{>0}$. Averaged for all graphs in the L3 block.

11. **VSM Similarity**: an array/list of similarities $S_{p_ip_k} \in \mathbb{R}, 0 \geq S_{p_ip_k} \leq 1$ with respect to other patterns.

Similarly, the action output log-files record information and meta-data relation to actions a_t taken by the agent, as shown in the list below.

1. **Term value**: t_i on which action a_t operated e.g., "cat", "on" or "mat".

2. **POS Tag**: the actual POS tag of the term t_i when converting it to a concept or a relation.

3. **Type of action**: what did the action a_t do, e.g., convert a term to a node, or create an edge between nodes.

4. **Description**: a simple word describing if the action was random, heuristic, semantic, neural-based, etc.

5. **Value**: the value that made the agent take the action (e.g., if it was based on semantics, what was the value, if based on probabilities, what was the probability, etc).

For every single experiment (L3), an episode file and an action file are created. The L3 block is run using scripts in batches, and the L1 block (datasets) is also run using batches. What changes therefore is the L2 block; which Algorithms execute during a series of experiment batches. For every group of L1, L2 and L3 experiments, another script averages all the values, thus extracting the average of each value described above. Furthermore, every data in the agent's memory is serialised in a binary file (policies, graphs, probabilities, semantics, etc) for reuse.

Neural networks are trained once (albeit optimising them is a long and tedious process). In this way, the agent does not start in a *tabula rasa* state every time, but contains some prior (long-term) knowledge, by having already pre-trained the networks. On the contrary, probabilities and semantics are not loaded from disk; they are populated during training. Whilst this may appear as non-beneficial, it ensures that some of the experiments do not begin with background knowledge which would skew or bias towards later experiments performing better.

From the produced log-files, other meta-data files can be created (complexity data, state-action ratios, etc) which are then used to extrapolate dataset complexity, identify potential issues and visualise the agent's performance.

5.1.3 Accuracy Measures

The most commonly used measure for accuracy is the F_1 score [Powers, 2011; Metz, 1978], as shown in (2.34) in Section 2.7.7. However, the notion of precision and recall does not apply in the Icarus scenario: the precision *only relates* to *how similar* (or dissimilar) the output graphs are, e.g., $\Delta_G = G^y - G^{\hat{y}}$. Furthermore, there exists no *recall*, because the agent is not performing classification. Since the notion of *similarity* of graphs is based on their sets of *concepts, relations* and *edges*, I chose to examine the two most famous set similarity measures, the Jaccard coefficient [Real and Vargas, 1996] shown in (5.1) and the Dice-Sørensen coefficient [Rijsbergen, 1979] shown in (5.2).

$$S(A, B) = \frac{2 \mid A \cap B \mid}{\mid A \mid + \mid B \mid}. \tag{5.1}$$

$$J(A, B) = \frac{\mid A \cap B \mid}{\mid A \mid + \mid B \mid - \mid A \cap B \mid}. \tag{5.2}$$

In both equations (5.1) and (5.2) A and B are sets. The above equations are used for measuring the similarity coefficients between concepts, relations and edges, because a graph contains those sets (3.1). When measuring the similarity of an output $G^{\hat{y}}$ to the ideal graph G^y, the actual comparison is taking into account the coefficient of concepts and relations as a unitary set (since they both are nodes), and then treating both node sets V and edge sets E with the same *weight*, as shown in (5.3). Each of the functions in the following formulae is replaced by the actual coefficient in (5.1) or (5.2), where appropriate.

$$S(V^y, V^{\hat{y}}) = \frac{S(C^{\hat{y}}, C^y) + S(R^{\hat{y}}, R^y)}{2}. \tag{5.3}$$

Replacing $S(V^y, V^{\hat{y}})$ from (5.3) in (5.4), there is no bias towards nodes, since both node and edge sets are treated as equally important.

$$S(G^y, G^{\hat{y}}) = \frac{S(V^y, V^{\hat{y}}) + S(E^{\hat{y}}, E^y)}{2}. \tag{5.4}$$

In the case of Jaccard, the result penalises differences in sets not only for missing items of $G^{\hat{y}}$ in graph G^y, but also for additional items, which do not exist in the target/ideal graph. Similar to the Dice-Sørensen coefficient, the coefficient is calculated for each pair of sets (concepts, relations, edges), so that $J(C^y, C^{\hat{y}})$, $J(R^y, R^{\hat{y}})$ can be averaged, and then $J(E^y, E^{\hat{y}})$ is calculated and averaged. Then the node coefficient is calculated by averaging as shown in (5.5) and finally the node and edge coefficients are averaged, as shown in (5.6).

$$J(V^y, V^{\hat{y}}) = \frac{J(C^y, C^{\hat{y}}) + J(R^y, R^{\hat{y}})}{2}. \tag{5.5}$$

$$J(G^y, G^{\hat{y}}) = \frac{J(V^y, V^{\hat{y}}) + J(E^y, E^{\hat{y}})}{2}. \tag{5.6}$$

The Jaccard coefficient is a more strict measure which ideally would be used, however the Dice-Sørensen coefficient has *the same form* as the F_1 score [Intan et al., 2015, p. 158] and therefore functions as the primary accuracy quantifier. It is also important to note that similarity is respective to the first graph in Dice-Sørensen: the formula quantifies only how similar $G_i^{\hat{y}}$ is to G_i^y and its parameters are non-anadrome.

In NLU and dependency parsing, some metrics often used are [Choi et al., 2015; Zhang and Nivre, 2011; Liu et al., 2006; Nivre and Scholz, 2004]:

- Labelled attachment score (LAS): percent of correct labels and edges
- Unlabelled attachment score (UAS): percentile of correct edges
- Label accuracy (LS): correct labels
- Exact match (EM): exact tree/graph.

The LAS appears to refer to labels of edges/arcs, and as such it is non-applicable (n/a) in this thesis. The agent has no way or mechanism to infer labels, and CG literature does not use edge labels, but relations for this purpose. Comparing those measures to the aforementioned formulae and metrics, Table 5.1 gives an equivalence scenario.

NLU, NLP	**This Thesis**
LAS	n/a
UAS	Jaccard & Dice-Sørensen coefficient
LS	n/a
EM	Boolean Graph Accuracy

Table 5.1: Accuracy Metrics Equivalence.

Other kinds of meta-data were obtained from a combination of the log-files: state-action ratio, complexity, and hereinafter when a graph or plot is presented a multidimensional matrix of data is normally reduced using principal component analysis (PCA) [Wold et al., 1987]. Then a projection of the most significant Eigenvectors to a lower dimension (usually a 1D or 2D) is used. This approach is taken when a data matrix has more than 3 columns and is consistent throughout the thesis.

5.2 Semantic-Heuristic Experiments

5.2.1 Implementation

The first type of Algorithm implemented and used in Icarus is a Heuristic based on VSM, relational Semantics and a_t action swapping. The core notion behind the Algorithm is to *approximate* highly similar episodes and find potential actions which could be re-used in an episode which hasn't been experienced before. The components involved are: VSM indexing similar episodes (Section 2.7.2), the Semantic relation Algorithm (Section 3.7) and the MDP template, as described in the previous Chapter 3. The Algorithm queries the *topmost* similar episodes (above 50% similarity) and then orders them starting from the most similar to the least similar. It then iterates the similar episodes, finding identical actions which it can reuse, and for the entities which do not exist in the similar episodes it attempts to establish a semantic similarity. By finding the topmost similar label, it then re-uses the action in the similar episode for the new episode. The pseudo-code in Algorithm 5 describes the aforementioned heuristic. The Algorithm takes as input the actual pattern p_i and returns a set of candidate actions A_i, each one associated with a *score* which assigns a *suitability* value. The premise of suitability uses a linear equation of the VSM similarity coefficient and for each term replaced the actual semantic similarity value.

Algorithm 5: Semantic-Heuristic Decision Making.

```
Input: $p_i$
Output: $A_i$
1  $Q(S_k, A_k) = VSM(p_i)$
2  $Q(S_k, A_k) = max\{S_k, \dots\}$
3  for $a_t \in A_k$ do
4      $\beta(a_t) = Sim(V_{p_i}|A_k)$;
5      if $a_t(t_k) \in p_i$ then
6          if $N(a_t)$ then
7              $A_i^+; a_t(p_i) = \beta(a_t) \cdot 1$
8          else if $E(a_t)$ then
9              $e_{ij} = a_t$
10             if $t_j \in p_i$ then
11                 $A_i^+; a_t(p_i) = \beta(a_t) \cdot 1$
12             else if $v_{t_l t_j} \rightarrow t_l \in p_i$ then
13                 $A_i^+; a_t(p_i) = \beta(a_t) \cdot v_{t_l t_j}$
14     else if $v_{t_i t_k} \rightarrow t_i \in a_t(t_k)$ then
15         if $N(a_t)$ then
16             $A_i^+; a_t(p_i) = \beta(a_t) \cdot v_{t_i t_k}$
17         else if $E(a_t)$ then
18             $e_{kj} = a_t$
19             if $t_j \in p_i$ then
20                 $A_i^+; a_t(p_i) = \beta(a_t) \cdot v_{t_i t_k}$
21             else if $v_{t_l t_j} \rightarrow t_l \in p_i$ then
22                 $A_i^+; a_t(p_i) = \beta(a_t) \cdot (v_{t_i t_k} \cdot v_{t_l t_j})$
```

First the similar policies are obtained by finding the similar episodes. The list of similar policies $Q(S_k, A_k)$ is sorted by most similar to less similar (ignoring policy values). Then each policy is iterated and a *score* $\beta(a_t)$ is used to calculate how suitable this action is, so that for term t_i in p_i we can use a_t which is a candidate action in the set A_K obtained from another pattern p_k. For simple term to node shift-reduce there exist two scenarios: (a) t_i is identical to t_k, or (b) t_i is *semantically similar* to t_k (using equation 3.9). In the first scenario, a straight-forward calculation $A_i^+; a_t(p_i) = \beta(a_t) \cdot 1$ is used, suggesting that one should add to the candidate action sets A_i the action a_t, because the terms are identical; in the later scenario I reduce the value of the candidate by the amount of which t_i is similar to t_k, e.g.: $a_t(p_i) = \beta(a_t) \cdot v_{t_i t_k}$.

Edge actions are a bit more complex: since both terms that connect an edge are used, then both terms must either exist in p_i or be approximated by similarity. Therefore, the criteria are that either both nodes must exist in p_i or that either one or both of the nodes must have some relational similarity to a same-typed node found in p_i. In the event that both exist in p_i, then the score is same as before, e.g., $a_t(p_i) = \beta(a_t) \cdot 1$. If one of the nodes exists, but the other doesn't, then the score is biased by the similarity, so that $a_t(p_i) = \beta(a_t) \cdot v_{t_i t_k}$. In the case where none of the nodes exist, but there are semantic similarities between the edges, the score is further biased so that $a_t(p_i) = \beta(a_t) \cdot (v_{t_i t_k} \cdot v_{t_l t_j})$. The actual value of $\beta(a_t) = Sim(V_{p_i}|A_k)$ is the min-max normalised VSM coefficient, so that $0 \geq \beta(a_t) \leq 1$. This group of Algorithms [Gkiokas and Cristea, 2014a] was inspired from older research in parsing and PBE; they served as the starting point during my research, from there a transition towards Statistics followed suit. It was used because older research used somewhat similar Heuristic approaches, as such an easy starting point was to implement such an Algorithm.

5.2.2 Results and Discussion

This was the first Algorithm implemented and tested; it offers a Heuristic approach based on inference using Semantics. Such an approach was the norm [Carnap, 1948] in the previous decades by many Heuristics-based and Algorithmic-based designs. The testing process used only this Algorithm and a random action generated; the random actions were used only when the Algorithm was not capable of inferring actions.

The averaged recorded results were poor: only 52.55% F_1 was recorded for small input sizes, and medium to large input sizes ranged between 32% and 37%, as shown in Figure 5.1. This is attributed to the *amount* of random actions as a result of not approximating *semantic* actions. The amount of random actions was very large, ranging from 94.35% to 59.96%, whilst at the same time the amount of Semantics-based actions ranged from 5.65% up to 40.04%.

The conclusions drawn from observing the accuracy and action ratios are mostly based on the deduction that Semantic-based actions are not always possible, and when possible they still do not provide robust and accurate actions. The reasons for those conclusions are related to the criteria which must be satisfied in order for semantic actions to be possible.

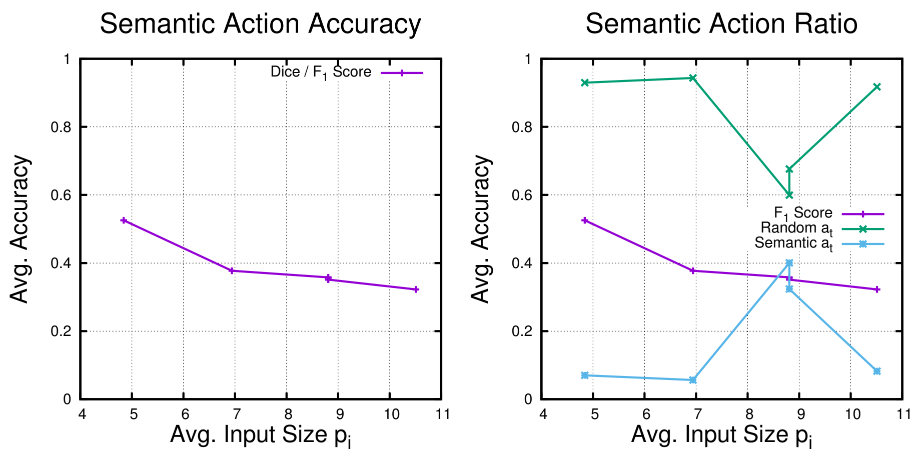


Figure 5.1: Semantic-Heuristic Accuracy.

First and foremost, similar episodes must exist and are thus used by the VSM (see line 1 in pseudo-code 5) a condition which empirically I discovered is rarely satisfied. Furthermore, *when* similar episodes do exist, then (a) there is no guarantee that their graphs are structurally similar, and (b) a different context, graph size or *meaning* does not imply that action substitution is a good action a_t.

Second, as seen in Figure 5.1 there is an unclear correlation between Semantic actions and F_1 score; it is impossible to identify if the random actions inhibit high accuracy, or if both random and Semantic-based actions are inaccurate. The action ratios are complementary to each other, and because of the large amount of random actions, proper measures of Semantic actions are not possible.

Third, the Semantic actions are rarely available, and even if available, they will be mixed with random actions within an episode. Due to their low availability and action candidacy, Algorithms based on such principles appear to be flawed or limited. The factors that limit their availability are:

- Similar episodes may not be exist.

- Identical terms may not exist.

- Similar terms may not exist.

Furthermore, Semantic-Heuristic Algorithms such as the one examined, are usually fallible to the following issues:

- Low similarity episodes may *hinder accuracy.*

- Low relational similarity between terms may also *hinder accuracy.*

- Not all terms are mapped in WordNet, therefore not all terms can be used to obtain relational similarity.
- Even if similar terms exist, semantically swapping them does not guarantee graph cohesion.

From the above results and conclusions, it should be apparent that such algorithms do not favour cognitive systems, even if they first appear to be tailored for them. Furthermore, the grounding problem is an issue, since background knowledge on which those algorithms often rely, is not available. That is not to say they should not be used, but rather that they may be used as a fallback or back-up plan in order to avoid random actions.

5.3 Probability-based Experiments

A large amount of research has focused on Statistics, probabilities and the Algorithms that employ them. Therefore, following the Algorithm presented and tested in Section 5.2, I designed and experimented with a **Statistics and probability** based Algorithm, using the theory from Section 3.6. Specifically, the formulae (3.2) and (3.3) were used to create a look-up table within the Icarus' long-term memory, as shown in Figure 3.1. This Section explains how that algorithm performed in comparison to similar research, using the dataset from Chapter 4 and measuring the accuracy with Dice-Sørensen (see Section 5.1.3).

5.3.1 Implementation

The core idea behind the action controller for approximation of actions a_t, given a state s_t, is to use the *known* statistical probabilities, in order to decide on actions. The actual probabilities are based on observations made during training, and during testing those observations are used to generalise the problem of creating actions for unknown states. Any state s_t given as an input may be known or unknown: *known* if the episode has been experienced before, *unknown* if it is a new episode; in the case of testing, episodes were always unknown, otherwise the agent would simply be iterating policies. The probabilistic approach therefore generalises the entire previous experience of actions a_t for *all episodes*, and the output is verbalised as *performing an action that is generally assumed to be correct.*

The pseudo-code in Algorithm 6 demonstrates how a term t_i is converted into a concept or relation. Preferential combinatorial probability is used if both term and POS tag probabilities exist and have been recorded. The agent uses the highest

Algorithm 6: Probabilistic-Heuristic Node Action.

```
    Input: $s_t$
    Output: $a_t$
 1  if $\exists\Big(p\big(V(term_i)\big)\Big) \wedge \exists\Big(p\big(V(tag_i)\big)\Big)$ then
 2      if $p\big(C(term_i)\big) \cdot p\big(C(tag_i)\big) > p\big(R(term_i)\big) \cdot p\big(R(tag_i)\big)$ then
 3          return $a_t(C)$;
 4      else
 5          return $a_t(R)$;
 6  else if $\exists\Big(p\big(V(term_i)\big)\Big) \wedge \nexists\Big(p\big(V(tag_i)\big)\Big)$ then
 7      if $p\big(C(term_i)\big) > p\big(R(term_i)\big)$ then
 8          return $a_t(C)$;
 9      else
10          return $a_t(R)$;
11  else if $\nexists\Big(p\big(V(term_i)\big)\Big) \wedge \exists\Big(p\big(V(tag_i)\big)\Big)$ then
12      if $p\big(C(tag_i)\big) > p\big(R(tag_i)\big)$ then
13          return $a_t(C)$;
14      else
15          return $a_t(R)$;
16  else
17      return $random\big(V(term_i)\big)$;
```

probability to convert terms to nodes, and in the event that no term probability exists, it falls back to using only the POS tag probability, whereas in the extremely rare event where no POS tag probability exists but a term probability does exist, it will use that instead. Only in the event that neither term nor tag probabilities are known, will the agent use a random action. Unknown or unrecorded probabilities are represented by -1, therefore the signed comparison remains valid.

Similar to the previous Algorithm 6, the pseudo-code in Algorithm 7 uses the probability values from the realisation value of $p(e)$, as shown in (3.2) to decide if an edge should be created or not. The actual realisation can be based upon the *terms* t_i and t_k acting as labels of the respective nodes (concept or relation), or their respective POS tag values. Whilst a combinatorial probability is preferred, a realisation of $p(e)$ may not exist for terms, in which case the agent reverts to using POS tag-based $p(e)$.

The main difference is that in this scenario, in order to use a probability, it must be over a threshold θ which is manually adjusted. The constant θ is required because the agent has to ignore $p(e)$ below a certain value as too small for edge creation. Furthermore, the probability of an edge based on the *normalised distance*

Algorithm 7: Probabilistic-Heuristic Edge Action.

```
    Input: $s_t$
    Output: $a_t$
1   if $\exists\Big(p\big(E_{term_{ik}}\big)\Big) \wedge \exists\Big(p\big(E_{tag_{ik}}\big)\Big)$ then
2       if $p\big(E_{term_{ik}}\big) \cdot p\big(E_{tag_{ik}}\big) \cdot p\Big(E(\Delta_{term_i term_k})\big)\Big) > \theta$ then
3           return $a_t\big(E_{term_{ik}}\big)$;
4   else if $\exists\Big(p\big(E_{term_{ik}}\big)\Big) \wedge \nexists\Big(p\big(E_{tag_{ik}}\big)\Big)$ then
5       if $p\big(E_{term_{ik}}\big) \cdot p\Big(E(\Delta_{term_i term_k})\big)\Big) > \theta$ then
6           return $a_t\big(E_{term_{ik}}\big)$;
7   else if $\nexists\Big(p\big(E_{term_{ik}}\big)\Big) \wedge \exists\Big(p\big(E_{tag_{ik}}\big)\Big)$ then
8       if $p\big(E_{tag_{ik}}\big) \cdot p\Big(E(\Delta_{term_i term_k})\big)\Big) > \theta$ then
9           return $a_t\big(E_{term_{ik}}\big)$;
10  else
11      return $random\big(E_{term_{ik}}\big)$;
```

as described by (3.3) is also employed. This type of probability *ignores* terms, POS tags, their values and labels; it is only concerned with how far within a sentence entities are connected by edges. This distance metric was initially not included, but early empirical testing indicated that when using it, accuracy was increasing. The distance metric is rarely unknown, in which case a -1 is returned, thereby making the final $p(e)$ fall below θ.

In the experiments where raw probability values were used, the FSM-styled conditional statements actively decided if actions were to be performed or not. Because the above Algorithms are able to operate on a *state-action* level, they were used frequently, by being given in a shift-reduce the terms of input pattern p_i, and then the entire set of possible edges based on concept to relation and relation to concept combinations. This approach in essence examines the entire space of possible edges and tries to filter the *good actions* using $p(e)$ and θ.

5.3.2 Probability Space Analysis

The first and foremost observation when experimenting with this Algorithm was the actual mapped probability values and their distribution. Analysing the distribution and visualising the values aided in taking decisions about further development and issues encountered.

As shown in Figure 5.2, all three histograms have a Y axis representing frequency of observations and an X axis representing the actual probability value.

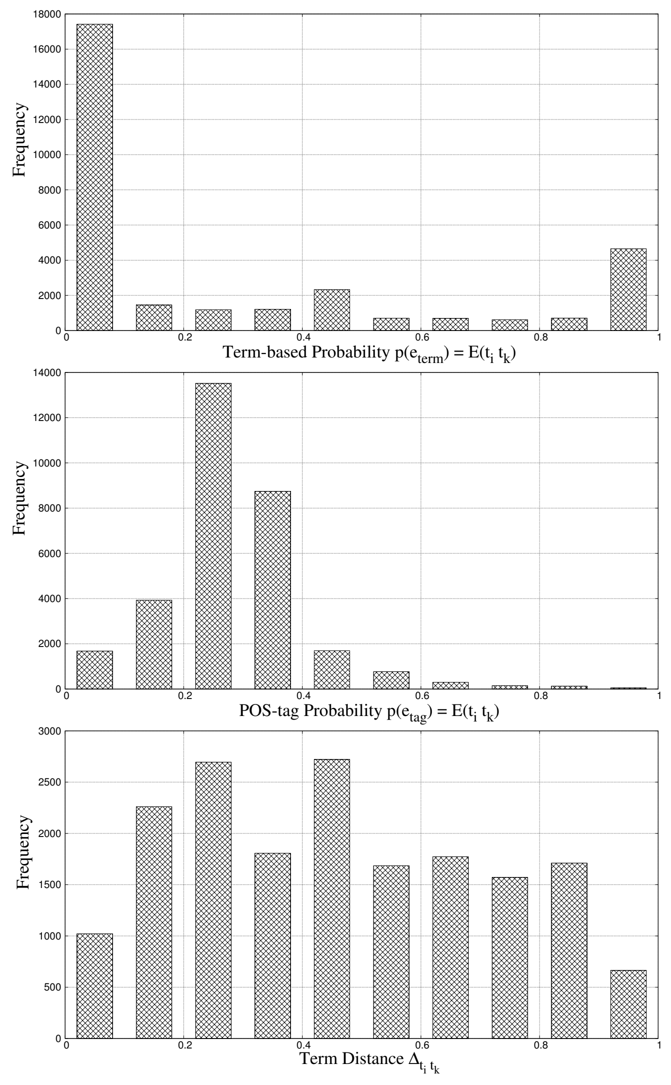


Figure 5.2: Probability Histograms.

The top histogram of $p(e) = E(term_{ik})$ demonstrates that edge samples based on term value mostly indicate which edges are *not valid* edges, e.g., the high frequency of very low $p(e)$ values. This conclusion implies that the probability map acts as a filter itself; through the statistical records it is possible to identify viable from non-viable edges.

The $p(e) = E(tag_{ik})$ histogram (middle one) is the only one which has what resembles a normal distribution bell curve with a positive skew. The actual distribution of $p(e)$ values based on POS tags is perplexing; the majority of edges appear to have a mean $\overline{x} = 27.97\%$ and a standard deviation $\sigma = 11.79\%$. However, the preliminary conclusion drawn from this histogram is that POS tag inferred edges are of low *certainty*, e.g., that there exists a large amount of contradicting or conflicting observations when using a POS tag as the identifier.

The last histogram contains the normalised and scaled distance metric probabilities; it appears to be somewhat homogeneous, with the biggest mass around the *close* to *near close* terms. The conclusion drawn is that edges tend to connect nodes which are very close to somewhat close nodes, rather than the extremes.

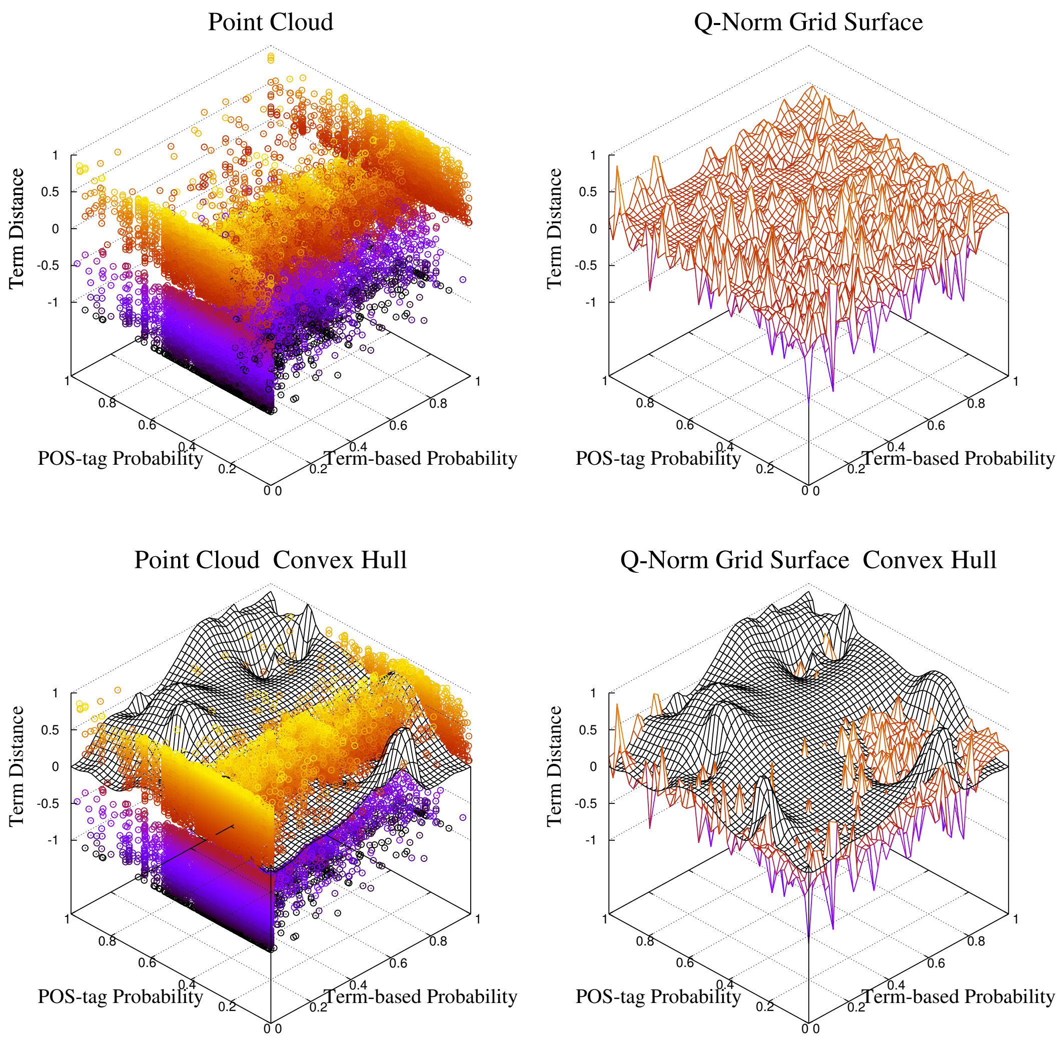


Figure 5.3: Probability Value Map

The Figure 5.3 shows a *point cloud*: recorded triplets/tuples of $p(e_{term})$, $p(e_{tag})$ and $p(e_{\Delta})$, when observing a single action a_t post-training. Because the statistical material is obtained not during but after training, the agent is capable of discovering which action a_t had the respective triplet values. Of those actions

a_t, some are by permutation and generate the "*bad*" samples recorded as $p(e) = 0$, whilst the ones found within the actual paradigm episodes are the "*good*" samples recorded as $p(e) \geq 0$.

All **purple to black** points used a negative term distance and are thus always invalid; **red to orange** points used a positive term distance and therefore *may be valid actions.* The sample space of cloud points in Figure 5.3 shows: in the top left corner, the point cloud; in the top right corner, a point cloud normalised as a Q-Norm grid surface; in the bottom left corner, the point cloud separated by a convex hull [Chazelle, 1993]; and in the bottom right corner, the Q-Norm grid surface, separated by the same convex hull. What the Figure 5.3 really demonstrates is that the probability triplets recorded post-training are *non-linearly* separable, as shown by the convex hull [Toussaint, 1983]. Furthermore, the largest mass (and as shown in the histograms earlier) is low-probability values associated with "*good*" actions a_t. This is mostly due to the contradictory nature of POS-tag based edges; isolated peaks of actions can be seen along the *ridges* of 0.6 POS-tag probabilities moving across the entire spectrum of term-based probabilities (mostly shown in the Q-norm surface grid, the top right plot). Yet another *ridge* can be identified originating at X: 0.2 and Y: 0.8 to 1.0, and moving along the X axis (where X axis is the POS-tag values and Y axis is the term-based values). This is a reasonable observation, since it implies that high term-based probabilities (Y Axis) persist regardless of the POS-tag probability value (X axis).

Ultimately, what is shown by the cloud point is that a triplet of probability values acquired from a single action a_t may define a suitable action in a new episode; however each probability value may not be enough as it is a partial descriptor of an edge under examination. Using only POS-tag based values may be too generalising and reduces accuracy; similarly term-based probabilities are not always available. Distance metrics on their own do not show the big picture are are used only as an aid to further improve the agent. The largest mass of extracted probabilities belong to the range of $0.2 \geq p(e_{tag}) \leq 0.6$, whereas $p(e_{term})$ has a very broad spectrum.

5.3.3 Results

First and foremost it must be noted that experiments with probability-based action selection were not carried out using only probability-based actions. A transition from Semantic-based to Probability-based actions was examined; which type of action should be preferred, what happens when the agent mixes different types of actions, what is the ratio of Semantics to Probability based actions, how to best fine-tune and optimise the θ parameters, and what are the results when compared to a *random* *walk*. Hereinafter I provide a thorough analysis of the results and describe that transition.

The first and most important factor is that all experiments carried out whilst optimising and adjusting algorithm preference, probability value filters and other parameters were carried out using the same sub-set of the dataset, with a limit of 6 input terms. The reason for taking this approach was that of practicality: training and evaluating using a smaller data set was faster and easier in order to optimise and draw conclusions.

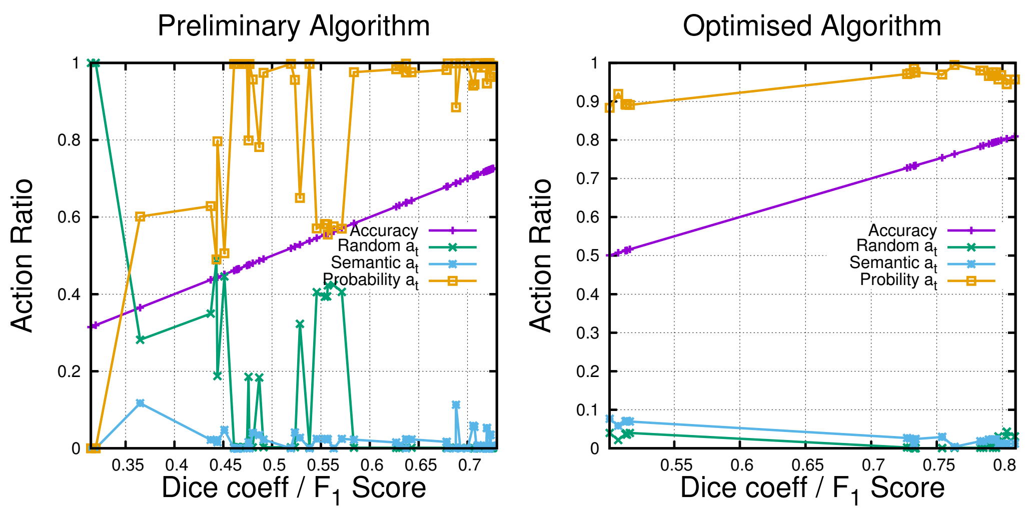


Figure 5.4: Probability Accuracy and Action Ratios.

First I used a *random walk* [Weiss, 2005], e.g., an action-selection mechanism using only random actions a_t throughout every episode. The actual performance when only random actions were taken ranged from 31% to 32%. During initial experiments, as shown in the left plot in Figure 5.4, there was a lot of variance in action ratios and accuracy was low, ranging from 31.56% to 72.62% accuracy. There is a clear trend of accuracy increase when random actions are kept to a minimum, but the effect of semantic actions in the left plot is inconclusive. It should be noted that in the early experiments an *edge count* probability was also included; the probability that a node would have a specific amount of edges, with more edges than what was generally acceptable. This was recorded in a similar fashion as was metric distance between terms.

Removing the *edge count* probability, using only the tripled values as aforementioned and using probability-based actions before Semantic-based or random actions increased accuracy and lowered uncertainty regarding action ratios. That is reflected in the right plot in Figure 5.4 with an increase in accuracy from 50.08%

to 80.95%, and the reduction of random actions. However there is another trend shown in the the right plot: as the amount of Semantic actions is reduced, so does the accuracy increase; therefore it is sound to assert that Semantic-based action selection is not as good as probability-based actions.

Deciding on the actual θ values for filtering $p(e)$ values is a major task; albeit it might first appear as a simple optimisation task, it in fact isn't. The probability space isn't linearly separable and using actual $p(e)$ values as described in this Algorithm creates questions: (a) what should each θ for every $p(e)$ value be? (b) should it be a constant or an adaptive value, (c) does θ influence accuracy, and if it does how exactly?

During initial investigation using permutations of θ values indicated that they do affect accuracy. However, after careful examination I was able to determine that they only acted as an indicator; this as shown in earlier Figure 5.3 is a non-linear space, and thus using constant θ values is not a plausible solution.

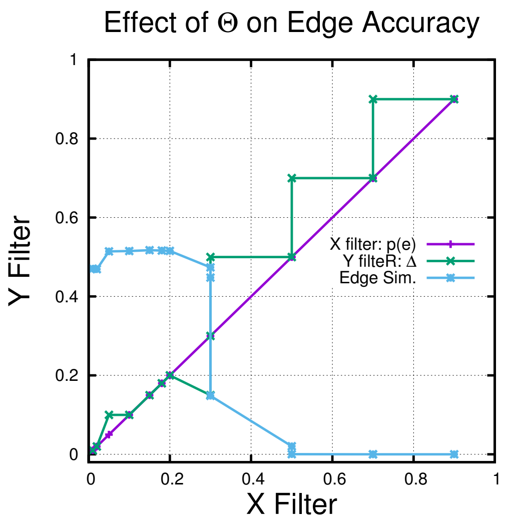


Figure 5.5: Filtering with θ

Because the $p(e)$ value space appears do be non-linear (as discussed in Section 5.3.2), therefore there does not exist a parametric solution to the problem of finding appropriate θ values. I did not use sophisticated separability identification methods [Elizondo, 2006] because some of those involve the use of machine learning (ML) models; therefore it appeared a better option to implement ML models instead of using a set of θ parameters.

Furthermore, the values extracted are a triplet, a combination of values; therefore the problem is not necessarily to filter individual values but their ensemble, thereby justifying more complex real-valued processing models. For example, a large portion of the samples obtained from statistical observation is spread across a range of $p(e_{term})$ values, whilst combined with a more *normally distributed* range of $p(e_{tag}$ values (see previous Section 5.3.2). The implication thereby is that any system dealing with such data should be able to identify specific patterns or features in the input and adapt to it. In comparison to edge creation, node creation was very accurate; it averaged 96.90% similar nodes, and relies on POS Tag accuracy (laPOS has a claimed 97.22% on the WSJ corpus) and *term-based* node frequency.

5.3.4 Discussion & Conclusion

The results, **albeit not directly comparable** to other research due to differences in *scoring* and **datasets** used, showcase that Icarus accuracy is relative to similar statistical and probabilistic work, shown in Table 5.2. Older systems and research is not shown, and albeit the Algorithm's results ranged from 50.08% to 80.95% it is important to note the vast differences between datasets.

Author	**System**	$F_1 Score$
Chen et al [Chen and Mooney, 2008]	WASP	72.00%
Chen et al [David L. Chen, 2010]	WASP	76.77%
Vlachos et al [Vlachos, 2012]	DAGGER	78.90%

Table 5.2: Statistic & Probability Oriented Research.

The presented Algorithm is a *generative* rather than a *discriminative* approach [Jordan, 2002], and as such it generalises. A more pragmatic approach taking into consideration conditional probabilities based on *previous edges* would most likely increase the accuracy, albeit that assertion remains unproven. An issue with the Algorithm presented here is that it does not take into consideration previous probabilities. Continuous Bag of words (CBOW) [Mikolov et al., 2013c] and Skip-Gram [Guthrie et al., 2006] do address such issues, but require a vastly different Algorithm implementation (feature vectoring).

Using cascading action-selection did work as intended to a degree; however as analysed earlier it is not a very good solution. Whereas Semantic-based actions are worse than probability-based ones, it is questionable how much better than random actions they actually are. Last but not least, the assumption that the triplet of values contains enough information to make good decisions is not proven: whereas at first I used four values: distance, edge count, $p(e_{term})$ and $p(e_{tag})$, eventually edge count was removed. Other issues arise when $p(e_{term})$ is not available, when nodes have been wrongly converted thus prohibiting creation of correct edges, and from the contradictory nature of $p(e_{tag})$.

5.4 Shallow Neural Experiments

Following the probability-based experiments and the conclusions derived from the results presented (Sections 5.3.3 and 5.3.4), I decided to use artificial neural networks (ANN) in order to select actions for new episodes. That decision was taken in light of the θ parametrisation issue discussed earlier; it appeared to be a *classification* and *approximation* process which in line with the general cognitive agent theory (discussed in Sections 3.5, 3.8 and based on the underpinnings of Bach, Haikonen and Icarus, as concluded in Sections 3.13.2, 3.13.3 and 3.13.5) was an associative memory task. To date the best associate memory models are neural networks, and therefore the use of ANN for classification and approximation is justified.

The Icarus agent, was once again evaluated using the dataset described in Chapter 4 and accuracy was measured with Dice-Sørensen (see Section 5.1.3). However, this set of experiments was compared to more recent research, some of which is considered state-of-the-art. The research cited hereinafter uses different datasets, often with larger sentences; yet what is important is the underlying mechanisms used are very similar (e.g., neural networks and parsing based on POS, Shift-Reduce, etc.).

5.4.1 Implementation

The first implementation of ANN was done using the FANN[1] library[Nissen, 2003]. I also developed a CUDA/*GPU*[2] ANN library[2] for my intents and purposes, but abandoned it once I realised that it would require me to spend more than a year for development. The final ANN implementation I used is OpenANN[3] [Fabisch et al., 2013], which is CPU-based but uses multi-core processing.

The action-selection mechanism when using ANN is non-cascading as the previous Algorithms, but a standalone process; the agent relies **only** on the ANN to select actions and does not fall back to other Algorithms. The actual input scheme is a *distributed* encoder: the actual input I_j is a real-valued vector representing:

1 The term-based edge probability value $p(e_{term}) = E(t_i t_k)$ as shown in (3.2).

2 The POS tag-based edge probability value $p(e_{tag}) = E(t_i t_k)$.

3 The min-max normalised and scaled term distance value of $\Delta_{t_i t_k}$ as shown in (5.7).

[1]`http://leenissen.dk/fann/`
[2]`https://github.com/alexge233/cuANN`
[3]`https://github.com/OpenANN/OpenANN`

The difference in this Algorithm is that the agent does **not** use a probability on distance observations, but the actual distance which is scaled and normalised depending on the input pattern p_i size.

$$\hat{\delta}_{t_i t_k} = \frac{\Delta_{t_i t_k} - min(p_i)}{max(p_i) - min(p_i)}. \tag{5.7}$$

Therefore the ANN input $I_j = \left[p(e_{term}), p(e_{tag}), \hat{\delta}_{t_i t_k}\right]$ is a *distributed* input vector based on observations of a statistical nature. Furthermore, the input values are all real values x, so that $0 \geq x \leq 1$, and because the probability values are within the same range they can be used directly.

In the event that a tuple of terms is unknown (e.g., one or both of the terms have never been observed before) then a secondary ANN will decide on the action, using as input $I_j = \left[p(e_{tag}), \hat{\delta}_{t_i t_k}\right]$. This introduces the issue of training two neural networks instead of one; the assertion that the same ANN could be used by replacing $p(e_{term}) = 0$ is erroneous because it implies that an unknown probability for a pair of terms is an action to be avoided (which may or may not be the case).

Algorithm 8: Neural Network Action Selector.

```
Input: $e(t_i t_k)$
Output: $a_t$
1 if $\left(\exists(p(e_{tag}) = e(t_i t_k))\right) \wedge \left(\exists(p(e_{term}) = e(t_i t_k))\right)$ then
2     $\hat{y}_i = f\left([p(e_{tag}), p(e_{term}), \hat{\delta}_{t_i t_k}]\right) \cdot Wi$;
3     if $\hat{y}_i[0] > \hat{y}_i[1]$ then
4         return $A_t^+$; $a_t = e(t_i t_k)$;
5 else if $\left(\nexists(p(e_{term}) = e(t_i t_k))\right) \wedge \left(\exists(p(e_{tag}) == e(t_i t_k))\right)$ then
6     $\hat{y}_i = f\left([p(e_{tag}), \hat{\delta}_{t_i t_k}]\right) \cdot Wi$;
7     if $\hat{y}_i[0] > \hat{y}_i[1]$ then
8         return $A_t^+$; $a_t = e(t_i t_k)$;
```

The pseudo-code in Algorithm 8 shows a *summary* of the process using a shallow neural network to classify information as candidate actions a_t. It creates all possible edges in a set E_t such that it contains all the possible combinations between t_i and t_k (as well as t_k and t_i). It then proceeds to classify each possible edge $e(t_i t_k)$ as a *potential* action.

Forward propagation is used, denoted as $f(I_i) \cdot W_i)$ and replacing the output of previous layers as $\sum(O_j) = f(I_i)$; this is a two-step process in a shallow network with one input layer, one hidden layer and one output layer. The input layer is made up of three input nodes (each one receiving as value one of the items in I_i)

or in the case of the second smaller network, of two input nodes, a variable amount of hidden nodes (which has been optimised and discussed later on) and two output nodes.

Each output node represents the *likely-hood* of either executing action a_t or not executing it. The Algorithm 8 indicates that if the first value in the output vector is larger than the second, then the agent should execute that action (by adding it in the set of actions A_t). This is consistent across all neural network implementations in Icarus; the first output node represents the action and the second an inaction. The activation function f (as described earlier in Section 2.4.1) can be either a *sigmoid* or a *tanh* function; the output action is a *soft-max* since they do best with classification tasks [Glorot and Bengio, 2010].

Throughout the shallow neural network experiments I tried various network architectures with a variable number of hidden nodes. An anecdotal formula circulating the web [Stackexchange, 2015] as shown in (5.8) was used; the formula itself appears to have been devised using findings [Sheela and Deepa, 2013] when researching methods to find fixed number of hidden neurons. The task of hyper-parametrisation of the ANN is a complex one and outside the scope of this thesis, yet it directly affects the outcomes of the Icarus agent.

$$N_h = \frac{N_s}{alpha \cdot (N_i + N_o)}. \tag{5.8}$$

The above equation (5.8) simply serves as a starting point for finding the amount of hidden nodes; N_h are the hidden nodes, N_i the input nodes, N_o the output nodes and *alpha* is a constant for adjusting/discounting the hidden ones. Other parametrisation factors are the activation function: I experimented using both sigmoid and tanh and eventually ended up with sigmoid for shallow networks.

Training the ANN was done after the agent iterates and processes the set of conceptual graphs used as paradigms from which to learn (see Section 3.2); first it must data-mine the statistics as described in Sections 3.6 and 5.3, and only then can it be trained using the data acquired.

Thus a cyclic process is formed: the agent learns via MDP and Q-learning, it extracts meta-data and trains the ANN, and finally the ANN is used to explore unknown policies for new MDPs for which no $Q(s_t, a_t)$ exist. This group of processes is shown in Figure 5.6 and is part of both the Icarus engine, as well as the core that drives the use of ANN for Icarus: ANN can't be trained without the MDP and Statistics memory, and the MDP explores non-random actions by using the ANN as classifier of actions. The Algorithm that produces the training samples for the

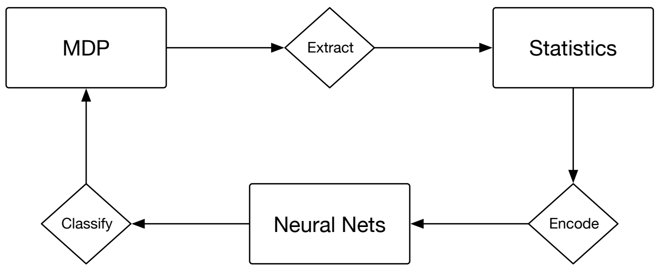


Figure 5.6: Cyclic Process: MDP to Statistics to ANN

ANN is shown in the pseudo-code in Algorithm 9, with a similar combinatorial loop as in Algorithm 8, lines 2 to 7.

That process is run after the statistics memory has been populated; it iterates possible actions and known "*good*" actions, and then creates a training sample which associates the underpinning input data which produces an action a_t. Because the sample space S contains both "*good*" and "*bad*" actions, it creates random actions by trying various combinations of nodes n_i and n_k found within the ideal graph G_i^y created for input pattern p_i.

Algorithm 9: Neural Network Training Kernel.

```
   Input: $p_i$
   Output: $S = \{\cdots\}$
1  $E_{ik} = \{\cdots\}$;
2  for $n_i \in G_i^y(p_i)$ do
3      for $n_k \in G_i^y(p_i)$ do
4          $E_{ik}^+; e(t_i t_k)$;
5  $\cdots$
6  for $Q(s_t, a_t) \in MDP(p_i)$ do
7      if $\exists(a_t) \in E_{ik}, a_t = e(n_i n_k)$ then
8          $S_i^+; \{I_i = [p(e_{tag}), p(e_{term}), \hat{\delta}_{n_i n_k}] \rightarrow y_i = [1, 0]\}$;
9      else if $\nexists(a_t) \in E_{ik}$ then
10         $S_i^+; \{I_i = [p(e_{tag}), p(e_{term}), \hat{\delta}_{n_i n_k}] \rightarrow y_i = [0, 1]\}$;
11 return $S$;
```

It then proceeds to *filter* the known actions which exist in the episode of p_i and which are associated with edges. For each action found and associated with an edge the sample S adds a new pair of vectors: I_i as the input and y_i as the output. This step is repeated for all episodes in memory, thereby creating a sample set which is proportionate to the possible edge combinations for each output graph G_i^y.

5.4.2 Results

Experimenting with a shallow ANN resulted in substantially better accuracy than all previous Algorithms and experiments (Sections 5.2 and 5.3). However, prior to reporting the actual results, a thorough analysis of why the ANN performed better is given.

Classification

The ANN used acts as a *classifier*; it has been trained on classification of "*good*" actions by receiving information which characterise them. As such, it has the ability to filter what it has learnt to be appropriate actions, with a certain degree of error. Generalisation is a useful attribute of ANN, and the most common issues relate to under-fitting (generalising too much, thus reducing accuracy) and over-fitting (adapting to the training material too much, thus being unable to generalise). The amount of training samples directly affects the ANN; in the Figure 5.7 the actual data space (probabilities and term distances) have been mapped by an ANN to showcase how it operates.

The *green* areas of the plots showcase the "*good*" (e.g., high value) actions as defined by $Q(s_t, a_t)$, whereas the yellow and orange areas are low value actions, and red areas are negative value actions. The *spikes* seen in the plots are distance increases between terms labelling nodes; they signify how certain large distance quantities in the action information triplets/tuples (see Section 5.3.2) are valid actions, e.g., outliers in the otherwise homogeneous low quality action area surrounding them. The inverse spikes (facing downwards) signify the opposite; that inverse edges (e.g., from $e(n_k n_i)$ instead of $e(n_i n_k)$) are negative valued actions.

There are four different plots in Figure 5.7 in order to demonstrate how ANN works in the Icarus scenario; it *maps* the data space described in Section 5.3.2 by being trained with meta-data samples. As the sample size increases (e.g., by using larger data-sets) then more information can be classified, and thus the agent becomes more accurate. At the last plot (bottom right corner) with 30,933 samples the agent is able to deal with the information obtained during *evaluation* using both classification, and the approximating nature of the ANN (e.g., classifying highly similar values).

Therefore the ANN solved the issue of providing a non-linear solution to the underlying meta-data of triplets/tuples characterising a decision which results in a

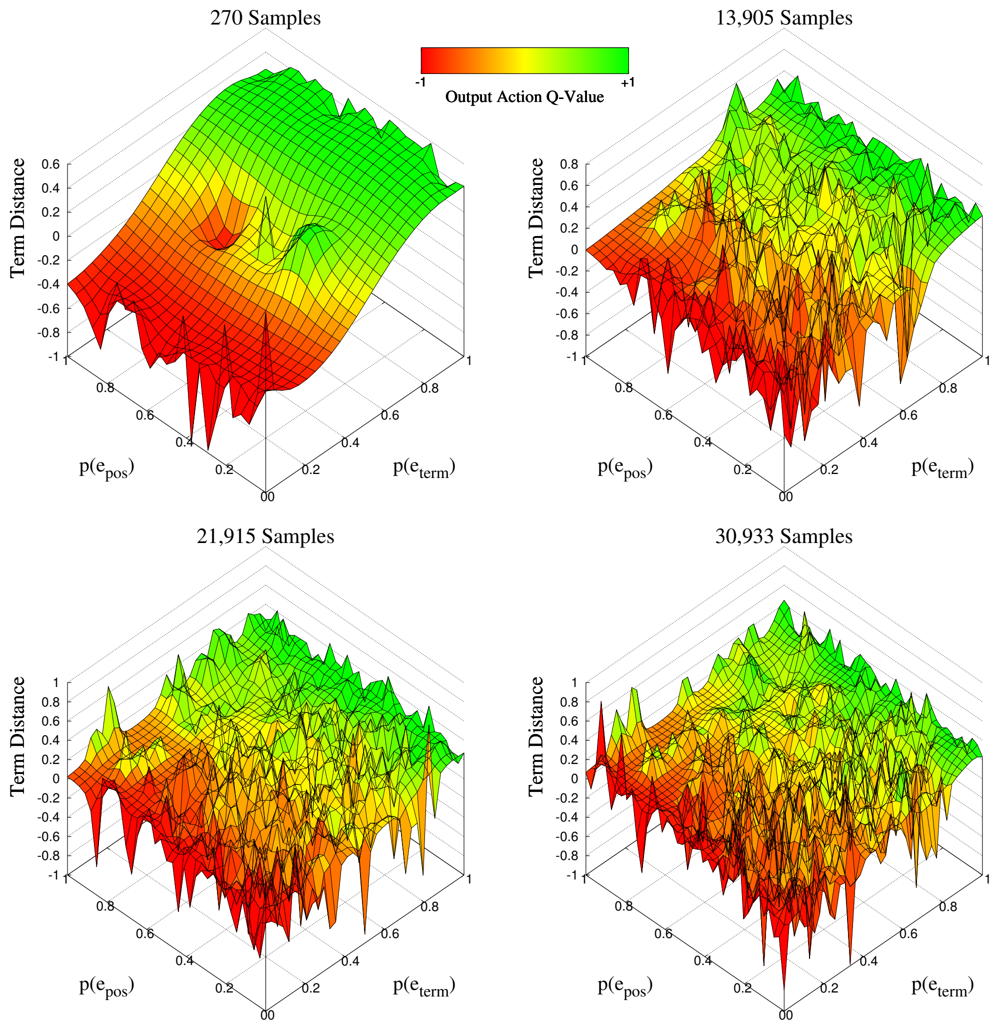


Figure 5.7: ANN Classifying and Mapping Search Space.

good or bad action a_t. Whilst they require training and careful hyper-parameter optimisation, they are indeed a viable solution. The actual ANN implementation used 3 input nodes, 320 hidden nodes, 2 output nodes, and a Sigmoid activation function. Variations were examined, such as networks using Tanh activation, a different number of hidden nodes (see formula 5.8), and the ANN with the best empirically obtained accuracy was re-used.

Accuracy

Using the metric of Dice-Sørensen coefficient/F_1 Score, the recorded the L1, L2 and L3 RBD results are shown in Figure 5.8 with all data averaged, as well as including

Jaccard coefficient, and the graph component accuracy: node and edge similarity.

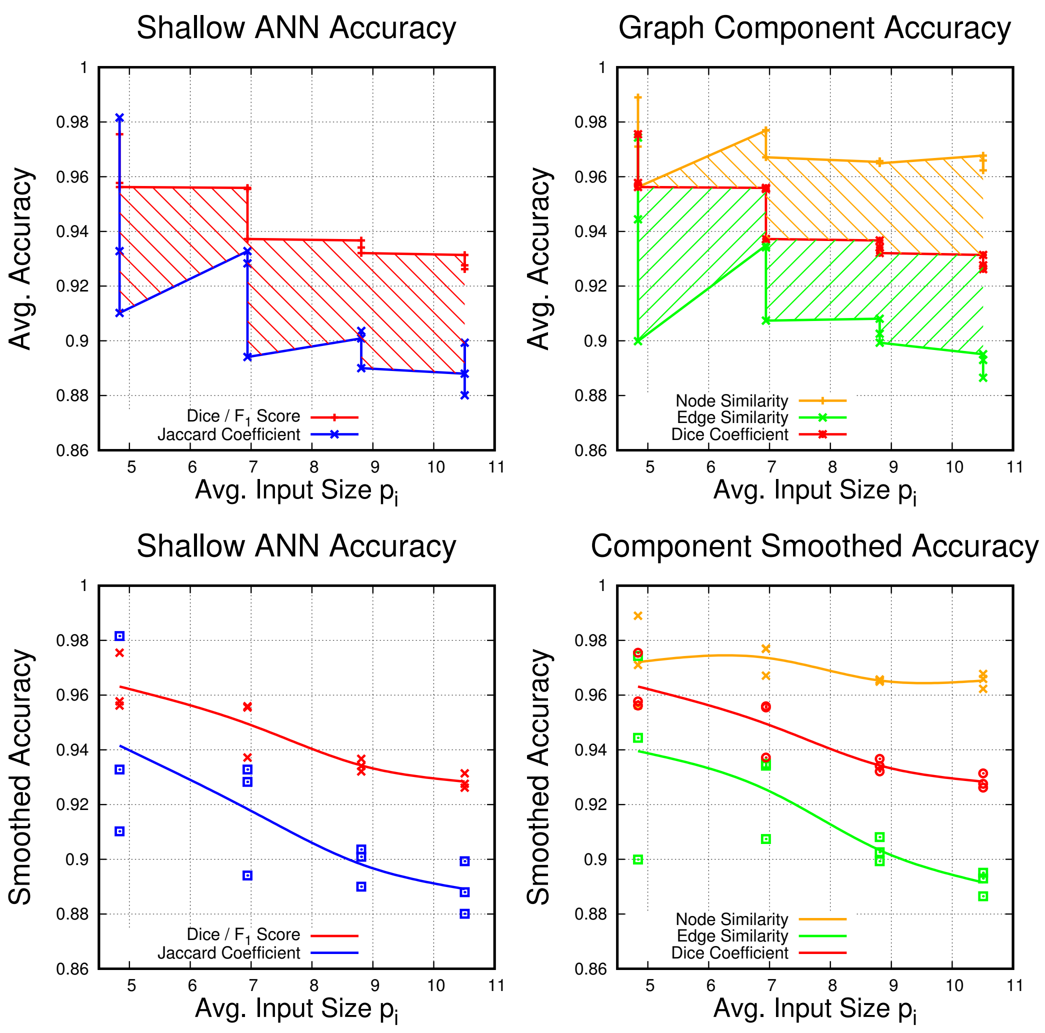


Figure 5.8: ANN Accuracy.

The deterioration of *edge* accuracy due to a decrease in node accuracy is shown, a key factor affecting the overall agent accuracy. Also, in-line with other research reports, increased input sentence/pattern size p_i reduces overall accuracy [Choi et al., 2015]. The clear difference between Jaccard and Dice/F_1 Score can be seen in the right-side plot; as the difference in ideal and actual graphs increases (not just missing edges, but edges which shouldn't exist) the score is penalised. Another observation is that node similarity slightly increases when processing larger input, prior to decreasing, resulting in a slight edge increase. As node accuracy remains stable towards larger size input, edge accuracy begins to suffer; the conclusion is that whilst node creation actions remain accurate, edge actions become increasingly more complicated and hard. The filled area between Dice and Jaccard coefficients serves the purpose of demonstrating how those two measures differ. Similarly, the filled area between Node and Dice as well as Edge and Dice coefficient demonstrate how the overall accuracy decreases due to edge similarity decreasing whilst node accuracy remains constant.

The accuracy of various tools as reported by Choi et al [Choi et al., 2015, pp 391] uses UAS and LAS. Because the notion of unlabelled accuracy does not exist, and since LAS encompasses both arcs (edges) and labels (nodes), the comparison following is with respect to Dice-Sørensen/F_1 Score and not to UAS. I compare to *projective trees* [McDonald et al., 2005b] since English is the language used in this thesis and historically projective trees have been used for English. Furthermore all frameworks appear to result in a better performance for projective trees but also use non-projective trees, therefore for completeness I provide that information as well towards the Jaccard coefficient (and not the Dice–Sørensen).

As described by Choi et al [Choi et al., 2015] most parsers report the aforementioned accuracy for sentences more than 10 terms/words long; therefore a comparison could be made only respective to the smaller input size (e.g., up to 11 words), as shown earlier in Section 4.1.2, Figure 4.2. Furthermore, since the aforementioned research uses other datasets, a direct comparison is not possible.

Choi et al mention that most parsers have a UAS accuracy of 93.49 to 95.5 for sentences under 10 terms, which declines to 81.66 and 86.61 for sentences larger than 50 terms; in comparison the L1 random datasets used for testing Icarus averages smaller input size (see Figure 4.2). Comparing large input datasets sees a drop in accuracy, yet it hasn't been possible to correlate if that drop in Icarus is consistent with the drop reported by other frameworks and research.

In terms of *exact match* accuracy (EM) the Icarus shallow ANN performance averaged a 61.65 per cent, slightly above the averages reported by Choi et al [Choi et al., 2015] which are 58.36 EM for UAS, taking into consideration the smaller input size and difference of datasets. However, the work done by Choi et al does not include Google's Syntaxnet [Andor et al., 2016; Weiss et al., 2015] which has provided the best to date F_1 scores of 94.44% on news data and 95.40% on question-answer data. The approach described by SyntaxNet is very similar with only one major difference: the use of feature vectors rather than distributed encoders.

Ultimately what can be concluded from the shallow ANN experiments and the agent implementation, is that it is quite similar to state-of-the-art performance; Section 2.7.6 and framework Tables 2.1 demonstrate a range of F_1 of 90% to 95.40%.

Although using non-directly comparable datasets due to input size differences, CG have an increased complexity due to partitioning nodes and then classifying edges. Whilst that does not ameliorate the complexity increase when dealing with larger input sentences, it is nonetheless a critical factor that should be taken into consideration.

Interestingly, Google's research [Andor et al., 2016] indicates that shallow ANN tend to perform just as good, if not better than more complex deep networks, a design which I explore in the next Section 5.5. Regardless of the somewhat simplistic design and scheme, a well tuned ANN is able to accurately create new actions, without the need to be retrained due to the distributed encoding nature; the metadata can be updated via Statistics and therefore the generalising and approximating nature of ANN allow their continuous use.

5.5 Deep Learning Experiments

The field of deep learning has been a trending topic in the past 8 years; albeit deep learning is a field of machine learning, it is ANN re-branded for use with GP^{2}U, nVidia's CUDA and a family of new activation functions and training algorithms (see Section 2.4.3 for a thorough list). This notion doesn't imply that traditional ANN and deep learning are the same; they do share a common origin and structure, but as models, algorithms and kernels they differ (as discussed in Sections 2.4.1 and 2.4.3). Because of the encouraging results from shallow ANN (Section 5.4.2) I decided to implement a deep learning module which would focus on the structured and hierarchical nature of NLU and CG. Similar to the previous sections, Icarus was tested using the dataset from Chapter 4 and measured using Dice-Sørensen (see Section 5.1.3) Following is the description of the implementation and the results.

5.5.1 Implementation

The actual design and implementation of deep learning uses a cascade of deep networks, as shown in Figure 3.10 and discussed in Section 3.9. The idea behind that approach is that the outer deep networks classify sparsely encoded vectors (see formulae 2.29 and 2.30) which represent a term t_i as an item with the lexicon vector; combining two of those sparse vectors produces a pair/tuple of an *edge* (or arc) such that $I_i = V_{m+n}$ where m is the size of the concept lexicon and n is the size of the relation lexicon. Those are highly sparse vectors: only one value is usually activated, and two different networks are needed, one for edges from concepts to relations, and one for edges from relations to concepts.

Algorithm 10: Deep Network Training Kernel.

```
    Input: $p_i$
    Output: $S = \{\cdots\}$
 1  $E_{ik} = \{\cdots\}$;
 2  for $n_i \in G_i^y(p_i)$ do
 3      for $n_k \in G_i^y(p_i)$ do
 4          $E_{ik}^{+}; e(t_i t_k)$;
 5  $\cdots$
 6  for $Q(s_t, a_t) \in MDP(p_i)$ do
 7      if $\exists(a_t) \in E_{ik}, a_t = e(n_i n_k)$ then
 8          $S_i^{+}; \{I_i = [V_m(t_i), V_n(t_k)] \rightarrow y_i = [1, 0]\}$;
 9      else if $\nexists(a_t) \in E_{ik}$ then
10          $S_i^{+}; \{I_i = [V_m(t_i), V_n(t_k)] \rightarrow y_i = [0, 1]\}$;
11  return $S$;
```

First the outer networks are trained on term-based edges and POS tag-based edges, then the inner networks are trained on classifying the encoder network output as a viable or non-viable action a_t. That approach aims to detect and learn characteristics of the triplets (term-based vectors, POS-based vectors and term distances) as sparse input, rather than distributed encoded input. The deep learning training Algorithm is shown in the pseudo-code in 10; the process is similar to the one described earlier in Algorithm 9.

The shallow ANN created two matrices with respective columns m and rows n, e.g., $I_{m,n}$ for input and Y_m, n for output with $I_{m=3}$ for input matrix and $Y_{m=2}$ for output matrix and n being the number of samples. The deep training sampling Algorithm uses input $I_{m=\|C\|+\|R\|}$ e.g., the width of the input matrix I which equals the cardinality of the concepts and relations lexicons (assuming the deep network for concept to relation edges). The relation to concept network has an input $I_{m=\|R\|+\|C\|}$ columns, whilst the n remains the number of samples and output matrix $Y_{m=2}$ is the same. Therefore shallow ANN process triplets acquired from probabilities, deep networks process features, or most specific, edges based on features as well as probability triplets.

The output of each network is the likelihood using soft-max, as described by equation (2.19) and all hidden layers and nodes use the ReLU function (equation 2.18); that combination is widely used and accepted as a good solution for fast learning [Tomczak, 2015; Dahl et al., 2013; Nair and Hinton, 2010]. Most networks were optimised manually over a period of a year; various combinations of hidden layers and hidden nodes were tried with a simple but effective methodological approach: small adjustments and re-training until the cross-evaluation accuracy peaked.

During evaluation, if a term t_i or t_k is unknown, e.g., not indexed in the relations or concepts lexicons, a fallback mechanism using the semantic distance as described in Section 3.7, and 3.10. Therefore, during training of the outer networks the input vector is sparse with a binary activation; this scheme *may be* replaced by a real valued sparse input vector if the term being processed is unknown. The actual value replacing the binary feature is derived from formula (3.8) but inverted so that the closer to zero it is it will represent no similarity, and the closer to one it is it will represent high similarity. The updated formula is shown in (5.9).

$$\begin{aligned} \tilde{v}_{t_i t_k} &= 1 - \tilde{v}_{t_i t_k} \\ &= \{\tilde{v}_{t_i t_k} \in \mathbb{R} \| 0 \geq \tilde{v}_{t_i t_k} \geq 1\}. \end{aligned} \tag{5.9}$$

This conversion affects sparse vectors, where the edge of nodes labelled by t_i and t_k indexes an unseen or unknown term (t_i) which has a known semantic relation to another term t_j. Because many similar terms may exist, the Algorithm will always chose the topmost similar one (if multiple topmost similar ones, then the first one). As a result, the input vector I_{ik} instead of being a binary vector, is transformed to a real-valued one as shown in (5.10).

$$I_{ik}(t_i|t_j) = \Big[0, 0, max(\tilde{v}_{t_i t_j}), 1, \ldots, 0\Big]. \tag{5.10}$$

The pseudo-code in Algorithm 11 demonstrates the high-level logic behind this approach; it tries to use semantic-based actions rather than syntactic-based actions. The Algorithm uses a similar approach to the shallow ANN described in Section 5.4; the difference is that instead of using the triplet values the input is a sparse vector.

An edge from concept to a relation is defined as $e^{C|R}$ whereas an edge from a relation to a concept is defined as $e^{R|C}$. Thus, two different networks are used as aforementioned, by concatenating the lexicon/set indexing concepts and relations or vice versa. In the event that the sum of the vectorised input $\sum \hat{y}_{l=1}^{term}$ is greater than one, then the assertion is that the vector can be used by the network; a vector with a sum equal to one or less is not usable, and the Algorithm will fallback to using POS tags only. Any value less than one implies that (a) the vector is not binary, and (b) it may be a vector with two real valued indexes with less than 0.5 similarity.

The first layer of networks (input layer denoted by subscript $l = 1$) processes only sparse vectors and produces in turn the output $\hat{y}_{l=1}$ with each network corresponding to a either term or POS tag vectors. The second layer of networks (inner layer denoted by subscript $l = 2$) processes the output of the previous networks; thus two networks are used; one for both term and POS network output as well as distance metric, and one for only POS network output and distance metric. The final output $\hat{y}_{l=2}$ is a classification likelihood: if the first value is larger than the second, then the network cascade associates that information with an action.

A variety of training algorithms were used, such as MBSGD, LBFGS, Conjugate Gradient and LMA; ultimately it was MBSGD which provided the best results with careful fine-tuning. Most networks have four to six hidden layers, and around 350 hidden nodes, learning rates were kept small (0.01 to 0.1) from empirical testing and optimisation, and the networks with the best accuracy in cross-evaluation were re-used.

A methodology of an RBD was used when comparing deep networks, mostly in order to determine their comparison to sparse and non-sparse vector input; as such I tried a combination of sparsely encoded vector input and distributed input.

Algorithm 11: Deep Network Evaluation.

```
   Input: $e(t_i t_k)$
   Output: $a_t$
1  if $\left(e^{C|R}(t_i t_k)\right)$ then
2      $I_{l=1}^{term;C|R} = [C_i(t_i), R_k(t_k)]$;
3      $\hat{y}_{l=1}^{term} = f(I_{ik}^{term;C|R})$;
4  else if $\left(e^{R|C}(t_i t_k)\right)$ then
5      $I_{l=1}^{term;R|C} = [R_i(t_i), C_k(t_k)]$;
6      $\hat{y}_{l=1}^{term} = f(I_{ik}^{term;R|C})$;
7  $I_{l=1}^{POS} = [POS(t_i), POS(t_k)]$;
8  $\hat{y}_{l=1}^{POS} = f(I_{ik}^{POS})$;
9  if $\left(\sum \hat{y}_{l=1}^{term} > 1\right)$ then
10     $I_{l=2} = [\hat{y}_{l=1}^{term}, \hat{y}_{l=1}^{term}, \hat{\delta}_{t_i t_k}]$;
11     $\hat{y}_{l=2} = f(I_{ik})$;
12 else if $\left(\sum \hat{y}_{l=1}^{term} \leq 1\right)$ then
13     $I_{l=2} = [\hat{y}_{l=1}^{POS}, \hat{\delta}_{t_i t_k}]$;
14     $\hat{y}_{l=2} = f(I_{ik})$;
15 if $\left(\hat{y}_{l=2}[0] > \hat{y}_{l=2}[1]\right)$ then
16     Return $a_t = e(t_i t_k)$;
```

5.5.2 Results

The first and foremost observation is that training deep networks is considerably harder than training shallow networks, achieving sufficient accuracy has been very difficult. Part of the difficulty arises from the sampling kernel as described in Algorithm 10; it allows for *duplicate* samples which may have contradictory outputs. This as a result made accurate training very difficult with networks such as the POS network constantly under-fitting and having a cross evaluation accuracy of 79%. The term-based networks performed considerably better at 99% accuracy. However, the error from the POS propagated to the second layer networks, and as such did not perform well.

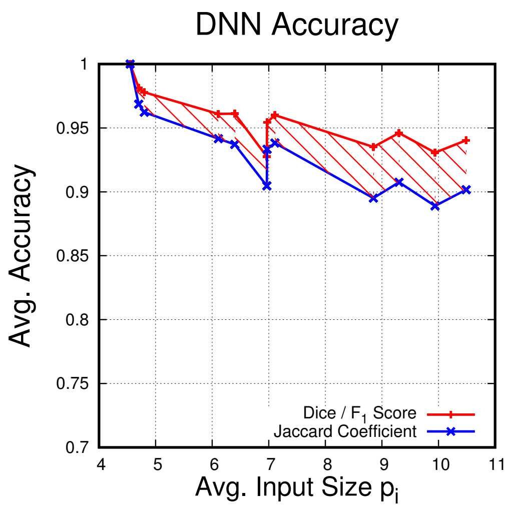
This issue was ameliorated (but not entirely removed) by not inserting duplicates in the samples; arguably a better solution would be to filter duplicates based on their *usefulness* (if that could be quantified and measured). By doing so, POS network cross-accuracy increased to 0.8995 still considerably lower than term-based accuracy.

Another reason why this is a problem is due to the contradictory nature of POS tags; whereas they form categories of words, terms and symbols, quite often those categories may generalise too much, therefore creating samples which induce under-fitting in the network. The contradictions are often enabled by ambiguity Gorrell [2006] in the underlying pattern/sentence p_i which is further generalised by using POS grouping. A solution to this problem is to prefer term-based actions since they remove part of the generalisation of POS, and offer a more specific and specialised point of view.

Sparse POS - Distributed Terms

In this scenario the POS network was a deep neural network (DNN) using sparse encoding, whilst the term processing networks used DNN processing distributed encoding as with the previous shallow ANN Section 5.4. The mean accuracy $\bar{x} = 0.9563$ and standard deviation $\sigma = 0.0213$, are higher than earlier results, whereas the full dataset accuracy at 10.5 average input is at 94%.

An interesting observation from Figure 5.9 is that accuracy does not deteriorate as much as it did earlier, in fact it appears to perform better than before with larger input. Furthermore it appears to have a "dip" at $\|p_i\| = 7$ which can be attributed to the fact that those datasets had a larger evaluation set than before, thus bigger room for error (see Figure 4.2). Even more interestingly, the DNN processing POS had a cross-evaluation (CE) accuracy of 0.8995 to which I can attribute the



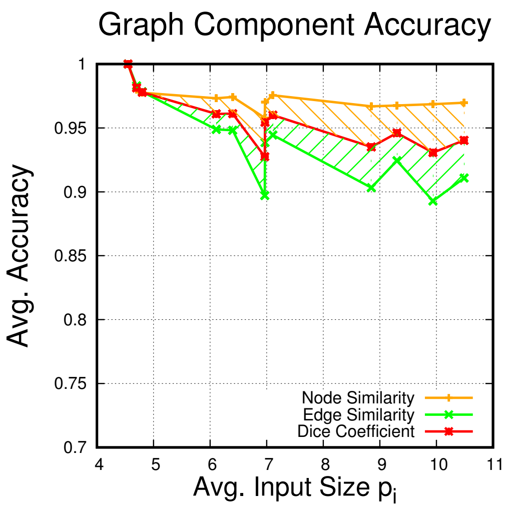


Figure 5.9: Sparse POS and Distributed Term Classifiers

decrease in edge accuracy when concerned with larger input accuracy.

Sparsely encoded DNN can become more accurate than distributed-encoded schemes, especially with large training samples. In comparison to shallow ANN, and taking into consideration that CE is at 0.8995 for POS-based edges, a 1.25% improvement was observed.

Sparse POS - Sparse Terms

The networks used were a deep sparse-encoding network for POS, and two deep sparse-encoding networks for term-based edges. The Dice/F_1 mean accuracy was $\hat{x} = 0.9384$ and standard deviation $\sigma = 0.0263$. Accuracy was therefore lower than shallow ANN, or distributed-encoded terms.

The term-based DNN had 0.9999 CE, therefore I cannot attribute the drop in accuracy to a non-optimised network. In fact, the only changed parameter is the shift from a distributed-encoding scheme to a sparsely-encoded scheme. The only other factor that may cause that decrease in accuracy is the option to use Semantics-based terms swapping.

Distributed POS - Sparse Terms

The networks used were a deep sparse-encoded network for term-based edges, and a DNN using distributed-encoded triplet values for POS.

The Dice/F_1 mean accuracy was $\hat{x} = 0.9382$ and standard deviation $\sigma = 0.0262$, shown in Figure 5.11. Accuracy below the shallow ANN benchmark, and the

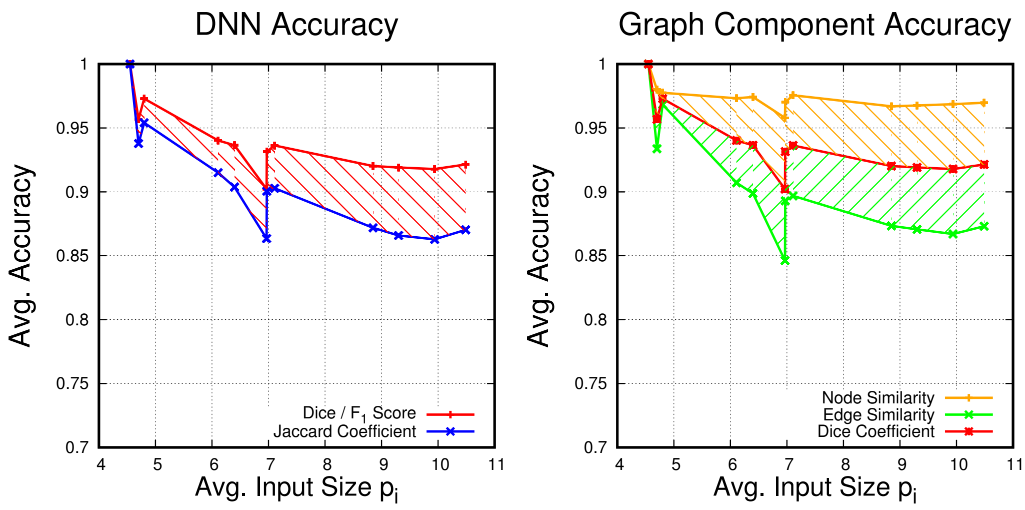


Figure 5.10: Sparse-Encoded POS and Term Edge Classifiers

previously DNN obtained results. Since the sparsely-encoded POS cross-evaluation was also lower at 0.8995 CE is ameliorated by the somewhat better term-based CE at 0.9999, yet the overall performance is below all previous schemes. This also proves that distributed encoding of POS-based edges is marginally worse than sparsely-encoded POS vectors. Similarly it further supports the notion that the distributed encoding of term-based edges is better than the sparsely-encoded term vectors.

Distributed POS - Distributed Terms

This final scenario used DNN which encoded distributed triplet values, similar to the shallow ANN earlier. Those networks aren't shallow, don't use logistic activation functions but linear (ReLU) and have 3 and 4 hidden layers, and 120 - 180 hidden nodes. From Figure 5.12 it is apparent that performance of distributed-encoded DNN is on par with shallow ANN; in fact it is marginally better by 0.51%. Whereas shallow ANN achieved an average $\hat{x} = 0.9438$ the equivalent DNN using the same data achieved $\hat{x} = 0.9489$ with a standard deviation of $\sigma = 0.0240$.

Therefore it is clear that when using the more traditional distributed-encoded scheme, actual performance doesn't really differ; the small margin could be attributed to larger training samples, L1 or L2 regularisation or random weight initialisation. The latest trend of deep learning has made considerable improvements in many fields, and whilst NLU and NLP are one of those fields, as indicated by Andor et al [Andor et al., 2016] shallow ANN can often perfrom just as good and achieve similar accuracy easier than DNN.

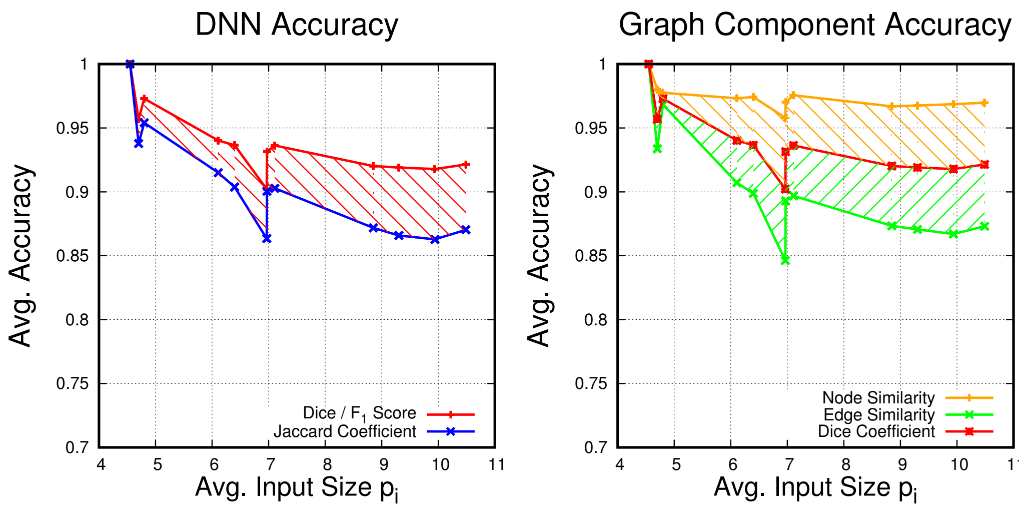


Figure 5.11: Sparse-Encoded Term Edge and Distributed-Encoded POS Classifier

Deep Learning Conclusions

Training deep networks is considerably more difficult that training shallow networks, and the hyper-parameters are a lot harder to optimise, as shown in the previous Section. It would be unsound to conclude that deep learning in NLU doesn't perform well; it appears to offer a better stability especially when using larger input size, yet fine-tuning it is a tedious task.

Increasing the training samples (which is often considered a good solution) did not often help, and when comparing shallow ANN to DNN the results were marginally better. What did make a difference, was using a sparse vector encoding for POS; it achieved the best accuracy of 94% in combination with a distributed encoding scheme of triplet values.

Considering that Google's Syntaxnet [Andor et al., 2016] achieved a 94.44% F_1 score on news data using only POS, whereas this work managed to achieve a 94% on data from a similar domain, but of smaller input size is encouraging; however there is still room for improvement when taking into account the somewhat low CE from the trained DNN.

I also tried using deep learning with POS-only edges; the agent only used POS tags and no term-based actions at all. Accuracy was considerably lower: the best attainable mean was $\hat{x} = 0.8895$ with $\sigma = 0.0374$. Those results support the notion that term-based edges are not only useful but preferable to POS tags.

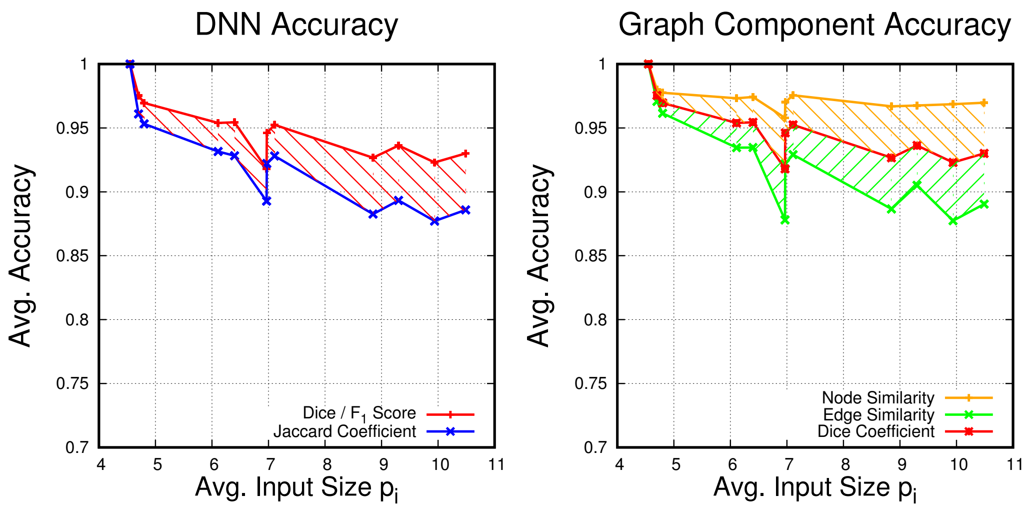


Figure 5.12: Distributed Term Edges and Distributed POS Edges

5.6 Experiment Conclusions

In this Chapter I described the methodology used for experimentation, how accuracy was quantified and how it relates to similar approaches from other research, the experiments done throughout my doctoral research, the Algorithm implementations in the highest level, the actual results and their analyses, as well as how the Icarus engine compares to state-of-the-art research.

The first and obvious conclusion to which all parsers are fallible is the input size and how that relates to complexity. It was disheartening to observe high accuracy and performance deteriorate with increased input size, regardless of the Algorithm used. I investigated as many attributes of complexity as possible and within my research scope (because complexity is a field on its own accord) and discovered that (a) all parsers suffer from this phenomenom as Choi et al [Choi et al., 2015] clearly indicate, and (b) that there appears to be a connection with the meaning and logic flow of a sentence and not simply the input size, although that complexity often increases in accordance with the input size.

This is best demonstrated in Figure 5.13; I used Principal Component Analysis (PCA) on the input pattern p_i features (word length and edge search space) and projected their most significant Eigenvectors on a single dimension on axis X, then used axis Y to represent the node to edge ratio which appears to be an indicator of graph complexity, and used axis Z to plot the graph average path length (see Section 4.2). The 4th dimension is the accuracy: a PCA on Boolean (Exact Match), Dice/F_1 Score and Jaccard, represented by the heatmap, e.g., the colour change of

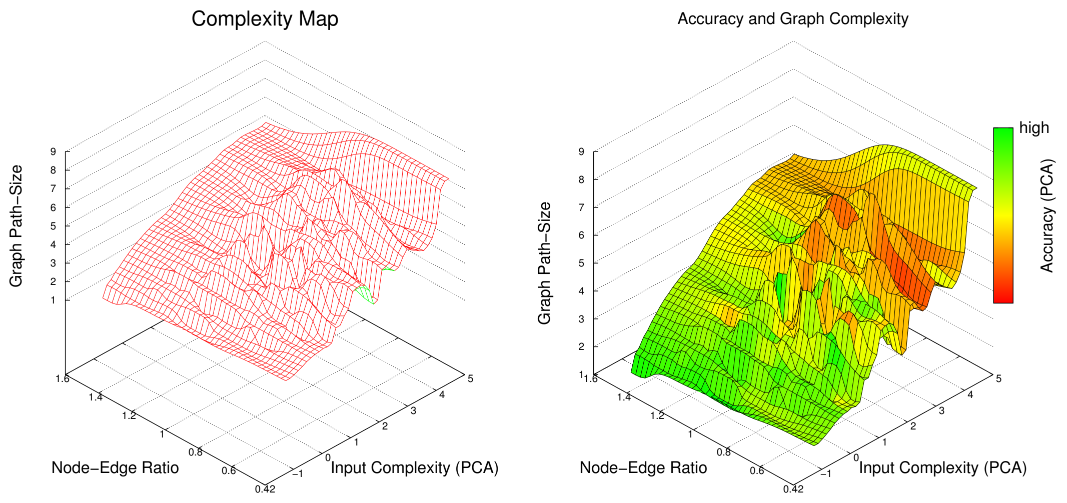


Figure 5.13: Complexity Mapping to Accuracy PCA

the right-side plot. From a visual analysis of the Figure it is easy to spot the most crucial factor: node to edge ratio, e.g, graph sparseness, which has been correctly identified as a complexity factor. Whilst graph path-size does increase (almost linearly) the complexity, it is in combination with sparseness that it creates the "ripple effect" seen in the Figure 5.13 which coincidentally has lower than average accuracy. This ripple effect is observable in all previous Figures in DNN, and can be seen as the "dip" at various stages, regardless of input size.

Overall the agent accuracy ranged depending on the Algorithm implementation, however the actual best performing Algorithm (DNN Section 5.5.2) performed somewhat similar to state of the art platforms taking into consideration the fact that larger input was not used, and that CG have an added level of complexity due to their bipartite nature. Coincidentally the actual implementations are also similar; albeit I've published theoretical approaches earlier [Gkiokas and Cristea, 2014b], it wasn't until later that results were submitted [Gkiokas and Cristea, 2016a].

The agent approach taken is vastly different from NLU, dependency parsing and the general field of NLP, it has been designed as a cognitive agent scheme taking into consideration *experience*, *paradigm decoding*, *associative learning* and *imitation*, rather than the idiomatic attributes of news data, the characteristics of the NLP domain, and the background work of NLU and NLP. This is evident by the different approaches taken in my doctoral research in comparison to the standards in NLU and NLP. However, as seen by the results, albeit Icarus engine is highly unorthodox it provides similar performance and accuracy. The evaluation is, due to the dataset and its characteristics, inconclusive on larger input. Whereas there are large sentences in the dataset, those are not many, in fact the dataset used favours smaller input (see Chapter 4, and in specific Figure 4.2) therefore a direct comparison to other research which uses larger input sizes is not possible.

Chapter 6

Conclusions and Future Work

Closing the thesis in this Chapter, I provide a discussion on what has been done within the four years of my Doctoral research, the design, implementation and choices made, and the contributions to the field of AI.

6.1 Conclusions

First and foremost, the approach taken has been unconventional for which there is a very good justification; during my first two years I tried to mimic and implement a cognitive agent which would be as similar to the imitating mechanisms found in humans as possible. Whereas I cannot claim it is biologically plausible or an emulation, it certainly does meet all criteria as set and described in Sections 2.5 and 3.13 by the theoreticians of Synthetic and Cognitive AI. What has primarily been achieved is the demonstration that a cognitive agent as an implementation in C++ does actually learn by example, and it does so by using associative memory models, reinforcement learning and statistical probabilities. Furthermore, not only does it work as intended, but in most cases it provided results which are in the same category as state-of-the-art results, with the potential to achieve even better results if the DNN used are further optimised or if big data is used to train the agent.

Therefore the importance of the main contribution is that this thesis describes one of the cases where theoretical work laid beforehand has been empirically and methodically proven to function, and more so by achieving a significant accuracy. The agent relies on a form of Deep Artificial Reinforcement Learning, similar to how DQN used deep learning with reinforcement learning. That premise is not unknown and is gaining momentum: standalone models can be enhanced when combined correctly with other Algorithms, such as in the case of DQN as demonstrated by DeepMind and Google.

The agent manages to efficiently decompose paradigms given as examples by the user; admittedly the field of example decomposition in learning by imitation requires further research as it could entail a variety of benefits to both robotics and software agents. Whereas in software agents decomposition in effect becomes parsing, in Robotics it is more closely related to computer vision, yet both decomposition sub-domains require a common framework with its foundations in AI and not NLP or Robotics. This in effect translates into the need of decomposing domain-specific material (e.g., images, video, text, audio) into high level KR (e.g., the CG used here or the MRL used in NLU) so that a unitary imitation mechanism can be developed which would encompass all information available to AI agents.

All of the original research questions have been answered. Furthermore, theoretical advice from Haikonen (Section 2.5.2) and Bach (Section 2.5.1) have been used as the template upon which Icarus was built (analysed in Section 3.13). Icarus was designed, implemented and tested as an interacting software agent, with the end-goal of being a node in an intelligent system which can provide KR from text, and then use it to reason. Whereas it was impossible to develop a complex cognitive reasoning system within the time-frame of this PhD, it is doable if using as the basis the Icarus engine, which could potentially also perform other high level cognitive functions, such as learning to reason by imitation, etc. Following is an analysis of those research questions, how they have been addressed and the contribution related to the field.

How can cognitive agents learn by imitation?

Agents, cognitive, robotic and artificial systems can learn as shown in this thesis, by using reinforcement learning as the MDP template serves the temporal-sequential role of both the learner and the dynamic program. By doing so, the agent is extracting *qualia* from the learning material, and internally re-creating it and learning it. This approach has two major advantages: (a) it is based upon reinforcement learning, which is itself based upon behaviouristic Psychology and is therefore an Algorithm inspired by the biological counterpart, and (b) it does so in a sequential-temporal manner instead of a one-off process. Doing so enables cascading of different families of Algorithms as seen in previous Chapter 5 and also enables beam-searching using the MDP as the indexing mechanism.

How can an AI agent acquire knowledge from the Internet via imitation?

The Icarus agent has been successful at acquiring knowledge (stored as CG) from news sources on the Internet (RSS feeds). Whilst not all Internet knowledge is stored in news articles, and a considerable amount of user content is very different from news articles, Icarus is able to answer that question. Using a combination of RL, ANN and Statistics Icarus achieved a satisfactory accuracy in projecting raw text onto KR, and it did that by being shown and then learning by imitation.

What are the differences between Learning by Imitation and Programming by Example?

Whereas PBE [Lieberman, 2000] is concerned about programming, AI imitation learning is mostly focusing on learning. Traditionally the field of imitation learning has seen research in Robotics and not in AI. On the contrary PBE was a field now eclipsed, which had most work (if not the entirety) done on graphical user interfaces. Most of the criticism on PBE (in Section 2.6.2) is what this thesis addressed:

- *Hidden States*: decoding examples into qualia is done via inference (heuristic or approximating Algorithms).
- *Machine Learning*: using ANN, Deep Learning and other ML models has clearly demonstrated big advantages, especially when compared to more traditional Heuristic, Statistic or Probabilistic approaches.
- *Sophistication and Complex Systems*: PBE systems used a naive and somewhat overly simplistic approach; Icarus on the other hand is a full-fledged cognitive AI agent, which comes with certain complexity but a much higher degree of flexibility and accuracy.

Whereas PBE made too early of an appearance in Computer Science, modern AI could potentially advance both user interfaces as well as Cognitive AI systems by PBE or learning by imitation. The results of this thesis have demonstrated that the lines between programming and learning can blur when the target is an agent and not an interface.

What are the advantages of agents which learn by imitation?

This question was asked by Dautenhanh [Dautenhahn and Nehaniv, 2002a] and remains valid today. Why should researchers or developers put additional effort into creating or enabling imitation in agents and systems alike? The answer to such a question cannot be easily answered and depends on the scenario or the domain. In general however, the advantages are:

- Agents can learn directly from users, user generated data, raw information (text or otherwise), thereby reducing the need for processing data, information, creating training samples or sets, etc.
- A learning agent can be autonomous, self-updated, and require little to no maintenance when compared to other systems.
- There is no need for searching, trial and error, state mapping, etc. This is perhaps the biggest advantage and often the one most undermined. With imitation there exists *knowledge transference* and thus agents don't need to discover; they learn by observation or interaction.

Researchers previously considered such agents as *researchware*, software with the sole purpose of being used in research. However that idea is beginning to change; modern IT giants such as IBM, Google, Microsoft, Yahoo, Apple and Amazon are investing in personal AI assistants (Siri, Alexa, Watson, Cortana) as well as cloud-based AI agents and systems. Therefore, a crucial part of those agents already is, or will soon be, learning by imitation.

How do artificially imitating agents compare to traditional software systems?

Chapter 3 set the theoretical foundations, based upon previous research but creating an imitating agent. Chapter 5 reported on the results and findings, evaluating and demonstrating that imitating agents can outperform traditional approaches, and are similar to state-of-the art researchware which rely on very similar technologies (e.g. SyntaxNet). Although Heuristic systems based on simple but robust mechanisms are still useful and have domain-specific applications, artificial agents are more adaptive and flexible and can constantly evolve or become better over time.

6.2 Criticism and Limitations

One of the cornerstone issues of parsers is accuracy versus speed. Researchware often focuses on how accurately it can parse, without taking into account the fact that applications must also be fast and optimised. Icarus is indeed researchware, optimisations have been made to increase accuracy and not speed.

Other limitations arise from the need to re-train when dealing with new domains, sub-domains or unknown symbols. There doesn't seem to exist a clear solution to this: big-data and extremely large training sets appear to be one way of handling such issues, another solution is the ability to continuously update. Icarus tackles this issue by being updatable and re-trainanable on-the-fly, as well as using semantic approximation (Sections 2.7.3, 3.10 and 5.2) in order to alleviate the need for retraining or domain-dependant training.

Arguable one of the biggest criticisms is the decision to use conceptual graphs instead of MRL which is the norm. The use of CGs is justified; this is not an NLP/NLU focused research, but instead relies on users to create training samples from which the agent learns. As such it was crucial that the KR scheme used would be visually simple and easy to use, and CG fulfil that criteria. However, they also induce additional errors: the node accuracy (which in essence is the amount of correctly selected components of the bipartite CG) propagates as error into the edge accuracy. Whereas the exact correlation of node to edge error propagation is unknown, it is certain that it exists (albeit its not the only reason why edge similarity declines). Using MRL instead of CG would have removed that error, and could therefore increase Icarus accuracy; furthermore it would allow Icarus to be trained with very large datasets which have been used in the past.

One of the limitations which I haven't been able to surpass but do have plans for the near future, is the ability to take into account context, e.g., previous term t_j and next term t_k when examining actions for t_i within a pattern p_i. Whereas the temporal nature of the MDP takes into consideration the creation of the KR, it does not take into consideration the semiotics of the previous and next entities. This could possibly be achieved with Bayesian inference, or recurrent neural networks (RNN). Whilst Andor et al [Andor et al., 2016] argue that ANN can perform better than RNN, it is important to note that processing time series is most always better done using RNN, echo state network (ESN) or similar models.

6.3 Future Work

A task for the very near future is to remove CG and implement Icarus using MRL and train it with existing datasets, and then examine how it performs. Furthermore, training it with multiple datasets (big data) could provide valuable insight into actual applications in the field of NLU and NLP.

Another task for future work is the development of a contextualising Algorithm when deciding on edge actions, preferably using Bayesian inference on top of the Statistical probabilities, and perhaps with the use of a ESN for processing time-series predictions and classifications.

The last task for the near future is to further develop a reasoning system based on Icarus, which would attempt to arrive to conclusions based on KR input given by the user, or based on text input.

Appendix A

Penn Treebank POS tags

Tag	Description
CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential *there*
FW	Foreign Word
IN	Proposition/Subordinating conjunction
JJ	Adjective
JJR	Adjective comparative
JJS	Adjective superlative
LS	List item marker
MD	Modal
NN	Noun, singular or mass
NNS	Noun, plural
NNP	Proper noun, singular
NNPS	Proper noun, plural
PDT	Predeterminer
POS	Possessive ending
PRP	Personal pronoun
PRP$	Possessive pronoun
RB	Adverb
RBR	Adverb, comparative
RBS	Adverb, superlative
RP	Particle
SYM	Symbol

TO	to
UH	Interjection
VB	Verb, base form
VBD	Verb, past tense
VBG	Verb, gerund or present participle
VBN	Verb, past participle
VBP	Verb, non-3rd person singular present
VBZ	Verb, 3rd person singular present
WDT	Wh-determiner
WP	Wh-pronoun
WP$	Possessive wh-pronoun
WRB	Wh-adverb

Appendix B

Conceptual Graph examples

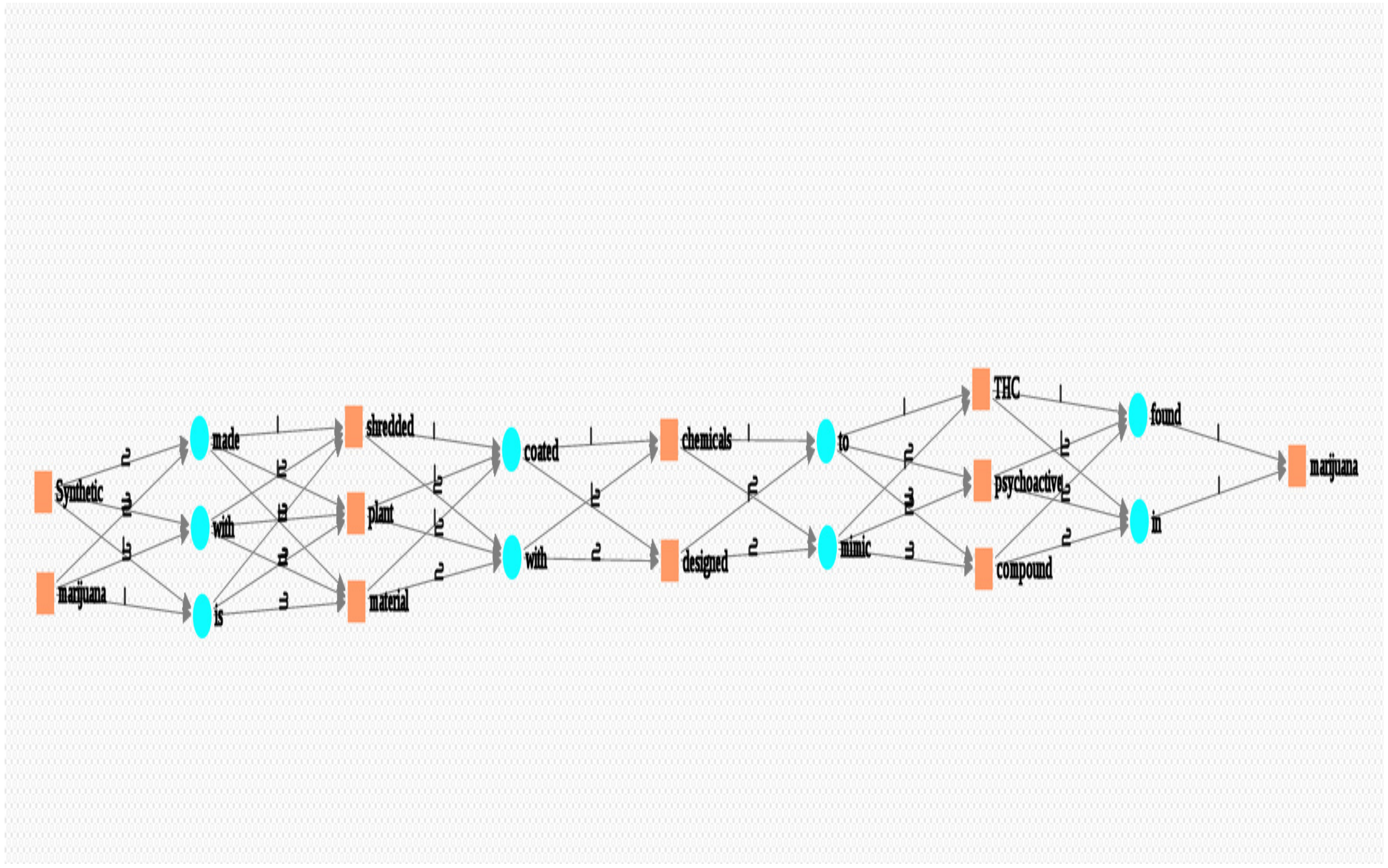


Figure B.1: Synthetic marijuana is made with shredded plant material coated with chemicals designed to mimic THC psychoactive compound found in marijuana.

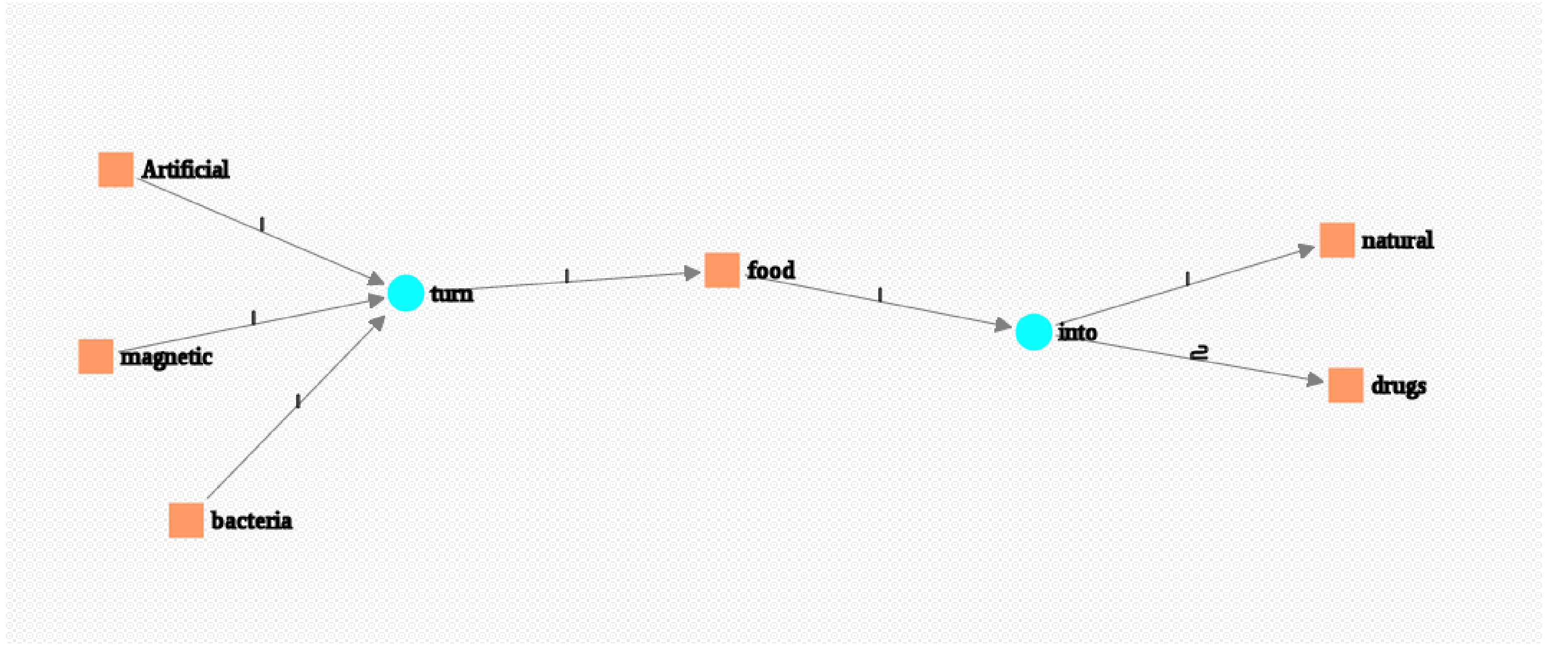


Figure B.2: Artificial magnetic bacteria turn food into natural drugs.

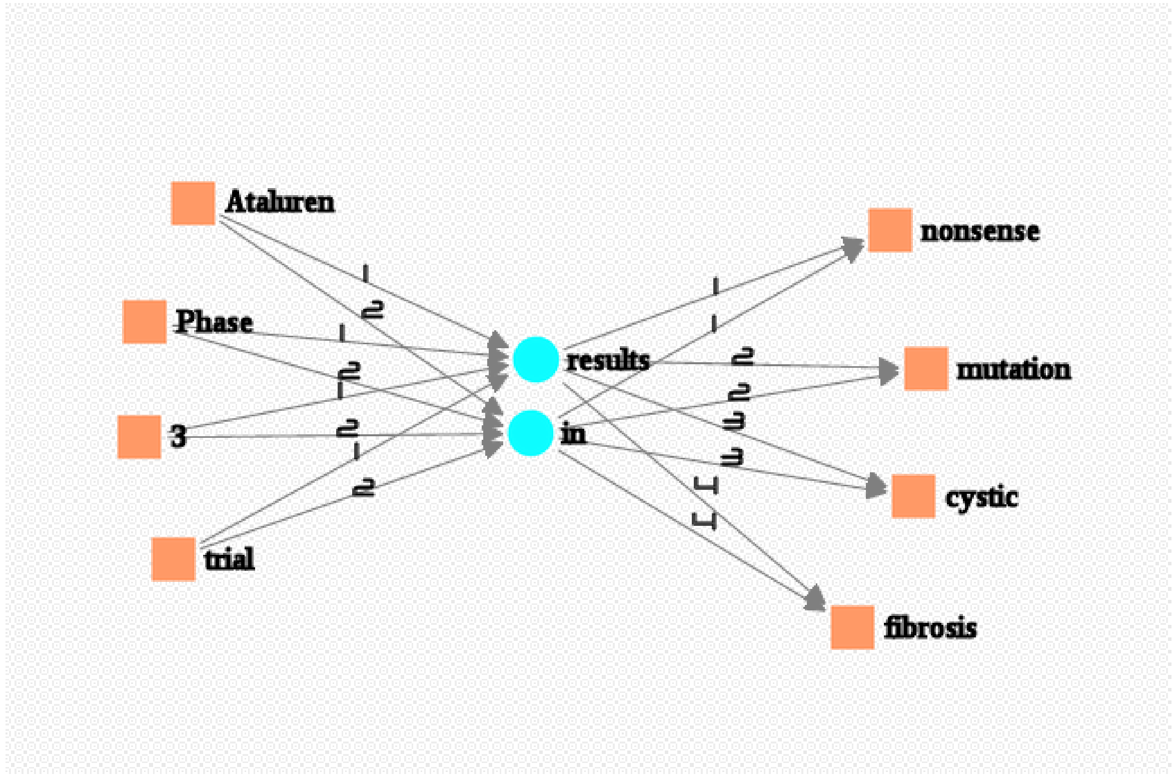


Figure B.3: Ataluren Phase 3 trial results in nonsense mutation cystic fibrosis.

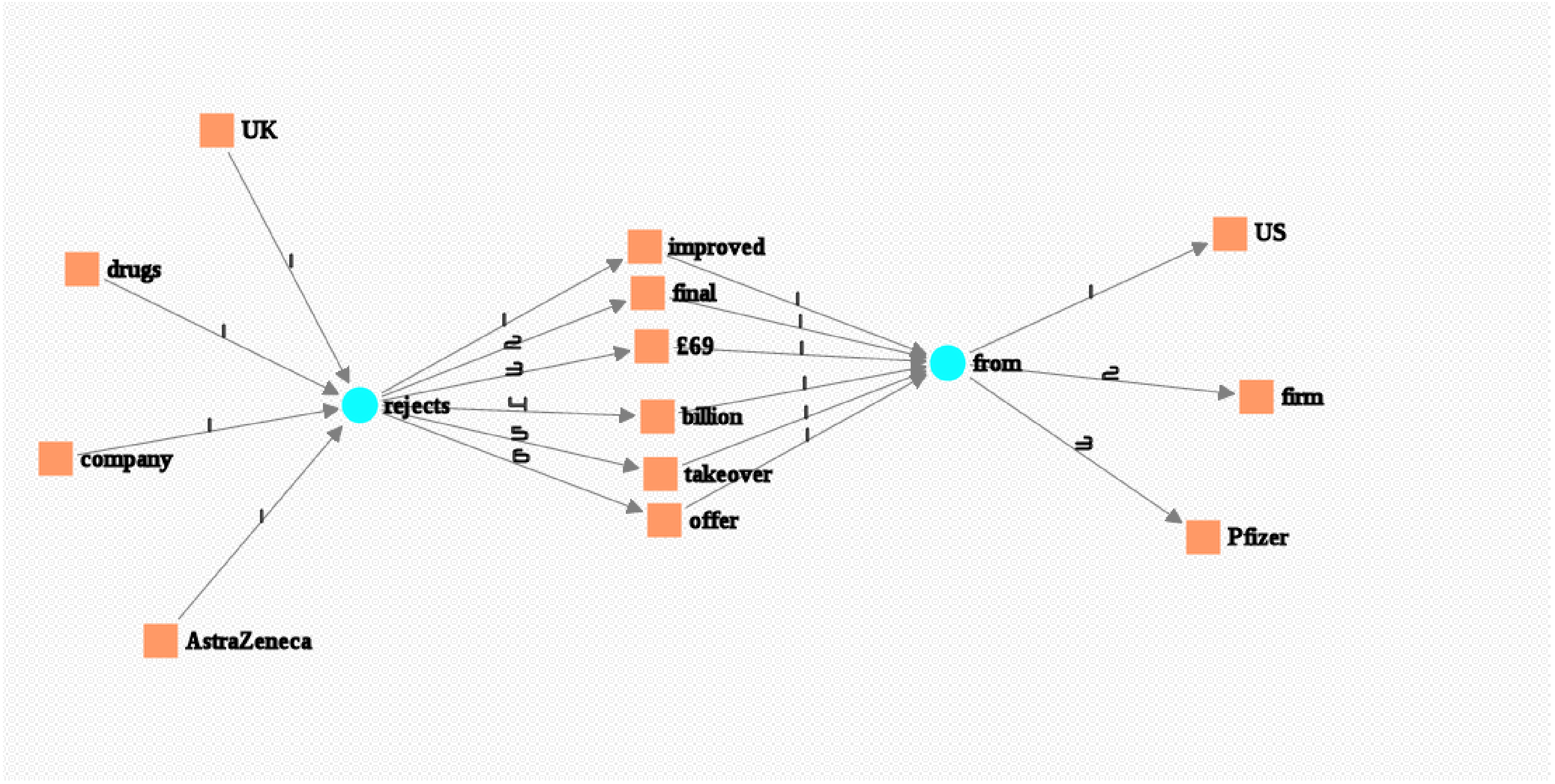


Figure B.4: UK drug company AstraZeneca rejects improved final £69 billion takeover offer from US firm Pfizer.

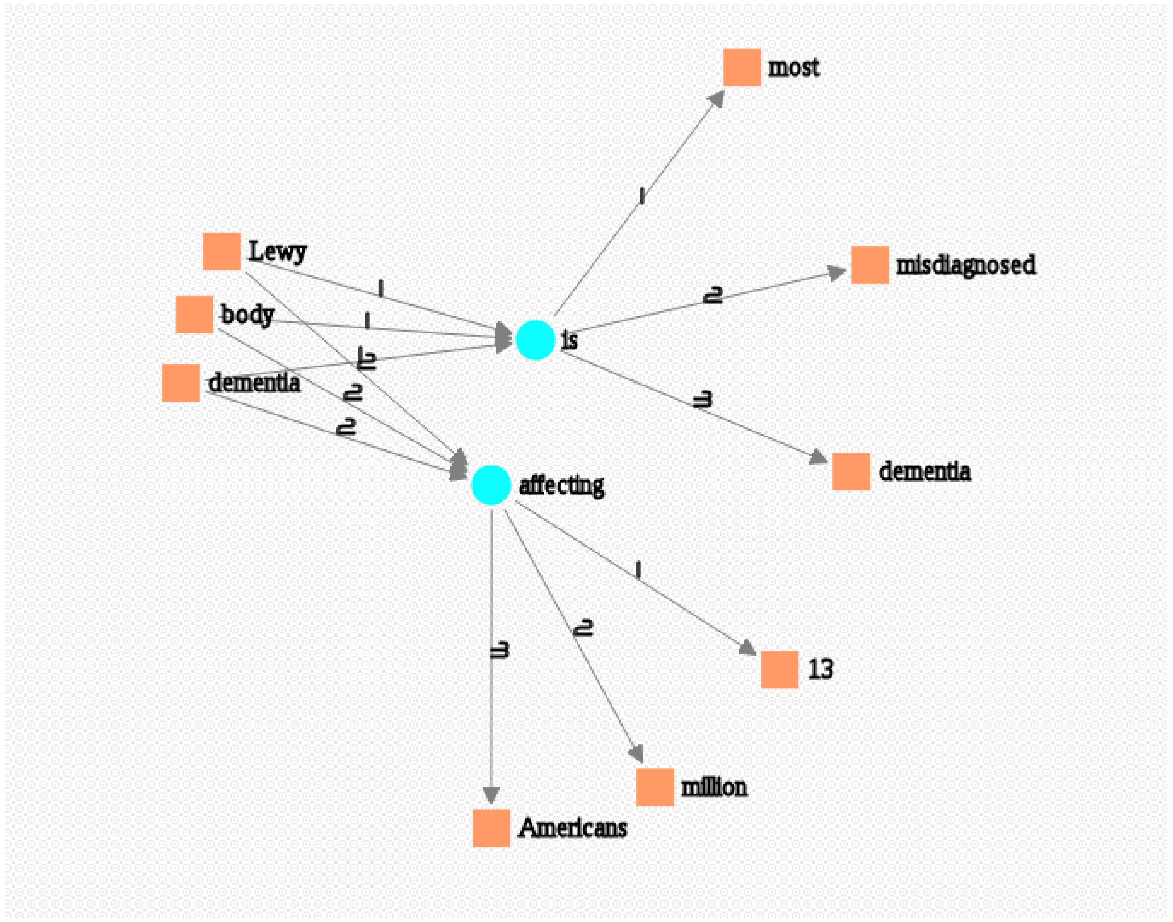


Figure B.5: Lewy body dementia is most misdiagnosed dementia affecting 13 million Americans.

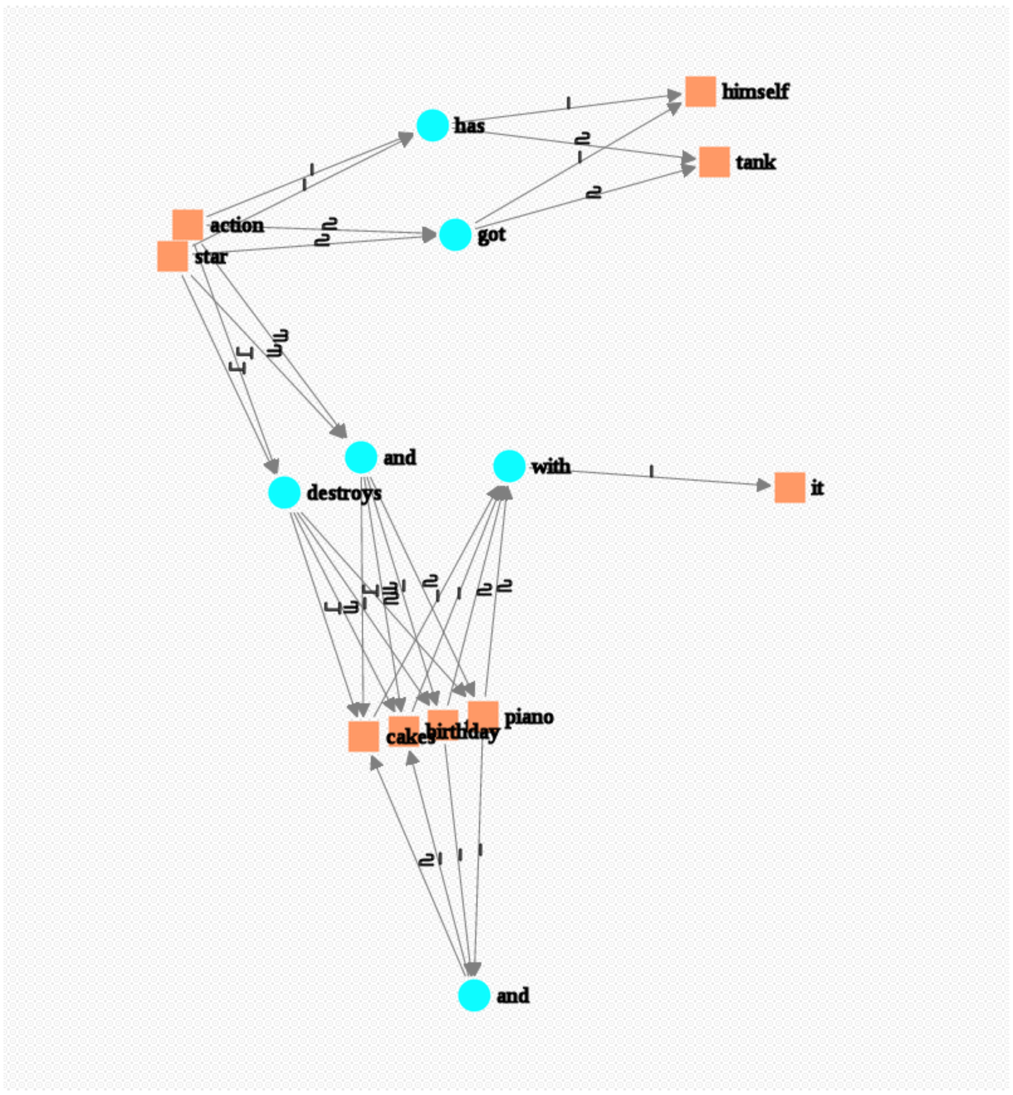


Figure B.6: Action star has got himself tank and destroys piano and birthday cake with it.

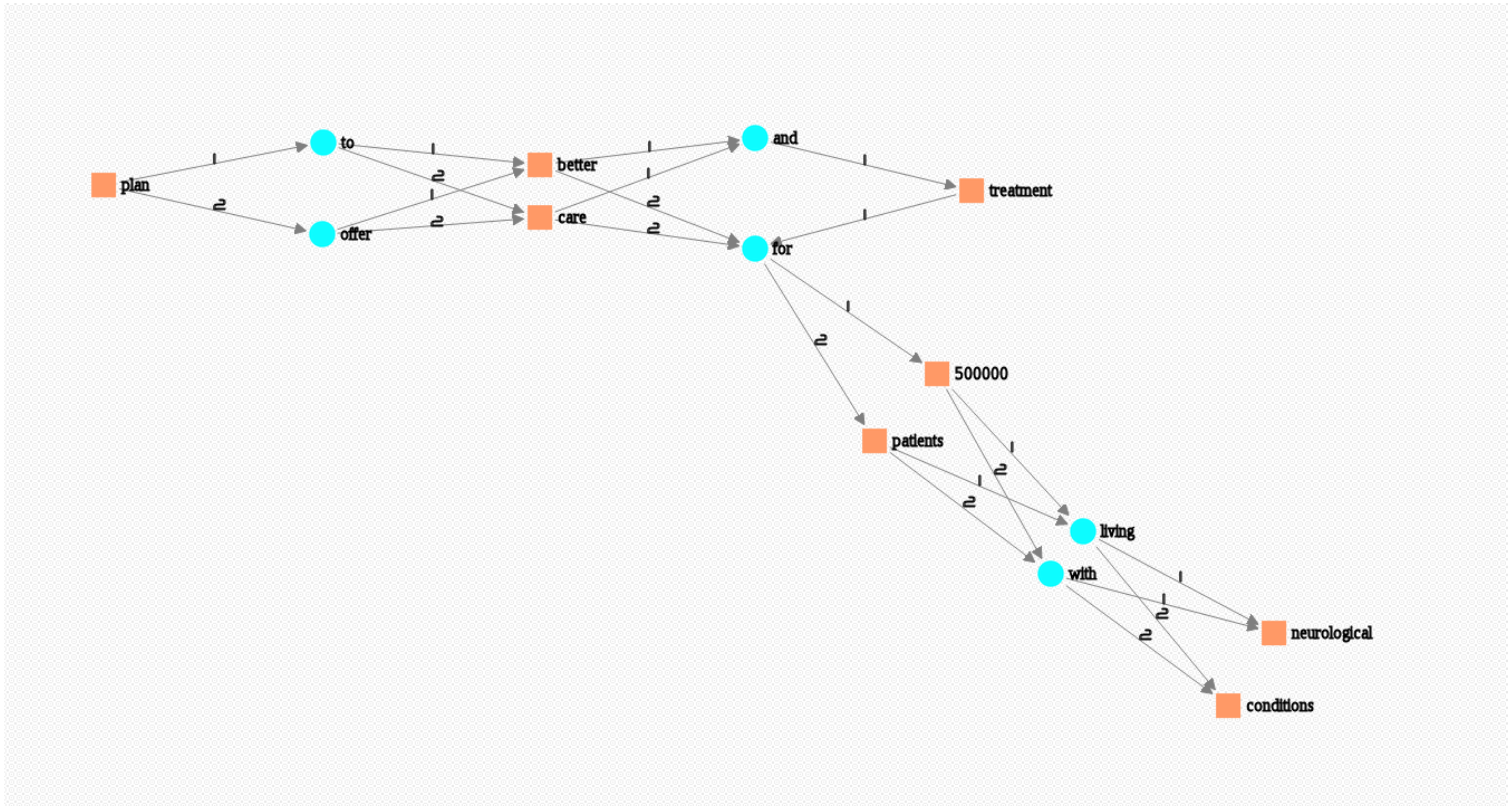


Figure B.7: Plan to offer better care and treatment for 500000 patients living with neurological conditions.

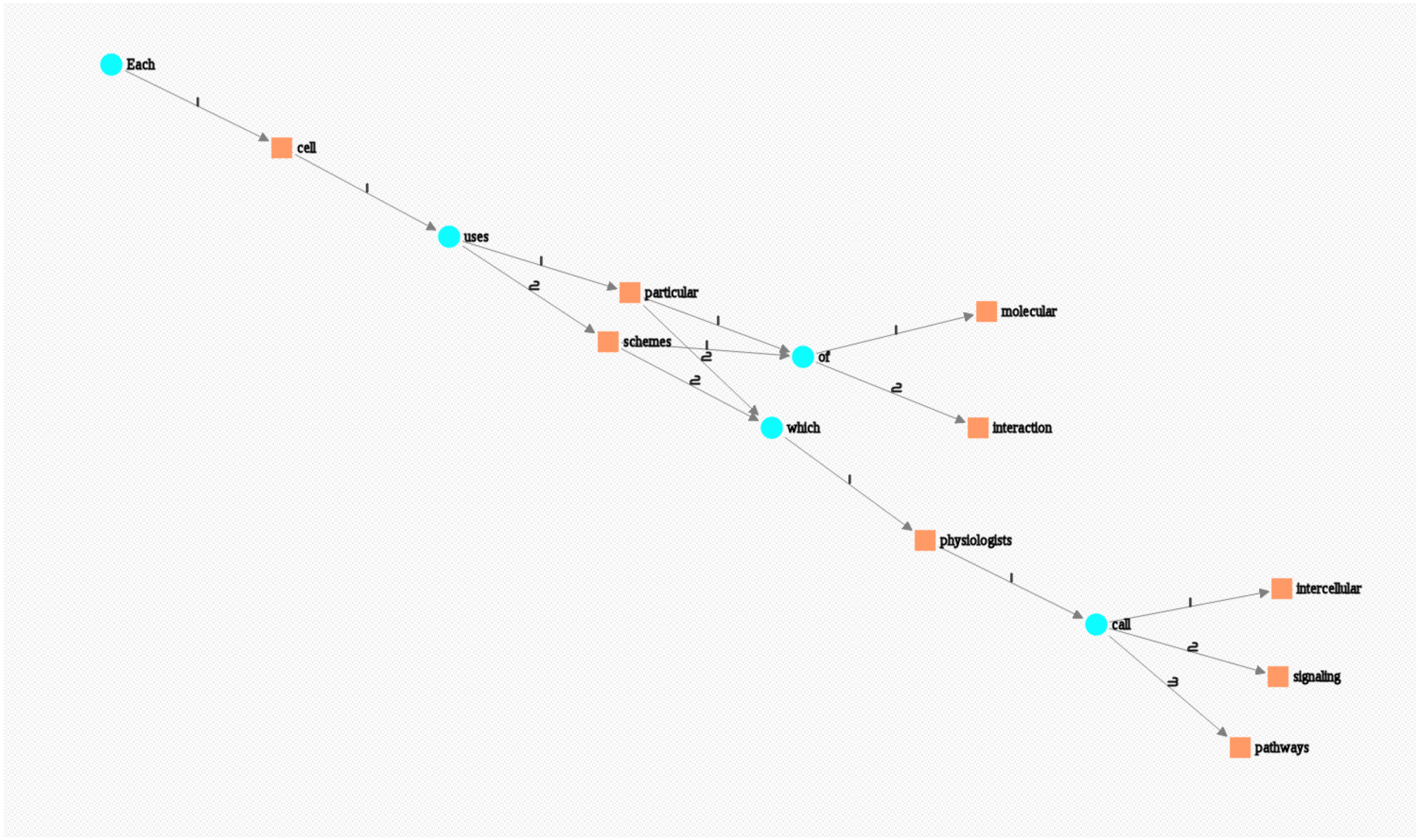


Figure B.8: Each cell uses particular schemes of molecular interaction which psychologists call intercellular signaling pathways.

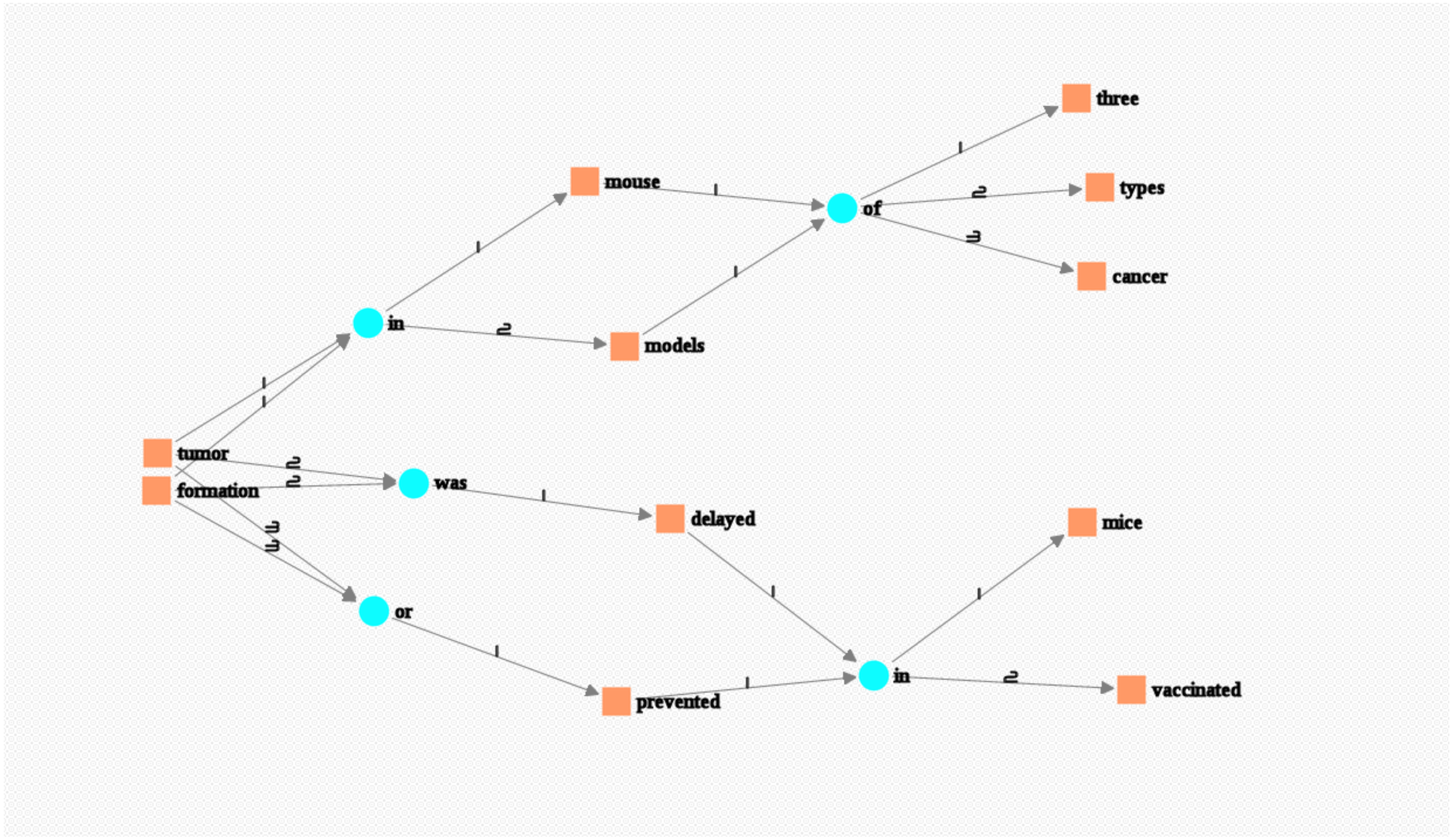


Figure B.9: Branching with Column Graphs

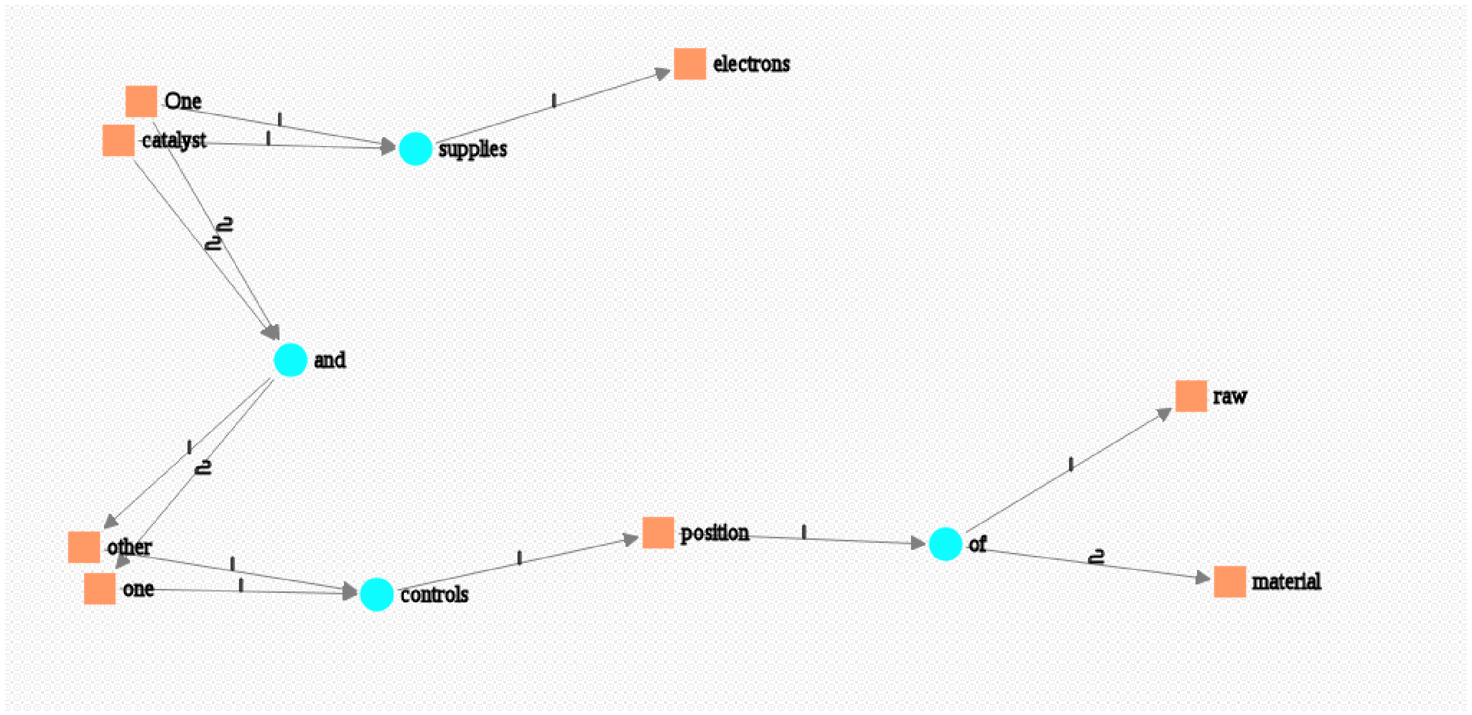


Figure B.10: Branching with Column Graphs

Bibliography

Surafel Lemma Abebe and Paolo Tonella. Natural language parsing of program element names for concept extraction. In *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, pages 156–159. IEEE, 2010.

Eneko Agirre, Enrique Alfonseca, Keith Hall, Jana Kravalova, Marius Paşca, and Aitor Soroa. A study on similarity and relatedness using distributional and wordnet-based approaches. In *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 19–27. Association for Computational Linguistics, 2009.

Robert Amant, Luke Zettlemoyer, Henry Lieberman, and Richard Potter. Visual generalization in programming by example. In Henry Liebermann, editor, *Your wish is my command: Programming by example*, chapter 19, pages 371–385. Morgan Kaufmann, 2001.

Gianni Amati and Iadh Ounis. Conceptual graphs and first order logic. *The Computer Journal*, 43(1):1–12, 2000.

John R Anderson. Act: A simple theory of complex cognition. *American Psychologist*, 51(4):355, 1996.

Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*, 2016.

Jacob Andreas, Andreas Vlachos, and Stephen Clark. Semantic parsing as machine translation. In *ACL (2)*, pages 47–52, 2013.

Joscha Bach. Seven principles of synthetic intelligence. *Frontiers In Artificial Intelligence And Applications*, 171:63, 2008.

Joscha Bach. *Principles of synthetic intelligence PSI: an architecture of motivated cognition*, volume 4. Oxford University Press, 2009.

Joscha Bach. Micropsi 2: the next generation of the micropsi framework. In *International Conference on Artificial General Intelligence*, pages 11–20. Springer, 2012.

Miguel Ballesteros, Bernd Bohnet, Simon Mille, and Leo Wanner. Deep-syntactic parsing. In *COLING*, pages 1402–1413, 2014.

Albert Bandura. *Social foundations of thought and action: A social cognitive theory.* Prentice-Hall, Inc, 1986.

Prabir Barooah and Joäo P Hespanha. Estimation on graphs from relative measurements. *Control Systems, IEEE*, 27(4):57–74, 2007.

Mathias Bauer, Dietmar Dengler, and Gabriele Paul. Trainable information agents for the web. In Henry Liebermann, editor, *Your wish is my command: Programming by example*, chapter 5, pages 87–114. Morgan Kaufmann, 2001.

J Neil Bearden and Terry Connolly. On optimal satisficing: how simple policies can achieve excellent results. In *Decision modeling and behavior in complex and uncertain environments*, pages 79–97. Springer, 2008.

Sean Bechhofer. Owl: Web ontology language. In *Encyclopedia of Database Systems*, pages 2008–2009. Springer, 2009.

Richard Bellman. A markovian decision process. Technical report, DTIC Document, 1957.

Yoshua Bengio. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

Jonathan Berant and Percy Liang. Semantic parsing via paraphrasing. In *ACL (1)*, pages 1415–1425, 2014.

Dale E Berger, Kathy Pezdek, and William P Banks. *Applications of cognitive psychology: Problem solving, education, and computing.* Routledge, 2013.

Dimitri P Bertsekas and John N Tsitsiklis. Neuro-dynamic programming: an overview. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 1, pages 560–564. IEEE, 1995.

Isabelle Bichindaritz and Sarada Akkineni. Concept mining for indexing medical literature. *Engineering Applications of Artificial Intelligence*, 19(4):411–417, 2006.

Irving Biederman and Peter C Gerhardstein. Recognizing depth-rotated objects: evidence and conditions for three-dimensional viewpoint invariance. *Journal of Experimental Psychology: Human perception and performance*, 19(6):1162, 1993.

Ann Bies, Justin Mott, Colin Warner, and Seth Kulick. English web treebank. *Linguistic Data Consortium, Philadelphia, PA*, 2012.

David F Bjorklund. Mother knows best: Epigenetic inheritance, maternal effects, and the evolution of human intelligence. *Developmental Review*, 26(2):213–242, 2006.

Stephan Bloehdorn and Alessandro Moschitti. Combined syntactic and semantic kernels for text classification. In *European Conference on Information Retrieval*, pages 307–318. Springer, 2007.

Bernd Bohnet. Very high accuracy and fast dependency parsing is not a contradiction. In *Proceedings of the 23rd international conference on computational linguistics*, pages 89–97. Association for Computational Linguistics, 2010.

Luca Bonini and Pier Francesco Ferrari. Evolution of mirror systems: a simple mechanism for complex cognitive functions. *Annals of the New York Academy of Sciences*, 1225(1):166–175, 2011.

Cynthia Breazeal and Brian Scassellati. Robots that imitate humans. *Trends in cognitive sciences*, 6(11):481–487, 2002.

Cynthia Breazeal, Daphna Buchsbaum, Jesse Gray, David Gatenby, and Bruce Blumberg. Learning from and about others: Towards using imitation to bootstrap the social understanding of others by robots. *Artificial life*, 11(1-2):31–62, 2005.

Kay H Brodersen, Cheng Soon Ong, Klaas E Stephan, and Joachim M Buhmann. The balanced accuracy and its posterior distribution. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 3121–3124. IEEE, 2010.

Jerome S Bruner. *The process of education.* Harvard University Press, 2009.

Peter Cabena, Pablo Hadjinian, Rolf Stadler, Jaap Verhees, and Alessandro Zanasi. *Discovering data mining: from concept to implementation.* Prentice-Hall, Inc., 1998.

Charles F Cadieu, Ha Hong, Daniel LK Yamins, Nicolas Pinto, Diego Ardila, Ethan A Solomon, Najib J Majaj, and James J DiCarlo. Deep neural networks rival the representation of primate it cortex for core visual object recognition. *PLoS Comput Biol*, 10(12):e1003963, 2014.

Keith E Campbell and Mark A Musen. Representation of clinical data using snomed iii and conceptual graphs. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, page 354. American Medical Informatics Association, 1992.

Rudolf Carnap. *Introduction to semantics*, volume 1042. Harvard University Press Cambridge, Massachusetts, 1948.

Raymond B Cattell. Theory of fluid and crystallized intelligence: A critical experiment. *Journal of educational psychology*, 54(1):1, 1963.

Marco Cavazzuti. *Optimization Methods: From Theory to Design Scientific and Technological Aspects in Mechanics*. Springer Science & Business Media, 2012.

William B Cavnar, John M Trenkle, et al. N-gram-based text categorization. *Ann Arbor MI*, 48113(2):161–175, 1994.

Gavin C Cawley and Nicola LC Talbot. Preventing over-fitting during model selection via bayesian regularisation of the hyper-parameters. *Journal of Machine Learning Research*, 8(Apr):841–861, 2007.

Eugene Charniak. Tree-bank grammars. In *Proceedings of the National Conference on Artificial Intelligence*, pages 1031–1036, 1996.

Bernard Chazelle. An optimal convex hull algorithm in any fixed dimension. *Discrete & Computational Geometry*, 10(1):377–409, 1993.

Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, and Kevin Skadron. A performance study of general-purpose applications on graphics processors using cuda. *Journal of parallel and distributed computing*, 68(10):1370–1380, 2008.

Michel Chein and Marie-Laure Mugnier. *Graph-based knowledge representation: computational foundations of conceptual graphs*. Springer Science & Business Media, 2008.

Danqi Chen and Christopher D Manning. A fast and accurate dependency parser using neural networks. In *EMNLP*, pages 740–750, 2014.

David L. Chen and Raymond J. Mooney. Learning to sportscast: A test of grounded language acquisition. In *Proceedings of 25th International Conference on Machine Learning (ICML-2008)*, Helsinki, Finland, July 2008.

Jinho D Choi, Joel Tetreault, and Amanda Stent. It depends: Dependency parser comparison using a web-based evaluation tool. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL*, pages 26–31, 2015.

Dan C Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *IJCAI Proceedings-International Joint Conference on Artificial Intelligence*, volume 22, page 1237, 2011.

Daoud Clarke. Simple, fast semantic parsing with a tensor kernel. *arXiv preprint arXiv:1507.00639*, 2015.

Carol E Cleland. Is the church-turing thesis true? *Minds and Machines*, 3(3): 283–312, 1993.

Ronan Collobert. Deep learning for efficient discriminative parsing. In *AISTATS*, volume 15, pages 224–232, 2011.

Madalina Croitoru, Bo Hu, Srinandan Dasmahapatra, Paul Lewis, David Dupplaw, Alex Gibb, Margarida Julia-Sape, Javier Vicente, Carlos Saez, Juan Miguel Garcia-Gomez, et al. Conceptual graphs based information retrieval in health agents. In *Computer-Based Medical Systems, 2007. CBMS'07. Twentieth IEEE International Symposium on*, pages 618–623. IEEE, 2007.

Aron Culotta and Jeffrey Sorensen. Dependency tree kernels for relation extraction. In *Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, page 423. Association for Computational Linguistics, 2004.

George E Dahl, Ryan P Adams, and Hugo Larochelle. Training restricted boltzmann machines on word observations. *arXiv preprint arXiv:1202.5695*, 2012.

George E Dahl, Tara N Sainath, and Geoffrey E Hinton. Improving deep neural networks for lvcsr using rectified linear units and dropout. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8609–8613. IEEE, 2013.

Kerstin Dautenhahn and Chrystopher L. Nehaniv. The agent-based perspective on imitation. In Kerstin Dautenhahn and Chrystopher L. Nehaniv, editors, *Imitation in Animals and Artifacts*, pages 1–40. MIT Press, Cambridge, MA, USA, 2002a. ISBN 0-262-04203-7. URL `http://dl.acm.org/citation.cfm?id=762896.762898`.

Kerstin Dautenhahn and Chrystopher L. Nehaniv, editors. *Imitation in Animals and Artifacts*. MIT Press, Cambridge, MA, USA, 2002b. ISBN 0-262-04203-7.

Raymond J. Mooney David L. Chen, Joohyun Kim. Training a multilingual sportscaster: Using perceptual context to learn language. *Journal of Artificial Intelligence Research*, 37:397–435, 2010.

Winston Davies and Pete Edwards. Dagger: A new approach to combining multiple models learned from disjoint subsets. *machine Learning*, 2000:1–16, 2000.

Julien Offray de La Mettrie. *Man a machine*. Open court publishing Company, 1912.

J Decety, T Chaminade, J Grezes, and AN Meltzoff. A pet exploration of the neural mechanisms involved in reciprocal imitation. *Neuroimage*, 15(1):265–272, 2002.

Georgiana Dinu and Rui Wang. Inference rules and their application to recognizing textual entailment. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 211–219. Association for Computational Linguistics, 2009.

David A Duffy. *Principles of automated theorem proving*. John Wiley & Sons, Inc., 1991.

Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.

Greg Durrett and Dan Klein. Neural crf parsing. *arXiv preprint arXiv:1507.03641*, 2015.

David Elizondo. The linear separability problem: Some testing methods. *Neural Networks, IEEE Transactions on*, 17(2):330–344, 2006.

Susan L Epstein. Capitalizing on conflict: The forr architecture. In *Proceedings of the Workshop on Computational Architectures for Supporting Machine Learning* *and Knowledge Acquisition, Ninth International Machine Learning Conference*, 1992.

Oren Etzioni, Michele Banko, and Michael J Cafarella. Machine reading. In *AAAI*, volume 6, pages 1517–1519, 2006.

Alexander Fabisch, Yohannes Kassahun, Hendrik Wöhrle, and Frank Kirchner. Learning in compressed space. *Neural Networks*, 42:83–93, 2013.

Xiaocong Fan, Shuang Sun, and John Yen. On shared situation awareness for supporting human decision-making teams. In *AAAI Spring Symposium: AI Technologies for Homeland Security*, pages 17–24, 2005.

Christiane Fellbaum. *WordNet.* Wiley Online Library, 1998.

Pier F Ferrari, Elisabetta Visalberghi, Annika Paukner, Leonardo Fogassi, Angela Ruggiero, and Stephen J Suomi. Neonatal imitation in rhesus macaques. *PLoS Biol*, 4(9):e302, 2006.

Melvin Fitting. First-order logic. In *First-Order Logic and Automated Theorem Proving*, pages 97–125. Springer, 1990.

Stan Franklin and FG Patterson Jr. The lida architecture: Adding new modes of learning to an intelligent, autonomous, software agent. *pat*, 703:764–1004, 2006.

Yoav Freund, Robert Schapire, and N Abe. A short introduction to boosting. *Journal-Japanese Society For Artificial Intelligence*, 14(771-780):1612, 1999.

Johannes Fritz and Kurt Kotrschal. On avian imitation: Cognitive and ethological perspectives. In Kerstin Dautenhahn and Chrystopher L. Nehaniv, editors, *Imitation in Animals and Artifacts*, pages 133–155. MIT Press, Cambridge, MA, USA, 2002. ISBN 0-262-04203-7. URL `http://dl.acm.org/citation.cfm?id=762896.762902`.

Bernd Fritzke et al. A growing neural gas network learns topologies. *Advances in neural information processing systems*, 7:625–632, 1995.

Bennett G Galef Jr. Imitation in animals: history, definition, and interpretation of data from the psychological laboratory. *Social learning: Psychological and biological perspectives*, 28, 1988.

Howard Gardner. *Frames of mind: The theory of multiple intelligences.* Basic books, 2011.

Arthur Gill et al. *Introduction to the theory of finite-state machines.* McGraw-Hill, 1962.

Alexandros Gkiokas and Alexandra I Cristea. Training a cognitive agent to acquire and represent knowledge from rss feeds onto conceptual graphs. *IARIA COGNITIVE*, pages 184–194, 2014a.

Alexandros Gkiokas and Alexandra I Cristea. Unsupervised neural controller for reinforcement learning action-selection: Learning to represent knowledge. In *Neural Network Applications in Electrical Engineering (NEUREL), 2014 12th Symposium on*, pages 99–104. IEEE, 2014b.

Alexandros Gkiokas and Alexandra I Cristea. Cognitive agents and machine learning by example: Representation with conceptual graphs. submitted, 2016a.

Alexandros Gkiokas and Alexandra I Cristea. Deep learning and encoding in natural language understanding: Sparse and dense encoding schemes for neural-based parsing. submitted, 2016b.

Alexandros Gkiokas, Alexandra I Cristea, and Matthew Thorpe. Self-reinforced meta learning for belief generation. In *Research and Development in Intelligent Systems XXXI*, pages 185–190. Springer, 2014.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Aistats*, volume 9, pages 249–256, 2010.

Fernand Gobet, Peter CR Lane, Steve Croker, Peter CH Cheng, Gary Jones, Iain Oliver, and Julian M Pine. Chunking mechanisms in human learning. *Trends in cognitive sciences*, 5(6):236–243, 2001.

Ben Goertzel and Cassio Pennachin. *Artificial general intelligence*, volume 2. Springer, 2007.

Dina Goldin and Peter Wegner. The church-turing thesis: Breaking the myth. In *Conference on Computability in Europe*, pages 152–168. Springer, 2005.

Paul Gorrell. *Syntax and parsing*, volume 76. Cambridge University Press, 2006.

Edward Grefenstette, Phil Blunsom, Nando de Freitas, and Karl Moritz Hermann. A deep architecture for semantic parsing. *arXiv preprint arXiv:1404.7296*, 2014.

Julie Grèzes, Jorge L Armony, James Rowe, and Richard E Passingham. Activations related to mirror and canonical neurones in the human brain: an fmri study. *Neuroimage*, 18(4):928–937, 2003.

David Guthrie, Ben Allison, Wei Liu, Louise Guthrie, and Yorick Wilks. A closer look at skip-gram modelling. In *Proceedings of the 5th international Conference on Language Resources and Evaluation (LREC-2006)*, pages 1–4, 2006.

Pentti O Haikonen. *The cognitive approach to conscious machines.* Imprint Academic, 2003.

Pentti O Haikonen. *Robot brains: circuits and systems for conscious machines.* John Wiley & Sons, 2007.

Pentti O Haikonen. *Consciousness and robot sentience*, volume 2. World Scientific, 2012.

Pentti OA Haikonen. Qualia and conscious machines. *International Journal of Machine Consciousness*, 1(02):225–234, 2009.

Daniel Conrad Halbert. *Programming by example.* PhD thesis, University of California, Berkeley, 1984.

Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1-3):335–346, 1990.

David Hart and Ben Goertzel. Opencog: A software framework for integrative artificial general intelligence. *Frontiers in Artificial Intelligence and Applications*, 171:468, 2008.

John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Applied statistics*, pages 100–108, 1979.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

Jeff Heaton. *Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks.* Heaton Research Inc, 2015.

Donald Olding Hebb. *The organization of behavior; a neuropsychological theory.* Wiley, 1949.

Charles T Hemphill, John J Godfrey, and George R Doddington. The atis spoken language systems pilot corpus. In *Proceedings of the DARPA speech and natural language workshop*, pages 96–101, 1990.

Louis M. Herman. Vocal, social, and self-imitation by bottlenosed dolphins. In Kerstin Dautenhahn and Chrystopher L. Nehaniv, editors, *Imitation in Animals and Artifacts*, pages 63–108. MIT Press, Cambridge, MA, USA, 2002. ISBN 0-262-04203-7. URL `http://dl.acm.org/citation.cfm?id=762896.762900`.

John Hertz, Anders Krogh, and Richard G Palmer. *Introduction to the theory of neural computation*, volume 1. Basic Books, 1991.

Cecilia Heyes. Transformational and associative theories of imitation. In Kerstin Dautenhahn and Chrystopher L. Nehaniv, editors, *Imitation in Animals and Artifacts*, pages 501–523. MIT Press, Cambridge, MA, USA, 2002. ISBN 0-262-04203-7. URL `http://dl.acm.org/citation.cfm?id=762896.762916`.

James J Higgins. *Introduction to modern nonparametric statistics.* Cengage Learning, 2003.

Matthew Honnibal and Mark Johnson. An improved non-monotonic transition system for dependency parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, page 13731378, 2015.

John J Hopfield. Artificial neural networks. *IEEE Circuits and Devices Magazine*, 4(5):3–10, 1988.

Eduard Hovy, Mitchell Marcus, Martha Palmer, Lance Ramshaw, and Ralph Weischedel. Ontonotes: the 90% solution. In *Proceedings of the human language technology conference of the NAACL, Companion Volume: Short Papers*, pages 57–60. Association for Computational Linguistics, 2006.

Ronald A Howard. *Dynamic programming and Markov processes.* MIT Press, 1970.

Marco Iacoboni. Neural mechanisms of imitation. *Current opinion in neurobiology*, 15(6):632–637, 2005.

Marco Iacoboni and Mirella Dapretto. The mirror neuron system and the consequences of its dysfunction. *Nature Reviews Neuroscience*, 7(12):942–951, 2006.

Marco Iacoboni, Roger P Woods, Marcel Brass, Harold Bekkering, John C Mazziotta, and Giacomo Rizzolatti. Cortical mechanisms of human imitation. *Science*, 286(5449):2526–2528, 1999.

Jelena Ignjatović, Miroslav Ćirić, Stojan Bogdanović, and Tatjana Petković. Myhill–nerode type theory for fuzzy languages and automata. *Fuzzy sets and Systems*, 161(9):1288–1324, 2010.

François Felix Ingrand, Michael P Georgeff, and Anand S Rao. An architecture for real-time reasoning and system control. *IEEE expert*, 7(6):34–44, 1992.

Rolly Intan, Chi-Hung Chi, Henry N Palit, and Leo W Santoso. *Intelligence in the Era of Big Data: 4th International Conference on Soft Computing, Intelligent Systems, and Information Technology, ICSIIT 2015, Bali, Indonesia, March 11-14, 2015. Proceedings*, volume 516. Springer, 2015.

Jay J Jiang and David W Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. *arXiv preprint cmp-lg/9709008*, 1997.

Anjali Ganesh Jivani et al. A comparative study of stemming algorithms. *Int. J. Comp. Tech. Appl*, 2(6):1930–1938, 2011.

Susan S Jones. The development of imitation in infancy. *Philosophical Transactions of the Royal Society of London B: Biological Sciences*, 364(1528):2325–2335, 2009.

A Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. *Advances in neural information processing systems*, 14:841, 2002.

Dan Jurafsky and James H Martin. *Speech & language processing.* Pearson Education India, 2000.

Marcel Adam Just and Sashank Varma. A hybrid architecture for working memory: Reply to macdonald and christiansen (2002). *Psychological Review*, 2002.

Nal Kalchbrenner, Edward Grefenstette, and Phil Blunsom. A convolutional neural network for modelling sentences. *arXiv preprint arXiv:1404.2188*, 2014.

László Kalmár. An argument against the plausibility of Church's thesis. In *Constructivity in mathematics: Proceedings of the colloquium held at Amsterdam, 1957 (edited by A. Heyting)*, Studies in Logic and the Foundations of Mathematics, pages 72–80, Amsterdam, 1957. North-Holland Publishing Co.

Nanda Kambhatla. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, page 22. Association for Computational Linguistics, 2004.

Jin-Dong Kim, Yue Wang, Toshihisa Takagi, and Akinori Yonezawa. Overview of genia event task in bionlp shared task 2011. In *Proceedings of the BioNLP*

Shared Task 2011 Workshop, pages 7–15. Association for Computational Linguistics, 2011.

Ross D King, Stephen Muggleton, Richard A Lewis, and MJ Sternberg. Drug design by machine learning: The use of inductive logic programming to model the structure-activity relationships of trimethoprim analogues binding to dihydrofolate reductase. *Proceedings of the national academy of sciences*, 89(23):11322–11326, 1992.

Witold Kinsner. Towards cognitive machines: Multiscale measures and analysis. In *2006 5th IEEE International Conference on Cognitive Informatics*, volume 1, pages 8–14. IEEE, 2006.

Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*, pages 340–347. ACM, 1997.

Graham Klyne and Jeremy J Carroll. Resource description framework (rdf): Concepts and abstract syntax. *W3C Recommendation*, 2005.

Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

Boicho N Kokinov. The dual cognitive architecture: A hybrid multi-agent approach. In *ECAI*, pages 203–207, 1994.

Ben Taskar Carlos Guestrin Daphne Koller. Max-margin markov networks. *Advances in Neural Information Processing Systems (NIPS)*, 17, 2014.

Richard E Korf. Linear-space best-first search. *Artificial Intelligence*, 62(1):41–78, 1993.

Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Ray Kurzweil. *The singularity is near: When humans transcend biology.* Penguin, 2005.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, pages 282–289, 2001.

John E Laird, Allen Newell, and Paul S Rosenbloom. Soar: An architecture for general intelligence. *Artificial intelligence*, 33(1):1–64, 1987.

Pat Langley and Dongkyu Choi. A unified cognitive architecture for physical agents. In *Proceedings of the National Conference on Artificial Intelligence*, volume 21, page 1469. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2006.

Pat Langley, Daniel Shapiro, Meg Aycinena, and Michael Siliski. A value-driven architecture for intelligent behavior. In *Proceedings of the IJCAI-2003 Workshop on Cognitive Modeling of Agents and Multi-Agent Interactions*, pages 10–18, 2003.

Pat Langley, Kirstin Cummings, and Daniel Shapiro. Hierarchical skills and cognitive architectures. In *Proceedings of the twenty-sixth annual conference of the cognitive science society*, pages 779–784. Citeseer, 2004.

Pat Langley, John E Laird, and Seth Rogers. Cognitive architectures: Research issues and challenges. *Cognitive Systems Research*, 10(2):141–160, 2009.

Anna T Lawniczak and Bruno N Di Stefano. Computational intelligence based architecture for cognitive agents. *Procedia Computer Science*, 1(1):2227–2235, 2010.

Agnieszka Ławrynowicz and Jedrzej Potoniec. Fr-ont: An algorithm for frequent concept mining with formal ontologies. In *International Symposium on Methodologies for Intelligent Systems*, pages 428–437. Springer, 2011.

Quoc V Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013.

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

Yann A LeCun, Léon Bottou, Genevieve B Orr, and Klaus-Robert Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–48. Springer, 2012.

Chin Yang Lee. An algorithm for path connections and its applications. *Electronic Computers, IRE Transactions on*, EC-10(3):346–365, 1961.

Honglak Lee, Chaitanya Ekanadham, and Andrew Y Ng. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems*, pages 873–880, 2008.

Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-516-1. doi: 10.1145/1553374.1553453. URL `http://doi.acm.org/10.1145/1553374.1553453`.

Tao Lei, Yu Xin, Yuan Zhang, Regina Barzilay, and Tommi Jaakkola. Low-rank tensors for scoring dependency structures. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1381–1391, Baltimore, Maryland, June 2014. Association for Computational Linguistics. URL `http://www.aclweb.org/anthology/P/P14/P14-1130`.

Daniel Leivant. Higher order logic., 1994.

Hector J Levesque and Ronald Brachman. *A fundamental tradeoff in knowledge representation and reasoning.* Laboratory for Artificial Intelligence Research, Fairchild, Schlumberger, 1984.

Mu Li, Tong Zhang, Yuqiang Chen, and Alexander J Smola. Efficient mini-batch training for stochastic optimization. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 661–670. ACM, 2014.

Percy Liang and Christopher Potts. Bringing machine learning and compositional semantics together. *Annu. Rev. Linguist.*, 1(1):355–376, 2015.

Henry Lieberman. Programming by example. *Communications of the ACM*, 43(3): 72–72, 2000.

Henry Lieberman. *Your wish is my command: Programming by example.* Morgan Kaufmann, 2001.

Henry Liebermann, Bonnie A. Nardi, and David J. Wright. Training agents to recognize text by example. In Henry Liebermann, editor, *Your wish is my command: Programming by example*, chapter 12, pages 227–244. Morgan Kaufmann, 2001.

Dekang Lin. An information-theoretic definition of similarity. In *ICML*, volume 98, pages 296–304, 1998.

Dekang Lin and Patrick Pantel. Dirt@ sbt@ discovery of inference rules from text. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 323–328. ACM, 2001a.

Dekang Lin and Patrick Pantel. Discovery of inference rules for question-answering. *Natural Language Engineering*, 7(04):343–360, 2001b.

Seppo Linnainmaa. The representation of the cumulative rounding error of an algorithm as a taylor expansion of the local rounding errors. *Master's Thesis (in Finnish), Univ. Helsinki*, pages 6–7, 1970.

Seymour Lipschutz and L Gersting Judith. *Schaum's Outline of Theory and Problems*. McGraw-Hill, 1916.

Bing Liu. Sentiment analysis and opinion mining. *Synthesis lectures on human language technologies*, 5(1):1–167, 2012.

Ting Liu, Jinshan Ma, Huijia Zhu, and Sheng Li. Dependency parsing based on dynamic local optimization. In *Proceedings of the Tenth Conference on Computational Natural Language Learning*, pages 211–215. Association for Computational Linguistics, 2006.

Moshe Looks, Andrew Levine, G Adam Covington, Ronald P Loui, John W Lockwood, and Young H Cho. Streaming hierarchical clustering for concept mining. In *2007 IEEE Aerospace Conference*, pages 1–12. IEEE, 2007.

Maryellen C MacDonald, Neal J Pearlmutter, and Mark S Seidenberg. The lexical nature of syntactic ambiguity resolution. *Psychological review*, 101(4):676, 1994.

James MacQueen et al. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, volume 1, pages 281–297. Oakland, CA, USA., 1967.

Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini. Building a large annotated corpus of english: The penn treebank. *Computational linguistics*, 19(2):313–330, 1993.

André FT Martins and Mariana SC Almeida. Priberam: A turbo semantic parser with second order features. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 471–476, 2014.

André FT Martins, Noah A Smith, Eric P Xing, Pedro MQ Aguiar, and Mário AT Figueiredo. Turbo parsers: Dependency parsing by approximate variational inference. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 34–44. Association for Computational Linguistics, 2010.

André FT Martins, Noah A Smith, Pedro MQ Aguiar, and Mário AT Figueiredo. Dual decomposition with many overlapping components. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 238–249. Association for Computational Linguistics, 2011.

André FT Martins, Miguel B Almeida, and Noah A Smith. Turning on the turbo: Fast third-order non-projective turbo parsers. In *Proceedings of the Conference*, page 617, 2013.

Makoto Matsumoto and Takuji Nishimura. Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 8(1):3–30, 1998.

Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

Richard McDaniel. Demonstrating the hidden features that make an application work. In Henry Liebermann, editor, *Your wish is my command: Programming by example*, chapter 8, pages 163–174. Morgan Kaufmann, 2001.

Ryan McDonald, Koby Crammer, and Fernando Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd annual meeting on association for computational linguistics*, pages 91–98. Association for Computational Linguistics, 2005a.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530. Association for Computational Linguistics, 2005b.

Ryan T McDonald and Fernando CN Pereira. Online learning of approximate dependency parsing algorithms. In *EACL*, 2006.

Aditya Krishna Menon, Omer Tamuz, Sumit Gulwani, Butler W Lampson, and Adam Kalai. A machine learning framework for programming by example. *ICML (1)*, 28:187–195, 2013.

Charles E Metz. Basic principles of roc analysis. *Seminars in Nuclear Medicine*, 8 (4):283–298, 1978.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013b.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *HLT-NAACL*, pages 746–751, 2013c.

Marvin L Minski and Seymour A Papert. Perceptrons: an introduction to computational geometry. *MA: MIT Press, Cambridge*, 1969.

Marvin L Minsky. *Computation: finite and infinite machines.* Prentice-Hall, Inc., 1967.

Mike Mintz, Steven Bills, Rion Snow, and Dan Jurafsky. Distant supervision for relation extraction without labelled data. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, pages 1003–1011. Association for Computational Linguistics, 2009.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

Francesmary Modugno. *Extending end-user programming in a visual shell with programming by demonstration and graphical language techniques.* PhD thesis, Carnegie Mellon University, 1996.

Alessandro Montalto, Giovanni Tessitore, and Roberto Prevete. A linear approach for sparse coding by a two-layer neural network. *Neurocomputing*, 149:1315–1323, 2015.

Manuel Montes-y Gómez, Alexander Gelbukh, and Aurelio López-López. Text mining at detail level using conceptual graphs. In *Conceptual Structures: Integration and Interfaces*, pages 122–136. Springer, 2002.

Jorge J Moré. The levenberg-marquardt algorithm: implementation and theory. In *Numerical analysis*, pages 105–116. Springer, 1978.

Frederic Morin and Yoshua Bengio. Hierarchical probabilistic neural network language model. In *Aistats*, volume 5, pages 246–252. Citeseer, 2005.

Stephen Muggleton. Alan turing and the development of artificial intelligence. *AI communications*, 27(1):3–10, 2014.

Brad Myers and Richard McDaniel. Demonstrational interfaces: Sometimes you need a little intelligence, sometimes you need a lot. In Henry Liebermann, editor, *Your wish is my command: Programming by example*, chapter 3, pages 48–60. Morgan Kaufmann, 2001.

Brad Myers and Brad Vander Zanden. Environment for rapidly creating interactive design tools. *The Visual Computer*, 8(2):94–116, 1992.

Brad A Myers, Francesmary Modugno, Rich McDaniel, David Kosbie, Andrew Werth, Rob Miller, John Pane, James Landay, Jade Goldstein, and Matthew A Goldberg. The demonstrational interfaces project at cmu. In *1996 AAAI Symposium*, 1995.

Masako Myowa-Yamakoshi, Masaki Tomonaga, Masayuki Tanaka, and Tetsuro Matsuzawa. Imitation in neonatal chimpanzees (pan troglodytes). *Developmental science*, 7(4):437–442, 2004.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 807–814, 2010.

Shin'ichiro Nakaoka, Atsushi Nakazawa, Fumio Kanehiro, Kenji Kaneko, Mitsuharu Morisawa, Hirohisa Hirukawa, and Katsushi Ikeuchi. Learning from observation paradigm: Leg task models for enabling a biped humanoid robot to imitate human dances. *The International Journal of Robotics Research*, 26(8):829–844, 2007.

Tetsuya Nasukawa and Jeonghee Yi. Sentiment analysis: Capturing favorability using natural language processing. In *Proceedings of the 2nd international conference on Knowledge capture*, pages 70–77. ACM, 2003.

A Newell and JC Shaw. A variety op intelligent learning in a general problem solver. *RAND Report P-1742, dated July*, 6, 1959.

Allen Newell and Herbert Simon. The logic theory machine–a complex information processing system. *IRE Transactions on Information Theory*, 2(3):61–79, 1956.

Andrew Y Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. In *Proceedings of the twenty-first international conference on Machine learning*, page 78. ACM, 2004.

Jiquan Ngiam, Adam Coates, Ahbik Lahiri, Bobby Prochnow, Quoc V Le, and Andrew Y Ng. On optimization methods for deep learning. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 265–272, 2011a.

Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 689–696, 2011b.

John Nickolls, Ian Buck, Michael Garland, and Kevin Skadron. Scalable parallel programming with cuda. *Queue*, 6(2):40–53, 2008.

Steffen Nissen. Implementation of a fast artificial neural network library (fann). *Report, Department of Computer Science University of Copenhagen (DIKU)*, 31, 2003.

Joakim Nivre. An efficient algorithm for projective dependency parsing. In *Proceedings of the 8th International Workshop on Parsing Technologies (IWPT*. Citeseer, 2003.

Joakim Nivre and Mario Scholz. Deterministic dependency parsing of english text. In *Proceedings of the 20th international conference on Computational Linguistics*, page 64. Association for Computational Linguistics, 2004.

Vilém Novák, Irina Perfilieva, and Jiri Mockor. *Mathematical principles of fuzzy logic*, volume 517. Springer Science & Business Media, 2012.

Marek Obitko. *Translations between ontologies in multi-agent systems*. PhD thesis, Czech Technical University, 2007.

Franz Josef Och and Hermann Ney. A systematic comparison of various statistical alignment models. *Computational linguistics*, 29(1):19–51, 2003.

Bruno A Olshausen and David J Field. Sparse coding with an overcomplete basis set: A strategy employed by v1? *Vision research*, 37(23):3311–3325, 1997.

Bruno A Olshausen and David J Field. Sparse coding of sensory inputs. *Current opinion in neurobiology*, 14(4):481–487, 2004.

John D Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E Lefohn, and Timothy J Purcell. A survey of general-purpose computation on graphics hardware. *Computer graphics forum*, 26(1):80–113, 2007.

Gordon W. Paynter and Ian H. Witten. Domain-independent programming by demonstration in existing applications. In Henry Liebermann, editor, *Your wish is my command: Programming by example*, chapter 15, pages 297–320. Morgan Kaufmann, 2001.

Ismael Peña-López et al. Manual for measuring ict access and use by households and individuals. Technical report, ITU, 2009.

Matt Pharr and Randima Fernando. *Gpu gems 2: programming techniques for high-performance graphics and general-purpose computation*. Addison-Wesley Professional, 2005.

Gualtiero Piccinini. The physical church–turing thesis: Modest or bold? *The British Journal for the Philosophy of Science*, page axr016, 2011.

Phil Picton. *Introduction to neural networks*. Macmillan Publishers Limited, 1994.

Fiora Pirri. The usual objects: a first draft on decomposing and reassembling familiar objects images. In *Proceedings of XXVII Annual Conference of the Cognitive Science Society*, pages 1773–1778, 2005.

David Martin Powers. Evaluation: from precision, recall and f-measure to roc, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2011.

Sameer S Pradhan, Wayne Ward, Kadri Hacioglu, James H Martin, and Daniel Jurafsky. Shallow semantic parsing using support vector machines. In *HLT-NAACL*, pages 233–240, 2004.

Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

Roy Rada, Hafedh Mili, Ellen Bicknell, and Maria Blettner. Development and application of a metric on semantic nets. *Systems, Man and Cybernetics, IEEE Transactions on*, 19(1):17–30, 1989.

Vilayanur S Ramachandran. Mirror neurons and imitation learning as the driving force behind the great leap forward in human evolution. *Edge Website article http://www. edge. org/3rd_culture/ramachandran/ramachandran_p1. html*, 2000.

Ruziana Binti Mohamad Rasli, Faudziah Ahmad, and Siti Sakira Kamaruddin. A comparative study of conceptual graph and concept map. *Journal of Engineering and Applied Sciences*, 9(9):1442–1446, 2014.

Mohammad Sadegh Rasooli and Joel Tetreault. Yara parser: A fast and accurate dependency parser. *arXiv preprint arXiv:1503.06733*, 2015.

Raimundo Real and Juan M Vargas. The probabilistic basis of jaccard's index of similarity. *Systematic biology*, pages 380–385, 1996.

Martin Riedmiller and Heinrich Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference On*, pages 586–591. IEEE, 1993.

C. J. Van Rijsbergen. *Information Retrieval.* Butterworth-Heinemann, Newton, MA, USA, 2nd edition, 1979. ISBN 0408709294.

Edmund T Rolls and Alessandro Treves. The relative advantages of sparse versus distributed encoding for associative neuronal networks in the brain. *Network: computation in neural systems*, 1(4):407–421, 1990.

Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

Kenji Sagae. Analysis of discourse structure with syntactic dependencies and data-driven shift-reduce parsing. In *Proceedings of the 11th International Conference on Parsing Technologies*, pages 81–84. Association for Computational Linguistics, 2009.

Kenji Sagae and Alon Lavie. A best-first probabilistic shift-reduce parser. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 691–698. Association for Computational Linguistics, 2006.

Ruslan Salakhutdinov and Geoffrey E Hinton. Deep boltzmann machines. In *AISTATS*, volume 1, page 3, 2009.

Robert E Schapire and Yoram Singer. Improved boosting algorithms using confidence-rated predictions. *Machine learning*, 37(3):297–336, 1999.

John R Searle. Church-turing thesis. *The MIT Encyclopedia of the Cognitive Sciences*, page 115, 2001.

Philip H. K. Seymour. Cognitive architecture of early reading. In Ingvar Lundberg, Finn Egil Tønnessen, and Ingolv Austad, editors, *Dyslexia: Advances in Theory and Practice*, pages 59–73. Springer Netherlands, Dordrecht, 1999. ISBN 978-94-011-4667-8.

David F Shanno. On broyden-fletcher-goldfarb-shanno method. *Journal of Optimization Theory and Applications*, 46(1):87–94, 1985.

K Gnana Sheela and SN Deepa. Review on methods to fix number of hidden neurons in neural networks. *Mathematical Problems in Engineering*, 2013, 2013.

Lei Shi and Rada Mihalcea. Putting pieces together: Combining framenet, verbnet and wordnet for robust semantic parsing. In *Computational linguistics and intelligent text processing*, pages 100–111. Springer, 2005.

Stuart M Shieber. Sentence disambiguation by a shift-reduce parsing technique. In *Proceedings of the 21st annual meeting on Association for Computational Linguistics*, pages 113–118. Association for Computational Linguistics, 1983.

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

Burrhus Frederic Skinner. *Contingencies of reinforcement: A theoretical analysis*, volume 3. BF Skinner Foundation, 2014.

Richard D Smallwood and Edward J Sondik. The optimal control of partially observable markov processes over a finite horizon. *Operations Research*, 21(5):1071–1088, 1973.

David Canfield Smith, Allen Cypher, and Tesler Larry. Novice programming comes of age. In Henry Liebermann, editor, *Your wish is my command: Programming by example*, chapter 1, pages 8–19. Morgan Kaufmann, 2001.

Paul Smolensky. Connectionist ai, symbolic ai, and the brain. *Artificial Intelligence Review*, 1(2):95–109, 1987.

Moshe Sniedovich. *Dynamic programming: foundations and principles.* CRC press, 2010.

Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

Richard Socher, John Bauer, Christopher D Manning, and Andrew Y Ng. Parsing with compositional vector grammars. In *ACL (1)*, pages 455–465, 2013a.

Richard Socher, Alex Perelygin, Jean Y Wu, Jason Chuang, Christopher D Manning, Andrew Y Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the conference on empirical methods in natural language processing (EMNLP)*, volume 1631, page 1642. Citeseer, 2013b.

J. F. Sowa. *Conceptual Structures: Information Processing in Mind and Machine.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1984. ISBN 0-201-14472-7.

John F Sowa. *Knowledge representation: logical, philosophical, and computational foundations.* Course Technology, 1999.

Nitish Srivastava, Geoffrey E Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Stackexchange. How to choose the number of hidden layers and nodes in a feedforward neural network?, 2015. URL `http://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw` `1097#1097`. [Online; accessed 2015-09-30].

William Stallings. *Handbook of computer-communications standards; Vol. 1: the open systems interconnection (OSI) model and OSI-related standards.* Macmillan Publishing Co., Inc., 1987.

Mike Stannett. X-machines and the halting problem: Building a super-turing machine. *Formal Aspects of Computing*, 2(1):331–341, 1990.

Robert J Sternberg. *Handbook of human intelligence.* CUP Archive, 1982.

Morag Stuart and Max Coltheart. Does reading develop in a sequence of stages? *Cognition*, 30(2):139–181, 1988. ISSN 0010-0277.

Wael Suleiman, Eiichi Yoshida, Fumio Kanehiro, Jean-Paul Laumond, and André Monin. On human motion imitation by humanoid robot. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2697–2704. IEEE, 2008.

Ron Sun and Xi Zhang. Accounting for a variety of reasoning data within a cognitive architecture. *Journal of Experimental & Theoretical Artificial Intelligence*, 18(2): 169–191, 2006.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, 1998. ISBN 0-262-19398-1.

Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984.

Idan Szpektor and Ido Dagan. Learning entailment rules for unary templates. In *Proceedings of the 22nd International Conference on Computational Linguistics-Volume 1*, pages 849–856. Association for Computational Linguistics, 2008.

Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708, 2014.

Lappoon R Tang and Raymond J Mooney. Using multiple clause constructors in inductive logic programming for semantic parsing. In *Machine Learning: ECML 2001*, pages 466–477. Springer, 2001.

Chris J Thompson, Sahngyun Hahn, and Mark Oskin. Using modern graphics architectures for general-purpose computing: a framework and analysis. In *Proceedings of the 35th annual ACM/IEEE international symposium on Microarchitecture*, pages 306–317. IEEE Computer Society Press, 2002.

Edward L Thorndike. The mental life of the monkeys. *The Psychological Review: Monograph Supplements*, 3(5):i, 1901.

Jakub Mikolaj Tomczak. Improving neural networks with bunches of neurons modeled by kumaraswamy units: Preliminary study. *arXiv preprint arXiv:1505.02581*, 2015.

Godfried T Toussaint. Solving geometric problems with the rotating calipers. In *Proc. IEEE Melecon*, volume 83, page A10, 1983.

John C Trueswell, Michael K Tanenhaus, and Susan M Garnsey. Semantic influences on parsing: Use of thematic role information in syntactic ambiguity resolution. *Journal of memory and language*, 33(3):285, 1994.

Reut Tsarfaty, Joakim Nivre, and Evelina Ndersson. Evaluating dependency parsing: robust and heuristics-free cross-nnotation evaluation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 385–396. Association for Computational Linguistics, 2011.

Reut Tsarfaty, Joakim Nivre, and Evelina Andersson. Cross-framework evaluation for statistical parsing. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 44–54. Association for Computational Linguistics, 2012a.

Reut Tsarfaty, Joakim Nivre, and Evelina Andersson. Joint evaluation of morphological segmentation and syntactic parsing. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Short Papers-Volume 2*, pages 6–10. Association for Computational Linguistics, 2012b.

Yoshimasa Tsuruoka, Yusuke Miyao, and Jun'ichi Kazama. Learning with lookahead: can history-based models rival globally optimized models? In *Proceedings of the Fifteenth Conference on Computational Natural Language Learning*, pages 238–246. Association for Computational Linguistics, 2011.

Alan M Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

Peter D Turney, Patrick Pantel, et al. From frequency to meaning: Vector space models of semantics. *Journal of artificial intelligence research*, 37(1):141–188, 2010.

Stephen Ullmann. The principles of semantics. *American Anthropologist*, 1959.

Johan Van Benthem and Kees Doets. Higher-order logic. In *Handbook of philosophical logic*, pages 275–329. Springer, 1983.

Jorge Villalon and Rafael A Calvo. Concept extraction from student essays, towards concept map mining. In *2009 Ninth IEEE International Conference on Advanced Learning Technologies*, 2009.

Elisabetta Visalberghi and Dorothy Fragaszy. "do monkeys ape?": Ten years after. In Kerstin Dautenhahn and Chrystopher L. Nehaniv, editors, *Imitation in Animals and Artifacts*, pages 471–499. MIT Press, Cambridge, MA, USA, 2002. ISBN 0-262-04203-7. URL `http://dl.acm.org/citation.cfm?id=762896.762915`.

Andreas Vlachos. Evaluating unsupervised learning for natural language processing tasks. In *Proceedings of the First workshop on Unsupervised Learning in NLP*, pages 35–42. Association for Computational Linguistics, 2011.

Andreas Vlachos. An investigation of imitation learning algorithms for structured prediction. In *EWRL*, pages 143–154. Citeseer, 2012.

Andreas Vlachos and Stephen Clark. A new corpus and imitation learning framework for context-dependent semantic parsing. *Transactions of the Association for Computational Linguistics*, 2:547–559, 2014.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

Duncan J Watts and Steven H Strogatz. Collective dynamics of small-worldnetworks. *nature*, 393(6684):440–442, 1998.

David Weiss, Chris Alberti, Michael Collins, and Slav Petrov. Structured training for neural network transition-based parsing. *arXiv preprint arXiv:1506.06158*, 2015.

G. Weiss. *Aspects and Applications of the Random Walk (Random Materials & Processes S.)*. North-Holland, 2005. ISBN 0444816062. URL `http://www.worldcat.org/isbn/0444816062`.

George Weiss. Dynamic programming and markov processes. *Science*, 132(3428): 667–667, 1960. ISSN 0036-8075. doi: 10.1126/science.132.3428.667. URL `http://science.sciencemag.org/content/132/3428/667.1`.

Kathleen Anne Wellman. *La Mettrie: Medicine, Philosophy, and Enlightenment*. Duke University Press, 1992.

Paul Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, 1974.

Paul J Werbos. Applications of advances in nonlinear sensitivity analysis. In *System modeling and optimization*, pages 762–770. Springer, 1982.

Norbert Wiener. *Cybernetics: Control and communication in the animal and the machine*. Wiley New York, 1948.

Theresa Wilson, Janyce Wiebe, and Paul Hoffmann. Recognizing contextual polarity in phrase-level sentiment analysis. In *Proceedings of the conference on human language technology and empirical methods in natural language processing*, pages 347–354. Association for Computational Linguistics, 2005.

Ben James Winer, Donald R Brown, and Kenneth M Michels. *Statistical principles in experimental design*, volume 2. McGraw-Hill New York, 1971.

Eric Winsberg. Simulated experiments: Methodology for a virtual world. *Philosophy of science*, 70(1):105–125, 2003.

Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1):37–52, 1987.

Yuk Wah Wong and Raymond J. Mooney. Learning for semantic parsing with statistical machine translation. In *Proceedings of the Main Conference on Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, HLT-NAACL '06, pages 439–446, Stroudsburg, PA, USA, 2006. Association for Computational Linguistics.

William A Woods and James G Schmolze. The kl-one family. *Computers & Mathematics with Applications*, 23(2-5):133–177, 1992.

Jesse O Wrenn, Peter D Stetson, and Stephen B Johnson. An unsupervised machine learning approach to segmentation of clinician-entered free text. In *AMIA Annual Symposium Proceedings*, volume 2007, page 811. American Medical Informatics Association, 2007.

E Maitland Wright. The number of connected sparsely edged graphs. *Journal of Graph Theory*, 1(4):317–330, 1977.

Thomas Wynn. Piaget, stone tools and the evolution of human intelligence. *World archaeology*, 17(1):32–43, 1985.

Hiroyasu Yamada and Yuji Matsumoto. Statistical dependency analysis with support vector machines. In *Proceedings of IWPT*, volume 3, pages 195–206, 2003.

Alexander Yates, Michael Cafarella, Michele Banko, Oren Etzioni, Matthew Broadhead, and Stephen Soderland. Textrunner: open information extraction on the web. In *Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations*, pages 25–26. Association for Computational Linguistics, 2007.

Lotfi A Zadeh. *Fuzzy sets, fuzzy logic, and fuzzy systems: selected papers by Lotfi A Zadeh*, volume 6. World Scientific, 1996.

Mehdi Rezaeian Zadeh, Seifollah Amin, Davar Khalili, and Vijay P Singh. Daily outflow prediction by multi layer perceptron with logistic sigmoid and tangent sigmoid activation functions. *Water resources management*, 24(11):2673–2688, 2010.

Brad Vander Zanden and Brad Myers. Automatic, look-and-feel independent dialog creation for graphical user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 27–34. ACM, 1990.

Amir Zeldes. The gum corpus: creating multilayer resources in the classroom. *Language Resources and Evaluation*, pages 1–32, 2016.

Dell Zhang and Wee Sun Lee. Question classification using support vector machines. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 26–32. ACM, 2003.

Hao Zhang and Ryan McDonald. Generalized higher-order dependency parsing with cube pruning. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 320–331. Association for Computational Linguistics, 2012.

Yuan Zhang, Tao Lei, Regina Barzilay, and Tommi Jaakkola. Greed is good if randomized: New inference for dependency parsing. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1013–1024. Association for Computational Linguistics, 2014a.

Yuan Zhang, Tao Lei, Regina Barzilay, Tommi Jaakkola, and Amir Globerson. Steps to excellence: Simple inference with refined scoring of dependency trees. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, pages 197–207. Association for Computational Linguistics, 2014b.

Yue Zhang and Stephen Clark. A tale of two parsers: investigating and combining graph-based and transition-based dependency parsing using beam-search. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 562–571. Association for Computational Linguistics, 2008.

Yue Zhang and Joakim Nivre. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 188–193. Association for Computational Linguistics, 2011.

Zhisong Zhang and Hai Zhao. High-order graph-based neural dependency parsing. In *29th Pacific Asia Conference on Language, Information and Computation pages*, pages 114–123, Shanghai, China, 2015.

Maosheng Zhong, Jianyong Duan, and Jian Zou. Indexing conceptual graph for abstracts of books. In *Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on*, volume 3, pages 1816–1820. IEEE, 2011.

Jie Zhou and Wei Xu. End-to-end learning of semantic role labelling using recurrent neural networks. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 2015.

Muhua Zhu, Yue Zhang, Wenliang Chen, Min Zhang, and Jingbo Zhu. Fast and accurate shift-reduce constituent parsing. In *ACL (1)*, pages 434–443, 2013.