OPTIMALITY AND HIERARCHICAL REPRESENTATION IN EMERGENT NEURAL TURING MACHINES AND THEIR VISUAL NAVIGATION

By

Zejia Zheng

A DISSERTATION

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

Computer Science – Doctor of Philosophy

2018

ABSTRACT

OPTIMALITY AND HIERARCHICAL REPRESENTATION IN EMERGENT NEURAL TURING MACHINES AND THEIR VISUAL NAVIGATION

By

Zejia Zheng

Traditional Turing Machines (TMs) are symbolic in the sense that representations in these TMs are static and hand-crafted. This paper presents a new kind of TM – emergent neural Turing Machine. By neural, we mean that the control of the TM has neurons as basic computing elements. By emergent, we mean that the internal representations are formed during learning without hand-crafting. Developmental Network-1 (DN-1) uses emergent representation to perform Turing Computation but the internal hierarchy is handcrafted with emergent features. The major novelty of the proposed TM (Developmental Network-2) over DN-1 is that the representational hierarchy inside DN-2 is emergent and fluid. DN-2 grows complex hierarchies by dynamically allowing initialization of neurons with different domains of connection. Its optimality in terms of maximum likelihood properties is established under the conditions of limited learning experience and resources. Although DN-2 is meant for general learning tasks, we experimented with a complex task— vision-guided navigation in simulated and natural worlds using DN-2. Real-world and simulated navigation experiments showed that DN-2 successfully learned rules of navigation with image and other inputs. The formed hierarchical representation in DN-2 focuses on important navigation features like road edges while disregarding the distractors like shadows edges.

TABLE OF CONTENTS

LIST OF TABLES vi

LIST OF FIGURES vii

LIST OF ALGORITHMS x

CHAPTER 1 INTRODUCTION 1
- 1.1 Related Work 3
- 1.2 Optimality of DN-2 8
- 1.3 Discussion: Sensorimotor Recursive Experiments vs. Batched Data Classification Experiments 11
- 1.4 Example: Shadows edges and road edges 11

CHAPTER 2 DEVELOPMENTAL NETWORK 2 15
- 2.1 Main Procedure 15
- 2.2 Details of the algorithm 16
  - 2.2.1 Patterning 16
  - 2.2.2 Response Computation and Competition 17
  - 2.2.3 Hebbian Learning of excitation 18
  - 2.2.4 Initialization 18
  - 2.2.5 Controlling $\alpha(t)$ for multistage learning 18
  - 2.2.6 Receptive fields initialization 19
  - 2.2.7 Synaptic maintenance 20

CHAPTER 3 MAXIMUM LIKELIHOOD ESTIMATION IN DN-2 22
- 3.1 Review of the three theorems in DN-1 22
- 3.2 Definition of DN-2 23
- 3.3 Conditions on DN-2 learning 23
- 3.4 Lemma 1: DN-2 optimizes its weight under maximum likelihood for each update, conditioned on **C**. 24
- 3.5 Theorem: DN-2 learns optimally under maximum likelihood from its incremental learning experience, conditioned on **C**. 24
- 3.6 Discussion 25
  - 3.6.1 On curse of dimensionality 25
  - 3.6.2 On over-fitting 25
  - 3.6.3 On local minima 26

CHAPTER 4 EMERGENT TURING MACHINE AND HIERARCHICAL REPRESENTATION IN DN2 27
- 4.1 The Challenges 27
- 4.2 Emergent FA learning in DN-2 28
- 4.3 Hierarchical representation in DN-2 29

4.4 Universal Turing Machine and Developmental Network 30
4.4.1 Definition of UTM 30
4.4.2 Differences between UTM and DN-2 31
4.5 Autonomous navigation needs Emergent Turing Machine 31

CHAPTER 5 SIMULATION WITH MAZE ENVIRONMENT 34
5.1 The environment and the teacher FA 34
5.2 The agent 35
5.2.1 Stage 1: Learning concept where, what and scale 37
5.3 Stage 2: Skills, route, and distance 37
5.3.1 Teaching skills 37
5.3.2 Teaching route concept 39
5.3.3 Evaluation of performance 40
5.4 Experiment results and analysis 40
5.4.1 Skilling learning and chaining 40
5.4.2 GPS blurring and noisy inputs 40
5.5 Discussion 41

CHAPTER 6 REAL-WORLD EXPERIMENTS AND RESULTS 42
6.1 Batched vision data learning 42
6.2 Real-world navigation: incremental training and testing 45
6.2.1 Training and testing 46
6.2.2 Visualization: Bottom-up weights of type 100 neurons 47
6.2.3 Visualization: Lateral weights from type 100 neurons to type 111 neurons . 48

CHAPTER 7 CONCLUSION 50

APPENDIX 51

BIBLIOGRAPHY 57

LIST OF TABLES

Table 2.1: Y neuron types and perfect match value θ 16

Table 4.1: Multiple types of neurons learning FA transition 28

Table 5.1: Experiment result 41

Table 6.1: DN-2 performance compared to benchmarks on AIML 2016 44

Table A.1: Real-time training and testing detail 52

Table A.2: Real-time testing results 53

LIST OF FIGURES

Figure 1.1: Summary of the DN-2 framework. DN-2 has seven types of internal neurons with different types of connections. Blue lines: top-down connection from Z. Red lines: lateral connection from Y. Green lines: bottom-up connection from X. In this paper, we show the optimality in the learning procedure of DN-2 by proving Lemma 1 and Theorem 1, shown in this figure. The verbal version of Theorem 1 can be found in the italic paragraph in the Introduction. Details about the theorem is presented in Sec. 3.5. 9

Figure 1.2: Demonstration of how low-level where-what representation facilitates learning complex navigation rules. Low-level 100 Y neurons look at local input patterns. Some neurons learn to recognize road edges (red square) at specific locations, and others learn to recognize shadow edges (blue square). Before synaptic maintenance, the high level Y neuron of type 011 learns different firing patterns from low-level 100 Y neurons. After synapse maintenance, the stable connections (on constant road edges) are kept while the unstable connections (on varying shadow edges) are cut from the high-level Y neurons, forming a shadow-invariant representation to learn the navigation rule "correct facing direction when left road edge is in the middle of the input image". 12

Figure 1.3: Demonstration of how top-down navigation context affects internal firing. In this example, all type 100 Y neurons are learning local features (road edges and shadow edges). But given different GPS navigation context (red letter in Z) motor, different high-level neurons (type 011) would fire. The attention of the network would thus be shifted toward the most relevant feature based on the current navigation context. 13

Figure 1.4: Demonstration of bottom-up and top-down effects in internal firing combined. The network shifts its attention according to the navigation context (red letter. The demonstrations in this Figure are synthetic examples for the purpose of demonstration.) 13

Figure 5.1: Simulated maze environment and the agent design. (a) and (b): simulated environment and corresponding GUI. The environment is 450 pixels by 450 pixels, with the maze no larger than nine by nine blocks. The agent is 20 pixels by 20 pixels, moving continuously in the maze environment. (1) wall blocks are red blocks in the GUI. (2) obstacle blocks are blue blocks in the GUI. (3) destination blocks are green block in the GUI. (4) reward/punishment blocks are yellow blocks in the GUI. (5) The agent's route is presented in the GUI. Black routes are skills already learned or going back routes. Red routes are novel skills to be learned. (6) Agent in the environment. Discussed in detail in subfigure (c). (7) and (8) Information panel for training and testing. (9) Traffic light block with two states. In (a) these traffic lights are at 'go' state, and in (b) these traffic lights are at 'stop' state. (10) 2D vision image is seen by the agent. (c) agent representation in GUI. (11) GPS signal from the environment. GPS indicates where the destination is in the simulated environment. But GPS is not aware of the obstacles in the environment thus the agent needs to learn skill "avoid obstacle". (12) Vision sensor of the agent. Here to simplify things we show seven vision lines out of the 43 vision lines forming a 210-degree vision ranging 75 pixels around the agent. 36

Figure 5.2: Skills and routes taught to the navigation agent in the simulation experiment. Skill 1: move forward when the next block is open. Skill 2: avoid obstacle and correct facing direction. Skill 3: turn left on the corner with GPS indicates left and avoid the obstacle. Skill 4: move through the narrow path. Skill 5: turn right when GPS indicates right. Skill 6: turn left when GPS indicates left. Skill 7: turn left and then turn right to reach the destination. During teaching, traffic lights would be placed along the route as shown in Fig. 5.1, the agent needs to learn to follow the rules of the traffic light as well. Although the agent has 43 vision lines as shown in Fig. 5.1, we show only 7 of these for clarity. A full video of training and testing is available at https://youtu.be/CbhS1qvWZn0. 39

Figure 6.1: Real-world experiment flow chart. The lower level Y neurons (type 100) detect local features in the input image. The higher level Y neurons are more robust against changes in local variances, while forming a temporal context for navigation. 42

Figure 6.2: Sample data from the AIML 2016 navigation dataset. All images are labeled with four Z motor concepts: action, gps, attention and type. Data is collected around university campus using an HTC One M8 android device. 43

Figure 6.3: AIML 2016 vision performance comparison among different network configurations. The X axis is the number of high level neurons in DN-2 (i.e. type 101, type 011 and type 111). The Y axis is the error action rate with four fold cross validation on the AIML testing data. Each configuration is labeled with (receptive size in type 100 neurons, number of type 100 neurons at each pixel location and the type of high level neurons used). DN-2 with local 100 neurons and 500 high level 111 neurons offers the best performance with reasonable computational resource requirements. 43

Figure 6.4: Real-time experiment setup. 1. Stereo USB camera connected to the mobile device. The cameras are streaming a 480 by 640 stereo RGB image pair in real-time. 2. Oneplus 5 phone running DN-2 equipped navigation application in CPU and GPU mode. 3. Moga pro controller for motor supervision and testing result recording. 4. The navigation application interface for training and testing. 45

Figure 6.5: Bottom-up weights learned by type 100 neurons. Each 100 neuron focuses on a 5 by 5 region on the input image as shown in Fig. 6.1, and conduct local dynamic top-k competition. As the neurons take stereo RGB inputs, we show only the weights corresponding to the left camera. Each small square is the 5x5 rgb weight corresponding to a specific neuron. The weight values are normalized within range 0 and 1. These bottom-up weights are projected to image space in Fig. 6.6 and Fig. 6.7. 47

Figure 6.6: Projected lateral weights for type 111 neurons with no synaptic maintenance. Type 111 neurons with low firing ages forms evenly distributed attention due to the local competition zones of lower level 100 neurons. 48

Figure 6.7: Projected lateral weights for type 111 neurons with synaptic maintenance. After synaptic maintenance the high-level neurons focuses attention on consistent road edges and become invariant to the changes in the highly variant shadow shapes. The connections from type 100 neurons to type 111 neurons shifts towards these consistent features, while the highly unstable connections are cut from the range of connection. 49

Figure 6.8: Training and testing routes around the university during different times of day and with different natural lighting conditions. Disjoint testing sessions were conducted along paths that the machine has not learned. 49

LIST OF ALGORITHMS

Algorithm 1: Environment for teaching individual skills 37

Algorithm 2: Environment for teaching route concept 38

CHAPTER 1

INTRODUCTION

The learning process in human children appears to be cumulative, with the task complexity progressively increasing through its exploration and interaction with the learning environment (e.g., supervised by a teacher or reinforced by self-exploration). The view that new knowledge must be constructed from existing knowledge is supported by researches in the field of developmental psychology (e.g., [40], [52]). Most importantly, this scaffolding learning principle has been demonstrated to happen across modalities, from language acquisition [2], to text understanding [34], and to motor skill learning [45].

In this paper we investigate the basic mechanisms that allow a neural network to learn like a child, i.e., learn incrementally from simple to complex while being task non-specific. By task-nonspecific, we mean that the programmer does not know about the environment when programming the learning agent. After birth, the agent learns to perform tasks according to its sensor and motor experience, while developing its hierarchical internal representations without the teacher manually selecting features. To demonstrate the power of such a computational framework (i.e. Developmental Network-2, or DN-2), a simulated agent and a real-world application equipped with DN-2 would learn to incrementally navigate in complex scenarios using only visual input (simulated vision input for the simulated agent and stereo RGB input for the real-world application) and GPS signals. The following conceptual steps guided us toward the targeted framework:

Concept 1: Turing Machine and Universal Turing Machine. A lot of the real-world traffic rules can be broken down into state transitions. For example, when in the state of "moving forward", an agent should take action "stop" if the input is recognized as "obstacle". These rules can be formulated as a Finite Automaton: $q(t) \xrightarrow{\sigma(t)} q(t+1)$, where $q(t)$ is the state of the agent at time t, and $\sigma(t)$ is the input observed by the agent at time t. However, in real-world navigation we not only act according to the input, but we are also changing the environment. E.g., we are actively changing the signal of the traffic light when we are pressing the wait button at a traffic light.

A more subtle example is that we often associate landmarks with the concept of distance during navigation, which is similar to hunters putting down stone piles in a forest to mark how far they have traveled. These actions are all changing the environment and are beyond the computational capability of a Finite Automaton (FA).

Turing machine (TM) [49, 18, 33] is a better computation model in this sense as it offers an additional read-write head that allows the agent to alter the input tape. The input tape in our case is the environment. Following the definition of Turing Machine, we can thus formulate a navigation sequence as $T = (Q, \Sigma, \Gamma, q_0, \delta)$, where Q is the set of states (navigation states as in "moving forward", "turning left", "arriving"), Σ is the input (current input images), Γ is the tape alphabet (all possible images), q_0 is the initial state ("start navigation"), and δ is the transition function:

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{R, L, S\} \tag{1.1}$$

where {R, L, S} are the head motion right, left and stationary in the context of Turing Machine, but can be redefined and expanded to the actions taken by the agent to alter the environment. Our DN-2 aims to incrementally learn these individual Turing Machines of single navigation segments. Thus our DN-2 aims to behave like a universal Turing Machine (UTM) [18, 33]. The state Q in Eq. (1.1) are reflected in the concept zones of the network in the output layer. However, the formulation above uses symbolic representation (i.e., the states and symbols are human-defined with clear boundaries among those symbolic states), which requires human hand-crafting and prior knowledge about the task.

Using emergent representation (i.e., representation that emerges from the interactions between the agent and the environment without human hand-crafting) to learn clear logic is difficult. Marvin Minsky and others stated that symbolic models are logical and neat, but connectionist models (using emergent representations) are analogical and scruffy [37]. Weng argued that DN-1, a connectionist model, learns an FA error-free while using emergent representation, thus the DN family resolves the scruffy abstraction problem and has clean logic inside. DN-2 inherits this from DN-1 [55]. Detailed correspondence between our DN-2 and a UTM is also presented in Sec. 4.4 and Sec. 4.5.

Concept 2: Hierarchical representation. The formulation in Eq. (1.1) requires the neural network to learn a complete set of tape alphabet Γ, which in the context of vision-based navigation is all possible input images. This is computationally impossible as real-world images contain distractors of numerous shapes, colors, and scales.

Hierarchical representation is helpful in this sense as it abstracts lower level details into noise invariant representations. The lower level representations are often formed within a local receptive field, limiting the number of variations of appearances within that area. A higher level representation sums up the local features and produces robust high-level feature maps for decision making. Ideally, this feature map should achieve invariance with regard to the numerous low-level distractors while saving computational resources.

Symbolic models (models that use symbolic representation. E.g., Markov models based on FA [13, 10], graphic models [3, 20] or belief nets [16, 17]) use hierarchical representation to cut off symbolic linking, as they cannot handle all possible linkings thus disregarding most of the links. This hierarchy is external and can never emerge internal connections, as they rely on outside restriction by human hand-crafting. Emergent brain-like hierarchical models for navigation are proposed in [47, 30], but the hierarchy inside their methods is rigid (static region boundaries) and one-directional (no top-down context for internal representations). The landmark concept tree for navigation in [51] is only in the concept area of the neural network, while the internal representation is still without hierarchy. In DN-2, the hierarchy is not external, but internal and emergent. We allow all possible connections to be learned automatically because DN-2 uses natural sensor and motor patterns as internal states. Each neuron has its competition zone. Thus the hierarchy inside DN-2 is fluid instead of rigid hierarchy compared to other methods.

1.1 Related Work

As a brain-inspired learning framework, DN-2 touches many aspects in the field of autonomous mental development, machine learning, and neural networks. Here we only compare DN-2 with the researches and experiments most relevant to the scope of this paper.

Comparison with incremental learning Kraft et. al. proposed an incremental learning system that builds object representations and associated object-specific grasp affordances based on exploration with no supervision[23]. Incremental learning is also investigated in error minimized extreme learning machine (EM-ELM) [9], where the true label of the training data is always provided during learning. The difference between DN-2 learning and such systems lies in the learning mode of DN-2: the agent updates its internal representation based on the current context in sensor and motor, regardless of being supervised or not. This allows the teacher to partially supervise the output layer (e.g. only supervise higher level concepts while leaving lower level concepts emergent), and so allowing concept scaffolding.

Comparison with transfer learning Transfer learning refers to the situation where the target domain is different from the source domain, and the knowledge representation learned in the source domain needs to be transferred to the target domain[39]. The experiments in this paper can be viewed as a transfer-learning problem where the source domain is where we train the agent with low-level representations and then apply the low-level representation to the source domain where we train the agent to learn navigation skills. However, the DN-2 framework by itself is more flexible than transfer-learning scenarios as the DN-2 does not differentiate between different domains. The source and target domains are implicitly defined by the teacher when teaching the network different concepts, but the learning algorithm itself requires not such separation among domains.

Comparison with SLAM SLAM (simultaneous localization and mapping) comprises the simultaneous estimation of the state of a robot equipped with on-board sensors, and the construction of a model (the map) of the environment that the sensors are perceiving [8, 5]. Because we rely on the GPS embedded onto the smart phone and the Google map API to give general guidances of direction, we do not aim to tackle the problem of localization, which is heavily investigated in many SLAM based outdoor navigation researches (e.g., [35, 36, 5]). Compared to brain inspired SLAM architectures such as RatSLAM, our DN-2 forms a hierarchical representation that are more similar to the local to global hierarchies found in visual cortex [50] than to the topological representations

found in the hippocampus [36].

Comparison with neural architectures for vision There are many neural network architectures are also inspired by visual cortex. Convolutional Neural Networks (i.e., CNNs) [25, 26, 6] gained state of the art performance on image recognition by stacking spatially invariant convolution operation with pooling operation. The learning mechanisms in these networks are based on stochastic gradient descent (SGD), which requires many iterations of training to reach convergence thus is unsuitable for incremental learning. Moreover, global tuning with stochastic gradient descent also disrupts long-term memory, preventing the network to keep old knowledge (lower-level skills) when learning the newer ones (higher-level skills). DN-2 uses competitive learning and hebbian learning instead of SGD. Finally, the hierarchies in CNNs (and also HMAX, a feedforward hierarchical neural network of alternating template matching and pooling operations to model object recognition in visual cortex [41]) are static in the sense that the network's structure is munually picked and predefined by the programmer. In DN-2 neurons in different levels of hierarchy are initialized when current match is bad, forming a fluid hierarchy based on the learning experience. In the experiment section we present a baseline model using a CNN architecture.

Other related works The problem we are address in this paper is similar to the problem addressed by the Qualitative Learner of Actions and Perception algorithm (QLAP) [38]. QLAP, however, assumes a perfect detection result from the environment by assuming "the agent has trackers for a small number of moving objects (including its own body parts) within an otherwise static environment". This means that the hierarchy in QLAP is built in the concept area, while DN-2 is building hierarchy of representation internally to handle real-world input with noises and distractors.

The Hebbian learning mechanism used in DN framework by itself resembles the learning in Self Organizing Maps (SOM) [22]. However, DN-2 differs from SOM in the following aspects: 1) SOM initializes all neurons at $t = 0$ where DN-2 releases neurons based on goodness of match. The initialization of neurons in DN-2 is more similar to the Growing Neural Gas algorithm (GNG)

[12]. However, 2) SOMs and GNG do not have hierarchy of representations inside. DN-2 uses lateral connections to form hierarchical representations. 3) SOM and GNG do not take temporal information into account. DN-2 updates recursively thus has a temporal aspect in its internal representation.

The experiments in this paper can be categorized as 'topological map based outdoor navigation' as proposed in [4], where a topological map is defined as a graph of landmark nodes associated with specific instructions (e.g. cross road, turning, or go straight ahead). In our case, the topological map is the instruction (e.g. turn left, turn right, move forward, or arrive) from the GPS system at each intersection way-point, defined by the Google map API. Unlike the researches listed in [4], where most experiments are performed on unmanned vehicles or robots with mechanical effectors, in this paper the agent's actions are broadcasted as audio instructions to the human user with the user performing the actions according to the output. The developed application is intended to help visually impaired people to walk around campus using smart phone with a pair of stereo camera and the proposed application.

Comparison with Neural Networks doing Turing Computations Siegelmann and Sontag showed that there is a finite recurrent neural network with the capability of Turing computation [42, 43]. However, their simulation in [43] is still symbolic, as the binary tape used in their experiments carries hand-crafted specific meaning.

Graves et al. proposed a neural network with read-write head with external memory in [15, 14]. However, Graves et al.'s network did not deal with emergent representation and did not use firing patterns as a stage and action. Unlike the work of Siegelmann and Sontag [42, 43], Graves et al. [15, 14] did not prove their network's equivalence to Turing Machine.

In contrast, Weng already proved that DN-1 could perform Turing computations in [56]. DN-2 can achieve the computation capabilities of DN-1 by using just type 101 neurons (neurons with connections from only X, the input area, and Z, the motor area).

Comparison with DN-1 As Weng pointed out in [56], the Developmental Network (DN) was the first general-purpose emergent FA that:

1. uses fully emergent representations,
2. allows natural sensory firing patterns,
3. allows the motor area to have subareas where each subarea represents either an abstract concept (cost, skill, means, etc.) or natural muscle actions (e.g., move forward or turn left/right)
4. learns incrementally – taking one-pair of sensory pattern and motor pattern at a time to update the network and discarding the pair immediately after
5. realizes the maximum likelihood estimate of the network, conditioned on the limited computational resources in the network and the limited learning experience in the network's "lifetime".

The DN-1 framework has already had several implementations named as Where-What Networks (WWN) which are used to recognize and localize foreground objects directly from cluttered scenes. WWN-1 and WWN-2 [21] recognizes two types of information for single foregrounds over natural backgrounds: type recognition given location information and location finding given type information. WWN-3 [31] recognizes multiple objects in natural backgrounds. WWN-4 [32] demonstrates advantages of direct inputs from the sensory and motor sources. In WWN-5 [44], object apparent scales are learned using the added scale motor concept zone. WWN-6 [54] uses synapse maintenance to form dynamic receptive fields in the hidden layers incrementally without handcrafting. WWN-7 [59] learns multiple scales for each foreground object using short time video input.

However, the computation in DN-1 relies on exact matching between the store weights and the current input pattern (from both sensor area X and the motor area Z). There is no hierarchy inside the representation of DN-1.

This paper presents DN-2, a novel neural network architecture based on DN-1. DN-2 extends DN-1 by introducing the following mechanisms into the original framework:

First, lateral connections enable hierarchies of representation. To compute the internal firing patterns $Y(t)$ DN-1 relies only on the sensor input pattern $X(t-1)$ and motor input pattern $Z(t-1)$. This is primitive in several ways: (1) such feature is concrete with no abstraction, and (2) such feature does not have temporal context from the last internal state of the network. In DN-2 the sources of input for internal area $Y(t)$ extends to the previous internal firing pattern $Y(t-1)$. This allows the network to develop a hierarchy of representations, discussed in Sec. 4.3. The learned hierarchy of representation is thus (1) more abstract and thus more invariant to the noises in the input and (2) more extensive in temporal dependency.

Second, multiple levels of neuron form a hierarchy in representation. There are seven types of Y neurons in DN-2 (shown in Fig. 1.1). DN-1 only has one type of neuron (with only bottom-up and top-down connection, a.k.a. type 101. We use three binary digits to represent whether the neuron has connection from input, internal and output areas respectively.) There is no hierarchy of representation in DN-1. In DN-2, type 100 and 001 neurons form the lowest level of representation, with their attended information integrated into higher levels of neurons (e.g., 011, 110, and 111). The lower level representation may be sensitive to local changes in the input patterns, while the higher levels neurons (with global receptive fields) use the firing pattern from lower level neurons to incorporate both global view and local finer details within the current input.

1.2 Optimality of DN-2

The most important theorem about DN-2 in this paper can be summarized in the following sentence:

Under the constraint of skull-closed incremental learning and the pre-defined network hyper-parameters, DN-2 optimizes internal parameters to generate maximum likelihood firing patterns in network areas, conditioned on its sensory and motor experience up to the network's last update.

The theorem is formally introduced in Sec. 3.5, separated into Lemma 1 and Theorem 1, and

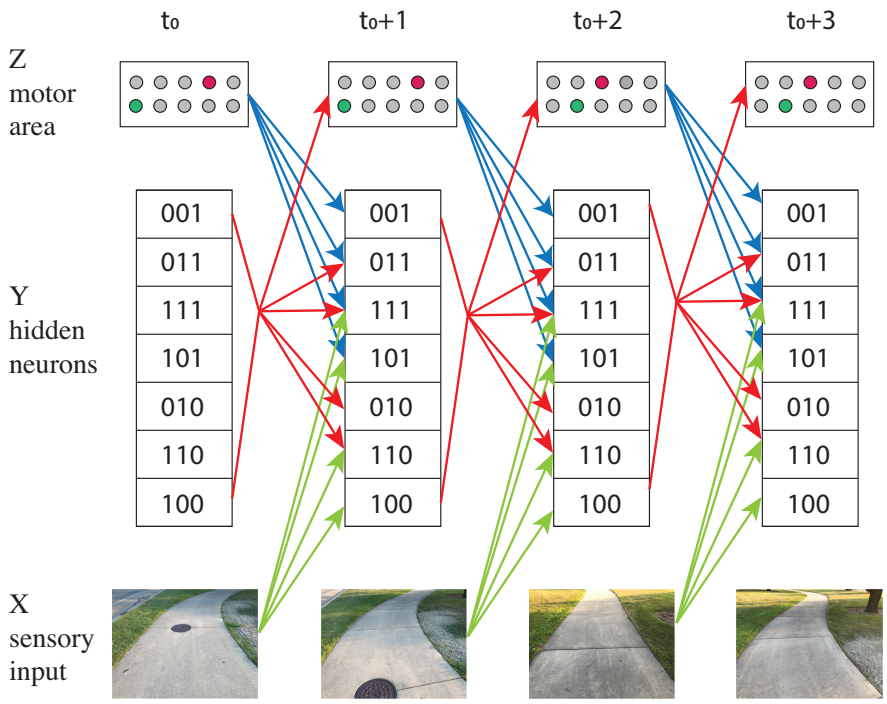

Figure 1.1: Summary of the DN-2 framework. DN-2 has seven types of internal neurons with different types of connections. Blue lines: top-down connection from Z. Red lines: lateral connection from Y. Green lines: bottom-up connection from X. In this paper, we show the optimality in the learning procedure of DN-2 by proving Lemma 1 and Theorem 1, shown in this figure. The verbal version of Theorem 1 can be found in the italic paragraph in the Introduction. Details about the theorem is presented in Sec. 3.5.

also illustrated in Fig. 1.1. To prove this theorem, we formulated the learning problem under Maximum Likelihood Estimation, which is attached in the Appendix.

DN-2 mitigates some of the most notorious learning problems:

Brittle symbolic modules There are no symbolic modules in DN-2. The environment has too many rules and variances to hand-craft clean, symbolic representations without missing an aspect. Emergent representations in DN and DN-2 are not restricted by handcrafted design, with new representations initialized to learn the specific input if current best match is insufficient. DN-2 is thus immune from the brittleness of the symbolic representation (if all neurons are already initialized then the network performs maximum likelihood estimation as stated by the theorem).

Curse of dimensionality problem DN-2 is less likely to suffer from curse of dimensionality. Curse of dimensionality is caused by using input of high-dimensionality during learning [19]. In the case of DN-2, this curse of dimensionality is dealt with local-to-global hierarchy developed incrementally according to the network's learning experience. All receptive fields in DN-2 develop invariance. Such invariance is developed using attention context from Z area. The group concept neuron will only take the relevant components as input and disregard irrelevant components because when the group concept neuron fires, relevant component neurons stably fire but irrelevant component neurons do not consistently fire. This problem is discussed again in Sec. 3.6.1.

Over-fitting problem DN-2 is less likely to over-fit. The over-fitting problem refers to the phenomenon where the learning agent remembers only the training data but generalizes poorly over new data, often due to the large number of parameters and the small number of training samples. DN is less likely to over-fit because DN starts with no internal neurons and gradually initializes new neurons only when the trained neurons cannot recognize the current input well. When the input is one frame in the receptive field for each neuron, then the formed representation is optimal. Later input frames for each neuron are taken into account through recursive average, which is optimal based on maximum likelihood estimation. Neuron initialization is introduced in detail in Sec. 2.2.4. This problem is discussed again in Sec. 3.6.2.

Local minima problem DN-2 is less likely to be stuck in local minima. Typical learning agents (e.g. agents using error back-propagation) aim to minimize a task-specific loss function, and would thus often get stuck at local minima during optimization. As we state in Theorem 1 (and also reiterate in the introduction above), DN-2 optimizes (in the sense of Maximum Likelihood Estimation) over the entire learning experience, conditioned on incremental learning and limited resources. The side-effect is that DN-2's performance is now dependent on the external teaching schedule (i.e. learning experience). This is discussed again in Sec. 3.6.3.

Please note we do not claim DN-2 completely solves these learning issues, but by conceptually investigating the learning procedures and properties of DN-2 we believe these issues are to a large

extent mitigated.

1.3 Discussion: Sensorimotor Recursive Experiments vs. Batched Data Classification Experiments

Should we compare DN's performance on standard image classification datasets (e.g. NORB [28], MNIST [27], CIFAR-10 [24] and ImageNet [7]) with other image classification networks?

Standard image classification datasets are widely used to benchmark performance for different neural network architectures. These datasets usually contain large number of labeled images or video sequences labeled with corresponding classification information. Batched data experiments are not sensorimotor recursive, meaning that the motor output of the agent does not have impact on the training or testing data.

The focus of this paper, however, is to build a neural network that uses emergent representation and incrementally learns to behave like a Turing Machine with a read-write head. This requires the agent to perform in a sensorimotor recursive environment. DN can of course handle image classification tasks with proper teaching: Wagel and Weng used a variation of Where-What Network for CIFAR-10 classification task and achieved an error rate of 19.8% . The same network architecture achieved an error rate of 5.0% on Norb dataset [53]. However, a learning agent that is task non-specific, general purpose and learns incrementally should not be unfairly compared to agents optimized with batch data as these agents are task specific. It is like human brains cannot be compared to an electronic calculator at the task of long number multiplication, as the calculator is specifically designed for this task and a human brain as a general purpose would make more mistakes and also be slower at long number multiplication.

1.4 Example: Shadows edges and road edges

In this example, we outline how DN-2 uses the learned where-what representation to form robust representation for navigation. The following discussion corresponds to the steps illustrated in Fig. 1.2.

In the context of navigation, the agent is required to pay attention to the left road edge and right

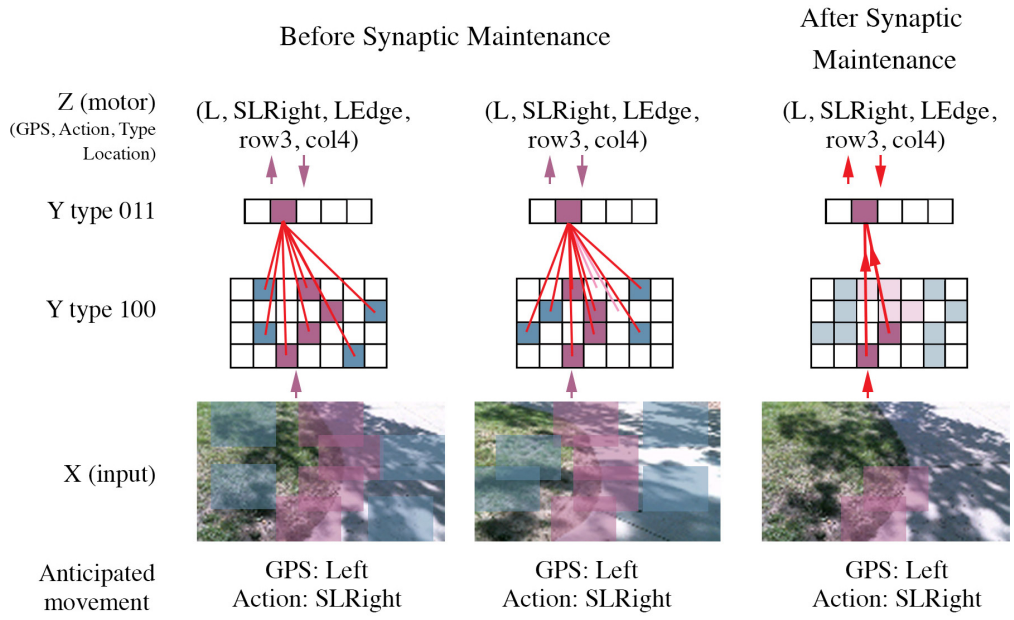

Figure 1.2: Demonstration of how low-level where-what representation facilitates learning complex navigation rules. Low-level 100 Y neurons look at local input patterns. Some neurons learn to recognize road edges (red square) at specific locations, and others learn to recognize shadow edges (blue square). Before synaptic maintenance, the high level Y neuron of type 011 learns different firing patterns from low-level 100 Y neurons. After synapse maintenance, the stable connections (on constant road edges) are kept while the unstable connections (on varying shadow edges) are cut from the high-level Y neurons, forming a shadow-invariant representation to learn the navigation rule "correct facing direction when left road edge is in the middle of the input image".

road edge. When making a left turn, the agent must adjust its facing direction by turning slightly right when the left road edge is recognized in the center of the image (unless there is an obstacle at the right-hand side). However, the recognition of road edges is often disrupted by shadows, which are usually monotone and of various shapes.

Bottom-up feature hierarchy Low level neurons with local receptive fields would be paying attention to local road edges (stable features) and shadow edges (unstable features). During learning a higher level neuron would group these low level features together. Synapse maintenance cuts away the unstable features, focusing the network's attention on important stable features. See Fig. 1.2 (a).

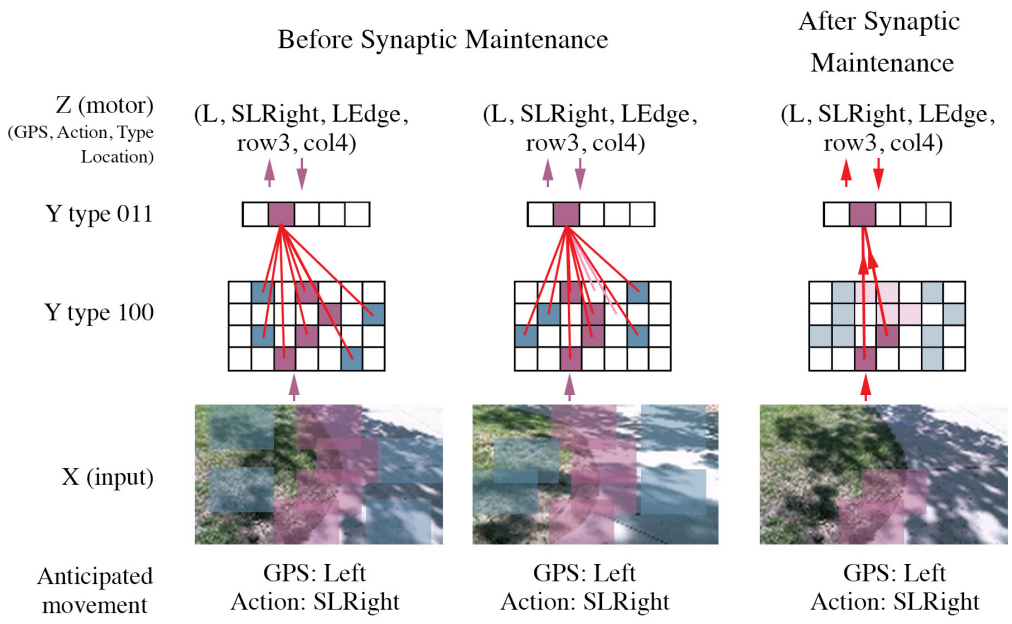

Figure 1.3: Demonstration of how top-down navigation context affects internal firing. In this example, all type 100 Y neurons are learning local features (road edges and shadow edges). But given different GPS navigation context (red letter in Z) motor, different high-level neurons (type 011) would fire. The attention of the network would thus be shifted toward the most relevant feature based on the current navigation context.

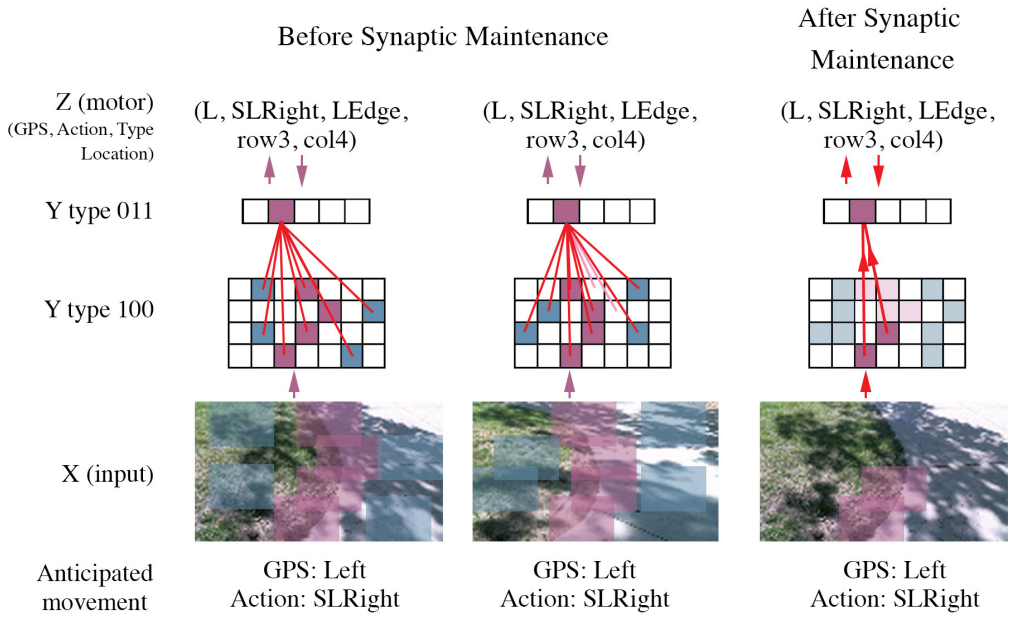

Figure 1.4: Demonstration of bottom-up and top-down effects in internal firing combined. The network shifts its attention according to the navigation context (red letter. The demonstrations in this Figure are synthetic examples for the purpose of demonstration.)

Top-down context affects firing pattern With a different firing context, different higher-level neurons would fire. These neurons are linked with different groups of lower-level neurons, thus focusing on different part of the same input image. See Fig. 1.2 (b).

Combined effect Combining these effects together we have a dynamic attention system that is not only focusing on important bottom up features but also incorporating top-down navigation contexts from the previous computation. See Fig. 1.2 (c).

This is just one example under the context of real-time navigation. We want our DN-2 to automatically form these rules that are too numerous to hand-craft. The detailed DN-2 mechanism to achieve this is introduced in the following section.

CHAPTER 2

DEVELOPMENTAL NETWORK 2

2.1 Main Procedure

X is the sensory zone with input $\mathbf{x}(t)$ represented as a vector (e.g. input image reshaped to a vector). Y is the hidden layer(s) inside the network with all neurons' final response values represented as a vector $\mathbf{y}(t)$. Z is the motor area with multiple concept zones. Concatenate all responses in Z and we have $\mathbf{z}(t)$.

The dimension and representation of X and Z areas are based on the sensors and effectors specified by the programmer (or DNA). Y is skull-closed, not directly accessible from the outside.

In the scope of this paper, the teacher/programmer is required to design the teaching schedule and the concept zones (motor areas) of the learning agent. Motor areas need to be designed to accomplish corresponding tasks (e.g., if there is no action motor then it would be impossible for the agent to do navigation, no matter how 'smart' the agent is). However, the internal representation of the agent remains emergent and skull-closed.

1. For Y area, all Y neurons enter initialization stage (not active) with random weights and zero firing ages. Z neurons initialize their adaptive weights to zero weights. All Z neurons are active at birth time. N_y is the adaptive part of the hidden layer. N_z is the adaptive part of the motor area.

2. At time $t = 0$, supervise initial state $\mathbf{z}$. Input the first sensory input $\mathbf{x}$. Internal states in Y are all zeros as no neurons are firing. Thus $\mathbf{y}(0) = \mathbf{0}$.

3. At time $t = 1, ...$, repeat the following steps:

   a) Compute Y area's response vector for all active Y neurons in parallel. Also update the

Table 2.1: Y neuron types and perfect match value θ

Type	X (Sensor)	Y (Hidden)	Z (Motor)	θ
001	No	No	Yes	1
010	No	Yes	No	1
011	No	Yes	Yes	2
100	Yes	No	No	1
101	Yes	No	Yes	2
110	Yes	Yes	No	2
111	Yes	Yes	Yes	3

adaptive part of the hidden layer with the new N_y'.

$$(\mathbf{y}(t), N_y') = f_y(\mathbf{p}_y, N_y) \tag{2.1}$$

where $\mathbf{p}_y$ denotes the tuple of $(\mathbf{x}(t-1), \mathbf{y}(t-1), \mathbf{z}(t-1))$, and f_y is the response computation, response competition, and learning function defined in Sec.2.2.2 and Sec. 2.2.3. Also, one or several Y neurons in initialization stage enters learning stage if all current active Y neurons can not match the input vector well, defined in Sec.2.2.4.

b) Compute Z area's response vector for all Z neurons in parallel. Update the adaptive part of the motor layer with the new N_z'.

$$(\mathbf{z}(t), N_z') = f_z(\mathbf{p}_z, N_z) \tag{2.2}$$

where $\mathbf{p}_z = \mathbf{y}(t-1)$.

2.2 Details of the algorithm

2.2.1 Patterning

For human brains, the early wirings between neurons are determined before birth (by genes or DNA) [46, 11]. Inspired by this, we create several types of early Y neurons with different connection regions to simulate early connections. The seven Y neuron types are shown in Table 2.1.

Each type of neuron has a different growth rate controlled by $\alpha(t)$ in Eq. (2.6). Top-k competitions, described in Sec. 2.2.2, are performed among each type of neurons.

2.2.2 Response Computation and Competition

The area functions f_y and f_z are based on the theory of Lobe Component Analysis (LCA) [58].

Each active Y neuron calculates its preresponse. The bottom-up part of the preresponse in neurons with bottom-up connection is calculated as follows:

$$r'_{b,i} = \langle \frac{\mathbf{x}(t-1)}{\|\mathbf{x}(t-1)\|}, \frac{\mathbf{w}_{b,i}}{\|\mathbf{w}_{b,i}\|} \rangle \tag{2.3}$$

where $\mathbf{w}_{b,i}$ is the bottom-up weight of that neuron. The brackets indicate inner product of two unit vectors. This equation calculates the cosine similarity between the stored pattern (i.e. $\mathbf{w}_{b,i}$) and the input vector.

Top-down response $r'_{t,i}$ and lateral response $r'_{l,i}$ are calculated in a similar way.

For neuron i, its pre-response r'_i is then calculated as:

$$r'_i = r'_{b,i} + r'_{t,i} + r'_{l,i}. \tag{2.4}$$

If the neuron does not have lateral connection (or bottom-up connection, or top-down connection), then $r'_{l,i} = 0$ (or $r'_{b,i} = 0$, or $r'_{t,i} = 0$).

To simulate inhibitions within Y, we define dynamic competition set for each neuron. A neuron can only fire when among top-k winners in its dynamic competition. In this paper, $k = 1$.

In this paper, each neuron's dynamic competition set is defined as $Y_i = \{j | \text{type}_j = \text{type}_i, (\text{rf}_j \cap \text{rf}_i \neq \emptyset) || (\text{type}_i \neq 100)\}$, where rf_i is initialized according to Sec. 2.2.6.

If neuron i is the neuron with the highest response within its competition zone, it fires with $y_i = 1$. Otherwise, $y_i = 0$. After all Y neurons compute their responses in parallel, we obtain $\mathbf{y}'$ (the new Y response).

The Z area computes its response $\mathbf{z}'$ similarly. In this paper, the Z neuron's competition zone is within the neuron's concept zone (i.e. all neurons within the same concept zone compete with each other).

2.2.3 Hebbian Learning of excitation

If a neuron wins in the multistep lateral competition described above (meaning its firing rate is greater than zero), its bottom-up weight (top-down weight, or lateral weight, depending on the type of neuron) would update using the following Hebbian learning rule:

$$\mathbf{w}_i \leftarrow \beta_1 \mathbf{w}_i + \beta_2 r_i \mathbf{x}(t-1)$$

$\beta 1$ and $\beta 2$ determine retention and learning rate of the neuron, respectively:

$$\beta_1 = \frac{m_i - 1}{m_i}, \beta_2 = \frac{1}{m_i} \tag{2.5}$$

with $\beta_1 + \beta_2 \equiv 1$, m_i is the neuron's firing age, i.e. $m_i = 1$ when the neuron first enters learning stage, and increases by one every time the neuron wins competition.

2.2.4 Initialization

The random GDN initialization method is used in DN2. If the top-1 winner in Y has a pre-response lower than almost perfect match $m(t)$, take a neuron in initialization stage to fire with its response equal almost perfect match. The almost perfect match $m(t)$ is defined as follows:

$$m(t) = \alpha(t)(\theta - \epsilon) \tag{2.6}$$

where ϵ is the machine zero, θ is the perfect response value for each type defined in Table 2.1 and $\alpha(t)$ is a handcrafted table to control the speed of neuronal growth. We discuss the design of $\alpha(t)$ in Sec. 2.2.5.

This new neuron would thus suppress other neurons from firing, as it's guaranteed to win in the competition. Because its firing age is 1 at this time, its learning rate would be 1. According to LCA it will learn the current input perfectly.

2.2.5 Controlling $\alpha(t)$ for multistage learning

By controlling the growth rate $\alpha(t)$ for each type of neuron, the DN releases the seven types of neurons at different speeds. The growth speed is fastest when $\alpha(t) = 1$, at which time $m(t) = \theta - \epsilon$

(the almost perfect response for that type of neuron) according to Eq.(2.6). A small variation in the input would make the highest response among Y neurons less than the perfect value, thus initializing a new neuron. When $\alpha(t) = 0$, no new neurons are initialized.

In the experiments we presented in in this paper, multistage learning is realized by adjusting the teaching schedule according to the grow rate table of the network. From $t = 0$ to 1 hour the growth rate of type 100 neurons is set to 0.85, while the growth rate of the other types is set to 0. Thus we train the network to recognize basic visual information like edges, corners, etc. After $t = 1$ hour the growth rate of type 111 neurons is 0.8, and the growth rates of other types are 0, then we train the network with higher level concepts with more complicated actions.

2.2.6 Receptive fields initialization

In the scope of this paper, the receptive fields for neurons of type 001, 010, 011, 101, 110, and 111 are initialized as global receptive fields. Any neuron of type 011 would have global connections to other types of Y neurons and global connections to the Z area.

Type 100 neurons, which are used in real-world experiments to learn basic local visual features like edges, corners, color information, etc., have local receptive fields in the bottom-up input. Their functionality is the same as the locally-connected layer with unshared weights in the context of Convolutional Neural Networks, with a stride of 1, a predefined filter size, and a predefined filter number.

According to the definition of competition zone (see Sec. 2.2.2), type 100 neurons perform local top-1 competition with each other (only compete with neurons of the same type and also have overlapping receptive fields). Thus during network update, this type of neurons generates a dispersed feature map.

Z neurons in this paper all start with global receptive field.

2.2.7 Synaptic maintenance

Synaptic maintenance is conducted by each neuron after a certain number of firing. The goal of synaptic maintenance is to dynamically fine-tune the receptive fields.

A neuron i's synaptic weight vector and input are denoted as $\mathbf{v}_i = (v_{i1}, v_{i2}, ...v_{id})$ and $\mathbf{p}_i = (p_{i1}, p_{i2}, ..., p_{id})$ respectively. v_{ij}, j-th component of $\mathbf{v}_i$, is the weight value between neuron j and neuron i. p_{ij}, j-th component of $\mathbf{p}_i$, is the input from neuron j to $i(j = 1, 2, ..., d)$. We use amnesic average of l_1-norm deviation of match between v_{ij} and p_{ij} to measure expected uncertainty for each synapse. Suppose that $\sigma_{ij}(n_i)$ is the deviation at neuron's firing age n_i. The expression is as follows:

$$\sigma_{ij}(n_i) = \begin{cases} \epsilon & \text{if } \Delta n \leq n_0 \\ \beta_1(\Delta n_{ij})\sigma_{ij}(n_i - 1) & \\ +\beta_2(\Delta n_{ij})|v_{ij} - p_{ij}| & \text{otherwise} \end{cases} \tag{2.7}$$

$$\sigma_{ij}(n_i) = \beta_1(\Delta n_{ij})\sigma_{ij}(n_{i-1}) + \beta_2(\Delta n_{ij})|x_{ij} - w_{ij}| \tag{2.8}$$

where ϵ is the machine zeros, $\Delta n_{ij} = n_i - n_{ij}$ is the number of firings this synapse advanced, $\beta_2(\Delta n_{ij})$ is the learning rate dependent on the firing age (counts) of this synapse, and $\beta_1(\Delta n_{ij})$ is the retention rate, $\beta_1(\Delta n_{ij}) + \beta_2(\Delta n_{ij}) \equiv 1$. These are similar to the β_1 and β_2 defined in Eq. (2.5), while the difference is using synapse age instead of neuron age. n_0 is the waiting latency (e.g. $n_0 = 20$). The expected synaptic deviation among all the synapses of a neuron is defined by:

$$\bar{\sigma}_i(n_i) = \frac{1}{d}\sum_{j=1}^{d} \sigma_{ij}(n_i). \tag{2.9}$$

We define the relative ratio:

$$r_{ij}(n_i) = \frac{\sigma_{ij}(n_i)}{\bar{\sigma}_i(n_i)}. \tag{2.10}$$

and introduce a smooth synaptogenic factor $f(r)$ defined as:

$$f(r) = \begin{cases} 1 & \text{if } r < \beta_s(t) \\ \frac{\beta_b(t) - r}{\beta_b(t) - \beta_s(t)} & \text{if } \beta_s(t) \leq r \leq \beta_b(t) \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

where $\beta_s(t)$ and $\beta_b(t)$ are parameters (can be updated with step through time) to control the number of synapse with active connection ($f(r) > 0$). In this paper we set $\beta_s(t) = 0.8$, and $\beta_b = 1.5$. Meaning that the synapses deviating more than 1.5 times the average of the deviation would be cut, and the synapses deviating less than 0.8 times the average of the deviation would be intact.

Then trim the weight vector $\mathbf{v}_i = (v_{i1}, v_{i2}, ..., v_{id})$ to be

$$v_{ij} \leftarrow f(r_{ij})v_{ij} \tag{2.12}$$

$j = 1, 2, ..., d$. Similarly, trim the input vector $\mathbf{p}_i = (p_{i1}, p_{i2}, ..., p_{id})$.

In short, synaptic maintenance cuts the connections where the weight's deviation is too large (i.e. more than β_b times the average deviation among the neuron's connections). Only the stable connections are kept after synaptic maintenance.

CHAPTER 3

MAXIMUM LIKELIHOOD ESTIMATION IN DN-2

3.1 Review of the three theorems in DN-1

In order the understand the property of DN-2, we need to look back at the three important properties of DN-1 proved in [56]:

1. With enough neurons, DN-1 incrementally learns any FA error-free by observing the transitions in the target FA only once with two updates of the network, supervised by inputs from X and Z.

2. When frozen, DN-1 generates responses in Z with maximum likelihood estimation, conditioned on the last state of the network.

3. With limited resources, DN "thinks" (i.e., learns and generalizes) recursively and optimally in the sense of maximum likelihood.

The proof in [56] is under two assumptions, which are no longer satisfied in the context of DN-2:

1. Only type 101 Y neurons. DN-1 only uses Y neurons with bottom-up and top-down connections. However, DN-2 now has seven types of neurons, as shown in Fig. 1.1.

2. Global top-1 competition among Y neurons. Only one Y neuron in DN-1 is firing at any given time, thus learning the exact X pattern with a specific Z pattern associated to this Y neuron. However, DN-2 has several Y neurons firing at any given time.

Thus the assumptions for DN-1 optimality is no longer available, requiring a new way to prove the optimality of learning in DN-2.

3.2 Definition of DN-2

A DN-2 network at time t can be defined as:

$$N(t) \quad = \quad \{\theta(t), D(t), \Gamma\} \tag{3.1}$$

Where Γ denotes the hand-picked hyper parameters for the network, $D(t)$ denotes the long-term statistics inside the network. Parameters in $D(t)$ are not used for optimization. $\theta(t)$ contains the weights that are actually being optimized at time t.

$$\begin{aligned} \Gamma &= \{G, k, l_{\text{in}}, l_{\text{out}}, n_y, n_z\} \\ D(t) &= \{\{g|g \in Y, Z\}, \{\bar{g}(t)|g \in Y, Z\}, a(t), L(t)\} \\ \theta(t) &= \{W(t)\} \end{aligned}$$

G is the growth rate table. k is the top-k parameter for competition. l_{in} is the limit of connections to a specific neuron. l_{out} is the limit of connections from a neuron to other neurons. n_y is the maximum number of neurons in Y zone. n_z is the maximum number of neurons in Z zone. g is the set of neurons in Y and Z area. $\bar{g}(t)$ is the inhibition field of the specific neuron at time t. $L(t) = \{L(g,t)|g \in Y, Z\}$ is the set of receptive field (line subspace) for each neuron at time t. $a(t) = \{a(g,t)|g \in Y, Z\}$ is the set of connection ages for each neuron at time t. $W(t) = \{W(g,t)|g \in Y, Z\}$ is the of weights for each neuron at time t.

3.3 Conditions on DN-2 learning

During learning, DN-2 is constrained by the following conditions, denoted as $\mathbf{C} = \{C_i, i = 1, 2, 3\}$

C_1: Incremental learning. This means the network never stores training data, learns on-line, and has no pre-programmed priors about the task at the birth time.

C_2: Skull-closed learning. Human teachers can only supervise the motor and input areas of the network once it begins learning.

C_3: Limited resources. The hand-picked hyper-parameter Γ (which limits the number of neurons used by the network) remains unchanged during learning.

3.4 Lemma 1: DN-2 optimizes its weight under maximum likelihood for each update, conditioned on C.

Lemma 1 *Define* $\mathbf{p}(t+1) = \{\mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t)\}$ *(*$\mathbf{p}$ *is thus a set of all responses in the three zones). At time* $t+1$*, DN incrementally adapts its parameter* θ *as the Maximum Likelihood (ML) estimator for input in* Y *and* Z*, based on its learning experience with limited resources* Γ *:*

$$\theta(t+1) = \max_{\theta} f\{\mathbf{p}(t+1)|N(t), \boldsymbol{C}\}, t \geq 0 \tag{3.2}$$

The probability density $f\{\mathbf{p}(t+1)|N(t)\}$ *is the probability density of the new observation* $\mathbf{p}(t+1)$*, conditioned on the last status of the network* $N(t)$*, based on the network's learning experience.*

Proof for Lemma 1 is attached in the appendix.

Lemma 1 states each time DN-2 is providing the best estimate. Inside the 'skull' neurons' weights are updated optimally based on $\mathbf{C}$. But the external environment providing the optimal teaching schedule is not guaranteed.

The following theorem is proposed by using Lemma 1 recursively:

3.5 Theorem: DN-2 learns optimally under maximum likelihood from its incremental learning experience, conditioned on C.

Theorem 1 *Define* $\mathrm{X}_{t_0}^{t} = \{\mathbf{x}(t_0), \mathbf{x}(t_0+1), ...\mathbf{x}(t-1)\}, \mathrm{Z}_{0}^{t_0} = \{\mathbf{z}(t_0), \mathbf{z}(t_0+1), ...\mathbf{z}(t-1)\}$*. At time* $t+1$ *DN-2 adapts its parameter as the ML estimator for the current* $\mathbf{p}(t+1) = \{\mathbf{x}(t), \mathbf{z}(t)\}$*, conditioned on the sensory experience* X_t *and* Z_t *with limited resources* Γ*:*

$$\theta(t+1) = \max_{\theta} f'\{\mathbf{p}(t+1)|\mathrm{X}_0^t, \mathrm{Z}_0^t, \boldsymbol{C}\}, t \geq 0 \tag{3.3}$$

The probability density $f'\{\mathbf{p}(t+1)|\mathrm{X}_0^t, \mathrm{Z}_0^t, \Gamma\}$ *is the probability density of the new observation* $\mathbf{p}(t+1)$*, conditioned on the entire sensory experience and the pre-defined hyper-parameters.*

Proof for theorem 1 is attached in the Appendix.

This theorem is important as it shows that a DN-2 equipped agent behaves in a maximal likelihood fashion while learning incrementally and immediately without the need to store batch data or iterate through training data for multiple times.

3.6 Discussion

Having introduced the algorithm in detail, we can discuss more about the properties listed in the Introduction in depth.

3.6.1 On curse of dimensionality

As shown in Sec. 2.2.4, DN is less likely to suffer from curse of dimensionality for two reasons: 1) local receptive fields of type 100 neurons is of lower dimension even though the input maybe of higher dimension, and 2) synapse maintenance cuts away unstable connections, therefore keeping the receptive field focusing on important features.

3.6.2 On over-fitting

DN is less likely to over-fit because DN starts with no internal neurons and gradually initializes new neurons by comparing the current best match with the almost perfect match $m(t)$. If large numbers of training samples cluster around similar samples (e.g., training navigation along the same route repeatedly), best match value would constantly be high, and so DN will simply retrain several of the neurons representing these samples instead of tuning the network globally to fit these training samples, avoiding over-fitting the entire network on these samples. Uncommon but important samples will not be ignored by DN-2 as their goodness of match would be low and new neurons would be assigned to remember these samples. In summary, the model's capacity will be determined by the diversity of training.

3.6.3 On local minima

DN is task non-specific, not needing to design a handcrafted loss function, and learning incrementally. In DN, only the winner neurons fire and update their weights with the Hebbian learning rule. Such an incremental computation of local average of many vectors does not suffer from the well-known local-minima problem since it converts a highly nonlinear global optimization problem into a composition of many local linear problems (the incremental update of local weight is a local linear problem), which does not have local minima. Unfortunately, there is still a local minima problem in the external teaching. For example, if the teacher teaches a complex task first and then a simple task, the learner does not have the skills from the simple task to learn the complex task. But this "local minima" problem is outside the network. The network itself is still internally optimal.

CHAPTER 4

EMERGENT TURING MACHINE AND HIERARCHICAL REPRESENTATION IN DN2

4.1 The Challenges

In navigation, skills are the navigation movements learned in different sections of the environment. For example, one of the skills can be "turn slightly left when seeing an obstacle". During real-time training, the agent learns different skills during several training sessions, each with a different context (e.g., in a simulation there was a concept zone with individual neurons corresponding to each skill; in the real-world environment the starting and ending point would be different for each training session.). Individual navigation skills can be viewed as separated transition entries inside the transition table of the Finite Automata that navigates in the desired environment. The challenges are:

1. Going beyond static hierarchy. In other hierarchical neural networks (e.g. Convolutional Neural Nets) the hierarchies are static with pre-defined layers and connection patterns. DN-1 has several areas simulating the visual pathways inside human brain (e.g. the PP, IT and L4 areas in [31]). Static hierarchy is rigid and non-developmental. Hand-crafting region boundaries before training results in suboptimal resource distribution. In DN-2 we have emergent hierarchy which means that we have dynamic boundaries and dynamic number of regions, controlled by the learning experience and growth rate of the network. DN-2 initializes new neurons when needed, forming a fluid hierarchy among regions.

2. Emergent Turing Machine (ETM) from the fluid hierarchy. Our network uses emergent representation to function as a Turing Machine. Based on the lower-level concepts (e.g. where and what), higher-level concepts (e.g. skills and actions) are invariant to the changes in input image (e.g. appearance variances and shadows) as long as the lower-level concepts abstract correctly. The firing pattern in Z is thus supported by the fluid hierarchy of internal

Table 4.1: Multiple types of neurons learning FA transition

Zone \ Time		t	$t+0.33$	$t+0.66$	$t+1$
Z (Motor)		$\mathbf{z}(t)$	$\mathbf{z}(t)$	$\mathbf{z}(t)$	$\mathbf{z}(t+1)$
Y	011	$*$	$*$	$y_{011}(t) = \{-, y_{100}(t), \mathbf{z}(t)\}$	$y_{011}(t)$
	100	$*$	$y_{100}(t) = \{\mathbf{x}(t), -, -\}$	$y_{100}(t)$	$y_{100}(t)$
X (Input)		$\mathbf{x}(t)$	$\mathbf{x}(t)$	$\mathbf{x}(t)$	$*$

representation so that invariance emerges from multi-stage learning.

4.2 Emergent FA learning in DN-2

In order the understand the property of DN-2, we need to look back at the three important properties of DN-1 proved in [56]:

1. With enough neurons, DN-1 learns any FA incrementally error-free by observing the transitions in the target FA for only once with two updates of the network, with supervision in X and Z.

2. When frozen, DN-1 generates responses in Z with maximum likelihood estimation, conditioned on the last state of the network.

3. Under limited resources, DN "thinks" (i.e., learns and generalizes) recursively and optimally in the sense of maximum likelihood.

The proof of these properties relies on two steps: 1) exact matching between the current input $(\mathbf{x}(t), \mathbf{z}(t))$ and the stored patterns (top-down and bottom-up weights) in the firing Y neuron, and 2) the firing Y neuron corresponds to an output state $\mathbf{z}(t+1)$. With enough resources, a single Y neuron corresponds to a single entry in the FA's transition table $\{\mathbf{x}(t), \mathbf{z}(t)\} \rightarrow \mathbf{z}(t+1)$. With limited resources, the Y neurons are doing tessellation in the $\{X, Z\}$ space, and optimality (in the sense of maximum likelihood) comes from the tessellation.

DN-2 extends DN-1 by introducing lateral connections among neurons and multiple types of neurons. In the above proof of DN-1, we observe that the 1-to-1 correspondence between Y

neuron and the transition state can be relaxed, we only need to guarantee that each transition state is uniquely represented by the firing Y neuron that are connected to Z area. The output in Z can be connected to this firing Y neuron in the same way as in DN-1. There are multiple ways to represent $(\mathbf{x}(t), \mathbf{z}(t))$ uniquely in the Y area in DN-2, with the help of multiple types of Y neurons:

1. Type 101 neurons. With only type 101 neurons, DN-2 is down-graded to DN-1, and DN-1's properties still hold in this situation.

2. Type 100 + type 011 neurons. The network needs to update twice to fire a type 011 neuron that corresponds to this transition, as illustrated in Table 4.1. Assuming top-1 competition in different types of neurons (i.e. only one neuron firing) and the network can always initialize new neurons when matching is not perfect, then the higher-level 011 neuron would always be able to learn the transition after two updates.

3. Type 110 + type 001 neurons. Similar argument as the type 100 + type 011 situation.

4. Type 111 neurons. With only type 111 neurons, DN-2 is performing tessellation in the $\{X, Y, Z\}$ space. The 111 neurons are memorizing transition $\{\mathbf{x}(t), \mathbf{y}(t), \mathbf{z}(t)\} \rightarrow \mathbf{z}(t+1)$, which is a superset of the transitions in the FA defined in DN-1.

Thus, the error-free learning of FA in DN-1 still holds with multiple types of neurons DN-2. The optimality in the sense of maximum-likelihood under the condition of limited resources is a natural result of tessellation, thus can be adapted from the proof in [56] with little modification.

4.3 Hierarchical representation in DN-2

We will use type 100 neurons (lower level representation, local receptive field) and type 111 neurons (higher level representation, global receptive field) as an example. Other types follow similar discussion. We present a simplified case of learning specific groups of features $\mathbf{F} = \{f_i | i = 1, 2, ..., m\}$ across all possible locations $\mathbf{L} = \{L_j | j = 1, 2, ..., l\}$ inside the input. Z motor thus has a location concept and a type concept.

During stage 1 the network is developing representation in type 100 neurons (no other types of neurons have been initialized, controlled by $\alpha(t)$), with each neuron learning a specific feature f_i at location l_j. Depending on the number of lower-level features presented in the input image, one or several low-level neurons learning specific features at specific locations $\{f_i, l_j\}$ would be firing at any given time.

During stage 2 ($t = t_2$, where t_2 is the starting time of stage 2), $\alpha(t)$ for type 100 neurons decreases, meaning that few new lower-level neurons are initialized. Also $\alpha(t)$ for type 111 neurons increases, meaning that the network is now growing higher-level neurons. As a specific group of f_i is presented, one or more 100 neurons would fire, forming a lower-level firing pattern $\mathbf{y}^{100}(t)$. At time $t + 1$, a higher level type 011 neuron y^{111} would fire and learn this lower level firing pattern, thus forming a hierarchy of representation, denoted as $\mathbf{y}^{100}(t) \rightarrow \mathbf{y}^{111}(t + 1)$.

Multi-stage learning allows coarse-to-fine integration of representation. DN-1 has only global matching of representation using type 101 neurons. Under limited resources, each neuron in DN-1 are averaging over multiple input globally where detailed local features are blurred and neglected. Synaptic maintenance resolves this issue to some extent by shaping the receptive field gradually but this process requires extensive training. In DN-2, finer, detailed representation from local receptive fields of type 100 neurons supports firing in the coarser, higher level firing among type 111 neurons.

4.4 Universal Turing Machine and Developmental Network

4.4.1 Definition of UTM

A Universal Turing machine (UTM) simulates the behavior of any TM, given the TM and the input is encoded onto the input tape of the UTM. Formally, a UTM receives an input string in the form of $e(T)e(x)$ on the input tape, where e is the encoding function, T is the targeted TM and x is the input. It simulates the computation of T on data x, and output $e(z)$, where z is the output of T on data x. The UTM does not really know the meaning of z [33].

4.4.2 Differences between UTM and DN-2

The encoding function under the UTM framework is hand-crafted. There are many possible ways to build such an encoding function, which is designed to represent the symbolic transition table. DN-2 does not have this encoding function because it does not use symbolic representation. DN-2 uses natural input as patterns in X. It also uses emergent state as patterns in Z. The actual encoding is the transformation from the $\{X, Z\}$ patterns to the neuronal weights in the network. This transformation is not hand-crafted encoding, but rather the result of competition based on the biologically inspired mechanisms of DN such as Hebbian Learning in Lobe Component Analysis and dynamic inhibition regions.

For a UTM, the input tape contains both the TM part $e(T)$ and input part $e(x)$. In DN-2, there is no fixed or static division between rules T and data x. Earlier sensorimotor experiences from X and Z tend to be considered as data x for the agent to recognize and extract rules. Later sensorimotor experiences enable the learning agent to get rules very quickly with very few examples because the rules are in abstract forms in such experiences. In other words, X and Z experience contains both the instruction and data.

To summarize in computational terms, UTM performs using an encoding function. It searches current input $e(x)$ in $e(T)$, and produces corresponding state $e(q)$ and movement $e(z)$. DN performs using emergent patterns in X, Y and Z. It searches the entire learning experience embedded in all weights of the network. After competition, the corresponding states and movement emerge in Z.

4.5 Autonomous navigation needs Emergent Turing Machine

Here we apply our DN-2 theory to a real-world scenario: autonomous navigation. Autonomous navigation requires general-purpose learning like DN-2 instead of hand-craft rules and feature detectors for the following two reasons:

1. Dynamic environment. In a real-world setting, we cannot anticipate the kind of landmarks, concepts, and context needed for an unknown driving environment.

2. Complex traffic rules. The sheer number of rules for a navigation FA is too large to enumerate. These rules also interact with each other and the interaction is impossible to hand-craft.

In DN-2, the interactions among different rules are handled by lateral connections connections. The representation is emergent from the context. Concepts and contexts are generated 'on demand'. I.e., if the agent is familiar with the current navigation setting (internal firing value close to perfect), no new context would be created. If not, new context would be created on the fly that make up of this lacking.

In a navigation setting, we have the X inputs as the sensor inputs (e.g. GPS input, vision input from cameras, or LIDAR input from the laser sensors).

The low level Z skills correspond to recognition results (recognized type information and location information) at a given location. High-level Z concepts correspond to the actions taken in different driving settings (e.g., turn slightly right when an obstacle is detected on the left-hand side).

Using this framework, DN-2 learns multiple Z concepts using different levels of reasoning: "what" action to take at a specific location ("where") and "which" recognized object is most relevant to the current situation.

Unlike a TM where the internal states must be human-defined and their transition rules must be hand-crafted, DN-2 uses emergent representation internally to learn clear logic with optimality.

In the following section we present three experiments of DN-2:

1. Simulated navigation with maze environment. This experiment is similar to the simulation in [48] but we are training the agent with hierarchical concepts and simulated visual inputs. The DN-2 agent in this simulation is equipped with three sensory areas (X areas) and six concept areas (Z areas). It is taught to recognize object types and locations as its lowest level of concept. Based on the recognition results the agent is then taught different skills (small navigation segments) with the action motor supervised and the recognition motors unsupervised. The agent is then taught about the two major routes from start to finish, with

only the highest level of concept "route" supervised. The agent needs to chain different lower-level skills together with no supervision from the teacher.

2. Batch vision data learning. We compare the performance of standard Convolutional Neural Network (AlexNet with 20 convolution kernels of size 5 + 2 by 2 max pooling + 50 convolution kernels of size 5 + 2 by 2 max pooling + 500 neuron fc layer + fc layer to output [25, 60]), DN-1 and DN-2 using the same dataset. Although data is collected in batch mode and the benchmark CNN is trained with iterations, we train the DNs with incremental on-line learning set up. The CNN iterated through the dataset for 500 iterations to reach convergence, while the DN networks only viewed the dataset twice with better performance. This experiment demonstrated the capability of DN-2 and also helped us to find desirable hyper-parameters.

3. Real-world navigation with stereo RGB images. We put the DN-2 agent with the hyper-parameters in the previous experiment into a real-world scenario. The agent is taught to recognize objects at different locations and then learn to navigate in different sections. The testing route is now disjoint from the training routes, and the agent needs to learn to generalize what it learned in these novel settings.

CHAPTER 5

SIMULATION WITH MAZE ENVIRONMENT

The navigation agent and a sample of the simulated environment are presented in Fig. 5.1. This simulation helps us to verify our argument that DN-2 learns the navigation FA perfectly with enough resources. The video demonstrating the training and testing process can be found at https://youtu.be/CbhS1qvWZn0.

5.1 The environment and the teacher FA

The environment is a block-based maze with different types of blocks: open, wall, obstacle, traffic light, and destination. Each block, except for the traffic light, can be only of one type, with a size of 50 pixels in height and 50 pixels in width. Traffic lights are small objects on top of a wall block with 20 pixels in width and 20 pixels in height. Open blocks are transparent. Wall blocks are red. Obstacle is blue. Destination is dark green. Traffic light has two colors, purple(stop) and light green(pass).

The interface shows the maze environment from a top-down viewing angle with all the blocks plotted in 2D. The vision sensor of the agent gives it a 43 by 30 image, as shown in Fig. 5.1. The walls are 30 pixels tall, while the obstacles are of 15 pixels tall and occupy the bottom half of the image. The traffic lights are 15 pixels tall and occupy the top half of the image.

The teacher FA follows the hand-crafted rules listed below:

1. Follow GPS direction when there is no obstacle or traffic light within the range of agent's vision input.

2. If there is an obstacle seen by the agent and the obstacle is less than 20 pixels away (or, wider than 8 pixels in the vision image of the agent), the agent should turn and move toward the open block to avoid hitting the obstacle.

3. If there is a purple traffic light seen by the agent, the agent should stop and wait.

4. If there is a light green traffic light seen by the agent, the agent follows the other rules.

5. When stepping from a block to its adjacent block, the agent should put down a marker to keep track of distance.

6. After the agent reaches the destination, it should go back to the starting point.

5.2 The agent

The agent is of size 20 by 20 pixels and moves continuously inside the simulated environment.

The network inside the agent has three X areas:

X_{vision}: The agent has vision of 210 degrees ranging 75 pixels. This 210 degrees view gives the agent a 2D RGB image of 43 by 20 pixels.

X_{gps}: The agent is also equipped with a simulated GPS to find the correct turn at each cross-road. GPS is shown as a black arrow in the GUI. GPS cannot sense the presence of obstacles thus the agent cannot follow the GPS signals blindly. There are three input neurons in this area, corresponding to left, forward, and right.

X_{mark}: The agent is equipped with a marking sensor (a single neuron) which would be firing with value one if the agent senses its previously put down marker.

The agent also has 7 Z areas:

Z_{where}: Lowest level of concept. There are 43 neurons in this area corresponding to the 43 possible different locations in the input image.

Z_{what}: Lowest level of concept. There are four types of objects the agent needs to recognize: open, obstacle, traffic light (stop), traffic light (go).

Z_{scale}: Lowest level of concept. The agent is trained to recognize objects at different scales (from a scale of 1 pixel wide to scale of 21 pixels wide).

Z_{action}: Mid-level concept. There are four neurons in this area: forward, left, right and stop. Each forward movement advances the agent's position toward its heading position by 1 pixel. Each left/right turn increases/decreases the agent's heading by 10 degrees.

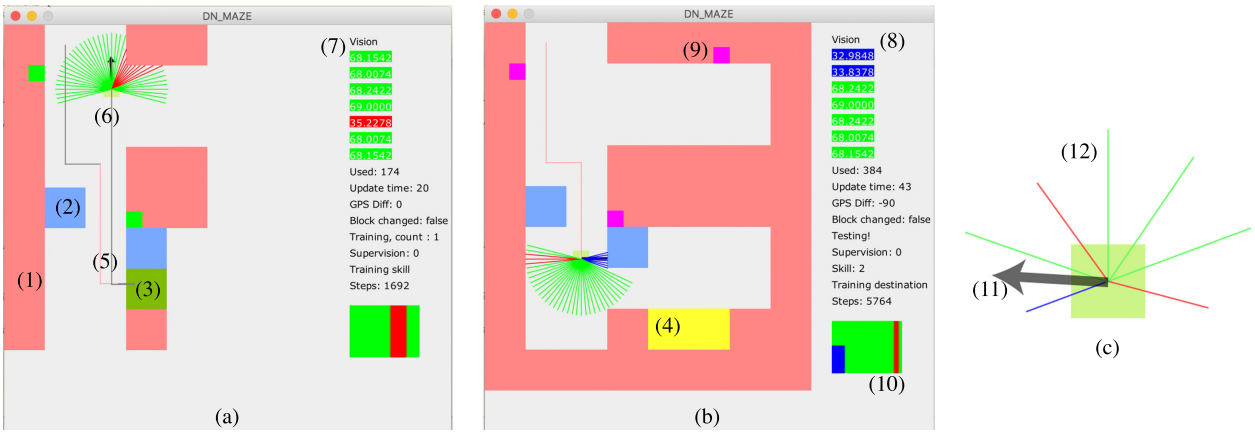

Figure 5.1: Simulated maze environment and the agent design. (a) and (b): simulated environment and corresponding GUI. The environment is 450 pixels by 450 pixels, with the maze no larger than nine by nine blocks. The agent is 20 pixels by 20 pixels, moving continuously in the maze environment. (1) wall blocks are red blocks in the GUI. (2) obstacle blocks are blue blocks in the GUI. (3) destination blocks are green block in the GUI. (4) reward/punishment blocks are yellow blocks in the GUI. (5) The agent's route is presented in the GUI. Black routes are skills already learned or going back routes. Red routes are novel skills to be learned. (6) Agent in the environment. Discussed in detail in subfigure (c). (7) and (8) Information panel for training and testing. (9) Traffic light block with two states. In (a) these traffic lights are at 'go' state, and in (b) these traffic lights are at 'stop' state. (10) 2D vision image is seen by the agent. (c) agent representation in GUI. (11) GPS signal from the environment. GPS indicates where the destination is in the simulated environment. But GPS is not aware of the obstacles in the environment thus the agent needs to learn skill "avoid obstacle". (12) Vision sensor of the agent. Here to simplify things we show seven vision lines out of the 43 vision lines forming a 210-degree vision ranging 75 pixels around the agent.

Z_{skill}: Higher level concept. There are eight neurons in this area corresponding to 8 different scenarios that are essential for navigation in the simulated environment.

Z_{route}: Highest level concept. There are two ways to reach the same destination in the final test. Thus this area has three neurons: route 1, route 2, and go back.

Z_{mark}: There are two neurons in this area, corresponding to putting down marker or not putting down marker. This motor can be viewed as the write head of our ETM, and the marker would be sensed by X_{mark} in the following iterations of navigation in the same environment.

We allocated 1300 Y neurons inside the agent. We organized the lessons such that at only type 101 neurons are released when learning where-what and basic skills. Type 011 neurons are released when learning individual routes.

Algorithm 1: Environment for teaching individual skills

initialize agent DN2; initialize environment;
for $z_{\text{skill}} \leftarrow$ *1 to skill_num* **do**
  for $i \leftarrow 1$ *to epoch_num* **do**
    Initialize random maze with crucial blocks;
    // Agent moves to the destination block;
    while *Agent is not at destination* **do**
      Get supervised action $\mathbf{z}_{\text{action}}$ from teacher;
      Get current X input $\mathbf{x}$ (for all X areas);
      Supervise agent with $\mathbf{x}$ and $\mathbf{z}_{\text{action}}$, $\mathbf{z}_{\text{skill}}$ (other concepts are zeros vectors);
      Update agent's DN2;
      Agent moves according to supervision;
    end
    // Agent goes back to initial position ;
    while *Agent is not at initial position* **do**
      Get supervised action $\mathbf{z}_{\text{action}}$ from teacher;
      Get current X input $\mathbf{x}$ (for all X areas);
      $\mathbf{z}_{\text{skill}} \leftarrow$ go back Supervise agent with $\mathbf{x}$ and $\mathbf{z}_{\text{action}}$, $\mathbf{z}_{\text{skill}}$ (other concepts are zeros vectors);
      Update agent's DN2;
      Agent moves according to supervision;
    end
  end
end

5.2.1 Stage 1: Learning concept where, what and scale

We randomly generate mazes and put the agent into random locations of the maze. At each time the teacher supervises the where, what and scale motors according to the current object in the input image. We trained the agent for 1000 images with where what information. At this stage only type 101 neurons are used.

5.3 Stage 2: Skills, route, and distance

5.3.1 Teaching skills

At this stage, we train the agent seven basic skills to navigate in the maze environment. These skills are presented in Fig. 5.2. Note that the presented mazes are only one instance among numerous

Algorithm 2: Environment for teaching route concept

for $z_{\text{route}} \leftarrow$ *1 to route_num* **do**
  Initialize maze with specific route;
  while *Agent is not at destination* **do**
    Get current X input $\mathbf{x}$ (for all X areas);
    Get current $\mathbf{z}_{\text{mark}}$ from environment;
    Supervise agent with $\mathbf{x}$, $\mathbf{z}_{\text{route}}$ and $\mathbf{z}_{\text{mark}}$ (other concepts are emergent from previous update);
    Update agent's DN2, record emergent skill and action;
    agent moves according to emergent action;
  end
  // Agent goes back to initial position;
  while *Agent is not at initial position* **do**
    Get supervised action $\mathbf{z}_{\text{action}}$ from teacher;
    Get current X input $\mathbf{x}$ (for all X areas);
    $\mathbf{z}_{\text{skill}} \leftarrow$ go back;
    $\mathbf{z}_{\text{route}} \leftarrow$ go back;
    Supervise agent with $\mathbf{x}$, $\mathbf{z}_{\text{action}}$, $\mathbf{z}_{\text{skill}}$, and $\mathbf{z}_{\text{route}}$;
    Update agent's DN2;
    agent moves according to supervision;
  end
end

randomly generated maze environment with key blocks unchanged. Thus the learned skills are environment invariant when learning takes enough iterations.

When teaching skills, we are supervising Z_{action} and Z_{skill} during each network update. The lower-level concepts (where, what and scale) are emergent, based on the learning results of the previous stage. The higher level concepts are all zero vectors and no connection is learned in the irrelevant concept zones. Teacher's behavior follows Algorithm 1.

After the agent reaches the destination, the teacher leads the agent to move back to the starting position. This is to simulate the continuous context of the agent inside the simulation with no sudden change of the external environment.

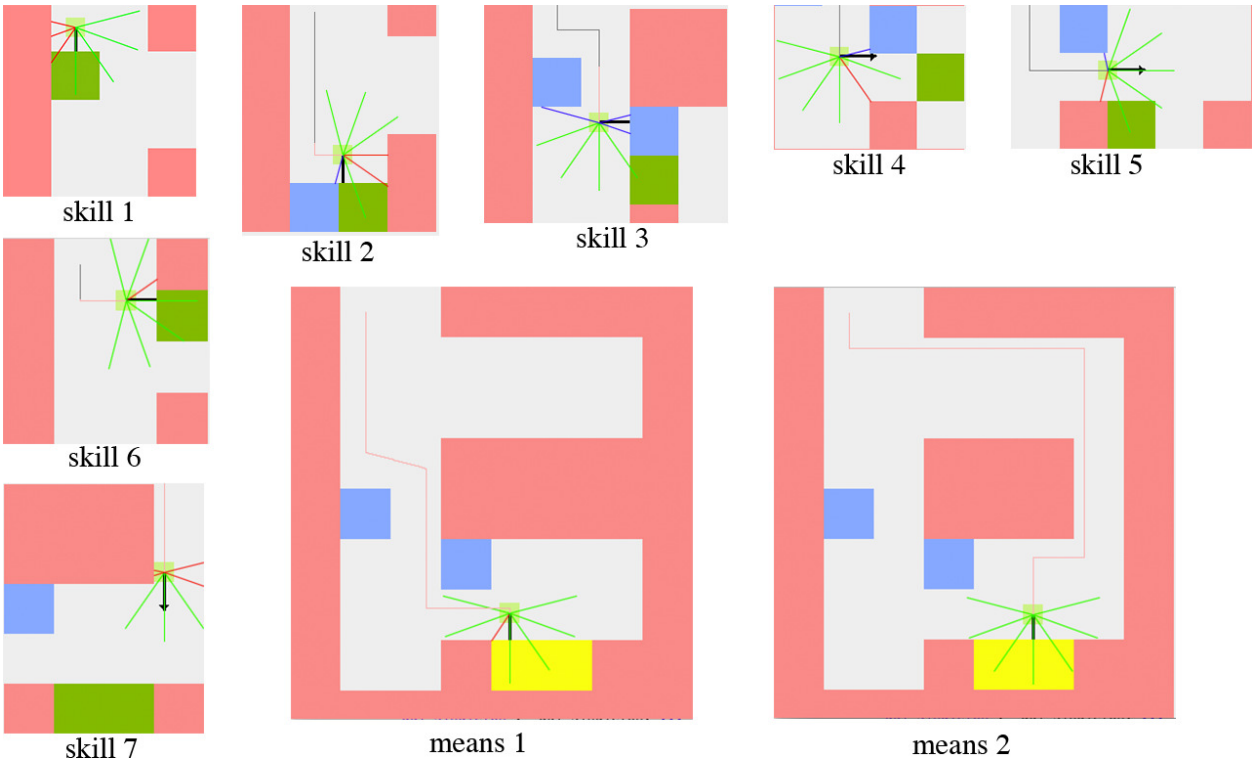

Figure 5.2: Skills and routes taught to the navigation agent in the simulation experiment. Skill 1: move forward when the next block is open. Skill 2: avoid obstacle and correct facing direction. Skill 3: turn left on the corner with GPS indicates left and avoid the obstacle. Skill 4: move through the narrow path. Skill 5: turn right when GPS indicates right. Skill 6: turn left when GPS indicates left. Skill 7: turn left and then turn right to reach the destination. During teaching, traffic lights would be placed along the route as shown in Fig. 5.1, the agent needs to learn to follow the rules of the traffic light as well. Although the agent has 43 vision lines as shown in Fig. 5.1, we show only 7 of these for clarity. A full video of training and testing is available at https://youtu.be/CbhS1qvWZn0.

5.3.2 Teaching route concept

After each skill is learned for enough iterations, we teach the agent two ways to navigate to the entrance at the lower right corner of a specific environment. The teacher's behavior is presented in detail in Algorithm 2.

At this stage the Z_{action} and Z_{skill} are emergent (emerged from the computation of the agent). The teacher only supervises Z_{route} and the write head Z_{mark} of the network.

5.3.3 Evaluation of performance

Performance of the simulated agent is evaluated at two stages:

1. Skill chaining with higher level concept supervised. When learning Z_{route}, the lower-level motors are emergent. The agent needs to successfully chain these skills together. We evaluate the success rate of skill chaining under such situation and report it in Sec. 5.4.1.

2. Navigation with higher level concept unsupervised. The agent only relies on the GPS and vision input, with all Z motors emergent. We evaluate the success rate of navigation (reaching the final destination) under such situation and report it in Sec. 5.4.1.

5.4 Experiment results and analysis

5.4.1 Skilling learning and chaining

We initialized 15 DN-2 agents in parallel to learn the lower-level skills first according to Algorithm 1. Then after these basic skills are mastered by the agent, the teacher leaves the lower-level skills emergent while supervised the higher level concept zones according to Algorithm 2. At this stage, the agent needs to chain different lower-level skills together, with the navigation context different from the context during the previous stage. As shown in Table 5.1, all of the 15 DN-2 agents successfully chained these lower level skills together when training higher-level skills.

5.4.2 GPS blurring and noisy inputs

The next experiment we did is to blur the GPS input and the sensor inputs at the same time by different degrees, shown in Table 5.1. As learning route-1 (5 subtasks) is a more difficult task compared to learning route-2 (3 subtasks), more agents failed at learning route-1 when the GPS is blurred. Nevertheless, all agents succeeded in learning both routes when the noise level is relatively low (less than five percent).

Table 5.1: Experiment result

noise level	0%	5%	10%	20%
route 1 chaining	15/15	15/15	10/15	2/15
route 1 GPS blurring	15/15	15/15	6/15	0/15
route 2 chaining	15/15	15/15	15/15	15/15
route 2 GPS blurring	15/15	15/15	14/15	13/15
total	60/60	60/60	55/60	32/60
success rate	100%	100%	91.67%	53.33%

5.5 Discussion

The video recording the entire training and testing scenario can be found at https://youtu.be/CbhS1qvWZn0. The network successfully learned the navigation rules listed in Sec. 5.1 with 100% accuracy when noise in input or GPS signals is small. The result verifies our claims with enough resources, DN-2 learns the FA error-free using emergent representation.

CHAPTER 6

REAL-WORLD EXPERIMENTS AND RESULTS

6.1 Batched vision data learning

AIML 2016 contest [1] provided batch data of three modalities (vision, audition, and text) for on-line learning agents to test their learning capability with limited resources. In this paper, we use AIML vision data to validate our design and select hyperparameters for our real-world navigation application. This dataset contains around 4,109 images collected using a mobile device and the onboard camera. Each image (gray scaled image, rescaled to 38 by 38 pixels) is labeled with detailed information about the desired action, the corresponding GPS signal, the most prominent landmark and the landmark's location. Sample images from this dataset are presented in Fig. 6.2.

The baseline model (implemented with Convolutional Neural Network, i.e., an AlexNet with fully-connected layer concatenated with GPS information) achieved an error action rate of 24.70%, with around 500 epochs of iteration through the training data [60] Best performing model of the contest achieved an error rate of 26.40% at the first epoch of training with 1500 globally connected neurons in Y area. With multiple views and additional resources (comparison in Table 6.1) a DN-1

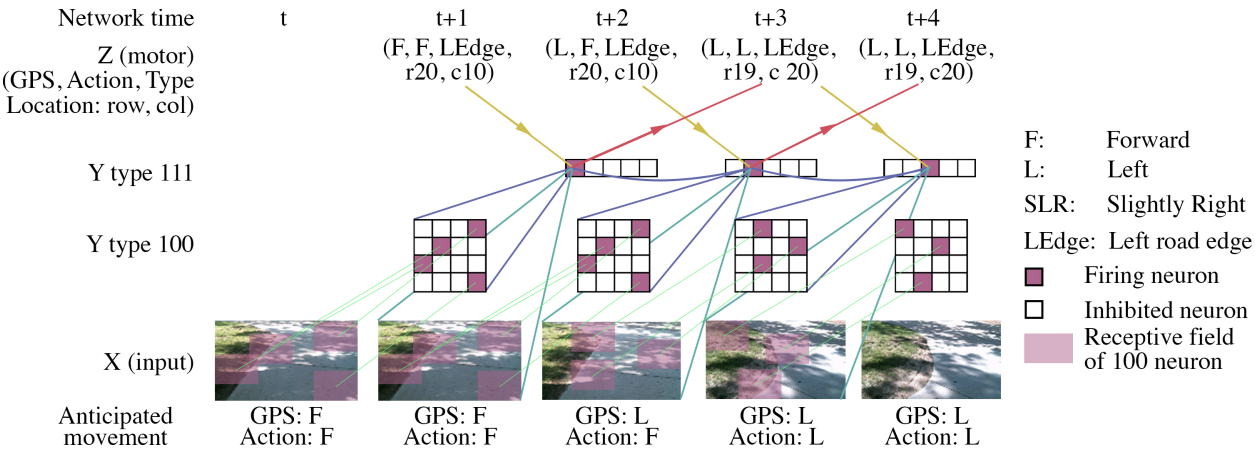

Figure 6.1: Real-world experiment flow chart. The lower level Y neurons (type 100) detect local features in the input image. The higher level Y neurons are more robust against changes in local variances, while forming a temporal context for navigation.

(SR, F, M, OffR) (St, F, UL, Cr) (F, F, M, *)

Z: (Action, GPS, Attention, Type)

Action: R, SR, F, SL, L, St
R: right SR: slightly right
F: forward SL: slightly left
L: left St: stop

GPS: R, F, L, St
R: right F: forward
L: left St: stop

Attention: UL, LL, UR, LR, M, *
UL: upper left LL: lower left
UR: upper right LR: lower right
M: middel *: free

Type: Ob, Cr, OffR, *
Ob: obstacle Cr: crossroad
OffR: offroad *: free

Figure 6.2: Sample data from the AIML 2016 navigation dataset. All images are labeled with four Z motor concepts: action, gps, attention and type. Data is collected around university campus using an HTC One M8 android device.

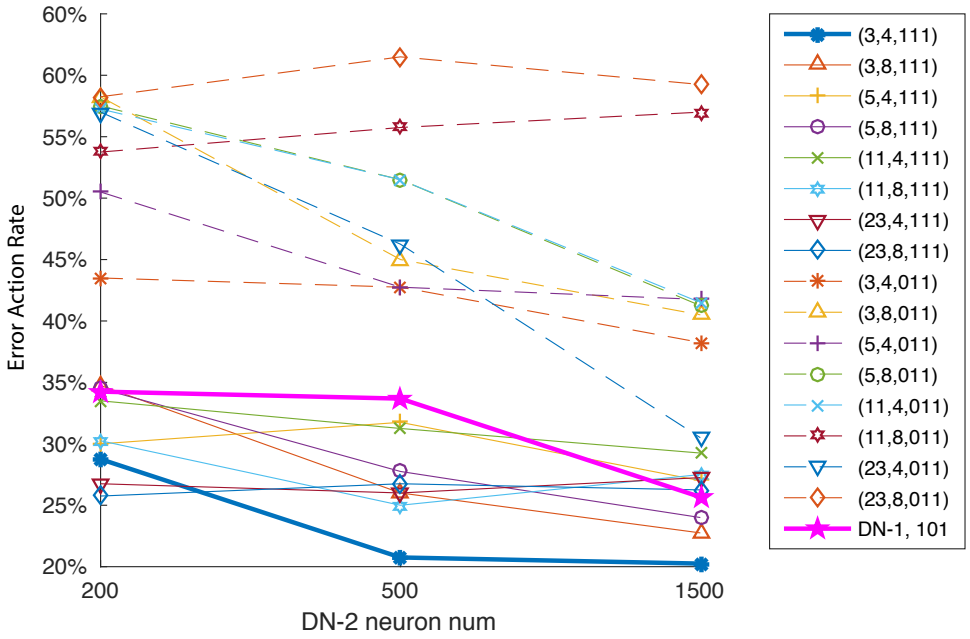

Figure 6.3: AIML 2016 vision performance comparison among different network configurations. The X axis is the number of high level neurons in DN-2 (i.e. type 101, type 011 and type 111). The Y axis is the error action rate with four fold cross validation on the AIML testing data. Each configuration is labeled with (receptive size in type 100 neurons, number of type 100 neurons at each pixel location and the type of high level neurons used). DN-2 with local 100 neurons and 500 high level 111 neurons offers the best performance with reasonable computational resource requirements.

Table 6.1: DN-2 performance compared to benchmarks on AIML 2016

		CNN	DN_1_Best	AIML_DN	DN_2
# of neurons	100	46,986	0	0	4900
	101		3,900	1,500	0
	111		0	0	500
best performance		24.70%	22.80%	26.50%	20.75%

implementation achieved an error rate of 22.80% [60].

As discussed in previous sections, DN-1 does not have a hierarchy in Y (spatially or temporally) thus have limited power of representation. We conducted experiments with DN-2 with different types and numbers of high level Y neurons, see Fig. 6.3. To make a fair comparison and simulate incremental learning scenario, all networks (except for Convolutional Neural Net) went through the following training and testing processes:

1. Training low-level representations with where-what information. If we are training a DN-2 network then we first release the type 100 neurons to form low-level representations like edges and gradients. With DN-1 there is no type 100 neuron thus this step is skipped.

2. Training higher level representations with navigation action, GPS information, and where-what recognition results. With a DN-2 network type 111 (or 011, 010, etc.) neurons are released according to the previously decided growth rate table $m(t)$. At this stage, higher level representation is formed based on the lower level representation at stage 1.

3. Testing action accuracy with network frozen. The network is no longer updating at this stage. We test the performance of the network by recording the accuracy of its generated action given each image.

We experimented several different network configuration with four-fold cross-validation. The final result is presented in Fig. 6.3 and Table 6.1. The best network, with type 100 and 1500 type 111 neurons, achieved an error rate of 20.01% after viewing the training data for only once, which is more accurate than the CNN and DN-1 benchmark.

Figure 6.4: Real-time experiment setup. 1. Stereo USB camera connected to the mobile device. The cameras are streaming a 480 by 640 stereo RGB image pair in real-time. 2. Oneplus 5 phone running DN-2 equipped navigation application in CPU and GPU mode. 3. Moga pro controller for motor supervision and testing result recording. 4. The navigation application interface for training and testing.

6.2 Real-world navigation: incremental training and testing

Guided by the batch experiment results, we developed an android application for real-time pedestrian navigation using DN-2.

The general setup for the navigation application is presented in Fig. 6.4. The application is running on an OnePlus5 android phone connected to a stereo camera (two USB webcams mounted on top of a helmet). The four motors, shown in Fig. 6.1, can all be supervised via the Moga Pro Bluetooth controller during training. The controller is also used to record the network's performance during testing.

The network is trained around the campus of the university to learn the task of autonomous navigation on the sidewalk. Fig. 6.8 provides an illustration of the extensiveness of training and testing. The inputs to the DN were from the same mobile phone that performs computation, including the stereo image from a separate camera and the GPS signals from the Google Directions

interface. The outputs of the system include heading direction or stop, the location of the attention, and the type of the object at the attended location (which detects a landmark), and the scale of attention.

The wide variety of real-world visual scenes implied by the extensive routes in Fig. 6.8 presented great and rich challenges to this camera-only system without using any laser device. DN-2 uses multiple types of neurons to form robust representations about the road edges and obstacles as discussed in Sec.1.1.

6.2.1 Training and testing

We set the growth hormone of DN-2 to use only two types of neurons: type 100 for low-level image feature extraction and type 111 for robust high-level representations, according to the result in our batch experiment.

Testing is performed with the network frozen (weights not updating but the neurons are still generating responses). As shown in Fig. 6.8 the testing routes are novel settings to the learned network, but with similar obstacles, roads, and bushes compared to the training settings.

Performance of the network is summarized in Table .1. Performance of the network is evaluated using two different metrics: different from user's intention (diff) and absolution errors (error). The difference is that in a 'diff' situation the network still can recover from the movement in subsequent frames (e.g., zig-zagging instead of going straight forward), while the absolute errors are defined as the situations where the network gets stuck into an unrecoverable action (e.g., stop and not moving or bump into obstacles).

According to Table .2, the most errors we got are from untrained obstacles (e.g., bicycles in the middle of the lane, or pedestrians on skateboards). This can be resolved by more extensive training and a larger network.

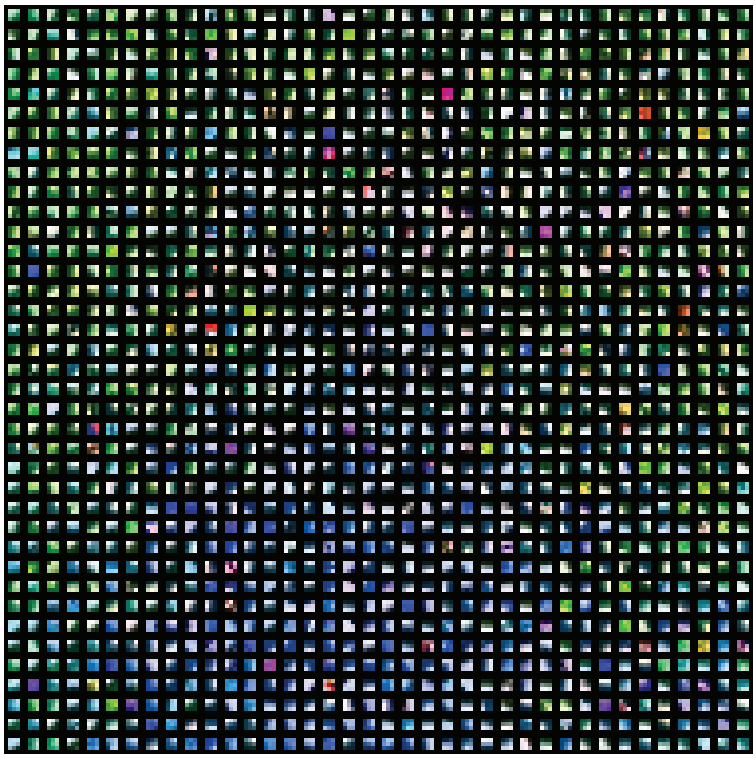

Figure 6.5: Bottom-up weights learned by type 100 neurons. Each 100 neuron focuses on a 5 by 5 region on the input image as shown in Fig. 6.1, and conduct local dynamic top-k competition. As the neurons take stereo RGB inputs, we show only the weights corresponding to the left camera. Each small square is the 5x5 rgb weight corresponding to a specific neuron. The weight values are normalized within range 0 and 1. These bottom-up weights are projected to image space in Fig. 6.6 and Fig. 6.7.

6.2.2 Visualization: Bottom-up weights of type 100 neurons

Fig. 6.5 shows part of the bottom-up weights of the type 100 neurons in the trained network. As shown in the figure, the lower-level neurons pick up basic features such as edges and color gradients in the image. After dynamic top-k competition, the firing neurons among these low-level neurons focuses on the most prominent features locally, which can be either road edges (important features) or shadow edges (distractors). The firing pattern in those lower-level neurons is fed into type 111 neurons (higher-level of representation), which use synaptic maintenance to cut away the unstable connections with distractors, forming a more robust representation.

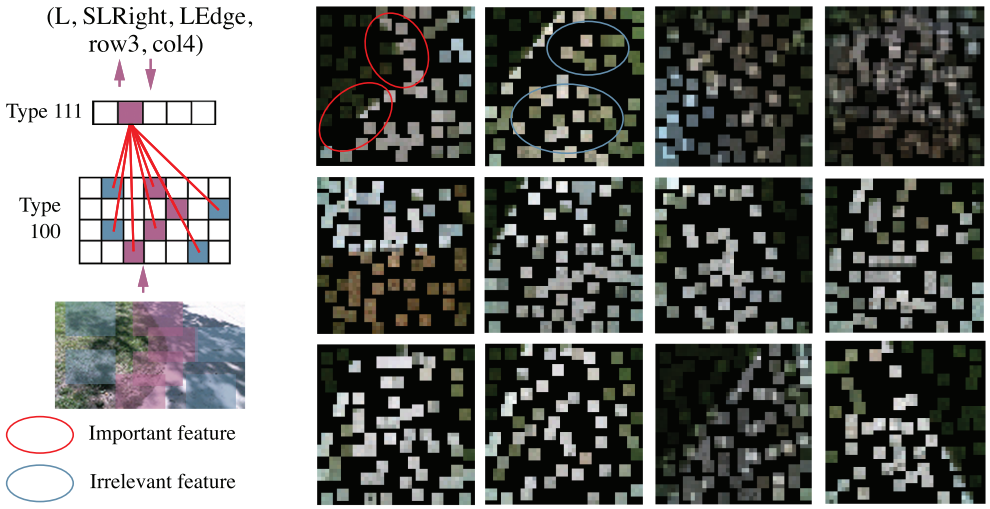

Figure 6.6: Projected lateral weights for type 111 neurons with no synaptic maintenance. Type 111 neurons with low firing ages forms evenly distributed attention due to the local competition zones of lower level 100 neurons.

6.2.3 Visualization: Lateral weights from type 100 neurons to type 111 neurons

Fig. 6.6 and Fig. 6.7 show the projected lateral weights of high level 111 neurons in the trained network. Each sub-figure shows the Y to the Y connection of a 111 neuron, with the connected 100 neurons' bottom-up weights projected into the image space. Each subfigure is equivalent to showing only the red receptive fields for type 111 neurons in Fig. 6.1.

111 neurons in Fig. 6.6 have lateral connections evenly distributed among lower level 100 neurons, as they have not entered the synaptic maintenance stage due to their low firing ages.

Lateral weights of 111 neurons that have entered synaptic maintenance stage are shown in Fig. 6.7. Synaptic maintenance cuts away the unstable connections with high variances. As shown in the visualization, these neurons focus their attention on road edges and become invariant to the changes in monotone shadows.

This visualization proves that using multiple types of neurons combined with lateral connections, we can form robust hierarchies of representation with the internal features.

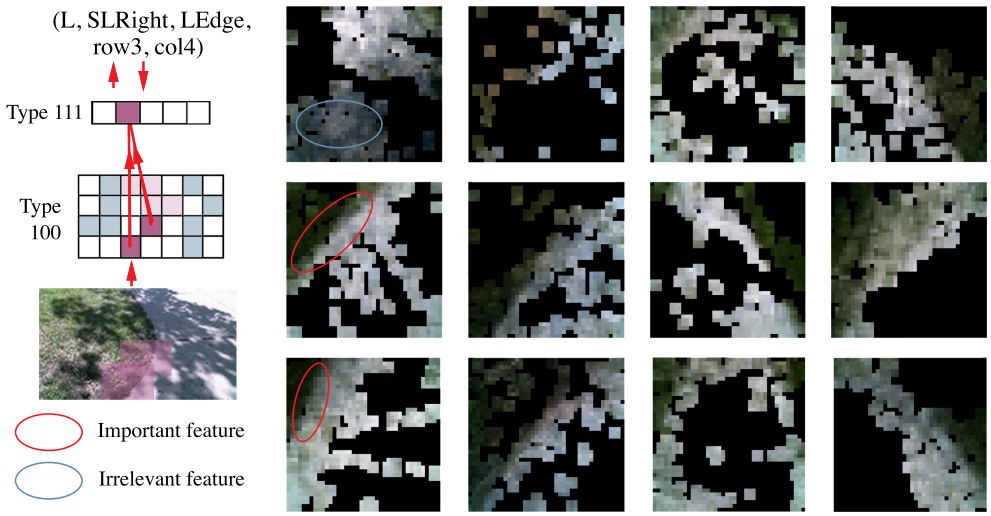

Figure 6.7: Projected lateral weights for type 111 neurons with synaptic maintenance. After synaptic maintenance the high-level neurons focuses attention on consistent road edges and become invariant to the changes in the highly variant shadow shapes. The connections from type 100 neurons to type 111 neurons shifts towards these consistent features, while the highly unstable connections are cut from the range of connection.

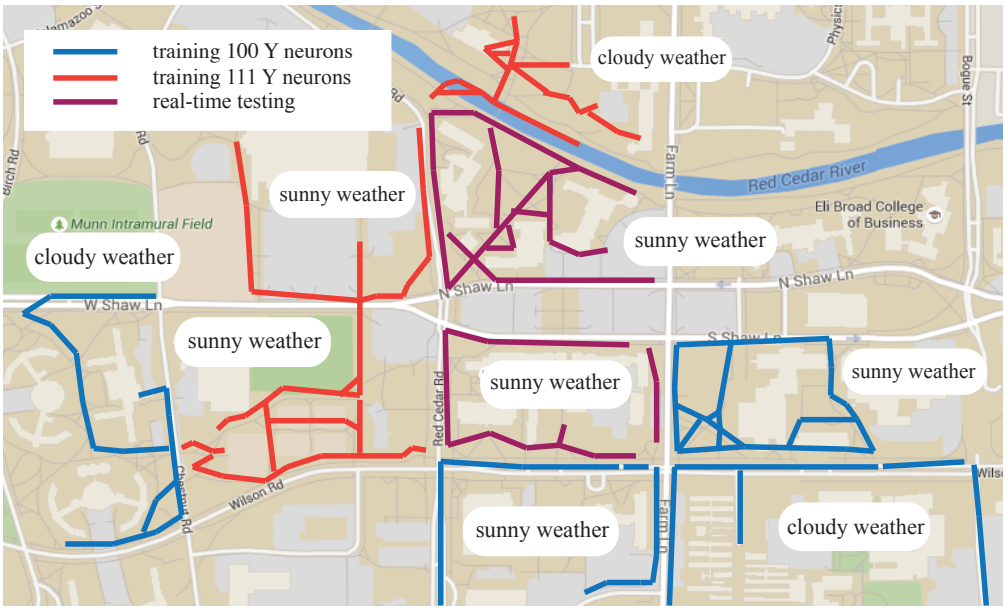

Figure 6.8: Training and testing routes around the university during different times of day and with different natural lighting conditions. Disjoint testing sessions were conducted along paths that the machine has not learned.

CHAPTER 7

CONCLUSION

In this paper, we established that DN-2 is an emergent Turing Machine that learns the rules of a navigation TM with hierarchical representation. DN-2 uses multiple types of neurons to form hierarchical representation internally. Compared to FA based methods with no hierarchy in representation, this framework forms robust representations of important features while disregarding distractors in inputs. Experimental results demonstrated that DN-2 learns navigation rules with satisfactory performance.

APPENDIX

Table A.1: Real-time training and testing detail

<table>
<tr><td rowspan="5">Training</td><td>Date</td><td>Time</td><td>Detail</td><td># of samples</td></tr>
<tr><td>8/5/2017</td><td>14:30 - 15:30</td><td>Training type 100</td><td>3051</td></tr>
<tr><td>8/6/2017</td><td>14:15- 15:30</td><td rowspan="2">Training type 111 Neuron num: 500</td><td>1492</td></tr>
<tr><td>8/7/2017</td><td>14:00- 15:45</td><td>940</td></tr>
<tr><td colspan="4">Right: 214, SLRight: 338, Forward: 1182, SLLeft: 352, Left: 210, Stop: 136</td></tr>
<tr><td rowspan="3">Testing</td><td colspan="4">Total Segment: 18, Total Steps: 1155</td></tr>
<tr><td colspan="4">Diff count: 62 (5.36%) Error Count: 9 (0.78%)</td></tr>
<tr><td colspan="4">Right: 92, SLRight: 163, SLLeft: 156, Left: 97, Forward: 560, Stop: 87</td></tr>
</table>

Video recordings of the experiment can be found on our youtube video list https://www.youtube.com/playlist?list=PLVs0MJh9CrcFZqYxCJ9pM6zRQU90kCxj9.

A.1 Proof of Lemma 1

Proof. The proof is broken down into two steps. Step 1 demonstrates the firing pattern in each zone is actually a binary ML estimator of the current input conditioned on $N(t)$. Step 2 shows after learning the weights are then incrementally updated according to ML estimation.

Step 1, case 1 (new neuron enters learning stage at time $t+1$): When a new neuron g enters learning stage for the first time, it memorizes the input vector $\mathbf{p}(t+1)$ perfectly (explained in Sec. 2.2.4). This neuron also suppresses other neurons inside its $\bar{g}$ from firing (via competition as the neuron's firing value is perfect), thus becoming the ML estimator of the current input $\mathbf{p}(t+1)$.

As is stated in Sec. 2.2.4, a new neuron would be initialized when the current match is bad. The current match would be bad for two reasons: 1) the current input is a novel input. In this case the new neuron would not affect learning of the previous neurons as no active neurons have learned this input yet. This new neuron is the only neuron that has learned the current input thus is the ML estimator of the current input. 2) the current input has already been learned by an active neuron j

Table A.2: Real-time testing results

Section	Total	Diff	Error	L	SL	F	SR	R	Stop	Descriptions	Detail
1	50	4	0	7	6	27	10			F->SL->F-> L ->SR->L->F	rain stain, rocks, facing direction correction
2	50	3	0		8	21	5	11	5	F->R->SL->R ->F->SR->F->Stop	rain stain, overturn, facing direction correction
3	59	1	0		8	41	3	7		F->SR->F->R ->SL->R->F	rocks, overturn, facing direction correction
4	60	3	0	10	11	18	5	8	8	F->L->SR->L ->F->R->SL ->R->F->Stop	rain stain, overturn, facing direction correction
5	107	4	0		17	62	13	10	5	F->SR->SL->SR ->R->SL->R->F ->Stop	shadows, dirt road, protruding trees, overturn, facing direction correction
6	104	6	3	27	13	35	22		7	F->SL->SR->SL ->L->SR->L->SL ->SR->F->Stop	shadows, protruding trees, overturn, facing direction correction, error at overturn
7	73	5	2		7	40	9	13	4	F->R->SR->F-> SL(Error)->F->Stop	noval obstacle, shadows, bushes, overturn, error at untrained obstacle
8	75	4	2	12	14	29	8	7	5	F->SL(Error)->F-> L->F->SL->R->F->Stop	shadows, facing direction correction, overturn
9	56	3	2	11		26	7	6	6	F->SR->L->F->Stop	facing direction correction, shadows
10	55	2	0		7	33	12		3	F->SR->SL->F-> SL->Stop	untrained obstacle, shadows, bushes
11	55	0	0		9	27		13	6	F->SL->F->SL-> F->R->F->Stop	facing direction correction, untrained obstacle
12	66	7	0	13	7	25	13		8	F->SR->L->F-> SL->F->Stop	facing direction correction, bushes, shadows
13	71	5	0	11	12	23	8	10	7	F->SR->F->SL-> L->SR->L->F-> R->SL->R->F->Stop	facing direction correction, overturn, rocks on the side of rode, untrained obstacles, bushes
14	53	2	0		10	26	14		3	F->SR->F->SL->F ->SR->F->SL->SR ->F->SL->F->SR-> SL->SR->F->Stop	winding road, constant facing direction correction
15	41	3	0	6	5	20	6		4	F->SR->SL->F->SR ->L->F->Stop	winding road, bushes, facing direction correction, overturn
16	59	8	0		7	34	11		7	F->SR->F->SL->SR ->R->SL->F->Stop	shadows, uphill, facing direction correction, overturn
17	70	0	0		11	41	6	7	5	F->SR->F->SL->F ->SR->F->R->F->Stop	untrained obstacles, bushes, winding road
18	51	2	0		4	32	11		4	F->SR->F->SR->F ->SL->F	untrained obstacles, bushes, winding road
Total	1155	62	9	97	156	560	163	92	87		

(current best match). This rarely happens as

$$\begin{aligned} m(t) &> r_j = \langle \frac{\Sigma \mathbf{x}_i}{||\Sigma \mathbf{x}_i||}, \frac{\mathbf{p}}{||\mathbf{p}||} \rangle \\ &\geq \frac{1}{n} \Sigma \langle \frac{\mathbf{x}_i}{||\mathbf{x}_i||}, \frac{\mathbf{p}}{||\mathbf{p}||} \rangle \end{aligned} \tag{1}$$

where $\mathbf{p} = \mathbf{p}(t)$ is the current input, $m(t)$ is the current almost perfect match, $\mathbf{x}_i$ are the inputs that are learned by neuron j, and r_j is the response of neuron j. This means at least one input should be greatly different than $\mathbf{p}$ (their similarity measure much less than $m(t)$), but it still got picked up by the neuron in previous learning. We rarely observe this during learning.

Step 1, case 2 (no new neurons are added at time $t+1$): For each neuron $g \in Y, Z$, consider its inhibition zone at time t: $\bar{g}(t)$. $c(g,t)$ is the number of neurons in its inhibition zone $\bar{g}(t)$. The normalized weights of these c neurons can be denoted as $(\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_{c(g,t)})$.

Then we can define $c(g,t)$ Voronoi regions $R_j, j = 1, 2, ..., c(g,t)$ in $L(g,t)$ (receptive field of g at time t, which is a linear subspace of $X \times Y \times Z$), where each R_j contains all $\mathbf{p} \in L(g,t)$ that are closer to $\mathbf{w}_j$ than to other $\mathbf{w}_i$:

$$R_j = \{\mathbf{p} | j = \arg \max_{1 \le i \le c} \mathbf{w}_i \cdot \mathbf{p}\}, \quad j = 1, 2, ..., c$$

Given observation $\mathbf{p}(t+1)$, the conditional probability density $h(\mathbf{p}(t+1)|L(g,t), W(g,t))$ is zero if $\mathbf{p}(t+1)$ falls out of the Voronoi region of neuron g:

$$f\{\mathbf{p}(t+1)|L(g,t), W(g,t)\} = \begin{cases} f_i\{\mathbf{p}(t+1)|L(g,t), W(g,t)\}, \\ \qquad \text{if} \quad \mathbf{p}(t+1) \in R_i; \\ 0, \quad \text{otherwise} \end{cases} \tag{2}$$

where $f_i\{\mathbf{p}(t+1)|L(g), W(g)\}$ is the probability density within R_i. Note that the distribution of $f_i\{\mathbf{p}(t+1)|L(g), W(g)\}$ within R_i is irrelevant as long as it integrates to 1.

Given $\mathbf{p}(t+1)$, the ML estimator for the binary vector $\mathbf{y}(t)$ (also $\mathbf{z}(t)$) needs to maximize $f\{\mathbf{p}(t+1)|N(t)\}$, which is equivalent to finding the set of firing neurons $n(t+1)$:

$$\begin{aligned} n(t+1) &= \{\arg \max_{g \in Y,Z} f\{\mathbf{p}(t+1)|N(t)\}\} \\ &= \{g | g = \arg \max_{g \in Y,Z} f\{\mathbf{p}(t+1)|L(g,t), W(g,t)\} \\ &= \{g | g = \arg \max_{j \in \bar{g}(t)} \mathbf{w}(j) \cdot \mathbf{p}(t+1)\} \end{aligned} \tag{3}$$

since finding the ML estimator in Eq. 2 is equivalent to finding the Voronoi region where $\mathbf{p}(t)$ belongs. This is exactly what the Y area does, supposing $k = 1$ for top-k competition for each neuron's competition zone.

Step 2: Here we demonstrate the statistical efficiency of the LCA learning rule of $W(g)$, where $g \in n(t)$ (meaning that g is among the firing neurons at time t).

Eq. (11) in [58] shows the *candid* version of LCA is actually an incremental estimation of the average of the inputs that trigger firing in that specific neuron:

$$\begin{aligned} \mathbf{w}(g,t+1) &= \frac{a(g,t)-1}{a(g,t)}\mathbf{w}(g,t) + \frac{1}{a(g,t)}\mathbf{p}(t+1) \\ &= \frac{1}{a(g,t)}\Sigma_{i=1}^{a(g,t)}\mathbf{p}(t_i) \end{aligned} \tag{4}$$

where $\mathbf{w}(g,t)$ is the weight vector $\mathbf{w}(g)$ at time t. $a(g,t)$ is the age of neuron g at time t. $t_i, i = 1,2,...,a(g,t)$ are the times where neuron g wins dynamic top-k competition and fires.

Following the proof of optimality in LCA [58], statistical estimation theory reveals for many distributions (e.g., Gaussian and exponential distributions), the sample mean is the most efficient estimator of the population mean. This follows directly from Th. 4.1, p.429-430 in [29], which states that under some regularity conditions satisfied by many distributions (such as Gaussian and exponential distributions), the maximum likelihood estimator (MLE) $\hat{\boldsymbol{\theta}}$ of the parameter vector $\boldsymbol{\theta}$ is asymptotically efficient, in the sense that its asymptotic covariance matrix is the Cramer-Rao information bound (the lower bound) for all unbiased estimators via convergence in probability to a normal distribution:

$$\sqrt{n}(\hat{\boldsymbol{\theta}} - \boldsymbol{\theta}) \xrightarrow{p} N\{0, I(\boldsymbol{\theta})^{-1}\} \tag{5}$$

in which the Fisher information matrix $I(\boldsymbol{\theta})$ is the covariance matrix of the score vector:

$$\{(\partial f(\mathbf{x},\boldsymbol{\theta}))/(\partial\theta_1), ..., (\partial f(\mathbf{x},\boldsymbol{\theta}))/(\partial\theta_k)\} \tag{6}$$

$f(\mathbf{x},\boldsymbol{\theta})$ is the probability density of random vector $\mathbf{x}$ if the true parameter value is $\boldsymbol{\theta}$. The matrix $I(\boldsymbol{\theta})^{-1}$ is called information bound since under some regularity constraints, any unbiased estimator $\tilde{\boldsymbol{\theta}}$ of the parameter vector $\boldsymbol{\theta}$ satisfies $\text{cov}(\tilde{\boldsymbol{\theta}} - \boldsymbol{\theta}) \geq I(\boldsymbol{\theta})^{-1}/n$ (see, e.g., [29], p. 428 or [57], p. 203-204]).

Thus, as Weng et al. showed in [58], the LCA algorithms learns the optimal weights in the sense of maximum likelihood estimation.

Combining step 1 and step 2 we would have:

$$\theta(t+1) = \max_{\theta}(t) f\{\mathbf{p}(t)|N(t-1)\} \tag{7}$$

Please note that Lemma 1 applies to not only Y neurons, but also Z neurons as well. Z neurons act the same as Y neurons with a fixed range of lateral inhibition area to form different concept zones. Lemma 1 applies to any neuron that learns with regard to its inhibition zone and updates its weights using LCA learning rules.

A.2 Proof of Theorem 1

Proof. Intuitively, although $f'\{\mathbf{p}(t+1)|\mathrm{X}_0^t, \mathrm{Z}_0^t, \Gamma\}$ is in a different format compared to the $f\{\mathbf{p}(t+1)|N(t)\}$ in Lemma 1, the two probability functions are the same as $N(t)$ is determined by $\mathrm{X}_t, \mathrm{Z}_t$ and Γ. As there is no random weight initialization in DN-2, two DN-2 equipped learning agents would be exactly identical given the same hyper-parameter Γ and learning experience $\mathrm{X}_t, \mathrm{Z}_t$. To formally prove this we are going to recursively use the conclusion of Lemma 1.

At $t+1$, we can reuse Eq. (7) from Lemma 1.

$$\begin{aligned}
\theta(t+1) &= \max_{\theta(t)} f\{\mathbf{p}(t+1)|N(t)\} \\
&= \max_{\theta(t)} f\{\mathbf{p}(t+1)|\mathrm{L}(N(t-1), \mathbf{x}(t), \mathbf{z}(t), \Gamma)\} \\
&= \max_{\theta(t)} f^1\{\mathbf{p}(t+1)|N(t-1), \mathrm{X}_t^t, \mathrm{Z}_t^t, \Gamma\} \\
&= \max_{\theta(t)} f^2\{\mathbf{p}(t+1)|N(t-2), \mathrm{X}_{t-1}^t, \mathrm{Z}_{t-1}^t, \Gamma\} \\
&= ... \\
&= \max_{\theta(t)} f^t\{\mathbf{p}(t+1)|N(0), \mathrm{X}_0^t, \mathrm{Z}_0^t, \Gamma\} \\
&= \max_{\theta(t)} f'\{\mathbf{p}(t+1)|\mathrm{X}_0^t, \mathrm{Z}_0^t, \Gamma\} \qquad (8)
\end{aligned}$$

where L is the learning function of the network, and f^i is the probability density function of the current input $\mathbf{p}$, conditioned on the network i time steps ago and the learning experience from $t-i$ to t.

BIBLIOGRAPHY

BIBLIOGRAPHY

[1] Artificial intelligence machine learning contest 2016. http://www.brain-mind-institute.org/AIMLcontest/index-2016.html. Accessed: 2016-10-02.

[2] Arthur N Applebee and Judith A Langer. Instructional scaffolding: Reading and writing as natural language activities. *Language arts*, 60(2):168–175, 1983.

[3] David Barber. *Bayesian reasoning and machine learning*. Cambridge University Press, 2012.

[4] Francisco Bonin-Font, Alberto Ortiz, and Gabriel Oliver. Visual navigation for mobile robots: A survey. *Journal of intelligent and robotic systems*, 53(3):263–296, 2008.

[5] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.

[6] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jurgen Schmidhuber. Convolutional neural network committees for handwritten character classification. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1135–1139. IEEE, 2011.

[7] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[8] Gamini Dissanayake, Shoudong Huang, Zhan Wang, and Ravindra Ranasinghe. A review of recent developments in simultaneous localization and mapping. In *Industrial and Information Systems (ICIIS), 2011 6th IEEE International Conference on*, pages 477–482. IEEE, 2011.

[9] Guorui Feng, Guang-Bin Huang, Qingping Lin, and Robert Gay. Error minimized extreme learning machine with growth of hidden nodes and incremental learning. *IEEE Transactions on Neural Networks*, 20(8):1352–1357, 2009.

[10] Shai Fine, Yoram Singer, and Naftali Tishby. The hierarchical hidden markov model: Analysis and applications. *Machine learning*, 32(1):41–62, 1998.

[11] Sharon E Fox, Pat Levitt, and Charles A Nelson III. How the timing and quality of early experiences influence the development of brain architecture. *Child development*, 81(1):28–40, 2010.

[12] Bernd Fritzke. A growing neural gas network learns topologies. In *Advances in neural information processing systems*, pages 625–632, 1995.

[13] Paul A Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation*. John Wiley & Sons, 2017.

[14] A. Graves, G. Wayne, M. Reynolds, T. Harley, I. Danihelka, A. Grabska-Barwińska, S. G. Colmenarejo, E. Grefenstette, T. Ramalho, J. Agapiou, A. P. Badia, K. M. Hermann, Y. Zwols, G. Ostrovski, A. Cain, H. King, C. Summerfield, P. Blunsom, K. Kavukcuoglu, and D. Hassabis. Hybrid computing using a neural network with dynamic external meomory. *Nature*, 538:471–476, 2016.

[15] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[16] Geoffrey E Hinton. Deep belief networks. *Scholarpedia*, 4(5):5947, 2009.

[17] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

[18] John E Hopcroft, Rajeev Motwani, and Jeffrey D Ullman. Automata theory, languages, and computation. *International Edition*, 24, 2006.

[19] Gordon Hughes. On the mean accuracy of statistical pattern recognizers. *IEEE transactions on information theory*, 14(1):55–63, 1968.

[20] Finn V Jensen. *An introduction to Bayesian networks*, volume 210. UCL press London, 1996.

[21] Zhengping Ji, Juyang Weng, and Danil Prokhorov. Where-what network 1:"where" and "what" assist each other through top-down connections. In *Development and Learning, 2008. ICDL 2008. 7th IEEE International Conference on*, pages 61–66. IEEE, 2008.

[22] Teuvo Kohonen. The self-organizing map. *Neurocomputing*, 21(1-3):1–6, 1998.

[23] Dirk Kraft, Renaud Detry, Nicolas Pugeault, Emre Baseski, Frank Guerin, Justus H Piater, and Norbert Kruger. Development of object and grasping knowledge by robot exploration. *IEEE Transactions on Autonomous Mental Development*, 2(4):368–383, 2010.

[24] Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. The cifar-10 dataset. *online: http://www. cs. toronto. edu/kriz/cifar. html*, 2014.

[25] 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.

[26] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[27] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *AT&T Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, 2, 2010.

[28] Yann LeCun, Fu Jie Huang, and Leon Bottou. Learning methods for generic object recognition with invariance to pose and lighting. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–104. IEEE, 2004.

[29] Erich Leo Lehmann. *Theory of point estimation*. New York: Wiley, 1983.

[30] Martin Llofriu, Gonzalo Tejera, M Contreras, Tatiana Pelc, Jean-Marc Fellous, and Alfredo Weitzenfeld. Goal-oriented robot navigation learning using a multi-scale space representation. *Neural Networks*, 72:62–74, 2015.

[31] Matthew Luciw and Juyang Weng. Where what network 3: Developmental top-down attention with multiple meaningful foregrounds. In *International Joint Conference on Neural Networks*, pages 4233–4240, 2010.

[32] Matthew Luciw and Juyang Weng. Where-what network-4: The effect of multiple internal areas. In *Development and Learning (ICDL), 2010 IEEE 9th International Conference on*, pages 311–316. IEEE, 2010.

[33] John C Martin. *Introduction to Languages and the Theory of Computation*, volume 4. McGraw-Hill NY, 1991.

[34] Danielle S McNamara and Walter Kintsch. Learning from texts: Effects of prior knowledge and text coherence. *Discourse processes*, 22(3):247–288, 1996.

[35] Michael J Milford and Gordon F Wyeth. Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1643–1649. IEEE, 2012.

[36] Michael J Milford, Gordon F Wyeth, and David Prasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 1, pages 403–408. IEEE, 2004.

[37] Marvin L Minsky. Logical versus analogical or symbolic versus connectionist or neat versus scruffy. *AI magazine*, 12(2):34, 1991.

[38] Jonathan Mugan and Benjamin Kuipers. Autonomous learning of high-level states and actions in continuous environments. *IEEE Transactions on Autonomous Mental Development*, 4(1):70–86, 2012.

[39] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[40] Jean Piaget. *The Grasp of Consciousness (Psychology Revivals): Action and Concept in the Young Child*. Psychology Press, 2015.

[41] Thomas Serre, Lior Wolf, and Tomaso Poggio. Object recognition with features inspired by visual cortex. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 994–1000. Ieee, 2005.

[42] Hava T Siegelmann and Eduardo D Sontag. Turing computability with neural nets. *Applied Mathematics Letters*, 4(6):77–80, 1991.

[43] Hava T Siegelmann and Eduardo D Sontag. On the computational power of neural nets. *Journal of computer and system sciences*, 50(1):132–150, 1995.

[44] Xiaoying Song, Wenqiang Zhang, and Juyang Weng. Where-what network 5: Dealing with scales for objects in complex backgrounds. In *Neural Networks (IJCNN), The 2011 International Joint Conference on*, pages 2795–2802. IEEE, 2011.

[45] Jason Stanley and John W Krakauer. Motor skill depends on knowledge of facts. *Frontiers in human neuroscience*, 7, 2013.

[46] Joan Stiles and Terry L Jernigan. The basics of brain development. *Neuropsychology review*, 20(4):327–348, 2010.

[47] Huajin Tang, Weiwei Huang, Aditya Narayanamoorthy, and Rui Yan. Cognitive memory and mapping in a brain-like system for robotic navigation. *Neural Networks*, 87:27–37, 2017.

[48] Jun Tani and Naohiro Fukumura. Learning goal-directed sensory-based navigation of a mobile robot. *Neural networks*, 7(3):553–563, 1994.

[49] Alan Mathison Turing. On computable numbers, with an application to the entscheidungsproblem. *Proceedings of the London mathematical society*, 2(1):230–265, 1937.

[50] David C Van Essen and John HR Maunsell. Hierarchical organization and functional streams in the visual cortex. *Trends in neurosciences*, 6:370–375, 1983.

[51] Horatiu Voicu. Hierarchical cognitive maps. *Neural Networks*, 16(5-6):569–576, 2003.

[52] Lev Semenovich Vygotsky. *Mind in society: The development of higher psychological processes*. Harvard university press, 1980.

[53] Nikita Wagle and Juyang Weng. Developing dually optimal lca features in sensory and action spaces for classification. In *Development and Learning and Epigenetic Robotics (ICDL), 2012 IEEE International Conference on*, pages 1–8. IEEE, 2012.

[54] Yuekai Wang, Xiaofeng Wu, and Juyang Weng. Brain-like learning directly from dynamic cluttered natural video. In *Proc. International Conference on Brain-Mind, pages*, pages 1–8.

[55] J Weng. Why have we passed "neural networks do not abstract well"?". *Natural Intelligence: the INNS Magazine*, 1(1):13–22, 2011.

[56] Juyang Weng. Brain as an emergent finite automaton: A theory and three theorems. *International Journal of Intelligence Science*, 5(02):112, 2015.

[57] Juyang Weng, Thomas S Huang, and Narendra Ahuja. *Motion and structure from image sequences*. Springer Science & Business Media, 1993.

[58] Juyang Weng and Matthew Luciw. Dually optimal neuronal layers: Lobe component analysis. *IEEE Transactions on Autonomous Mental Development*, 1(1):68–85, 2009.

[59] Xiaofeng Wu, Qian Guo, and Juyang Weng. Skull-closed autonomous development: Wwn-7 dealing with scales. In *Proc. International Conference on Brain-Mind. East Lansing, Michigan: BMI Press*, pages 1–8. Citeseer, 2013.

[60] Zejia Zhengj and Juyang Weng. Mobile device based outdoor navigation with on-line learning neural network: A comparison with convolutional neural network. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 11–18, 2016.