Advances in Cognitive Systems 7 (2018) 97–116 Submitted 6/2018; published 12/2018

# The Partial Mental State Inducer: Learning Intuition with Few Training Examples and K-line Theory

**Tristan Thrush** TRISTANT@MIT.EDU
**Patrick Winston** PHW@MIT.EDU
Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 32 Vassar Street, Cambridge, MA 02139 USA

## Abstract

In this paper, we explain the Partial Mental State Inducer (PMSI), which models how humans can learn intuition for problem solving. We believe that this is the first model that unifies theories of K-lines, human inner language, human problem solving, and neural reinforcement learning. The result is human-like learning in a training environment that is too data-starved for traditional reinforcement learning to be successful. We present experiments in three distinct problem domains (natural language queries, equation solving, and robot planning) with only 20 training problems in each domain. A typical deep Q-network set-up often does not test better than a random agent on average, whereas PMSI can learn how to perform at a level that is close to optimal.

## 1. Introduction

Humans are able to develop an incredible intuition for solving problems with impressively few practice problems. Mathematicians can prove theorems with lemmas that inexplicably come to mind and game players can act on hunches that are more insightful than extensive searches. This indicates that there is a significant component of the human problem-solving capacity which is beyond the explanation barrier, even though humans can explain their solutions at some level.

We present the Partial Mental State Inducer (PMSI), which models part of the human capacity to develop intuition for a problem domain from only a few relevant practice problems.[1] Given the state of a problem and a goal, PMSI suggests relevant operator sequences for a separate general-purpose problem solver to use. The problem solver has access to operators that can manipulate the state of a problem and check goal conditions, and PMSI's suggestions are learned through a combination of reinforcement learning and chunking action sequences.

To the best of our knowledge, PMSI is the first computational model that unifies standard theories of human problem solving, accounts of long-term memory, symbolic representations, and reinforcement learning with neural networks. There are four key components that, taken together, make PMSI unique in the literature:

1. By *few*, we mean substantially fewer problems than state-of-the-art general reinforcement learning models need.

- it takes as input a universal symbolic language (called Innerese) that can represent any problem and goal;
- it relies on a society of collaborating agents to select actions, based on learned perceptions, where the agents are hierarchically structured by their different hypothesis spaces,[2] which are created by grouping learned macro-operators (a compiled sequence of operators);
- it uses methods called *B-brains* that detect when an agent is in trouble and can switch to another one; and
- it acquires effective expertise about problem solving from only a few training problems.

We present experiments that show PMSI can learn how to solve unseen problems in three domains (natural language queries, equation solving, and robot planning) with 20 practice problems each; the system is successful even when it must learn in all of these domains at once. A typical reinforcement learning approach with a single deep Q-network, or DQN (Mnih et al., 2015), fails in such a data-starved training environment. We demonstrate that PMSI produces human-like behavior, such as functional fixedness (Duncker, 1945) and capture errors (Reason, 1990). We close by discussing theoretical motivations for PMSI from related work in cognitive psychology.

## 2. Background

PMSI uses a symbolic language to interpret problems and calls on a problem solver to address them. Both the symbolic language and problem solver, which have been implemented in the Genesis Story Understanding System (Winston, 2012), are detailed here. We also briefly describe some of our main theoretical tenets, which are useful to know before reading the next section. Finally, we solidify our abstract conversation by providing a worked example of how we want PMSI to behave with a specific problem.

### 2.1 Innerese

PMSI takes a special symbolic language as input, which is called *Innerese* (or inner language) because it resembles symbolic logic from linguistic theories of semantics. This paradigm, such as Fodor's (1975) Language of Thought Hypothesis, suggests that humans parse natural language into logical expressions, which form a universal language of thought. Some hypotheses state that people have several such languages (e.g. Sloman, 1978). Regardless, it is generally agreed that languages of thought can be interpreted and manipulated by a problem-solving apparatus and that they have compositional semantics, as we can construct meanings by combining other meanings. Our Innerese model has these properties.

The Genesis group uses the START natural language system (Katz, 1997) to parse English into Innerese representations. START analyzes natural language by breaking it into sequences of units called *ternary expressions* that have the form **<subject relation object>**. Ternary expressions can be hierarchically nested in other ternary expressions. As an example, we show an Innerese parse in Expression 1, of the English sentence "Trump shocked Melania with his question." This parse is a sequence of two ternary expressions.

2. In this paper, we use the term *hypothesis space* to mean the set of actions that are available to a problem-solving agent, either during training or evaluation.

**<<Trump shocked Melania> with question>**, **<question related-to Trump>** (1)

Human inner language need not come from natural language. In principle, it can come from any input that a person might access. Likewise, Innerese need not come from a START parse. For example, one can construct a parser that takes in the state of the world from a robot's perception and represents it as Innerese. An example of an Innerese description of a physical environment could be **<wrench on-top-of table>**.

**Innerese Assumes Low-Level Agents** PMSI is an architecture that deals with problem solving at the level of general symbolic thought (Innerese), which is a high-level part of human cognition. We assume that there are domain-specific agents that can communicate perception information to PMSI. Creating perception systems that can observe a problem in a natural form and robustly generate sensible Innerese is a significant challenge that goes beyond the purpose of this paper. In the view of PMSI, these agents represent operators that return Innerese.

It is apparent that humans must also use domain-specific agents to retrieve general thoughts about many problems. We can access the sensor-invariant information that someone is walking nearby using our eyes, our ears, or both organs, but we cannot see things with our ears and we cannot hear them with our eyes.

### 2.2 The Self-Aware Problem Solver

Winston's (2017) Self-Aware Problem Solver has access to a variety of operators that can manipulate the state of a problem and check goal conditions. All of its actions return Innerese expressions about how they updated the state of a problem, which means that it solves a problem in a way that generates an Innerese explanation. Because it can also interpret problems represented in Innerese, the problem solver can solve problems about its explanations, which generates other Innerese explanations. The reason why "Self-Aware" is part of its name is because it can recursively solve problems about how it solved problems; in this sense, it can reason about its own reasoning.

We believe that *self awareness* is a suitcase term (Winston, 2017), i.e., it describes a cognitive concept so abstract that many different meanings can fit inside. We will not attempt to argue that the Self-Aware Problem Solver implements all aspects of this broad term. We only mean that it contains a key component of our interpretation of *self aware*, which is the ability of a system to recursively solve problems about how it solved problems.

PMSI and the Self-Aware Problem Solver can combine to create a human-like problem-solving system. The Self-Aware Problem Solver does not know how to best apply its actions, given the state of a problem and goal, and PMSI can learn to fulfill this function. Due to the problem solver's ability to apply its actions to problems regarding its solutions to other problems, PMSI is also able to learn a kind of *meta-level* intuition about how to solve problems that concern how it solved other problems. These meta-level aspects are not explored in this paper, but we consider this possibility important to mention, as it is a next step for our work. For now, we only use the Self-Aware Problem Solver as a resource through which PMSI can apply operators and receive Innerese responses about non-meta-level problems.

### 2.3 Theoretical Tenets from K-lines, B-brains, and Near-Miss Learning

Minsky's K-line Theory states that a human is able to remember the set of mental agents that were active when they were solving a problem, so that they can be primed with all of those agents when they encounter a similar problem. Such collections are called K-lines; they induce a learned subset of a mind's resources, known by Minsky as a Partial Mental State, from which PMSI gets its name. K-lines can contain other K-lines, and more generic K-lines are composed of more specific K-lines. A Level Band is the region of the mind with K-lines at the best-suited level of abstraction for a problem, so that the K-lines do not overfit or underfit on the problem at hand.

Minsky's theory predicts that human problem solvers should sometimes get stuck in K-lines outside the Level Band that are too specific and do not cluster enough agents. Alternatively, a K-line might contain too many agents, so the analysis of the problem is too shallow. The latter is indeed observed as a common type of human error (Reason, 1990), but humans are usually able to recognize that they are stuck and recover.[3] Minsky addresses the ability to recover from capture errors with the idea of B-brains, which can suggest another part of the mind to replace a failing part or offer some deliberation to guide the mind in a good direction. Sometimes a human can take more actions than necessary to solve a problem, but still achieve success, through a generally applicable K-line that they have learned; this is observed as functional fixedness (Duncker, 1945).

Finally, we consider Winston's (1970) Theory of Near-Miss Learning. This idea explains that the ability of humans to learn useful concepts from few examples stems from the ability to identify attempted solutions that are almost correct and learn the small corrections. So, to be a good model of human cognition, PMSI should be able to learn general solution paths that are almost correct for many of the problems, and it should require few training examples. As our experiments show, our implementation is able to achieve this goal by using ideas from the K-line and B-brain theories.

### 2.4 Worked PMSI Example

To clarify our research goal, we trace the desired behavior of PMSI on a problem that it solves in one of our experiments. Suppose that the Self-Aware Problem Solver has access to six operators, which will presumably only be useful if the problem involves solving equations. The names are self-explanatory, and exact specifications for how the actions work are not necessary for understanding this example.

- *add like terms*
- *simplify fractions*
- *simplify products*
- *apply quadratic formula*
- *divide both sides by relevant multiple*
- *check if solution is explicit*

The *simplify products* operator only works if fractions are simplified. Similarly, *apply quadratic formula* only works if an equation is in quadratic form, and *divide both sides by relevant multiple*

3. Humans cannot always recover; an example is learning phonetic categories (Goto, 1971).

only works if an equation is sufficiently simplified. Suppose now that we have a problem and goal, given in English by Expression 2.

$$x \text{ is a variable. } -5 \cdot x \cdot x \cdot 0.7 - x + 4 \cdot 2 - x^2 = 3 \cdot x \text{ is an equation. Solve the equation.} \quad (2)$$

Because this is an English representation, START can parse it into Innerese. A parse for the problem and goal is given in Expression 3; in this case, the problem is on the first line and the goal is on the second.

$$\begin{aligned} &\textbf{<class variable } x\textbf{>}, \textbf{<class equation } -5 \cdot x \cdot x \cdot 0.7 - x + 4 \cdot 2 - x^2 = 3 \cdot x\textbf{>} \\ &\textbf{<I solve equation>} \end{aligned} \quad (3)$$

PMSI should propose a relevant operator or sequence of them that the Self-Aware Problem Solver can use to get closer to a solution. In this case, *simplify products* would be a good suggestion because there are products that can be multiplied, but *simplify fractions* would not be useful because there are no fractions in this equation. If PMSI suggests *simplify products*, then the parse for the modified problem would be:

$$\begin{aligned} &\textbf{<class variable } x\textbf{>}, \textbf{<class equation } -3.5 \cdot x^2 - x + 8 - x^2 = 3 \cdot x\textbf{>} \\ &\textbf{<I solve equation>} \end{aligned} \quad (4)$$

For Expression 4, it would be useful for PMSI to suggest *add like terms*, which would put the equation into a form appropriate for *apply quadratic formula*. A component of PMSI might have learned to chunk these two actions into a macro-operator and remove the original operators from consideration, to reduce its hypothesis space. PMSI might not know that *add like terms* will definitely enable *apply quadratic formula* to work, but humans generally do not know if their hunches will work either. After suggesting this macro-operator, the Innerese would be:

$$\begin{aligned} &\textbf{<class variable } x\textbf{>}, \textbf{<class equation } x = -1.85\textbf{>}, \textbf{<class equation } x = 0.96\textbf{>} \\ &\textbf{<I solve equations>} \end{aligned} \quad (5)$$

It would then be useful for PMSI to suggest *check if solution is explicit*, which is a special operator because it decides whether the goal is met and can tell the system to stop.

This simple problem is a good illustration of how we want PMSI to work. Innerese is our model for the language of human thought, and so PMSI is our model for how humans can gain and use intuition for any problem they can think about.

## 3. PMSI Implementation

We have implemented PMSI in a reinforcement learning framework. The state is given by Innerese for a problem and goal, the actions are macro-operators from the Self-Aware Problem Solver's actions, and the reward is the number of primitive operators needed to reach a solution, with fewer steps counting as more desirable.

### 3.1 Architecture

PMSI consists of a hierarchy of deep Q-networks, or DQNs (Mnih et al., 2015), that work together to suggest insightful actions the Self-Aware Problem Solver should use. By *hierarchy*, we mean that there is a stack of DQNs where the first DQN initially has control, but it can hand off control to a DQN below it, and so on. A DQN is a neural network that outputs Q-value estimates for each action that an agent can take; a Q-value is a score for an action given the current state of a problem.

All of PMSI's DQNs have an architecture that takes Innerese for the current state of a problem and goal as input. However, each DQN returns Q-values for a different set of actions, that is, K-lines, as they are a realization of Minsky's idea of the same name (1980). DQNs in PMSI are arranged by increasingly smaller hypothesis spaces that are made of increasingly larger operator chunks. Interactions between the DQNs are regulated by modules called B-brains, which are also a realization of Minsky's idea of the same name (1980). DQNs with the largest and fewest K-lines are tried first, and a B-brain may cause a coarse DQN to yield control to a finer DQN if none of the chunked suggestions seem helpful. During training, DQNs with existing K-lines spark the development of new DQNs with more aggregated K-lines, as discussed in Section 3.2.1.

#### *3.1.1 K-line Realization*

For our purposes, K-lines are a way to reduce the hypothesis space of a problem-solving system in a way that generalizes. If a system has access to a set of 20 operators, but it can learn that only three different operator sequences from that set can solve all of its training problems, then it can essentially reduce its hypothesis space to three.

Per this idea, our definition of a K-line in the context of PMSI is a learned macro-operator that captures some amount of general applicability. The fundamental idea of macro-operators is not new, but we choose to brand these as K-lines because of how they are learned and used. K-lines are learned by combining other K-lines and are used with perceptual guidance from DQNs. A search through the K-lines (and corresponding DQNs) can find Level Bands of abstraction.

#### *3.1.2 DQN Design*

Each DQN in PMSI is made of an encoder and a Q-value approximator. The encoder takes in a sequence of Innerese for the problem and goal, and it produces a feature vector encoding. The Q-value approximator then uses this encoding to estimate Q-values for the DQN's K-lines.

Before entering the neural portion of the encoder, a sequence of Innerese expressions is turned into tokens according to the Tokenizer module in Table 1. We use a dictionary of pre-trained word embeddings to turn these tokens into vectors. The dictionary is also used in the Tokenizer itself.[4] The Tokenizer treats special Innerese characters such as "<" as words and phrase structure hyphens as spaces between words. The Tokenizer essentially translates Innerese to tokens word-for-word, unless an Innerese word is not in our dictionary of embeddings. If it is not, then the module assumes that the word is a special entity that deserves to be analyzed at the character level and treats its component characters as words (an example of such a special word that will not be found in our

4. Pennington et al. (2014) provide an explanation of word embeddings.

*Table 1.* The Tokenizer module, which converts Innerese to tokens.

```
1:  tokens ← []
2:  for Innerese expression in Innerese sequence do
3:      for word in Innerese expression do
4:          if word in embeddings then
5:              tokens.append(word)
6:          else
7:              for character in word do
8:                  tokens.append(character)
9:      tokens.append(special expression ending character)
10: return tokens
```

embeddings is a math equation). We assume that our set of embeddings contains every character and many English words, as Innerese uses English words to represent meanings.

The tokens are then translated into embeddings by the embedding layer, which uses 50-dimensional pre-trained GloVe embeddings (Pennington et al., 2014). Because Innerese is hierarchically structured, we hope to capture hierarchical dependencies by feeding the sequence of embeddings into a bidirectional Long Short-Term Memory Cell, or LSTM (Hochreiter & Schmidhuber, 1997). A bidirectional LSTM takes input in the forward direction, then the backward direction, and concatenates the outputs from both directions. We then apply a Max Pool across all of the hidden states produced by the LSTM to get our vector encoding. It has been shown that pooling over all LSTM hidden states can construct a better encoding of a problem from natural language than the final hidden state for the purpose of answer selection (Lei et al., 2016). An example of the pooling procedure is

$$\begin{left}\begin{aligned} h_0 &= [1 \quad 2 \quad 0 \quad -4 \quad 5 \quad 2 \quad 6 \quad -1] \\ h_1 &= [5 \quad 9 \quad 1 \quad 6 \quad -3 \quad 0 \quad -5 \quad 0] \\ h_2 &= [0 \quad 7 \quad 4 \quad -1 \quad -1 \quad 3 \quad 6 \quad 4] \end{aligned}\right\} \text{Max Pool across this dimension} \\ \overbrace{\qquad\qquad}^{\text{LSTM forward}} \quad \overbrace{\qquad\qquad}^{\text{LSTM backward}} \\ \implies enc = [5 \quad 9 \quad 4 \quad 6 \quad 5 \quad 3 \quad 6 \quad 4] \end{left} \tag{6}$$

where $h_i$ are hidden states of the bidirectional LSTM and $enc$ is the resulting encoding.

The red component in Figure 1 illustrates a neural encoder. This encoder is applied once on the Innerese sequence for the problem and again (with the same parameters) on the Innerese sequence for the goal. The final encoding is the concatenation of these two encodings, which is then passed into two fully connected layers with a ReLU activation (Nair & Hinton, 2010) in between. These layers serve to take the encoding for the state and estimate a Q-value for each of the DQN's K-lines, as shown by the blue component in Figure 1.

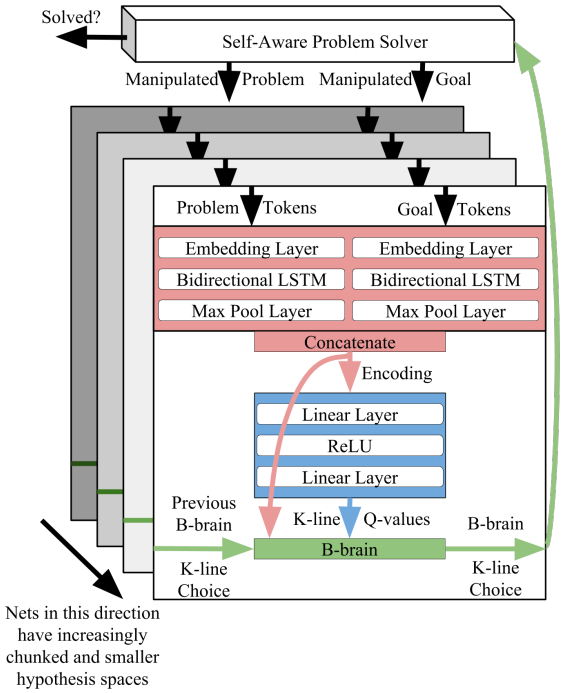


*Figure 1.* The PMSI architecture, which consists of a stack of DQNs. Each has an encoder (red), Q-value approximator (blue), and B-brain (green). The DQNs work together to suggest K-lines to the Self-Aware Problem Solver. (The Innerese parser and Tokenizer are not shown.)

### *3.1.3 DQN Hierarchy, B-brain Realization, and Finding a Level Band*

PSMI is composed of several DQNs that are arranged from largest hypothesis space (where K-lines correspond to single operators) to smallest hypothesis space (where there are fewer K-lines that are composed of substantially chunked action sequences). Initially, PMSI will use the top-level DQN (smallest hypothesis space) to suggest K-lines to the Self-Aware Problem Solver, but a B-brain may choose to switch DQNs. Each one has an associated B-brain, which is designed to detect if that DQN gets stuck and thus recover the problem-solving process. Our implementation of a B-brain can help in two ways:

1. **B-brain Prevent**
   When a DQN suggests a particular K-line, the B-brain remembers that K-line until that DQN's Innerese encoding changes. If the DQN tries to suggest the same K-line before this happens, the B-brain will force the DQN to select the K-line with the highest Q-value that has not been tried since the last time the encoding changed (assuming there is one). The B-brain does this because it is a mistake for the DQN to suggest the same K-lines over and over if they do not alter the perceived state of the problem. Because the B-brain looks at the DQN's encoding of the problem instead of the actual Innerese, the idea should still work as a good heuristic if the

DQN is made recurrent to better handle partially observable Markov decision processes, as we plan to explore in our future work.

2. **B-brain Yield**
   Because the top-level DQNs use macro-operators and do not consider the component operators individually, it is possible that they will encounter a problem that they cannot solve, but that a lower DQN can solve. For example, if the Self-Aware Problem Solver has access to several actions for robot planning, such as "pick" and "place", but a top-level DQN has a hypothesis space that chunks "pick, place" as a sequence, then it cannot actually command the robot to pick anything up and will need to hand off control to a lower-level DQN. This is a particularly bad scenario, and we expect these hand-offs to be rare in practice if the K-lines are chunked in an insightful way. Regardless, this issue is addressed by the B-brain. Each B-brain has a manually specified hyperparameter, $n$, such that if its DQN suggests enough K-lines to add up to $n$ or more single operators, and the problem is still not solved, the B-brain will give up control to the DQN and its B-brain at the next lower level. *B-brain Yield* can be seen as a way to recover from capture errors (Reason, 1990).

The overall goal of the B-brains is to find what Minsky referred to as Level Bands (1980). The Level Band is an area of the mind that contains the appropriate mental agents for a given problem. If PMSI uses a DQN that has a hypothesis space that is too large, then it risks entering a search space that is not feasible for the amount of training that the DQN has had. A DQN in this region may have overfit on the training set and "hallucinate" about its training problems, entering infinite loops of action choices on test problems. If PMSI uses a DQN that has a hypothesis space that is too small, then it risks causing capture errors. Figure 1 shows the B-brain and DQN arrangement.

### 3.2 Learning Procedure

PMSI learns in several stages and the number is adjustable via a user-controlled hyperparameter $s$. The system starts with a single DQN, with Q-value outputs that correspond to the Self-Aware Problem Solver's operators. PMSI is connected to the Self-Aware Problem Solver and given a few training problems. This system iterates through several epochs of these training problems, treating each one as a reinforcement learning task that can be solved with a path of K-lines. During training, it turns off its B-brains (to not disturb Q-value learning) and uses only the top-level DQN for action selection. Once the DQN from a given stage has completed training, PMSI switches into an evaluation mode that enables it to use its full problem-solving capabilities and runs a module called the K-line Compiler.[5] The K-line Compiler can use PMSI's solutions to the training problems to insightfully chunk actions into K-lines that represent a smaller hypothesis space than that of the current top-level DQN. A new one is initialized with this hypothesis space, which is then added to the stack as the new top-level DQN. The next stage of learning then commences to train the new network, and so on.

During training, PMSI cannot use B-brains to find a good Level Band and prevent itself from getting stuck in capture errors. To compensate, there is a user-specified hyperparameter, $f$, such that,

5. In our experiments, we found that it was not too damaging to simplify this procedure and just give the compiler access to the top-level DQN's capabilities.

if a DQN takes more than $f$ steps and it still cannot solve a problem, then the problem is removed from the training set while that DQN is learning and corresponding backpropagation updates to the network are undone.

To learn a good policy during the training mode, a DQN is given a probability of picking a random action that starts out as the hyperparameter $\epsilon_{start}$ and converges to $\epsilon_{end}$. The probability is

$$\epsilon_{end} + (\epsilon_{start} - \epsilon_{end})e^{-\epsilon_{current}/\epsilon_{decay}}, \tag{7}$$

where $\epsilon_{current}$ is the current epoch and $\epsilon_{decay}$ is a parameter that specifies the speed of decay. If a random action is not taken, the K-line corresponding to the DQN's highest Q-value is taken.

Our optimization procedure uses a target network as well as a policy network, as this has been shown to stabilize the learning process (Mnih et al., 2015). The target network has an associated hyperparameter that specifies how often it should be updated with the weights of the policy network. The loss $\mathcal{L}$ is calculated as

$$\delta = Q^p(s, a) - (r - \gamma \max_{a'} Q^t(s', a')) \tag{8}$$

$$\mathcal{L}(\delta) = \begin{cases} \frac{1}{2}\delta^2 & \text{for } |\delta| \leq 1, \\ |\delta| - \frac{1}{2} & \text{otherwise.} \end{cases} \tag{9}$$

$$\mathcal{L} = \frac{1}{|B|} \sum_{(s,a,s',r) \in B} \mathcal{L}(\delta) \tag{10}$$

In Expression 8, $Q^p$ is the Q-value estimate function from our policy network, $Q^t$ is the Q-value estimate function from our target network, $s$ is the state before taking action $a$, $r$ is the reward from taking action $a$, $\gamma$ is the discount factor, and $s'$ is the state after taking action $a$. The maximization produces the Q-value estimate for the next step, assuming that the agent will take the best action, $a'$. $\delta$ is known as the temporal difference error and, per the Bellman equation for reinforcement learning, its magnitude should be minimized (Mnih et al., 2015). In our case, the reward, $r$, is $-lc$ if the K-line did not solve the problem and $p - lc$ if the K-line did solve the problem, where $c$ and $p$ are positive hyperparameters and $l$ is the number of operators comprising the K-line. This reward function encourages PMSI to use the fewest operators to solve a problem.

In Expression 9, $\mathcal{L}(\delta)$ is the Huber (1964) loss, which can minimize the magnitude of the temporal difference error while keeping large values from having a huge impact on the optimization. Expression 10 denotes the final loss that we hope to minimize, where $B$ represents a batch of transitions (state, action, next state, reward) that are sampled from memory replay. We use memory replay (which stores a history of transitions that the DQN generates) because it can stabilize the learning procedure (Mnih et al., 2015). Finally, we use the Adam optimizer (Kingma & Ba, 2015) to minimize $\mathcal{L}$.

Before we conclude this section, we would like to emphasize that a small number of training examples does not mean that PMSI takes a short amount of time to train. However, compared to other neural net systems, PMSI is fairly fast. Our experiments used a laptop GPU, on which it took roughly an hour to train on individual domains and several hours to train on all domains at once.

*Table 2.* The K-line Compiler module, which creates a reduced hypothesis space in a helpful way.

| | |
|---|---|
| 1: | *subpaths* ← *getCommonSubpaths*(*solution paths*) |
| 2: | *subpaths* ← *filterSubpaths*(*subpaths*) |
| 3: | *subpaths* ← *topEDiverseSubpaths*(*subpaths*) |
| 4: | *new K-lines* ← *subpaths* ∪ *existing K-lines* |
| 5: | *new K-lines* ← *filterNewKlines*(*new K-lines*) |
| 6: | **return** *new K-lines* |

### *3.2.1 K-line Compiler*

The goal of the K-line Compiler is to take a list of PMSI's solutions to the training problems at the end of a stage and use them to generate the K-lines for the next stage's top-level DQN. The compiler should create a further reduction in hypothesis space size (or at least a space of the same dimensionality) compared to that of the previous top-level DQN and produce K-lines that have general applicability. Table 2 provides pseudocode for the K-line Compiler module, which has five main subroutines:

- *solution paths* is a list of PMSI's solutions (as a sequence of single operators) for each of the training problems;
- *getCommonSubpaths* returns a set of longest shared operator sequences, excluding any that are the same as already known K-lines, from all possible path pairings (with the exception of the same path paired with itself);
- *filterSubpaths* removes subpaths that are superpaths of other subpaths in the collection. This routine helps PMSI to avoid creating unnecessarily long K-lines;
- *topEDiverseSubpaths* returns the top $e$ (or less, out of necessity) most frequently occurring subpaths, where frequency is measured as the number of solutions in which the subpath occurs and $e$ is a hyperparameter. It skips over subpaths if they do not contain at least one of the current top-level DQN's K-lines as a strict subsequence, such that the K-line is not contained in an already selected subpath. By selecting for subpaths that are very frequent (and sufficiently diverse) across many solutions, it should capture those that are general enough to be useful;
- *new K-lines* is momentarily the union of the current top-level DQN's K-lines with the output of *topEDiverseSubpaths*, which is then filtered such that K-lines that are subsequences of other K-lines are removed.

Because all of the *new K-lines* from *topEDiverseSubpaths* each have at least one different K-line from the current top-level DQN as a strict subsequence, *filterNewKlines* will always return a number of K-lines which is less than or equal to the number in the current top-level DQN. This satisfies our requirement that the DQNs in PMSI be arranged from the least chunked and largest hypothesis space to the most chunked and smallest one.

Also note that the K-line compiler can use human-provided solutions to generate another DQN, instead of relying on PMSI's solutions. Human solutions could be better than PMSI's at some stages and so could enhance the learning procedure.

## 4. Experiments and Discussion

In this section, we report tests of PMSI in three different problem domains. We present the results of training on each domain individually (with only the relevant operators) and also training on all domains at once (with all of the operators). These tests show that our unique Innerese-interpreting DQN architecture is successful in the three domains and also that the hierarchical DQN arrangement in PMSI is useful, because a single DQN, with Q-values corresponding to the operators, fails in our data-starved training environment. We use the lowest-level full hypothesis space DQN of PMSI as the comparison DQN; we refer to this as the single DQN.

For each of the domains, we constructed 50 problems, all of which are solvable by some combination of operators from the Self-Aware Problem Solver. Each group was randomly divided into a training, development, and test set of 20, 15, and 15 problems, respectively. The tests use many of the same hyperparameters,[6] although some hyperparameters were tuned on the development set. The number of learning stages, $s$, is tuned, along with the K-line compile cutoff, $e$, for each stage. The values for each B-brain yield threshold, $n$, are also tuned. We discuss the necessity of tuning these parameters in Section 4.5.

The development set is also used to determine if some amount of random sampling via a Softmax over Q-value outputs is necessary for agents to solve unseen problems. Neural agents may get stuck in loops where they keep suggesting the same action(s); this issue may be caused by a lack of training data, and we show that this is a serious problem for the single DQN. One way to prevent loops is by choosing actions according to a sample from a Softmax (Bishop, 2006) over the Q-values, instead of always taking the action with the highest Q-value (in PMSI *B-brain Prevent* may limit the action candidates, but the same idea holds). The Softmax is

$$P(a_j) = \frac{e^{a_j}}{\sum_{k=1}^{K} e^{a_k}} \text{ for } j = 1, ..., K, \tag{11}$$

where there are $K$ Q-values and $a_j$ is a Q-value. Note that, although we perform some tuning of hyperparameters, we do not spend significant effort to give PMSI small percentage improvements. In all of our experiments, both PMSI and the single DQN achieve near perfect performance on the training set, so it is indeed the case that we give each of them an environment in which they learn well; the single DQN just cannot learn a policy that generalizes.

The remainder of the section analyzes and discusses PMSI's performance, with our numerical results shown in Figure 2 and Table 3. The latter reports the fraction of development set problems that PSMI and the single DQN failed to solve in each domain (without using Softmax sampling), as well as when presented with all four domains in one training run. The table also displays the mean and variance of the solution length on the test set, with systems that use Softmax sampling added as deemed necessary from the development set. For more information, we provide a GitHub link that has all of the training, testing, and development problems and goals, as well as the hypothesis spaces that our test PMSIs learn.[7]

6. Unless otherwise specified, the hyperparameters were: Bidirectional LSTM hidden layer size $= 20$, Q-value approximator hidden layer size $= 20$, $\gamma = 0.99$, $c = 0.1$, $p = 0.6$, $\epsilon_{start} = 0.9$, $\epsilon_{end} = 0.1$, $\epsilon_{decay} = 30$, batch size $= 20$, memory replay size $= 300$, target net update frequency $=$ every 14 problems solved, $f = 1000$, epochs $= 30$.
7. See `https://github.com/TristanThrush/pmsi_problems`.

### 4.1 Natural Language Queries

In our natural language query domain, a goal has the form "verify that x is before y", and a problem is in the form of a story in English, as in the example

Adam got his first ticket because he was speeding. He had lunch after he got his first ticket.
Verify that "Adam was speeding" is before "Adam had lunch." (12)

Problems and goals in the domain are converted into Innerese by the START parser (Katz, 1997). This experiment includes six operators:

- *years to after*
- *after to before*
- *because to before*
- *in-back-of to in-front-of*
- *in-front-of to before*
- *chain "before" relations to verify the goal*

The "g to h" actions insert entailed h relations from g relations. For instance, *years to after* means that, if the problem has "WW1 ended in 1918" and "WW2 ended in 1945", then "WW2 ended after WW1 ended" would be added.

We chose $s = 4$ (meaning that four DQNs were constructed), with $e$ (K-line compile threshold) values of 3, 2, and 1. Furthermore, PMSI did not need Softmax sampling, as it could solve all of the development problems without becoming stuck. This contrasts with the single full-hypothesis-space DQN, which could not solve five out of 15 of the development problems without sampling. Consequently, we used the single DQN with Softmax sampling on the test set to give it better performance. We also found that on the development set PMSI only needed the top DQN to effectively solve every problem, so this DQN's B-brain yield threshold, $n$, was set to $\infty$.

The K-lines of this PMSI were chunked by the K-line Compiler as action sequences that were generally applicable. In the top DQN, all of the operators dealing with the spatial interpretation of "before" were chunked into one K-line (with Action 6 at the end), and all of the other actions were chunked into two K-lines (also with Action 6 at the end). The result was that this DQN could solve any problem in the test set with either one or two K-lines. In this domain, we observed a very chunked hypothesis space to be extremely useful on its own, without much, or any, need for the B-brain actions. This is likely because actions cannot set the problem back; if incorrect actions are chosen, they just do nothing, or generate "before", "in-front-of", or "after" relations that are irrelevant for satisfying the specific goal.

### 4.2 Equation Solving

In the equation-solving domain, a goal takes the form of "solve the equation(s)", and a problem has the form of English statements that designate the equation(s) and variable of interest. All of the problems involve first or second order polynomial equations. This is the same problem domain from the worked example in Section 2.4. We again found that on the development set PMSI needed only the top DQN to effectively solve every problem, so this DQN's B-brain yield threshold, $n$, was

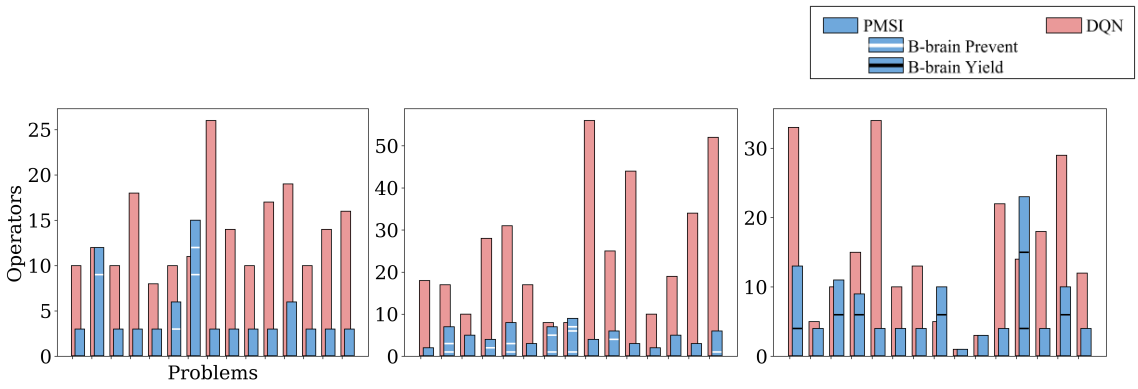


*Figure 2.* The length of solutions found by PMSI and by the single on each of 15 test problems, for the three domains. Plots (a), (b), and (c) correspond to language queries, equation solving, and robot planning, respectively. For brevity, we have not included plots for training the systems on all domains at once.

set to $\infty$. In this case, we set $s = 2$, with an $e$ value of 3. PMSI did not need Softmax sampling, although our single DQN still did; without sampling it failed to solve six out of 15 problems on the development set.

PMSI's K-lines were generally applicable in this scenario as well. In the top DQN, one K-line contained most of the generic equation-simplifying operators, another handled the specifics of solving simplified first order equations, and another handled the specifics of solving simplified second order equations. In this domain, actions also cannot set the problem back; each action will either simplify a problem more or do nothing. In a way, this domain is easier than the natural language domain, because an action here cannot produce a useless addition to the problem. We therefore expected both a very chunked hypothesis space and *B-brain Prevent* to be useful, which is what we observed.

### 4.3 Robot Planning

Our third experiment examined a robot planning domain, in which a high-level simulator returns object locations without the need for actual perception modules. A goal has the form of "ensure that x is on top of y", and a problem takes the form of the robot's view of the world, which includes the location of objects relative to each other. We used START to translate the natural form of goals (English) into Innerese. For the problems, we had to program another parser that translates a robot's simulated perception outputs into Innerese. This experiment used four operators:

- *attempt to place x on y*
- *unstack item that is above x, to reach x*
- *unstack item that is above y, to make room for x*
- *update Innerese representation of the world from simulated perception*

*Table 3.* Empirical results for PMSI vs. a single DQN that encodes the entire hypothesis space. The single DQN fails to solve a greater fraction of development set problems than PMSI and, for the tasks it solves, has substantially higher means and variances for lengths of test set solutions. 'Random' denotes an agent that has access to all of the operators in a single DQN, but that chooses them at random from a uniform distribution.

| | | Language Queries | Equation Solving | Robot Planning | All Domains at Once |
|---|---|---|---|---|---|
| No. Unsolved Dev. Set Problems (no sampling) | PMSI | 0/15 | 0/15 | 1/15 | 6/45 |
| | DQN | 5/15 | 6/15 | 7/15 | 27/45 |
| Mean of Solution Length | PMSI | 4.80 | 4.93 | 7.20 | 28.32 |
| | DQN | 13.67 | 25.13 | 14.93 | 73.74 |
| | Random | 15.73 | 24.07 | 36.20 | 67.44 |
| Variance of Solution Length | PMSI | 13.9 | 5.46 | 30.63 | 8571.55 |
| | DQN | 22.69 | 228.85 | 103.87 | 22507.68 |
| | Random | 213.33 | 180.80 | 5222.09 | 6393.15 |

The effects of these operators are conditional. For instance, if there is an object above x, the *place* operator fails. If there is an object above y and y can only hold one object, *place* counter-productively moves x on top of that object; x will then need to be unstacked. *place* also reports whether the goal is met. The *unstacking* operations only unstack one item at a time, so several such operations may be necessary.

Note that this domain differs from those in the other experiments because the hypothesis spaces of DQNs near the top level of PMSI may be incapable of solving certain problems (Action 1 can be destructive), so the *B-brain Yield* capability may be essential.

In this case, we set $s = 4$, with $e$ values of 3, 2, and 1. We decided to provide Softmax sampling for tests of PMSI, because it got stuck on one of the 15 development problems. Our single full-hypothesis-space DQN was still much worse; it failed to solve seven out of 15 problems in the development set, so we used Softmax sampling on tests for this agent as well. This time, we set PMSI's B-brain yield thresholds to $\infty, 20, 10$, and 1 from the lowest to top DQN.

In general, the generated K-lines were chunks of many unstacking operations, sometimes with the *place* action as the final move. This makes sense, because it is important for PMSI to clear the appropriate surfaces, before trying to move the relevant object to the relevant place. *place* can otherwise set the problem back by moving an object to an undesired location.

### 4.4 Multiple Domains at Once

In the tests above, we only gave PMSI operators relevant to the domain that in which we tested it. However, in many situations, nobody tells a human which subset of their many operators is relevant to solving a domain of problems. The notion of a domain is a bit fuzzy anyway; for example, we could have separated our mathematical equation domain into two domains, one for first-order polynomials and another for second-order polynomials.

For this reason, we ran another experiment that placed the system under more stress. We trained PMSI on all three domains at once. In this case, we made $s = 6, e = 9, 6, 4, 2, 1$ and $n = \infty, 1, 1, 1, 1, 60$ (where the $e$ and $n$ values are from lowest to top DQN). We also changed the memory replay size to 900, batch size to 60, both hidden layer parameters to 60, and $f$ to 150. Additionally, we added Softmax sampling to the four lowest DQNs (including the single lowest DQN used for comparison). The result of this training was that the learned K-lines mostly contained actions for one of the domains and were generally applicable to many problems in that domain.

Even though the multi-domain PMSI is still an improvement over the single DQN, it is doubtful that it offers a good model for all human intuition. Early in the training, the system tried to perform actions such as picking up equations and combining like terms of physical objects. It seems strange for a human to even think about trying such actions as they learn, which indicates that a better model for human intuition might include commonsense reasoning and clustering at a higher level than PMSI. These high-level procedures would try to extract subsets of operators that are helpful for specific domains. Then, several PMSIs could be trained with domain-specific operators, just as in our first three experiments.

### 4.5 Tuning the Partial Mental State Parameters

We found that our system can be sensitive to the tuning of $s$, $e$, and $n$. This is expected, because these parameters play a large role in the partial mental states that are developed and how they are used. If PMSI constructs too many DQNs, then a Level Band would take a while to find, and if it constructs too few DQNs, then a good Level Band might not be found at all. If $e$ is not suitable, then a good Level Band might not be constructed regardless of the number of DQNs ($e$ can adjust the size and chunking of DQN hypothesis spaces). These parameters are especially important in domains where falling back on multiple DQNs is a necessity, as in robot planning.

To examine PMSI's sensitivity to these parameter values, we ran another study in which we varied the setting for each while holding the other two constant. We used Softmax sampling in all of the tests to reduce the effects of an occasional outlier problem that cannot be solved otherwise. In a few cases, parameters depend on each other, so fixing two was not possible. Table 4 presents development set results in each of the individual domains. The table shows that perturbations in the partial mental state parameters can lead to considerable performance differences. PSMI's score in robot planning can drop almost to that of a random agent.

## 5. Intellectual Precursors to PMSI

As we have asserted throughout this paper, PMSI models how a person learns and uses intuition when engaged in problem solving within a domain. We have already discussed our primary sources of theoretical inspiration (K-lines and Near-Miss Learning), but these concepts are part of a particular school of thought that is tied very closely into the AI and cognitive systems literature. In this section, we provide additional theoretical motivation for PMSI from a cognitive and learning psychology background. We suggest that it can be seen, in part, as an integration of the Gestalt notion of restructuring with problem-space search, which is a more typical theory of problem solving in the information-processing community. We also examine motivation for PMSI from a classic study of a human that learns from experience and work on a general theory of human problem solving.

*Table 4.* Development set solution length averages and variances (in number of single operators) for PMSI with different parameter choices, with $n$ and $e$ values arranged from lowest to highest DQN.

| | Language Queries | Equation Solving | Robot Planning |
|---|---|---|---|
| $s = 6, e = 5, 4, 3, 2, 1$ | Avg: 8.27<br>Var: 36.93<br>(Top $n = \infty$) | Avg: 10.8<br>Var: 22.36<br>(Top $n = \infty$) | Avg: 32.00<br>Var: 1435.27<br>$(n = \infty, 20, 10, 1, 1, 1)$ |
| $s = 4, e = 5, 4, 3$ | Avg: 11.8<br>Var: 173.16<br>(Top $n = \infty$) | Avg: 8.13<br>Var: 29.38<br>(Top $n = \infty$) | Avg: 27.67<br>Var: 1224.02<br>$(n = \infty, 20, 10, 1)$ |
| $s = 4, e = 3, 2, 1$ | Avg: 6.6<br>Var: 10.84<br>(Top $n = \infty$) | Avg: 9.47<br>Var: 14.05<br>(Top $n = \infty$) | Avg: 5.27<br>Var: 7.20<br>$(n = \infty, 20, 10, 1)$ |
| $s = 4, e = 3, 2, 1$ | Avg: 7.53<br>Var: 35.92<br>$(n = \infty, 12, 8, 4)$ | Avg: 9.73<br>Var: 40.40<br>$(n = \infty, 12, 8, 4)$ | Avg: 6.54<br>Var: 18.32<br>$(n = \infty, 12, 8, 4)$ |

### 5.1 Elements of Human Problem solving

Newell, Shaw, and Simon's (1958) paper on a theory of human problem solving reviews several concepts from cognitive psychology that have influenced PMSI. They postulate that problem solving involves several memories that contain symbolized information, that there are primitive operators which act on these memories, and that there are rules for combining such processes. Their paper described a computer system, the Logic Theorist, that could prove theorems in symbolic logic. The authors argued that the Logic Theorist exhibited several characteristics of human cognition, including preparatory set, directional set, utilization of hints, heuristics (or "insight") to reduce the search space, and hierarchies of process.

However, the authors indicate that the Logic Theorist does not show concept formation because it is mainly a performance system; the system could not learn how to perceive similarities in classes of problems. PMSI extends the Logic Theorist in this way, as there is some evidence that it can learn new concepts. We have seen in our experiments that PMSI constructs K-lines that can tackle general categories of problems (e.g., that can handle first-order and second-order polynomials). Furthermore, when trained on multiple domains, PMSI mostly creates K-lines that only contain actions within the same domain.

### 5.2 Restructuring

According to Ohlsson (1984a), restructuring is the process of changing a mental representation of a problem as a way to solve it. This theory is relevant to PMSI because Gestalt psychology views restructuring as a way to provide immediate intuition during problem solving. His account includes several principles that, together, constiture the (somewhat vaguely worded) theory of restructuring. These principles include:

1. Every situation contains important structural relations.
2. Problems are situations with structural gaps between them.
3. Restructuring closes structural gaps in a mental representation of a problem.
4. Restructuring is involuntary and just "happens", often suddenly and dramatically.
5. A restructuring event always makes progress toward a solution.
6. A restructuring event becomes more likely after a problem and goal have been analyzed more deeply and after several unsuccessful solution attempts.
7. Restructuring leads to perceiving a problem differently.

Some of these ideas appear to be too strongly stated. Among other criticisms, Ohlsson (1984a) noted that there is evidence to the contrary of Principle 5. Additionally, empirical tests of restructuring are limited because there is no known way to reliably trigger restructurings in test subjects, without also driving the solution in a specific direction.

Nevertheless, it is evident that restructuring does happen to some extent in human problem solving, and it is paramount in some cases. This idea is illustrated well with a geometry example that Ohlsson (1984b) provides. A person may discover that a geometry problem involving complicated overlapping shapes can be broken into simpler shapes, such as triangles. This realization might completely change the person's perception of the problem; the person may now search for a solution in a space involving theorems that are related to triangles instead of complicated shapes. Another restructuring event may build off the first one, and enable the person to piece together the triangles to form squares, so they no longer need triangle theorems.

The geometry example highlights an important point: humans use hybrid search and restructuring. People can search through a typical problem space using operators that change a problem's state, but they can also search through a description space (a space of restructurings), which changes their perception and applicable actions, but not a problem's state (Ohlsson, 1984b). The decision to trade off between search in the problem space and the description space is supposedly guided by meta heuristics in a way similar to the principles presented above.

In PMSI, the meta heuristics that provoke restructurings are B-brains. It is important to note that it is difficult to interpret restructuring as simply the creation and utilization of macro-operators alone (Ohlsson, 1984b). However, B-brains actually trigger the switching of neural networks that have been trained with different actions (albeit all actions are chunks from some basic set of operators), and therefore each have distinct perceptions of a problem. Additionally, B-brains can trigger the utilization of neural networks that break macro-operators into their component parts at many different levels of granularity.

### 5.3 Learning by Doing

Anzai and Simon (1979) present a careful analysis of mechanisms involved in a test subject's ability to learn how to improve at the Tower of Hanoi problem, which was novel to the subject before the testing started. Even though they study one subject and one task, the authors discover processes that they assert are generally useful for learning to solve problems.

There are four successive problem-solving episodes that the subject undergoes. The authors make these observations, some of which have analogs in PMSI:

1. Even in the first episode, the subject's initial behavior was not random; she did not form goals, but she knew to avoid action loops and successive movements of the same disk.
2. The subject clearly had a different strategy in the second episode. She guided herself by explicitly mentioning intermediate goals and avoided making mistakes from the first episode.
3. In the third episode, the subject formulated recursive subgoals (goals that are triggered as a way to satisfy larger goals) and began chunking moves together.
4. In the fourth episode, the subject began speaking of sets of disks and moving chunks of them. It was evident that her perception of the problem had changed to view collections of disks as pyramid-like units, instead of individually.

The use of Innerese is an analog to part of the subject's behavior, because both PMSI and the subject can, to some extent, explain themselves in natural language. Additionally, PMSI does not use random search either, even if it has not been trained. *B-brain Prevent*, although simple, can provide task-independent deliberation to prevent wasteful loops.

Finally, PMSI shares part of the subject's ability to chunk actions and build different strategies from experience. It seems that, at least in the fourth episode, the subject was able to learn an obvious perceptual shift: a restructuring of the problem. In the previous section, we mentioned that PMSI can restructure by shifting to lower DQNs. However, PMSI can also learn new restructurings by creating and training new DQNs with different hypothesis spaces.

## 6. Conclusion and Contributions

Our paper shows that the development and use of intuition in human problem solving from limited training data can be modeled by a fusion of high-level problem-solving theories and low-level pattern-matching techniques. Humans use inexplicable hunches to help solve problems in ways that can beat extensive searches. Although there is more work to be done, particularly on implementing commonsense procedures to help PMSI separate distinct domains before it starts training, PMSI can learn human-like intuition for general-purpose problem solving. Our model for a language of human thought and our realization of K-line Theory are important to PMSI's success in our three experimental domains (and combinations of them); they enable PMSI to learn like a human in a training environment that is too data starved for a typical DQN to work. Not only is PMSI quantifiably successful, but it also exhibits several human behaviors observed in cognitive psychology.

## Acknowledgements

We thank Dylan Holmes for his suggestions about PMSI itself, and our paper. We appreciate Pat Langley's detailed suggestions, Leslie Kaelbling's meeting about related work, the reviewers' comments, and Gary, Sharon, and Arlene Fraser's edits.

## References

Anzai, Y., & Simon, H. A. (1979). The theory of learning by doing. *Psychological Review*, *86*, 124–140.

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Berlin: Springer-Verlag.

Duncker, K. (1945). On problem solving. *Psychological Monographs*, *58*, 1–113.

Fodor, J. A. (1975). *The language of thought*. Cambridge, MA: Harvard University Press.

Goto, H. (1971). Auditory perception by normal Japanese adults of the sounds "l" and "r". *Neuropsychologia*, *9*, 317–323.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural Computation*, *9*, 1735–1780.

Huber, P. J. (1964). Robust estimation of a location parameter. *The Annals of Mathematical Statistics*, *35*, 73–101.

Katz, B. (1997). Annotating the World Wide Web using natural language. *Proceedings of the 1997 Computer-Assisted Information Searching on the Internet Conference* (pp. 136–155). Montreal, Canada: Centre de Hautes Etudes Internationales d'Information Documentaire.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *Poster Session of the Third International Conference on Learning Representations*. San Diego, CA: University of Amsterdam.

Lei, T., Joshi, H., Barzilay, R., Jaakkola, T., Tymoshenko, K., & Moschitti, A., et al. (2016). Semi-supervised question retrieval with gated convolutions. *Proceedings of the 2016 Conference of the Association for Computational Linguistics: Human Language Technologies* (pp. 1279–1289). San Diego, CA: ACL.

Minsky, M. (1980). K-lines: A theory of memory. *Cognitive Science*, *4*, 117–133.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A., Veness, J., & Bellemare, M., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, *518*, 529–533.

Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. *Proceedings of the Twenty-Seventh International Conference on Machine Learning* (pp. 807–814). Haifa, Israel: Omnipress.

Newell, A., Shaw, J. C., & Simon, H. A. (1958). Elements of a theory of human problem solving. *Psychological Review*, *65*, 151–166.

Ohlsson, S. (1984a). Restructuring revisited. *Scandinavian Journal of Psychology*, *25*, 65–78.

Ohlsson, S. (1984b). Restructuring revisited. *Scandinavian Journal of Psychology*, *25*, 117–129.

Pennington, J., Socher, R., & Manning, C. (2014). Glove: Global vectors for word representation. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing* (pp. 1532–1543). Doha, Qatar: ACL.

Reason, J. T. (1990). *Human error*. New York: Cambridge University Press.

Sloman, A. (1978). What about their internal languages? *Behavioral and Brain Sciences*, *1*, 602–603.

Winston, P. (1970). *Learning structural descriptions from examples*. Doctoral dissertation, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.

Winston, P. (2012). The next 50 years: A personal view. *Biologically Inspired Cognitive Architectures*, *1*, 92–99.

Winston, P. (2017). *Self-aware problem solving*. Unpublished manuscript, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA.