The Emergence of Symbolic Structure from Data in Prototype Neural Networks

by
Aaron Traylor
Bachelor of Science, University of Massachusetts Amherst, 2018

A dissertation submitted in partial fulfillment of the requirements for the Degree of Doctor of Philosophy in the Department of Computer Science at Brown University

Providence, Rhode Island
May 2024
This dissertation by Aaron Traylor is accepted in its present form by the Department of Computer Science as satisfying the dissertation requirement for the degree of Doctor of Philosophy.

Date 4/29/2024

Ellie Pavlick, Director

Date 4/29/2024

Roman Feiman, Reader

Date 4/29/2024

George Konidaris, Reader

Date 4/29/2024

Tomer Ullman, Reader
Harvard University

Approved by the Graduate Council

Date

Thomas A. Lewis
Dean of the Graduate School
Acknowledgements

Okay, here we go.

Thanks to my sister, Amanda, for being the best sister in the universe.

Thanks to my mom, Diana, for the lens with which she taught me to view the world.

Thanks to my dad, Scott, for his unwavering support and interest in all that I do.

Thanks to my advisor, Ellie, for being an incredible mentor. I don’t think I could put into words how much I learned here, and I will be forever grateful. If I could do it all over again, I’d come back to the LUNAR lab at Brown.

Thanks to my co-advisor, Roman, for his likewise incredible mentorship.

Thanks to the members of the LUNAR and BLT labs past and present.

Thank you to all of the people who make the Brown CIT so wonderful.

  – Thanks to Talie for adventures innumerable.
  – Thanks to Catherine and Jack, my co-conspirators, and Michal and Apoorv and Ruochen and Mikey and Roma and Charlie and Albert and Thao and Eric and Jason and Max and Sam and Franco and Lucy and Kweku and William. Being in Providence with you all was amazing and I will always remember it. Don’t be a stranger.
  – Thank you to Lauren, Dawn, and Genie, who helped me and everyone along the way.

Thanks to the Brown Japanese department.

Thanks to Najoung and Cookie for making this journey a lot more bearable.

Thanks to Casey, Julia, and Sarah; for growing up together.
Thank you to everyone from UMass:

- Matt and Tyler, for extending the campus to Providence
- Vinny, Zach, Colin, and Big Dan, the fireworks in my life
- Amanda, for friendship
- Nika, I’m so glad we kept in touch
- Bailey and Priyanka, from whom I learned a great deal about competition
- Nick, for everything

And lastly, thank you to the Pokémon people:

- Wolfe and Aaron, for exploring the stars
- Brendan and Ed, for proving red boats go faster
- Jake, for teaching me what it means to be a good friend
- Yuki, for adventures around the world
- Scott, Evan, Huy, Duy, Paul, Cassie, Len, Anthony, Mike, Rushan, Danny, TJ, Nic, Gene, Adam; for being a whole team of best friends
- Zach, Mont, Egg, Madeline, Dani, Chuppa, Clayton, Pedro, Trista; I’ve never laughed so hard
- PARC, one day we’ll build that AI
- Steph, David, and Maru; for making Pokémon and academia a little closer together
- Haley, for her enthusiasm

The world is very big, and it is full of people and love, and I am very small. How lucky am I to have existed here at the same time as you all?
Contents

List of Tables viii
List of Figures ix

1 Introduction 1

2 Related Work 6
  2.1 Symbolic Structure 6
  2.2 Learning Structure with Neural Networks 7
  2.3 Architectures 8
    2.3.1 RNNs and LSTMs 8
    2.3.2 Transformers 8

3 Using Formal Languages to Study Language Models’ Representations of Symbolic Structure 10
  3.1 Introduction 10
  3.2 Experimental Design 11
    3.2.1 Dataset Generation 11
    3.2.2 Models and Training 12
  3.3 Related Work 13
  3.4 Results and Discussion 13
  3.5 Discussion 16

4 Investigating Weaker Precursors of Symbolic Structure in Object-Tracking Neural Networks 17
  4.1 Introduction 17
  4.2 Background 18
    4.2.1 Two Tests of Reasoning by Exclusion 18
    4.2.2 Explicit vs. Implicit Logical Reasoning 18
    4.2.3 This Work 19
  4.3 Experimental Design 19
    4.3.1 Evaluation Task 19
    4.3.2 Proof-of-Concept 22
List of Tables

3.1 Propositional logic grammar. ................................................................. 11
3.2 Summary of language modeling performance. For each model, on each training dataset, we report \textit{PPL / \%Syn / \%Sem} where PPL is the perplexity on heldout data (drawn from the same distribution as the training corpus), \%Syn is the percentage of generated sentences that are syntactically well formed (i.e., parseable), estimated on a set of 1,000 generations sampled from the trained model, and \% Sem is the percentage of generated sentences that are semantically well formed (i.e., satisfiable), estimated on the same set of 1,000. ........................................ 12

4.1 Results on the Two-Cup task and the control datasets. The models’ predictions before the critical frame are always split roughly evenly between the two cups, when relevant. In the Outside experiments, “elsewhere” is the correct answer, because the ball is outside of both cups. ........................................................................... 24
List of Figures

3.1 Each value in this graph represents average classification score across 125 iterations of a simple nearest neighbor probing classifier averaged across 5 random seeds of the model (625 accuracy numbers per box and whiskers plot). The dotted line is random chance / maximum class accuracy (33%).

3.2 PCA of the representations created by the Medium Transformer model.

4.1 Frames from the Two-Cup task test video. Frames a–d set up the scene and create ambiguity between the two possible hiding locations for the ball, and frames e–h are the essential part of the task on which we evaluate the model. The goal is to determine the location of the small yellow ball, which drops at frame d. At frame e, the subject should know that the ball is either in the left cup or the right cup, but its exact location is unknown. When the left occluder drops at frame f, the subject should know that the ball is not in the left cup, and must be in the right cup, despite not seeing it. This is the critical frame.

4.2 Frames from the Invisible Displacement task within our experimental framework. At frame b, because of the random stopping, it is ambiguous whether the ball has landed behind the yellow or the red occluder. At the critical frame c, it becomes possible to infer the ball’s location.

4.3 Key frames of the Two-Cup task, visualizing the particles of the ADEPT model. There are 100 particles, and each circle is one particle’s representation of its beliefs about the ball’s location.

4.4 Accuracy (% of examples that place bounding box for the ball in the "correct cup") on the Two-Cup task over time when finetuned on the Two-Cup task and pretrained using various methods. Each is averaged across 5 random seeds. There are 5,000 data points in one epoch.

5.1 Graphical diagram of the path-patching process. Attention heads are represented as circles (layer, head index), and contextual representations of each token (as well as the next token prediction) are represented as rectangles.

5.3 Model behavior when predicting same/different (token 15) is shown. We measure attention visualized as a shade of purple, with deeper shade corresponding to higher attention to that token. We create “corrupted” minimal pairs in which changing a token (light blue) either changes the correct label at index 15 (examples b, c, e) or does not (d, f). We make small path-patching edits with the minimal pair to targeted network components (layer 1 keys for b, c, d, f; queries for e, f). In other words, we replace specific components (denoted with red text) with their corresponding representation from the “corrupted” sequence, but hold all other representations constant, and run the model and get a new same/different prediction. In all test examples, making the small patch successfully results in the model’s prediction changing to align with the “corrupted” example.

5.4 Model performance over training on patching subtasks. Each row corresponds to an individual model’s training loss (solid line) and subtask accuracy (dashed line) over time; blue and orange lines respectively correspond to models which reach 100% test accuracy and to those that do not.
Chapter 1

Introduction

In recent years, neural network models have revolutionized the field of artificial intelligence. For example, large language models, which are neural networks that are trained to predict the next word in a sentence, generate language at a quality that would have seemed impossible even 10 years ago (Brown et al., 2020). Similarly, neural computational vision models demonstrate complex scene understanding (Dosovitskiy et al., 2020). Creating computational systems that have human-level cognitive reasoning capability is a long-standing goal of the field of artificial intelligence, and these models appear to be a major step towards that goal.

Modern neural language models are able to succeed at many complex tasks, such as answering user questions, holding dialogue, writing code, summarizing documents, or generating novel text, among others. Models are even capable of generalizing zero-shot to novel tasks that are held out from their training data entirely (Sanh et al., 2021). Given these successes, it has become a common expectation that a large enough model trained on enough data will not only succeed on tasks seen during training, but it may even be able to generalize to entirely new tasks “for free”, simply as a byproduct of the training process. However, large language models have failed in some settings at tasks such as logical reasoning (J. Liu et al., 2020), negation (Ettinger, 2020; She, Potts, Bowman, & Geiger, 2023), and compositional generalization (Kim & Linzen, 2020).

Many of the tasks for which LMs attract the most attention—both for their successes and their failures—are those that are argued to require some form of symbolic structure. In symbolic structure, there are entities (e.g. a cat, a muffin, a variable in code) which are represented as atomic symbols (CAT, MUFFIN, X), and there are content-independent functions that operate over those symbols. Symbolic structure is defined differently in many different contexts, and in this work we consider it a broad superset of the “language of thought” hypothesis defined in Quilty-Dunn, Porot, and Mandelbaum (2023), which invokes that within Fodor and Pylyshyn (1988) without making claims with regards to systematicity and productivity. Examples of symbolic structure include code written in a programming language, where variables stand for whatever they are assigned to, and functions and conditionals are abstract structure which affect those variables. Human language is similarly symbolic, where words can refer to specific entities and the structure of language conveys additional meaning. More tasks argued to require symbolic structure include logical reasoning and deduction, variable binding, and arithmetic, among others.
Whether or not neural network models can learn tasks with symbolic structure or whether symbolic structure is even necessary in order to solve this category of tasks is the subject of much debate. Generalizing symbolic structure has classically been argued to be something that humans do freely, but is outside of the capability of neural network models (Fodor & Pylyshyn, 1988; Marcus, 2001). Such arguments insist on a need for non-neural approaches in which symbolic structure is built into the system by design, or integrating those systems into neural architectures through “neurosymbolic” approaches. On the other hand, proponents of neural network-based models call into question whether or not learning symbolic structure is necessary to solve these tasks, and argue that neural network models are capable of generalizing any task so long as the training data and environment sets it up to succeed (LeCun, Bengio, & Hinton, 2015).

The evidence which suggests that models struggle with tasks with symbolic structure is at odds with their general success in the modern era, where models are used successfully to accomplish a broad variety of tasks. The pattern of successes and failures in experimental evidence— as well as the debate surrounding them— is ultimately nuanced, and requires more investigation in order to determine the extent of models’ capabilities on tasks that are argued to be symbolic.

Precisely determining the extent to which large-scale modern models could succeed on symbolic structure tasks is further challenging because of the same reasons for which these models succeed. Model size, training data scope, and lack of task-specific structure within the model all contribute to difficulty characterizing the limits of their capacity for symbolic structure. While large models with vast amounts of unstructured parameters are useful for capturing the complexity of language, language models frequently learn and exploit brittle heuristic methods for solving tasks (McCoy, Pavlick, & Linzen, 2019; Kim & Linzen, 2020). As a result, it is challenging to distinguish whether large models that show success at symbolic tasks have learned the underlying symbolic structure or are exploiting various complex heuristics that on the surface lead to similar task performance. Furthermore, the large datasets that language models are trained on often include examples of symbolic reasoning, such as logical inference tasks. However, the inclusion of such structure is not rigorous and controlled, meaning it is unclear what aspects of underlying symbolic structure are good or poor targets with which to evaluate model generalization in these settings.

It is tempting to say that the ability for representing symbolic structure can arise as a result of training given the task performance of successful models. However, it is challenging to determine that for sure within the modern neural training paradigm, in which training datasets are enormous and all-encompassing, models have trillions of parameters, and there is little control over system and task design. In this work, we investigate the necessary conditions for modern neural networks to succeed on tasks thought to have symbolic structure when learned from training data. In order to do so within an empirical and controlled setting, we scale back the problem greatly compared to modern large-scale models in two main regards.

First, we use prototype neural networks— i.e. smaller instances of the same neural architectures underlying large-scale high-performance models. The dominant training paradigm of modern neural models is to train relatively unconstrained neural networks— i.e., neural networks without any task-specific or cognitively-motivated inductive biases, or “vanilla” models. For example, vanilla Transformer models have no specialized architecture for working memory, in spite of their ability to succeed at tasks which appear to require it (Brown et al., 2020; Huang, Abbeel, Pathak, & Mordatch, 2022; Du, Yu, & Zheng, 2021; Lewkowycz et al., 2022).
and vanilla Transformer models without any such inductive biases remain the dominant architecture for the modern AI systems which are especially visible in their success on complex tasks (Brown et al., 2020; Touvron et al., 2023). In this work, we train smaller vanilla neural architectures, which we call “prototype” neural networks. Although these networks are not minimally small, they are orders of magnitude smaller than the largest modern models (10⁶ parameters versus 10¹²).

Second, we generate symbolic tasks algorithmically, and tightly control the composition of train and test data. This allows for the evaluation of the model’s emergent capabilities given the symbolic tasks. Emergent capabilities refer to those learned by neural models during the training process, in contrast to built-in architectural capabilities or external components. As an example, within large language models that can generate grammatically correct English sentences, a mechanism for English syntax emerges (Brown et al., 2020). Such language models are often only trained with the objective of predicting the next word; although syntax is certainly an important underlying component of doing so successfully, representing syntax is by no means a requirement of the training objective. Identifying what behavior does or does not emerge from training is thus an important component of qualifying a model’s effectiveness.

The paradigm of small-scale controlled experiments does not provide insight as to whether symbolic capabilities emerge with model size, for which there is at least some evidence (Wei, Tay, et al., 2022). Rather, creating a tightly controlled model and data environment lets us investigate through an empirical lens how modern vanilla neural architectures perform at tasks argued to contain symbolic structure, and to determine what causes these networks to succeed or fail. Because of the underlying fundamental similarity of vanilla models, results in this area can help us understand how vanilla models in other contexts learn and generalize, what kind of task representations emerge as a result of the training process “for free”, and which failure cases might require significant changes in architecture or scale in order to solve.

Furthermore, it is not necessarily the case that success in this setting means that neural networks do or do not learn and generalize symbolic structure. Merely succeeding at a task thought to be symbolic does not mean that the neural network has learned a representation which captures symbolic structure in its entirety; it may have partially captured some elements of symbolic structure, or it may have learned something functionally equivalent to task-relevant aspects of symbolic structure that would not generalize if extracted from the model and the setting, meaning that representations of symbolic structure are not in fact necessary to solve the task. We compare these possibilities when relevant upon success of the evaluated models.

In summary, we use the same neural network training paradigm as used by large-scale models when trained on corpora of natural language text, but we scale down the model size by several orders of magnitude into a “prototype” model, and train and evaluate it on algorithmically generated symbolic tasks, with the goal of identifying whether neural models in these settings can learn symbolic structure, or whether symbolic structure is required in order to solve these tasks.

**Thesis statement.** We hypothesize that prototype neural networks can learn to perform well at a set of tasks previously argued to involved symbolic structure. We evaluate this hypothesis on small LSTM- and Transformer-based neural models trained from scratch within a framework of carefully controlled experimental setups testing symbolic structure.
We find mixed results across our three chapters, suggesting the answer to our question is nuanced.

First, in Chapter 3, we investigate the conditions under which controlled distributional signal within a training dataset is sufficient for models to differentiate content that grounds to differing symbolic meaning. Modern language models are able to create very good representations of language from raw text (“form”) alone, e.g. the words “Barack Obama”, without ever observing meaning, e.g. the person in the physical world, Barack Obama. The extent to which a model trained only on form can capture or even distinguish meaning in its entirety is an open question. We examine this within a domain of propositional logic, where form and meaning are both precisely defined, meaning the degree to which the model captures them can be measured. We first “pre-train” LSTM and Transformer neural language models on strings from a synthetic propositional logic language and then measure the degree to which their representations distinguish words with identical syntax (i.e. appearing in the same place within the string) but different semantics (i.e. affect the meaning of the propositional logic string differently). We find that while the language models do a good job at representing the syntax of the language, they do a poor job of capturing meaning; models both fail to consistently generate logically valid sentences that follow the constraints of the dataset, and under certain settings fail to differentiate the meaning of logical operators. These results show that distributional signal alone is not always sufficient for models to differentiate the role and meaning of symbolic logical operators, which both suggests that symbolic task performance does not necessarily emerge as a result of the training process, and that additional context may be required to learn the meaning of symbolic concepts.

It may be the case that models do not readily learn to represent full-fledged aspects of symbolic structure, such as logical operators, but instead are adept at learning weaker “implicit” versions of symbolic structure. Evidence from experiments in developmental psychology suggests that prior to learning logic, young children may first learn logical precursors that are incomplete and context-dependent, but accomplish similar behavior to full logical ability in certain settings (Völter & Call, 2017; Bermudez, 2003; Cesana-Arlotti et al., 2018; Feiman, Mody, & Carey, 2022). In Chapter 4, we evaluate neural models under a similar paradigm; a model with simplified logical precursors would perform as if it had representations of logical structure, but it would perhaps only partially capture the functionality of the operator, or only do so within a given context or environment. For example, if a model has learned that physical objects can’t be in two places at once, that knowledge may constrain its representations of objects in a manner that is functionally similar to logical inference. We investigate this possibility with neural LSTM-based object-tracking models trained on a visual task used in developmental psychology to test infants for similar logical precursors. Our setting is analogous to the standard neural training paradigm; we train the model to predict the location of a target object under occlusion in various scenes showcasing the dynamics of the environment, and then we evaluate whether successful object tracking leads to generalization to competence on a held-out logical reasoning test within the same environment. We find that models are able to track objects around the scene successfully, but doing so does not lead to any generalization at the target logical task, suggesting that the models have not learned weak precursors of symbolic logical reasoning, which further demonstrates the difficulty of learning symbolic reasoning from data for neural models.

In our last chapter, we investigate whether architectural similarity between symbolic functionality in the
human brain and the neural network architecture can lead to success on a symbolic task from training. Vanilla Transformer models are able to succeed at a wide variety of tasks which require hierarchical planning ability and working memory when performed by humans (Huang et al., 2022; Du et al., 2021; Lewkowycz et al., 2022). Human performance on such tasks is known to depend on a neural mechanism for gating (Frank & Badre, 2012; Badre & Frank, 2012; Chatham, Frank, & Badre, 2014; Rac-Lubashevsky & Kessler, 2016; Rac-Lubashevsky & Frank, 2021), which controls whether new information is maintained in working memory or not, the address in memory where it is stored, and the address from which stored information is recalled in response to a task. Transformer models do not have explicitly built in “working memory” or “gates” as part of their model structure, as LSTMs do, yet find success, raising the question whether a mechanism for gating is learned during training. Vanilla Transformers are good candidates for learning gating behavior because of inductive biases within the self-attention mechanism, i.e., the Transformer’s defining architectural component, which functions analogously to neurobiological circuitry in humans and other animals (O’Reilly & Frank, 2006; Frank & Badre, 2012; Calderon, Verguts, & Frank, 2022). We train vanilla Transformer models on a symbolic working memory task that was specifically designed to evaluate models of selective gating and working memory in computational neuroscience (O’Reilly & Frank, 2006; Rac-Lubashevsky & Frank, 2021), and we use recent techniques from mechanistic interpretability (Olah, 2022; Nanda & Bloom, 2022) to expose the mechanism that the Transformer uses in order to perform the task. We find that the models categorically succeed at the task, and that in order to succeed their self-attention mechanisms specialize in a way that resembles existing models of input-output gating— in other words, a mechanism for gating emerges within the model, which appears to capture some element of symbolic structure representationally.

In Chapters 3 and 4, we find that the models struggle to learn useful representations of full-fledged logical operators from data within a propositional logic language modelling setting, and we find that even behaviorally equivalent weaker precursors do not always emerge. However, in Chapter 5, we find that models succeed on a hierarchical memory task, suggesting that certain architectural inductive biases can lead to representations mimicking brain functionality, and highlighting the relevance of model architecture to learning key components of symbolic structure. Overall, our results show that although architectural biases such as attention in Transformer models lend themselves well to learning particular aspects of symbolic structure, symbolic structure does not always emerge “for free” within neural networks as a result of training.
Chapter 2

Related Work

2.1 Symbolic Structure

Symbolic structure at its core involves atomic symbols and functions which manipulate them. In this thesis, an arbitrary number of symbols are generated with no explicit meaning (as made explicit in Chapters 3 and 5); however, in more natural settings, symbols are representative of some content (e.g. the word “Providence” may refer to the city of Providence, located in Rhode Island).

Although this framework is operationalized many different ways in many different settings, such as in propositional or first-order logic, in this work we most closely align with symbolic structure as defined by Quilty-Dunn et al. (2023). In their work, symbols and the functions that combine them are two key components of a larger “language of thought” that underlies human cognition evidenced by experiments and cognitive theory, which is in their words a “a combinatorial, symbolic representational format that facilitates logical, structure-sensitive operations”. The language of thought hypothesis was defined by Fodor et al. (1975), and revisited by Fodor and Pylyshyn (1988). Quilty-Dunn et al. (2023) revisit it yet again under a modern lens with emphasis on neural networks as an alternative yet ultimately insufficient model of human reasoning and language faculties.

The subject of whether neural networks are able to learn and generalize symbolic structure is a key component of a long-standing debate in the fields of artificial intelligence and cognitive computational science about whether neural networks are good models of human cognition. This criticism dates back to the earliest iterations of neural networks (Minsky & Papert, 1988) and has resurfaced many times throughout their various resurgences (Fodor & Pylyshyn, 1988; Marcus, 2001; Quilty-Dunn et al., 2023). In our work, the experiments and results are not directly related to whether neural networks are good or bad models of human cognition, and have little to say on the surface explicitly about whether neural networks capture elements of the language of thought. Rather, we use the tenets of this criticism and debate to inspire our experimental paradigm where we investigate empirically the extent to which modern models can represent symbolic structure, or if they find other means of succeeding at tasks argued to be symbolic.
2.2 Learning Structure with Neural Networks

There are many studies investigating large language models’ ability to learn symbolic structure in empirical settings with natural language. These studies largely use full-size models trained on large datasets of natural language corpora, with the intention of creating domain-general representations of the language and transferring those representations to tasks explicitly requiring semantic knowledge (Peters et al., 2018; Devlin, Chang, Lee, & Toutanova, 2018; Brown et al., 2020). J. Liu et al. (2020) find that large-scale language models show limited success in comparison to human experiments on natural language logic and deduction experiments. Ettinger (2020) find corroborative evidence of weak logical reasoning abilities and highlight the failure of models on tasks where negation is explicitly required (e.g. BERT Transformer models are insensitive to the negation in sentences such as “A robin is a bird” versus “A robin is not a bird”.

The work in this thesis concerns Transformer models trained on small, algorithmically generated symbolic tasks. Other work that does so includes tests of whether neural architectures can identify truth conditions in propositional logic languages (Evans, Saxton, Amos, Kohli, & Grefenstette, 2018) and memory tasks (B. Liu, Ash, Goel, Krishnamurthy, & Zhang, 2023). Other notable examples include the SCAN task, used for measuring compositionality and generalization (Lake & Baroni, 2018), and work which investigates LM knowledge acquisition and fact memorization using a synthetic dataset of entity-relation tuples (Kassner, Krojer, & Schütze, 2020). Kim and Linzen (2020) train and evaluate models on a semantic parsing task highlighting compositional generalization, and find that LSTM and Transformer models both struggle to generalize to held-out test examples that contain elements from training but are structurally different. In this work, we create a suite of tasks from various domains (propositional logic, computational developmental psychology, computational cognitive neuroscience) that adds to this body of algorithmically generated tasks.

There is a body of work that studies the emergence of symbolic properties within larger models (hundreds of billions or trillions of parameters). The “scratchpad” or “chain-of-thought” approaches to large language model prompting employ various techniques to get large language models to generate answers where they break down their answers into step-by-step components, causing their logical reasoning ability to skyrocket (Nye et al., 2021; Wei, Wang, et al., 2022). For instance, Nye et al. (2021) find that some large language models can succeed at 8-digit addition when employing this technique. Wei, Tay, et al. (2022) find a general pattern of results across the field that shows that large language models perform better at some tasks the larger the models are and the longer they are trained for (from the perspective of scaling to trillions of parameters). The tasks that models succeed at in this setting are arguably symbolic in nature and the benefits of chain-of-thought style approaches merit further research. As previously discussed in Chapter II, the empirical methods employed in this thesis cannot address the performance of models at large scale. Although chain-of-thought prompting may lead to performance gains at larger scale on tasks thought to be symbolic, we ultimately choose to evaluate other elements of the model training paradigm.
2.3 Architectures

Artificial intelligence systems are based on the neural network architecture (McCulloch & Pitts, 1943). Neural network models are powerful learning algorithms and function approximators that can be applied to any computational task. Indeed, a simple multilayer perceptron with just one hidden layer can provably approximate any function, provided that the hidden layer is large enough (Hornik, Stinchcombe, & White, 1989). Although they take many forms, all neural networks learn associatively from data; given a dataset with inputs and outputs, the neural network will minimize prediction error when mapping each input to each output. Our work investigates the learning behavior of various neural architectures, discussed briefly below.

2.3.1 RNNs and LSTMs

RNNs (Recurrent neural networks) are classically standard neural approaches for sequence modelling tasks. They are sequential models; they calculate a representation for one element in the sequence (e.g. one token)—also called a “timestep”—before moving on to the next element, meaning that in order to process a sequence of n elements, n computations must be done in sequence (Elman, 1990). A major weakness of RNNs is the “vanishing gradient” problem; because they maintain one perpetual representation over the course of the sequence that is sequentially updated at each timestep, for very long sequences the model struggles to maintain key information from earlier in the sequence within that representation.

LSTM (Long Short-Term Memory) models are an extension of recurrent neural network invented and popularized in the late 1990s (Hochreiter & Schmidhuber, 1997). LSTMs solve the vanishing gradient problem for recurrent neural networks by including memory “gates” within the representation that is maintained over the entire sequence, analogously allowing the model to choose what information can be remembered or forgotten over the course of the sequence. As a result, LSTMs are capable of performing well over much longer sequences than RNNs. Before the introduction of Transformer models in 2017, many high-performance language models used LSTMs, such as the ELMo (Peters et al., 2018) and BERT (Devlin et al., 2018) models.

2.3.2 Transformers

Transformers (Vaswani et al., 2017) are powerful language models which create contextualized representations of sequences of words, where they learn to to predict the next token one at a time using an “attention” mechanism (Bahdanau, Cho, & Bengio, 2014) to scan the previously seen tokens for relevant information. These models are able to learn and represent any arbitrary sequence modelling task. In contrast to recurrent sequence models, Transformer models operate in parallel; they create a representation for every word in the sentence at the same time.

For a given prediction, Transformer attention generates three separate vectors at each position in the sequence: a query, key, and value ($q$, $k$, $v$). The query vector scans the previous context (including the current token) for relevant keys, and calculates how much the current prediction should “attend” to those positions. Then, the value vectors at those positions are multiplied by the corresponding weights, summed up, and added to the next representation: for token $i$ at layer $j$, the contextual representation is $\sum_k q^j_i \cdot k^j_i \ast v^j_i$. Thus, the next token prediction includes information from earlier in the sequence by combining the value vectors
from previous tokens. In other words, Transformer attention can be viewed as a read/write mechanism: for a given token, the queries and keys dictate which tokens to read from, the values are the content that is read proportional to the attention calculated by the keys and queries, and the summed content is written to a new representation at the given token.
Chapter 3

Using Formal Languages to Study Language Models’ Representations of Symbolic Structure

3.1 Introduction

A current open question in natural language processing is to what extent language models (LMs; neural networks trained to predict the likelihood of word forms given textual context) are capable of truly understanding language. [Bender & Koller, 2020] argue that, since such models are trained exclusively on the form of language, they cannot possibly learn the meaning of language. We argue that the question of whether language models can learn meaning cannot be settled a priori. While language models only have direct access to form, linguistic form often correlates with meaning. The strength of the correlation varies across both different aspects of language and different tests of linguistic competence. While several intuitive tests of understanding (e.g., demonstrating knowledge of the word dog by identifying pictures of dogs) are out of scope for LMs, many tasks which NLP aspires to solve (e.g., question answering, machine translation) operate entirely on natural language input and output. Thus, a relevant question is whether models which operate only on the forms of language can nonetheless learn to differentiate meanings.

Our goal is to focus on a tractable subproblem in order to improve our intuitions about the types of distributional signals that LMs can use to extract information relevant to meaning. We simulate a language modeling setup using propositional logic, in which we can naturally operationalize form to be strings of symbols in the language and meaning to be truth conditions. We define the semantic transparency of a text-only training corpus to be the degree to which an LM trained on that corpus learns to differentiate between aspects of form that affect truth conditions and aspects of form that do not. We have two primary research questions. First, what constraints on corpus generation produce greater semantic transparency? And second, are any such constraints sufficient for an LM to adequately differentiate meanings?
Table 3.1: Propositional logic grammar.

3.2 Experimental Design

3.2.1 Dataset Generation

We consider the form of a sentence to be simply the observed, syntactically-valid strings of characters and the meaning to be the truth conditions. Propositional logic is a simple language in which we can characterize both form and meaning. We use the grammar in Table 3.1 with standard semantics.

We focus our analysis on whether the representations of logical operators (\(\wedge, \vee, \neg\)) are influenced by distributional patterns that go beyond their superficial syntactic similarity evident in the grammar. That is, if a trained LM identifies that the meanings of \(\wedge_1 \cdots \wedge_k\) are identical to one another, and different from the meanings of \(\vee_1 \cdots \vee_l\), we expect the embeddings for the \(\wedge_i\) to be more similar to one another than they are to any of the \(\vee_i\) or the \(\neg_i\). We consider a corpus to be semantically transparent if an LM trained on the corpus learns semantically-clustered representations of the logical operators.

We generate four different training corpora, motivated by different assumptions one might make about how natural language corpora arise. These constraints are as follows, ordered roughly from weakest to strongest:

1. Syntactic Constraint. Speakers only generate sentences which are syntactically well-formed (that can be parsed by a syntactic parser). Here, this amounts to sampling from the grammar without additional constraints.

2. Truthfulness Constraint. Speakers of the language are constrained to generate sentences that are true in some context, i.e., that evaluate to True in at least one possible world. To implement this, we again sample from the grammar but additionally check with a satisfiability checker and omit sentences which are not satisfiable. E.g., \((\text{sym}_1 \wedge (\neg(\text{sym}_1)))\) would not appear.

3. Informativity Constraint. Speakers generate sentences not just to state true facts, but to provide listeners with information about a particular state of affairs. To simulate such a constraint, we randomly sample a set of “target worlds” \(T\) and a set of “alternative worlds” \(A\) such that \(T \cap A = \emptyset\). We then generate the shortest sentence \(s\) such that \(s\) is true in every world in \(T\) and \(s\) is false in every world in \(A\). We experiment with several sizes of \(T\) and \(A\), but report only on \(|T| = |A| = 2\) as this provides the right balance of contextual diversity.

4. Explicit Grounding. We consider a setting in which speakers explicitly dictate the full state of affairs, without ambiguity. This is not intended as a realistic model of how corpora are generated, but rather to provide an upper bound on semantic transparency by giving models a corpus in which form is perfectly correlated with
meaning. We generate this corpus in the same way as the Truthfulness corpus, but append an explicit marker of the truth values of the variables in the sentence, e.g.: \((\text{sym}_1 \land (\neg (\text{sym}_2)))\) <sep> \text{sym}_1 \text{T} \text{sym}_2 \text{F}.

<table>
<thead>
<tr>
<th>Model</th>
<th>Syntactic</th>
<th>Truthfulness</th>
<th>Informativity</th>
<th>Grounded</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small LSTM (192K)</td>
<td>21.2 / 87.7 / 87.7</td>
<td>17.6 / 88.7 / 88.6</td>
<td>21.5 / 99.6 / 99.5</td>
<td>21.2 / 87.5 / 87.5</td>
</tr>
<tr>
<td>Medium LSTM (545K)</td>
<td>17.6 / 90.2 / 90.1</td>
<td>17.5 / 89.6 / 89.5</td>
<td>20.9 / 99.9 / 99.8</td>
<td>8.3 / 89.3 / 86.8</td>
</tr>
<tr>
<td>Small Trans. (311K)</td>
<td>11.8 / 86.9 / 84.6</td>
<td>12.4 / 87.2 / 85.4</td>
<td>21.7 / 98.4 / 98.2</td>
<td>10.3 / 86.2 / 83.1</td>
</tr>
<tr>
<td>Medium Trans. (377K)</td>
<td>11.4 / 91.3 / 90.6</td>
<td>9.9 / 92.0 / 91.3</td>
<td>18.1 / 99.5 / 99.5</td>
<td>9.1 / 91.7 / 89.8</td>
</tr>
</tbody>
</table>

Table 3.2: Summary of language modeling performance. For each model, on each training dataset, we report PPL / %Syn / %Sem where PPL is the perplexity on heldout data (drawn from the same distribution as the training corpus), %Syn is the percentage of generated sentences that are syntactically well formed (i.e., parseable), estimated on a set of 1,000 generations sampled from the trained model, and % Sem is the percentage of generated sentences that are semantically well formed (i.e., satisfiable), estimated on the same set of 1,000.

**Sampling Parameters.** Each dataset consists of 100K training and 1K validation sentences. We set the number of non-reserved symbols (N in the above grammar) to 5,000, and the number of “synonyms” of each logical symbol (K,L,M) to be 5. Thus, a sentence in one of our datasets might look like \((\text{sym}_1 \land 3 (\neg 4 (\text{sym}_{85})))\), and would be true if and only if \text{sym}_1 \text{ true} and \text{sym}_{85} \text{ false}.

We generate sentences using a probabilistic context-free grammar with the rules shown above. The tree depth \(d\) of a generated sentence is controlled by a parameter \( \gamma \) such that \( P(d|d-1) = \gamma^d \). The number of unique variables in a sentence is sampled from a non-zero Poisson distribution parameterized by \( \lambda \). We set \( \lambda = 2 \) and \( \gamma = .85 \) in the reported experiments, but don’t find parameter choice affects our conclusions. Note that the Informativity dataset is generated deterministically, and thus sampling parameters do not apply and sentences in that dataset are shorter.

### 3.2.2 Models and Training

We consider LSTM and Transformer LMs of differing sizes, shown in Table 3.2. Each model is trained on one of the above datasets until convergence on the associated validation set using early stopping with a patience of 15 epochs. The LMs were implemented in PyTorch (Paszke et al., 2019) and took roughly 5 hours to converge on TitanV, TitanRTX, and QuadroRTX GPUs. We randomly initialize the embedding layer. We train 5 random restarts of each setting. Due to the regular nature of our synthetic data, we found larger models overfit the training data quickly, and thus focus on smaller models.

1. Sampled from the set of satisfying variable assignments.
2. We began by experimenting with many different dataset sizes and vocab counts. However, we did not find that models behaved differently on larger datasets and so focused on the smaller ones for convenience.
3. We set a maximum number of variables per sentence in order to bound the number of possible variable assignments.
3.3 Related Work

Our work builds upon a large body of research intended to probe which aspects of language and meaning are being captured by large LMs. Most closely related is work that assesses whether models can perform symbolic reasoning about language \cite{Kassner2020} e.g., quantifiers or negation \cite{Talmor2020, Ettinger2020, Kassner2020, A_Wang2018} or by measuring the systematicity of models’ inferences \cite{Goodwin2020, Kim2020, Yanaka2020, Warstadt2019}. Such work has tended to find that LMs reason primarily contextually as opposed to abstractly. Our evaluation method– which asks whether word embeddings cluster according to their truth-conditional meaning– is related to recent work which defines text-only models as “grounded” if the learned embedding space is isomorphic to the similarity function defined over a ground-truth meaning representation \cite{Merrill2021}. More distantly related is work on LMs’ ability to reason about numbers \cite{Wallace2019} or perform multi-hop reasoning \cite{Yang2018}.

3.4 Results and Discussion

Language Modeling Performance. We first sanity check that the trained models indeed function as LMs before evaluating the lexical representations. We compute the models’ perplexity on heldout data. However, since perplexity is not comparable across conditions (since each constraint leads to differently distributed corpora) we also sample 1,000 generated sentences from each model and compare by measuring whether the sentences are 1) syntactically well-formed (i.e., parseable) and 2) semantically well-formed (i.e., satisfiable). Even in the case of models trained with the Syntactic constraint, as seen in Table 3.2 most of the sentences produced are nonetheless satisfiable. We see no difference between the Syntactic, Truthfulness, and Explicit Grounding conditions on these metrics. (The Informativity numbers are likely higher due to the shorter sentences that result from that generative process.) The fact that models trained only on satisfiable sentences nonetheless generate sentences which do not abide by such constraints suggests the models fail to encode less overt distributional patterns, which depend, for example, on recognizing abstract relations such as “sameness” of symbols in order to recognize violations (e.g., \((A \land \neg A)\)). The failure to capture such properties of the data even in this simplified setting might have negative implications for the models’ ability to infer abstract semantic relationships from more complex natural language corpora.

Representations of Logical Symbols. Again, our first question is: What constraints on corpus generation yield the greatest amounts of semantic transparency? We quantify this by measuring how well the embeddings learned by the trained LMs correspond to our truth-theoretic notions of semantic equivalence: e.g., are \(\land_1\) and \(\land_2\) more similar to one another than \(\land_1\) and \(\lor_1\)? We use a nearest neighbors probing classifier to evaluate whether models distinguish the operators at the lexical level. We run \(k\)-fold cross validation, in each iteration choosing one symbol per class (i.e., one \(\land\), one \(\lor\), one \(\neg\)) as the class exemplars, and then classifying the remaining points using cosine similarity. We set \(k\) to 125, so that we observe every symbol combination as exemplars. We report accuracy averaged across folds and random restarts.
Figure 3.1: Each value in this graph represents average classification score across 125 iterations of a simple nearest neighbor probing classifier averaged across 5 random seeds of the model (625 accuracy numbers per box and whiskers plot). The dotted line is random chance / maximum class accuracy (33%).

Probing classifier results are shown in Figure 3.1. Figure 3.2 shows an embedding visualization for one model (Medium Transformer). We find that training on the Syntactic and on the Explicit Grounding dataset leads to the least and the most distinguishable operators respectively for all models, and the other conditions end up between these values.

These results address our first question: there is some difference in semantic transparency between differently constrained datasets. Interestingly, the Transformer models perform better in the Truthfulness condition than in the Syntactic condition, which the LSTMs fail to differentiate. This suggests that, even if it does not necessarily manifest in the models’ generations (Table 3.2), the Transformer architecture may nonetheless be capable of picking up on some of the more abstract distributional patterns via which syntax and semantics are correlated. Further work on larger models would be required to explore this in depth.

In addition, we observe little difference between the quality of the representations learned in the Informativity condition and those learned in the Truthfulness condition; one exception might be in the Medium LSTM, though we cannot confirm that this difference is robustly reproducible. Thus, based on our experiments, there is no evidence that Informativity alone yields greater semantic transparency. However, we note that the experimental setup for Informativity is not directly comparable to the others (e.g., sentences are shorter and less diverse than in Truthfulness) and thus further study would be needed to make strong claims, positive or negative.

Finally, we note that in nearly all cases, models are able to differentiate ¬ from the other operators, likely because it is a unary operator and thus syntactically different from the binary operators. Thus the difference in accuracy is almost entirely due to whether the representations of ∧ and ∨ are differentiated (as shown in Figure 3.2). This gives a negative answer to our second question concerning whether any constraints are sufficient for an LM to adequately differentiate meaning. Apart from the Small Transformer on the Explicit Grounding condition, none of the models can completely distinguish between symbols that are similar in form but different in meaning.
Figure 3.2: PCA of the representations created by the Medium Transformer model.
3.5 Discussion

Using propositional logic corpora to simulate a controlled language modeling setting, we ask: 1) Do properties of the training corpus affect LMs’ abilities to differentiate the meanings of logical operators? and 2) Do any training corpora lead to models that differentiate these meanings to a satisfactory degree? Our results imply a positive answer to (1): Models trained on corpora generated with different constraints appear to perform differently at the task of separating $\land$ from $\lor$. However, these differences are a function of both data and model. For example, the Transformer architecture seems better able to learn from weaker signal (corpora generated only with a Truthfulness constraint), while LSTMs require more explicit signal (direct access to truth values). On question (2), our results are largely negative for the syntactically similar operators. Even the most semantically transparent training data did not enable models to separate the representations of symbols with similar form but different meaning. Only the Small Transformer trained on the Explicit Grounding condition can perfectly differentiate $\land$ from $\lor$ at the lexical level, despite the task’s controlled nature. However, every model did separate $\neg$ from both $\land$ and $\lor$, illustrating how syntactic differences can support differentiation of meaning.

Overall, we contribute a novel framework, based on syntax and semantics of propositional logic, via which we can explore questions of the linguistic capabilities and weaknesses of neural LMs. Our experiments represent a first step in this line of work, but further work is needed to fully appreciate the implications of these results in natural language settings, in particular, how closely the constraints explored here mirror real corpora, and how such learning is influenced by noise and ambiguity found in human language. One specific limitation of our experiments is that we constrain our analysis to the lexical representations—i.e., we assume that differences between the meanings of $\land$ and $\lor$ should be encoded in the lexicon, via context-invariant type embeddings. While this assumption is commonplace in formal semantics, neural LMs open the possibility of alternative representations of lexical and compositional semantics. Our results do not rule out the possibility that the relevant semantic distinctions are encoded elsewhere in the model, above the lexical layer. However, we take the combination of the lexical probing results and LM generation results as suggestive but not confirmational evidence of a more general negative finding.
Chapter 4

Investigating Weaker Precursors of Symbolic Structure in Object-Tracking Neural Networks

4.1 Introduction

People have the capacity for flexible logical reasoning. For example, given two alternatives (A or B), and subsequent information that allows ruling out one of them (not A), people can conclude that the other is true with certainty (therefore B), termed reasoning by exclusion. It is an open question whether achieving similar reasoning with neural models will require explicit logical components to be built into the network architecture or if the capacity for such reasoning can be learned from data.

Prior work on logical reasoning in neural networks (Marcus, 2001; Evans et al., 2018) has focused on whether models are able to acquire abstract, domain-general logical operators, such as the ¬ and ∨ found in first order logic. However, recent psychological studies of logic in non-human animals and human infants have suggested that this powerful reasoning machinery does not appear in adults fully formed, ex nihilo. Rather, this work has argued that both over the human lifespan and across species, domain-general logical operators may develop and evolve from scaffolding provided by precursors that are themselves more limited. These precursors are implicit logical operators that can differ from the explicit, domain-general forms in two ways: they might only be able to operate on content in specific domains, and they might perform only some of the functional role of their full-fledged counterparts (Völter & Call, 2017; Bermúdez, 2003; Cesana-Arlotti et al., 2018).

In this work, we ask whether neural network models, which lack explicit representations of the logical operators for negation and disjunction, can nonetheless acquire implicit representations of such operators via self-supervised training. In particular, we focus on implicit logic within the domain of intuitive physics, as

---

1See Feiman et al. (2023) for additional discussion.
this is one of the earliest domains in which such reasoning emerges in young children (Cesana-Arlotti et al., 2018; Cesana-Arlotti, Kovács, & Téglás, 2020; Feiman et al., 2022). We design a set of experiments in which models are trained to reason about the dynamics of the physical world, and then evaluated on a task from developmental psychology that requires reasoning about the location of a hidden object and is considered to be a face-valid test for the representation of (implicit) negation and disjunction. We find that, by most measures, neural networks are unable to generalize zero-shot to the logical reasoning test, even when given training data which directly illustrates the requisite reasoning pattern. However, in transfer learning experiments, we find evidence that neural networks encode some degree of structural similarity between visually distinct but logically equivalent tasks, suggesting that they may yet be capable of representing the desired operators. Future work will need to determine the exact training conditions under which they would do so.

In summary, our primary contributions are: (1) We introduce the notion of implicit logic, taken from developmental and comparative psychology, into the repertoire of neural network evaluation; (2) We adapt a standard test of logical inference in humans and use it to evaluate neural network models; (3) We present a series of studies which present primarily negative results regarding neural networks’ ability to learn implicit logical reasoning in the physical domain, but which offer some suggestive evidence regarding the models’ ability to transfer representations between logically equivalent tasks.

4.2 Background

4.2.1 Two Tests of Reasoning by Exclusion

To test whether computational models can reason using implicit negation and disjunction, we adapt two tasks previously used with infants (Feiman et al., 2022; Piaget, 1954) and many species of non-human animals (Call, 2004; Völter & Call, 2017). In the "Two-Cup" task (see Figure 4.1), participants first see two cups, which are then hidden behind a screen. An object (e.g., a toy or food) is then lowered behind the screen into one cup (setting up A or B). The screen is then removed, showing that one cup is empty (not A), licensing the inference that the object must be in the other cup (therefore B). Finally, participants are invited to search. Success requires representing (explicitly or implicitly) that the ball is not in the empty cup in order to avoid searching there. Infants and many animal species succeed on this task in a zero-shot setting. In Piaget’s (1954) “Invisible Displacement” paradigm, participants see a hand holding an object. The hand closes to hide the object, moves behind an occluder, and then emerges again, empty palm facing the participant (A or B; not A). This licenses the inference that the object must have been deposited behind the occluder (therefore B). In this work, we use the Two-Cup task as our target test task. In Sections §4.4.2 and §4.4.3, we train on Invisible Displacement in order to assess whether a neural model trained to solve one reasoning-by-exclusion task can transfer its representations to another task that is formally similar but visually distinct.

4.2.2 Explicit vs. Implicit Logical Reasoning

One way to solve both tasks is with explicit symbolic logical reasoning: represent the initial possibilities for the object’s location as \( A \lor B \), and represent evidence ruling out one of them as \( \neg A \), thus licensing the conclusion \( B \).
However, more minimal solutions are also possible. Feiman et al. (2022) propose that logical representations (negation, disjunction), can be \textit{implicit} in two senses. First, while explicit logic is characteristically domain-general (OR and NOT can compose with any concepts, regardless of their content), implicit logical operations can be domain-specific, operating only over certain kinds of content (e.g. representations of objects’ locations). Second, implicit logical representations might perform only part of the function of their explicit counterparts. For example, a function that compares two arguments (e.g. \textit{blue} and \textit{red}) for incompatibility is an implicit negation in this sense. It plays part of the functional role of representing \textit{blue as not red} even as it would (correctly) not represent the negation of \textit{red} as equivalent to \textit{blue} in other computational contexts. In the Two-Cup task, participants could use an implicit representation of negation to represent that the cup being \textit{empty} is incompatible with it \textit{containing the object}.

After ruling out the empty cup from consideration, further deriving the certain conclusion that the object \textit{must} then be in the other cup is a signature of \textit{disjunction}. This conclusion could be licensed by an explicit logical operator \((A \lor B)\), but it could also be the consequence of an implicit representation with only part of the functional role of \(\lor\). For instance, the two options could be linked probabilistically, such that learning that the object is \textit{not} in one cup increases the probability that it is in the other (Feiman et al., 2022; Mody & Carey, 2016; Rescorla, 2009).

\subsection*{4.2.3 This Work}

Feiman et al. (2022) propose that such implicit logical representations – both with partial function and limited to a specific domain – could be evolutionary and developmental precursors to their explicit counterparts. On this account, implicit logical representations could emerge from specific content knowledge that is not logically structured, such as the intuitive physical understanding that predicts how objects might move in space based on visual input. In this paper, we test whether models trained to represent intuitive physics will possess emergent representations of negation and disjunction that are implicit in both senses: representations limited to the content domain of intuitive physics, and to the function of detecting incompatibility between two input states.

\section*{4.3 Experimental Design}

\subsection*{4.3.1 Evaluation Task}

To test whether a model has an implicit representation of negation and disjunction, we evaluate its performance on the Two-Cup task. Figure 4.1 shows our implementation of this task. We convert the task into a 2D video format using \textit{pygame} and \textit{Box2D}. Our environment consists of wedges and cups (which are static), a ball (which obeys gravity), and occluders (which move up and down from the bottom of the screen). The task is to predict the location of the ball at each frame of the video. In each sequence, the ball falls from the top of the screen, and can either roll off the wedge into the left or the right cup, each of which are occluded. Because the ball’s path off the wedge is also occluded, it is ambiguous which cup the ball falls into until the occluders are

\footnote{See Feiman et al. (2022) for discussion.}
removed. The occluder in front of the cup that does not contain the ball is the first to fall, thus licensing the negation inference.

We designate the frame at which the first cup is revealed to be empty (frame $f$ in Figure 4.1) to be the critical frame—i.e., the frame at which a model with implicit logic should be able to determine with certainty the location of the ball. We thus evaluate models by comparing before and after the critical frame whether the model predicts the ball to be in the correct cup, the incorrect cup, or elsewhere on the screen. After our initial tests showed that variance in performance across examples was low, we constructed a set of 100 test examples over which to compute this metric.

Figure 4.1: Frames from the Two-Cup task test video. Frames a–d set up the scene and create ambiguity between the two possible hiding locations for the ball, and frames e–h are the essential part of the task on which we evaluate the model. The goal is to determine the location of the small yellow ball, which drops at frame d. At frame e, the subject should know that the ball is either in the left cup or the right cup, but its exact location is unknown. When the left occluder drops at frame f, the subject should know that the ball is not in the left cup, and must be in the right cup, despite not seeing it. This is the critical frame.

Model

To test whether the capacity for implicit logical reasoning emerges in systems designed to represent an intuitive understanding of Newtonian physics, we use a neural network model that has a built-in notion of discrete objects, but which must learn the physical dynamics (and their generalization to logical reasoning tasks) from data. In addition to being consistent with developmental evidence on the emergence of logic, this architectural choice simplifies the learning problem and thus allows us to study the phenomena we aim to study without needing to train extremely large models, as would be the case if we took a tabula rasa approach.

We use the Object-Permanence Network (OPNet) model described by Shamsian, Kleinfeld, Globerson, and Chechik [2020]. OPNet is an LSTM trained on video data that tracks objects as they move and are occluded by other objects. It thus learns to track and predict object locations without a built-in representation of object physics. Specifically, OPNet consists of a) an object detection module that segments a frame into visible elements (rather than use this module, we feed in ground truth bounding boxes), b) an LSTM that attends to the scene and produces a distribution over current objects to identify which the target object (in our case, the ball) is currently behind, if any, and finally c) a "where is it" LSTM that predicts a bounding box for the target object based on a weighted average of components a) and b).  

\[^3\]The loss in the original OPNet model consists of MAE compared to the ground truth location of the ball, and a "consistency error" that penalizes the model for moving its prediction from one frame to the next. In order to ensure that the model can update its predictions during the Two-Cup task, we remove the consistency loss.
Training

In our experiments (§4.4.1, 4.4.2, 4.4.3), we consider a variety of training conditions in order to determine which, if any, allow the model to encode an implicit negation and disjunction to succeed on the Two-Cup task. In all experiments, we require at a minimum that the model has been exposed to the basic elements of the Two-Cup task, so that the visual appearance of this task is in-distribution. We also require that the evaluation task itself (namely, the sub-sequence of frames (e)–(h) in Figure 4.1) remains entirely unseen in training, to ensure that the model cannot exploit specific heuristics of that sequence in order to solve the task. With these constraints, we design three sets of training data schemas, described below in Section 4.4. Each training set contains 5,000 videos of 57 frames each.

Physical Reasoning Schema We generate simple scenes which show the basic physical phenomena of the environment. Each example has a randomly selected template, each of which enumerates different interactions the ball may have within the environment; e.g. it falls into the cup, it falls onto a wedge and then into a cup, it falls onto a wedge and then rolls along the ground, etc. In each template, the ball falls from a randomly chosen point on the x-axis (though the height from which it falls is kept constant) at a random time within the first 40 frames of the video. Randomly generated occluders appear throughout the scene and may or may not block the path of the ball. Attributes of the cups and wedges (e.g. x-location, size, height) are also chosen uniformly at random from pre-set ranges. When the ball rolls along the ground of the scene, it will stop at a randomly selected x location.

Two-Cup Ablations Training Schema We construct a set of training examples which are designed to expose the model to the general concepts necessary to solve the Two-Cup task, without giving it access to heuristics it could use to succeed on evaluation without representing implicit logic. Specifically, we considered all variations and perturbations of the Two-Cup task’s sequence of events, e.g., varying the number of cups, number of occluders, and the order in which the occluders rise and fall. We then filter out any sequence of events which either (1) were logically equivalent to the Two-Cup task and thus would prevent the Two-Cup tasks from being held out or (2) had direct overlap with the true Two-Cup task’s frames at and after the critical frame, even if they did not require the same logical reasoning. For example, if in Figure 4.1 we changed the order of the events in frames b–d from “small occluder rise”, “large occluder rise”, “ball fall” to “ball fall”, “small occluder rise”, “large occluder rise” (d–b–c), it would not be permissible in this training set because frames e–h remain the same as the Two-Cup task. We found that 38 perturbations of the Two-Cup Scene were permissible. For each video made using this procedure, we randomly select a permissible perturbation, and vary elements such as cup size, wedge size, and x-axis of the midpoint of the scene, but keep the order of events and location of the ball constant.

Invisible Displacement Training Schema Finally, we generate a training set which exposes the model to the same logical reasoning pattern that occurs in the Two-Cup task, but which is visually distinct. We use

\[ \text{This constraint ensures compatibility with the Invisible Displacement training schema. In the examples derived from the Two-Cup task, the ball does not roll along the ground.} \]
this training data in a subset of our experiments (§4.4.2) in which we seek to provide the model with a better chance at passing the Two-Cup task without training on the Two-Cup task itself. To generate such a training set, we use a variation on Piaget’s Invisible Displacement experiments (§4.2.1). In our Invisible Displacement data, as shown in Fig. 4.2, the ball rolls off of a wedge and then behind two occluders that are positioned directly next to each other. The ball stops at a random point behind the occluders (similar to the procedure in the Physical Reasoning schema), and its location is ambiguous until one occluder falls, revealing that the location behind it is empty.

![Figure 4.2: Frames from the Invisible Displacement task within our experimental framework. At frame b, because of the random stopping, it is ambiguous whether the ball has landed behind the yellow or the red occluder. At the critical frame c, it becomes possible to infer the ball’s location.](image)

4.3.2 Proof-of-Concept

Our primary studies (§4.4) ask whether neural network models can learn implicit negation and disjunction in the context of physical reasoning tasks. In this section, as a proof-of-concept, we demonstrate that such implicit reasoning is possible. Using a model that has a built-in (rather than learned) physics engine, we illustrate that a model with an explicit representation of physical dynamics is able to reason using implicit negation and disjunction, and can succeed on the Two-Cup task.

Model

We use a pared-down version of ADEPT (Smith et al., 2019), a model of humans’ object tracking as constrained by their intuitive physics. This model is not trained and works out-of-the-box. ADEPT uses a Bayesian filtering algorithm to represent beliefs about the locations of occluded objects, and probabilistically represents potential future trajectories of those objects using a built-in physics simulator. This model is not trained and works out-of-the-box. For a video sequence $x_1, x_2, \ldots, x_T$, the model receives ground truth object segmentations of bounding boxes around the visible objects in the scene at each frame $x_i$. Objects that are occluded are not included.

An important feature of ADEPT is its ability to maintain beliefs about multiple possible worlds, which is done explicitly using a particle filter. ADEPT maintains a set of “particles”, each of which is a representation
of the complete scene at time $t$. The next step of each particle is generated using a noisy internal physics simulator. After several steps, the particles are resampled based on their likelihood, which is calculated based on the overlap between the particle’s representation of the world and the observations.

We evaluate the accuracy of the ADEPT model on the Two-Cup task data using 100 particles. We calculate the likelihood of each particle at the critical frame, and then group all particles based by where they represent the ball in the scene. If a particle represents the ball as being in the same cup that the ground truth ball is in, that particle is coded as “correct cup”, if it’s in the other cup it is coded as “incorrect cup”, and anywhere else on the screen is “elsewhere”. We then sum the likelihood of the particles in each of the three groups to arrive at a final score.

**Results and Discussion**

Our results are shown in Figure 4.3. Before the critical frame, the ADEPT model is uncertain about the location of the ball (frame $f$), and its belief is distributed roughly evenly over the two cups. However, after the critical frame, its belief is shifted nearly entirely to the correct cup (frame $g$). This pattern of results is consistent with the negation and disjunction logical inferences, and the ADEPT model thus succeeds at the Two-Cup task.

![Figure 4.3: Key frames of the Two-Cup task, visualizing the particles of the ADEPT model. There are 100 particles, and each circle is one particle’s representation of its beliefs about the ball’s location.](image)

These results illustrate how implicit representations of negation and disjunction can work. The model succeeds at the Two-Cup task, but it does so using a representational mechanism specific to the physical reasoning domain (here, the particle filter). The disjunction is implicit by virtue of all the particles being simulated in either cup at the same time, prior to the reveal of the empty cup. The negation is then implicit in how ADEPT responds to the empty cup. The probability of resampling any particle in which the ball had been in the empty cup is reduced to zero. Clearly, this mechanism could not readily transfer to other logical reasoning tasks (e.g., linguistic reasoning) in the way a formal logical operator could. These results thus illustrate a plausible way in which neural network models might learn to solve the same task by modeling the physical world. Namely, if the neural network learns to represent the physics of objects’ motion, as well as a representation of uncertainty and resampling in the face of unexpected observations, it should be able to pass the Two-Cup task.

---

5We experimented with many values of particle count, and found that the choice did not affect the results as long as there was a reasonable chance that at least one particle would end up in each cup.
### 4.4 Experiments

We present three experiments, which get progressively easier, each requiring less generalization from the model than the previous. This experimental structure is due to the fact that we observe negative results in the earlier experiments, and thus progressively relax requirements in order to understand the conditions under which the model can succeed (if at all). First, we train OPNet only on a basic object tracking task, and evaluate whether such training is sufficient for the model to encode an implicit negation and disjunction, as measured by the Two-Cup task (§4.4.1). Second, we train on the Invisible Displacement task (§4.2.1) in addition to basic object tracking (§4.4.2). Finally, we consider a transfer learning paradigm, in which the model is trained first on the Invisible Displacement task and then fine-tuned on the Two-Cup task (§4.4.3).

#### 4.4.1 Experiment 1: Zero-Shot Transfer from Object Tracking

We first test whether a model trained only on object tracking will acquire a capacity for implicit logical reasoning that is sufficient to generalize directly to the Two-Cup task. We train an OPNet model on Physical Reasoning + Two-Cup Ablations data. To compare this model’s performance to a plausible ceiling benchmark, we also train a model on a separate dataset of 5,000 Two-Cup task videos, which are drawn from the same distribution as the held-out Two-Cup test set and do not follow the same generation constraints as the other training datasets. We compare the accuracy of the two models to “hypothetical success” on the Two-Cup task – before the critical frame, the ball could be in either cup, but after the critical frame in which one cup is revealed to be empty, an ideal model will always predict that the ball is in the other cup. We select the model that minimizes loss on the dev set (100 examples for Physical Reasoning + Two-Cup Ablations data, 50 examples for Two-Cup data).

### Control Experiments

In order to ensure that the experimental model has learned to track objects in this environment, we also test it on several control datasets. These control datasets are permutations of the Two-Cup task and are intended to ensure that the model has succeeded in encoding the basic physical knowledge (other...
than the logical reasoning patterns) on which the Two-Cup task depends. First, Two-Cup Freebie is designed to rule out failure based on the task setup. This test set contains the same setup as Two-Cup; but the first revealed cup shows the ball, thus removing the need to reason by exclusion. Next, to ensure that the model can track the ball’s location, even when occluded, we create the Known Specific Location and Known Cup Location test sets. In the former, we reorder the frames in Fig. 4.1 such that the ball falls, then the occluders rise, before proceeding like normal. This should give away the ball’s location before any inference is necessary. Similarly, in the latter, to ensure that the model can track an occluded ball with only general knowledge of its location (i.e. it knows the ball is in a cup, but not exactly where), we reorder the frames such that the small occluders rise, then the ball falls, then the scene proceeds. Finally, to ensure that the model is tracking the ball and not just guessing a cup, we create the Outside control set. In this set we use the Two-Cup setup, but the ball falls entirely outside of the cup-and-wedge structure. There are two settings of this set: occluded, where it falls behind a third occluder, and visible, where it is never obscured.

The experimental model succeeds with high accuracy (88%-100%) on each of the control sets, as shown in Table 4.1.

### Two-Cup Task Experiment

As shown in Table 4.1, the predictions of the experimental model (Physical Reasoning + Two-Cup Ablations) do not change after the critical frame. In each test video, the occluder falling to reveal the empty space does not cause the experimental model to update its prediction. Comparatively, when the model is trained directly on the Two-Cup task, it predicts the correct cup after the critical frame roughly 81% of the time. These results show that the neural network model trained only on object tracking within the physical constraints of the Two-Cup setup does not learn an implicit representation of the logical operators required to pass the Two-Cup task.

#### 4.4.2 Experiment 2: Zero-Shot Transfer from Reasoning-by-Exclusion Task

One explanation for the failure observed in the above experiment is that the models never encountered a situation in which the ball could be in either of two, non-contiguous locations at training time, and thus never learned to represent and reason about this form of uncertainty, or to update its beliefs about the location of the ball once the uncertainty is resolved (as the ADEPT model from 4.3.2 was able to do). To test this possibility, we provide the model with much richer training data, training it on the logical structure of the task without giving away the visual details of the solution. We train an OPNet model not only on our original Physical Reasoning + Two-Cup Ablations data, but also on Invisible Displacement data, which requires the same inferences as the Two-Cup task but uses a visually different setup. We then observe that model’s performance on the Two-Cup task.

#### Control Experiments

Similarly to the Physical Reasoning + Two-Cup Ablations experiments, we also introduce control experiments in order to rule out the possibility that the model fails to learn the necessary object tracking. We test the Physical Reasoning + Two-Cup Ablations + Invisible Displacement model on all of the earlier control sets, plus three designed to test the model’s ability to learn from the Invisible Displacement training task— Invisible Displacement Test, Invisible Displacement Freebie and Roll-then-occlude. Invisible
Displacement Test is held-out test data. In Invisible Displacement Freebie, to rule out failure in the object tracking required for the Invisible Displacement task itself, when the first occluder drops in the Invisible Displacement task, the ball is found. Finally in Roll-then-occlude, similarly to the “Known Specific/Cup Location” experiments, we test to see that the model can track an observed ball behind occlusion. The ball rolls and comes to a stop before the occluders rise, and then the scene progresses like normal.

The model succeeds at the control experiments, (including the Invisible Displacement test), scoring between 95-100% accuracy on all datasets.

Two-Cup Task Experiment  Again, as in the prior experiment, the model performs at chance on the Two-Cup test. The neural model makes the same errors as in the other tests: when the occluder falls, it does not update its prediction. Even though the model is trained on examples of the exact reasoning pattern on which it is tested, it is still unable to generalize to the Two-Cup task. There are at least two possible explanations. First, the failure might suggest that the model did not learn to solve the Invisible Displacement task in the “right way”, and thus despite learning to solve that task at training time, the learning does not result in representations that are reusable on the Two-Cup task. A second explanation is that the representations learned do capture the logical structure of the task, but the model is not able to recognize the similarity between the tasks in the zero-shot transfer setting. This latter failure would suggest that the neural network could succeed with additional training focused on recognizing visually novel instantiations of the same logic.

4.4.3 Experiment 3: Transfer Learning Across Reasoning-by-Exclusion Tasks

In our final experiment, we test whether the neural models that failed to reason by exclusion might nevertheless be learning some representation relevant to the logical structure of both tasks. If they do, pretraining on Invisible Displacement and then finetuning on the Two-Cup task might allow it to succeed more quickly at the held-out Two-Cup test than training on the Two-Cup task alone. To examine this, we compare how quickly OPNet models begin to succeed on the Two-Cup task when first trained on Invisible Displacement to one model trained on the Two-Cup training data initialized from scratch, as well as to a control model pretrained only on the Physical Reasoning schema.

Figure 4.4 shows the results. We find that the model that is first trained on Invisible Displacement and then trained on the Two-Cup training data succeeds at the held-out Two-Cup test set before both a model that is trained on the Two-Cup task from scratch and a model trained only on object tracking.

4.5 Discussion

This work asks whether neural network models are capable of learning *implicit* representations of logic from self-supervised training in the domain of physical reasoning. Overall, the results in Section 4.4 are primarily negative. We see in 4.4.1 that the neural networks trained on a basic physical reasoning task fail to generalize to the Two-Cup task. More importantly, in 4.4.2 we see the same models failed even when trained on the Invisible Displacement task, a task which should in principle rely on exactly the same conceptual representations as does the target Two-Cup task. Thus, the evidence strongly suggests that the
neural networks are learning to solve the training tasks using representations other than those which the tasks are designed to require. However, in Section 4.4.3, we see suggestive evidence that the neural network might learn some desirable representations related to logical reasoning, demonstrated by the fact that training on one logical reasoning task speeds learning on another logically identical task. While this falls short of our test for possession of an (implicit) logical representation, it does indicate that the network may be capable of developing the desired representation, e.g., under different training conditions.

4.6 Related Work

As a proof-of-concept, the ADEPT model we analyzed in §4.3.2 was able to succeed on the Two-Cup task. This indicates that, at a minimum, a good model of the physical dynamics of the environment combined with a good model of possible worlds is sufficient to deploy implicit logical operators for reasoning. Thus, when we observe that the neural network models fail on our evaluations, it implies that they are failing to learn at least one of the relevant components of ADEPT’s model of intuitive physics. Our control conditions presented in Sections 4.4.1 and 4.4.2 suggest the model learns the physical dynamics of the environment reasonably well. Thus, we are inclined to conclude that the neural model fails because it does not learn to represent uncertainty over multiple possibilities and/or lacks mechanisms for updating beliefs about possibilities in response to observations. Future work considering alternative model architectures and loss functions may well reveal more positive results.

Intuitive physics is thought to be one of the domains of “core knowledge” in human cognition, which are the foundations of higher-level mental processes (Spelke & Kinzler, 2007; Carey, 2009). In recent years, computational approaches to intuitive physics problems have gained popularity—Duan, Dasgupta, Fischer, and Tan (2022) have conducted a comprehensive survey. We focus on models with object-centric representations (Smith et al., 2019; Shamsian et al., 2020).

There are several datasets of intuitive physics tasks for computational models to train and test on. IntPhys
2019 is a benchmark dataset, on which models are trained only on positive examples; the test task is then to distinguish examples that do and do not conform to the physics of the environment (Ricochet et al., 2018). Published concurrently is Piloto et al. (2018), which takes a similar approach. Rather than distinguishing consistent/inconsistent image sequences, the PHYRE task is a dataset of physical reasoning tasks where the objective is to take one action to manipulate the scene and reach a goal state (Bakhtin, van der Maaten, Johnson, Gustafson, & Girshick, 2019). Within the realm of object tracking in video, the CATER dataset contains videos of a target object as it is occluded and shuffled behind and within other objects in a three-dimensional scene (Girdhar & Ramanan, 2020). Although it is not an intuitive physics task per se, similarly to this work, the ACRE dataset borrows the “blicket detector” task from developmental psychology in order to measure the ability of neural network-based and neurosymbolic reasoning models at the task of causal induction, or inferring causality from limited data (Zhang, Jia, Edmonds, Zhu, & Zhu, 2021).

4.7 Conclusion

In this work, we measured neural models’ ability to learn implicit representations of negation and disjunction from data. We found that, despite having a good representation of the physics of the environment, the neural models were not able to generalize to a diagnostic task requiring implicit negation and disjunction without observing examples from that diagnostic task itself. Furthermore, when trained on visually dissimilar data that requires the same logical capabilities, transferring the representations to the target task did not improve performance. However, when training on the diagnostic task, pretraining on a similar logical task caused the models to learn the target task faster than other initialization methods. These results show a potential weakness of models that attempt to learn and generalize logical capability from data.
Chapter 5

Transformer Mechanisms Mimic Frontostriatal Gating Operations When Trained on Human Working Memory Tasks

5.1 Introduction

Computational models based on the Transformer architecture (Vaswani et al., 2017) have seen success on a wide variety of tasks that appear to require complex “cognitive branching”: the ability to maintain pursuit of one over-arching goal while performing other subtasks along the way. For example, Transformer-based large language models (LLMs) have demonstrated impressive abilities in not just language (Brown et al., 2020), but planning (Huang et al., 2022), navigation (Du et al., 2021), and problem solving (Lewkowycz et al., 2022).

In humans, performance on such tasks is known to depend on a neural mechanism for gating (Frank & Badre, 2012; Badre & Frank, 2012; Chatham et al., 2014; Rac-Lubashevsky & Kessler, 2016; Rac-Lubashevsky & Frank, 2021), which controls whether new information is maintained in working memory or not, the address in memory where it is stored, and the address from which stored information is recalled in response to a task. Typical Transformer models have no specialized architecture for working memory, in spite of their ability to succeed at tasks which appear to require it. Although some Transformer models have additional built-in structure for memory (Dai et al., 2019; Burtsev, Kuratov, Peganov, & Sapunov, 2020), recurrence (Dai et al., 2019), or hierarchy (Y.-S. Wang, Lee, & Chen, 2019), vanilla Transformer models without any such inductive biases remain the dominant architecture for the modern AI systems which are especially visible in their success on complex tasks (Brown et al., 2020; Touvron et al., 2023). This raises the question: in solving such tasks, does a mechanism for selective input and output gating emerge within the vanilla Transformer?

Transformers are good candidates for learning gating behavior because of inductive biases within the
self-attention mechanism, i.e., the Transformer’s defining architectural component. Within self-attention, numerous “attention heads” construct contextual representations for each item in their input sequence through a learned weighted combination of the previous items in the sequence. Attention heads could in principle learn gating behavior by marking sequence elements with a key (input gating) and reading out those values later by querying for those keys (output gating). Moreover, the attention mechanism in Transformers is decomposed in a way that enables it to readily differentiate “reading” and “writing” operations. This behavior is analogous functionally to the neurobiological roles of corticostratial circuits in humans and other animals, in which isolated clusters of prefrontal neurons represent distinct “addresses” in memory that can be updated or read out from via selective gating actions triggered by basal ganglia and thalamus \cite{O'Reilly2006, Frank2012, Calderon2022}. In computational neuroscience models of this process, the prefrontal clusters (or “stripes”) can also serve as latent abstract “roles” that condition how to interpret content within them, affording functions such as variable binding, indirection, and hierarchical generalization to new situations \cite{O'Reilly2006, Collins2013, Kriete2013, Bhandari2018}. Thus, Transformers may use their attention heads to learn a gating strategy that mimics certain functions of the brain.

In this work, we train vanilla Transformer models with self-attention on a working memory task paradigm that was specifically designed to evaluate models of selective gating and working memory in computational neuroscience \cite{O'Reilly2006, Rac-Lubashevsky2021}. We use recent techniques from mechanistic interpretability \cite{Olah2022, Nanda2022} to expose the mechanism that the Transformer uses in order to perform the task. We find that, as a result of training, the self-attention mechanism specializes in a way that resembles existing models of input-output gating. Specifically, we find that the trained model uses the key vectors within the attention mechanism to control input gating, i.e., determining which elements in the input to consider vs. ignore, as well as controlling how the information is stored– in other words, assigning it a role such that the model can access it later. The model uses the query vectors to control output gating, i.e., determining which information is accessed in order to complete a task. Our findings highlight the importance of considering the emergent mechanisms that result from training in addition to the innate architectural mechanisms when drawing comparisons between AI systems and human cognitive processes, and opens the door for future analysis and work which can enable more principled studies of the similarities and differences between human vs. machine cognition.

### 5.2 Background

#### 5.2.1 Gating Mechanisms

Working memory in human brains is known to make use of a gating mechanism, which processes and stores information roughly analogously to gates being opened and closed \cite{Rac-Lubashevsky2016, Rac-Lubashevsky2021, Bhandari2018}. The input gate controls which information is stored or not stored in memory, and if stored, into which “address”. The output gate controls which content within working memory is accessed in order to produce a response or to make subsequent gating operations. Both input and output gating policies are also dependent on the learned task-dependent context (i.e. role) of
the information to be stored and accessed in working memory.
In cognitive neuroscience, a variety of tasks are used to study the capacity of working memory. In this work, we focus on a variant of the “reference-back 2” task (Rac-Lubashevsky & Frank, 2021), a human paradigm meant to mimic a task designed to showcase the need for selective gating of independent contents of information in frontostriatal neural networks (O’Reilly & Frank, 2006). In the reference-back paradigm, symbols such as letters or numbers are viewed one at a time, and the subject must determine whether the current symbol is the same or different as that stored in memory for a given role (letter or number). They also are given a cue to indicate whether to update the current symbol as the “reference” to be compared on subsequent trials of the same role, or if instead they should continue to maintain the previous reference. Thus this task requires selective updating and accessing of information in a role-addressable manner.

Limitations as Cognitive Models: Transformers have properties which make them obviously bad models of human sequence processing. In particular, because Transformers can attend to any part of the sequence when creating a representation, they are not limited by memory representation constraints. Transformers could thus solve tasks that would push the limits of human working memory, but it should be noted they accordingly require large amounts of training data. The question addressed in this work is orthogonal to these limitations. That is, we focus specifically on if and how Transformers can learn to implement an efficient gating mechanism to solve tasks with human working memory demands.

Figure 5.1: Graphical diagram of the path-patching process. Attention heads are represented as circles (layer, head index), and contextual representations of each token (as well as the next token prediction) are represented as rectangles.
5.2.2 Mechanistic Interpretability

We use a set of recently introduced analysis tools ([Elhage et al., 2021]) which enable us to uncover specific mechanisms defined in terms of model weights within the Transformer. Specifically, we use path-patching ([K. Wang, Variengien, Conmy, Shlegeris, & Steinhardt, 2022; Goldowsky-Dill, MacLeod, Sato, & Arora, 2023]), a generalization of causal mediation analysis ([Pearl, 2001]) that allows us to determine which components of a neural model (e.g., attention heads) work together in order to produce observed behavior on a task. The discovered components are referred to as a circuit ([Käuker, Ho, Casper, & Hadfield-Menell, 2023]).

Path-patching involves making an incisive edit to the representations of a trained model and observing how the model’s behavior is affected (see Fig. 5.1). Path-patching typically requires a minimal pair of examples: the “clean” example and the “corrupted” example, in which one token from the clean example is changed, as well as the correct label. Given representations from the model for both the clean and the corrupted examples (the blue and orange components in the figure), we can choose a specific component anywhere in the model (referred to as the “sender”), and insert the corrupted representation at that component into the clean representation. We then use the model to recalculate the representations up until another component of the model (the “receiver”), thus “patching” the path. In the figure, we send from layer 0, head 0 to layer 1, both heads 0 and 1. All clean representations that are not along this path are not modified and are unaffected by the patch. The model then recomputes all representations after the receiver (the “patched” representations), and arrives at a new prediction. If the model output matches the corrupt prediction rather than the clean one, that prediction is causally dependent on the path from sender to receiver. See ([K. Wang et al., 2022] and [Goldowsky-Dill et al., 2023]) for a more comprehensive review of path-patching methods.

Figure 5.2: Above: example of textual reference-back task as model input. Below: step-by-step task process; models do not view task-internal grey words. “Update Instruction” executes after “Answer” despite appearing earlier sequentially.
5.3 Task

We create a modified text-based version of the reference-back 2 task (Rac-Lubashevsky & Frank, 2021) designed to tax selective WM gating (O’Reilly & Frank, 2006). The textual reference-back task requires making same/different judgments between incoming symbols assigned to a particular “register” in memory, with respect to those seen previously and linked to those same registers. Like the original tasks, the textual reference-back task is sequential, and requires the maintenance and independent updating of two memory registers, each containing one of $S$ arbitrary symbols at a time. The contents of the registers are updated over the course of the task. At the beginning of each sequence, each register is initialized individually to one symbol $s \in S$ (the pool of symbols is shared between registers, which was shown to more substantively tax gating mechanisms in O’Reilly & Frank, 2006). Each sequence is composed of $L$ tuples, each containing register address $Reg_i$, symbol $Sym_i$, same/different label $Ans_i$, and update instruction $Ins_i$. For a tuple $i \in L$, the answer $Ans_i$ is a binary value that is either same if symbol $Sym_i$ is currently stored in the register with address $Reg_i$, or different otherwise. The update instruction $Ins_i$ also takes one of two values, evenly distributed: if ignore, then there is no effect further on in the sequence. If store, then from that point on in the sequence, $Sym_i$ is stored in the register with address $Reg_i$ until otherwise updated. An example is shown in Fig. 5.2.

We implement each reference-back task example in our data as a single sequence, and measure models’ ability to predict same versus different for each $Ans_i$. Each sequence has 10 same/different answers, and we generate 100,000 train, 1,000 dev, and 1,000 held-out test sequences.

The class balance of same to different answer labels in the train/test datasets is roughly 1:2, making a “maximum class” heuristic solution 0.66 accuracy, 0.33 precision, and 0.5 recall. We test several other heuristics, the strongest of which is predicting same if another tuple including Store and the target register and target symbol exists in the sequence, which scores 0.80 accuracy, 0.82 precision, and 0.85 recall.

5.4 Model

We use vanilla Transformers in order to facilitate interpretability, as done in prior work that analyzes emergent mechanisms (Elhage et al., 2022). Our models contain two decoder-only layers, each with only two heads (four in total), and no multilayer perceptrons or layer normalization, followed by a linear “unembed” layer to project the output of the last decoder into the space of the entire vocabulary at each timestep\textsuperscript{1}. Our network uses absolute positional embeddings (Vaswani et al., 2017). The vocabulary contains all possible tokens, represented individually with embedding size $E$. Models are trained to predict the next token with the language modelling objective, meaning if the model is predicting $Ans_c$, it will have access to all $(Ins_i, Reg_i, Sym_i, Ans_i)$ tuples where $i < c$, as well as $Ins_c, Reg_c$, and $Sym_c$. However, the models only receive loss at positions where a same/different token must be predicted. Furthermore, each layer gets a causal attention mask– when constructing each token representation, it cannot look ahead at tokens further down the sequence.

The models are trained over 60 epochs of the 100k training data points, learning from 6 million examples in total. Models are evaluated on their accuracy (whether the correct $Ans_i$ is predicted for each tuple $i$), measured

\textsuperscript{1}In practice, only ‘same’ and ‘different’ are ever predicted.
in precision and recall, as well as the same versus different token logit difference.

5.5 Experiments

First, we select and analyze a single Transformer model which succeeds on the task, and upon investigation of its weights find that it learns a mechanism for input/output gating. Second, when we conduct a search over more trained models, we find that model performance correlates with markers of learning a gating policy, analogous to findings in the frontostriatal neural networks (Frank & Badre, 2012).

![Figure 5.3: Model behavior when predicting same/different (token 15) is shown. We measure attention visualized as a shade of purple, with deeper shade corresponding to higher attention to that token. We create “corrupted” minimal pairs in which changing a token (light blue) either changes the correct label at index 15 (examples b, c, e) or does not (d, f). We make small path-patching edits with the minimal pair to targeted network components (layer 1 keys for b, c, d, f; queries for e,f). In other words, we replace specific components (denoted with red text) with their corresponding representation from the “corrupted” sequence, but hold all other representations constant, and run the model and get a new same/different prediction. In all test examples, making the small patch successfully results in the model’s prediction changing to align with the “corrupted” example.](image)

We first establish that a Transformer model is able to succeed on the reference-back task. We perform a small hyperparameter search and select a model that reaches 100% accuracy on the held-out test data for further analysis. We determine the circuit that the model uses to solve the textual reference-back task through an array of path-patching experiments with a simplistic minimal pairs paradigm. Our “sender” within path-patching is always both attention heads at layer 0, and our “receiver” is always both attention heads at layer 1.

At layer 0, the model learns to condense the task-critical information from each tuple into one embedding, at the position for Sym\textsuperscript{i}. At this layer, the model pays 85.8% of total attention to the task-critical information to that tuple, and just 14.2% of attention to other tuples.

Redundantly, the model does the same at the position for Ans\textsuperscript{i}. Through additional experimentation, we determine that this is a quirk of Transformer learning, and does not impact our analysis.
“stored” tuple), and only 29.8% of attention to all other tokens. This behavior is tied to the target register
matching the register in the stored tuple, which is analogous to gating of the relevant role-addressable PFC
stripe. We focus our analysis on the Layer 1 representations which exhibit this learned gating policy, shown in
Fig. 5.3.

### 5.5.1 Input Gating through Key Vectors

Input gates in working memory control what incoming information is remembered and “role-addressable” — i.e.,
stored in memory in such a way that it is able to be freely accessed later when it is needed for task completion.
In a Transformer, the key vectors serve the role of addresses (analogous to PFC “stripes”), which are retrieved
based on their match to a query vector from the current or later timesteps during self-attention. Thus the input
gating in Transformers is controlled through key construction; the composition of the output of the Layer 0
attention controls which content is stored in the key vector at Layer 1 for later use. At a later timestep, a query
vector will address the information in the key vector.

In our model analysis, we find that key composition at the Symi position (positions 2, 6, 10, and 14 in
Fig. 5.3) roughly represents each tuple. A query’s ability to address this position depends on whether the
represented tuple contains a Store or an Ignore. Key vectors representing an Ignore tuple receive very little
attention (0.4% of layer 1 attention averaged over test set), whereas those representing a Store tuple receive
the bulk of the attention (86.8%). We determine this effect causally with path-patching (Fig. 5.1). First, we
create clean sequences sampled from our test set, and then corrupt these sequences by switching a Store
within tuple i to an Ignore. We then path-patch only the key vectors of i. We expect, if the key controls input
gating, that patching these key vectors should “block” attention to all of tuple i. An example attention pattern
is in Fig. 5.3 examples a and b. We find that the model’s attention shifts away from the tuple accordingly in
100% of patched instances. The presence of an Ignore or a Store within a tuple controls whether the key
construction acts as an open input gate or a closed input gate.

Key construction also depends on the role of the represented content; within our task, that means whether
Regi is Reg0 or Reg1. When making a same/different prediction, key vectors representing a tuple that matches
the target register receive most of the model’s attention (92.5% of total attention), while those that do not match
are not attended to (3.3% of total attention). Similarly to input gating, we use path-patching to determine that
key construction encodes roles. This time, given a target tuple i with a target register, we corrupt the register
of the stored tuple, changing it from Reg0 to Reg1 or vice versa; to predict the answer for i, the model must
attend to an earlier tuple with the target register. An example is Fig. 5.3 row c. The model’s attention shifts
away accordingly across every example in the test set. Note that the stored tuple must be modified; if the same
corruption is made earlier (as in row d), attention does not shift. This behavior shows that the gating within
self-attention is role-addressable: the registers within the task function as roles, and are embedded within the
key vectors as part of the representation.
5.5.2 Output Gating through Query Vectors

Within working memory, output gates control which addressable information is accessed in order to complete a task. Given that key vectors serve the role of addresses, query vectors in turn control which key vectors are accessed, through the final Q*K dot product in attention. Query construction thus performs the role of output gating within Transformers.

The query composition controls which addressable Sym representations are attended to based on the identity of the target register; changing Reg1 to Reg0 controls which role-addressable content is accessed. We determine this through a final set of path-patching experiments, an example of which is shown in Fig. 5.3, row e. Rather than editing the register in the stored tuple as done in row c, we corrupt the target register itself; this means that the query must now find representations corresponding to the other tuple. In row e, the model finds Sym5, and predicts different; and in row f, we patch in Sym4 at index 2, and the model predicts same. Upon inspection, the query is constructed to correspond with key vectors that represent tuples which also contain the target register. When we edit the target register in minimal pair experiments, we observe that the attention shifts from the original stored tuple (74.1% of attention) to the stored tuple that matches the edited register, and successfully makes an updated same/different comparison to the symbol in the edited register in every instance.

Editing aspects of the target tuple other than target register has minimal effect on the query construction behavior. No edits to the query cause the model to attend to a Ignore tuple, further evidencing of output gating behavior– only content that has been made “addressable” can be accessed for a response. Furthermore, we find that the target instruction and symbol do not factor into the query composition– changing them through path-patching to the query does not affect attention. This is notable because the model could employ other strategies for determining which tuples are eligible to be the stored tuple; e.g. attending to all symbols to match if any of them are the same as the target symbol.

5.5.3 Successful Task Performance is Related to Discovering Gating Policies

To further identify how readily Transformer models learn a gating policy, and how useful such a policy is to succeed on the task, we train new models with the same hyperparameters across many different random seeds, and measure their performance on the target task as well as on markers of the gating policy. We train 20 new models on the same textual reference-back data, each with a different random initialization, and measure both training loss and test set accuracy. 5 of the models succeed 100% of the time, and the other 15 models succeed between 94%-99.99% of the time, with a mean of 97.72% and a standard deviation of 2.03. The models are trained on the same amount of data (6 million examples).

We observed in the prior sections that the trained Transformer model uses its key composition to control input gating and its query composition to control output gating and role addressability. To identify whether the new models learn to gate similarly, we evaluate the key and query composition of all 20 new models by making minimal pair path-patching edits for every test example, where the answer changes from same to different or vice versa, similarly to Figure 5.3.

To evaluate the ability to open and close input gates, we corrupt the stored tuple’s Store to Ignore, and
path-patch only to the stored tuple’s key vectors. To evaluate the role addressability of the content during output gating, we corrupt the target register (changing Reg0 to Reg1 or vice versa), and path-patch only to the query vector for making the same/different judgment. This is a more challenging task than patching to all of the keys or queries respectively; a model will only succeed on these subtasks if it implements a gating policy with the same markers that the model analyzed in earlier sections does. A model makes the patched prediction successfully if its prediction matches the corrupted sequence and not the original sequence—i.e. the targeted path-patch was sufficient to change its same/different prediction.

Figure 5.4: Model performance over training on patching subtasks. Each row corresponds to an individual model’s training loss (solid line) and subtask accuracy (dashed line) over time; blue and orange lines respectively correspond to models which reach 100% test accuracy and to those that do not.

We visualize the 5 runs that reach 100% accuracy on the test data in Figure 5.4, as well as 5 randomly selected runs that do not reach 100% accuracy on the test data, and plot their training loss versus their accuracy (between 0 and 100) on the two patched subtasks over the course of training.

Two trends become apparent from the data: first, models which score a perfect test accuracy appear to succeed at the subtasks more readily than models which do not. Of the former, 3 of 5 models reach 100% accuracy on both subtasks readily, plateauing less than halfway through training duration. However, examples from the latter category of models do not reach such immediate success at the patched subtasks (including the 10 not pictured in this graph); in fact, many categorically fail, scoring as low as 49% accuracy. These results do not indicate that this class of models’ representations are useless for the task— they all score between 94% and 99.99%, well above heuristic performance. Failing to succeed at the targeted patching subtasks reveals that these models may implement some other strategy that is not gating, and may be brittle or heuristic in some way. We take these results as evidence that learning a robust policy for gating correlates with model performance at the textual reference-back task.
The second emergent trend is that many models across both classes have a sharp decline in training loss, which correlates with a similarly steep increase in accuracy on both subtasks. Sudden jumps in performance is a noted phenomenon that has been observed in other Transformer models in cases of e.g., grokking [Power, Burda, Edwards, Babuschkin, & Misra, 2022]. We interpret this phase transition as suddenly learning a gating mechanism. Models that do not exhibit phase transitions to the same degree take longer to fit the task, and do not reach high subtask accuracy. The cause of phase transitions and what is learned during this process is left for future work.

5.6 Summary and Discussion

In this work, we investigate Transformer models for emergence of a learned gating mechanism; a network component performing role-addressable gating, similar to that in working memory of humans. We observe that the model readily learns a gating policy, and upon training more models find that task performance is correlated with gating ability. Our results show how competence at cognitive branching tasks can emerge in Transformers, and suggest that integrating Transformer components may improve existing computational neuroscience models of working memory.

The Transformer models which were trained on the textual reference-back task are capable of making use of key composition for input gating and query composition for output gating. We find that making precise corruptions to specific architectural elements of the network causes the model’s prediction to change from same to different or vice versa, indicating that those components are causally responsible for the gating mechanism. The architectural biases of attention within the vanilla Transformer model lend themselves well to representing role-addressable content, as the learnable nature of keys, queries, and values allows the model to learn to create internal representations in a manner which allows it to represent roles and addresses, mimicking the variable binding and input / output gating mechanisms in biological neural networks [O’Reilly & Frank, 2006; Frank & Badre, 2012; Collins & Frank, 2013; Kriete et al., 2013]. In this work, we focused on characterizing the mechanism that the Transformer model learns as a result of training on cognitive branching tasks, and did not evaluate the robustness of the mechanism. The textual reference-back task is similar in nature to the FFLM task [B. Liu et al., 2023], comprised of 1 register, 2 symbols (0 or 1), and long sequences. B. Liu et al. found that similar Transformer models were able to succeed on FFLM data, but struggled to generalize outside of their training distribution. We leave experiments characterizing generalization of Transformer mechanism behavior on data with the reference-back paradigm– e.g. more than 2 registers, distinct sets of symbols, novel symbols introduced at test time-- for future work.

When we trained more models on the task, we found that the models which perform best on the task correlate with the markers of gating we observed in our circuit analysis, and that the learning trajectory shows a steep decrease in training loss and a steep rise in patched subtask accuracy simultaneously, suggesting that the model has learned a component of the gating policy at that time. Both findings are analogous to those of Frank and Badre [2012], in which they find that networks which learned a hierarchical gating policy performed better at a hierarchical learning task, and humans that learn this policy also show a sharp decrease in loss when they discover it.
Ultimately, finding connections between emergent behavior of Transformer models and human working memory serves to benefit both computational cognitive neuroscience and artificial intelligence. Although Transformer models themselves are limited in their biological plausibility, in this setting they learned behavior mimicking the functionality of working memory, and their application within computational models of the brain should be further explored. From the perspective of artificial intelligence, understanding the strengths and limitations of Transformer models on cognitive branching tasks may inform model analysis across the many diverse settings in which these models are applied.
Chapter 6

Conclusion

In this thesis, we investigated the extent to which symbolic structure can emerge in prototype neural networks. We measured this within various environments where data and model were tightly controlled, and found mixed positive and negative results. First, in two separate settings— one where language models were trained on propositional logic sentences, and one where object-tracking models were trained on visual scenes— we found that despite the relative simplicity of the tasks compared to more “natural” language and vision tasks, neither representations of symbolic structure nor weaker precursors emerged from training. Next, in a setting inspired by computational cognitive neuroscience, we found that Transformer models succeeded at the given symbolic memory task, suggesting that representations of symbolic structure can emerge within models. The discrepancy between the success of Transformer models at the working memory task and the lack of success of both Transformer and LSTM models in the logical tasks can potentially be explained by how well the structure of the architecture of the network lends itself to the underlying task. Ultimately, our results show that within modern neural network models, representations of symbolic structure do not always emerge “for free” from training data, and that success on symbolic tasks cannot be expected as a result of the neural network training process. Architectural developments and/or integrated neurosymbolic network components may be necessary in order for modern neural models to achieve success on symbolic reasoning tasks similar to their success within broader language and vision applications.

References

References


framework for transformer circuits. *Transformer Circuits Thread, 1.*


References


covariation. In *Proceedings of the ieee/cvf conference on computer vision and pattern recognition* (pp. 10643–10653).