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

Aaron Traylor,[1] **Jack Merullo,**[1] **Michael J. Frank,**[2] **& Ellie Pavlick**[1]
[1] Department of Computer Science
[2] Department of Cognitive, Linguistic & Psychological Sciences
Brown University, Providence, Rhode Island, USA
{aaron_traylor, jack_merullo, michael_frank, ellie_pavlick}@brown.edu

Abstract

Models based on the Transformer neural network architecture have seen success on a wide variety of tasks that appear to require complex "cognitive branching"– or the ability to maintain pursuit of one goal while accomplishing others. In cognitive neuroscience, success on such tasks is thought to rely on sophisticated frontostriatal mechanisms for selective *gating*, which enable role-addressable updating– and later readout– of information to and from distinct "addresses" of memory, in the form of clusters of neurons. However, Transformer models have no such mechanisms intentionally built-in. It is thus an open question how Transformers solve such tasks, and whether the mechanisms that emerge to help them to do so bear any resemblance to the gating mechanisms in the human brain. In this work, we analyze the mechanisms that emerge within a vanilla attention-only Transformer trained on a simple sequence modeling task inspired by a task explicitly designed to study working memory gating in computational cognitive neuroscience. We find that, as a result of training, the self-attention mechanism within the Transformer specializes in a way that mirrors the input and output gating mechanisms which were explicitly incorporated into earlier, more biologically-inspired architectures. These results suggest opportunities for future research on computational similarities between modern AI architectures and models of the human brain.

Keywords: transformers; neural networks; working memory; computational neuroscience; gating; computational cognitive science; mechanistic interpretability

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, Abbeel, Pathak, & Mordatch, 2022), navigation (Du, Yu, & Zheng, 2021), and problem solving (Lewkowycz et al., 2022).

In humans, there is strong evidence that performance on such tasks depends 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. 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 modern AI systems (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 corticostriatal 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 (O'Reilly & Frank, 2006; Frank & Badre, 2012; Calderon, Verguts, & Frank, 2022). 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 (O'Reilly & Frank, 2006; Collins & Frank, 2013; Kriete, Noelle, Cohen, & O'Reilly, 2013; Bhandari & Badre, 2018). 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 (O'Reilly & Frank, 2006; Rac-Lubashevsky & Frank, 2021).

In L. K. Samuelson, S. L. Frank, M. Toneva, A. Mackey, & E. Hazeltine (Eds.), *Proceedings of the 46th Annual Conference of the Cognitive Science Society.*

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, 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.

Background

Gating Mechanisms

There is strong evidence that working memory in human brains makes use of a *gating mechanism*, which processes and stores information analogously to gates being opened and closed (Rac-Lubashevsky & Kessler, 2016; Rac-Lubashevsky & Frank, 2021; Bhandari & Badre, 2018). 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. 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.

Transformers

Transformers 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 complex sequence modelling tasks.

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 $\sum_k q_i^j \cdot k_k^j * v_k^j$. Thus, the next token prediction includes earlier sequential information 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. As we shall see below, the comparison to role-addressable input and output gating operations is evident, whereby the key vectors form addresses analogous to the PFC "stripes", with key construction determining input gating, and the query vectors control which of them are accessed, with query construction determining output gating.

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.

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* (Räuker, Ho, Casper, & Hadfield-Menell, 2023).

Path-patching involves making a incisive edit to the representations of a trained model and observing how the model's

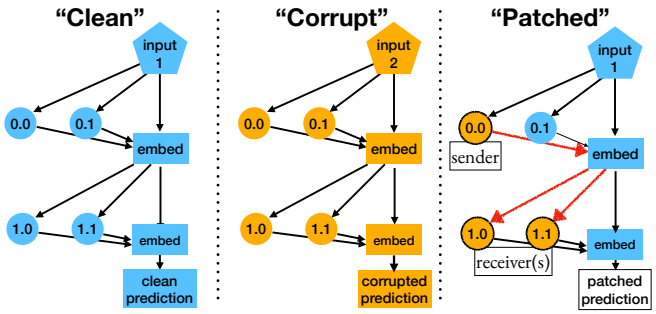

Figure 1: Graphical diagram of the path-patching process. Attention heads are represented as circles (layer,head index), and contextual representations of each token (as well as the next token prediction) are represented as rectangles.

behavior is affected (see Fig. 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 chose 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.

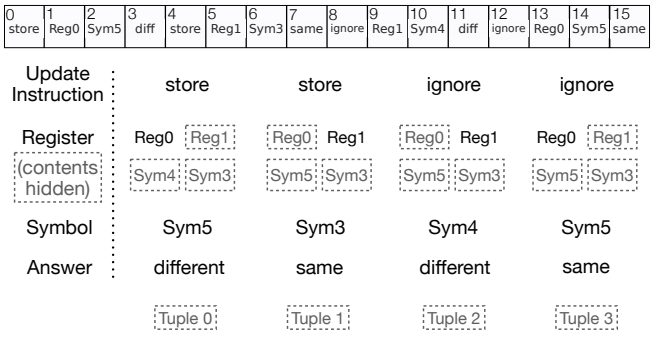

Figure 2: Above: example of textual reference-back task as model input. Below: step-by-step task process; models do not view task-internal grey words. "Update Instruction" executes after "Answer" despite appearing earlier sequentially.

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 $s \in S$ (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. 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.

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 [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

[1] In practice, only 'same' and 'different' are ever predicted.

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 in precision and recall, as well as the `same` versus `different` token logit difference.

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).

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 simple 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_i[2]. 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.

At layer 1, the attention heads learn to attend to the Sym_i key vector representing the tuple where information was last stored in the target register. The heads pay 70.2% of total attention to this tuple (the "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. 3.

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 Sym_i position (positions 2, 6, 10, and 14 in Fig. 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. 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. 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 Reg_i 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. 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.

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_i 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. 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

[2]Redundantly, the model does the same at the position for Ans_i. Through additional experimentation, we determine that this is a quirk of Transformer learning, and does not impact our analysis.

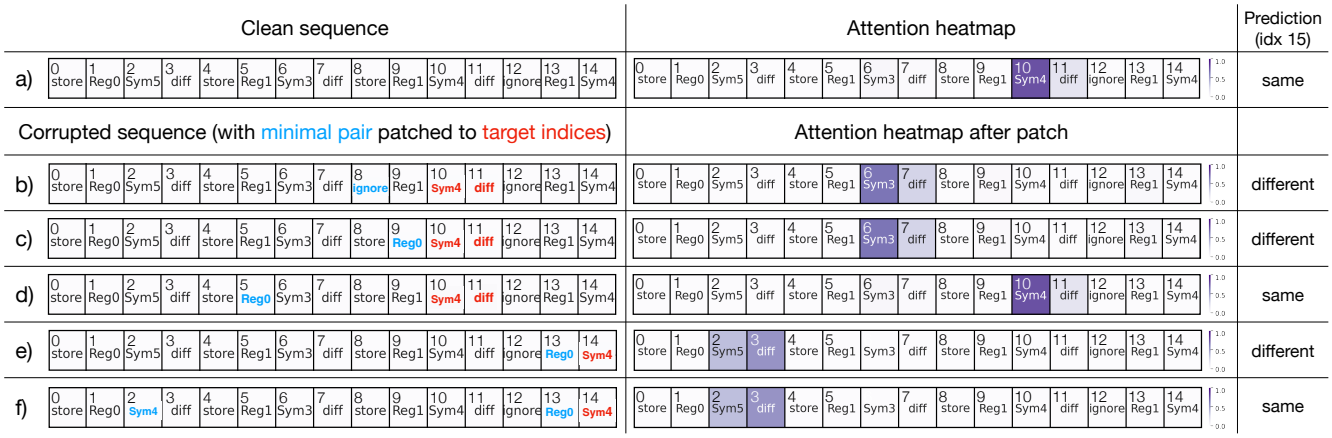

Figure 3: Model behavior when predicting same/different (token 15) is shown. We measure attention visualized as a shade of purple, with deeper shade corresponding to higher attention to that token. We create "corrupted" minimal pairs in which changing a token (light blue) either changes the correct label 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.

corresponding to the other tuple. In row e, the model finds Sym_5, and predicts `different`; and in row f, we patch in Sym_4 at index 2, and the model predicts `same`. Upon inspection, the query corresponds 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.

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 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.

We visualize the 5 runs that reach 100% accuracy on the test data in Figure 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.

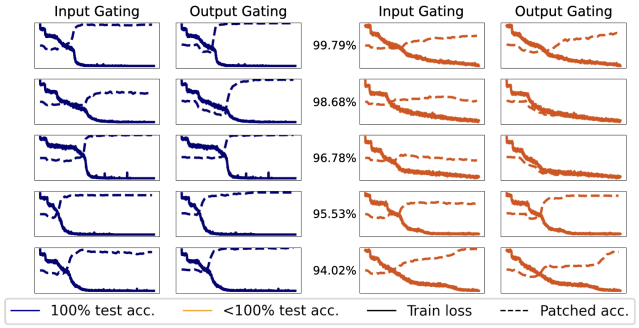

Figure 4: Model performance over training on patching subtasks. Each graph contains an individual model's training loss (solid line) and subtask accuracy (dashed line, between 0 and 1) over time; the line's color corresponds to whether the model reaches 100% accuracy on the general test set.

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.

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 are capable of making use of key composition for input gating and query composition for output gating on the task. 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 (Liu, Ash, Goel, Krishnamurthy, & Zhang, 2023), comprised of 1 register, 2 symbols (0 or 1), and long sequences. 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.

Acknowledgements

We would like to thank the members of the LUNAR lab and the Laboratory of Neural Computation and Cognition at Brown University for their thoughtful comments and discussion. This work was supported by ONR MURI Award N00014-23-1-2792.

References

Badre, D., & Frank, M. J. (2012). Mechanisms of hierarchical reinforcement learning in cortico–striatal circuits 2: Evidence from fmri. *Cerebral cortex*, *22*(3), 527–536.

Bahdanau, D., Cho, K., & Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Bhandari, A., & Badre, D. (2018). Learning and transfer of working memory gating policies. *Cognition*, *172*, 89–100. Retrieved from https://www.sciencedirect.com/science/article/pii/S0010027717303037 doi: https://doi.org/10.1016/j.cognition.2017.12.001

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... others (2020). Language models are few-shot learners. *Advances in neural information processing systems*, *33*, 1877–1901.

Burtsev, M. S., Kuratov, Y., Peganov, A., & Sapunov, G. V. (2020). Memory transformer. *arXiv preprint arXiv:2006.11527*.

Calderon, C. B., Verguts, T., & Frank, M. J. (2022). Thunderstruck: The acdc model of flexible sequences and rhythms in recurrent neural circuits. *PLoS Computational Biology*, *18*(2), e1009854.

Chatham, C. H., Frank, M. J., & Badre, D. (2014). Corticostriatal output gating during selection from working memory. *Neuron*, *81*(4), 930–942.

Collins, A. G., & Frank, M. J. (2013). Cognitive control over learning: creating, clustering, and generalizing task-set structure. *Psychological review*, *120*(1), 190.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., & Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*.

Du, H., Yu, X., & Zheng, L. (2021). Vtnet: Visual transformer network for object goal navigation. *arXiv preprint arXiv:2105.09447*.

Elhage, N., Hume, T., Olsson, C., Schiefer, N., Henighan, T., Kravec, S., ... Olah, C. (2022). Toy models of superposition. *Transformer Circuits Thread*. (https://transformer-circuits.pub/2022/toy_model/index.html)

Elhage, N., Nanda, N., Olsson, C., Henighan, T., Joseph, N., Mann, B., ... others (2021). A mathematical framework for transformer circuits. *Transformer Circuits Thread*, *1*.

Frank, M. J., & Badre, D. (2012). Mechanisms of hierarchical reinforcement learning in corticostriatal circuits 1: computational analysis. *Cerebral cortex*, *22*(3), 509–526.

Goldowsky-Dill, N., MacLeod, C., Sato, L., & Arora, A. (2023). Localizing model behavior with path patching. *arXiv preprint arXiv:2304.05969*.

Huang, W., Abbeel, P., Pathak, D., & Mordatch, I. (2022). Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning* (pp. 9118–9147).

Kriete, T., Noelle, D. C., Cohen, J. D., & O'Reilly, R. C. (2013). Indirection and symbol-like processing in the prefrontal cortex and basal ganglia. *Proceedings of the National Academy of Sciences*, *110*(41), 16390–16395.

Lewkowycz, A., Andreassen, A., Dohan, D., Dyer, E., Michalewski, H., Ramasesh, V., ... others (2022). Solving quantitative reasoning problems with language models. *Advances in Neural Information Processing Systems*, *35*, 3843–3857.

Liu, B., Ash, J. T., Goel, S., Krishnamurthy, A., & Zhang, C. (2023). Exposing attention glitches with flip-flop language modeling. *arXiv preprint arXiv:2306.00946*.

Nanda, N., & Bloom, J. (2022). *Transformerlens.* https://github.com/neelnanda-io/TransformerLens.

Olah, C. (2022). *Mechanistic interpretability, variables, and the importance of interpretable bases.* https://www.transformer-circuits.pub/2022/mech-interp-essay.

O'Reilly, R. C., & Frank, M. J. (2006). Making working memory work: a computational model of learning in the prefrontal cortex and basal ganglia. *Neural computation*, *18*(2), 283–328.

Pearl, J. (2001). Direct and indirect effects. In *Proceedings of the seventeenth conference on uncertainty in artificial intelligence* (p. 411–420). San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Power, A., Burda, Y., Edwards, H., Babuschkin, I., & Misra, V. (2022). Grokking: Generalization beyond overfitting on small algorithmic datasets. *arXiv preprint arXiv:2201.02177*.

Rac-Lubashevsky, R., & Frank, M. J. (2021). Analogous computations in working memory input, output and motor gating: Electrophysiological and computational modeling evidence. *PLoS computational biology*, *17*(6), e1008971.

Rac-Lubashevsky, R., & Kessler, Y. (2016). Dissociating working memory updating and automatic updating: The reference-back paradigm. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *42*(6), 951.

Räuker, T., Ho, A., Casper, S., & Hadfield-Menell, D. (2023). Toward transparent ai: A survey on interpreting the inner structures of deep neural networks. In *2023 ieee conference on secure and trustworthy machine learning (satml)* (pp. 464–483).

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., ... others (2023). Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones,

L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, *30*.

Wang, K., Variengien, A., Conmy, A., Shlegeris, B., & Steinhardt, J. (2022). Interpretability in the wild: a circuit for indirect object identification in gpt-2 small. *arXiv preprint arXiv:2211.00593*.

Wang, Y.-S., Lee, H.-Y., & Chen, Y.-N. (2019). Tree transformer: Integrating tree structures into self-attention. *arXiv preprint arXiv:1909.06639*.