DRAFT: Strange-Loop Networks (SLNs): A novel class of ”loopy” Neural Network Architectures that mimic the cerebral cortex to natively encode self-reflection, hemispheric specialization, and an infinite composition of functions

Francois Chaubard (fchaubar@stanford.edu)

July 29, 2024

Abstract

DRAFT: Since the advent of Generative Pretrained Transformer (Radford, et al. 2018), Machine Learning researchers have taken Richard Sutton’s Bitter Lesson to the extreme, delivering more and more capable yet, larger and larger ”single-lobed” language models (LLMs). Despite their impressive capabilities, these models are inherently and architecturally limited, as they have no native mechanism to ”ponder” as humans do. Instead, they are forced to predict their first token instantly after the last token of the prompt without an ”inner dialogue”. They are not natively able to self-reflect before emitting tokens, which results in hallucinations. Additionally, they are not able to ”adapt” the amount of compute required to solve a task to the difficulty of the problem, allocating more compute for more difficult tasks and less compute for less. Also, learned ”basis functions” in a single layer can not be called more than once per token emission, instead, such a function must be learned in each layer if required to do so to natively solve a ”recursive problem”, i.e. factorial, fibonacci, or search. Finally, modern LLMs can not natively self-improve via self-play, which limits the class of problems these models can solve, the skill-acquisition efficiency from which they learn, and the generalization ability to Out-of-Distribution (OOD) tasks.

Inspired by the ”loopiness” and ”competition” of the two hemispheres in the human cerebral cortex, our research introduces a new class of neural network architectures called Strange-Loop Networks (SLNs) defined as any architecture that has (at least) 2 distinct models (left model θ_{left} and right model θ_{right}) such that the input to the θ_{left} is the output of the θ_{right}, and the input to the θ_{right} is the output of the θ_{left}. They communicate in an Internal Dialogue Loop (IDL) until some stopping criteria is achieved, then emit an output externally. We show SLNs achieve significant improvement in generalization ability, holding model size, number of tokens, and context size constant, which Chollet, 2019 formally defines as ”the intelligence of a system IS”.

We propose many instantiations of these SLNs that are possible. For example:

1) In a ”tall” SLN configuration, we deploy two pretrained LLMs that are connected in a ”strange loop” where one outputs into the other and visa versa to simulate the dual hemispheres of the human brain and allow them to jointly learn to collaborate. These models engage in an ”Inner Dialogue Loop,” until they agree they have the answer, allowing them to collaborate with each other until both lobes are satisfied.

2) In a ”wide” SLN configuration, a main gating neural network directs the computation to the left, right or output lobe, where the right and left lobes are instead very shallow, using only one gating neural network and one MOE (Mixture of Experts) layer allowing the model to compose functions infinitely, where the gating network decides which expert to invoke, and then invokes the expert repeatedly until stopping criteria. They continue discussing in an ”Inner Dialogue Loop,” until the main gating neural network is satisfied with the answer and then invokes the output neural network. This architecture permits the model to synthesize the output of a number of selected functions at each iteration and we prove this results in strong fluidity and abstraction.

3) In a ”lateralized” SLN, we restrict one model’s output to specialize in outputting only a score for a trajectory and the other is permitted to output the full trajectory. This concept can be applied to tall or wide SLNs just the same. In this way, one can view θ_{left} to be the ”Agent Model”, ”Actor”, ”Generator” or ”Student”, and θ_{right} to be the ”Reward Model”, ”Critic”, ”Discriminator” or ”Teacher”, depending on the training technique applied; i.e. RLHF, A2C, GAN, DPO, model distillation, etc.

In all cases, we train these SLNs jointly allowing the weights to specialize to their role in the partnership. Both θ_{left} and θ_{right} are jointly learned to collaborate together to solve higher order problems. Thus, these "lobes" learn a new language to efficiently communicate with each other and to stop when the two models are satisfied with their response. Our "strange-loop" framework achieves remarkable improvements on GSM8k with just a 820m model of XX% which is comparable performance to a 8B model with 10x fewer weights and compute, proving SLNs are much more information-efficient achieving much higher generalization per parameter than traditional LLMs. The SLN architecture opens a new field of research and provides a crucial step towards creating more generalizable, and perhaps even self-aware, AI systems. We release all code, demos, and datasets at https://github.com/fchaubard/

1 Introduction

1.1 Motivation for this work:

- **Hallucinations**: Brains have 2 distinct sets of weights that are inhibited by a giant 'fissure' that is punctured only via the Corpus Collosum (a large sting-ray looking structure of fibers that connects the two hemispheres of the brain). LLMs do not have such a large fissure separating two distinct models that can run independently. External information enters the brain usually only by one hemisphere, and is first processed by that hemisphere, and then the result is "shared" with the other hemisphere. LLMs do not do this which we believe to cause hallucinations. Brains have an "inner dialogue" before beginning to respond. LLMs do not. They *must* respond directly after the last layer runs. Human Brains, on the other hand, build up an "intuition" before responding, almost doing a "dry run", and getting feedback, before emitting final output tokens. This is a regularizer on the generator that LLMs do not possess natively, and only recent work has allowed the model to review its one work post generation, i.e. Self-Refine, LDB, Code-Gen, Reflexion, etc. We seek to model build this self-reflection into the architecture itself. In particular, with "Lateralized SLNs" (as we will define more fully below), the right network can be viewed as either a judge to validate the left hemisphere's output, before letting the left hemisphere respond externally. Human's build up an internal confidence in our answer before answering, allowing us to second-guess ourselves a bit before emitting a response. LLMs do not have any self-reflection built in, so they hallucinate because there is no "other model" to sanity check the response before emission. SLNs permit this. You need a mirror to see yourself and LLMs have no mirror! SLNs act as this mirror.

- **Lateralization and Ensembling**: Each hemisphere in the human brain "specializes" in specific functions (left for language / speech and right for facial recognition and spatial processing tasks). This "lateralization" allows for more complex processing that would not be otherwise possible. Additionally, each hemisphere could be viewed as two different "models" with different biases, so the merge of the two can be viewed as an ensemble, which is known to result in higher generalization at the cost of higher compute at inference time. LLMs do not have 2 lobes so such ensembling and specialization does not occur natively, and only recent works of using many different LLMs and learning which LLM to use have been proven to improve performance.

- **Adaptive Compute Time**: Brains have the ability to ponder. They can allocate more compute for more complex tasks, and less for less. LLMs do not. They mandate a fixed set of FLOPs before the model is required to emit the first token. We believe this inherently "caps" their ability to perform some tasks as they "run out of layers" before they can get the right answer. In modern LLMs, the same amount of compute would be used to solve 2+2 vs. 2534*5423. In SLNs, this would not be the case.

- **Forcing Higher Levels of Abstraction**: Mordern LLMs do not "reuse" layers recursively if it makes sense to. Instead, if they need to call a function twice to perform a specific task (sorting, adding numbers, counting, etc), it must be learnt in two or more distinct layers. To solve this, we just make the model bigger and bigger and deeper and deeper so it can do so. However, this comes at the cost of generalization and finding higher levels of abstraction. LLMs can not call a learned "function" in layer 3 x times before giving an answers. SLNs can do this natively, which enables same or better performance with far less weights and training required to perform all

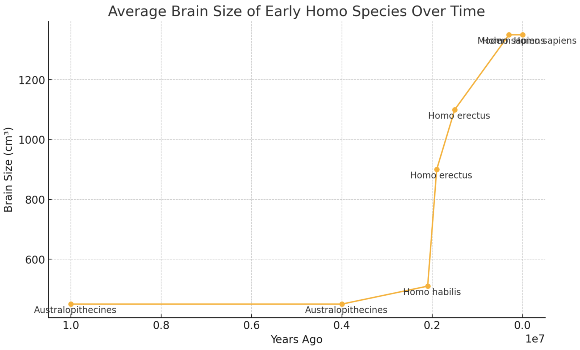

Figure 1: Brain Size Growth in early Homo species

tasks that require a recursive composition of a sub-function of the form $f(...(f(x)))$, and also higher levels of abstractions. Lets look at some examples:

  - i.e. Lets consider *MINIMAX* that is an algorithm commonly used in AI agents to play chess. In it, we roll out the game one step at a time trying to find the move that maxmizes our chance of winning and then minimizes the opponents chances of winning. LLMs can not natively rollout the game inside the model more than $d = depth$ times, and even if it can do it d times, then we would still have to embed this ability in each distinct layer. To enable this infinitely, we would have to add a procedure on top of the LLM artificially. However, in SLNs, we can let the network ”learn” to rollout a time sequence until some stopping criteria is met. It can “learn” MCTS effectively!

- **Self-Play:** Brains ”continually” update their weights just by *thinking*. Essentially, at test time they update their weights. All modern DL models do not do this. In ”Laterlized SLNs” the left lobe may be forced to update its weights as part of the inference procedure to maximize the R provided by the right lobe. Then the right lobe is updated by the externally provided R_{ext} at the end or by just not mapping the world correctly. This permits both models to learn from the experience in much different ways. Also, it permits the model to disvover new knowledge as shown in AlphaGoZero (Silver, 2017), where the policy and value models discovered novel strategies to play the game of Go that was well outside of the (non-existent) training distribution.

1.2 Evolution Perspective:

The evolution of our brains began with small and simple vertebrae' Central Nervous System structures, but went through many distinct periods of exceptional expansion of the neocortex. Around 10 million ago, early Homo species such as Australopithecines had relatively constant brain sizes of about 450 cm$^{\mathbf{3}}$ for over 6 million years. Then, with the emergence of Homo habilis (2.1 to 1.5 million years ago), brain size began gradually increasing to approximately 510 cm$^{\mathbf{3}}$. However, the largest jump in brain size called ”The Great Cortical Expansion” started with Homo erectus (1.9 million to 110,000 years ago), whose brain sizes doubled in size over a remarkably short time period to 900-1100 cm$^{\mathbf{3}}$. This continued and overlapped with Homo Sapiens (300,000 years ago), whose brain sizes grew to an average 1,350 cm$^{\mathbf{3}}$. However, then brain size stopped increasing ever since. That is still the average brain size for humans today.

This presents a puzzle. If bigger brain sizes (bigger LLMs) result in higher levels of generalization ability and thus as Darwin would say ”the most adaptable to change”, then why have our brains stop growing these last 300,000 years? One can infer that brain size stagnation signals that increasing brain size was no longer the highest gradient direction to improve evolutionarily. Larger brains require more energy and resources, and beyond a certain point, further increases in brain size might have diminishing returns, as evidenced by elephants, which have brains three times the size of humans but are not three times more mentally capable. Instead, natural selection discovered a better path toward higher competency is to reorganize itself through ”hemispheric specialization”, or ”lateralization”. A key development that enabled this was growth of the Corpus Callosum, the information superhigherway connecting the two hemispheres. During this time, the anterior and posterior regions of the Corpus Collosum grew in thickness to support higher bandwidth for interhemispheric communication which is crucial for complex cognitive processes and for elimination of redundant processing in each lobe. This improvement in the Corpus Collosum is believed by many neuroscientists to be the enabler of the pivotal leap towards the human condition, which parallels the profound impact of opposable thumb on humans ability to utilize tools. ”It may turn out that the oft-ignored corpus collosum, a fibre tract that is thought merely to exchange information between the two hemispheres, was the great enabler for establishing the human condition. Non-human brains reveal scant evidence for lateral specialization.” - Gazzinga, et al

Because of this growth, redundant functions in one hemisphere could be jettisoned, freeing up capacity to provide new functionality to the brain at large. Evolutionary neuroscientist have discovered that at the same time as the improvement in Corpus Collosum, 300,000 years ago, Homo Sapiens developed specialized language production and comprehension processing areas in the left hemisphere, Broca’s area and Wernicke’s area respectively, which was critical for the development of speech, language, hypothesis formation, critical thinking, and comprehension. Soon after (70k years ago), historians date the first use of tools, art, intentionally burying the dead (thus, religion), etc.

Furthermore, both hemispheres process much of the same information in very different but complimentary ways giving notions of ensemble learning. They jointly steer a ”central attention” toward other areas of the brain for it to determine what compute to do next. For example, both have a motor cortex that can trigger motion in the other half of the body. People with an intact Corpous Collosum can not attend to two things at once, while people with a severed Corpous Collosum can do so easily.

Finally, as the famous quote on consciousness goes: *”It is very hard to see yourself without a mirror”*. Having two lobes that can each ***see*** each other is commonly thought to be a prerequisite (and may be the missing link to enabling AI models) to become self-aware. To quote GEB: *”When and only when such a loop arises in a brain or in any other substrate, is a person — a unique new ‘I’ — brought into being. Moreover, the more self-referentially rich such a loop is, the more conscious is the self to which it gives rise”. - Hofstadter, 1979 .* Of course, we must disclaim that each model must be capable and big ***to*** see.

1.3 AI Perspective:

Recently, many AI researchers have converted LLMs into ”Agents”, where multiple instances of the same model (the same weights) are instantiated with distinct system prompts, and run iteratively to allow the agents to communicate with each other and themselves, reflecting on their previous outputs and each others outputs, and using that to improve their answers. This approach has yielded strong gains in many benchmarks. For example, Reflexion and Self-Refine showed a 10-15% on tasks such as HumanEval and MBPP. This is a ”Strange-Loop Network” as per the definition.

However, these LLMs were never ”trained” to self-reflect or work together and collaborate which as we show, allows for significantly more complexity and generalization ability.

Finally, one can interpret what is happening at each Transformer block in LLMs to be a bank of functions F, where the attention mechanism finds the appropriate $f \in F$ and then sends it to the Feed-Forward layer to apply the f to the input. But since there is no loop, and the architecture is completely feed-forward, so it can only do this a d number of times (where d = the depth of the LLM), so for easy tasks (like 2+2), this bank of functions is called d times even though one or two calls to F would be sufficient, and for difficult problems (like hard coding problems) F is only invoked d times where perhaps it should be invoked many more times since the challenge is so great. Similar to ”Adaptive Computation Time (ACT) [insert]”, our class of networks has the native ability to learn the number of computational steps required to solve the task which introduces meaningful inference compute efficiencies.

Similar to the evolution of the human brain, LLMs started out small, with limited capabilities. To improve performance, AI researchers followed Richard Sutton’s ”Bitter Lesson” and grew model sizes and training tokens exponentially which worked well. To boost performance further, the latest trend is to wrap the LLM in a while true loop to allow it to self-reflect on its own output and/or to allow it to communicate with other LLM ”Agents”, which then would form a loop and thus, would be in the class of a Strange-Loop Network. We propose the next trend is to allow the agents to ”train together”, to reorganize and recognize themselves natively.

While I agree with Yann LeCun that we did not need to make a machine that has flapping wings to enable flight, so perhaps we do not "need" two specialized hemispheres to achieve AGI, but it does seem like something nature has chosen in all vertebrates, and certainly Homo Sapiens, which are the most intelligent beings on planet earth, so something to try.

Additionally, using self-play in a multi-agent setting to improve performance has been proven to work in as early research as Paredis, 1995. Early work explored self-play using genetic algorithms (Paredis, 1995; Pollack et al., 1997; Rosin & Belew, 1995; Stanley & Miikkulainen, 2004). Sims (1994a) and Sims (1994b) studied the emergent complexity in morphology and behavior of creatures that "learned to collaborate" in a simulated 3D world. Open-ended evolution was further explored in the environments Polyworld (Yaeger, 1994) and Geb (Channon et al., 1998), where agents compete and mate in a 2D world, and in Tierra (Ray, 1992) and Avida (Ofria & Wilke, 2004), where computer programs compete for computational resources.

Most recently, in Baker, et al, 2020 "HIDE AND SEEK: EMERGENT TOOL USE FROM MULTI-AGENT AUTOCURRICULA" (insert paper) agents were trained against each other in thousands of games of hide-and-seek, leveraging self-play, multi-agent interaction, PPO and GAE. With no human instruction, just the reward signal from the game, as many as six completely novel and distinct strategies emerge. Each new strategy that was invented by the agents creates a previously non-existent competitive pressure for agents to progress to the next stage, inspiring the next generation of agents to be smarter and improve. Note, that there are no direct incentives for agents to interact with objects or to explore; rather, the emergent strategies shown below are a result of autocurriculum induced by multi-agent competition and the simple dynamics of hide-and-seek.

Since during optimization the "Entity Embeddings" used to train the agents included all the states of all agents, the two policies learned together thus may actually be caste as training two lobes of a single brain that can collaborate! "Agent policies are composed of two separate networks with different parameters – a policy network which produces an action distribution and a critic network which predicts the discounted future returns." and "At optimization time, we use a centralized omniscient value function for each agent, which has access to the full environment state without any information masked due to visibility, similar to Pinto et al. (2017); Lowe et al. (2017); Foerster et al. (2018)"

In our strange-loop architectures, one could easily show that this "infinite function composition" and rollout could permit a native "monte carlo tree search" to predict into the feature and perform simulation within the model architecture itself. This functionality is not possible inside a traditional LLM without a loop as it has a static number of layers and static compute.

Additionally there are many examples of using 2 different "agents" in machine learning with differing objectives to enable successful training. Our "lateralized SLNs" encode this overtly to mimic this strong trend in RL.

First, in a typical LLM training, we first must fine tune with SFT, then we train a Reward model (right lobe) on a preference dataset, then we use that reward model in PPO to train the Generative model (left lobe) that is able to take instruction and generative trajectories that are in line with the Reward model. However, the Reward model is not typically used at inference time and is instead thrown away. Also, these two models do not "learn together". Finally, it is worth noting, for this process to work the Reward model must be similar in size to the Generative model. We believe this to be supportive of why we have two equally sized yet largely independent hemispheres and provides the motivation for our Lateralized SLN.

Second, GANs (Generative Adversarial Networks). In GANs, we train two different networks to "compete" with each other to improve in each others performance, a Generator (left lobe) and a Discriminator (right lobe). The Generator's job is to generate images that are realistic and the Discriminator's job is to decipher what is a real image and what is a fake, generated image.

Third, Actor-Critic Algorithms in RL train an Actor (left lobe) to provide a policy given a state, and the Critic (right lobe) is trained to predict the value of a given state. They are often trained separately but used jointly to collaborate together to find the best policy given an action. Such 'specialization' results in huge improvements in performance over a single model that must do both.

Fourth, AlphaGo, SPIN, SRPO, and DPO all leverage two "different" models which are usually earlier checkpoints of itself to enable self-play to improve generalization ability and to enable the creation of new knowledge that is far outside of the training distribution. This can be viewed as two "competing" hemispheres with just a lag in "experience", like a student (left lobe) and a teacher (right lobe) playing each other to help the student improve at a faster rate than would otherwise until of course when student becomes master, and then the cycle continues again and again.

All of these examples can be viewed as supporting evidence that forming a large inhibiting fissure between two large models is better than just increasing the model size by double. We need two separate models that are trained together for us to be able to optimally generalize, self-reflect, and self-play.

1.4 Philosophical Perspective:

- TODO
- Discuss ”The Origin of Consciousness in the Breakdown of the Bicameral Mind”.
  - Julian Jaynes proves that ”The Double Brain” and lateralization were necessary for Consciousness to emerge where early Homo Sapiens (called the Bicameral man) specialized the left-brain to become our ’self image and inner dialogue’ and the right brain became our ’external God’ that would command us. And without changes to the genome, and just specific ”training”, we were able to harness these volitions to permit Consciousness.
- Discuss Gödel, Escher, Bach: An Eternal Golden Braid by Douglas Hofstadter explores the concept of strange loops across mathematics, art, and music.
- Discuss ”I Am a Strange Loop” by Douglas Hofstadter delves deeper into how strange loops relate to consciousness and self-awareness.
  - “In the end, we self-perceiving, self-inventing, locked-in mirages are little miracles of self-reference.”
  - ”In short, there are surprising new structures that looping gives rise to that constitute a new level of reality that could in principle be deduced from the basic loop and its detailed properties, but that in practice have a different kind of “life of their own” and that demand — at least when it comes to extremely finite, simplicity-seeking, pattern-loving creatures like us — a new vocabulary and a new level of description that transcend the basic level out of which they emerge.”

1.5 Primary Contributions:

We summarize our primary contributions here:

- We introduce a new class of neural networks called ”Strange-Loop Networks” that leverage ”strange-loops” natively encoded in the architecture, that address many of the limitations of traditional LLMs. Namely, they do not have the ability to self-reflect or ponder, they do not ”adapt” the amount of compute to the difficulty of the problem resulting in far too much compute on some problems and not enough on others, they do not have the ability to iteratively call the same function over and over again and must instead re-learn that function at each layer, [...etc...]
- We map advancements in LLM performance and advancements in early Homo species. Both similarly underwent a ”Great Cortical Expansion” then that capped out, and then we needed ”specialization” of the hemispheres, or ”lateralization”.
- We develop novel training procedures to train these SLNs and prove their convergence.
- We ablate and compare the capability of a single LLM (A), a homogenous LLM-based Single Agent with self-reflection (B), a homogenous LLM-based Multi Agent System with self-reflection (C), and finally our Strange-Loop Networks, a heterogeneous, jointly trained LLM-based Multi Agent System with self-reflection to form a ”strange-loop” in both tall (D), wide (E), and lateralized tall (F), and lateralized wide (G).
- We ablate different ”left lobe” and ”right lobe” strategies and architectures and methods of training and many different tasks.
- We prove that Strange-Loop Networks framework achieve meaningful improvements over all baselines, and achieves state-of-the-art results on many benchmarks. [NEED TO GO PROVE!]

2 Related Work

2.1 Paper 1

(TODO)

3 Method

3.1 Strange-Loop Networks

3.2 Lateralized Tall Strange Loop Network

To take a step back and try to reverse engineer how our brains may work as a way to draw inspiration, I believe the right hemisphere (critic/discriminator) looks at things "holistically", and outputs a "feeling of rightness or wrongness", while the left brain analyzes things, breaks them down into chunks, and generates new thoughts/actions from that analysis. The right brain develops a "taste" for things. Music, art, clothes, food, etc.. It maps inputs to an instant response of "yea that looks good" or "hmm I don't like that".

1. "Where" does negative emotion come from?
   (a) The right hemisphere, especially the right prefrontal cortex, is more involved in processing negative emotions. It is associated with feelings of sadness, fear, and withdrawal-related behaviors.
2. "Where" is your "taste" for music, for food, for art?
   (a) When you intuitively like or dislike art, music, or food, the right hemisphere is generally more active due to its role in processing emotions, holistic patterns, and aesthetic appreciation.
3. What makes us laugh? What happens in the brain? Why do we find somethings funny or others not?
   (a) The right hemisphere is more involved in processing the emotional and social aspects of humor. It plays a significant role in understanding the context, emotional tone, and non-verbal elements of humor, such as facial expressions and body language. The right hemisphere is also important for understanding incongruity and the "aha" moment when a joke's punchline is delivered.

In short, the right hemisphere (in one shot) gives us a non-word-based "feeling" for good or bad / right or wrong. Then the left hemisphere (actor/generator) tries to rationalize the "feeling", invent an explanation, why its not exactly right, and generates how it can be improved. Its specialized for logic, reasoning, analysis, breaking things apart. While the right lobe is specialized in synthesis, looking at things on the whole.

To use a real example. When you are reading this section, you may think to yourself... Does this section "make sense" holistically (right lobe)? Does it "match my world view" or not? You may have a "feeling" or "intuition" that is word-less to the extent to which you agree or disagree with what you are reading (right lobe). Then, if prompted, I may ask you to explain the extent to which you agree or disagree with this section, and you must then "generate" (left lobe) an explanation why you agree or disagree, that did not exist before I prompted you. Furthermore, if I asked you to "remember" this section because I am going to quiz you about it later, you would then need to stop reading.. close your eyes.. try to explain it a bit by generating some "trajectories" (left lobe, which is self-play and learn, to "imprint" that experience into the weights), and then you will again have a "feeling" whether or not you did a good job, if your trajectories match your memory of what you just read, and if it "makes sense" / matches your world model (right lobe).

To break this proposed learning process up into steps that we can then mimic in a machine learning model and training setup:

1. Some external input stimulus hits the brain **(encode / prompt step)**

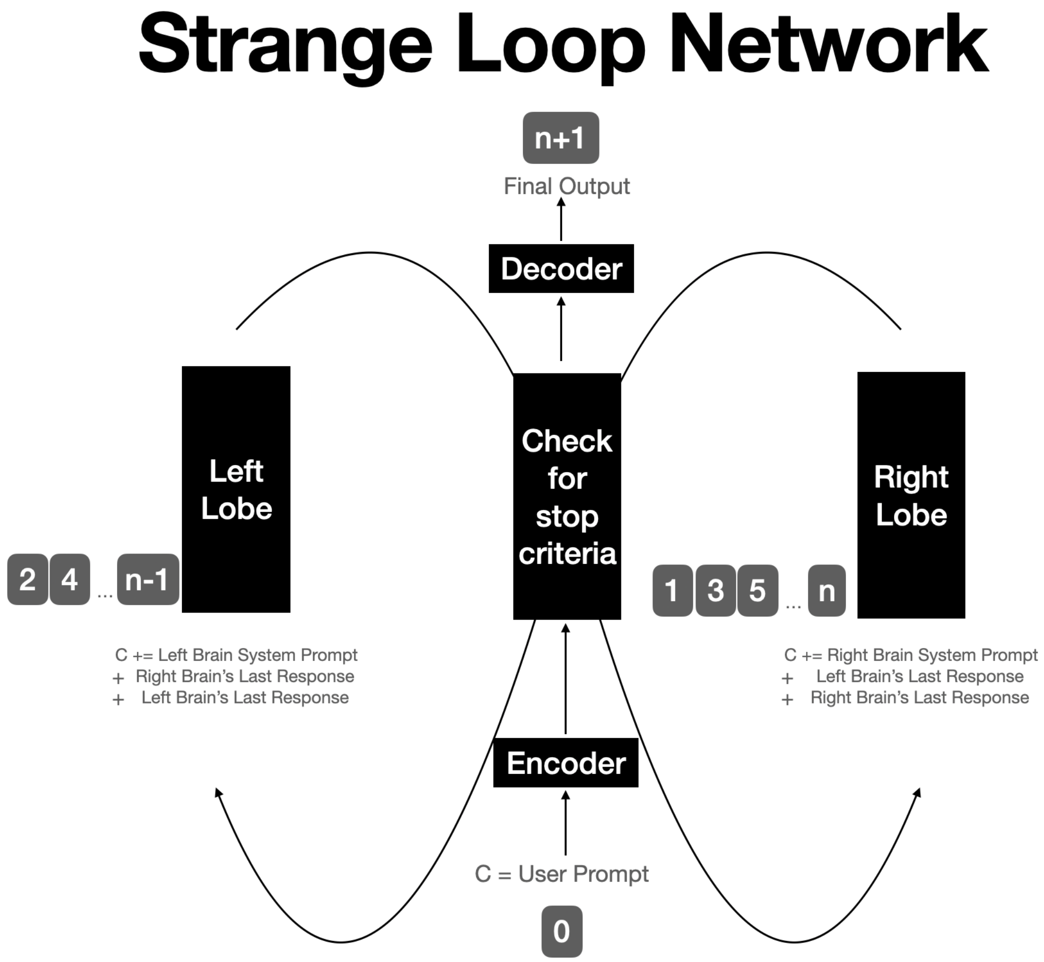
Strange Loop Network
n+1
Final Output
Decoder
Left Lobe
Check for stop criteria
Right Lobe
2
4
...
n-1
1
3
5
...
n
C += Left Brain System Prompt
+ Right Brain's Last Response
+ Left Brain's Last Response
C += Right Brain System Prompt
+ Left Brain's Last Response
+ Right Brain's Last Response
Encoder
C = User Prompt
0

Figure 2: Template Strange-Loop Network (SLN) defined as any architecture that has (at least) 2 distinct models (left model ”LM” and right model ”RM”) s.t. the input to the LM is the output of the RM and the input to the RM is the output of the LM until some stopping criteria is achieved. In some cases, it would make sense to an encoder network before inputting external input directly. In some cases, it would make sense to an decoder network after the Strange-Loop before emission of an output. Finally, there are many options for stopping criteria to break out of the Strange-Loop such as of Inner Dialogue Loops, a confidence score, a match between the output of the left and right lobe (agreement), etc.

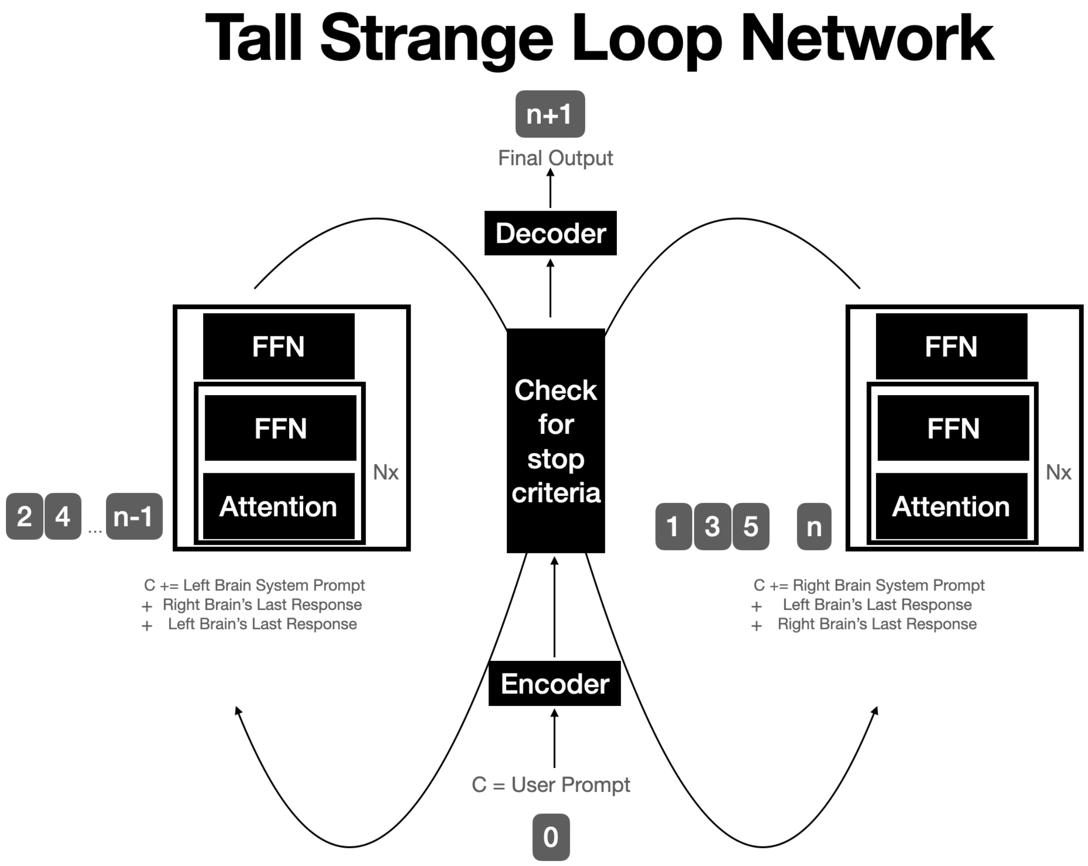
Tall Strange Loop Network
n+1
Final Output
Decoder
FFN
FFN
Attention
Nx
2
4
...
n-1
Check for stop criteria
1
3
5
n
FFN
FFN
Attention
Nx
C += Left Brain System Prompt
+ Right Brain's Last Response
+ Left Brain's Last Response
C += Right Brain System Prompt
+ Left Brain's Last Response
+ Right Brain's Last Response
Encoder
C = User Prompt
0

Figure 3: Example of a Tall Strange-Loop Networks. We take a pretrained LLM (like Llama 3 7B for example), copy it to a left lobe and a right lobe, we optionally add a system prompt to each to ”steer” it to the hemispheric specialization we want, and then let it perform an ”Inner Diologue Loop” (IDL) until a stop token is emitted, then the agent outputs its final answer.

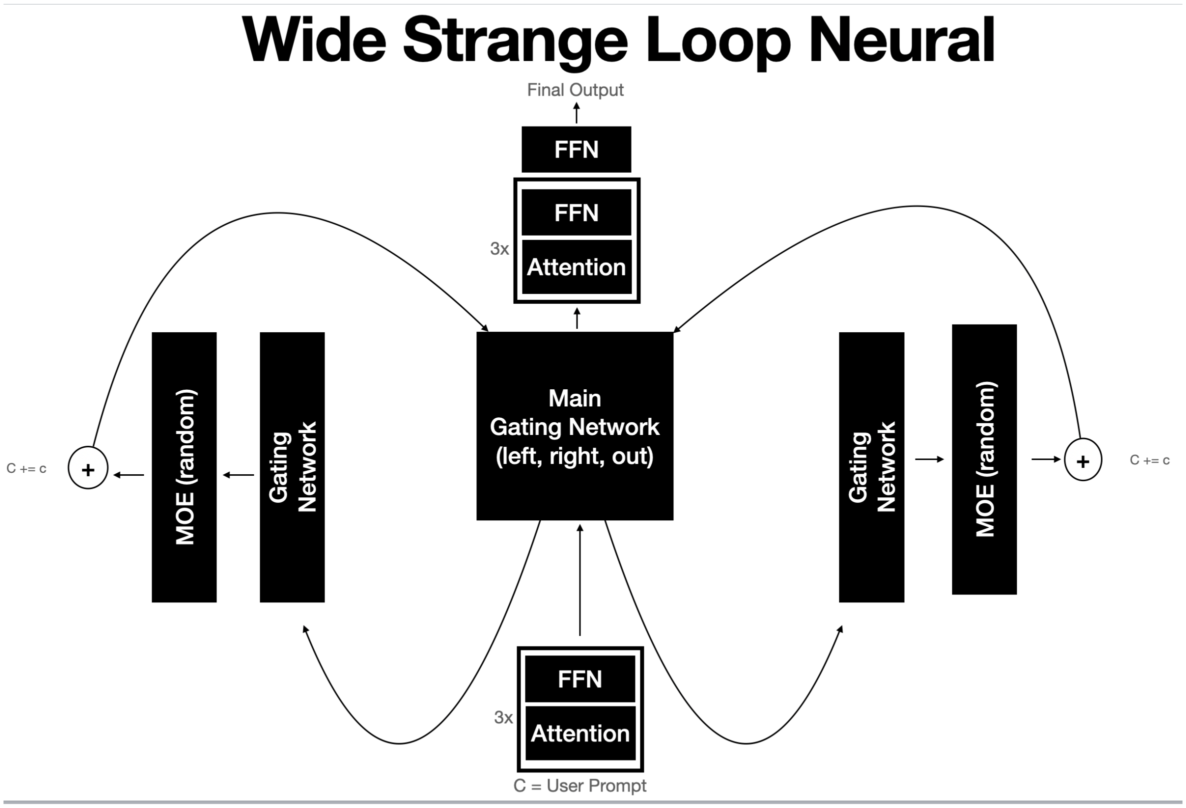

Figure 4: Example of a Wide Strange-Loop Networks. We take a pretrained LLM (like Llama 3) and perform a lobotomy. We input the tokenized input into the original LLM for only the first 3 Transformer Blocks, then introduce a Main Gating Network that will decide where to allocate the compute (to the left lobe, the right lobe or the output lobe), and allow the model to conduct an ”Inner Dialogue Loop” (IDL) until the Main Gating Network decides its time to output. Each lobe consists of only two layers, a gating network to what compute to conduct, and then the next layer (the expert) performs that compute. We then concatenate the results to the context matrix, and then send it back to the Main Gating Network to go again and again until the Main Gating Network is satisfied. At that point, the agent outputs its final answer through the last 3 Transformer Blocks and the final output FFN as per the original LLM than train this model over many tasks.

(a) If auditory prompt.. auditory cortex in left hemisphere..

(b) If reading prompt.. dorsal or ventral pathway to both lobes

(c) If “null prompt”.. a thought appears in ones head.. from wernickes area

2. For logic / reasoning / language tasks, the left hemisphere will slosh back and forth... will have a lot more activity than the right.. will create a “trajectory” (recursive Feed-Forward call) and push it to the right hemisphere. Neuroscientists have measured significantly more communication from left to right during logic / critical thinking tasks than communication from the right to the left. **(initial actor trajectory)**

3. Then will output through the Corpus Callosum to the right lobe (with an “expected” reward perhaps.. ?). The right lobe will process the output and “sanity check” the output and respond back with a “score” or a “feeling” (intuition?) if the trajectory is good or not perhaps with a dopamine release? **(initial critic response)**

4. If the response is not good, we then go back to 2 and try it again.. **(actor’s improved trajectory)** and back and forth, until the right lobe is happy.. **(critic approval and stopping criteria)**.

5. Then we have “the solution” in our head with agreement from both lobes.. but we are yet to articulate it. So we need to “decode” the thought from the prefrontal cortex to wernickes area to express the thought in words. **(decoder step)**

6. Sometimes an external reward may occur (we get the answer to a question right / wrong or the ball goes in the basket or not, etc). We can take the external r_t received from the environment, and then we must check if that matches with our expected reward (right lobe), and if its different than expected, then we are surprised, and then we must update / optimize our world view of a state S, $V(S = [S_1, S_2, \ldots, S_n])$ (with TD / Q / etc).. to update the right lobe first.. which may explain why we will *instantly* ”feel bad” if we didnt get an answer right right or we will instantly ”feel good” if we do get it right, well before we logic about why we got the answer right or wrong. **(right lobe learning step)**

7. An important side note, thinking of Pavlov’s classic conditioning, when we do have an external reward provided, we need to ”bleed” this view into previous states as well. Not just for state S, but the early contiguous subparts as well of $S = [S_1, S_2, \ldots, S_n]$. This is akin to eligibility traces and Exponential Decay Reward Shaping as is common in Q-Learning(λ).

8. After the right lobe has learned something, then how does this update to our world model pass to the left lobe? The left lobe could update in a few ways.. perhaps it updates / backprops online, in the generation step 3 and 5 to ‘backprop’ until the right lobe seems happy.. it takes repeated iteration to learn a song on the guitar ”muscle memory” (left lobe) even if we feel (right lobe) that we are not getting it right. But we keep iterating until we get there. So then the left lobe must start ”practicing” (generating trajectories and learning in the process) to get a good response out of the right lobe.. and when we do not, we backprop in the direction of the reward from the right lobe... maximizing our response to make the right lobe happy). **(left lobe learning step)**

4 Implementation Details

4.1 DataSets

- Train:
  - Game Play:
    - Gridworld
    - Tic Tac Toe
    - Checkers

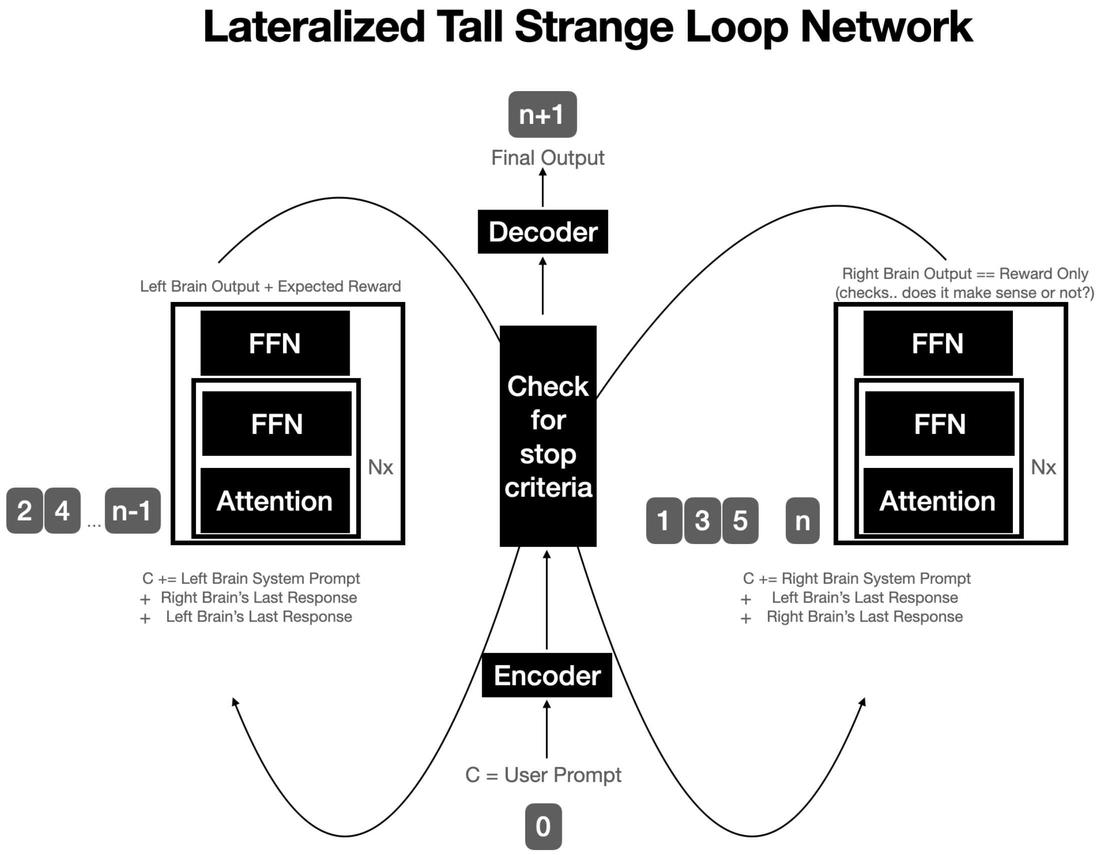
Lateralized Tall Strange Loop Network
n+1
Final Output
Decoder
Left Brain Output + Expected Reward
FFN
FFN
Attention
Nx
2
4
...
n-1
Check for stop criteria
Right Brain Output == Reward Only
(checks.. does it make sense or not?)
FFN
FFN
Attention
Nx
1
3
5
n
C += Left Brain System Prompt
+ Right Brain's Last Response
+ Left Brain's Last Response
C += Right Brain System Prompt
+ Left Brain's Last Response
+ Right Brain's Last Response
Encoder
C = User Prompt
0

Figure 5: Example of a Lateralized Strange-Loop Network. Similar as others but we overtly require the left brain to output its trajectory and its expected reward and the right brain to specialize to be a reward model. Inspired by Julian Jaynes ”Origin of Consciousness” where he states ”the advent of consciousness necessitated the inhibition of the auditory hallucinations of the right temporal cortex” meaning, only one lobe can speak, otherwise the ”self” is not formed, and we become schizophrenic.

        - Chess
    - Code:
        - APPS
        - DS-1000
        - LeetCode-Hard
        - Alpaca
        - CodeXGlue
            - Refinement
            - Text2Python
            - Text2Java
            - Text2Javascript
            - Text2Go
            - Text2SQL
        - Logic:
            - Math
            - MMLU dev
            - Winogrande 10k

- Test:
    - Game Play:
        - Gridworld?
        - Tic Tac Toe optimal Computer?
        - Stockfish for checkers?
        - Stockfish for chess?
    - Code:
        - MBPP-ET Test set
        - HumanEval-ET Test set
    - Logic:
        - MMLU Test
        - GSM8k

5 Candidate Training Procedures:

There are many ways to train SLNs as we will discuss below. Training SLNs can be quite expensive because in the forward pass, in each Inner Dialogue Loop (IDL) step, you must allow the first model to generate an inner dialogue sequence of length $m1$, so must forward pass $m1$ times, and then the second model must take that as input and generate an inner dialogue sequence of length $m2$, so must forward pass $m2$ times, and so on and so forth, and lets say we permit this exchange for d IDLs, making the number of forward passes to generate a single output trajectory to be $O(m1 * m2 * d)$ which is quite expensive compared to a single-lobed LLM forward pass, but more than offset by the reduction in number of layers and model size to achieve comparable generalization. Lets review a few of the options at our disposal.

We focus most of our efforts trying to train the ”Lateralized SLN” which permits the left model (θ_{left}) to output sequences of tokens, and the right model (θ_{right}) to only output a ”score” given a sequence of tokens, higher score if the sequence is of good quality, and lower if it is not. We try our best not to impose an explicit reward function and instead let the model determine its own reward.

In all cases, we have two distinct neural networks formed in a strange loop, where the output of one is the input to the other and visa versa. The left model will take in an input (x) from the external world and produce a full response ($\hat{x}_{left}$) (to complete the prompt), and the right lobe will take in a sequence of tokens, either $(x + \hat{x}_{left}\ )$ or $(x + y_{true})$, where y_{true} is the "true desired response" from natural text or from a human annotator or otherwise, and then the right model will score the input, to evaluate it, to see if it's any good and output a score response $\hat{y}_{right}$ which is just a probability scalar.

If the right model agrees with the response, it will output $\hat{y}_{right} = 1$, if the right model believes the left model can do better, then the right model will output $\hat{y}_{right} = 0$. In this case, then the left model will take in an updated input $(x+\hat{x}_{left}+\hat{y}_{right})$, and produce another response $\hat{x}_{left}$ which will be an attempt to try to improve on the first attempt.

It will continue doing this "inner dialogue loop" (IDL) until the right model achieves some threshold or if we hit some IDL_limit (or some other stopping criteria).

To convert from a "Lateralized SLN" to a normal SLN, we simply allow the right model to output a full response. So (after the first loop) the input the right model would be $(x + \hat{x}^{t}_{left} + \hat{x}^{t-1}_{right})$ and the input to the left model would be $(x + \hat{x}^{t}_{right} + \hat{x}^{t-1}_{left})$. However, in this setup its difficult to allow them to specialize and also difficult to understand when is a good time to stop. Additionally, it is much more computationally expensive this way since the right model will have to do equal amounts of work as the left model. However, Real-Time fMRI scans show that while performing logic tasks, the left hemisphere is far more active than the right, and the communication across the Corpous Collosum provides more information from the left to the right than from the right to the left suggesting that the Lateralized SLN is the more biologically supported architecture.

Now we must find a way to train this network. It is our goal to train the two models to learn how to collaborate with each other to improve and generalize from the provided (x, y)'s ideally without the requirement of expensive and hard to attain human preference data.

So how to do it? We examine a number of candidate approaches.

5.1 Classic SFT and RLHF Setup

The obvious thing to try is what currently works for LLMs. One can view the typical LLM training procedure to be a Strange-Loop Network where the models are trained separately and then the right lobe is ignored at inference time. In a typical LLM training procedure, we train the left model (θ_{left}) and the right model (θ_{right}) separately. First, we would train both to predict the next token via SFT on the target dataset P_{data} like so:

$$L_{\mathrm{SFT}}(\theta) = -\mathbb{E}_{\mathbf{x}\sim q(\cdot),\mathbf{y}\sim p_{\mathrm{data}}(\cdot|\mathbf{x})}\left[\log p_\theta(\mathbf{y} \mid \mathbf{x})\right] \tag{1}$$

Then we would leverage a preference dataset to train a Reward model (which would be the right model in this case) to take in a preference dataset, which consists of a series of (x, y_w, y_l) tuples, where x is the prompt, y_w is the winning response, and y_l is the losing response as determined by Human Feedback, and then the Reward model must learn to output a reward given an (x, y) pair. It is typically trained with a simple BCE (Binary Cross Entropy) procedure. Then, for the last stage, we would use PPO (Proximal Preference Optimization) to update the left model to maximize reward from the Reward Model until the "right model is satisfied" with our responses with the following loss function:

$$L_{\mathrm{RL}}(\theta) = \mathbb{E}_{\mathbf{x}\sim q(\cdot),\mathbf{y}\sim p_\theta(\cdot|\mathbf{x})}\left[r(\mathbf{x},\mathbf{y})\right] - \lambda\mathbb{E}_{\mathbf{x}\sim q(\cdot)}\left[\mathrm{KL}\left(p_\theta(\cdot \mid \mathbf{x})\|p_{\mathrm{ref}}(\cdot \mid \mathbf{x})\right)\right] \tag{2}$$

Where we sample an input x and compute $x_{left} \sim \theta_{left}(x)$ then compute the reward for such a trajectory, $r(x_{left}, y_{true})$, which comes directly from the Reward model, and finally we want to regularize θ_{left} with KL divergence to ensure we do not stray too far from the initial distribution.

Now, the left model is an "agent model", ready to take prompts and output good responses.

However, this does not permit the left model to "learn to collaborate" with the right model in any way. Additionally, this requires a lot of human preference data which is very expensive to collect and subjective in many cases. Finally, this does not permit the left model to "lean on" the right model at inference time to assist in generating a response which results in hallucinations.

To solve this, we could think of a naive procedure where we learn both at the same time, and treat the SLN as one large joint network. Here we would forward pass in a loopy way, and make it end-to-end differentiable, and train via CrossEntropy loss at the very end of the forward pass similar to how SFT is done today, however that would require us to Back-Propogate Through Time (BPTT) which would require retaining the $m1 * m2 * d$ activations (or recompute them on the fly via "activation checkpointing" or otherwise), which in either case would be cost prohibitive and slow for large models.

We attempt this to prove the point where we must backpropogate through the sampling process of generating sequences. We accomplish this by viewing the output probability at each forward pass of the left model to be "an attention" over the vocabulary to produce a "soft token" via interpolation:

$$\hat{y}^{t+1} = \text{LLM}\left([x^1, \ldots, x^n, \hat{x}^1, \ldots, \hat{x}^t], \tau = 0.001\right) \tag{3}$$

$$\hat{x}^{t+1} = \text{V}\hat{y}^{t+1} \tag{4}$$

where the vocabulary matrix is of size embedding depth by vocabulary size, $V \in \mathbb{R}^{dv}$.

To get this to work, we must use a very small (τ) to ensure the vector $\hat{x}^{t+1}$ produces a vector close to a word in the vocabulary so it is interpretable by the right model. This leads to convergence.

As discussed above however, in our initial efforts training such a model, a small model (such as PYTHIA-70M) with batch size = 1 explodes to require a huge amount of VRAM (in this example 3.9GB of VRAM), which is a 30x increase in memory footprint to model size. If we were to use this on modern "small" models (i.e. of size 8B) this would require 240GB of VRAM which would be preventative.

To solve this issue, we could do a similar procedure, but leverage Truncated Backpropagation Through Time (TBPTT), where we would cut off backpropogation after the 'last t IDL' loops, however this would limit what the model learns. We want to inspire each model (left and right) from start to finish, and at each IDL step, to "nudge" the solution toward a better answer. And this will only penalize it for the last step.

Thus, we must try to avoid BPTT and preference datasets as much as possible.

5.2 GAN (Generative Adversarial Network) Setup

Inspired by the competitive nature of how GANs are trained in Computer Vision, we can map the training of our Strange-Loop Network to two separate networks competing with each other, one trying to generate trajectories, and the other trying to discriminate real-world trajectories from the "fake" trajectories.

In this way, we can pose the left lobe to be a "Generator", G, of sequences, to take in an input prompt x, and generate a sequence of tokens, $G(x)$, and the right lobe to be a "Discriminator", D, which would be a similarly sized model trained to be able to take in both real-world sequences of tokens, $y \in P_{data}$, or generated sequences $\hat{x} \in G(x)$ as input, and discriminate whether or not the input came from P_{data} or $G(x)$. It is important to note that similar to above, we need to convert the output probabilities of the LLM to sampled tokens. We can do so in a similar way as discussed above:

$$G(x) = V * LLM(x) \tag{5}$$

In this setup, each IDL would consist of m_{left} number of forward passes and then only $m_{right} = 1$ forward pass since the right lobe just needs to "judge" the entire output of the left lobe in one shot.

In this way, here we would train in the following procedure:

Algorithm 1 Generative Adversarial Network (GAN) Training Procedure for Strange Loop Networks

```
for number of training iterations $t = [1 - T]$ do
    Sample minibatch of $b$ samples $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(b)}, y^{(b)})\}$ from $P_{data}$
    Use $G$ to generate trajectories $\{(x^{(1)}, G(x^{(1)})), \ldots, (x^{(b)}, G(x^{(b)}))\}$.
    Update $D$ to learn to differentiate by:
        $$\nabla_{\theta_D} \frac{1}{b} \sum_{i=1}^{b} \left[\log D(y^{(i)}) + \log\left(1 - D\left(G(x^{(i)})\right)\right)\right]$$
    Update $G$ to learn to better fool $D$ next time:
        $$\nabla_{\theta_G} \frac{1}{b} \sum_{i=1}^{b} \log\left(1 - D\left(G(x^{(i)})\right)\right)$$
end for
```

We could also update x after each loop through the Strange-Loop to allow it to continuously loop and by updating x as follows:

x $\leftarrow$ concat$(x_{initial_prompt}, G(x), D(G(x)))$ (6)

This would allow the model to "learn" to take in feedback from the Discriminator from the last run to iteratively improve.

Note, however to compute the gradients for $D(G(x^{(i)}))$ we must again perform BPTT, first through D then repeatedly through the sampling process at each token generated in G , so $m1$ times.

So we are no better off in this setup unless we again employ Truncated BPTT which is not ideal.

5.3 Actor-Critic Setup

Another approach is to move toward a classic RL Actor-Critic setup where the left lobe (Actor) takes an input x of sequence length n tokens and generates a response of probabilities over the vocabulary $\hat{x}_{\text{left}}$ of sequence length m tokens which we sample with neural sampling (or any other sampling method) and tries to mimic the true (and desired) response y.

$$\hat{x}^0_{\text{left}} = \text{Actor}(x)$$

The right lobe (Critic) takes the combined input $x + \hat{x}^0_{\text{left}}$ and outputs a score probability $\hat{y}_{\text{right}}$.

$$\hat{y}_{\text{right}} = \text{Critic}(x + \hat{x}^0_{\text{left}})$$

We could allow the Actor to then "try again (and again and again)" with this new feedback appended to the context to improve its response the next time around in a Strange-Loop like so until the Critic is satisfied:

$$\hat{x}^t_{\text{left}} = \text{Actor}(x + \hat{x}^{t-1}_{\text{left}} + \hat{y}^{t-1}_{\text{right}})$$

To train the Critic, as we do not want to require a preference dataset for our procedure, we have no "external reward", so we could simply use binary cross-entropy loss to learn the "accuracy" of the critic's evaluation, or how likely this is a sample from the real distribution P_{data} or if it came from the Actor, which simplifies to:

$$\mathcal{L}_{\text{critic}} = -\log(\hat{y}_{right\_true}) - \log(1 - \hat{y}_{right}) \tag{7}$$

where $\hat{y}_{right\_true}$ is the Critic's score of the true sequence y_{true}, $Critic(y_{true})$.

Then to train the Actor, we could follow a similar procedure as PPO with our version of the trained Critic providing the reward.

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min\left(r_t(\theta)\hat{y}_{\text{right}}, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{y}_{\text{right}}\right)\right] \tag{8}$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} = \frac{\hat{y}_{\text{left},\theta}(a_t)}{\hat{y}_{\text{left},\theta_{\text{old}}}(a_t)}$

Then we would follow the training procedure in Algorithm 2: "Actor-Critic Training Loop for Strange Loop Network"

NOTE: If we do have an "external reward" r_t like in certain settings where we have a clear reward for a response (i.e. user thumbs up/down, playing a game, trivia, Winogrande, code setting where we can get reward if the code compiles, more reward if it passes more unit tests, etc). Then we could use a more standard setup for Actor-Critic such as:

1. $\hat{y}_{\text{right\_true}} \leftarrow \text{Critic}(x + y_{true})$
2. $\hat{y}_{\text{right}} \leftarrow \text{Critic}(x_{left})$
3. $\mathcal{L}_{\text{critic}} = -\log(\hat{y}_{right\_true}) - \log(1 - \hat{y}_{right})$
4. With the external reward, we can compute TD error: $\delta_t = r_t + \gamma\hat{y}_{\text{right\_true}} - \hat{y}_{\text{right}}$
5. Update critic (right lobe): $\theta_{Critic} \leftarrow \theta_{Critic} + \alpha_c \delta_t \nabla_\theta Critic(x_{left})$
6. Compute the policy gradient for Actor : $\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(x_{left} \mid x)\delta_t$
7. Update actor: $\theta_{Actor} \leftarrow \theta_{Actor} + \alpha_a \nabla_\theta J(\theta)$

Algorithm 2 Actor-Critic Training Loop for Strange Loop Network

```
for each epoch do
    Sample minibatch of $b$ samples $\{(x^{(1)}, y^{(1)}_{true}), \ldots, (x^{(b)}, y^{(b)}_{true})\}$ from $P_{data}$ where x is a prompt and $y_{true}$ is the desired response (sequence of tokens).
    for $x, y_{true}$ in batch do
        Start Internal Dialogue Loop
        for $i = 1$ to IDL_limit do
            $x_{left} = x$
            for $j = 1$ to Max_Gen_Seq do
                $x^{j}_{\text{left}} \leftarrow \text{Actor}(x_{left})$ {Generate next token}
                $x_{left} \leftarrow \text{concat}(x, x^{j}_{\text{left}})$
                if $x^{j}_{left} ==$ EOS then
                    break
                end if
            end for
            $\hat{y}_{\text{right}} \leftarrow \text{Critic}(x_{left})$
            if $\hat{y}_{\text{right}} \geq$ threshold then
                break
            else
                $x \leftarrow \text{concat}(x, x_{left}, \hat{y}_{\text{right}})$
            end if
            $\mathcal{L}_{\text{actor}} = L^{\text{CLIP}}(\hat{y}_{\text{left}}, y_{\text{true}}, \hat{y}_{right})$
            $\mathcal{L}_{\text{actor}}$.backward()
        end for
        $\hat{y}_{\text{right\_true}} \leftarrow \text{Critic}(x + y_{true})$
        $\mathcal{L}_{\text{critic}} = -\log(\hat{y}_{right\_true}) - \log(1 - \hat{y}_{right})$
        $\mathcal{L}_{\text{critic}}$.backward()
    end for
    optimizer_critic.step()
    optimizer_actor.step()
end for
```

5.4 Actor-Critic Setup with self-play

Inspired by the above and how humans learn better and better with mental visualization and repetition, we can try to learn again and again from simulating in the left lobe and measuring against the updated right lobe / reward model itself. Even if we do not have an explicit external reward signal, we could use the Critic as our reward model to update the policy for the Actor as the Actor "plays" and learns in the process. After the Critic has learned something, we could have the Actor iteratively generate trajectories (can be a different Strange-Loop iters, can be subsets of the final output, can also be sampled at different temperatures (τ)), and update the Actor weights (θ_{left}), until those trajectories achieve a satisfactory reward from the Critic.

This "self-play and learn" step would update the algorithm to the following:

1. As per the initial Actor-Critic algorithm, per batch $\{(x^{(1)}, y^{(1)}_{true}), \ldots, (x^{(b)}, y^{(b)}_{true})\}$ sampled from P_{data}, we update Critic with the same loss as before:
    (a) $\hat{y}_{\text{right\_true}} \leftarrow \text{Critic}(x + y_{true})$
    (b) $\hat{y}_{\text{right}} \leftarrow \text{Critic}(x_{left})$
    (c) $\mathcal{L}_{\text{critic}} = -\log(\hat{y}_{right\_true}) - \log(1 - \hat{y}_{right})$
    (d) Update $\theta_{Critic} \leftarrow \theta_{Critic} + \alpha_a \nabla_\theta \left(\mathcal{L}_{\text{critic}}(\theta)\right)$
2. While $\hat{y}_{right}$ ¡ thresh:
    (a) Let Actor(x) "self-play and learn" until it the Critic is happy
    (b) Sample $x_{left} \sim Actor(x)$
    (c) Compute $\hat{y}_{\text{right}} \leftarrow \text{Critic}(x_{left})$
    (d) Compute the policy gradient: $\nabla_\theta J(\theta) = \nabla_\theta \log \pi_\theta(x_{left} \mid x)\hat{y}_{\text{right}}$
    (e) Update $\theta_{Actor} \leftarrow \theta_{Actor} + \alpha_a \nabla_\theta \left(J(\theta)\right)$

This has a natural intuition. As we learn a new topic, or new skill, we need rapid iterations to really learn it, try and try again until its "imprinted" in our minds (left lobe in particular). However, in this setup, we run the risk of "overfitting" the Actor to this minibatch if we are updating the Actor many more times per minibatch vs. the Critic which is only updated once.

We can ameliorate this in a few ways:

1. With large enough model-size, we are at less risk of overriding previous weights so less overfitting will occur.
2. By increasing batch size sufficiently we better align the surrogate loss function with the global loss function.
3. Weight regularization
4. Gradient clipping

5.5 Actor-Critic Setup with self-play and Pavlovian Gradient Decay

Finally, we have one large issue not addressed yet. When updating the Critic with "classification", first, we need to ensure that the Critic is robust to small "augmentations" of the input sequence (x) (and continuous subsets of (x) thereof), and also we need the minibatch to equally represent both positives and negatives samples. We can sample millions of negatives per positive, but we only have one true positive.

So far we only have one positive per sample, and many negatives sampled from the Actor.

1. Positive Samples:
    (a) $y_{true} = [S_1, S_2, \ldots, S_n]$
2. Negative Samples:

(a) x^T_{left} which is the last iteration of the Strange-Loop's left lobe response.

Instead, we can explode both positives and negatives to stabilize (and accelerate) training by doing the following.

Given a sample(x, y_{true}) we can explode this to:

1. Positive Samples:
   (a) $y^0_{true} = [S_1]$
   (b) $y^1_{true} = [S_1, S_2]$
   (c) ...
   (d) $y^n_{true} = [S_1, S_2, \ldots, S_n]$
2. Negative Samples (can be subsets of the list below, can also be sampled at different temperatures (τ)):
   (a) x^0_{left} which is the first iteration of the Strange-Loop's left lobe response.
   (b) x^1_{left} which is the second iteration of the Strange-Loop's left lobe response.
   (c) ...
   (d) x^T_{left} which is the last iteration of the Strange-Loop's left lobe response.

As a quick tangent, in "classic Pavlovian conditioning", when an Actor is given an external reward, the Actor then associate previous states with higher value or higher expected future reward. Learning rate (rate of conditioning) is inversely correlated to Interstimulus Interval (ISI) or $\Delta_T = t_{reward} - t_{stimulus}$. The smaller ISI, the more learning occurs. It actually follows this relationship.

$$C(\Delta T) = C_0 \cdot e^{-\lambda \Delta T} \tag{9}$$

where, C_0 is the maximum conditioning strength possible by the experiment (100% for example), and λ is the decay constant that determines how quickly the conditioning strength decreases. Typical λs in humans is 0.1 to 0.5 providing the "optimal" learning to happen at a ΔT of 1-4 seconds. In RL, when we "shape" rewards via an exponential decay function to prioritize recent experiences over older ones, this is called "Exponential Decay Reward Shaping". We are doing this somewhat to sample the positives, however, in our posing, we are not shaping any reward, but instead we are decaying the weighting of the classification gradients of the Critic. Additionally, in our sampling of negatives, we pose "recent" not to be in the "time" domain, but the "thought" domain or "compute" domain. We decay gradient as we go back to previous iterations of the Strange-Loop. So to differentiate these techniques, we call our approach "**Pavlovian Gradient Decay**".

Thus, we can weight the value of the gradient's impact on learning by such an exponential decay function in time updating the "reward" of a response to the following.

Given a sample(x, y_{true}) we can explode this to:

1. Positive Samples: (Exponential Decay Reward Shaping)
   (a) $y^0_{true} = [S_1]$, $r(\Delta T = n - 0) = C_0 \cdot e^{-\lambda(n-0))}$
   (b) $y^1_{true} = [S_1, S_2]$, $r(\Delta T = n - 1) = C_0 \cdot e^{-\lambda(n-1))}$
   (c) ...
   (d) $y^n_{true} = [S_1, S_2, \ldots, S_n]$, $r(\Delta T = n - n) = C_0 \cdot e^{-\lambda(n-n))}$
2. Negative Samples: (Pavlovian Gradient Decay)
   (a) x^0_{left} which is the first iteration of the Strange-Loop's left lobe response, $r(\Delta T = n - 0) = 1 - C_0 \cdot e^{-\lambda(n-0))}$
   (b) x^1_{left} which is the second iteration of the Strange-Loop's left lobe response, $r(\Delta T = n - 1) = 1 - C_0 \cdot e^{-\lambda(n-1))}$
   (c) ...

(d) x_{left}^{T} which is the last iteration of the Strange-Loop's left lobe response, $r(\Delta T = n - n) = 1 - C_0 \cdot e^{-\lambda(n-n))}$

For the positives, this will allow the right model to learn if the left model is "on the right track" while generating tokens, but not penalize it at very early steps of sequence generation too much.

For the negatives, we should not penalize the left model as much for earlier iterations of its response and allow the model to 'figure it out' by thinking. Especially because we WANT the model to learn to iterate through the strange-loop a bit before emitting a confident answer.

Now, how to scale this training to enterprise scale models. Almost all of the steps are remarkably parallelizable across nodes. Much like AlphaGoZero, you can take each sample (x, y_{true}) and produce trajectories as described above (in parallel), then score those trajectories with the Critic and backprop to accumulate gradient (in parallel), then gather$_alltoupdatetheCritic(inseries).NowthatwehaveanupdatesCriticmodel,w$
and Strange-Loop steps) as described above (in parallel), then score those trajectories with the updated Critic (in parallel), accumulate gradient for the Actor (in parallel), and then gather$_allandupdatetheActor.Thenrinsewashr$
$playandaccumulategradientinthewaydescribedabovesomayserveasaproxyforwhatwouldberequiredtotrainthismodel.$

TODO Describe LLaMAC (Large Language Model-based Actor-Critic), where they employed 3 actors (left, right, and judge), they perform such a procedure.

5.6 SPIN (Self Play Fine Tuning) Setup

Finally, SPIN. SPIN permits a DPO-like training that does not require an explicit, human labeled, preference dataset, or an explicit reward model, which is ideal.

In this setup, we can train the left and right lobes in the following way:

Algorithm 3 Self-Play Fine-Tuning (SPIN)

```
Require: ${(x_i, y_{true_i})}_{i \in [N]}$: SFT Dataset, $p_{\theta_0}$: LLM with parameter $\theta_0$, $T$: Number of iterations.
for $t = 0, \ldots, T-1$ do
    for $i = 1, \ldots, N$ do
        Generate initial synthetic sample of $x_i$ via $\hat{x}_{left_i}^{0} \sim \theta_{left_t}(x_i)$
        Generate initial scoring of the response of $\hat{x}_{left_i}^{0}$ via $\hat{y}_{\text{right}_i}^{0} \sim \theta_{right_t}(\hat{x}_{left_i}^{0})$
        Start Internal Dialogue Loop to generate more samples
        for $j = 1$ to IDL_limit do
            $x_i^j \leftarrow \text{concat}(x_{initial\_prompt}, \hat{x}_{left_i}^{j-1}, \hat{y}_{\text{right}_i}^{j-1})$
            Generate more loopy synthetic sample of $x_i$ via $\hat{x}_{left_i}^{j} \sim \theta_{left_t}(x_i, \tau = [0.1 - 0.9])$
            Generate score of the response of $\hat{x}_{left_i}^{j}$ via $\hat{y}_{\text{right}_i}^{j} \sim \theta_{right_t}(\hat{x}_{left_i}^{j})$
        end for
    end for
    $\theta_{left_{t+1}} = \arg\min_{\theta \in \Theta} \sum_{i \in [N]} \sigma\left(\lambda \log \frac{p_\theta(y_{true_i}|x_i)}{p_{\theta_{left_t}}(y_{true_i}|x_i)} - \lambda \log \frac{p_\theta(y_i'|x_i)}{p_{\theta_{left_t}}(y_i'|x_i)}\right).$
    $\theta_{right_{t+1}} = \arg\min_{\theta \in \Theta} \sum_{i \in [N * \text{IDL\_limit}]} -\log(\hat{y}_{right\_true}^{i}) - \log(1 - \hat{y}_{right}^{i})$
end for
Ensure: $\theta_{left_T}$
Ensure: $\theta_{right_T}$
```

6 Results

TODO :) single LLM (A), a homogenous LLM-based Single Agent with self-reflection (B), a homogenous LLM-based Multi Agent System with self-reflection (C), and finally our Strange-Loop Networks, a heterogeneous, and jointly trained LLM-based Multi Agent System with self-reflection to form a "strange-loop" in both tall (D) and wide setup (E).

	MBPP	HumanEval	GSM8K	Math
Single LLM (Llama3 8B)	—			
Single LLM (Llama3 8B) with Reflection				
Single LLM (Llama3 8B)				
SLN				
	—			

Table 1: Results Table.

Item	Quantity
Widgets	42
Gadgets	13

Table 2: An example table.

7 Analysis

7.1 Paper 1

(TODO)

8 Discussion

8.1 Paper 1

—

9 Conclusion

Therefore, the specialization and interplay of the two lobes is paramount to self-play, learning, self-reflection, abstraction, and full generalization.

10 APPENDIX

10.1 How to add Tables

Use the table and tabular environments for basic tables — see Table 2, for example. For more information, please see this help article on tables.

10.2 How to add Citations and a References List

You can simply upload a `.bib` file containing your BibTeX entries, created with a tool such as JabRef. You can then cite entries from it, like this: [Gre93]. Just remember to specify a bibliography style, as well as the filename of the `.bib`. You can find a video tutorial here to learn more about BibTeX.

If you have an upgraded account, you can also import your Mendeley or Zotero library directly as a `.bib` file, via the upload menu in the file-tree.

10.3 Good luck!

We hope you find Overleaf useful, and do take a look at our help library for more tutorials and user guides! Please also let us know if you have any feedback using the Contact Us link at the bottom of the Overleaf menu — or use the contact form at `https://www.overleaf.com/contact`.

References

[Gre93] George D. Greenwade. The Comprehensive Tex Archive Network (CTAN). *TUGBoat*, 14(3):342–351, 1993.