Sequential and Associative Learning in State Space Models with One Big State

Furkan Ozyurt
New York University
New York, USA
fo2109@nyu.edu

February 18, 2025

Abstract

The ability to process data sequentially plays an important role in the daily tasks that require reasoning and planning. A model architecture that processes the data heavily in parallel such as a transformer may not perform well in tasks that require reasoning and planning because these tasks require familiarity with taking an action, observing the outcome of that action, and taking another action in a sequential manner. During this process, there is a strong dependency between each consecutive step and a model that gets used to processing the data heavily in parallel may not catch all of these dependencies because some of these dependencies might be ignored during the training process to be able to achieve parallelism. This may result in a model that is good at associating different patterns in the input and bad at planning a course of action. Another key feature that is as crucial as sequential data processing is the ability to associate different patterns in the data with different parts of the internal representation of the real world because associative learning is crucial to catching the patterns in the events that seem unrelated to each other. Therefore, an intelligent system should have both the capability of sequential data processing and associative learning. One way to develop this system might be utilizing a state space model and integrating an associative learning mechanism into this model in such a way that the new information can be processed sequentially and different parts of the state can be associated with the other parts that are relevant to this new information.

1 Introduction

Today, many of the popular AI models still do not engage with the real world as much as we do and they are not designed to continuously learn from the data in real-time on their own. The backbone of some of these models is the transformer [1] model, which is ideal to be trained in a highly parallelized manner.

When we use sequential data such as video, which can be seen as sequential images, as input to the transformers, there is no need to wait for the patterns from the previous image(s) to be reflected in some common space that acts like a memory (like hidden state in RNNs). This makes the transformer model highly parallelizable because there is no need to process the information in the current step into some common space before proceeding to the next step and the inputs are processed as a whole.

However, because of this mechanism, the patterns in the previous step(s) are not reflected in the internal representation of the model sequentially. Instead, the patterns in a group of multiple distinct inputs (batches) are reflected in this internal representation all together in an associative manner. Because of this training process, the sequential dependencies in the data might be ignored, and this may cause the model to need much more data and supervision from us than necessary to capture those missing sequential dependencies and develop chain-of-thought reasoning.

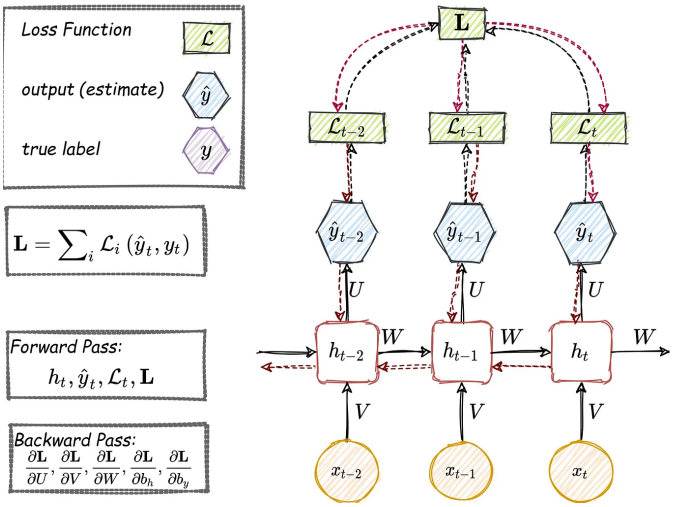

Figure 1: RNN [2]

As we make the architecture of the model more and more parallel for the sake of speeding up the training, we may end up damaging the potential chain-of-thought reasoning capability of the model [3] [4]. In the end, if we define planning as making sequential decisions step by step, how can a model that is trained in a highly parallel manner, that processes inputs as whole rather than step by step, and that tends to ignore the sequential dependencies can be good at planning on its own without being heavily supervised by us and/or trained on a very large amount of data?

In RNNs, the patterns in the incoming data are reflected in some common space (hidden state) in each step before processing the new information. In addition, the entire hidden state that contains the patterns of the information from the previous steps is transformed based on the new information in the current step and the current version of the hidden state. Therefore, the hidden state is different for each step. For instance, at time step $t = 2$, the hidden state contains all the patterns that were observed until $t = 2$ and does not contain any pattern that belongs to $t = 3$ or $t = 4$, unlike transformers. At the end of the sequence, the model parameters are updated with backpropagation through time (BPTT) step by step based on the predicted output and the hidden state that was used to predict output in each step. This process can be seen in Fig. 1.

However, the issue with RNNs is that it does not have a strong associative learning mechanism, unlike transformers. Associative learning is simply learning to associate different patterns with each other. When a baby moves his head to a refrigerator and looks at it, the visual information of the refrigerator he is exposed to at that moment fires a group of neurons in his brain. If he is exposed to air vibrations that represent the sound of "refrigerator" during this moment, this auditory information wires a bundle of some other brain regions with the brain regions that were fired after seeing the refrigerator. This is called associative learning and in this example, the patterns in the sound of "refrigerator" are associated with the patterns in the visual representation of the refrigerator. In this way, when the baby hears only the word "refrigerator" later, he can remember which object that word refers to and turn his head to the refrigerator because of associative learning. A similar mechanism should be part of an artificial intelligence system if we want it to be good at catching the patterns in the events that seem unrelated to each other. Hopfield networks [5] and attention mechanism [1] are some of the artificial mechanisms that can fill the role of associative learning in artificial intelligence systems. However, a better and more efficient way to associate different patterns in the data might be relying on the nature of the sequential data in real life.

In addition, in RNNs, the hidden state, which acts as working memory, is transformed in each step entirely. Considering that the incoming data's level of novelty and compatibility with the existent state may not be very high in each step, it may be wiser to use one big state from beginning to end, detect the parts in that state that are relevant to the incoming data, and only transform these parts in the state in each step. This approach may provide better propagation of the information since we won't have to constantly try to update the entire state in every single step.

Depending on its frequency of arrival, data can be classified as either continuous or discrete. If we divide the continuous data into fixed intervals, there is usually a strong dependency between the current view of the data and the next view which makes it more suitable to be processed sequentially with some kind of working memory. However, this is not always the case in discrete data types

such as speech sounds that aren't exposed continuously. Continuous data types are expected to leave sequential patterns on the internal representation of the model while discrete data types are expected to associate different patterns in the internal representation of the model. So, this means that the impact of discrete data on the internal representation of the model should be different from the impact of continuous data. In addition, the real world is full of continuous and discrete data types. Because of these reasons, an intelligent system should be capable of processing both of these data types and learning to transition from a continuous state to a discrete state or from a discrete state to a continuous state properly. This means that we need a discretization mechanism that transforms the continuous representations of the relevant patterns into some discrete space in which taking an action or making a decision is possible and efficient.

That's why instead of using RNNs, we can use a state space model, utilize one big state during its training process, and integrate associative learning and discretization mechanisms into it. In this paper, the goal is to propose how this system can be implemented.

2 Data

To test this new learning system, it is important for the data we use to be multi-modal and continuous/sequential and to be taken from real life. So, we can use SAYCam: A large, longitudinal audiovisual [6] dataset recorded from the infant's perspective to train these models.

3 Methods

Because of the reasons explained in the Introduction section, a new mechanism that is capable of sequential, associative, and continual learning is proposed in this paper. The architecture of this framework can be seen in Fig. 2.

In this framework, there is one big common state that is not transformed entirely during the training process. We can think of this state as a very large 2-dimensional matrix that is initialized randomly at the beginning. An element x_{ij} in this 2-dimensional matrix represents the strength of the connection (weight) between nodes i and node j. And we can use this state to represent/store the patterns in different formats of data associatively.

Because applying chains of matrix multiplication to entire state in every single step might be very inefficient and this approach may remove some patterns in the state in time, we may prevent this inefficiency and observe more permanent and associative memory by transforming only the parts in the state that are relevant to the incoming data in each step. Here is how we can achieve this: Let's call the groups of nodes that represent a specific pattern "circuit".

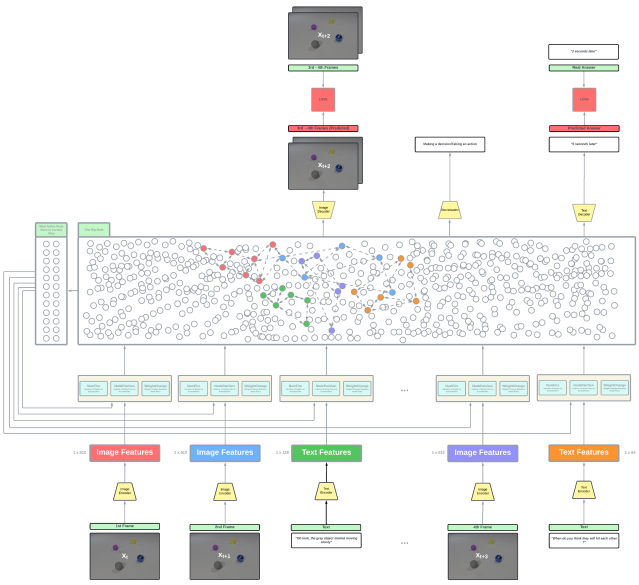

Figure 2: One Big State

When a new piece of information is detected in the current step, the features that represent this new piece of information are extracted. These features are then passed as input to a dedicated module responsible for determining

- the number of nodes that should be activated (some features may be complex and they may need to be represented by a large number of nodes, while some other features may be less complex and can be stored/represented in a smaller number of nodes),
- the indices of specific node pairs to activate, and
- the amount of changes in the weights between these specific nodes (a positive change means increasing the strength of the connection between a pair of nodes while a negative change means decreasing the strength of the connection between a pair of nodes)

By training this separate module, we can have a model specialized in distributing the patterns in different formats of data to the appropriate regions in one big state, and associating different formats of data with each other.

In addition, the information of the most active nodes in the one big state in the current moment can also be used as input to this module so that when the next piece of information arrives and the features of this new piece of information are extracted, this module can learn to represent these features in

such a way that the node groups that will store the patterns of this new piece of information are strongly connected with the node groups that were highly active when the new piece of information arrived.

This framework can be seen as a quite practical and efficient associative learning mechanism that doesn't need the computation of scaled dot-product. Because let's assume that a robot saw a refrigerator in the current step. The features of the image of the refrigerator are extracted with an image encoder, and the patterns in this image are represented in the weights of the specific pairs of node groups. In the next step, when the robot hears the word "refrigerator" from someone, the new patterns in this sound are encouraged to be represented in a group of nodes that have connections with the nodes that represent the image of the refrigerator because these nodes were highly active in the one big state when the sound of "refrigerator" was heard. In this way, the patterns of the visual scene of the refrigerator are associated with the patterns of the sound of "refrigerator".

Or, if a robot takes an action and moves an object in the table at the current step t, the nodes that made the robot move the object in the table will be highly active in the current moment. After this action, the new position of the object and the rest of the visual scene will be observed by the robot and the patterns in this new visual scene will be associated with the nodes that made the robot take specific actions in the previous step. That's why, the new representation of the new visual scene will be associated with the action taken in the previous step and in this way the model will be able to learn to associate the outcomes of different actions seamlessly because of the sequential nature of the real-time data, and the framework that is suitable to process sequential data.

Another advantage of this mechanism is that it may allow us to keep track of one big state with another separate model in different ways. For instance, we can develop another model that keeps track of the firing patterns in specific nodes or the firing order of the circuits and this model can call these circuits in the appropriate order when necessary or further update the one big state based on the patterns in the firing order of the circuits.

In addition, this framework allows for both parallel and sequential processing smoothly. At time step t, we can store the parameters of the module that determines the number of nodes to fire, specific node groups to fire, and the change in the weights of these node groups in multiple GPUs along with the current version of the one big state. And the amount of changes to the connection strengths between different node groups can be determined in parallel in each GPU node. Because we are not doing matrix multiplication, summing these updates coming from multiple GPU nodes is a good way to represent all of the patterns that were caught in parallel. In addition, we can also develop another mechanism that will determine when to process the data in parallel and when to process the data sequentially.

Lastly, in this new framework, we may try to enforce the strength of the connections (weights) between each node pair to have Gaussian distribution because this may encourage the model to store the patterns in various formats of data more accurately and efficiently.

4 Experiments

5 Conclusion

References

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[2] M. A. Intelligence, "The exploding and vanishing gradients problem in time series," 2020. [Online]. Available: https://medium.com/metaor-artificial-intelligence/the-exploding-and-vanishing-gradients-problem-in-time-series-6b87d558d22

[3] W. Merrill and A. Sabharwal, "The expressive power of transformers with chain of thought," in *The Twelfth International Conference on Learning Representations*, 2024. [Online]. Available: https://openreview.net/forum?id=NjNGlPh8Wh

[4] W. "Merrill and A. Sabharwal, "The parallelism tradeoff: Limitations of log-precision transformers," *Transactions of the Association for Computational Linguistics*, vol. 11, pp. 531–545, 2023. [Online]. Available: https://aclanthology.org/2023.tacl-1.31/

[5] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, no. 8, pp. 2554–2558, 1982.

[6] J. Sullivan, M. Mei, A. Perfors, E. Wojcik, and M. C. Frank, "Saycam: A large, longitudinal audiovisual dataset recorded from the infant's perspective," *Open Mind*, vol. 5, pp. 20–29, 05 2021. [Online]. Available: https://doi.org/10.1162/opmi_a_00039