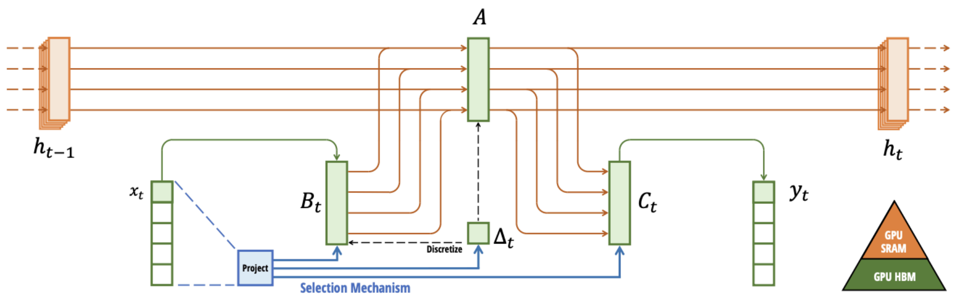
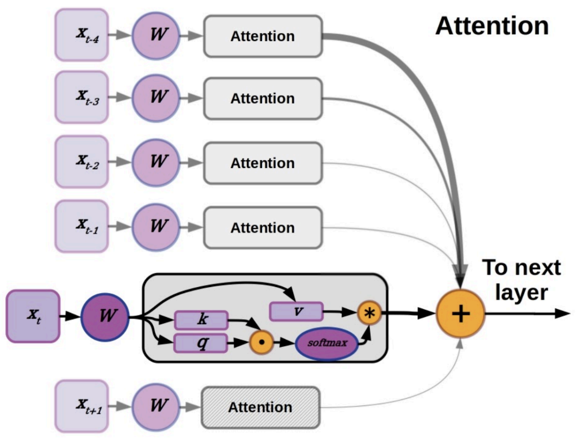
Mamba Series Intro

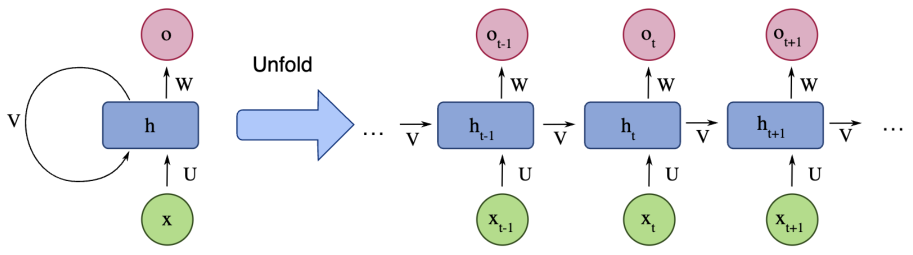


As promised, today we're starting our series(in English) of short paper reviews that will ultimately lead us to the cherry on top: a description of an architecture named Mamba that has recently made quite a buzz. Today, I'll give you a general introduction to what we'll be discussing over the next week.

So, what is Mamba? As mentioned, Mamba is a neural network architecture for sequential data designed to address the significant shortcoming of transformers, which is their quadratic complexity in terms of input length (for textual input, the length is the number of tokens in the context window). Although several improvements to the transformer's attention mechanism (which lies at the core of the issue) like FlashAttention2 have been proposed recently, transformers still struggle to work optimally with data that has context lengths on the order of millions of tokens.



So, how do we deal with the transformers' attention mechanism's quadratic complexity? The younger among us remember that we once had an architecture called RNN (Recurrent Neural Networks) for sequential data processing, which didn't have the problem of quadratic complexity (there were other drawbacks that we'll discuss later). RNNs and their refinements, such as GRU and LSTM, didn't need to account for the representation of all sequence elements (e.g., text tokens or image patches) explicitly as opposed to the transformer's attention mechanism. Instead, it compressed the information about the tokens (representations and their relationships) through a state vector (in LSTM, there are additional vectors for more efficient memory compression).



So, if we manage to compress all the essential information contained in the previous sequence elements well, then we don't need to consider the previous sequence elements during training and inference (next token prediction). Indeed, if “all we need to know” about the past sequence elements is contained in this compressed representation(=memory), then we'll use it instead of explicitly considering the previous elements.

However, this approach has two main flaws:

- RNN-anchored architectures failed to compress the previous tokens well when the context window length was long, which was the main reason these architectures failed in tasks involving processing long text segments. After all, if the model can't remember the essential information from the previous elements, it can't be expected to perform well in predicting the next element.
- You've probably heard that RNN architectures are not scalable. What does that actually mean? When we train a language model, the task is to predict sequence elements(=tokens) that we hide (mask) from the model. With transformers, we have the ability to predict all the hidden tokens simultaneously by using different masks each time (masking only what's needed). With RNNs, this is impossible because, for predicting each token, we need to calculate the memory representation that takes into account all the tokens that came before it. That is, for token number 1000, we need to perform sequential calculations (one after another) for 999 tokens that came before it. This action is not parallelizable due to the presence of non-linear functions in calculating the memory representation

(Mamba elegantly bypasses this obstacle). Of course, this is inefficient and poses a significant obstacle to the efficient use of computational resources (GPUs).

Now the question arises, can we overcome the quadratic complexity of transformers and at the same time get rid of the two disadvantages described in the previous paragraph? This is exactly the main research question that will be discussed in the following reviews that will lead us to the coveted Mamba architecture.

Finally, I'll give you a few small teasers about what you're going to see in the coming days:

The architectures we're going to talk about allow two modes of operation:

- Parallel training like with transformers during model training
- Fast inference like with RNNs

Surprisingly, this architecture has a structure similar to a convolutional network, only that the kernels of these convolutions are very long

These architectures draw inspiration from linear dynamic systems and are also connected to function approximation by orthogonal polynomials.