LEARNING FROM UNLABELED DATA BY EXPLOITING STRUCTURE WITHIN THE DATA ITSELF.

Self-Supervised Learning (SSL) Architectures for Time-Series Data: RNNs, LSTMs, and State Space Models

Reminders

- Your project title and abstract are due today. I will read your abstract and give you feedback on the project
- Thanks for booking your bi-weekly project zoom meetings with me. (Note: The meetings are on zoom unless you explicitly ask for an in-person meeting.) We shall use the *Zoom link for office hours* (see class webpage).
- Student-led talks:
  - Your presentation topics have been assigned. Each project group is assigned one topic (50 min talk + 10 min for Q&A). Please go to the "schedule" page to find out your topic.
  - Each group is free to survey the topic on their own and choose approximately 6 papers to present. (AI help is allowed.)
  - I will contact your group roughly two weeks before your presentation to recommend papers and finalize your paper selection.
  - Selected papers are shared on the website roughly one week before the talk.

Components of Self-Supervised Learning

Part I: The tokenizer

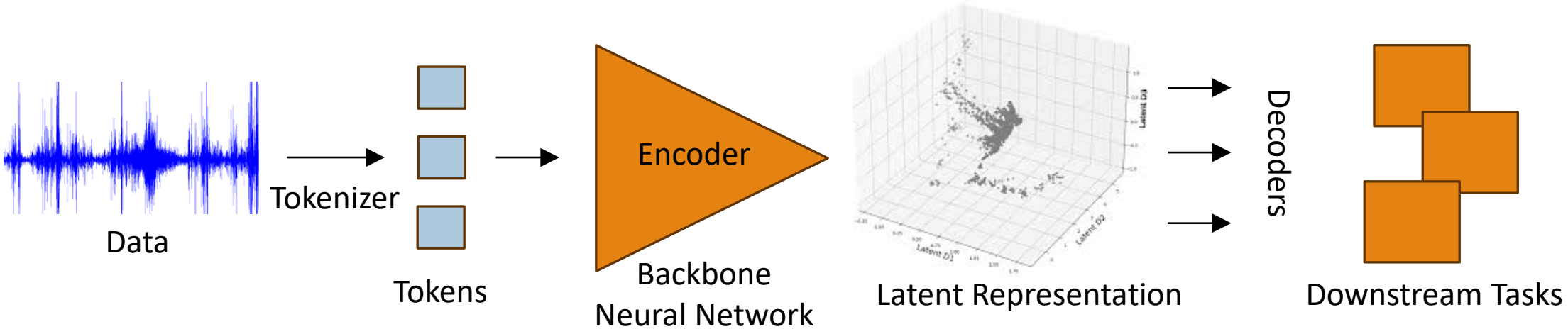
- Breaks the input stream into pieces to be individually encoded.

Part II: The backbone neural network model (or encoder)

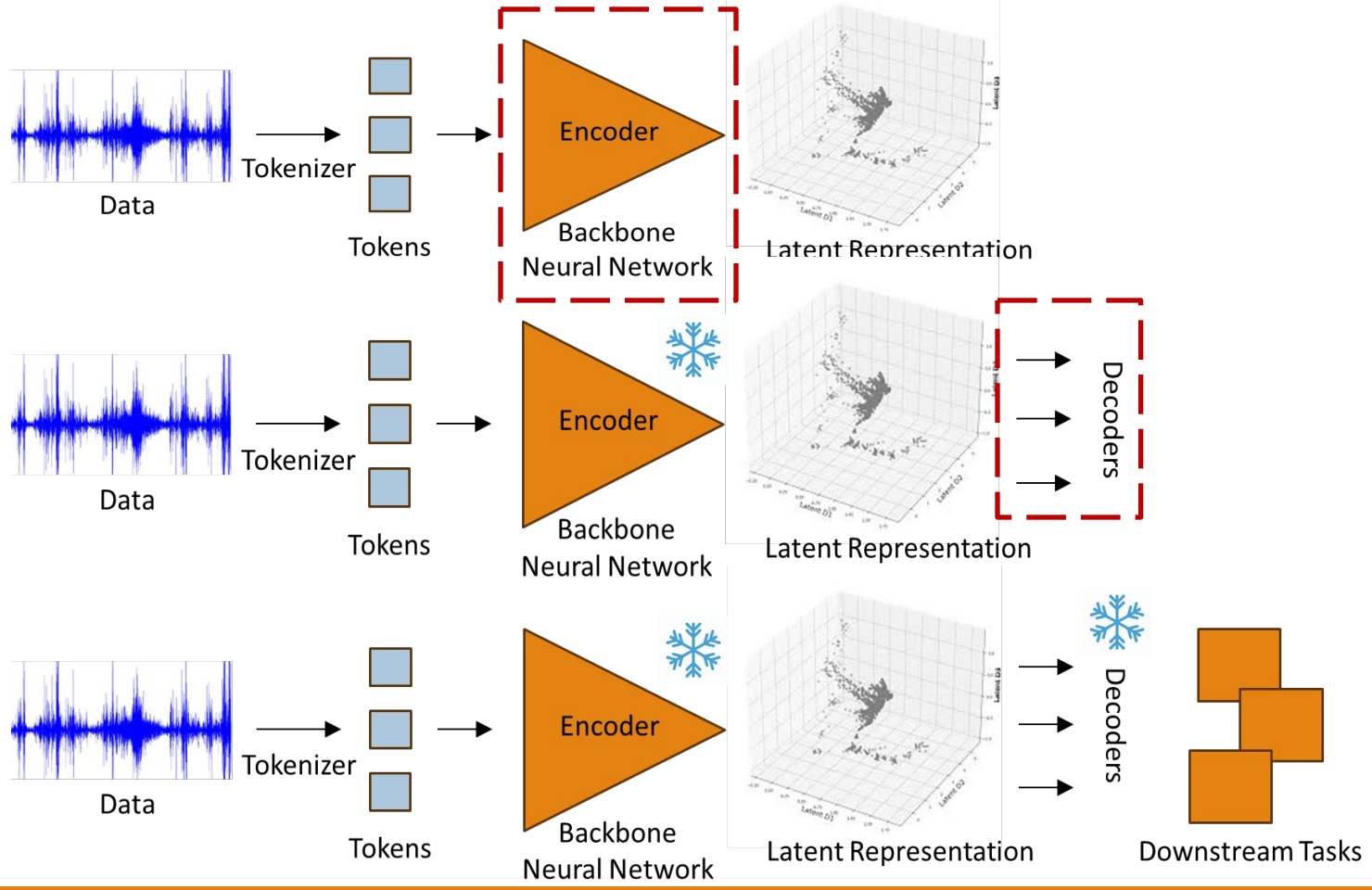
- The neural network that converts the stream of input tokens into a latent representation

Part III: The pretext task(s)

- The task that trains the encoder



Stages of Self-Supervised Learning



Training (with pretext task)

Fine-Tuning (supervised)

Testing/Deployment

Pretext Tasks and Representation Collapse

Poorly designed pretext tasks can lead to representation collapse, where everything gets encoded into the same latent representation

Example from Social Graphs (for Analysis of Polarization)

- Nodes represent people and content. Connections represent endorsement.
- Two nodes are put closer in the latent space if they are more tightly connected by social or information links (e.g., if they endorse the same content). Disconnected nodes add a “repulsion” force
- A node is put closer to the origin if it is more tightly connected to more mutually disconnected nodes

Emerging embedding properties:

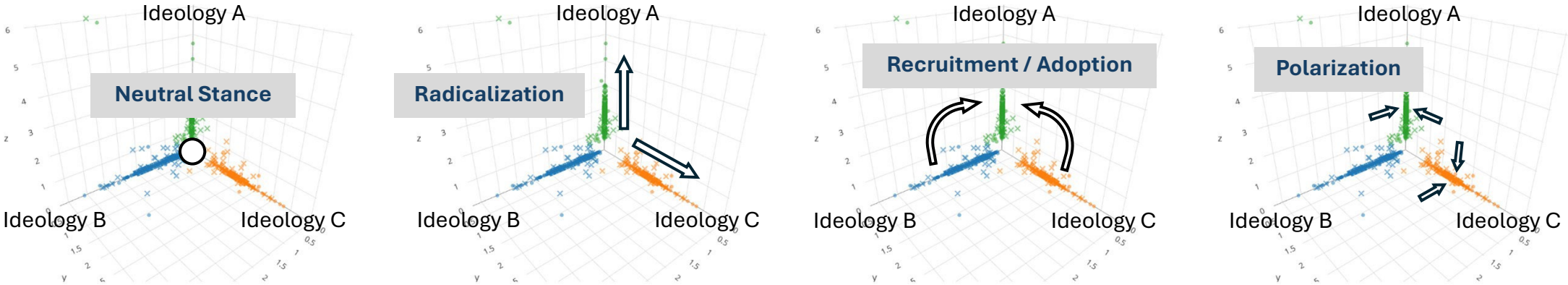


Figure: **Notional illustration of different dynamics in the latent belief space.**

Pretext Tasks and Representation Collapse

Poorly designed pretext tasks can lead to representation collapse, where everything gets encoded into the same latent representation

Graph example:

- Nodes represent people and content. Connections represent endorsement.
- Two nodes are put closer in the latent space if they are more tightly connected. ~~Disconnected nodes add a "repulsion" force~~
- A node is put closer to the origin if it is more tightly connected to more mutually disconnected nodes

Outcome?

Pretext Tasks and Representation Collapse

Poorly designed pretext tasks can lead to representation collapse, where everything gets encoded into the same latent representation

Graph example:

- Nodes represent people and content. Connections represent endorsement.
- Two nodes are put closer in the latent space if they are more tightly connected. ~~Disconnected nodes add a "repulsion" force~~
- A node is put closer to the origin if it is more tightly connected to more mutually disconnected nodes

In this example, all nodes will collapse to the origin of the latent space.

Preventing Collapse: Contrastive Learning Insights

- Inspired by insights from the graph example (in this case, the need for a “repulsion force”), what is the equivalent of the repulsion force in contrastive learning?

Contrastive Learning Insights

- Finding hard negative samples (that need to be separated in the latent space)!
  - What are examples of dissimilar samples (those that must be far apart in the latent space) that are not obviously different from superficial inspection?

Contrastive Learning Insights

- Finding hard negative samples!
  - What are examples of dissimilar samples (those that must be far apart in the latent space) that are not obviously different from superficial inspection?
  - Example: Consider a model for human response prediction to information stimuli (e.g., what a person of a given embedding think of “abortion”, “gun control”, “immigration”, etc).
  - How best to construct a contrastive pair?

Adam	**Sally**	**Ash**	**John**
Male	Female	Non-binary	Male
25 years old	40 years old	18 years old	25 years old
Brown eyes	Green eyes	Blue eyes	Brown eyes
Brown hair	Blonde hair	Black hair	Brown hair
5’ 2”	5’ 10”	6’ 1”	5’ 2”
Liberal	Conservative	Liberal	Conservative

Contrastive Learning Insights

- Finding hard negative samples!
  - What are examples of dissimilar samples (those that must be far apart in the latent space) that are not obviously different from superficial inspection?
  - Example: Consider a model for human response prediction to information stimuli (e.g., what would a person of a given embedding think of “abortion”, “gun control”, “immigration”, etc).
  - How best to construct a contrastive pair?

Adam
Male
25 years old
Brown eyes
Brown hair
5’ 2”
Liberal

Sally
Female
40 years old
Green eyes
Blonde hair
5’ 10”
Conservative

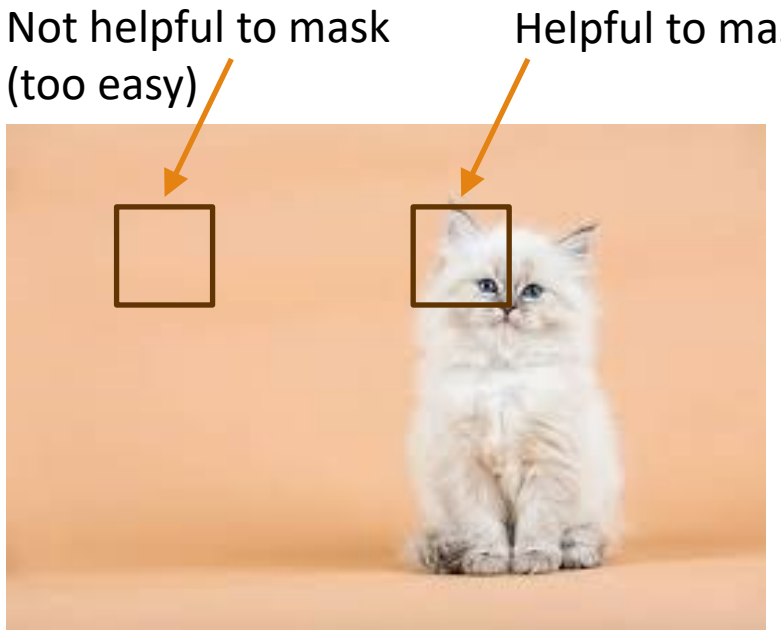
Ash
Non-binary
18 years old
Blue eyes
Black hair
6’ 1”
Liberal

John
Male
25 years old
Brown eyes
Brown hair
5’ 2”
Conservative

Hard negatives help prevent representation collapse

Preventing Collapse: Masking/Prediction Insights

- Masking (and reconstruction) of more informative parts of the input leads to better learning

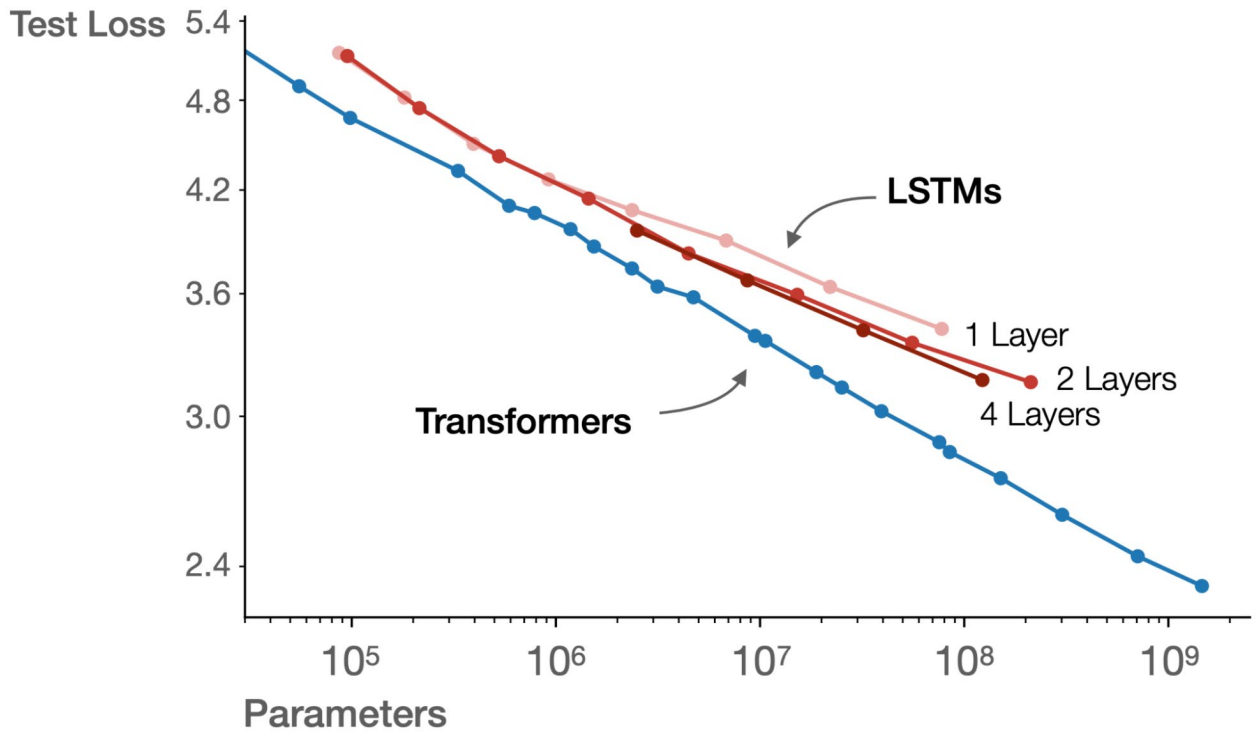


A good masking policy design can help prevent representation collapse

Scaling and Overfitting

Common wisdom: Larger models will do better.

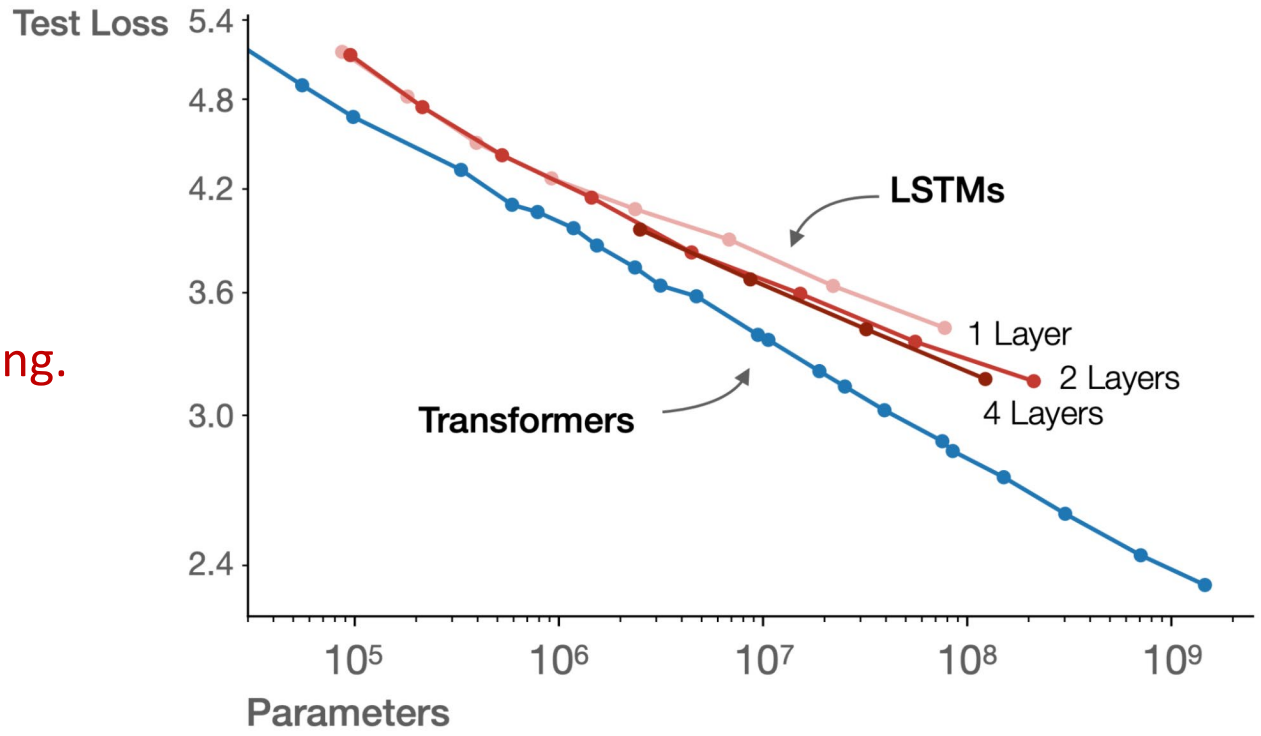
Caveat?



Scaling and Overfitting in IoT Data

Common wisdom: Larger models will do better.

Caveat: Must have enough data to train.
Otherwise, larger models will cause overfitting.



https://www.aisafetybook.com/textbook/scaling-laws

Self-Supervised Learning Architectures for Time-Series Data: RNNs, LSTMs, and State Space Models

A key characteristic of IoT data is that it usually comes from sensors

→ Sensors produce continuous streams of time-series data

What neural network (encoder) architectures are best suited to model *time-series data*?

[Background] Analytical Models of Time-Series Data: Auto-regressive Moving Average

Auto-regressive Moving Average Equation:

$$y(k) = \sum_{i=1}^{p} a_i \, y(k-i) \; + \; \sum_{j=0}^{q} b_j u \, (k-j)$$

where:

- $y(k)$: system output (or measured signal)
- $u(k)$ Input
- a_i: autoregressive coefficients
- b_j: moving-average coefficients

State-space representation:

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k) = Cx(k) + Du(k)$$

[Background] Analytical Models of Time-Series Data: Auto-regressive Moving Average

Auto-regressive Moving Average Equation:

$$y(k) = \sum_{i=1}^{p} a_i \, y(k-i) \; + \; \sum_{j=0}^{q} b_j u \, (k-j)$$

where:

- $y(k)$: system output (or measured signal)
- $u(k)$ Input
- a_i: autoregressive coefficients
- b_j: moving-average coefficients

State-space representation:

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k) = Cx(k) + Du(k)$$

Limitations: Linear functions; encode only a finite history

Recurrent Neural Networks

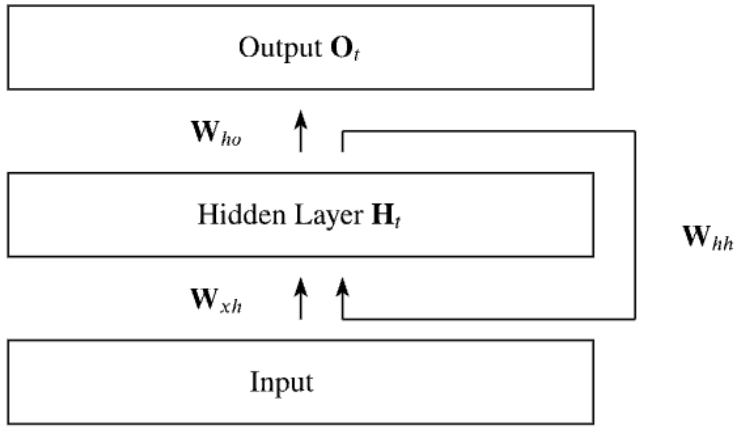
Auto-regressive Moving Average Equation:

$$y(k) = \sum_{i=1}^{p} a_i y(k-i) + \sum_{j=0}^{q} b_j u(k-j)$$

A non-linear version of auto-regressive moving average

where:

- $y(k)$: system output (or measured signal)
- $u(k)$ Input
- a_i: autoregressive coefficients
- b_j: moving-average coefficients



Recurrent Neural Network

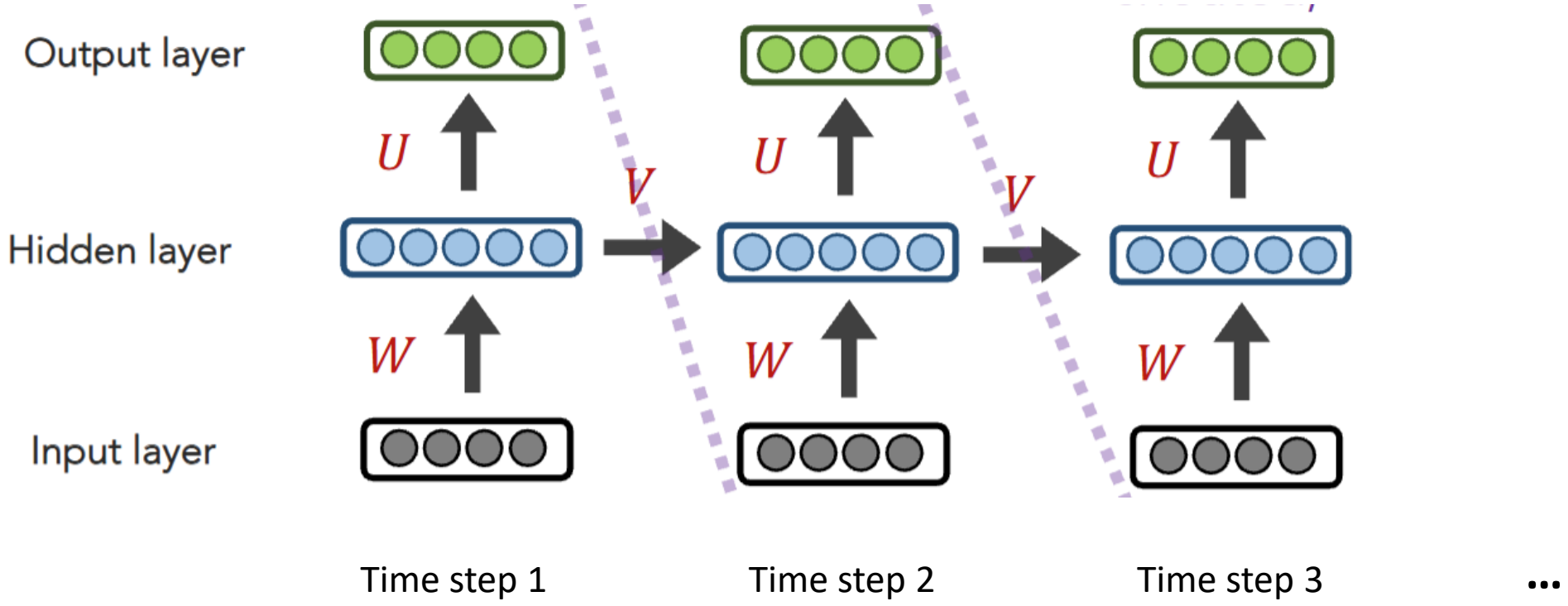
State-space representation:

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k) = Cx(k) + Du(k)$$

A non-linear version of auto-regressive moving average

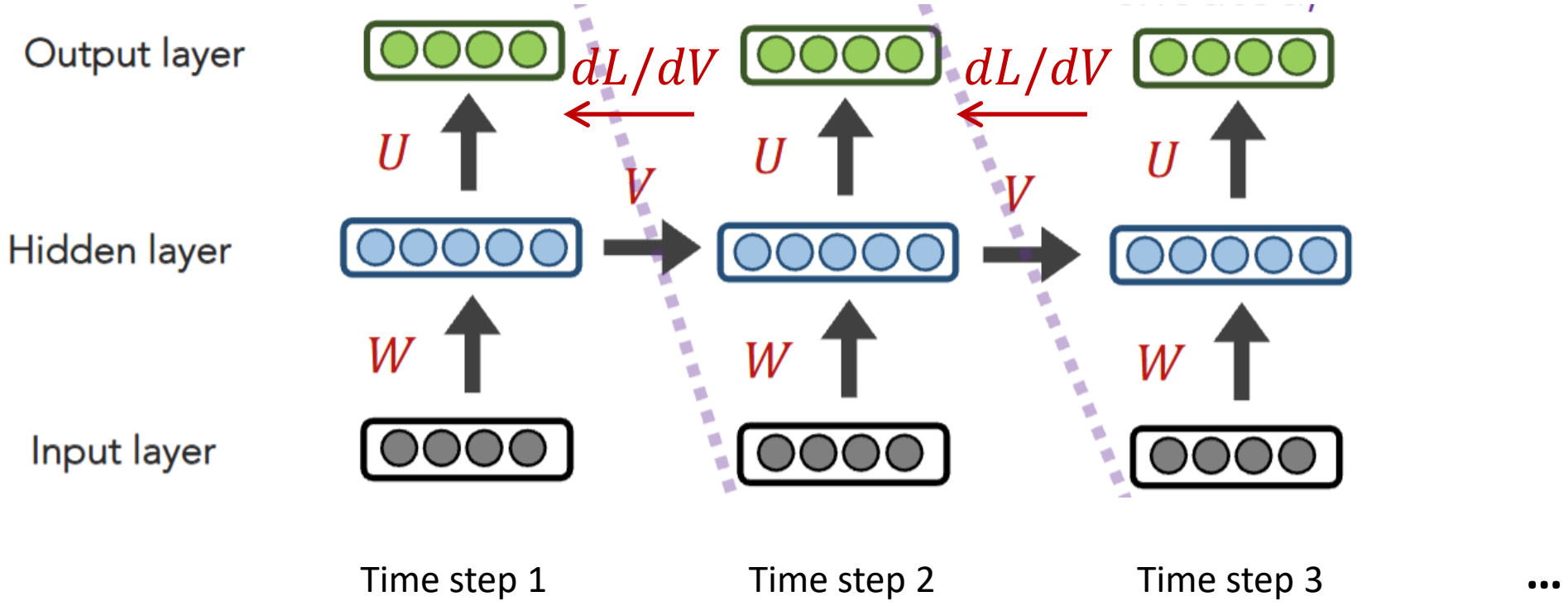
Vanishing/Exploding Gradient

Unrolling the RNN:



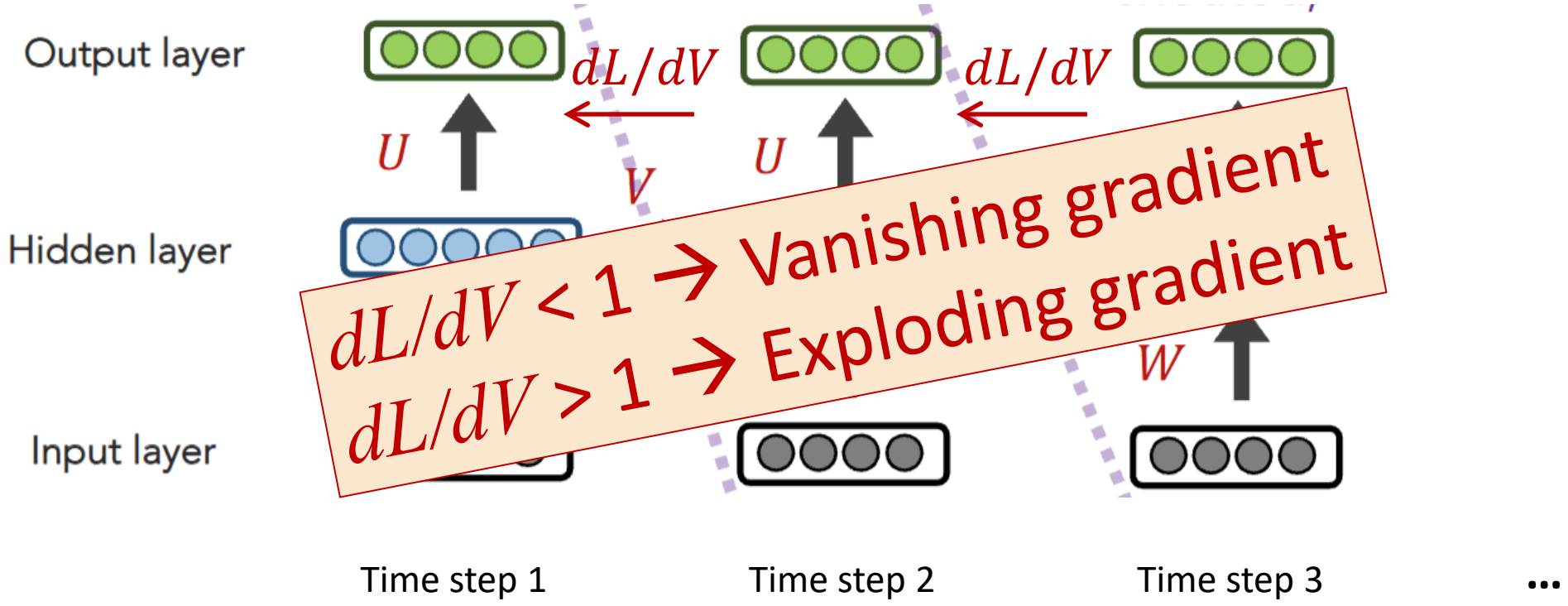
Vanishing/Exploding Gradient

Back Propagation (during training):



Vanishing/Exploding Gradient

Back Propagation (during training):



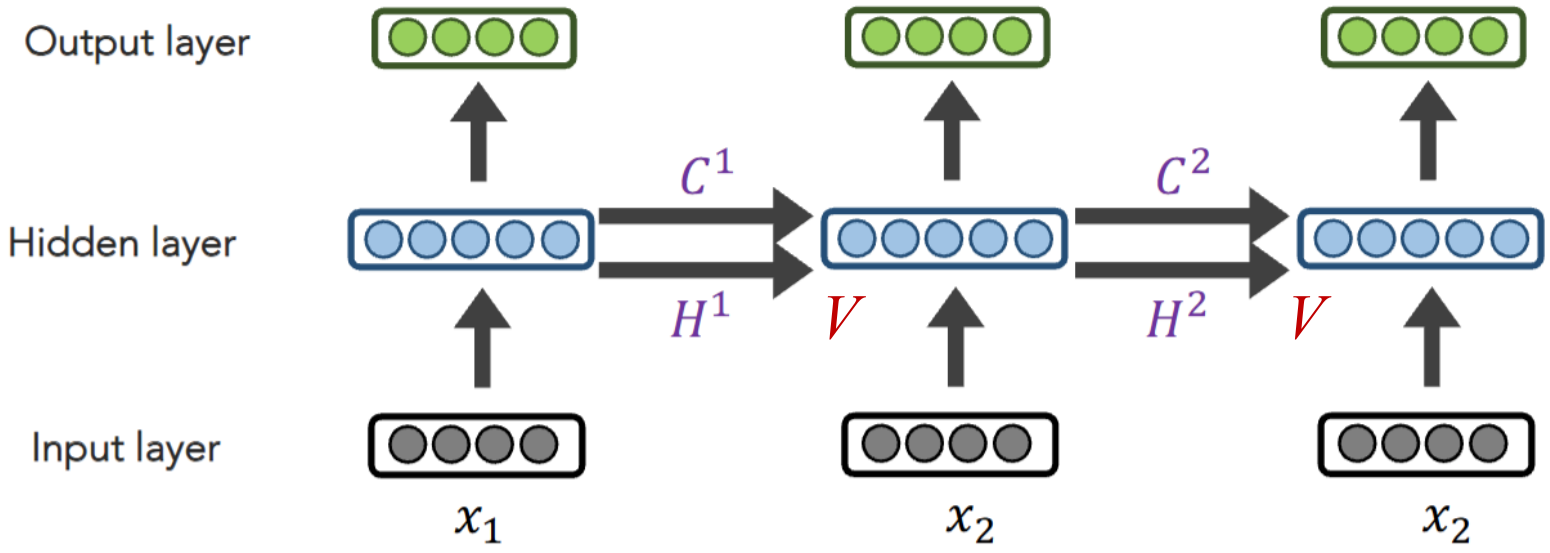
Vanishing/Exploding Gradient

A problem with RNNs: Cannot express long-range dependencies (i.e., those that exist between far apart time-steps) due to the vanishing/exploding gradient problem (too many time-steps in the middle)

Long Short-Term Memory (LSTM)

C: Long-term dependency

H: Short-term dependency

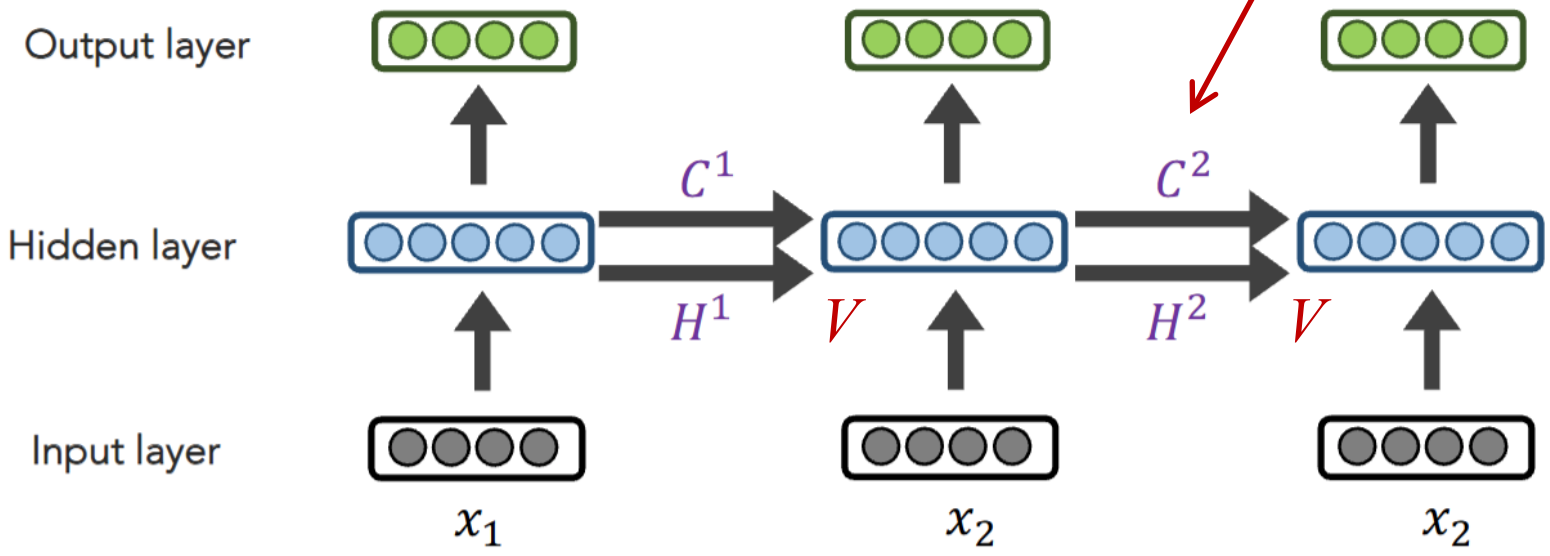


Long Short-Term Memory (LSTM)

C: Long-term dependency

H: Short-term dependency

Key idea: shortcut the “V” when necessary, thus preventing attenuation of old memories



Long Short-Term Memory (LSTM)

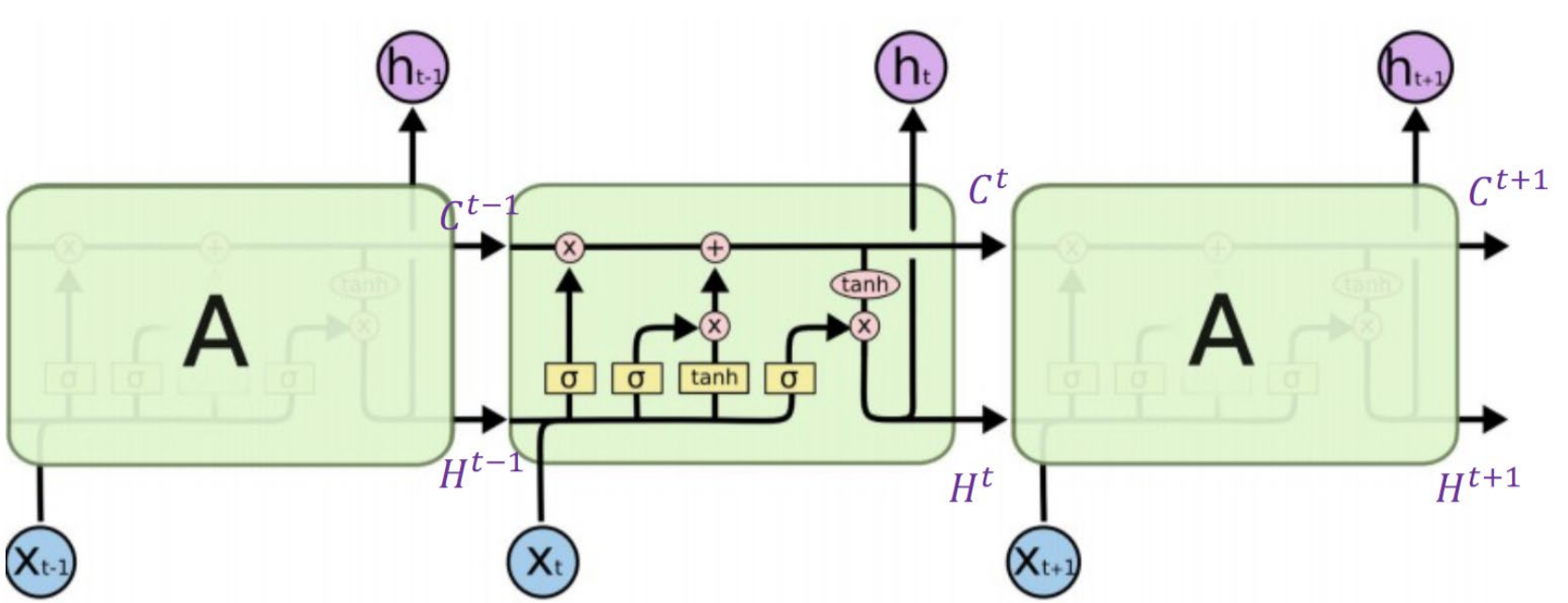
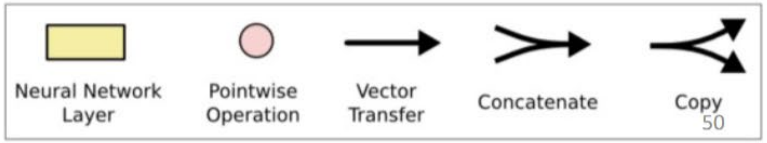


Diagram: https://colah.github.io/posts/2015-08-Understanding-LSTMs/

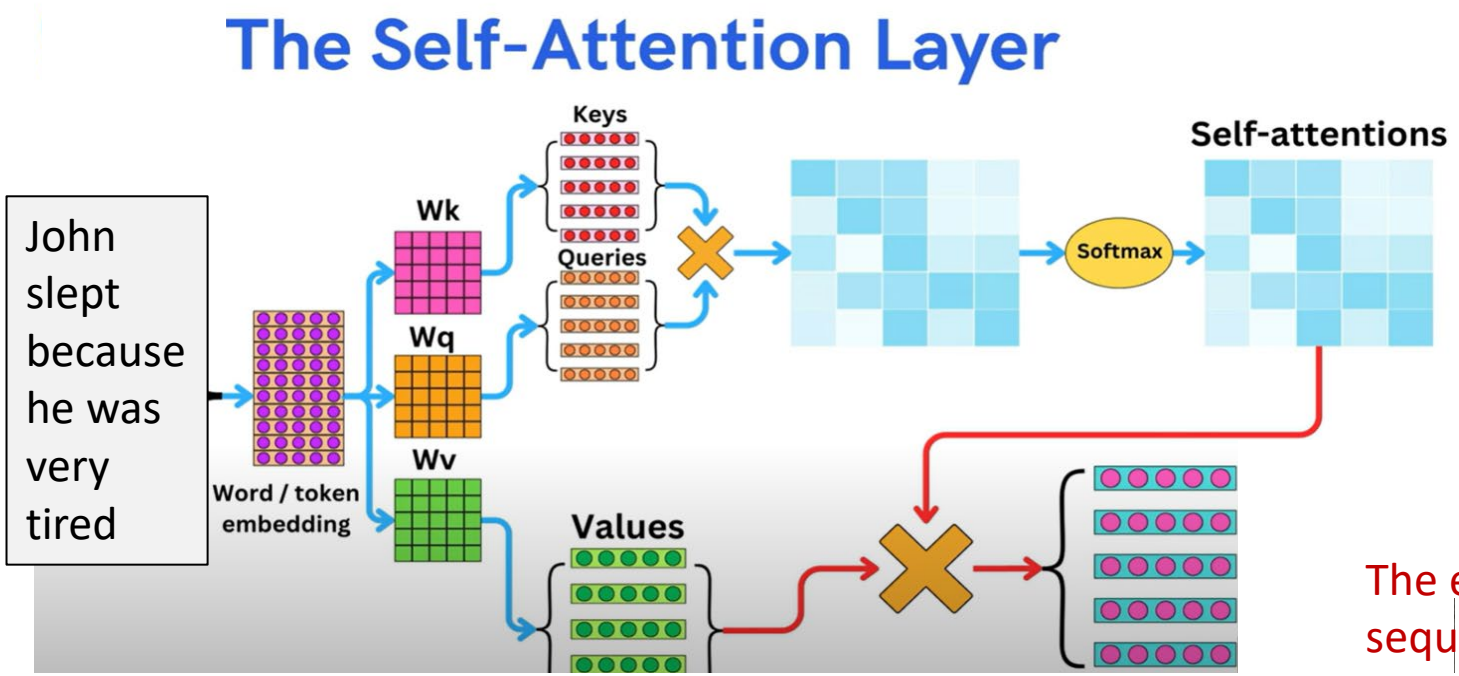


Long Short-Term Memory (LSTM)

Problem: With only two pathways (one for long- and one for short-term memory), the ability to retain a large number of individually important “old” memories is limited

Self-Attention and Transformers

- **Example: Transformers** (Attention is all you need):
  - https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf



The Ultimate non-linear "auto-regressive equation

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k) = Cx(k) + Du(k)$$

The embedding of each input token in the sequence is a function of the embedding of other input tokens in the sequence!

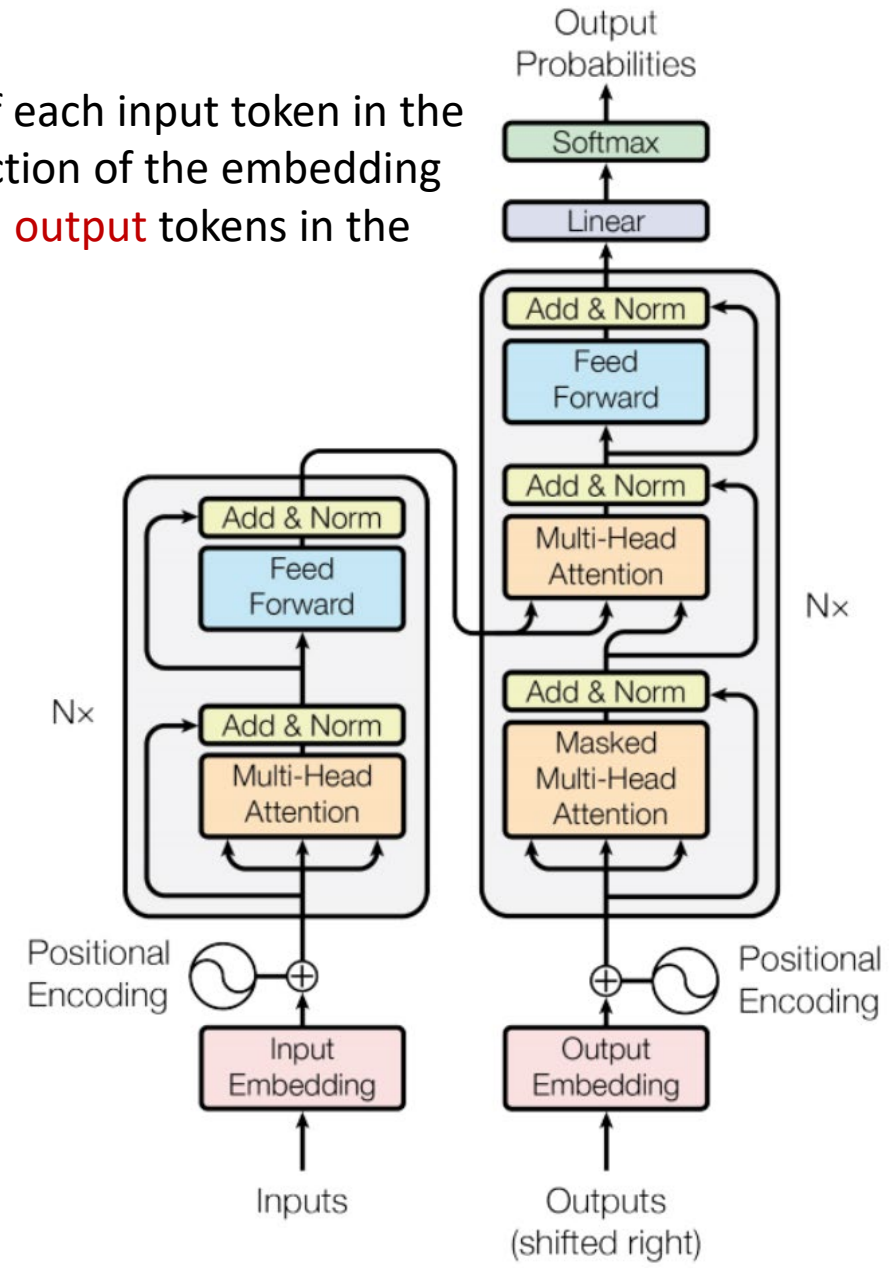
Transformers

The embedding of each input token in the sequence is a function of the embedding of other input and output tokens in the sequence!

The Ultimate non-linear "auto-regressive moving average" equation

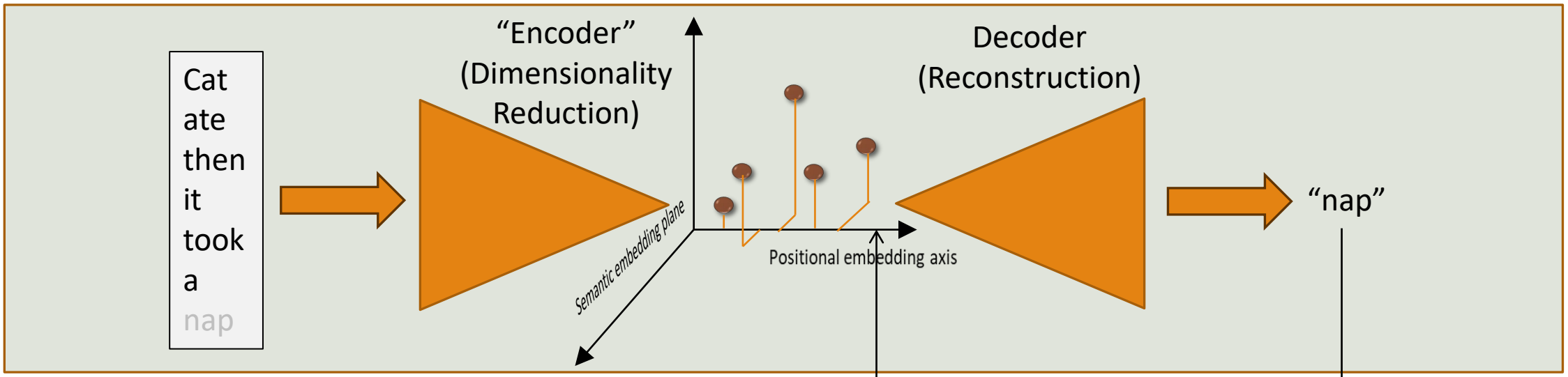
Generalizing the state equations:

$$x(k+1) = Ax(k) + Bu(k)$$
$$y(k) = Cx(k) + Du(k)$$



Transformers
Recall: Next Token Prediction

Cat ate then it took a nap



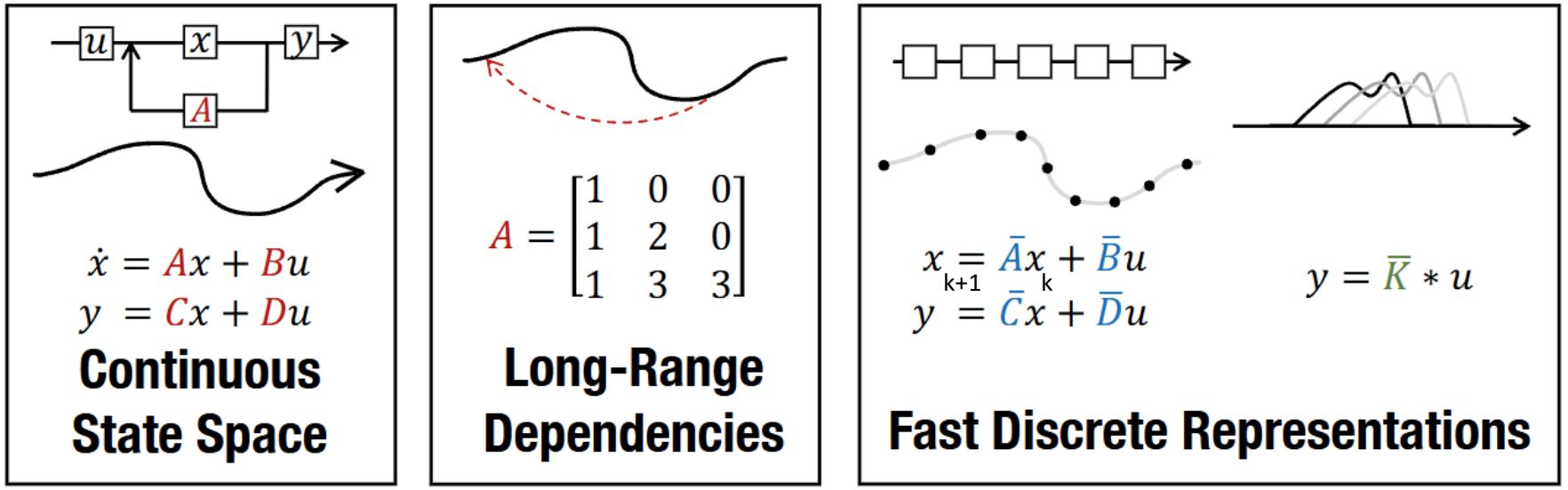
Observation

Autoregressive Next Token Generation

Transformers

Problem: Too computationally heavy (quadratic in the number of tokens)! Also, the dimensions expand as the number of input and output tokens increases.

Structured State Space Models

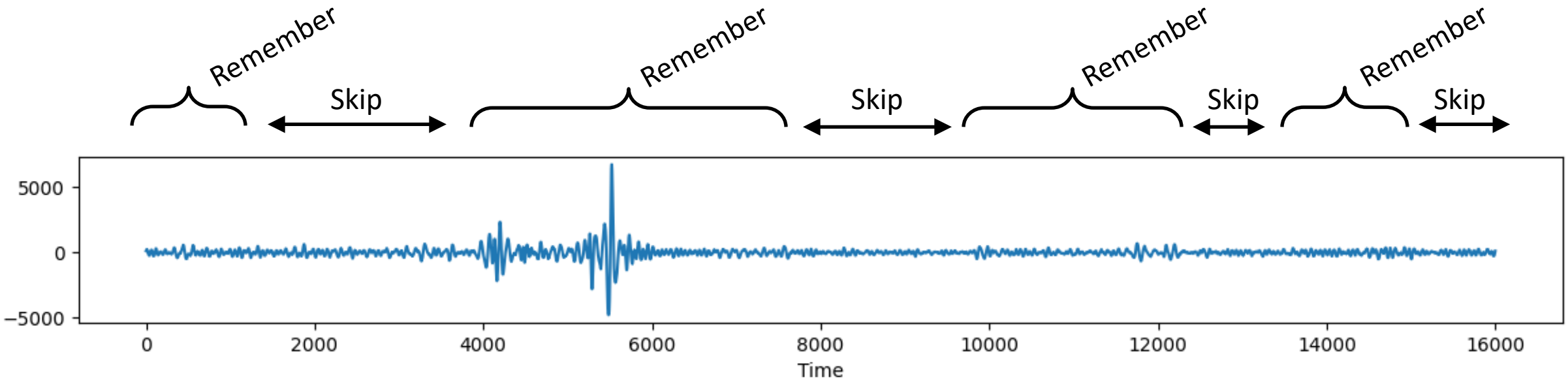


Structured State Space Models introduce ***a fixed-dimensional approximation*** of the discrete model representation that *remembers key long-range dependencies*

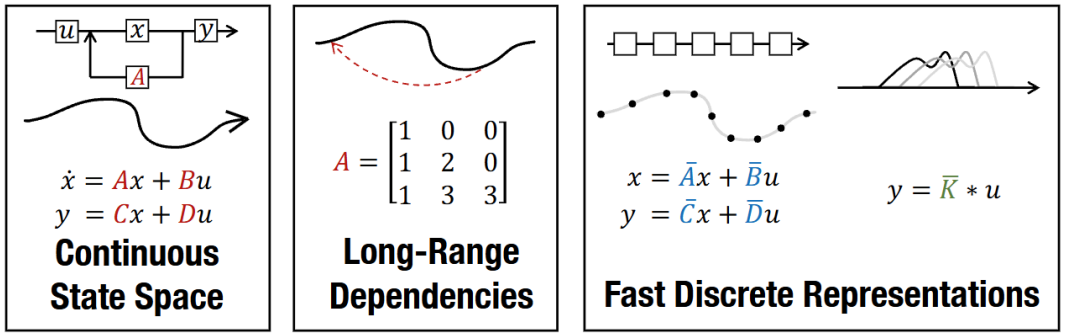
Structured State Space Models + Selection

Can we selectively ignore “noise” in between useful signals, not to burden the model with needing to remember useless inputs?

→ Must learn input-dependent “selection/skipping”



Mamba



Enhances prior structured state space models with an ability to skip/sub-select inputs

Algorithm 1 SSM (S4)

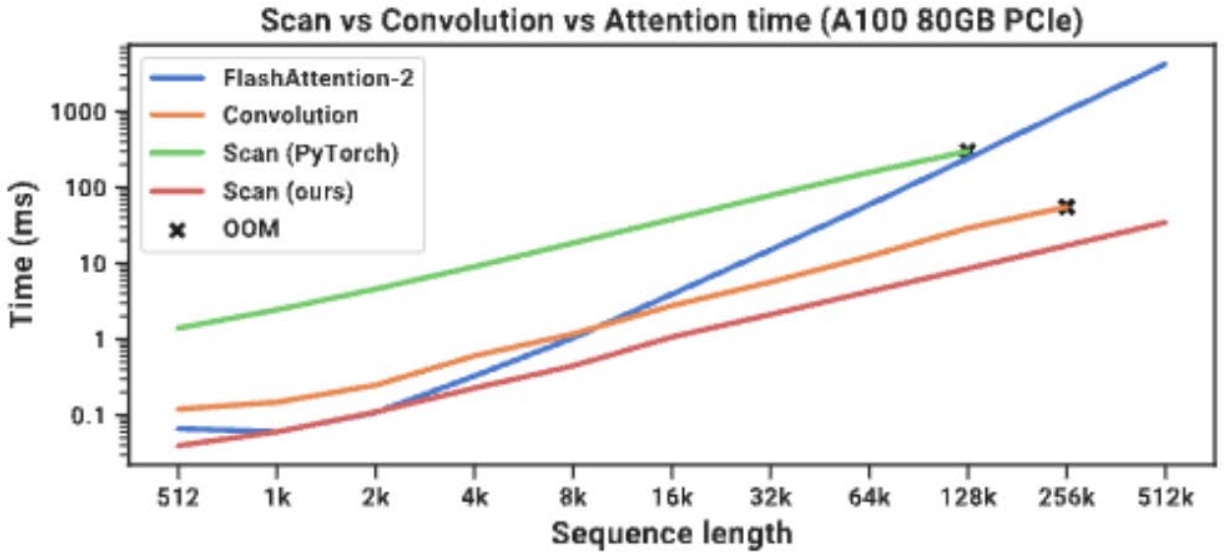
Input: $x : (\texttt{B},\texttt{L},\texttt{D})$
Output: $y : (\texttt{B},\texttt{L},\texttt{D})$
1: $A : (\texttt{D},\texttt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $B : (\texttt{D},\texttt{N}) \leftarrow$ Parameter
3: $C : (\texttt{D},\texttt{N}) \leftarrow$ Parameter
4: $\Delta : (\texttt{D}) \leftarrow \tau_\Delta(\text{Parameter})$
5: $\overline{A},\overline{B} : (\texttt{D},\texttt{N}) \leftarrow \text{discretize}(\Delta,A,B)$
6: $y \leftarrow \text{SSM}(\overline{A},\overline{B},C)(x)$
▷ Time-invariant: recurrence or convolution
7: **return** y

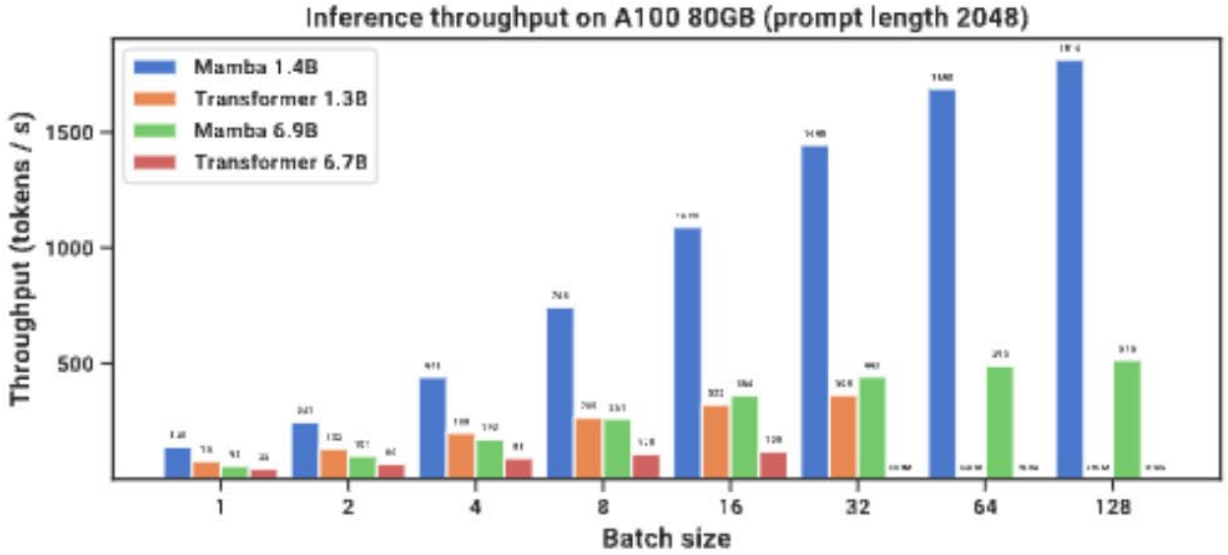
Algorithm 2 SSM + Selection (S6)

Input: $x : (\texttt{B},\texttt{L},\texttt{D})$
Output: $y : (\texttt{B},\texttt{L},\texttt{D})$
1: $A : (\texttt{D},\texttt{N}) \leftarrow$ Parameter
▷ Represents structured $N \times N$ matrix
2: $B : (\texttt{B},\texttt{L},\texttt{N}) \leftarrow s_B(x)$
3: $C : (\texttt{B},\texttt{L},\texttt{N}) \leftarrow s_C(x)$
4: $\Delta : (\texttt{B},\texttt{L},\texttt{D}) \leftarrow \tau_\Delta(\text{Parameter} + s_\Delta(x))$
5: $\overline{A},\overline{B} : (\texttt{B},\texttt{L},\texttt{D},\texttt{N}) \leftarrow \text{discretize}(\Delta,A,B)$
6: $y \leftarrow \text{SSM}(\overline{A},\overline{B},C)(x)$
▷ Time-varying: recurrence (*scan*) only
7: **return** y

Mamba – Efficiency

More efficient than transformers





Mamba – Inference Quality

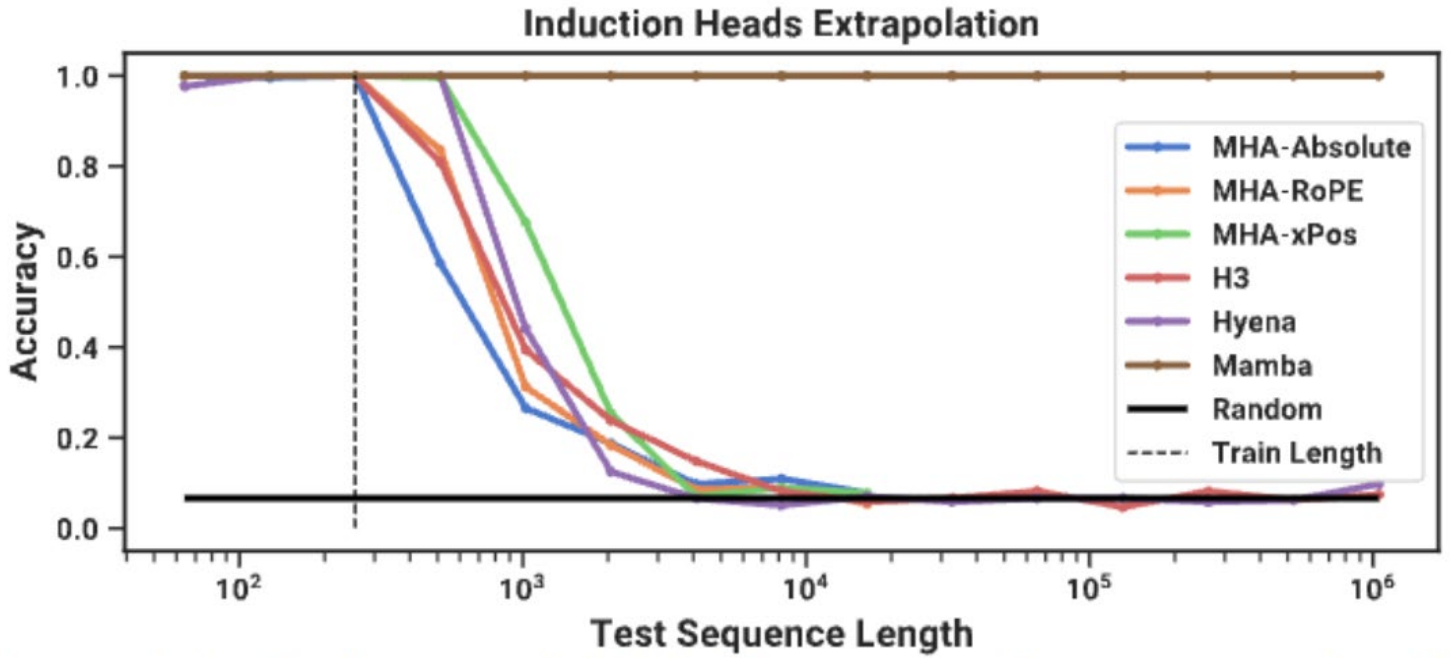


Figure: **(Induction Heads.)** Models are trained on sequence length $2^8 = 256$, and tested on increasing sequence lengths of $2^6 = 64$ up to $2^{20} = 1048576$.

Mamba – Perplexity (a Metric of Model “Confusion”)

Mamba produced lower-perplexity models than prior state of the art architectures