CENG501 – Deep Learning

Week 6

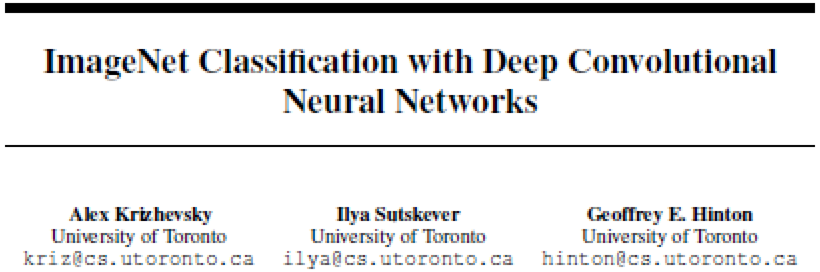
Fall 2024

Sinan Kalkan

Dept. of Computer Engineering, METU

Previously on CENG501

AlexNet (2012)

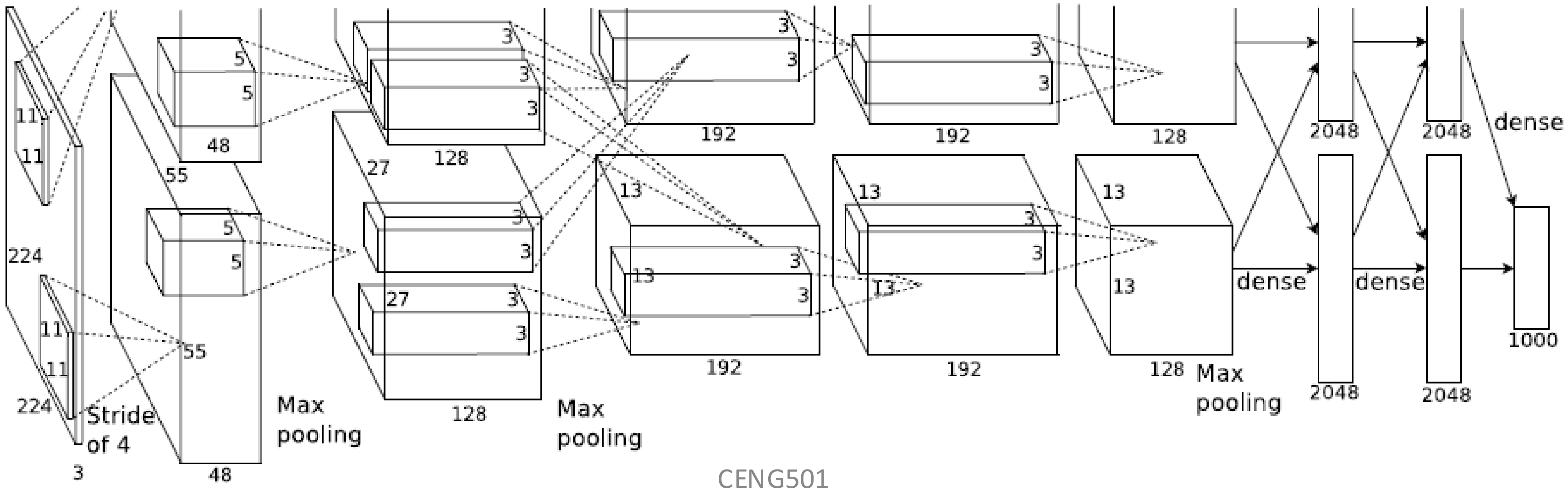

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

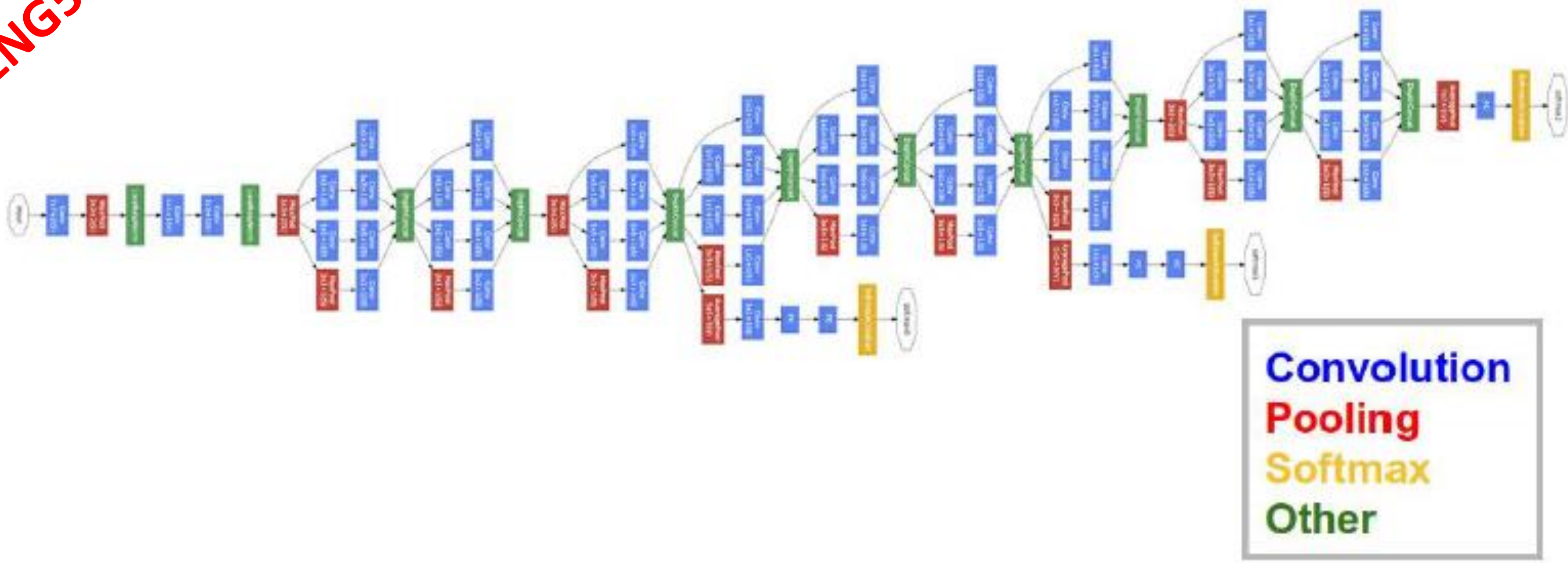
Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

- Popularized CNN in computer vision & pattern recognition
- ImageNet ILSVRC challenge 2012 winner
- Similar to LeNet
  - Deeper & bigger
  - Many CONV layers on top of each other (rather than adding immediately a pooling layer after a CONV layer)
  - Uses GPU
- 650K neurons. 60M parameters. Trained on 2 GPUs for a week.



Previously on CENG501



One of the main beneficial aspects of this architecture is that it allows for increasing the number of units at each stage significantly without an uncontrolled blow-up in computational complexity. The ubiquitous use of dimension reduction allows for shielding the large number of input filters of the last stage to the next layer, first reducing their dimension before convolving over them with a large patch size. Another practically useful aspect of this design is that it aligns with the intuition that visual information should be processed at various scales and then aggregated so that the next stage can abstract features from different scales simultaneously.

Previously on CENG501

Table 1: **ConvNet configurations** (shown in columns). The depth of the configurations increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold). The convolutional layer parameters are denoted as "conv⟨receptive field size⟩-⟨number of channels⟩". The ReLU activation function is not shown for brevity.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Previously on CENG501

ResNet (2015)

- Residual (shortcut) connections

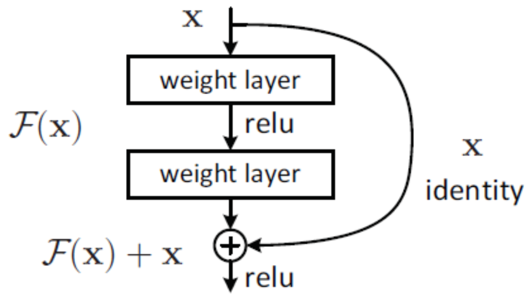


Figure 2. Residual learning: a building block.

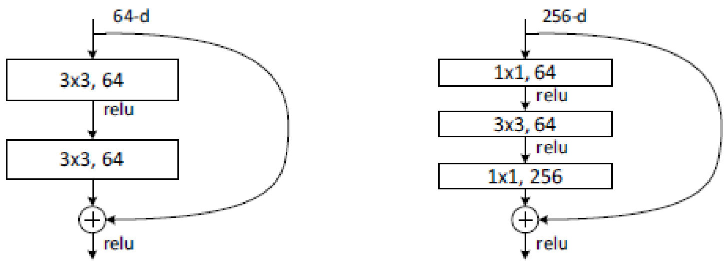
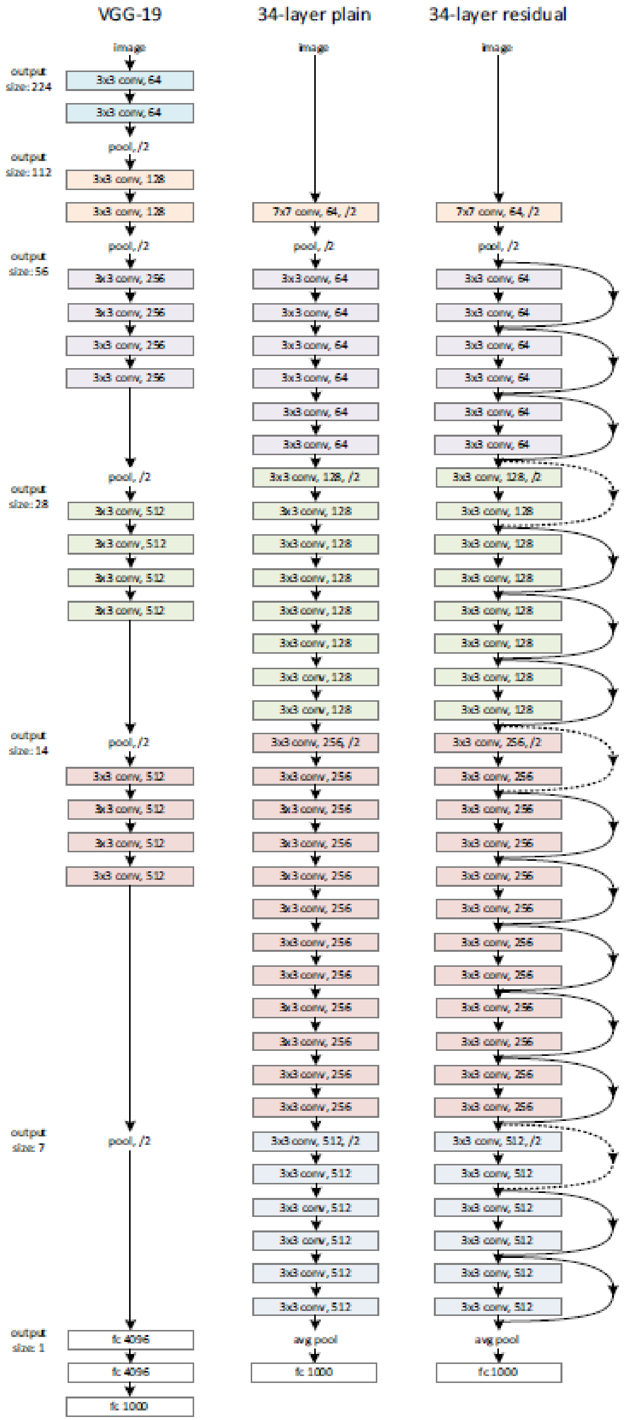
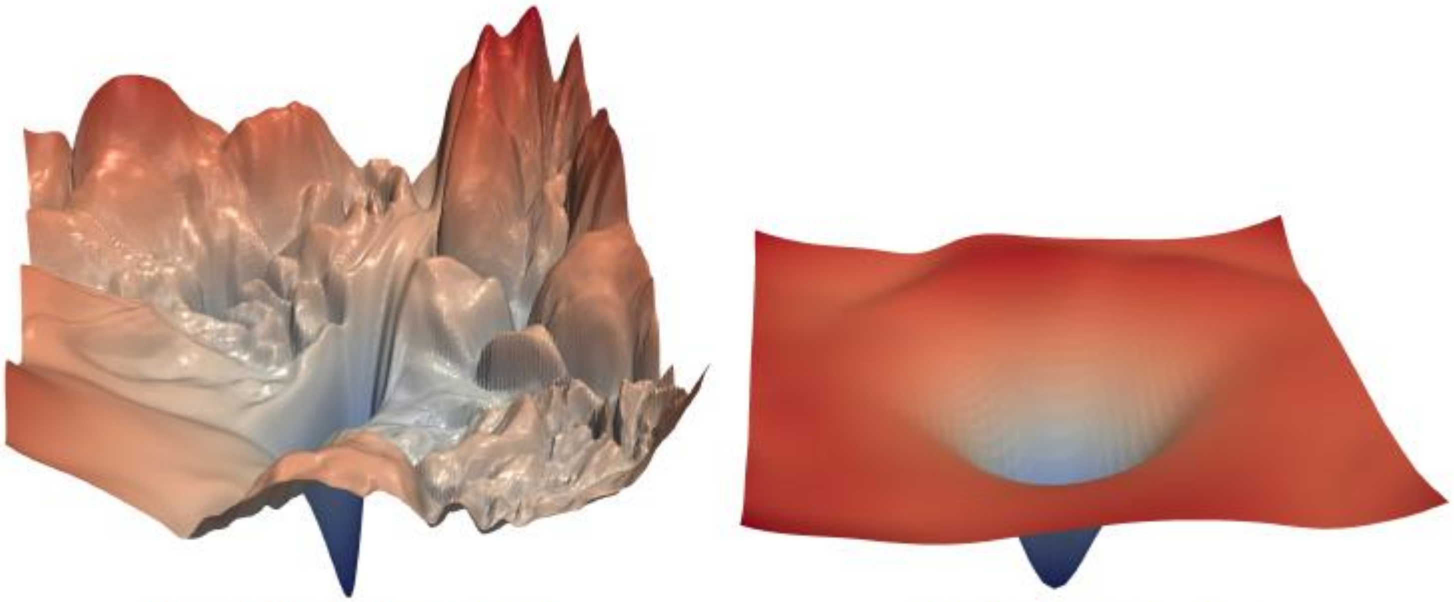


Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.



Previously on CENG501

Effect of residual connections



(a) without skip connections (b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The vertical axis is logarithmic to show dynamic range. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.

VISUALIZING THE LOSS LANDSCAPE OF NEURAL NETS

2018

Hao Li[1], Zheng Xu[1], Gavin Taylor[2], Christoph Studer[3], Tom Goldstein[1]
[1]University of Maryland, College Park, [2]United States Naval Academy, [3]Cornell University
{haoli,xuzh,tomg}@cs.umd.edu, taylor@usna.edu, studer@cornell.edu

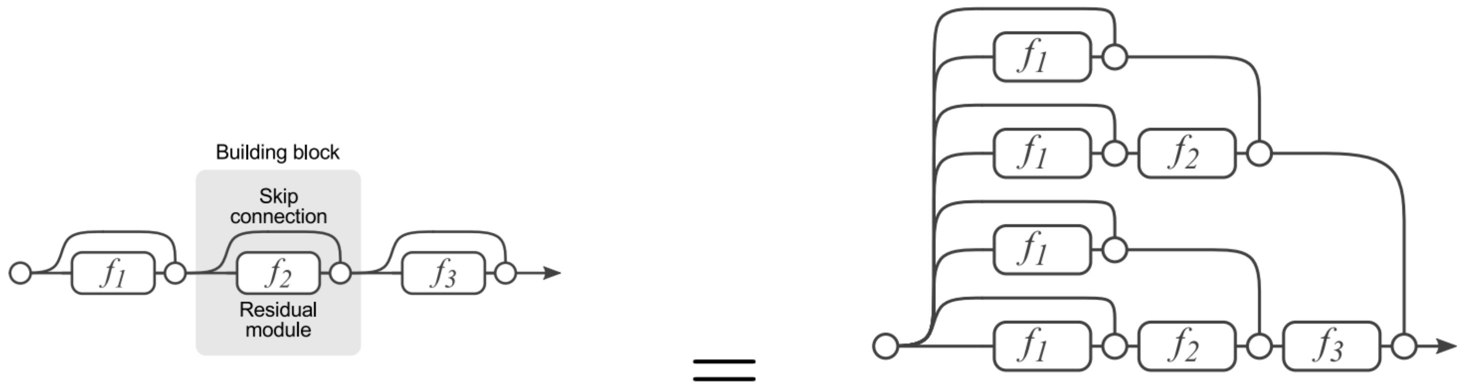
ResNet: Ensemble of Shallow Networks

Previously on CENG501

Residual Networks Behave Like Ensembles of Relatively Shallow Networks

Andreas Veit Michael Wilber Serge Belongie
Department of Computer Science & Cornell Tech
Cornell University
{av443, mjw285, sjb344}@cornell.edu

2016



(a) Conventional 3-block residual network (b) Unraveled view of (a)

Figure 1: Residual Networks are conventionally shown as (a), which is a natural representation of Equation (1). When we expand this formulation to Equation (6), we obtain an *unraveled view* of a 3-block residual network (b). Circular nodes represent additions. From this view, it is apparent that residual networks have $O(2^n)$ implicit paths connecting input and output and that adding a block doubles the number of paths.

ResNext

Previously on CENG501

Aggregated Residual Transformations for Deep Neural Networks

Saining Xie[1] Ross Girshick[2] Piotr Dollár[2] Zhuowen Tu[1] Kaiming He[2]
[1]UC San Diego [2]Facebook AI Research
{s9xie,ztu}@ucsd.edu {rbg,pdollar,kaiminghe}@fb.com

2017

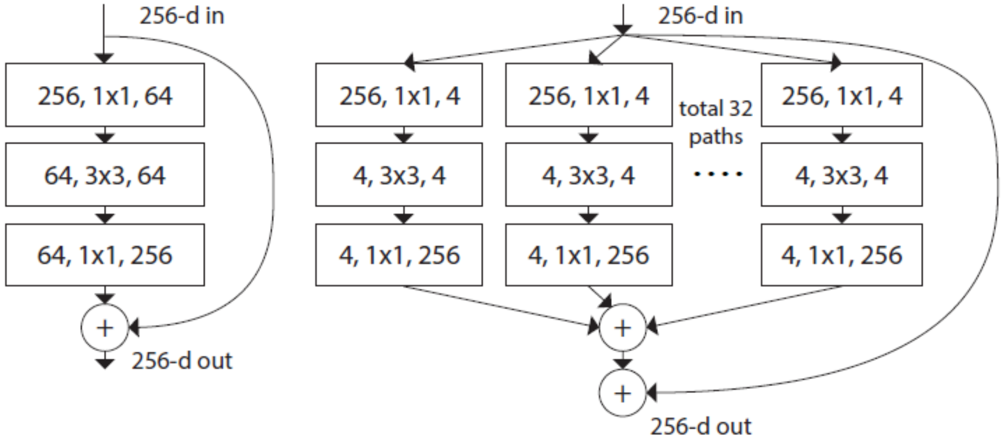
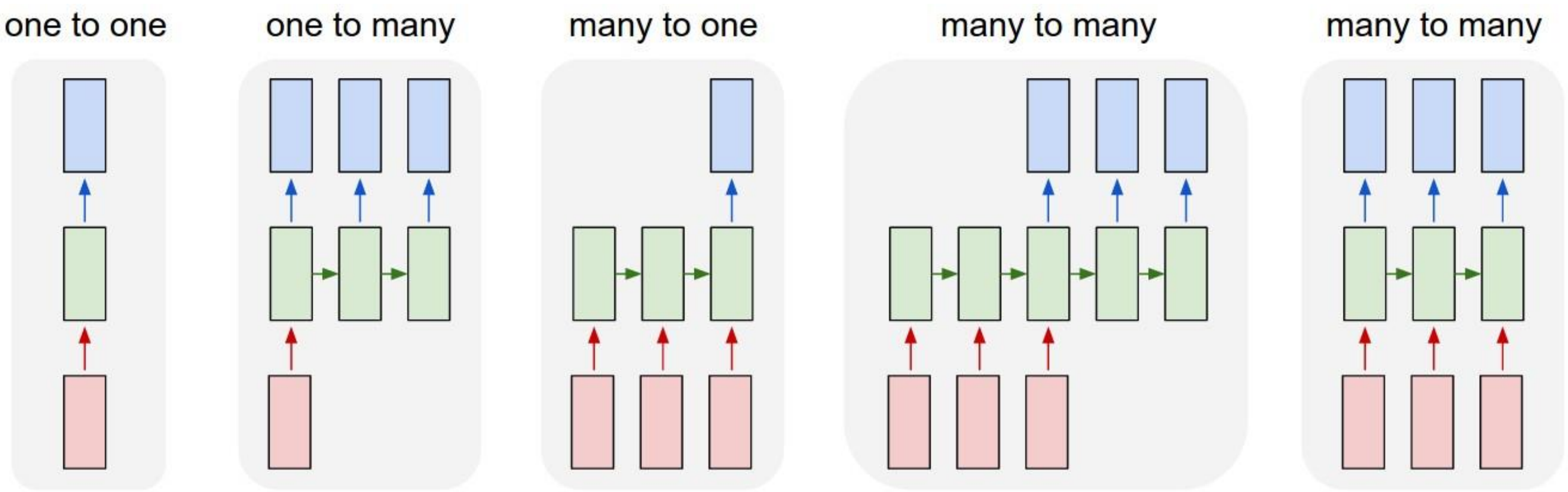


Figure 1. **Left**: A block of ResNet [14]. **Right**: A block of ResNeXt with cardinality = 32, with roughly the same complexity. A layer is shown as (# in channels, filter size, # out channels).

	setting	top-1 err (%)	top-5 err (%)
1× complexity references:			
ResNet-101	1 × 64d	22.0	6.0
ResNeXt-101	32 × 4d	21.2	5.6
2× complexity models follow:			
ResNet-200 [15]	1 × 64d	21.7	5.8
ResNet-101, wider	1 × **100**d	21.3	5.7
ResNeXt-101	**2** × 64d	20.7	5.5
ResNeXt-101	**64** × 4d	**20.4**	**5.3**

Different types of sequence learning / recognition problems

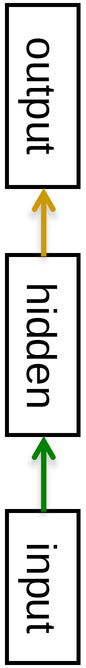
Previously on CENG501



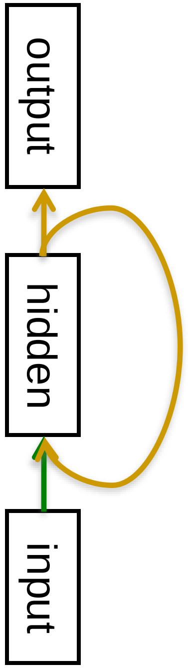
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Previously on CENG501

Recurrent Neural Networks (RNNs)



Feed-forward networks



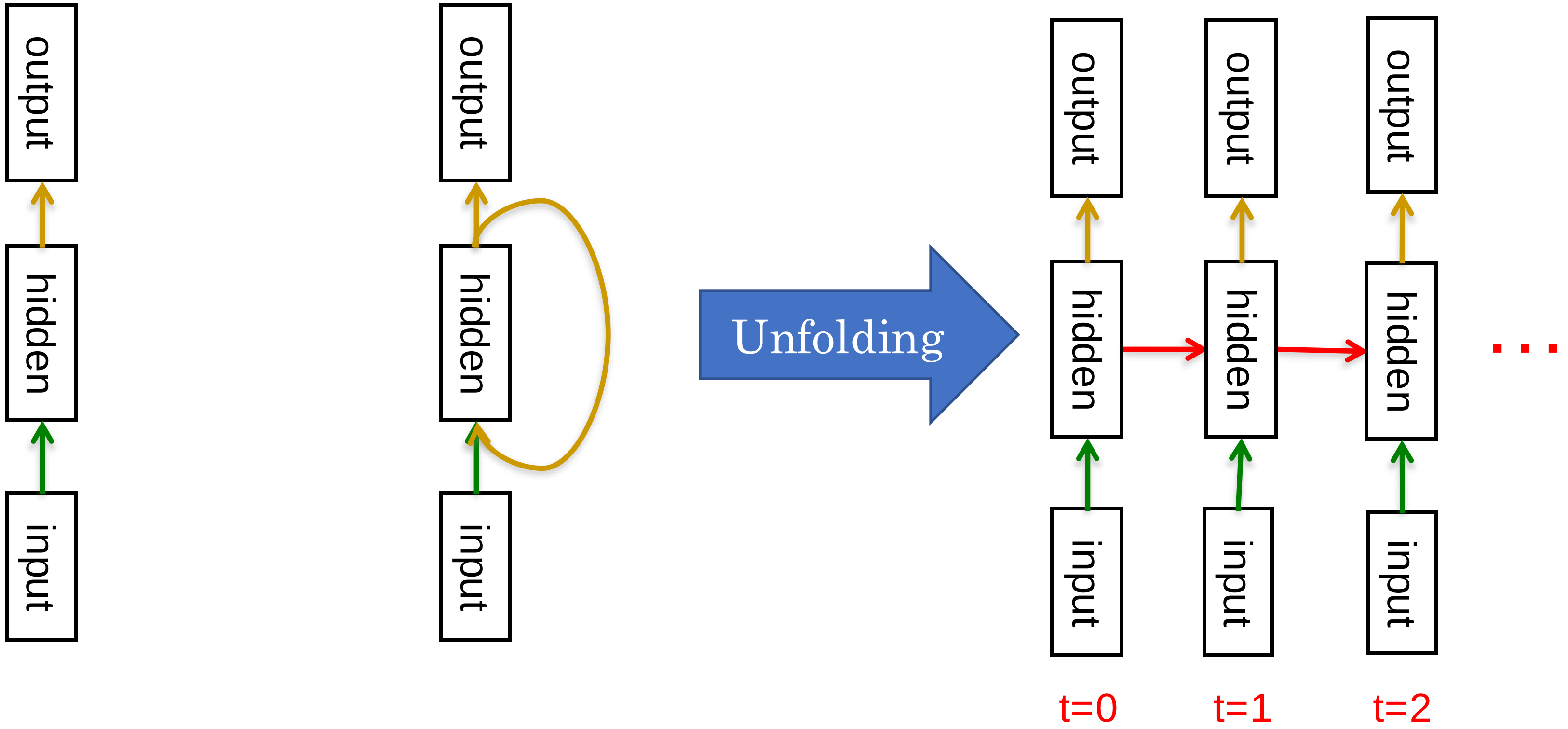
Recurrent networks

- RNNs are very powerful because:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.
- More formally, RNNs are Turing complete.

Adapted from Hinton

Previously on CENG501

Unfolding



Feed-forward networks

Recurrent networks

time →

Feedforward through Vanilla RNN

Previously on CENG501

The Vanilla RNN Model

First time-step ($t = 1$):

$$\mathbf{h}_1 = tanh\left(W^{xh} \cdot \mathbf{x}_1 + W^{hh} \cdot \mathbf{h}_0\right)$$

$$\hat{\mathbf{y}}_1 = softmax\left(W^{hy} \cdot \mathbf{h}_1\right)$$

$$\mathcal{L}_1 = CE(\hat{\mathbf{y}}_1, \mathbf{y}_1)$$

In general:

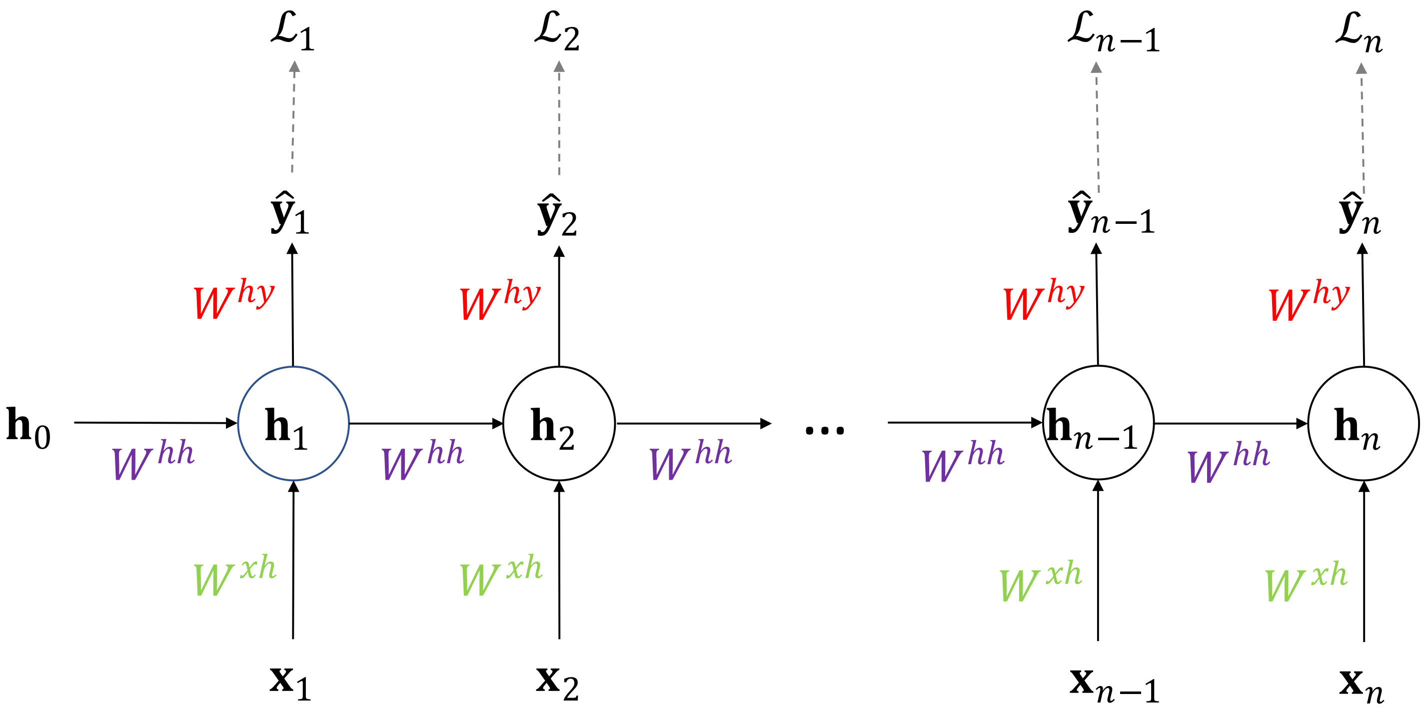
$$\mathbf{h}_t = tanh\left(W^{xh} \cdot \mathbf{x}_t + W^{hh} \cdot \mathbf{h}_{t-1}\right)$$

$$\hat{\mathbf{y}}_t = softmax\left(W^{hy} \cdot \mathbf{h}_t\right)$$

$$\mathcal{L}_t = CE(\hat{\mathbf{y}}_t, \mathbf{y}_t)$$

In total:

$$\mathcal{L} = \sum_t \mathcal{L}_t$$



Previously on CENG501

Key Problem

- Learning long-term dependencies is hard

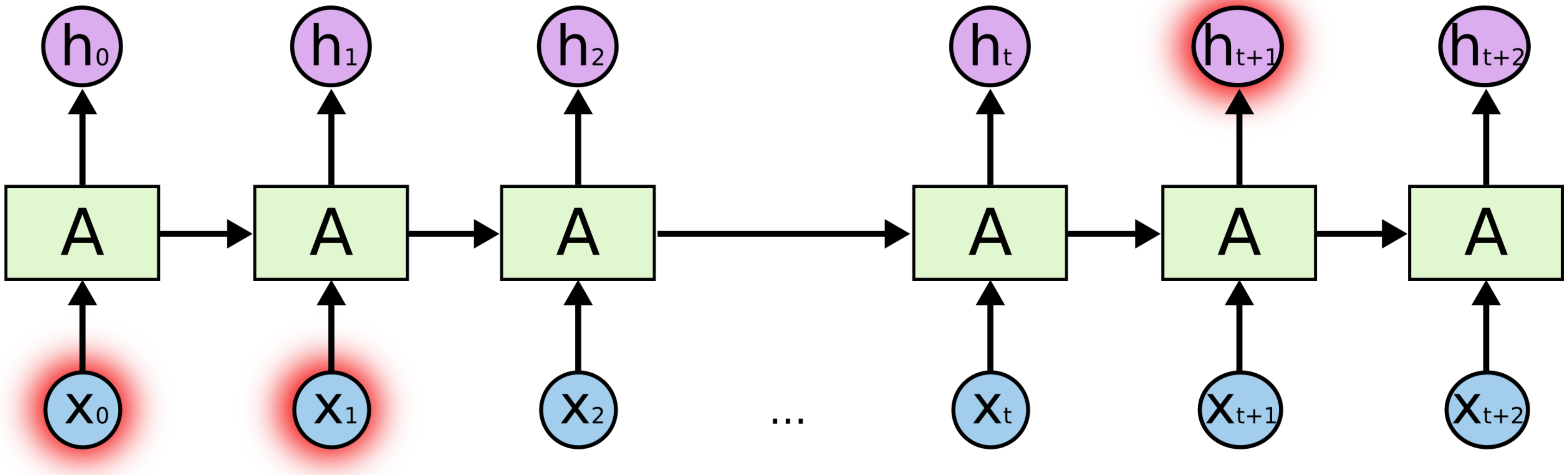


Image Credit: Christopher Olah (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

Previously on CENG501

LSTM in detail

- We first compute an activation vector, a:

$$a = W_x x_t + W_h h_{t-1} + b$$

- Split this into four vectors of the same size:

$$a_i, a_f, a_o, a_g \leftarrow a$$

- We then compute the values of the gates:

$$i = \sigma(a_i) \quad f = \sigma(a_f) \quad o = \sigma(a_o) \quad g = \tanh(a_g)$$

  where σ is the sigmoid.

- The next cell state c_t and the hidden state h_t:

$$c_t = f \odot c_{t-1} + i \odot g$$
$$h_t = o \odot \tanh(c_t)$$

  where $\odot$ is the element-wise product of vectors

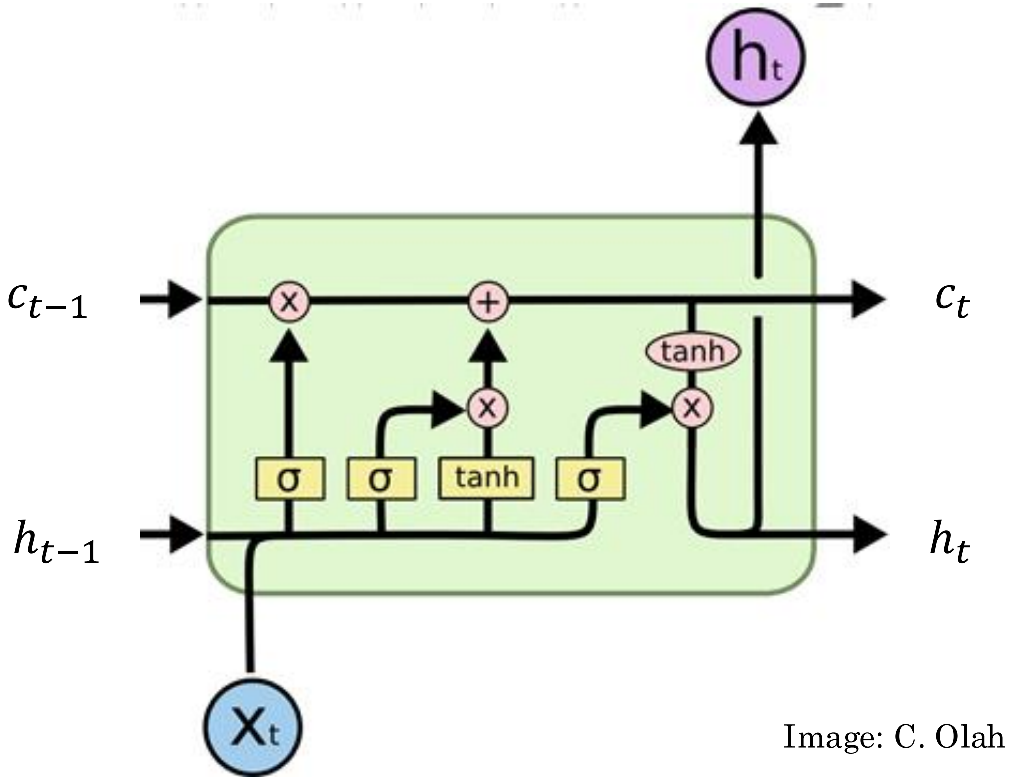


Image: C. Olah

Alternative formulation:

$$i_t = g(W_{xi}x_t + W_{hi}h_{t-1} + b_i)$$
$$f_t = g(W_{xf}x_t + W_{hf}h_{t-1} + b_f)$$
$$o_t = g(W_{xo}x_t + W_{ho}h_{t-1} + b_o)$$

Eqs: Karpathy

Character-level Text Modeling

Previously on CENG501

- Problem definition: Find c_{n+1} given $c_1, c_2, \ldots, c_n$.

- Modelling:

$$p(c_{n+1} \mid c_n, \ldots, c_1)$$

- In general, we just take the last N characters:

$$p(c_{n+1} \mid c_n, \ldots, c_{n-(N-1)})$$

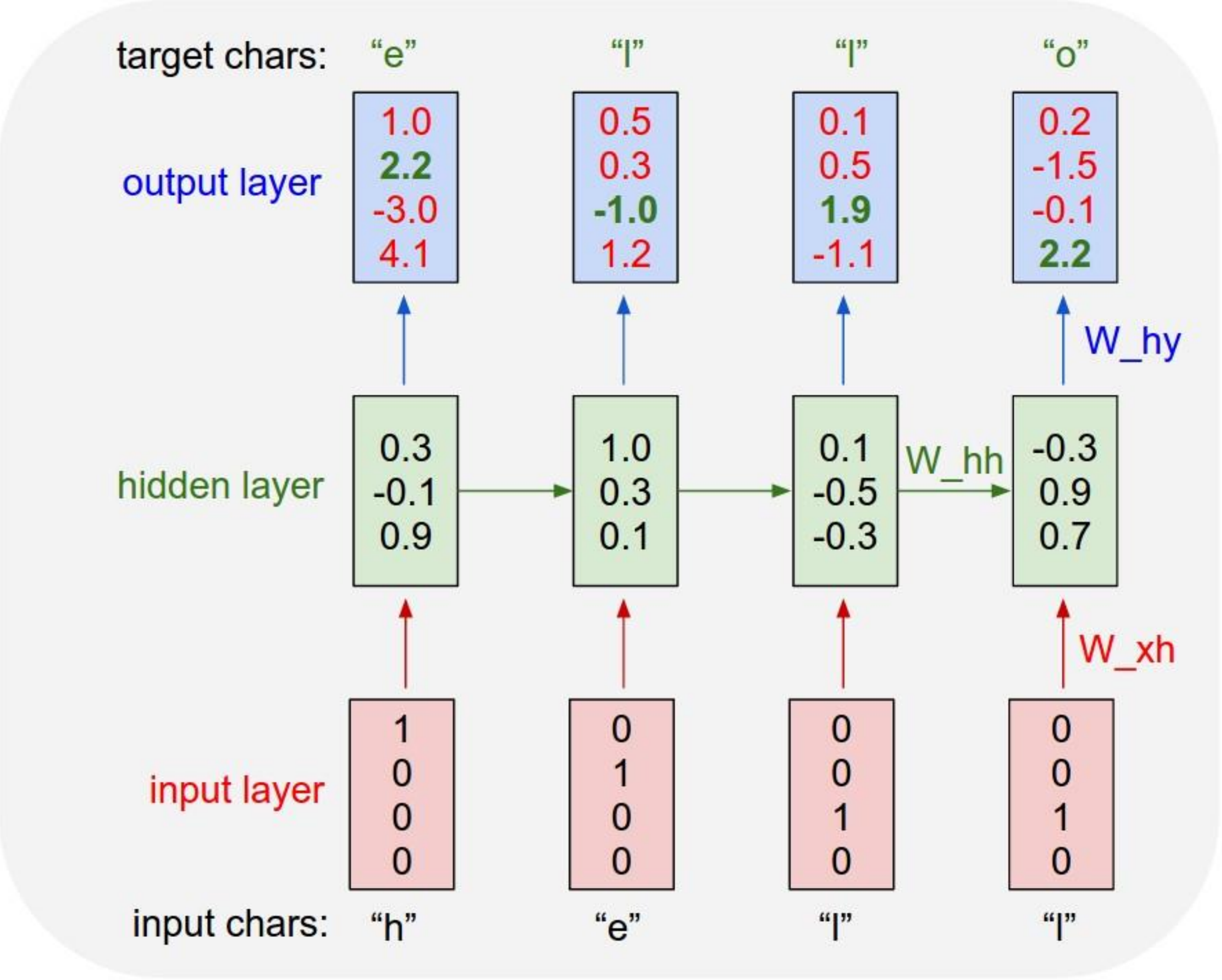
- Learn $p(c_{n+1} = 'a' \mid 'Ankar')$ from data such that

$$p(c_{n+1} = 'a' \mid 'Ankar') > p(c_{n+1} = 'o' \mid 'Ankar')$$

A simple scenario

Previously on CENG501

- Alphabet: h, e, l, o
- Text to train to predict: "hello"



http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Word-level Text Modeling

Previously on CENG501

- Problem definition: Find ω_{n+1} given $\omega_1, \omega_2, \ldots, \omega_n$.
- Modelling:

$$p(\omega_{n+1} \mid \omega_n, \ldots, \omega_1)$$

- In general, we just take the last N words:

$$p(\omega_{n+1} \mid \omega_n, \ldots, \omega_{n-(N-1)})$$

- Learn $p(\omega_{n+1} = 'Turkey' \mid 'Ankara\ is\ the\ capital\ of\ ')$ from data such that:

$$p(\omega_{n+1} = 'Turkey' \mid 'Ankara\ is\ the\ capital\ of\ ') \quad > \quad p(\omega_{n+1} = 'UK' \mid 'Ankara\ is\ the\ capital\ of\ ')$$

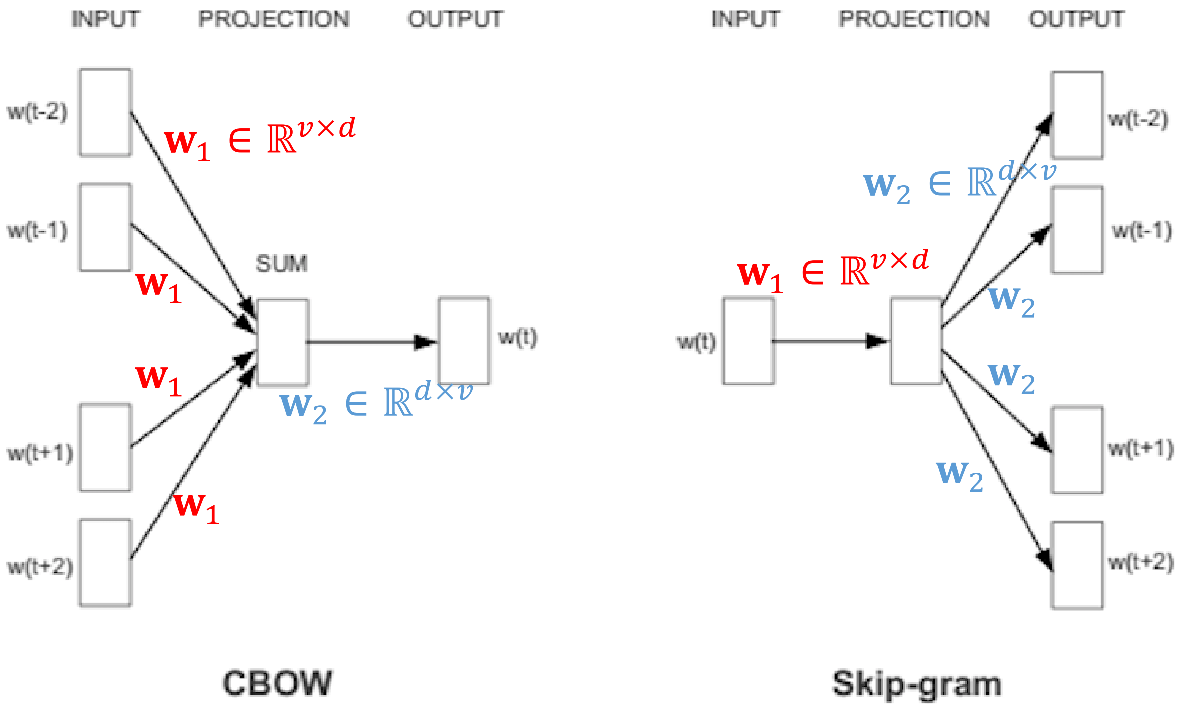
Previously on CENG501

Two different ways to train

1. Using context to predict a target word (~ continuous bag-of-words)
2. Using word to predict a target context (skip-gram)

- If the vector for a word cannot predict the context, the mapping to the vector space is adjusted
- Since similar words should predict the same or similar contexts, their vector representations should end up being similar

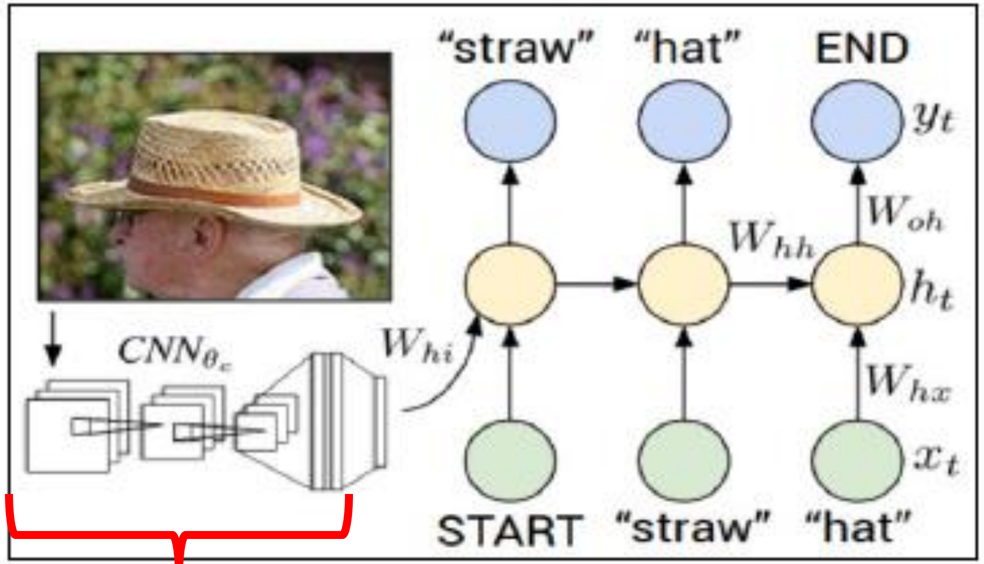
v: vocabulary size
d: hidden dimension



http://deeplearning4j.org/word2vec

Overview

Previously on CENG501



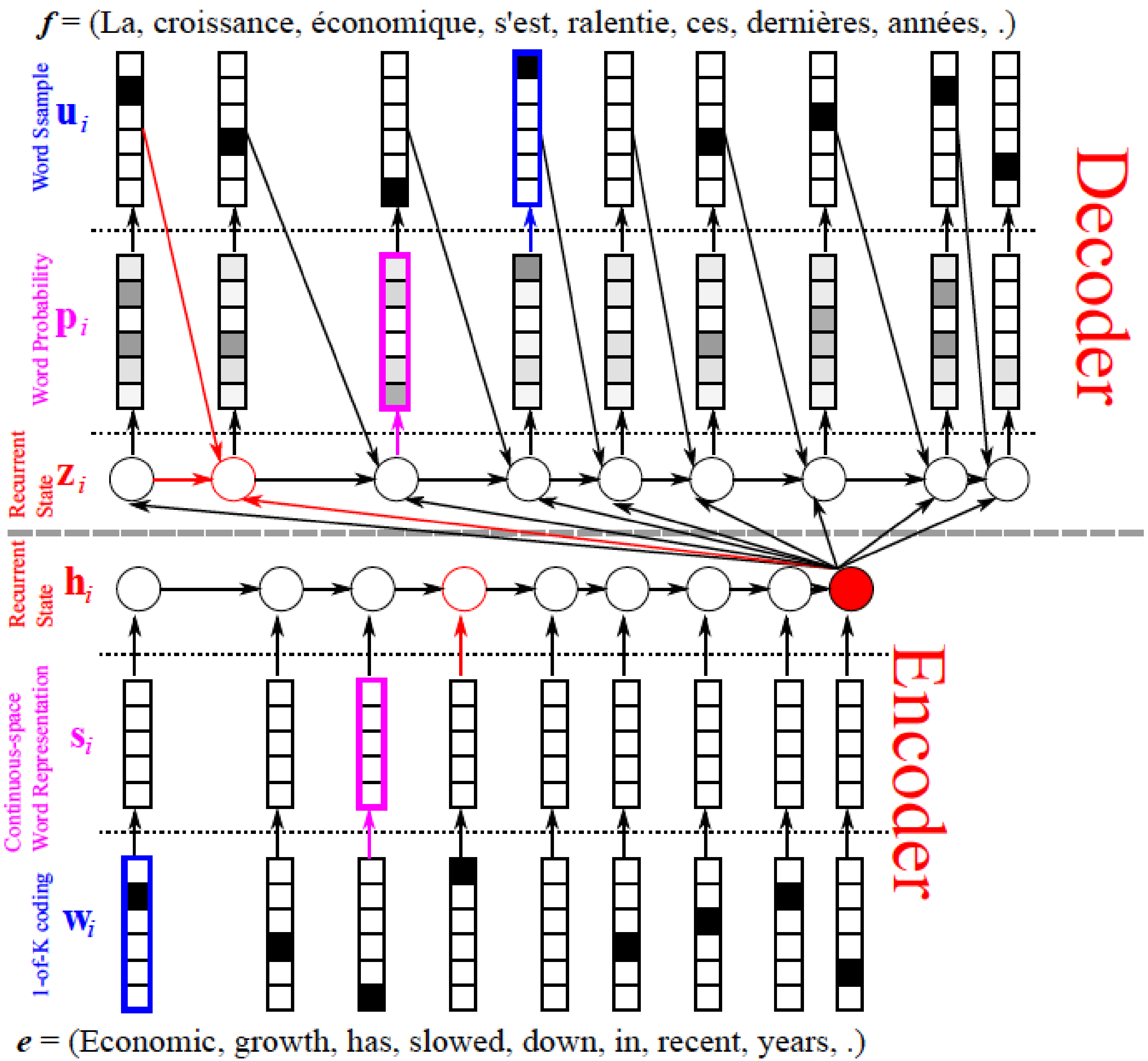
Pre-trained word embedding is also used

Pre-trained CNN
(e.g., on imagenet)

Image: Karpathy

Neural Machine Translation

Previously on CENG501



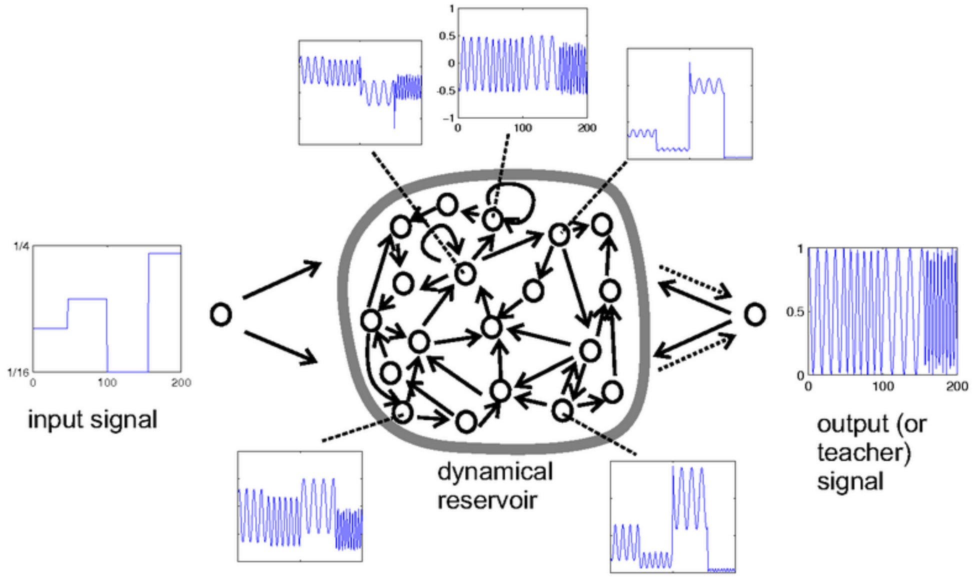
Cho: From Sequence Modeling to Translation

Today

- Echo State Networks
- Attention
- Self-attention
- Transformer
- Linear attention
- State-Space Models
- Mamba

Administrative Notes

- Paper Selection Finalized
- Time plan for the projects
  1. Milestone (November 24, midnight):
     - Github repo will be ready
     - Read & understand the paper
     - Download the datasets
     - Prepare the Readme file excluding the results & conclusion
  2. Milestone (December 8, midnight)
     - The results of the first experiment
  3. Milestone (January 5, midnight)
     - Final report (Readme file)
     - Repo with all code & trained models
- Sample Repo:
  - https://github.com/CENG502-Projects/CENG502-Spring2023/tree/main/Topcuoglu



Echo State Networks

Reservoir Computing

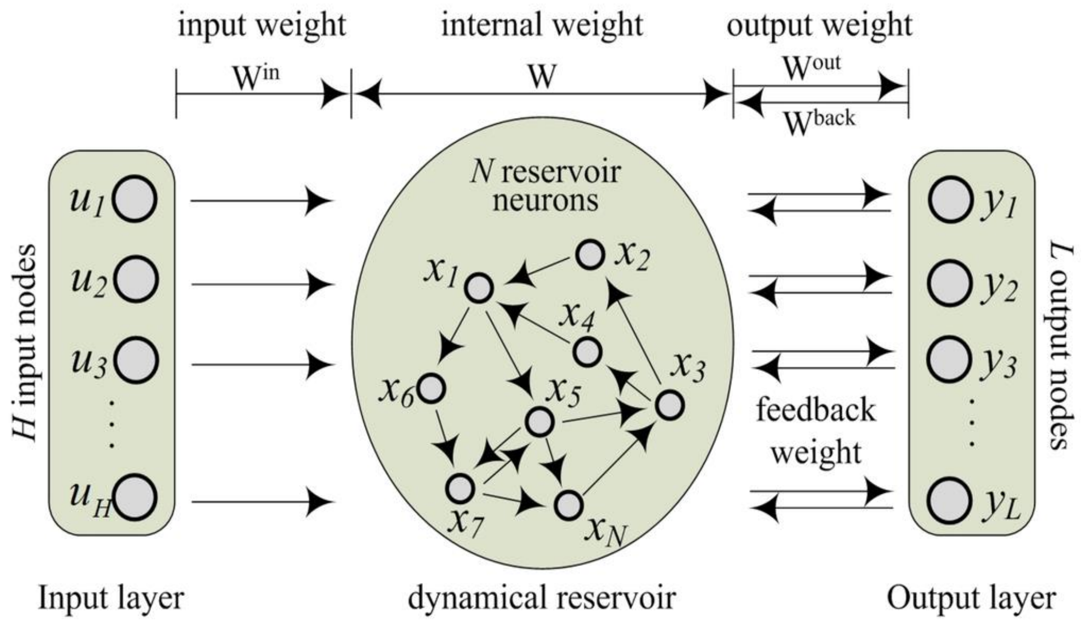
Motivation

- *"Schiller and Steil (2005) also showed that in traditional training methods for RNNs, where all weights (not only the output weights) are adapted, the dominant changes are in the output weights. In cognitive neuroscience, a related mechanism has been investigated by Peter F. Dominey in the context of modelling sequence processing in mammalian brains, especially speech recognition in humans (e.g., Dominey 1995, Dominey, Hoen and Inui 2006). Dominey was the first to explicitly state the principle of reading out target information from a randomly connected RNN. The basic idea also informed a model of temporal input discrimination in biological neural networks (Buonomano and Merzenich 1995)."*

http://www.scholarpedia.org/article/Echo_state_network

Echo State Networks (ESN)

- Reservoir of a set of neurons
  - Randomly initialized and fixed
  - Run input sequence through the network and keep the activations of the reservoir neurons
  - Calculate the “readout” weights using linear regression.
- Has the benefits of recurrent connections/networks
- No problem of vanishing gradient



Li et al., 2015.

The reservoir

- Provides non-linear expansion
  - This provides a “kernel” trick.
- Acts as a memory
- Parameters:
  - W_{in}, W and α (leaking rate).
- Global parameters:
  - Number of neurons: The more the better.
  - Sparsity: Connect a neuron to a fixed but small number of neurons.
  - Distribution of the non-zero elements: Uniform or Gaussian distribution. W_{in} is denser than W.
  - Spectral radius of W: Maximum absolute eigenvalue of W, or the width of the distribution of its non-zero elements.
    - Should be less than 1. Otherwise, chaotic, periodic or multiple fixed-point behavior may be observed.
    - For problems with large memory requirements, it should be bigger than 1.
  - Scale of the input weights.

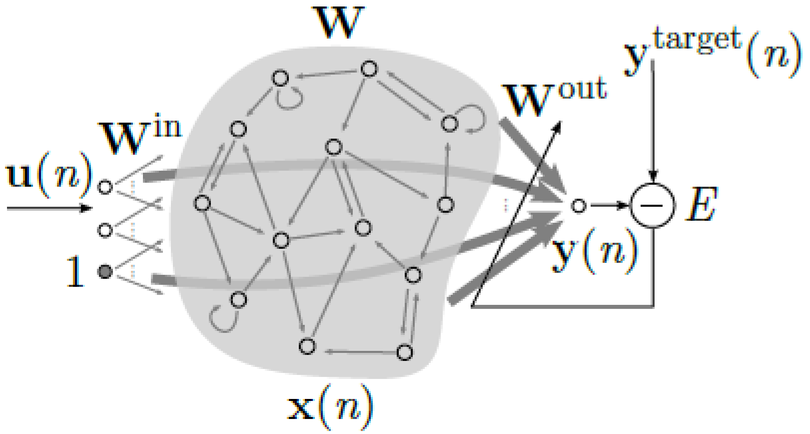


Fig. 1: An echo state network.

A Practical Guide to Applying Echo State Networks

Mantas Lukoševičius

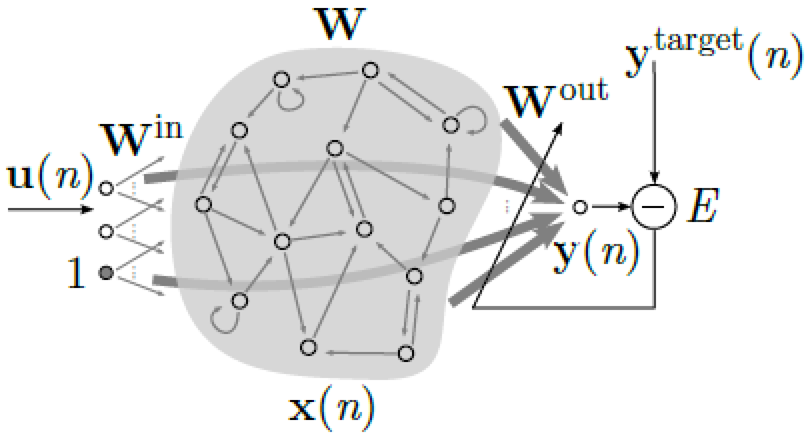
A Practical Guide to Applying Echo State Networks

Mantas Lukoševičius

$$\tilde{\mathbf{x}}(n) = \tanh\left(\mathbf{W}^{\mathrm{in}}[1; \mathbf{u}(n)] + \mathbf{W}\mathbf{x}(n-1)\right), \tag{2}$$

$$\mathbf{x}(n) = (1-\alpha)\mathbf{x}(n-1) + \alpha\tilde{\mathbf{x}}(n), \tag{3}$$

where $\mathbf{x}(n) \in \mathbb{R}^{N_{\mathrm{x}}}$ is a vector of reservoir neuron activations and $\tilde{\mathbf{x}}(n) \in \mathbb{R}^{N_{\mathrm{x}}}$ is its update, all at time step n, $\tanh(\cdot)$ is applied element-wise, $[\cdot;\cdot]$ stands for a vertical vector (or matrix) concatenation, $\mathbf{W}^{\mathrm{in}} \in \mathbb{R}^{N_{\mathrm{x}} \times (1+N_{\mathrm{u}})}$ and $\mathbf{W} \in \mathbb{R}^{N_{\mathrm{x}} \times N_{\mathrm{x}}}$ are the input and recurrent weight matrices respectively, and $\alpha \in (0, 1]$ is the leaking rate. Other sigmoid wrappers can be used besides the tanh, which however is the most common choice. The model is also sometimes used without the leaky integration, which is a special case of $\alpha = 1$ and thus $\tilde{\mathbf{x}}(n) \equiv \mathbf{x}(n)$.



$$\mathbf{y}(n) = \mathbf{W}^{\mathrm{out}}[1; \mathbf{u}(n); \mathbf{x}(n)],$$

Fig. 1: An echo state network.

again stands for a vertical vector (or matrix) concatenation. An additional nonlinearity can be applied to $\mathbf{y}(n)$ in (4), as well as feedback connections $\mathbf{W}^{\mathrm{ofb}}$ from $\mathbf{y}(n-1)$ to $\tilde{\mathbf{x}}(n)$ in (2). A graphical

Training ESN

$$\mathbf{Y}^{\text{target}} = \mathbf{W}^{\text{out}}\mathbf{X}$$

Probably the most universal and stable solution to (8) in this context is ridge regression, also known as regression with Tikhonov regularization:

$$\mathbf{W}^{\text{out}} = \mathbf{Y}^{\text{target}}\mathbf{X}^{\mathrm{T}}\left(\mathbf{X}\mathbf{X}^{\mathrm{T}} + \beta\mathbf{I}\right)^{-1}, \tag{9}$$

where β is a regularization coefficient explained in Section 4.2, and $\mathbf{I}$ is the identity matrix.

Overfitting (regularization):

$$\mathbf{W}^{\text{out}} = \underset{\mathbf{W}^{\text{out}}}{\arg\min} \frac{1}{N_{\mathrm{y}}} \sum_{i=1}^{N_{\mathrm{y}}} \left(\sum_{n=1}^{T} \left(y_i(n) - y_i^{\text{target}}(n)\right)^2 + \beta \left\|\mathbf{w}_i^{\text{out}}\right\|^2 \right),$$

Beyond echo state networks

- Good aspects of ESNs
  Echo state networks can be trained very fast because they just fit a linear model.
- They demonstrate that it's very important to initialize weights sensibly.
- They can do impressive modeling of one-dimensional time-series.
  - but they cannot compete seriously for high-dimensional data.

- Bad aspects of ESNs
  They need many more hidden units for a given task than an RNN that learns the hidden→hidden weights.

Slide: Hinton

Similar models

- Liquid State Machines (Maas et al., 2002)
  - A spiking version of Echo-state networks
- Extreme Learning Machines
  - Feed-forward network with a hidden layer.
  - Input-to-hidden weights are randomly initialized and never updated

Attention

Attention

BLEU: Bilingual Evaluation Understudy

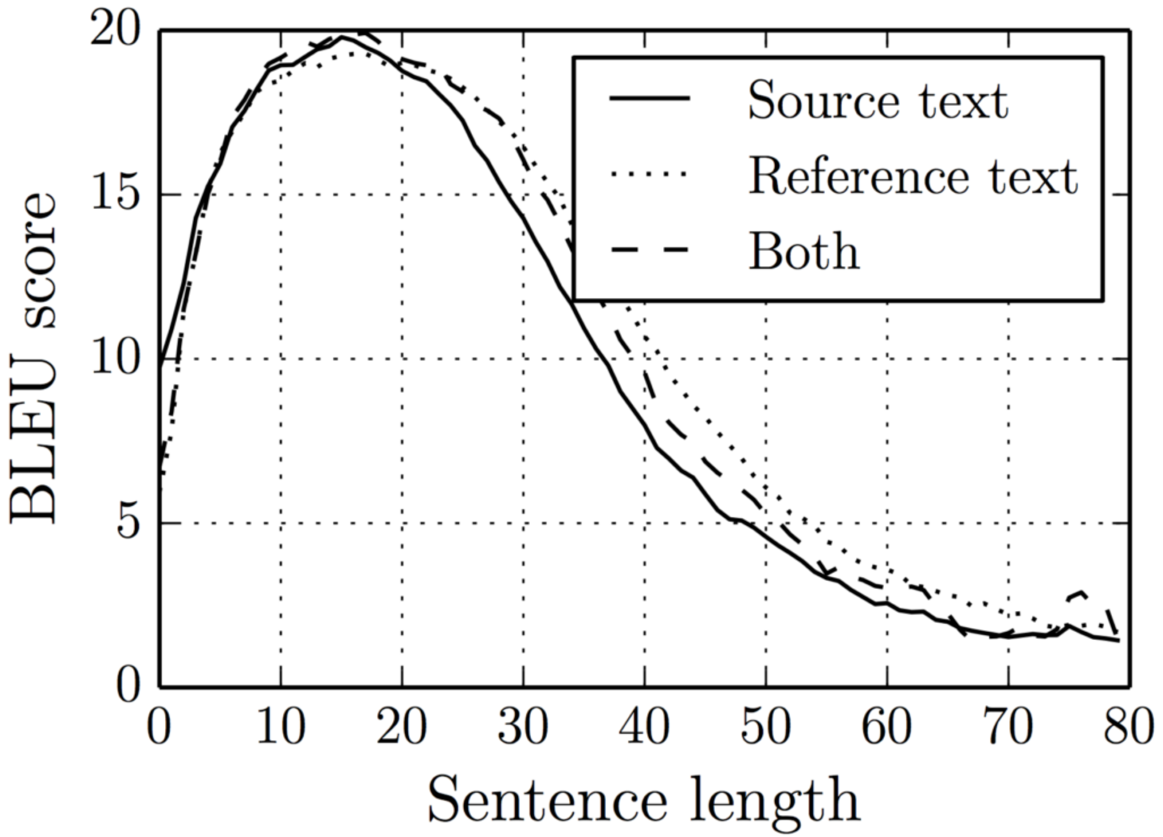
https://cloud.google.com/translate/automl/docs/evaluate#bleu

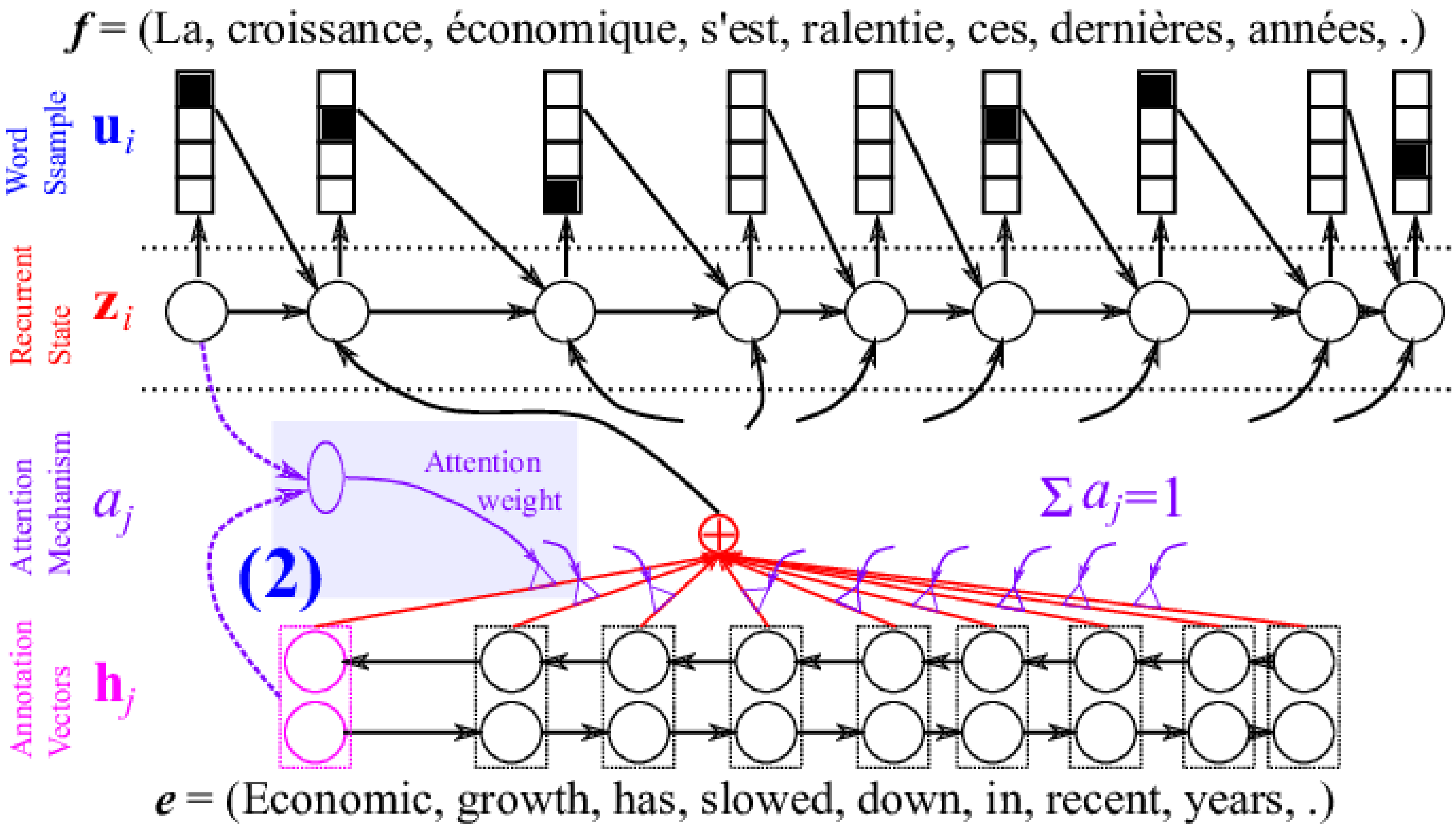
Published as a conference paper at ICLR 2015

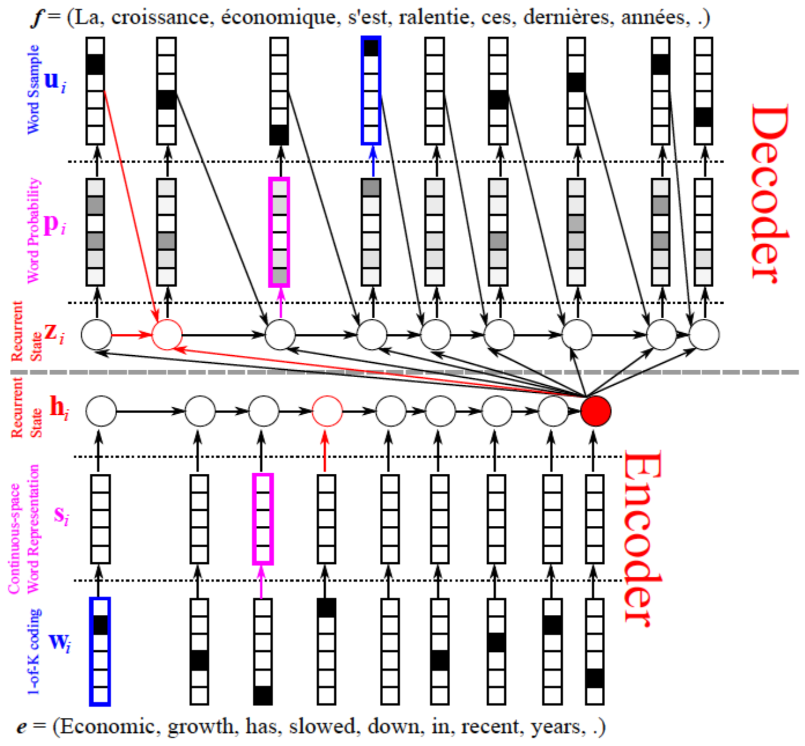
NEURAL MACHINE TRANSLATION
BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal




f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Recurrent State
$\mathbf{z}_i$
Attention Mechanism
a_j
Attention weight
$\Sigma a_j = 1$
(2)
Annotation Vectors
$\mathbf{h}_j$
e = (Economic, growth, has, slowed, down, in, recent, years, .)


f = (La, croissance, économique, s'est, ralentie, ces, dernières, années, .)
Word Ssample
$\mathbf{u}_i$
Word Probability
$\mathbf{p}_i$
Recurrent State
$\mathbf{z}_i$
Decoder
Recurrent State
$\mathbf{h}_i$
Continuous-space Word Representation
$\mathbf{s}_i$
1-of-K coding
$\mathbf{w}_i$
Encoder
e = (Economic, growth, has, slowed, down, in, recent, years, .)

Attention

Published as a conference paper at ICLR 2015

NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE

Dzmitry Bahdanau
Jacobs University Bremen, Germany

KyungHyun Cho **Yoshua Bengio***
Université de Montréal

In a new model architecture, we define each conditional probability in Eq. (2) as:

$$p(y_i|y_1, \ldots, y_{i-1}, \mathbf{x}) = g(y_{i-1}, s_i, c_i), \tag{4}$$

where s_i is an RNN hidden state for time i, computed by

$$s_i = f(s_{i-1}, y_{i-1}, c_i).$$

It should be noted that unlike the existing encoder–decoder approach (see Eq. (2)), here the probability is conditioned on a distinct context vector c_i for each target word y_i.

The context vector c_i depends on a sequence of *annotations* $(h_1, \cdots, h_{T_x})$ to which an encoder maps the input sentence. Each annotation h_i contains information about the whole input sequence with a strong focus on the parts surrounding the i-th word of the input sequence. We explain in detail how the annotations are computed in the next section.

The context vector c_i is, then, computed as a weighted sum of these annotations h_i:

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j. \tag{5}$$

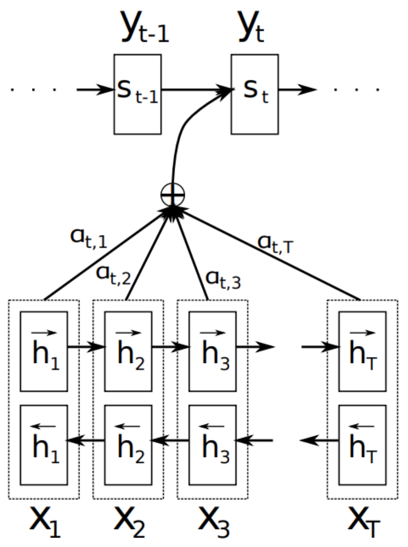


Figure 1: The graphical illustration of the proposed model trying to generate the t-th target word y_t given a source sentence $(x_1, x_2, \ldots, x_T)$.

The weight α_{ij} of each annotation h_j is computed by

$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}, \tag{6}$$

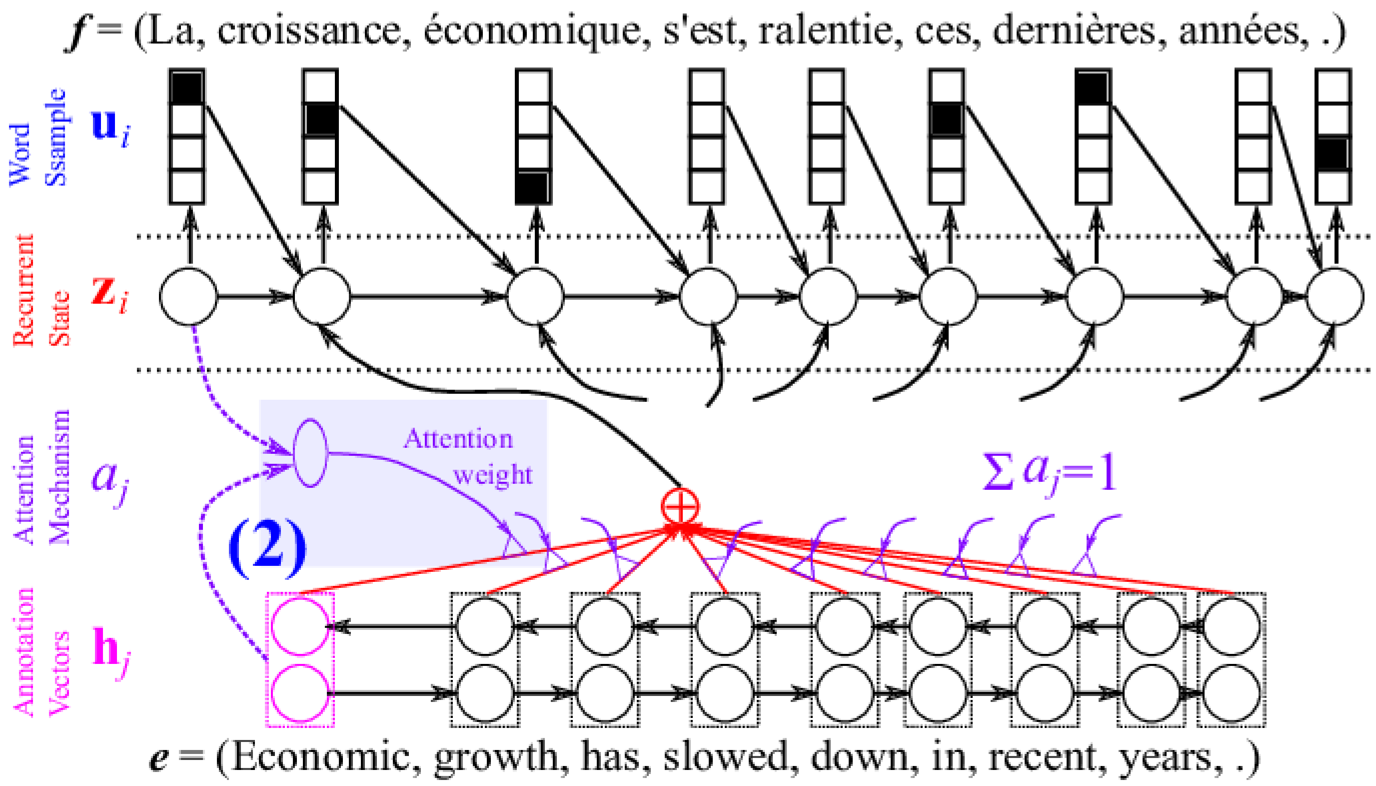
where

$$e_{ij} = a(s_{i-1}, h_j)$$

is an *alignment model* which scores how well the inputs around position j and the output at position i match. The score is based on the RNN hidden state s_{i-1} (just before emitting y_i, Eq. (4)) and the j-th annotation h_j of the input sentence.

We parametrize the alignment model a as a feedforward neural network which is jointly trained with all the other components of the proposed system. Note that unlike in traditional machine translation,

Attention



Attention mechanism: A two-layer neural network.

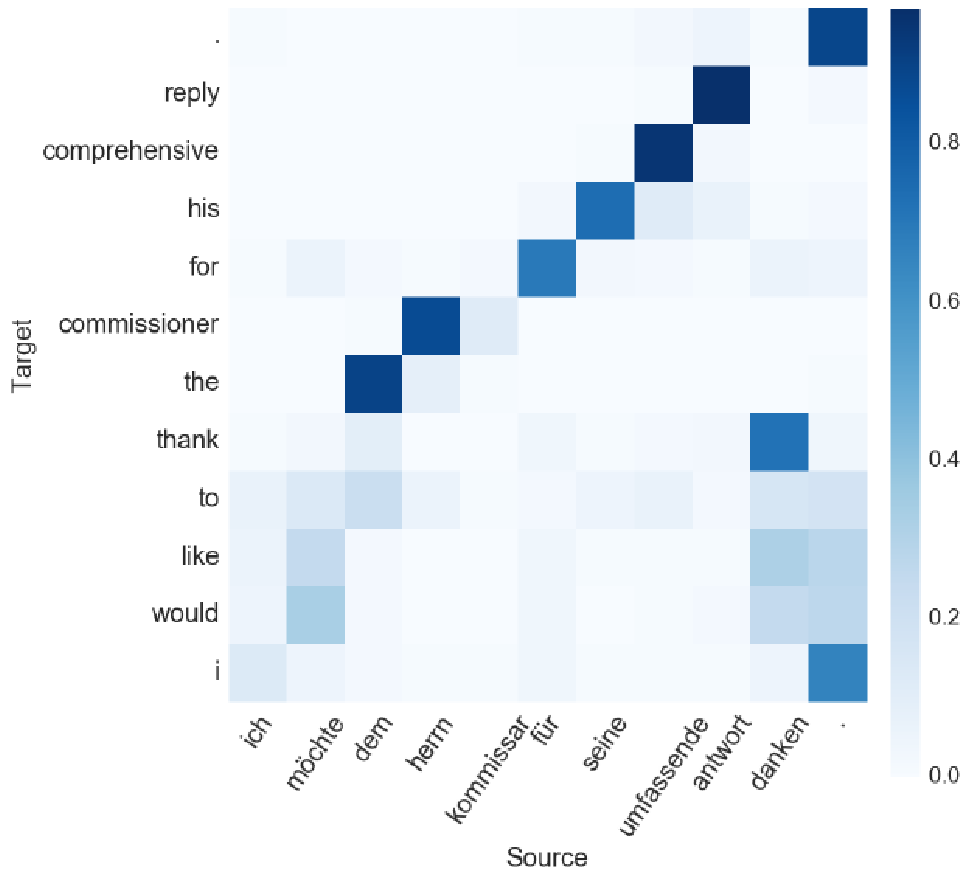
Input: z_i and h_j

Output: e_j, a scalar for the importance of word j.

The scores of words are normalized: $a_j = \mathrm{softmax}(\mathrm{e_j})$

https://devblogs.nvidia.com/introduction-neural-machine-translation-gpus-part-3/

Attention



What does Attention in Neural Machine Translation Pay Attention to?

Hamidreza Ghader and **Christof Monz**
Informatics Institute, University of Amsterdam, The Netherlands
`h.ghader, c.monz@uva.nl`

2017

Attention Types

- Let's rewrite Bahdanau et al.'s attention model:

$$\mathbf{c}_t = \sum_{i=1}^{n} \alpha_{t,i} \boldsymbol{h}_i \qquad \text{; Context vector for output } y_t$$

$$\alpha_{t,i} = \text{align}(y_t, x_i) \qquad \text{; How well two words } y_t \text{ and } x_i \text{ are aligned.}$$

$$= \frac{\exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_i))}{\sum_{i'=1}^{n} \exp(\text{score}(\boldsymbol{s}_{t-1}, \boldsymbol{h}_{i'}))} \qquad \text{; Softmax of some predefined alignment score..}$$

$$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$$

where both $\mathbf{v}_a$ and $\mathbf{W}_a$ are weight matrices to be learned in the alignment model.

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Attention Types

Name	Alignment score function	Citation
Content-base attention	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \text{cosine}[\boldsymbol{s}_t, \boldsymbol{h}_i]$	Graves2014
Additive(*)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[\boldsymbol{s}_t; \boldsymbol{h}_i])$	Bahdanau2015
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a \boldsymbol{s}_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	Luong2015
General	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \mathbf{W}_a \boldsymbol{h}_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	Luong2015
Dot-Product	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \boldsymbol{s}_t^\top \boldsymbol{h}_i$	Luong2015
Scaled Dot-Product(^)	$\text{score}(\boldsymbol{s}_t, \boldsymbol{h}_i) = \frac{\boldsymbol{s}_t^\top \boldsymbol{h}_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	Vaswani2017

(*) Referred to as "concat" in Luong, et al., 2015 and as "additive attention" in Vaswani, et al., 2017.
(^) It adds a scaling factor $1/\sqrt{n}$, motivated by the concern when the input is large, the softmax function may have an extremely small gradient, hard for efficient learning.

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Vanilla Self-attention

$$e_i{}' = \sum_j \frac{\exp\left(e_j^T e_i\right)}{\sum_m \exp\left(e_m^T e_i\right)} e_j$$

Attention: Transformer


Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com
Noam Shazeer* Google Brain noam@google.com
Niki Parmar* Google Research nikip@google.com
Jakob Uszkoreit* Google Research usz@google.com
Llion Jones* Google Research llion@google.com
Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu
Łukasz Kaiser* Google Brain lukaszkaiser@google.com
Illia Polosukhin* ‡ illia.polosukhin@gmail.com

- Vanilla self attention:

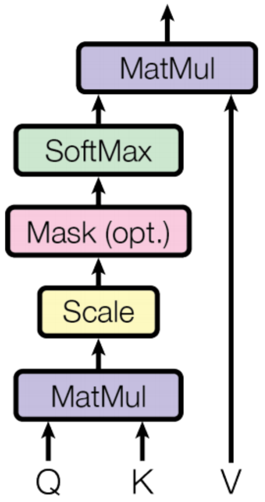
$$e_i{}' = \sum_j \frac{\exp\left(e_j^T e_i\right)}{\sum_m \exp\left(e_m^T e_i\right)} e_j$$

- Scaled-dot product attention:

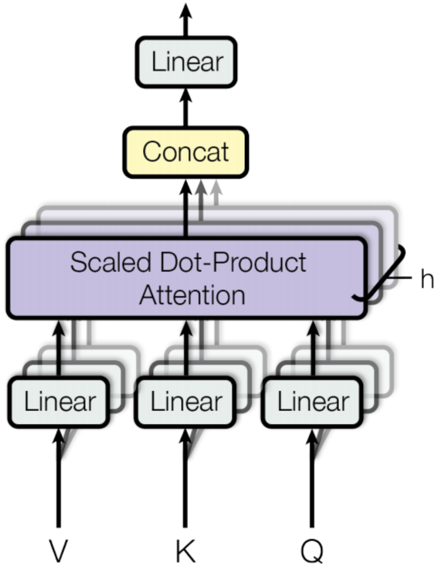
$$e_i{}' = \sum_j \frac{\exp\left(k\left(e_j^T\right) q\left(e_i\right)\right)}{\sum_m \exp\left(k\left(e_m^T\right) q\left(e_i\right)\right)} v\left(e_j\right)$$

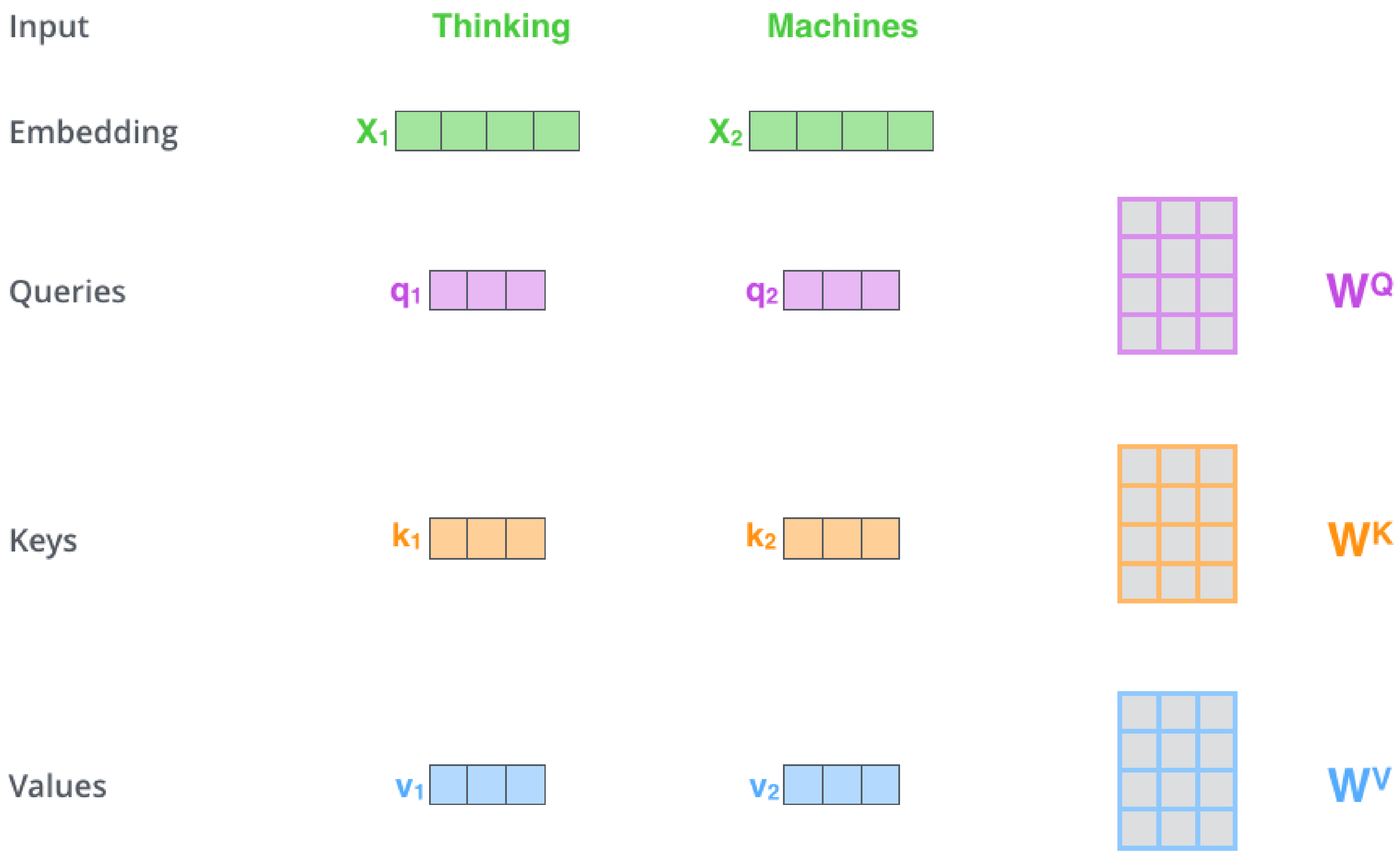
$$\text{Attention}(Q,K,V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

Scaled Dot-Product Attention

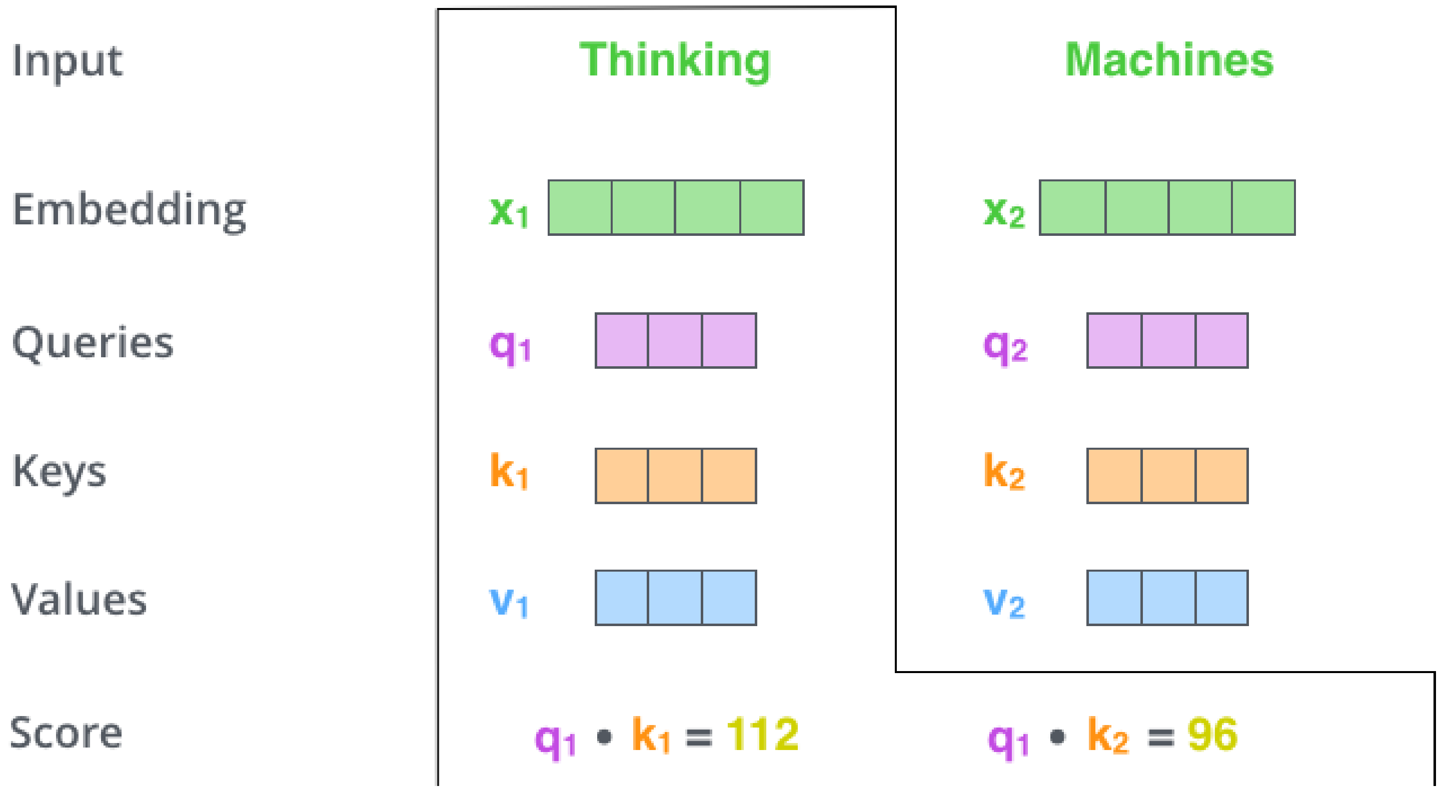


Multi-Head Attention

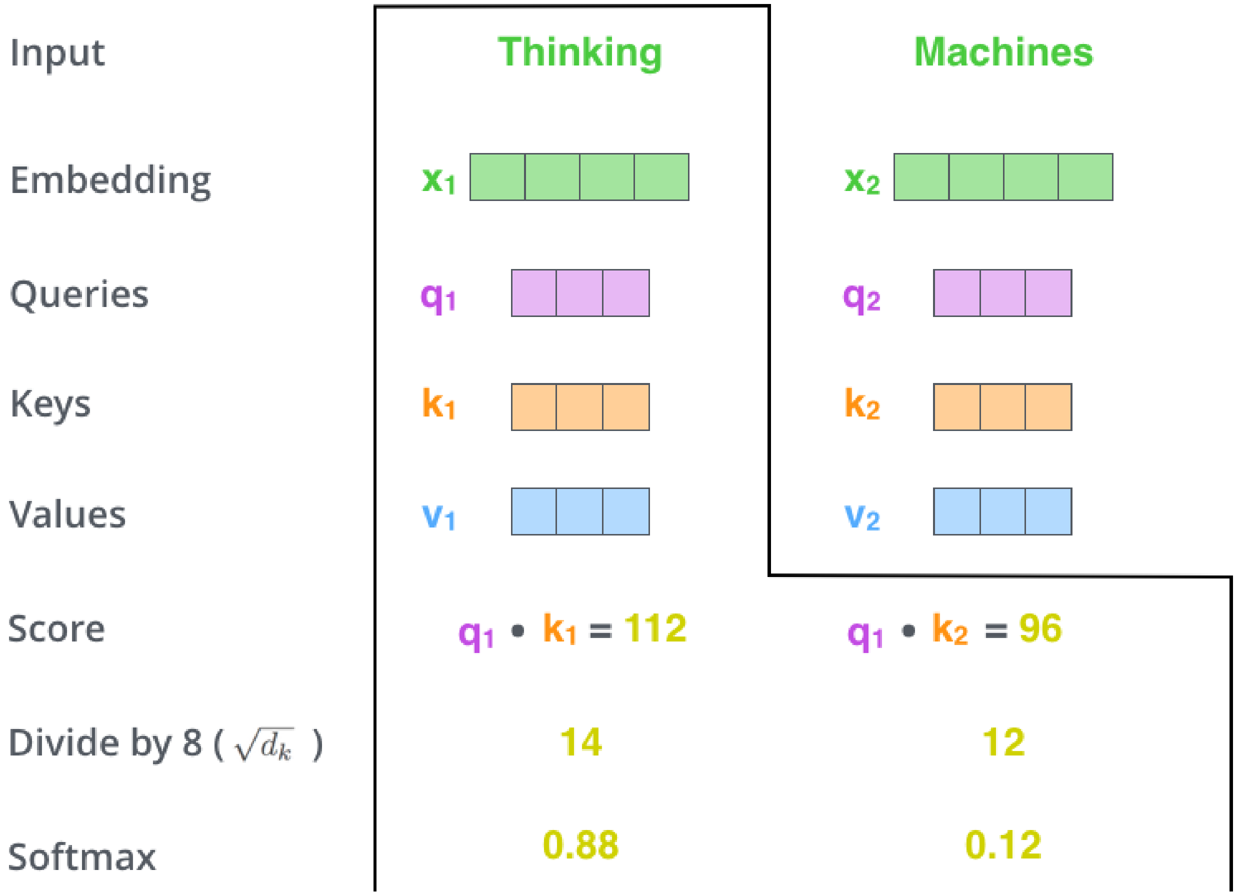




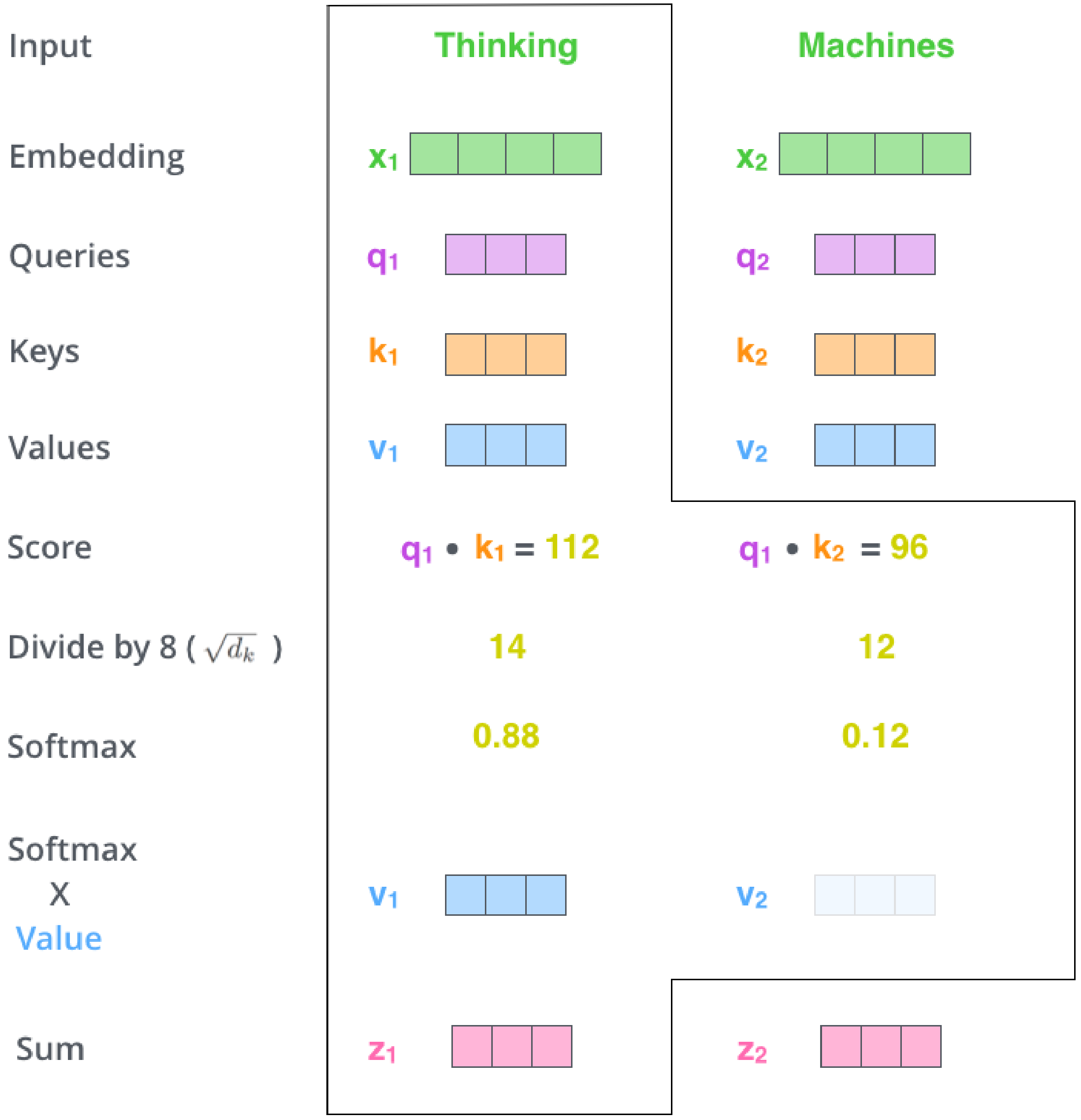
https://jalammar.github.io/illustrated-transformer/



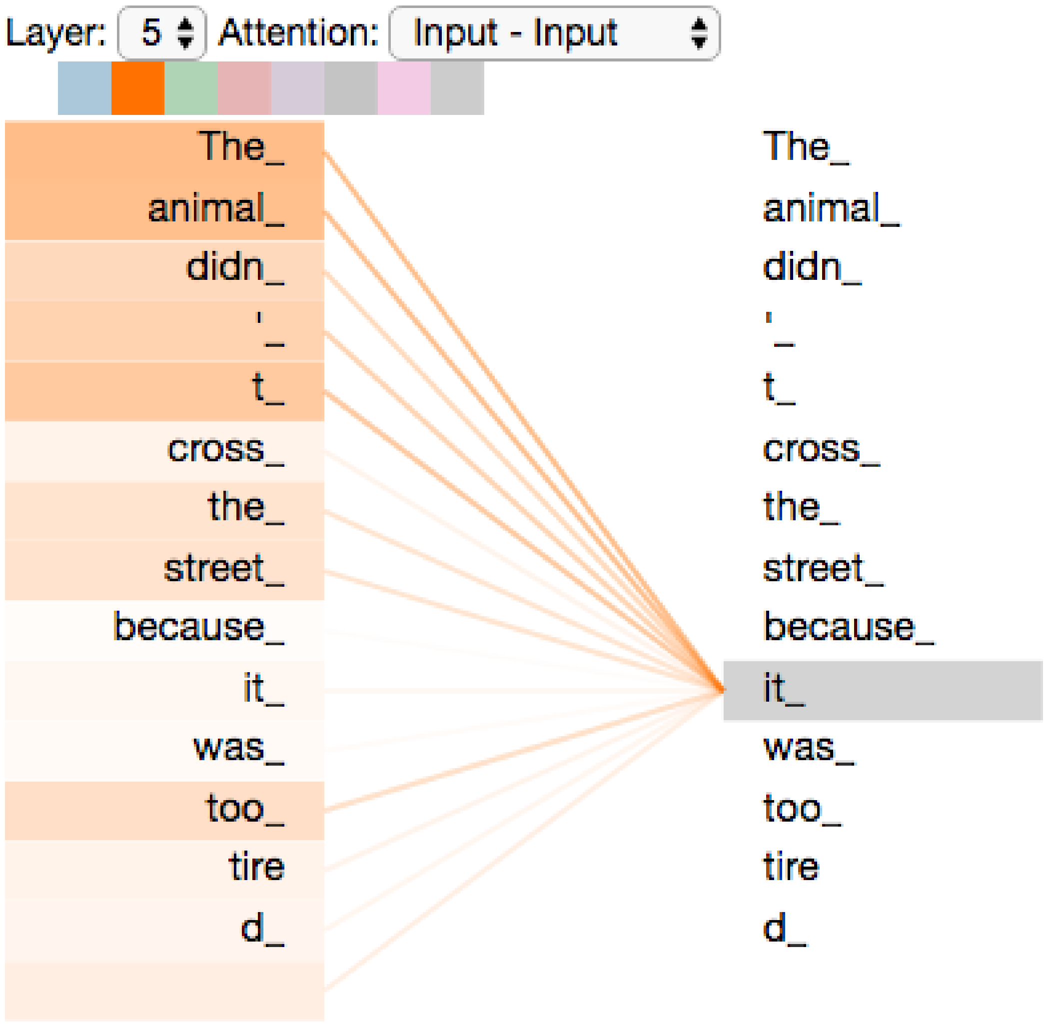
https://jalammar.github.io/illustrated-transformer/



https://jalammar.github.io/illustrated-transformer/



https://jalammar.github.io/illustrated-transformer/



https://jalammar.github.io/illustrated-transformer/

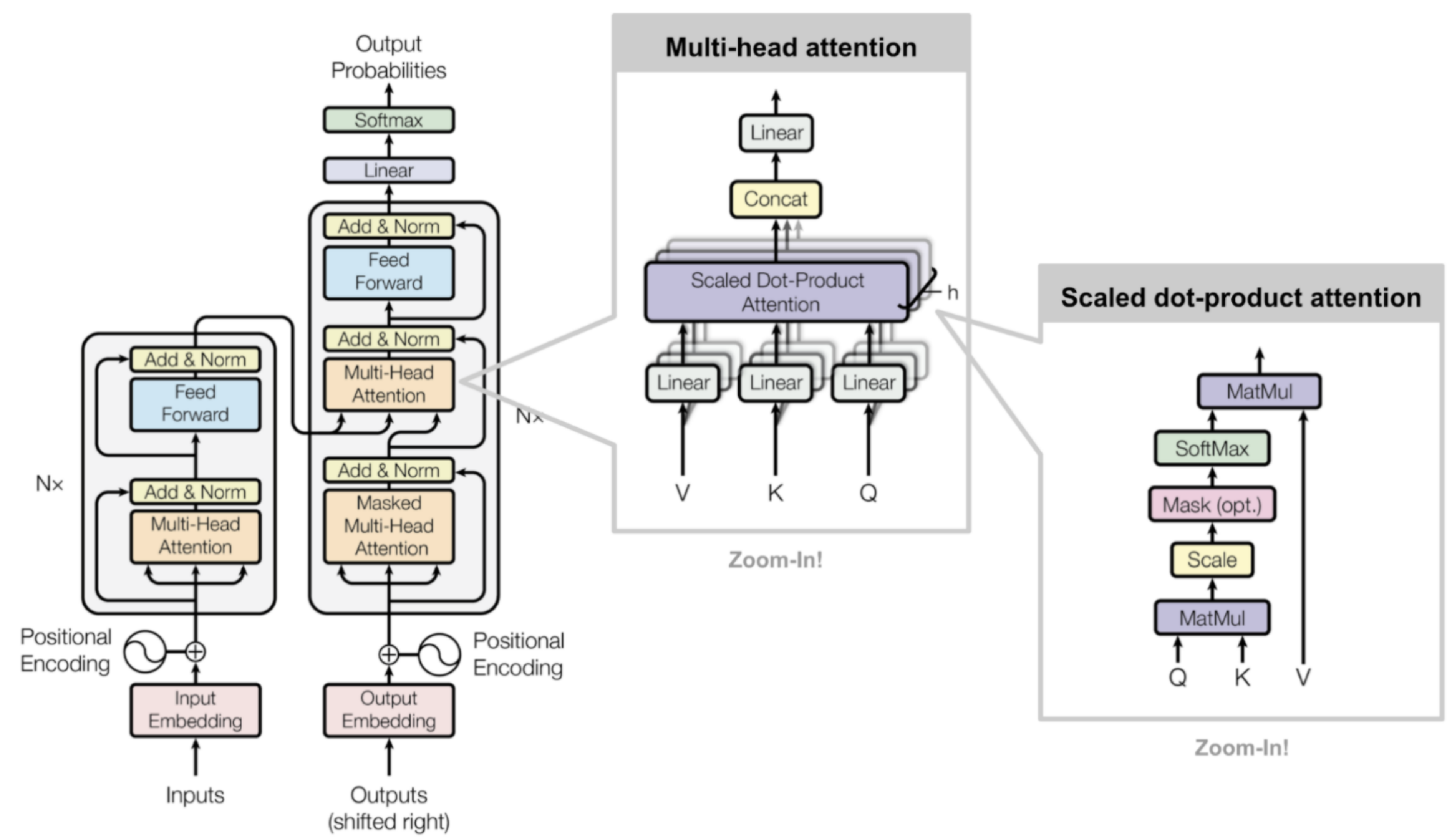


Fig. 17. The full model architecture of the transformer. (Image source: Fig 1 & 2 in Vaswani, et al., 2017.)

https://lilianweng.github.io/lil-log/2018/06/24/attention-attention.html

Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

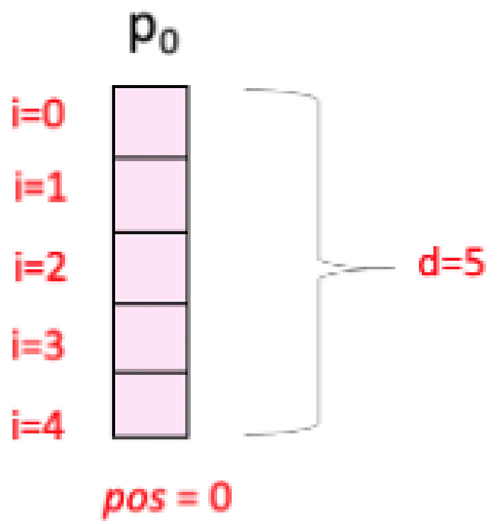


Fig from: https://www.youtube.com/watch?v=dichIcUZfOw

Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{\boxed{pos}}{10000^{\frac{2i}{d}}}\right)$$

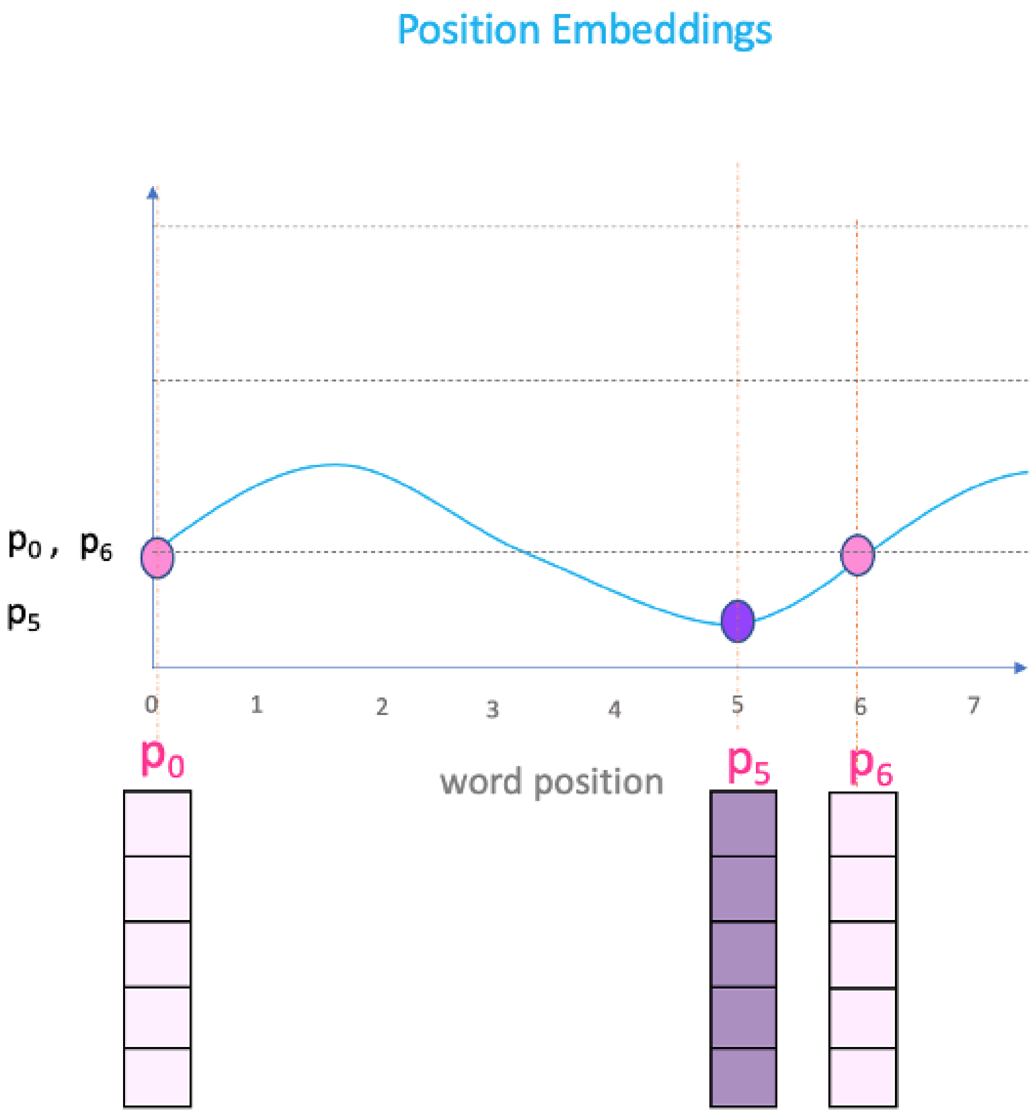


Fig from: https://www.youtube.com/watch?v=dichIcUZfOw

Positional Encoding

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{\frac{2i}{d}}}\right)$$

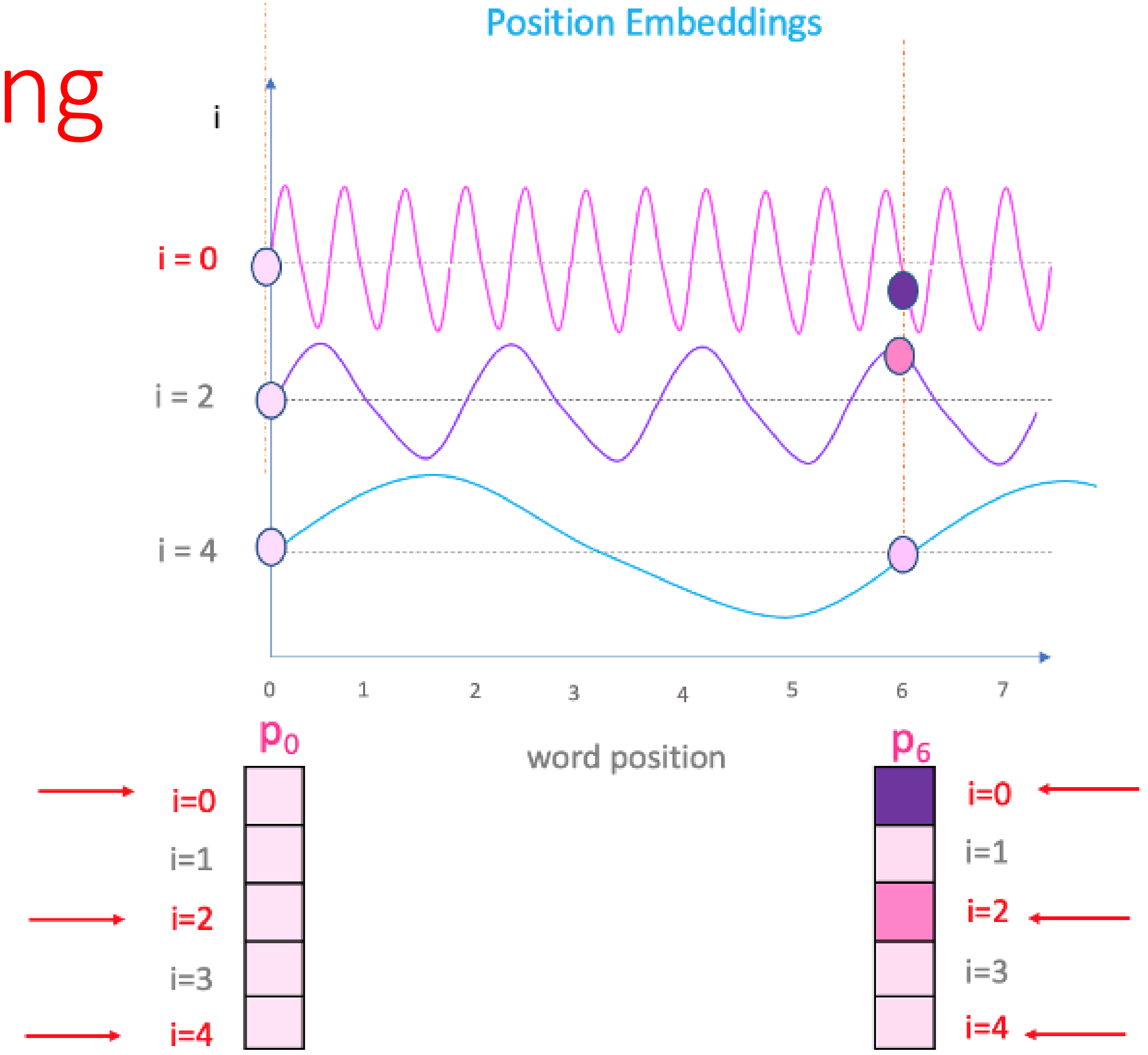
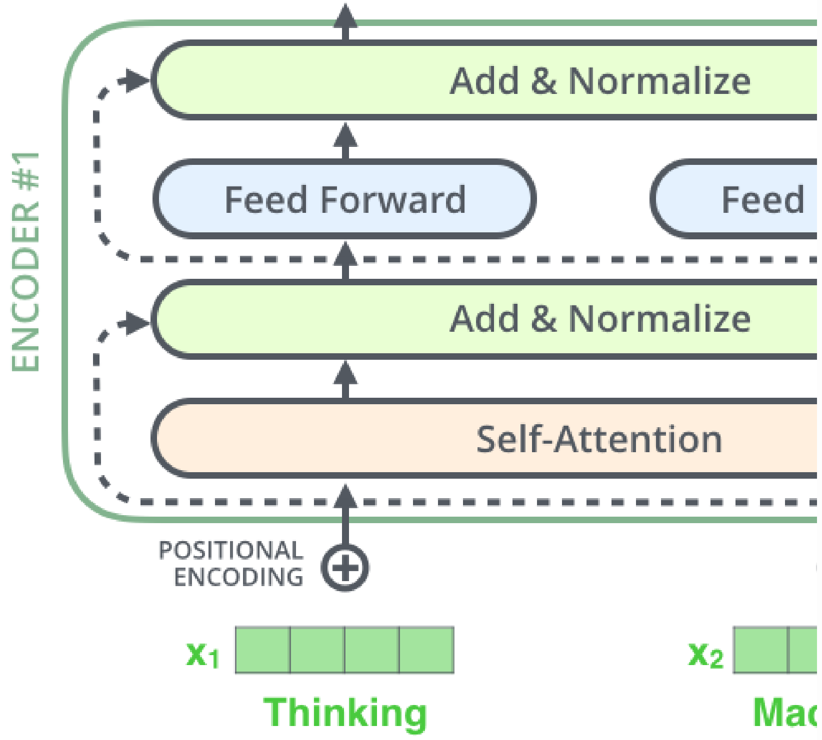
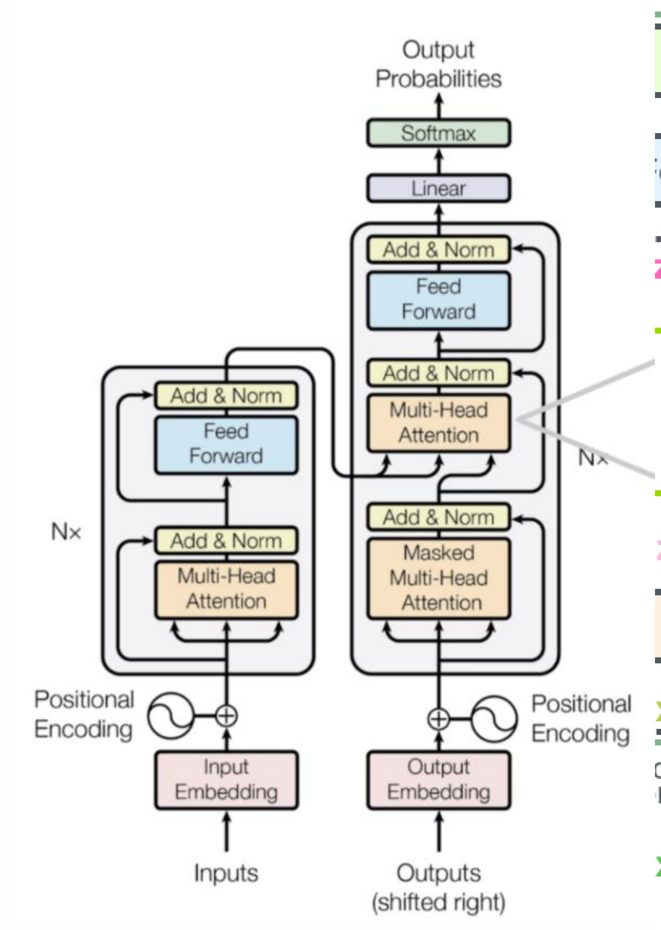
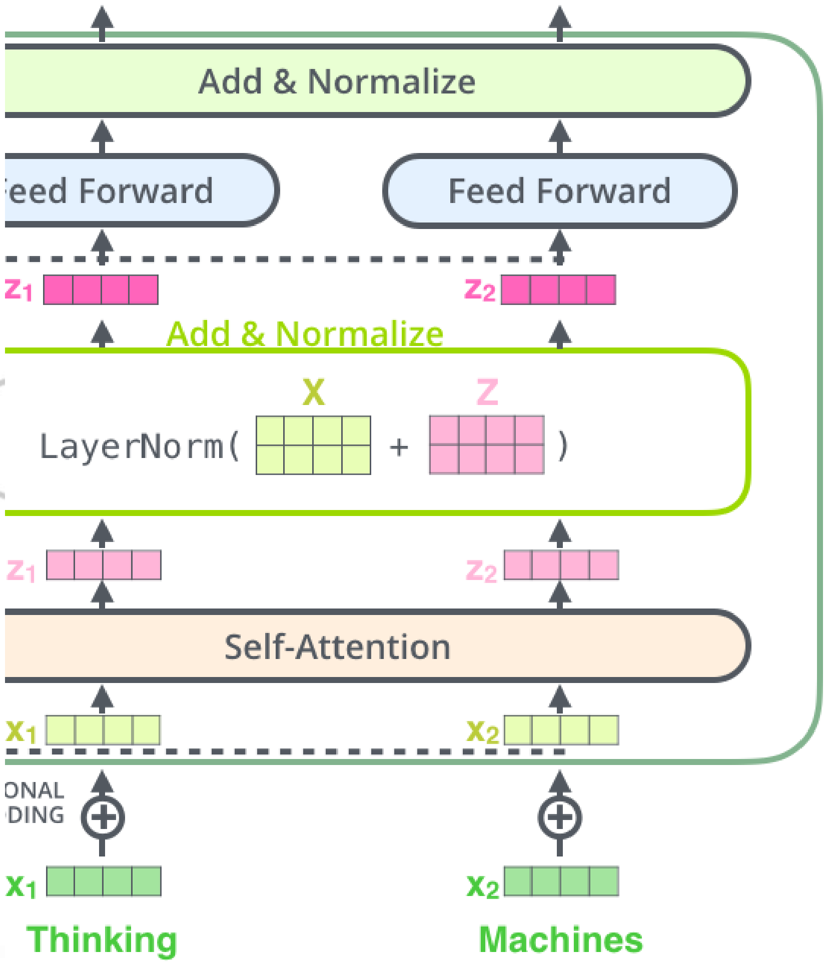


Fig from: https://www.youtube.com/watch?v=dichIcUZfOw

Skip Connections & Normalization

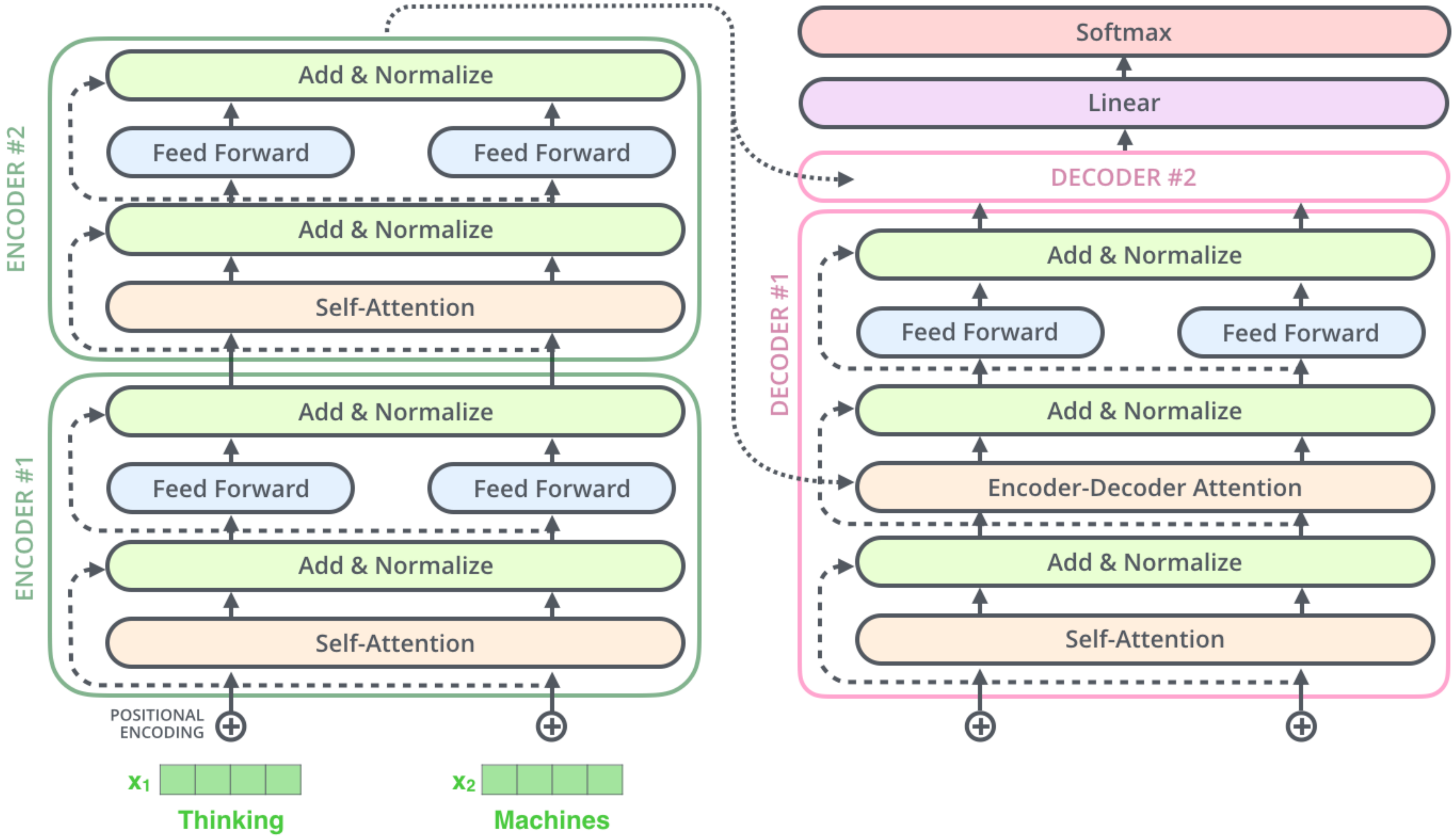






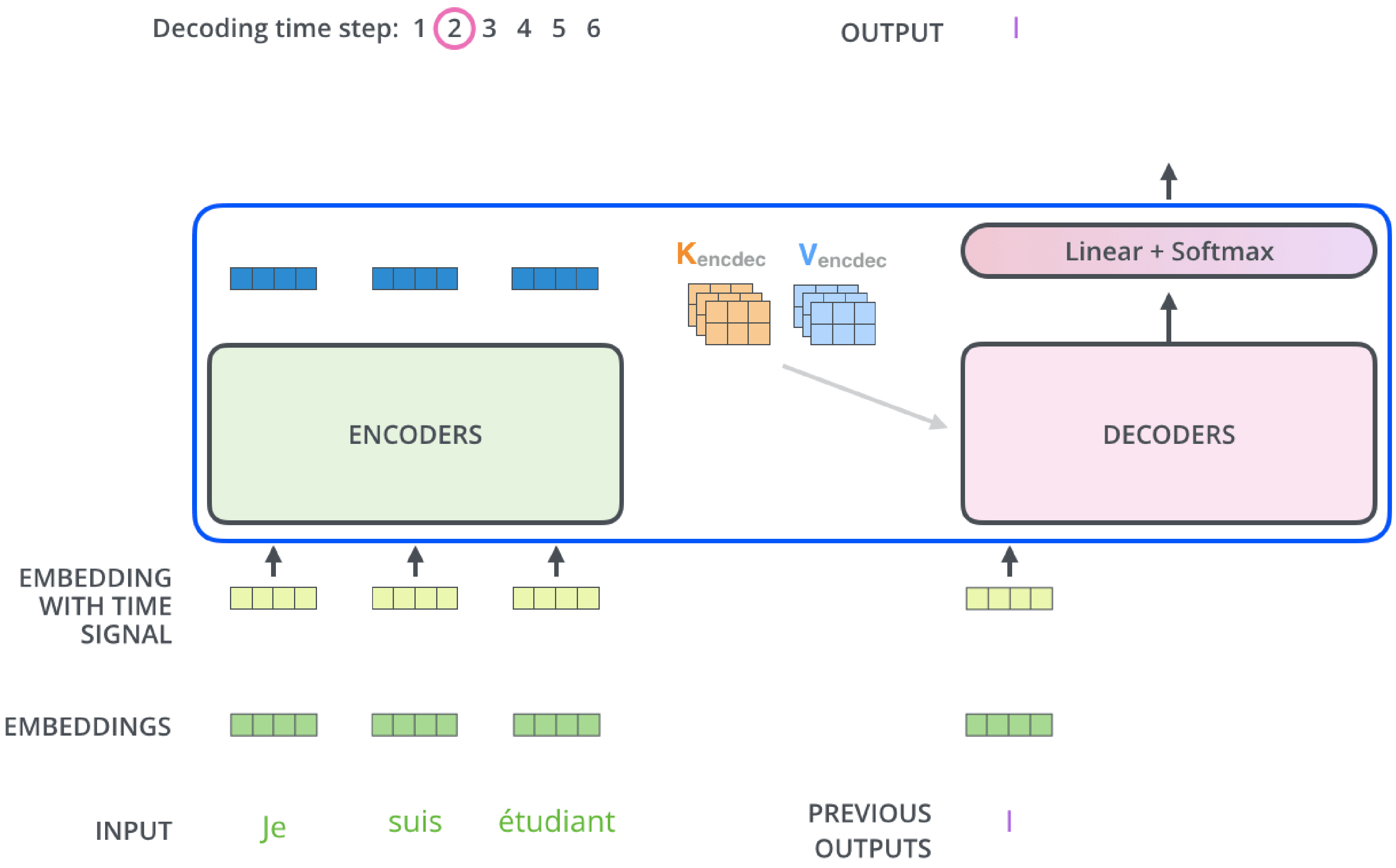
https://jalammar.github.io/illustrated-transformer/

Skip Connections & Normalization



https://jalammar.github.io/illustrated-transformer/

Decoder



Decoding time step: 1 2 3 4 5 6
OUTPUT
I
Linear + Softmax
Kencdec
Vencdec
ENCODERS
DECODERS
EMBEDDING WITH TIME SIGNAL
EMBEDDINGS
INPUT
Je
suis
étudiant
PREVIOUS OUTPUTS
I

Tutorial on transformers

- https://e2eml.school/transformers.html
- https://jalammar.github.io/illustrated-transformer/

A Significant Issue with Self-Attention: Complexity

$$e_i' = \sum_j \frac{\exp\left(k\left(e_j^T\right)q(e_i)\right)}{\sum_m \exp(k(e_m^T)q(e_i))} v(e_j)$$

- If there are n tokens/embeddings,
  - Updating a single tokens require $O(n)$ operations.
  - Overall: $O(n^2)$
- What is the complexity of an RNN layer with n time steps?

ICML 2020

Transformers are RNNs:
Fast Autoregressive Transformers with Linear Attention

Angelos Katharopoulos [1 2] Apoorv Vyas [1 2] Nikolaos Pappas [3] François Fleuret [2 4 *]

Linear Attention

Self-attention:

$$\begin{aligned} Q &= xW_Q, \\ K &= xW_K, \\ V &= xW_V, \\ A_l(x) = V' &= \operatorname{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V. \end{aligned} \tag{2}$$

Rewrite Eq 2 for one row of the matrix:

$$V_i' = \frac{\sum_{j=1}^{N} \operatorname{sim}(Q_i, K_j) V_j}{\sum_{j=1}^{N} \operatorname{sim}(Q_i, K_j)}. \tag{3}$$

Equation 3 is equivalent to equation 2 if we substitute the similarity function with $\operatorname{sim}(q, k) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$.

Constraint for sim(): It should be non-negative.
Then, we can choose any other kernel/function:

Given such a kernel with a feature representation $\phi(x)$ we can rewrite equation 2 as follows,

$\phi(x) = \operatorname{elu}(x) + 1$

$$V_i' = \frac{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j) V_j}{\sum_{j=1}^{N} \phi(Q_i)^T \phi(K_j)}, \tag{4}$$

and then further simplify it by making use of the associative property of matrix multiplication to

$$V_i' = \frac{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j) V_j^T}{\phi(Q_i)^T \sum_{j=1}^{N} \phi(K_j)}. \tag{5}$$

The above equation is simpler to follow when the numerator is written in vectorized form as follows,

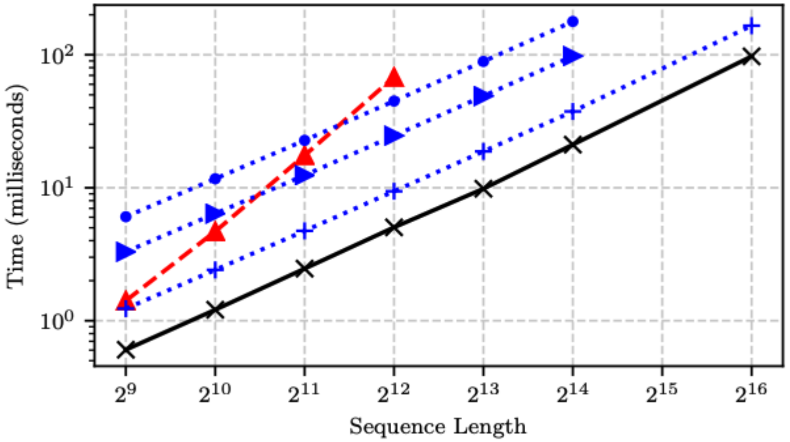
$$\left(\phi(Q)\phi(K)^T\right)V = \phi(Q)\left(\phi(K)^T V\right). \tag{6}$$

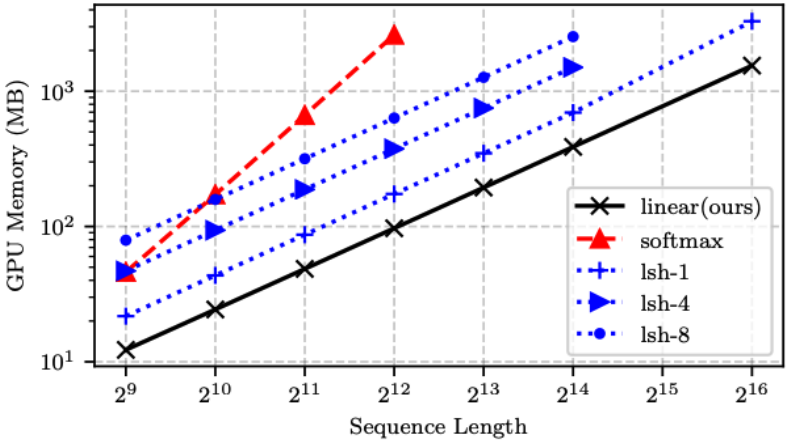
Note that the feature map $\phi(\cdot)$ is applied rowwise to the matrices Q and K.

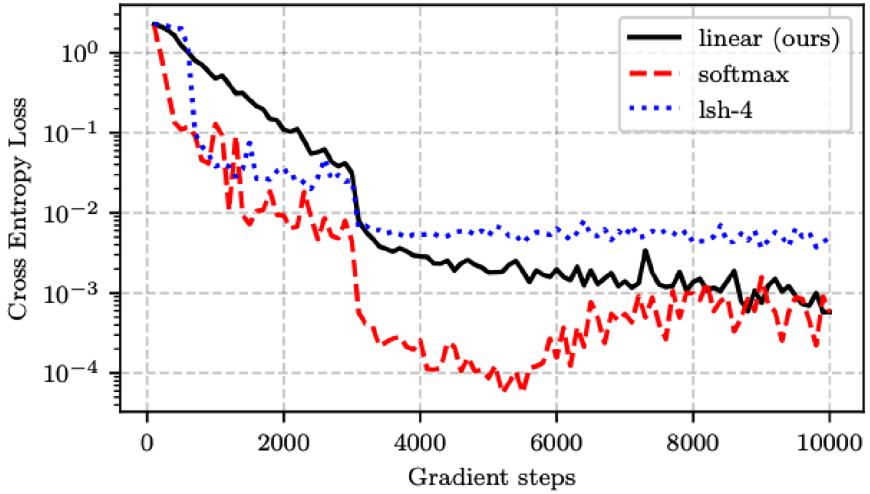
Transformers are RNNs: Fast Autoregressive Transformers with Linear Attention

Angelos Katharopoulos [1 2] Apoorv Vyas [1 2] Nikolaos Pappas [3] François Fleuret [2 4 *]

Linear Attention







Method	Validation PER	Time/epoch (s)
Bi-LSTM	10.94	1047
Softmax	5.12	2711
LSH-4	9.33	2250
Linear (ours)	8.08	**824**

Table 3: Performance comparison in automatic speech recognition on the WSJ dataset. The results are given in the form of phoneme error rate (PER) and training time per epoch. Our model outperforms the LSTM and Reformer while being faster to train and evaluate. Details of the exper-

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu*[1] and Tri Dao*[2]

[1]Machine Learning Department, Carnegie Mellon University

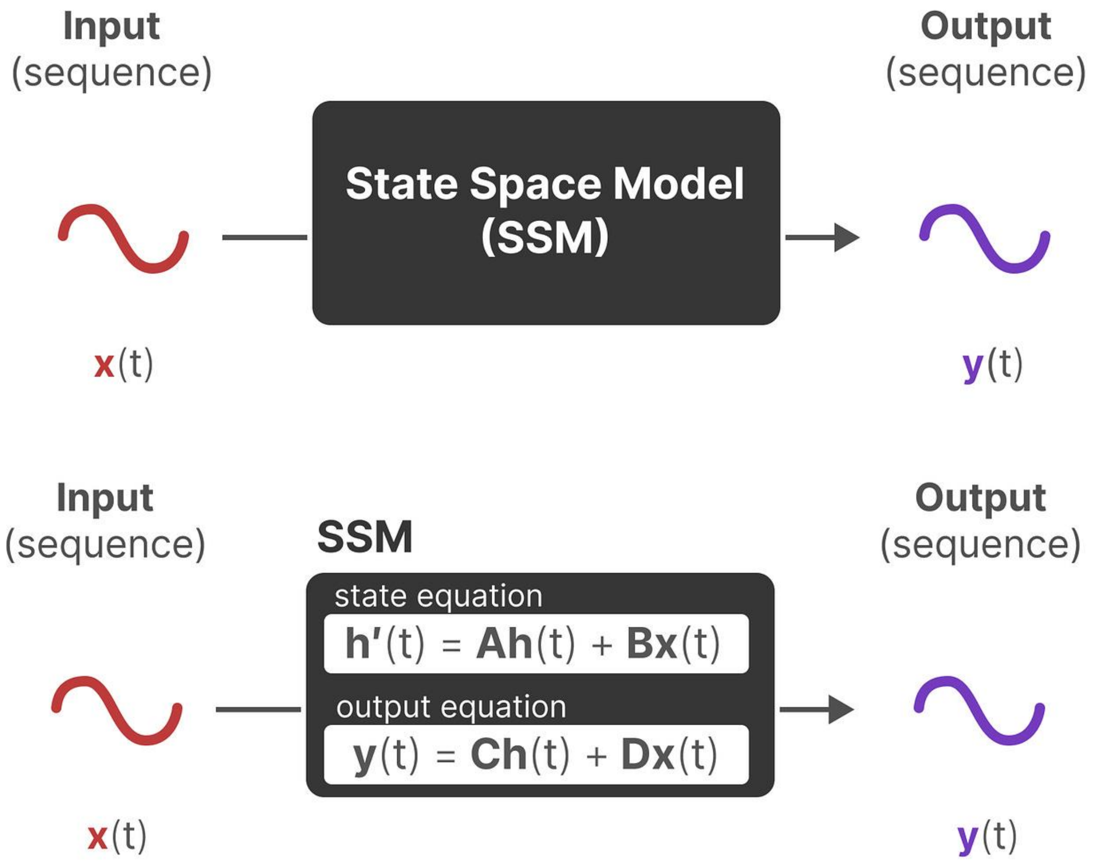
[2]Department of Computer Science, Princeton University
agu@cs.cmu.edu, tri@tridao.me

Rejected at ICLR2024

Structured State Space Sequence (S4) Model

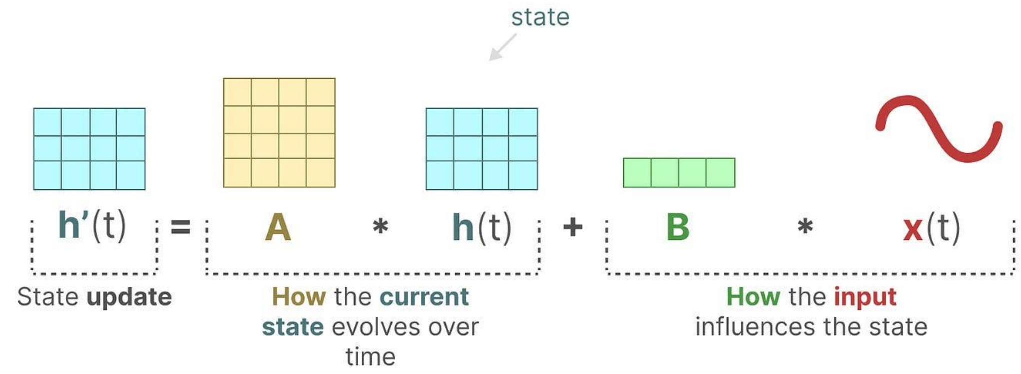
State Space Models (SSMs)

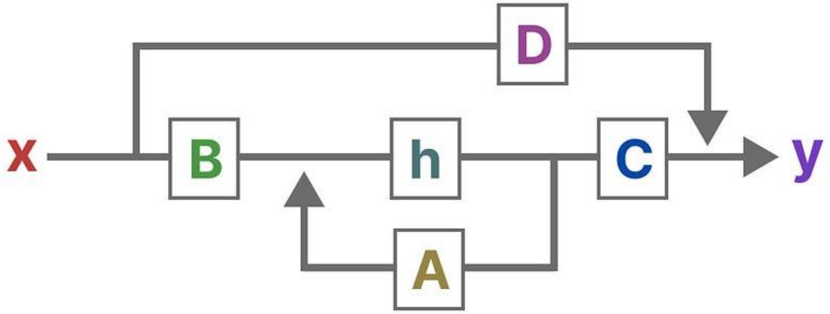
- Notation:
  - x(t): input (e.g., observation)
  - h(t): latent state representation
  - y(t): predicted output
- State update equation:
  - $h'(t) = A\,h(t) + B\,x(t)$
- Output equation:
  - $y(t) = C\,h(t) + D\,x(t)$
- A, B, C, D: learnable params

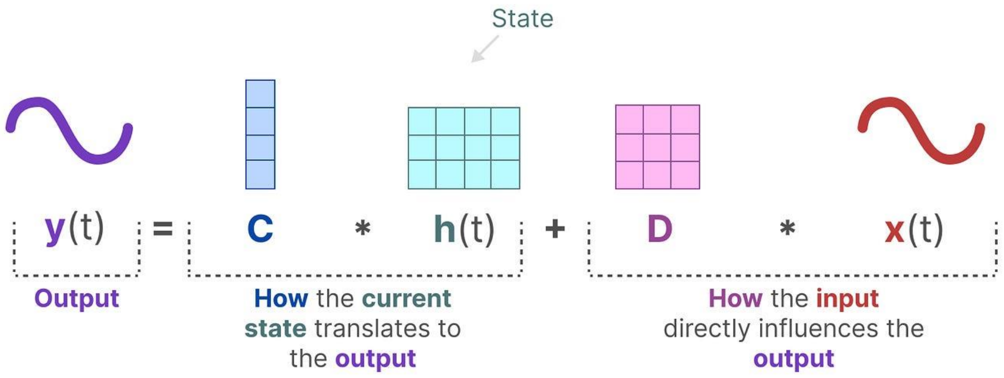


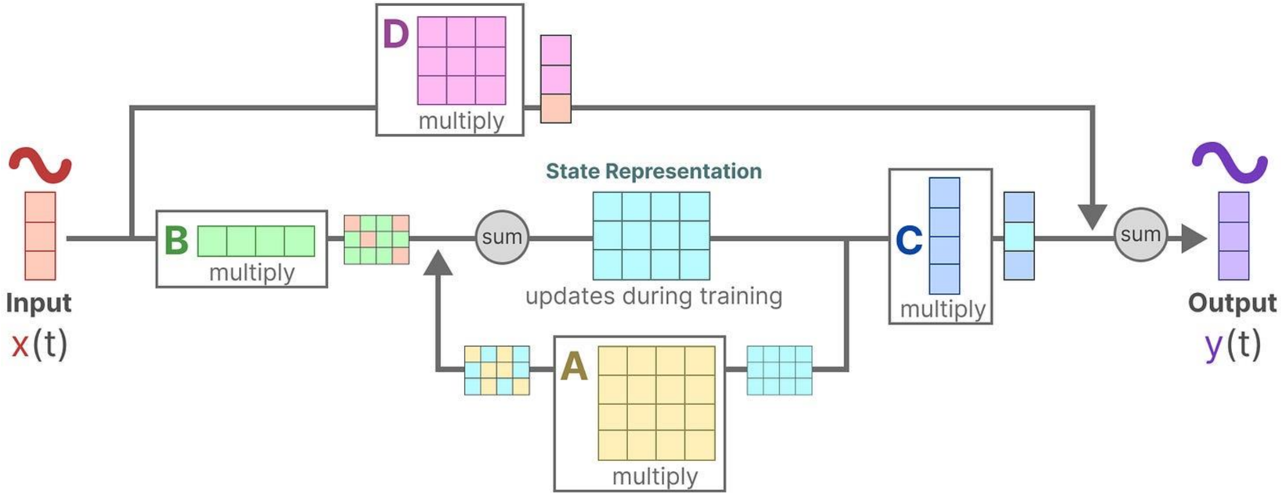
https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models (SSMs)



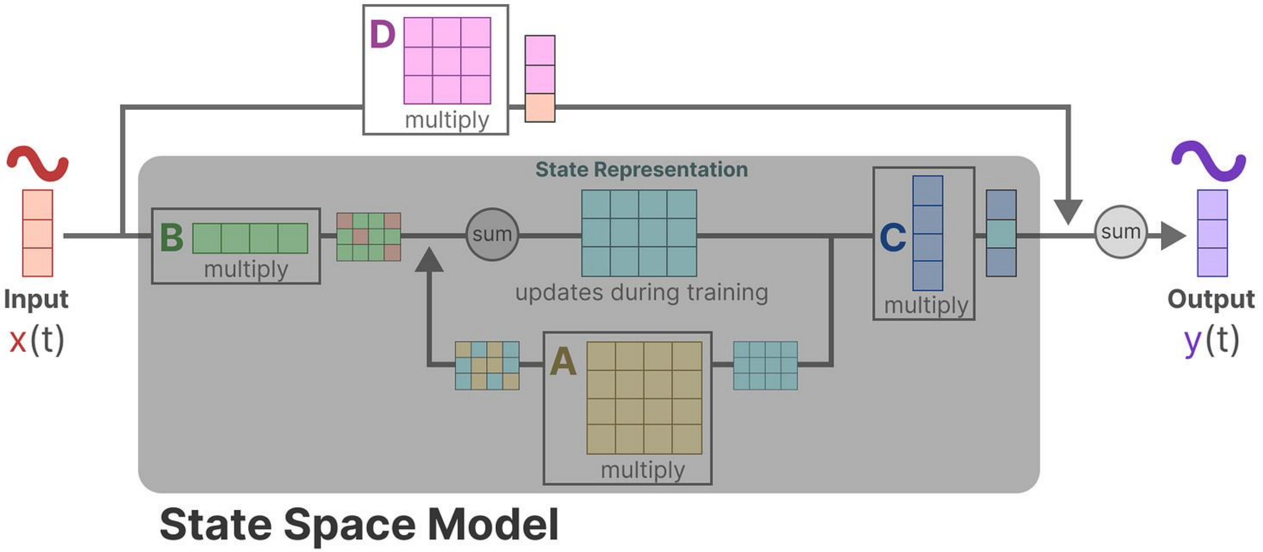


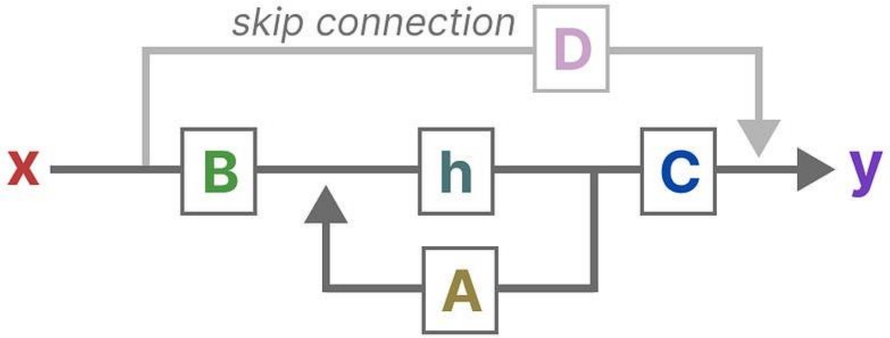


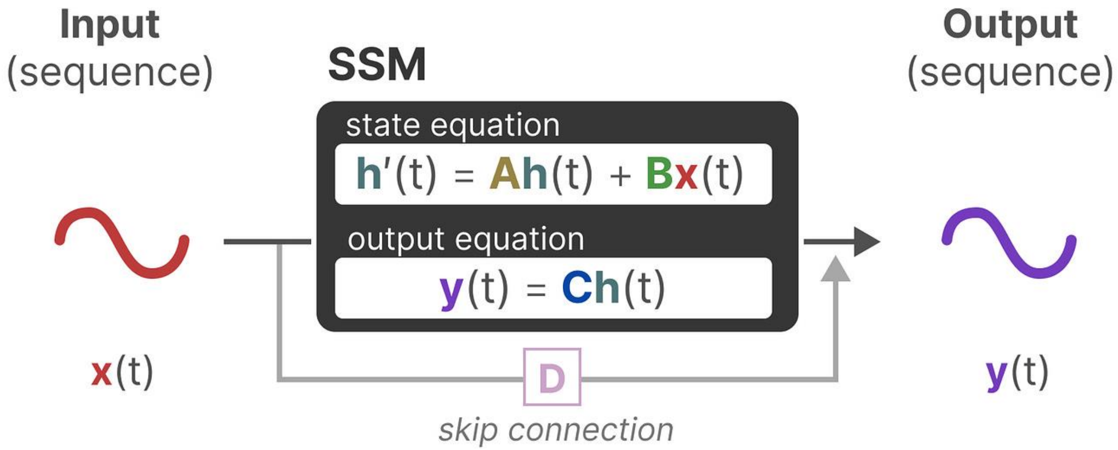


https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models (SSMs)



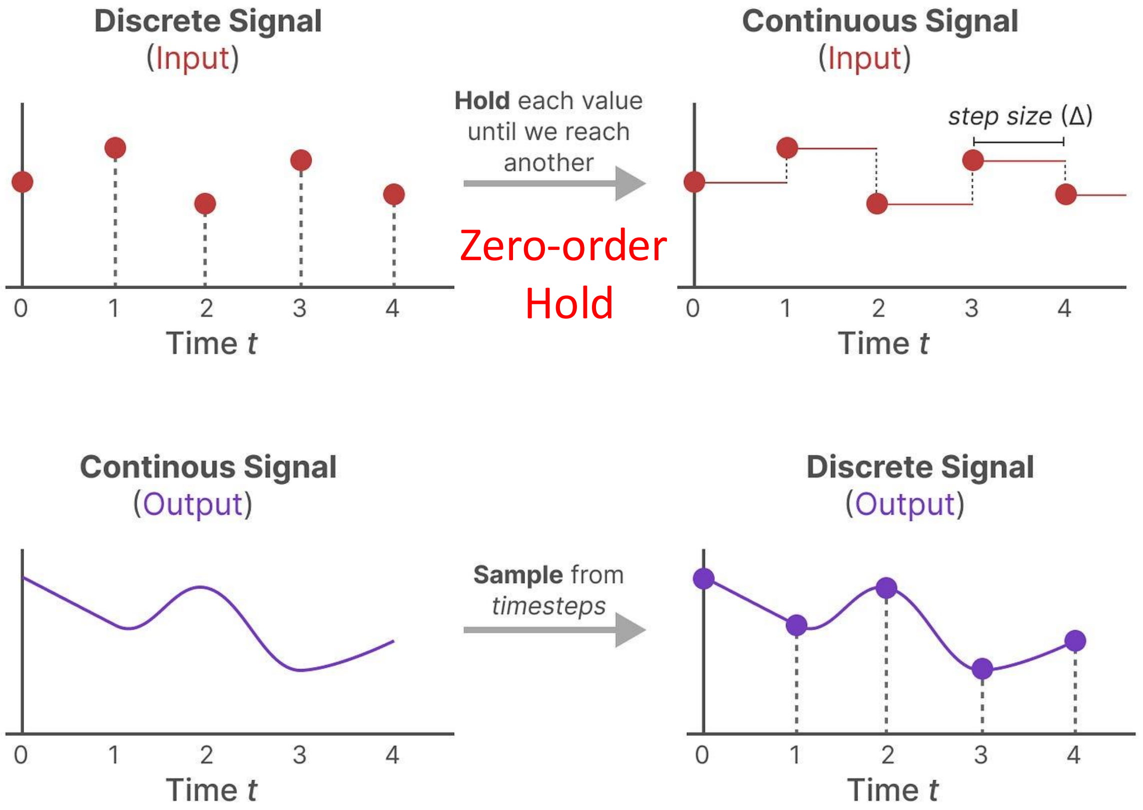




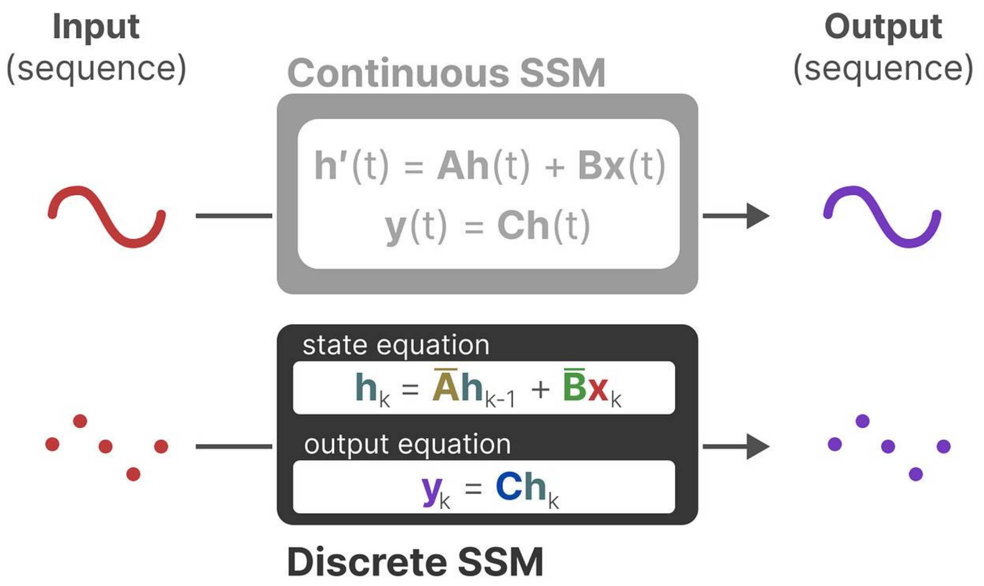
https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models (SSMs)

- Convert discrete signal to a continuous signal
- Obtain a continuous output
- Convert the continuous output to a discrete signal



Zero-order Hold



Δ: Hold interval -- Learnable

Discretized matrix **A** $\quad \bar{A} = \exp(\Delta A)$

Discretized matrix **B** $\quad \bar{B} = (\Delta A)^{-1}(\exp(\Delta A) - I) \cdot \Delta B$

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

To discretize the continuous case, let's use the trapezoid method where the principle is to assimilate the region under the representative curve of a function f defined on a segment $[t_n, t_{n+1}]$ to a trapezoid and calculate its area $T : T = (t_{n+1} - t_n)\frac{f(t_n)+f(t_{n+1})}{2}$.

We then have: $x_{n+1} - x_n = \frac{1}{2}\Delta(f(t_n) + f(t_{n+1}))$ with $\Delta = t_{n+1} - t_n$.
If $x'_n = \mathbf{A}x_n + \mathbf{B}u_n$ (first line of the SSM equation), corresponds to f, so:

$$
\begin{aligned}
x_{n+1} &= x_n + \frac{\Delta}{2}(\mathbf{A}x_n + \mathbf{B}u_n + \mathbf{A}x_{n+1} + \mathbf{B}u_{n+1}) \\
\Longleftrightarrow x_{n+1} - \frac{\Delta}{2}\mathbf{A}x_{n+1} &= x_n + \frac{\Delta}{2}\mathbf{A}x_n + \frac{\Delta}{2}\mathbf{B}(u_{n+1} + u_n) \\
(*) \Longleftrightarrow (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})x_{n+1} &= (\mathbf{I} + \frac{\Delta}{2}\mathbf{A})x_n + \Delta\mathbf{B}u_{n+1} \\
\Longleftrightarrow x_{n+1} &= (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}(\mathbf{I} + \frac{\Delta}{2}\mathbf{A})x_n + (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}\Delta\mathbf{B}u_{n+1}
\end{aligned}
$$

(*) $u_{n+1} \overset{\Delta}{\simeq} u_n$ (the control vector is assumed to be constant over a small Δ).

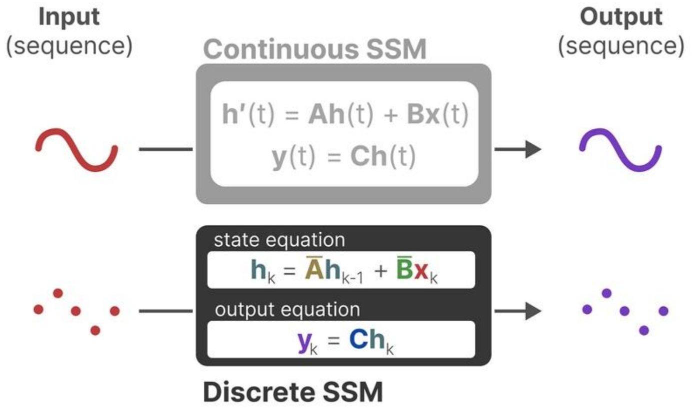
We've just obtained our discretized SSM!

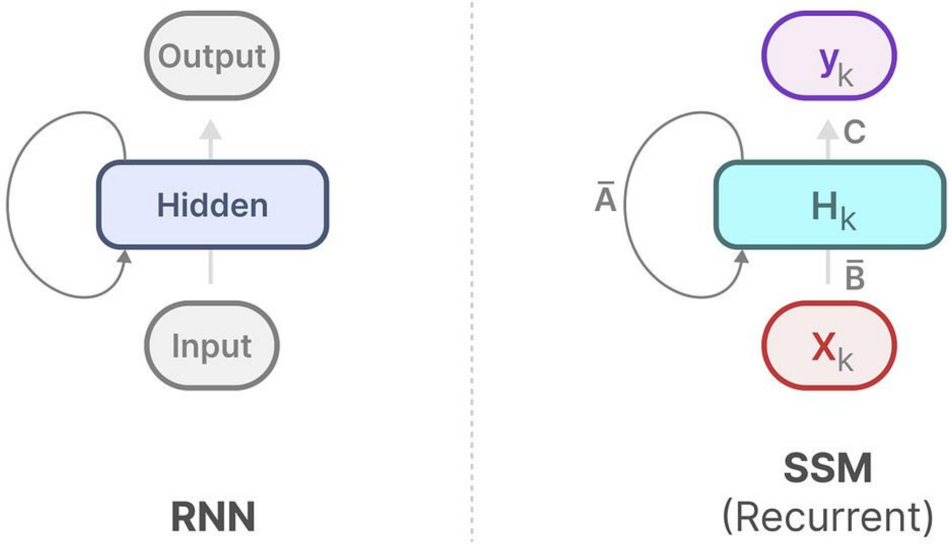
To make this completely explicit, let's pose :

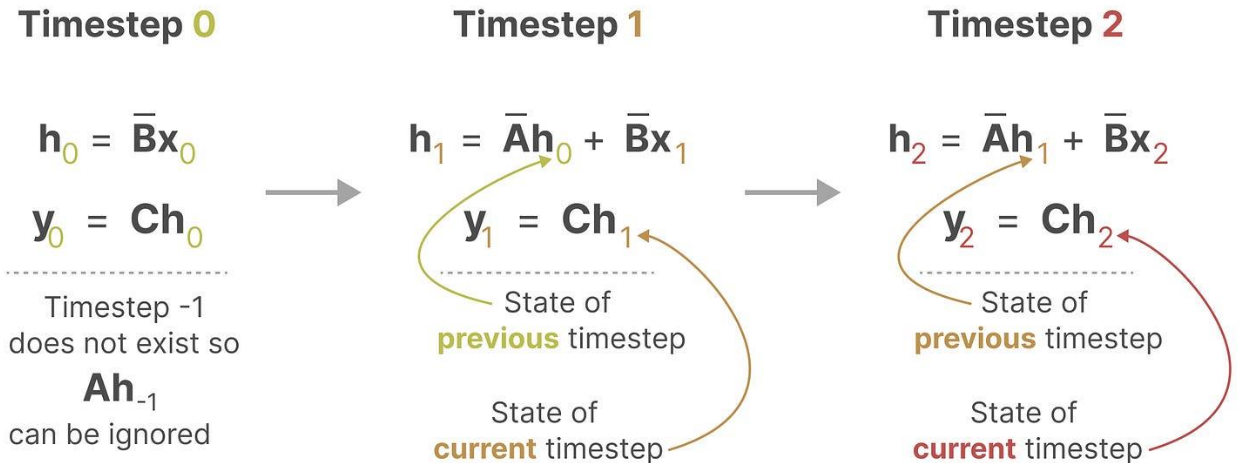
$$
\begin{aligned}
\bar{\mathbf{A}} &= (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}(\mathbf{I} + \frac{\Delta}{2}\mathbf{A}) \\
\bar{\mathbf{B}} &= (\mathbf{I} - \frac{\Delta}{2}\mathbf{A})^{-1}\Delta\mathbf{B} \\
\bar{\mathbf{C}} &= \mathbf{C}
\end{aligned}
$$

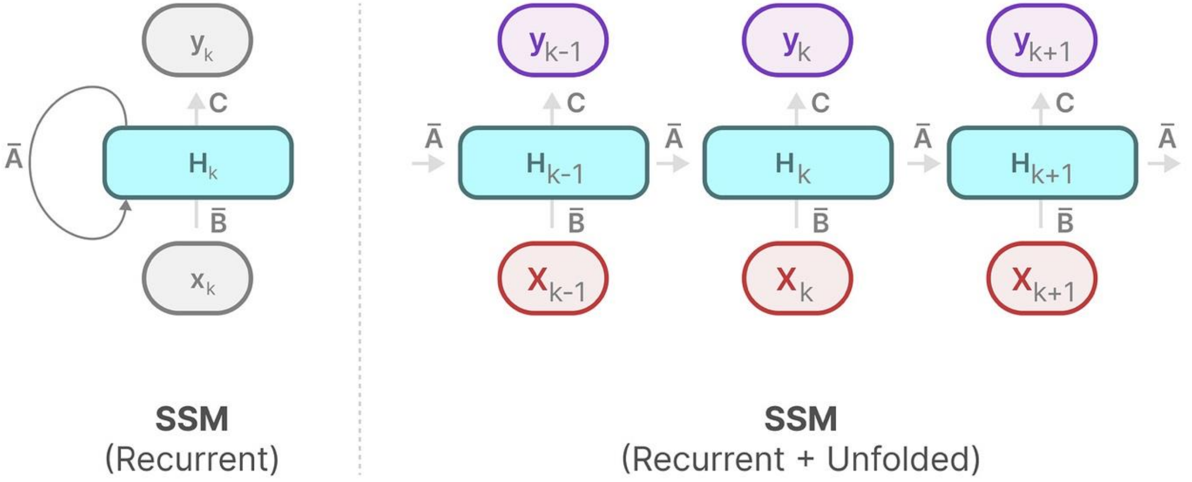
https://huggingface.co/blog/lbourdois/get-on-the-ssm-train

State Space Models (SSMs)



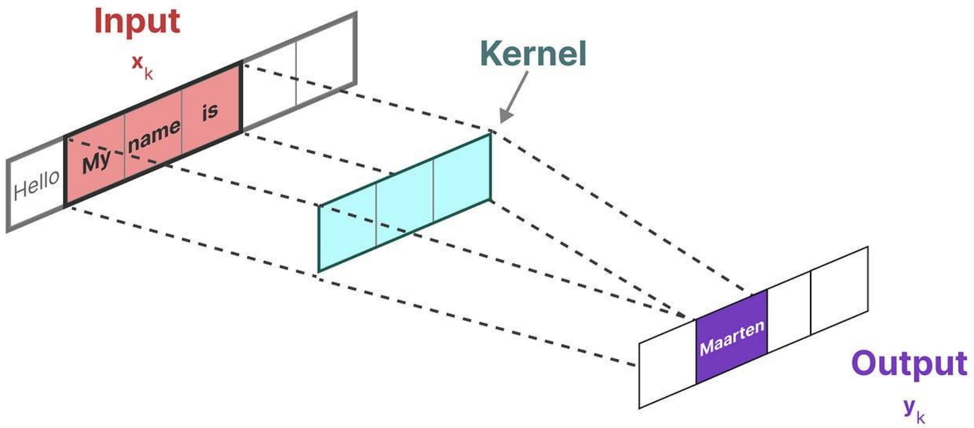


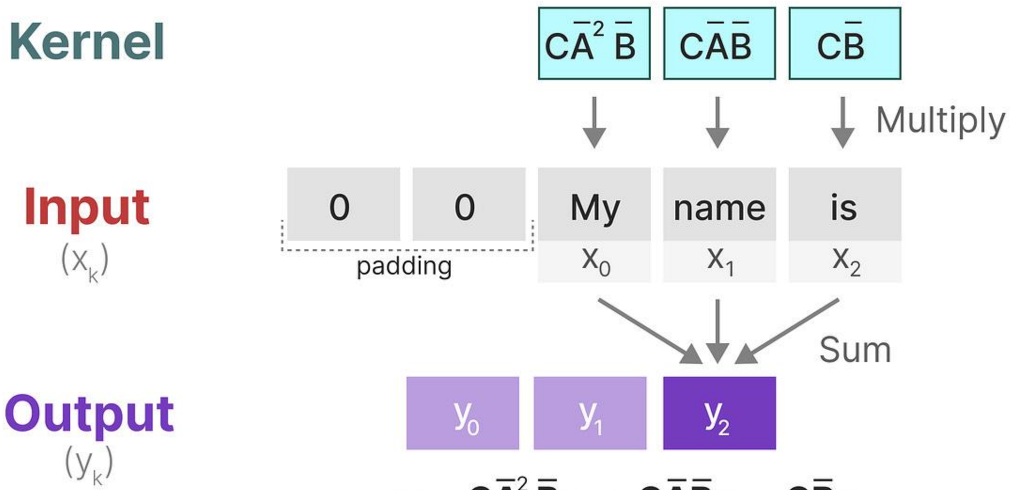




https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models (SSMs)





$$\mathbf{y}_2 = \mathbf{C}\overline{\mathbf{A}}^2\overline{\mathbf{B}}\mathbf{x}_0 + \mathbf{C}\overline{\mathbf{A}}\overline{\mathbf{B}}\mathbf{x}_1 + \mathbf{C}\overline{\mathbf{B}}\mathbf{x}_2$$

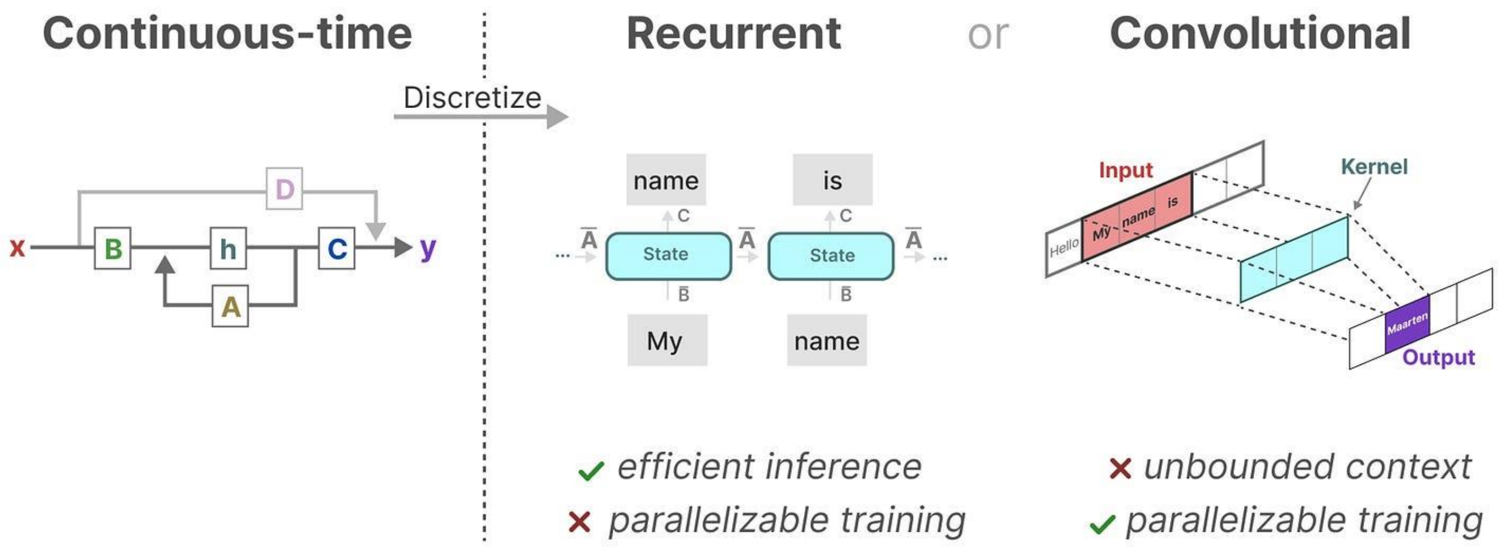
kernel $\longrightarrow$ $\overline{\mathbf{K}} = (\mathbf{C}\overline{\mathbf{B}}, \mathbf{C}\overline{\mathbf{A}}\overline{\mathbf{B}}, \ldots, \mathbf{C}\overline{\mathbf{A}}^k\overline{\mathbf{B}}, \ldots)$

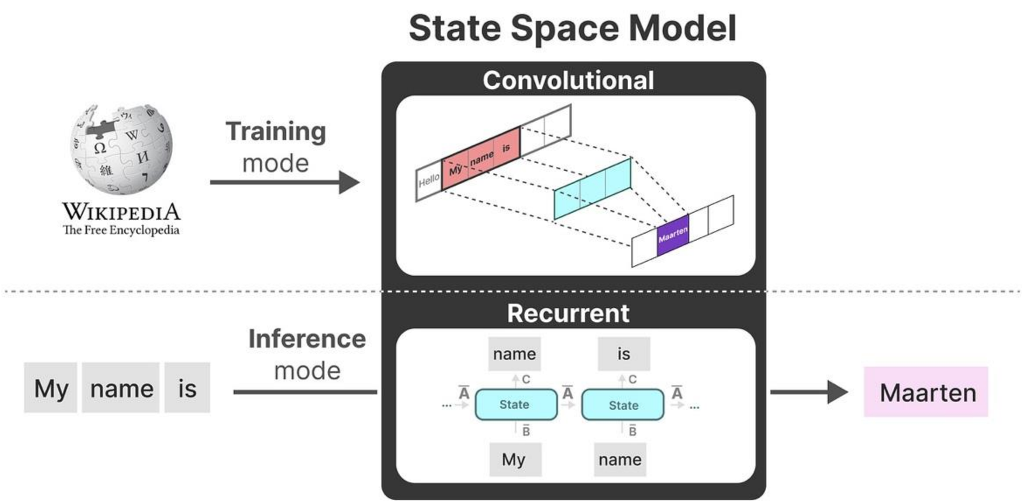
$$\mathbf{y} = \mathbf{x} * \overline{\mathbf{K}}$$

output input kernel

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models (SSMs)





These representations share an important property, namely that of ***Linear Time Invariance*** (LTI). LTI states that the SSMs parameters, *A*, *B*, and *C*, are fixed for all timesteps. This means that matrices *A*, *B*, and *C* are the same for every token the SSM generates.

In other words, regardless of what sequence you give the SSM, the values of *A*, *B*, and *C* remain the same. We have a static representation that is not content-aware.

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

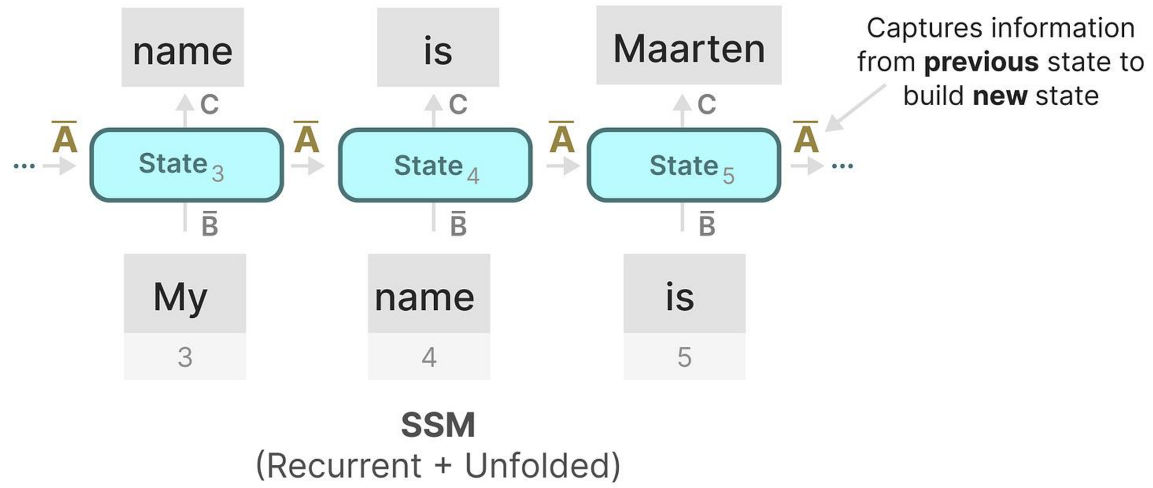
State Space Models (SSMs)

Gu, Albert, et al. "Hippo: Recurrent memory with optimal polynomial projections." Advances in neural information processing systems 33 (2020): 1474-1487.

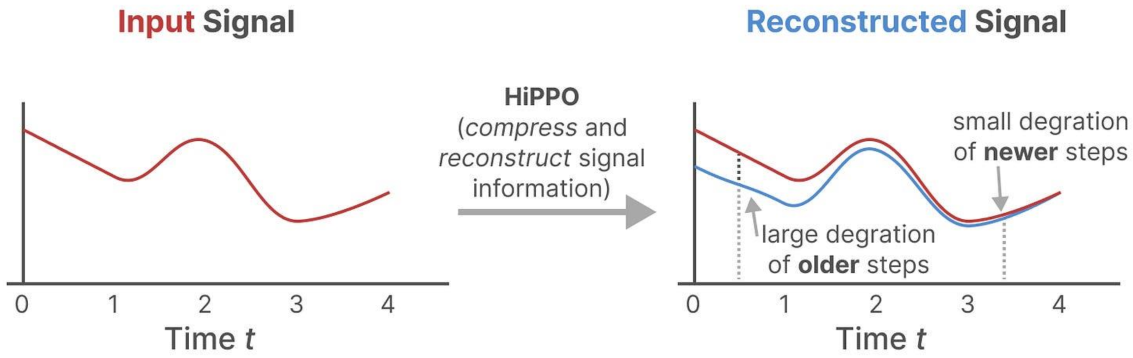
"So how can we create matrix A in a way that retains a large memory (context size)?"

Produces hidden state

$$h_k = \bar{A} h_{k-1} + \bar{B} x_k$$
$$y_k = C h_k$$



High-order Polynomial Projection Operators (HIPPO)



HiPPO Matrix

$$A_{nk} = \begin{cases} (2n+1)^{1/2}(2k+1)^{1/2} & \text{everything below the diagonal} \\ n+1 & \text{the diagonal} \\ 0 & \text{everything above the diagonal} \end{cases}$$

HiPPO Matrix

$$\begin{array}{c|c} & \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 2 & 0 & 0 \\ 1 & 3 & 3 & 0 \\ 1 & 3 & 5 & 4 \end{bmatrix} \; k \\ & n \end{array}$$

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

State Space Models (SSMs)

https://srush.github.io/annotated-s4/

"Prior work found that the basic SSM actually performs very poorly in practice. Intuitively, one explanation is that they suffer from gradients scaling exponentially in the sequence length (i.e., the vanishing/exploding gradients problem)."

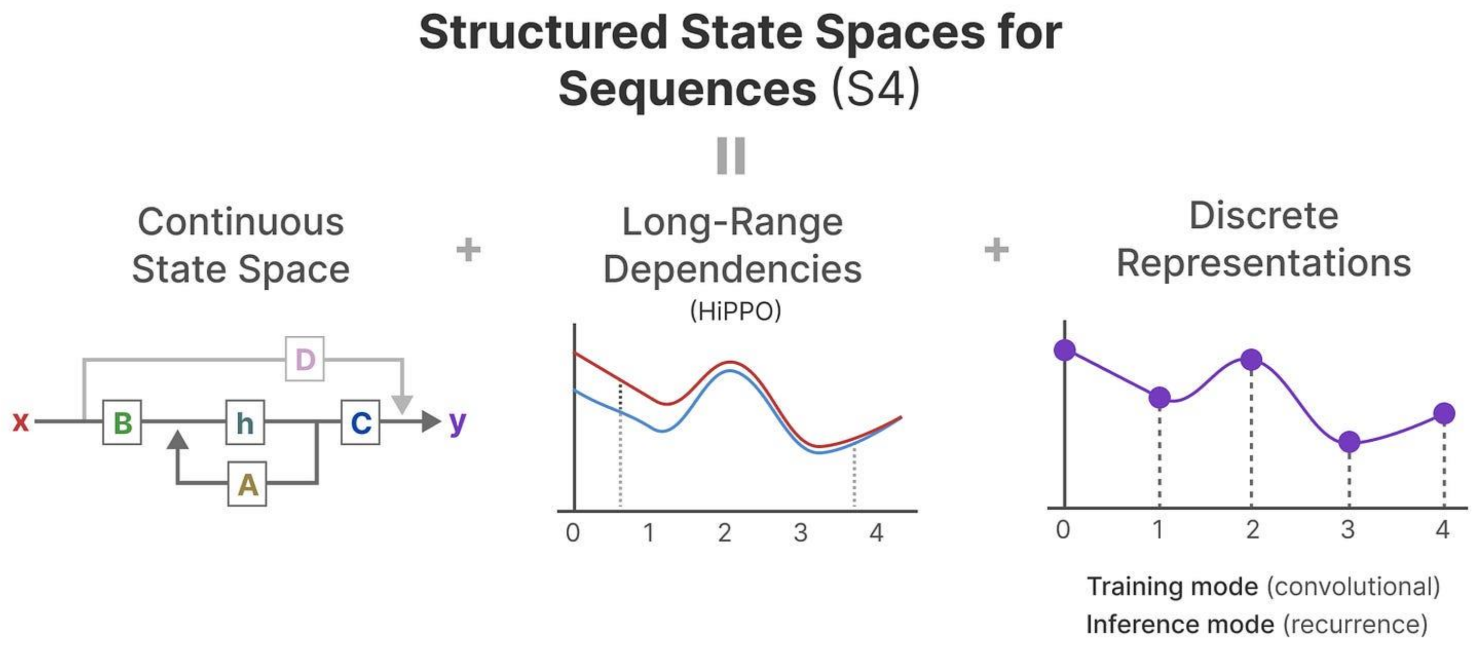
"Previous work found that simply modifying an SSM from a random matrix A to HiPPO improved its performance on the sequential MNIST classification benchmark from 60% to 98%."

"For our purposes we mainly need to know that: 1) we only need to calculate it once, and 2) it has a nice, simple structure (which we will exploit in part 2). Without going into the ODE math, the main takeaway is that this matrix aims to compress the past history into a state that has enough information to approximately reconstruct the history."

"Diving a bit deeper, the intuitive explanation of this matrix is that it produces a hidden state that memorizes its history. It does this by keeping track of the coefficients of a Legendre polynomial. These coefficients let it approximate all of the previous history."

Voelker, Aaron R.; Kajić, Ivana; Eliasmith, Chris (2019). Legendre Memory Units: Continuous-Time Representation in Recurrent Neural Networks (PDF). Advances in Neural Information Processing Systems.

State Space Models (SSMs)



For more on this: https://srush.github.io/annotated-s4/

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Reading material

- Introduction to SSM
  - https://huggingface.co/blog/lbourdois/get-on-the-ssm-train
- A History of SSM Models:
  - https://huggingface.co/blog/lbourdois/ssm-2022
- A Visual Guide to Mamba and SSMs:
  - https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

MAMBA

Mamba: Linear-Time Sequence Modeling with Selective State Spaces

Albert Gu*[1] and Tri Dao*[2]

[1]Machine Learning Department, Carnegie Mellon University

[2]Department of Computer Science, Princeton University
agu@cs.cmu.edu, tri@tridao.me

Rejected at ICLR2024

Abstract

Foundation models, now powering most of the exciting applications in deep learning, are almost universally based on the Transformer architecture and its core attention module. Many subquadratic-time architectures such as linear attention, gated convolution and recurrent models, and structured state space models (SSMs) have been developed to address Transformers' computational inefficiency on long sequences, but they have not performed as well as attention on important modalities such as language. We identify that a key weakness of such models is their inability to perform content-based reasoning, and make several improvements. First, simply letting the SSM parameters be functions of the input addresses their weakness with discrete modalities, allowing the model to *selectively* propagate or forget information along the sequence length dimension depending on the current token. Second, even though this change prevents the use of efficient convolutions, we design a hardware-aware parallel algorithm in recurrent mode. We integrate these selective SSMs into a simplified end-to-end neural network architecture without attention or even MLP blocks (**Mamba**). Mamba enjoys fast inference (5× higher throughput than Transformers) and linear scaling in sequence length, and its performance improves on real data up to million-length sequences. As a general sequence model backbone, Mamba achieves state-of-the-art performance across several modalities such as language, audio, and genomics. On language modeling, our Mamba-3B model outperforms Transformers of the same size and matches Transformers twice its size, both in pretraining and downstream evaluation.

Motivation: Gap in the literature

Foundation models (FMs), or large models pretrained on massive data then adapted for downstream tasks, have emerged as an effective paradigm in modern machine learning. The backbone of these FMs are often *sequence models*, operating on arbitrary sequences of inputs from a wide variety of domains such as language, images, speech, audio, time series, and genomics (Brown et al. 2020; Dosovitskiy et al. 2020; Ismail Fawaz et al. 2019; Oord et al. 2016; Poli et al. 2023; Sutskever, Vinyals, and Quoc V Le 2014). While this concept is agnostic to a particular choice of model architecture, modern FMs are predominantly based on a single type of sequence model: the Transformer (Vaswani et al. 2017) and its core attention layer (Bahdanau, Cho, and Bengio 2015) The efficacy of self-attention is attributed to its ability to route information densely within a context window, allowing it to model complex data. However, this property brings fundamental drawbacks: an inability to model anything outside of a finite window, and quadratic scaling with respect to the window length. An enormous body of research has appeared on more efficient variants of attention to overcome these drawbacks (Tay, Dehghani, Bahri, et al. 2022), but often at the expense of the very properties that makes it effective. As of yet, none of these variants have been shown to be empirically effective at scale across domains.

Motivation: Gap in the literature

Recently, structured state space sequence models (SSMs) (Gu, Goel, and Ré 2022; Gu, Johnson, Goel, et al. 2021) have emerged as a promising class of architectures for sequence modeling. These models can be interpreted as a combination of recurrent neural networks (RNNs) and convolutional neural networks (CNNs), with inspiration from classical state space models (Kalman 1960). This class of models can be computed very efficiently as either a recurrence or convolution, with linear or near-linear scaling in sequence length. Additionally, they have principled mechanisms for modeling long-range dependencies (Gu, Dao, et al. 2020) in certain data modalities, and have dominated benchmarks such as the Long Range

Arena (Tay, Dehghani, Abnar, et al. 2021). Many flavors of SSMs (Gu, Goel, and Ré 2022; Gu, Gupta, et al. 2022; Gupta, Gu, and Berant 2022; Y. Li et al. 2023; Ma et al. 2023; Orvieto et al. 2023; Smith, Warrington, and Linderman 2023) have been successful in domains involving continuous signal data such as audio and vision (Goel et al. 2022; Nguyen, Goel, et al. 2022; Saon, Gupta, and Cui 2023). However, they have been less effective at modeling discrete and information-dense data such as text.

Contributions

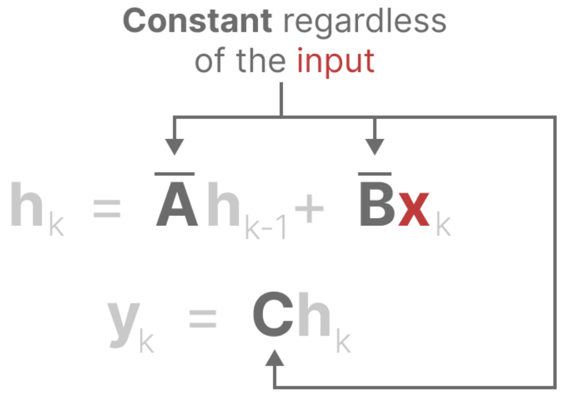
- "Selective Scan" Structured State Space Sequence (S6) Models

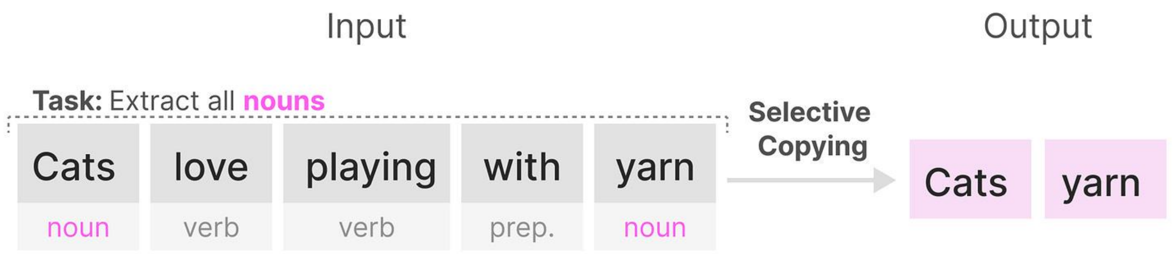
Selection Mechanism. First, we identify a key limitation of prior models: the ability to efficiently *select* data in an input-dependent manner (i.e. focus on or ignore particular inputs). Building on intuition based on important synthetic tasks such as selective copy and induction heads, we design a simple selection mechanism by parameterizing the SSM parameters based on the input. This allows the model to filter out irrelevant information and remember relevant information indefinitely.

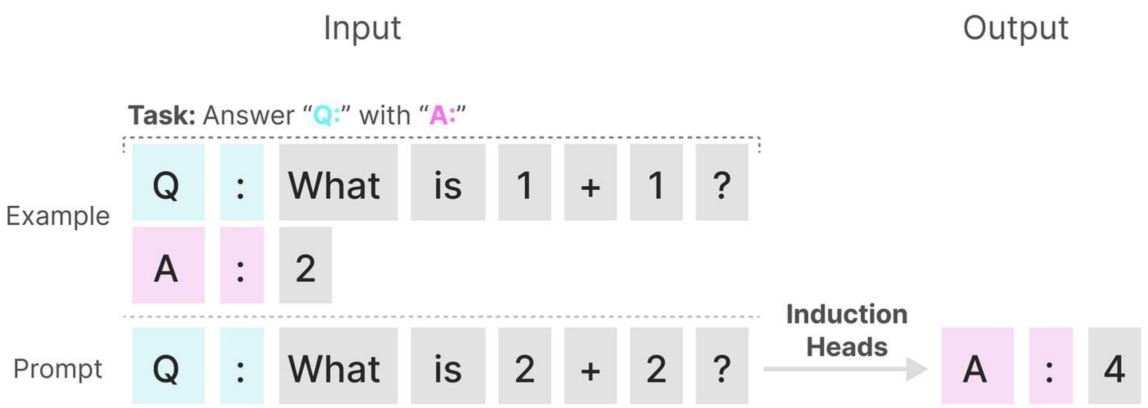
Hardware-aware Algorithm. This simple change poses a technical challenge for the computation of the model; in fact, all prior SSMs models must be time- and input-invariant in order to be computationally efficient. We overcome this with a hardware-aware algorithm that computes the model recurrently with a scan instead of convolution, but does not materialize the expanded state in order to avoid IO access between different levels of the GPU memory hierarchy. The resulting implementation is faster than previous methods both in theory (scaling linearly in sequence length, compared to pseudo-linear for all convolution-based SSMs) and on modern hardware (up to 3× faster on A100 GPUs).

Architecture. We simplify prior deep sequence model architectures by combining the design of prior SSM architectures (Dao, Fu, Saab, et al. 2023) with the MLP block of Transformers into a single block, leading to a simple and homogenous architecture design (**Mamba**) incorporating selective state spaces.

Tasks that are challenging for S4





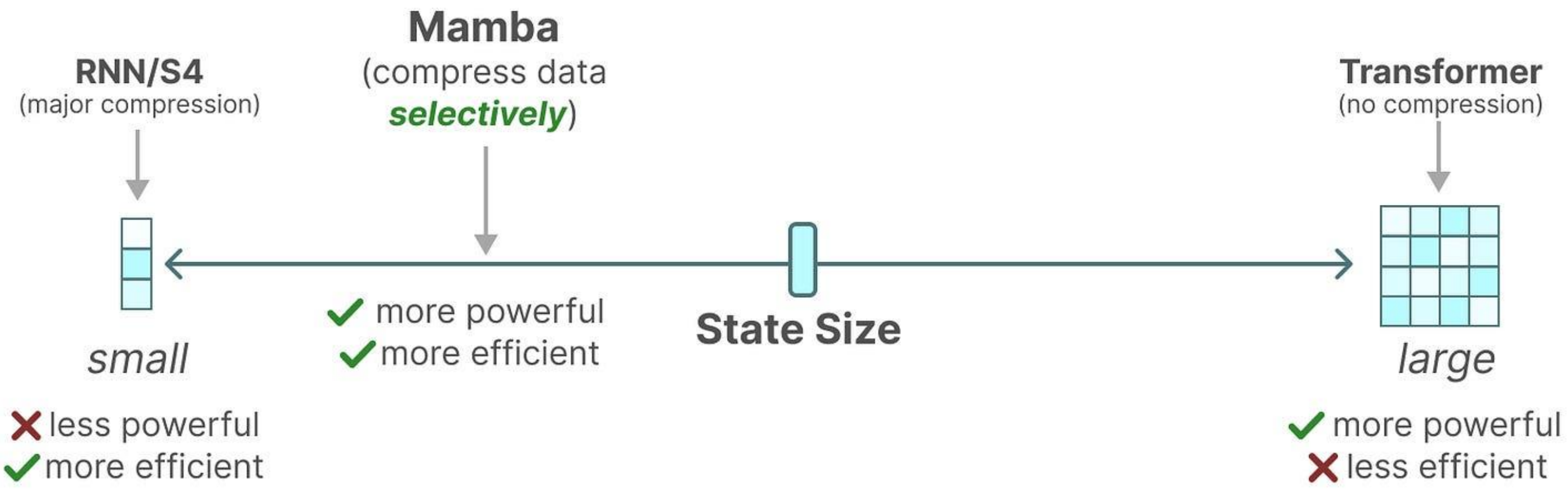


"However, a (recurrent/convolutional) SSM performs poorly in this task since it is Linear Time Invariant. As we saw before, the matrices A, B, and C are the same for every token the SSM generates."

"In the above example, we are essentially performing one-shot prompting where we attempt to "teach" the model to provide an "A:" response after every "Q:". However, since SSMs are time-invariant it cannot select which previous tokens to recall from its history."

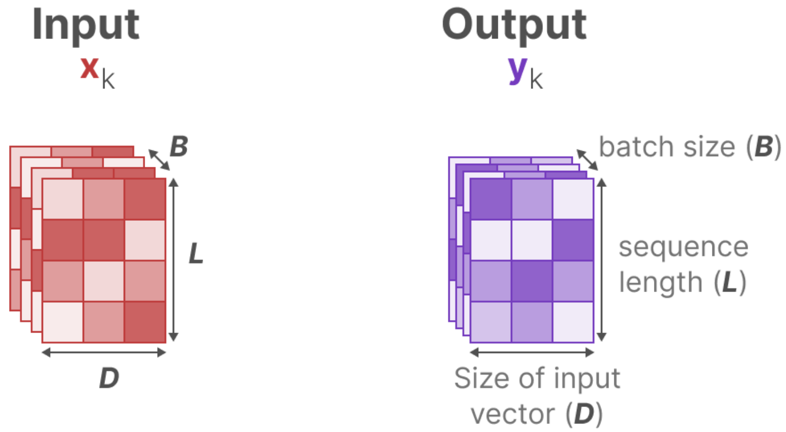
https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Mamba

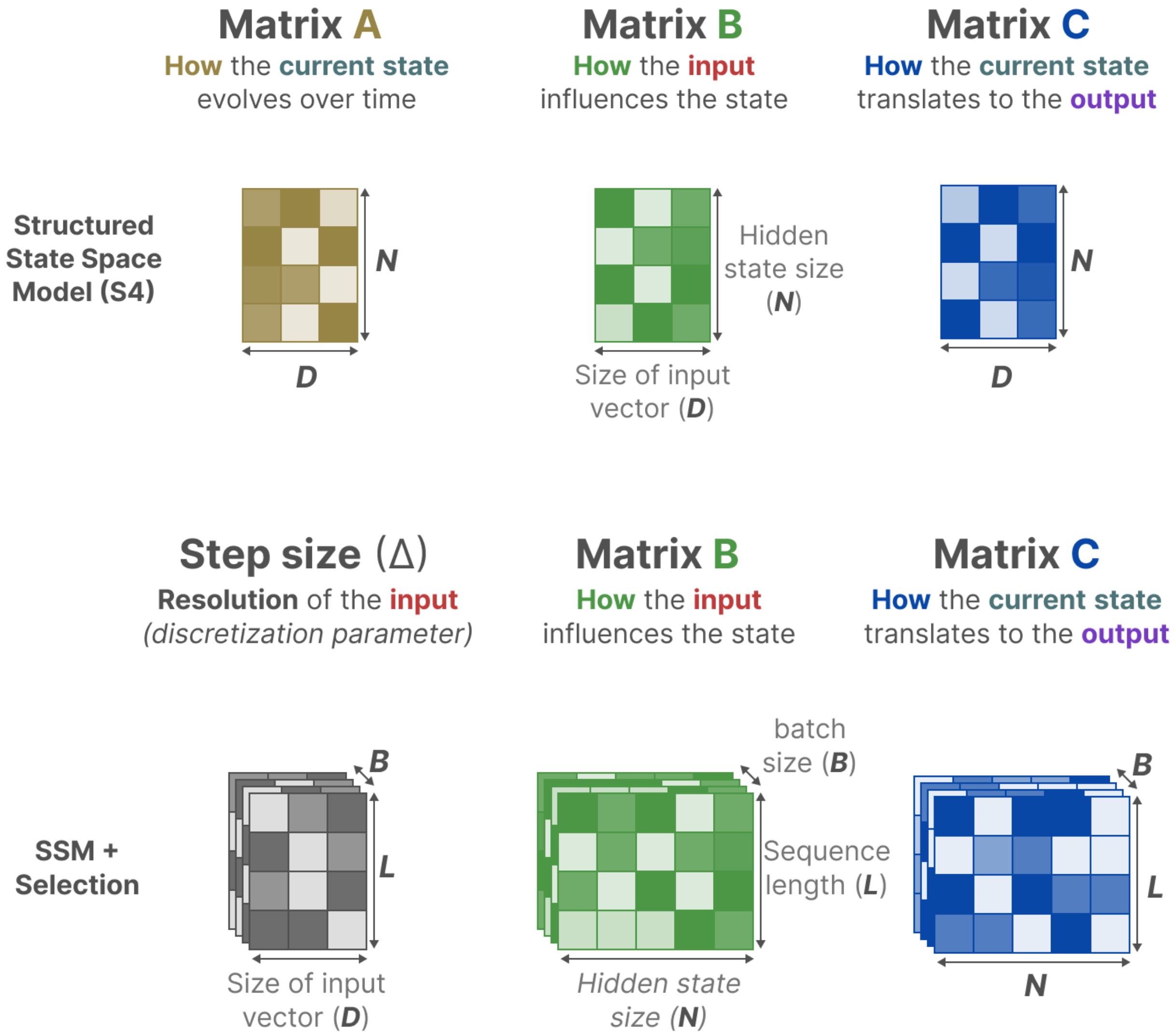


https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Mamba



In Mamba, the matrices are different for each time step:



https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

Mamba

Algorithm 1 SSM (S4)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

1: $A : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$ ▷ Represents structured $N \times N$ matrix

2: $B : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$

3: $C : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$

4: $\Delta : (\mathtt{D}) \leftarrow \tau_\Delta(\mathsf{Parameter})$

5: $\overline{A}, \overline{B} : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{discretize}(\Delta, A, B)$

6: $y \leftarrow \mathsf{SSM}(\overline{A}, \overline{B}, C)(x)$ ▷ Time-invariant: recurrence or convolution

7: **return** y

Algorithm 2 SSM + Selection (S6)

Input: $x : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

Output: $y : (\mathtt{B}, \mathtt{L}, \mathtt{D})$

1: $A : (\mathtt{D}, \mathtt{N}) \leftarrow \mathsf{Parameter}$ ▷ Represents structured $N \times N$ matrix

2: $B : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_B(x)$

3: $C : (\mathtt{B}, \mathtt{L}, \mathtt{N}) \leftarrow s_C(x)$

4: $\Delta : (\mathtt{B}, \mathtt{L}, \mathtt{D}) \leftarrow \tau_\Delta(\mathsf{Parameter} + s_\Delta(x))$

5: $\overline{A}, \overline{B} : (\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N}) \leftarrow \mathsf{discretize}(\Delta, A, B)$

6: $y \leftarrow \mathsf{SSM}(\overline{A}, \overline{B}, C)(x)$ ▷ Time-varying: recurrence (*scan*) only

7: **return** y

We specifically choose $s_B(x) = \mathsf{Linear}_N(x)$, $s_C(x) = \mathsf{Linear}_N(x)$, $s_\Delta(x) = \mathsf{Broadcast}_D(\mathsf{Linear}_1(x))$, and $\tau_\Delta = \mathsf{softplus}$, where Linear_d is a parameterized projection to dimension d. The choice of s_Δ and τ_Δ is due to a connection to RNN gating mechanisms explained in Section 3.5.

Mamba

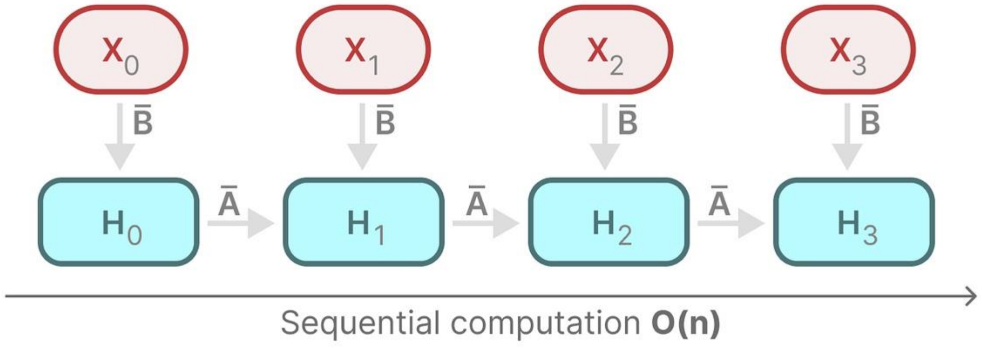
Concretely, instead of preparing the scan input $(\overline{A}, \overline{B})$ of size $(\mathtt{B}, \mathtt{L}, \mathtt{D}, \mathtt{N})$ in GPU HBM (high-bandwidth memory), we load the SSM parameters (Δ, A, B, C) directly from slow HBM to fast SRAM, perform the discretization and recurrence in SRAM, and then write the final outputs of size $(\mathtt{B}, \mathtt{L}, \mathtt{D})$ back to HBM.

To avoid the sequential recurrence, we observe that despite not being linear it can still be parallelized with a work-efficient parallel scan algorithm (Blelloch 1990; Martin and Cundy 2018; Smith, Warrington, and Linderman 2023).

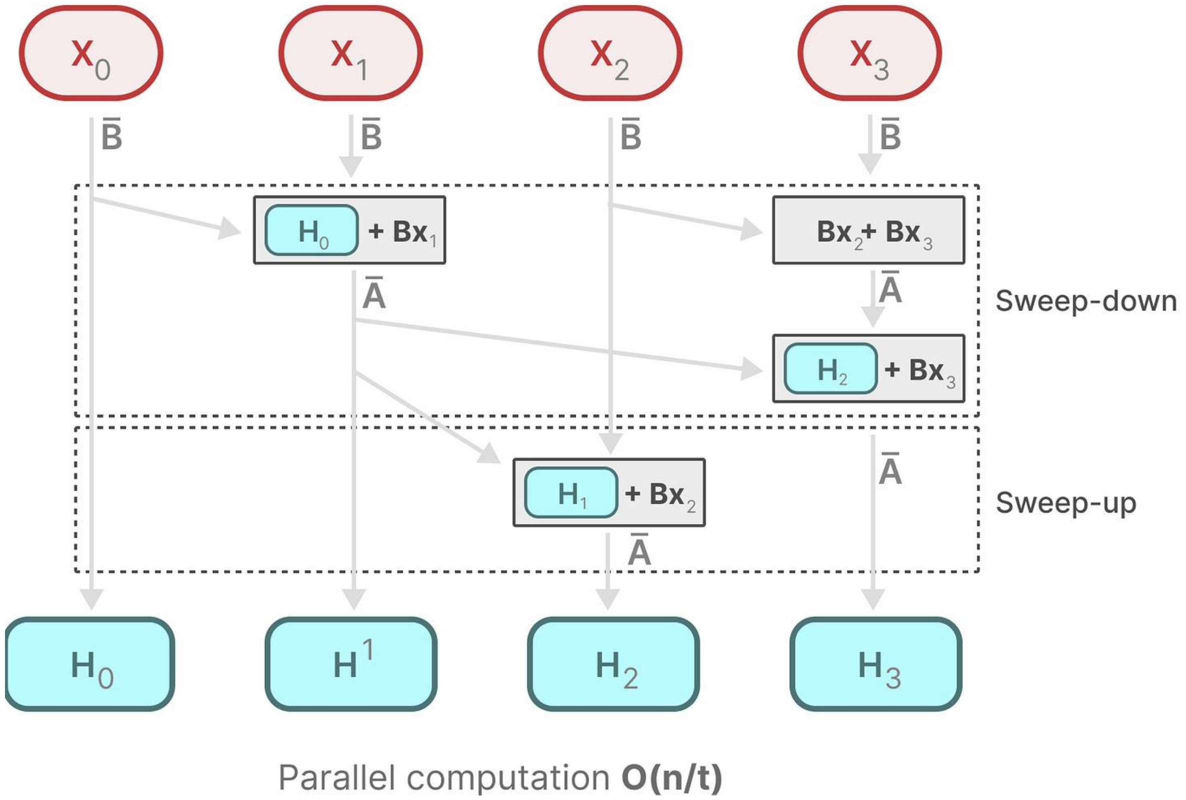
Finally, we must also avoid saving the intermediate states, which are necessary for backpropagation. We carefully apply the classic technique of recomputation to reduce the memory requirements: the intermediate states are not stored but recomputed in the backward pass when the inputs are loaded from HBM to SRAM. As a result, the fused selective scan layer has the same memory requirements as an optimized transformer implementation with FlashAttention.

Mamba

Sequential scan –
Not suitable for parallelization



Parallel scan



"Together, dynamic matrices B and C, and the parallel scan algorithm create the selective scan algorithm to represent the dynamic and fast nature of using the recurrent representation."

https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state

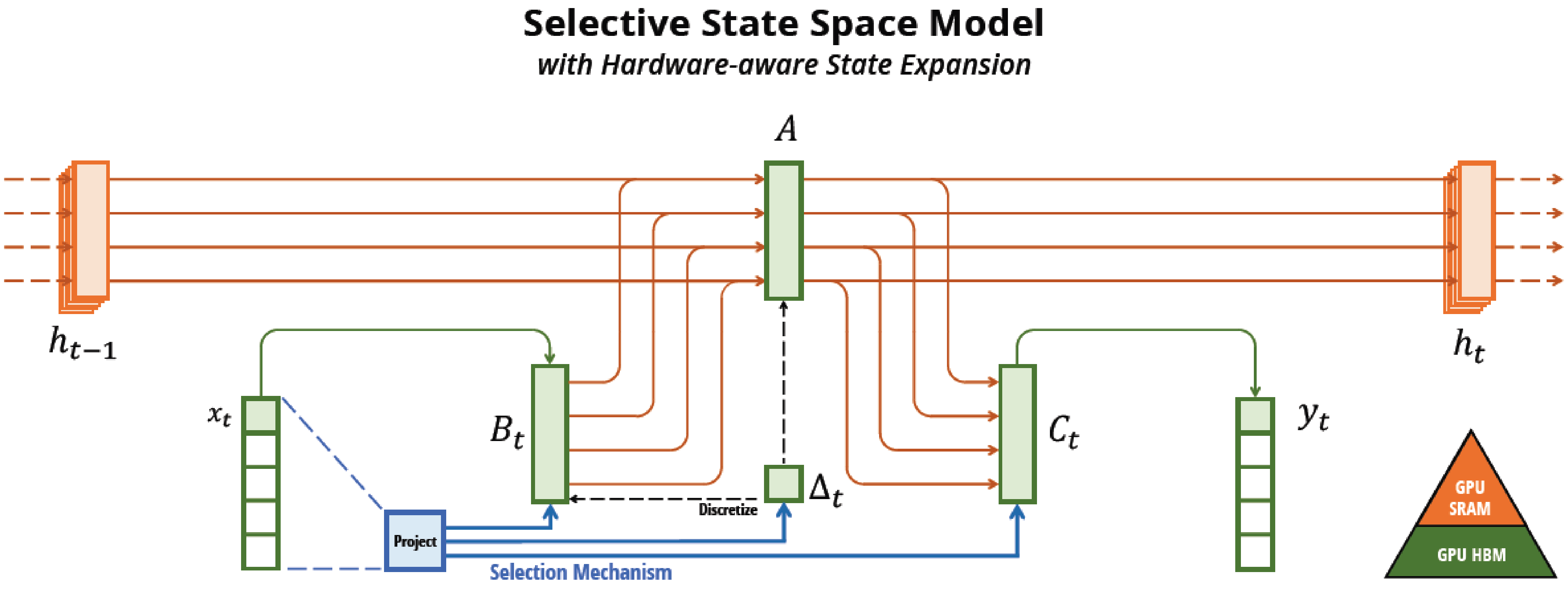


Figure 1: (**Overview.**) Structured SSMs independently map each channel (e.g. $D = 5$) of an input x to output y through a higher dimensional latent state h (e.g. $N = 4$). Prior SSMs avoid materializing this large effective state (DN, times batch size B and sequence length L) through clever alternate computation paths requiring time-invariance: the $(\Delta, \boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C})$ parameters are constant across time. Our selection mechanism adds back input-dependent dynamics, which also requires a careful hardware-aware algorithm to only materialize the expanded states in more efficient levels of the GPU memory hierarchy.

Mamba block

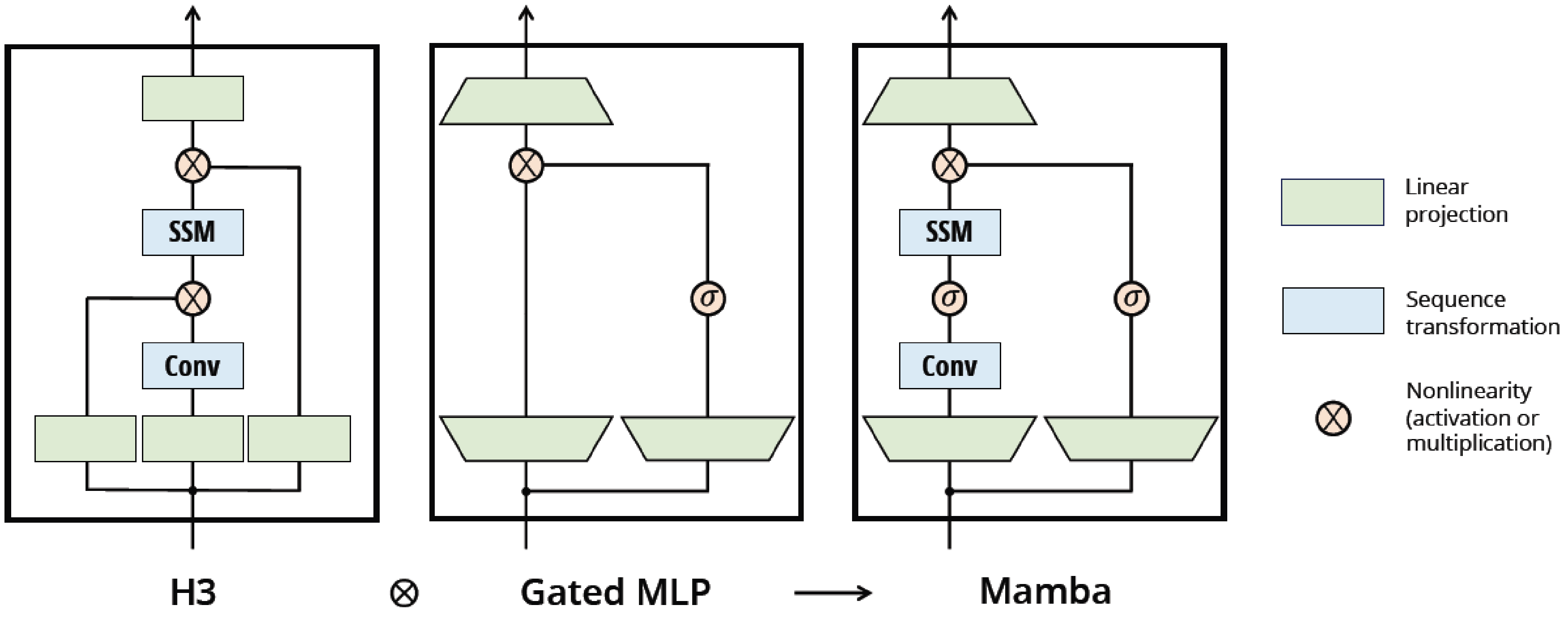


Figure 3: (**Architecture**.) Our simplified block design combines the H3 block, which is the basis of most SSM architectures, with the ubiquitous MLP block of modern neural networks. Instead of interleaving these two blocks, we simply repeat the Mamba block homogenously. Compared to the H3 block, Mamba replaces the first multiplicative gate with an activation function. Compared to the MLP block, Mamba adds an SSM to the main branch. For σ we use the SiLU / Swish activation (Hendrycks and Gimpel 2016; Ramachandran, Zoph, and Quoc V Le 2017).

	Training	Inference
Transformers	**Fast!** (parallelizable)	**Slow...** (scales **quadratically** with sequence length)
RNNs	**Slow...** (not parallelizable)	**Fast!** (scales **linearly** with sequence length)
Mamba	**Fast!** (parallelizable)	**Fast!** (scales **linearly** with sequence length + **unbounded** context)