Sequential data

1. Time series
   - Financial data analysis: stock market, commodities, Forex
   - Healthcare: pulse rate, sugar level (from medical equipment and wearables)
2. Text and speech: speech understanding, text generation
3. Spatiotemporal data
   - Self-driving and object tracking
   - Plate tectonic activity
4. Physics: jet identification
5. etc.

Sequence modelling I

Sequence classification

1. $\boldsymbol{x} = x_1, x_2, \ldots, x_n,\ x_i \in V$ - objects
2. $y \in \{1, \ldots, L\}$ - labels
3. $\{(\boldsymbol{x}^{(1)}, y_1), (\boldsymbol{x}^{(2)}, y_2), \ldots, (\boldsymbol{x}^{(m)}, y_m)\}$ – training data

Classification problem: $\gamma : \boldsymbol{x} \rightarrow y$

1. Activity recognition: x – pulse rate, y – activity (walking, running, peace)
2. Opinion mining: x – sentence, y – sentiment (positive, negative)
3. Trading: x – stock market, y – action (sell, buy, do nothing)

Sequence modelling II

Sequence labelling

1. $\boldsymbol{x} = x_1, x_2, \dots, x_n,\ x_i \in V$ - objects
2. $\boldsymbol{y} = y_1, y_2, \dots, y_n,\ y_i \in \{1, \dots, L\}$ - labels
3. $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \dots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$ – training data
4. exponential number of possible solutions : if $\texttt{length}(x) = n$, there are L^n possible solutions

Classification problem: $\gamma : \boldsymbol{x} \to \boldsymbol{y}$

1. Part of speech tagging: x – word, y – part of speech (verb, noun, etc.)
2. Genome annotation: x – DNA, y – genes
3. HEP tracking: x - a set of hits with backgrounds, y – hit classification

Sequence labelling tasks

POS tagging and Named Entity Recognition

X (words)	the	cat	sat	on	a	mat
Y (tags)	DET	NOUN	VERB	PREP	DET	NOUN

Table: POS tagging

Alex	is	going	to	Los	Angeles
B-PER	O	O	O	B-LOC	I-LOC

Table: NER (IOB2)

Alex	travels	with	Marty	A.	Rick	to	NY	city
S-PER	O	O	B-PER	I-PER	E-PER	O	B-LOC	E-LOC

Table: NER (IOBES)

Sequence modelling III

Sequence transduction / transformation

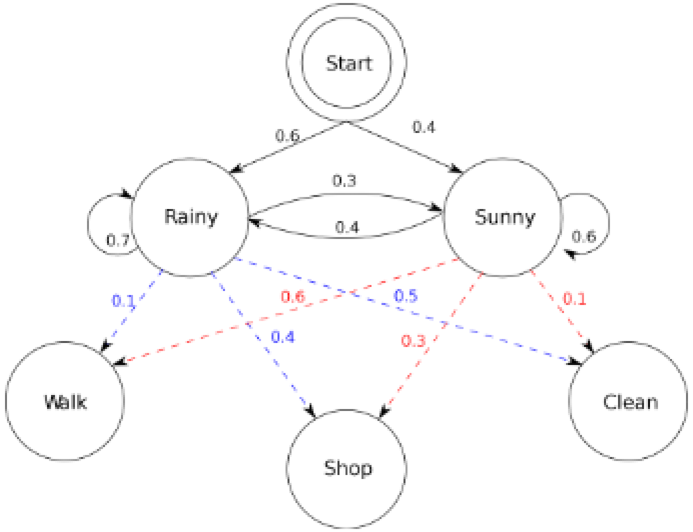
1. $\boldsymbol{x} = x_1, x_2, \dots, x_n,\ x_i \in V_{source}$ - objects
2. $\boldsymbol{y} = y_1, y_2, \dots, y_n,\ y_i \in V_{target}$ - objects
3. $\{(\boldsymbol{x}^{(1)}, \boldsymbol{y}^{(1)}), (\boldsymbol{x}^{(2)}, \boldsymbol{y}^{(2)}), \dots, (\boldsymbol{x}^{(m)}, \boldsymbol{y}^{(m)})\}$ – training data
4. $\boldsymbol{x}^{(1)}$, $\boldsymbol{y}^{(1)}$ are of different length

Transduction problem: $\boldsymbol{x}_{source} \rightarrow \boldsymbol{y}_{target}$

1. Machine translation: x – sentence in German, y – sentence in English
2. Speech recognition: x – spoken language, y – text
3. Chat bots: x – question, y – answer

Traditional ML approaches to sequence modeling

- Hidden Markov Models (HMM)
- Conditional Random Fields (CRF)
- Local classifier: for each x define features, based on x_{-1}, x_{+1}, etc, and perform classification n times



Problems:

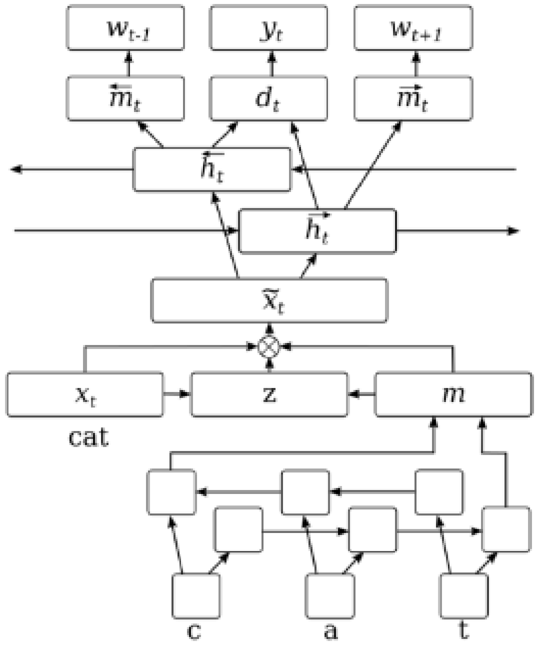
1. Markov assumption: fixed length history
2. Computation complexity

DL approaches to sequence modeling

- Neural networks
- Recurrent neural network and its modifications: LSTM, GRU, Highway
- 2D Convolutional Neural Network
- Transformer
- Pointer network

Problems:

1. Training time
2. Amount of training data



Neural language model

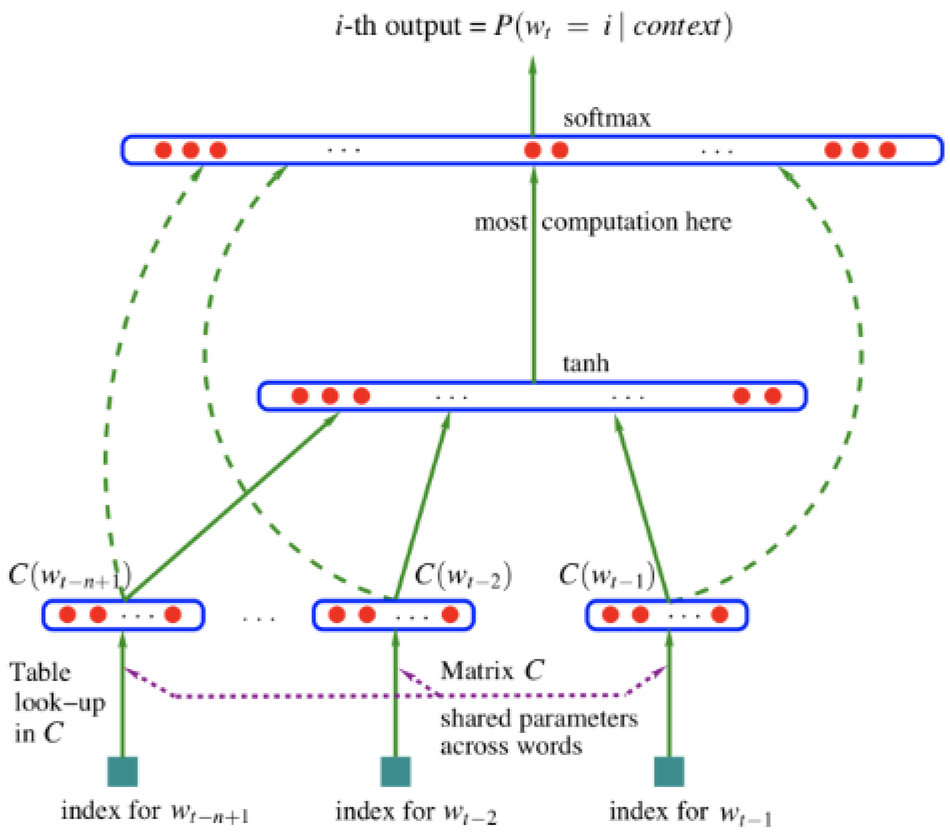
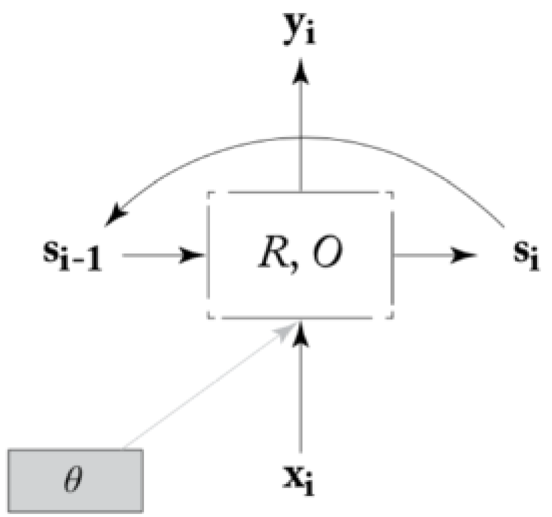


Figure: Neural language model

Recurrent neural network

- Input: sequence of vectors
- $x_{1:n} = x_1, x_2, \ldots, x_n, \; x_i \in \mathbb{R}^{d_{in}}$
- Output: a single vector
  $y_n = RNN(x_{1:n}), \; y_n \in \mathbb{R}^{d_{out}}$
- For each prefix $x_{i:j}$ define an output vector y_i:
  $y_i = RNN(x_{1:i})$
- RNN^* is a function returning this sequence for input sequence $x_{1:n}$:
  $y_{1:n} = RNN^*(x_{1:n}), \; y_i \in \mathbb{R}^{d_{out}}$



Sequence modelling with RNN

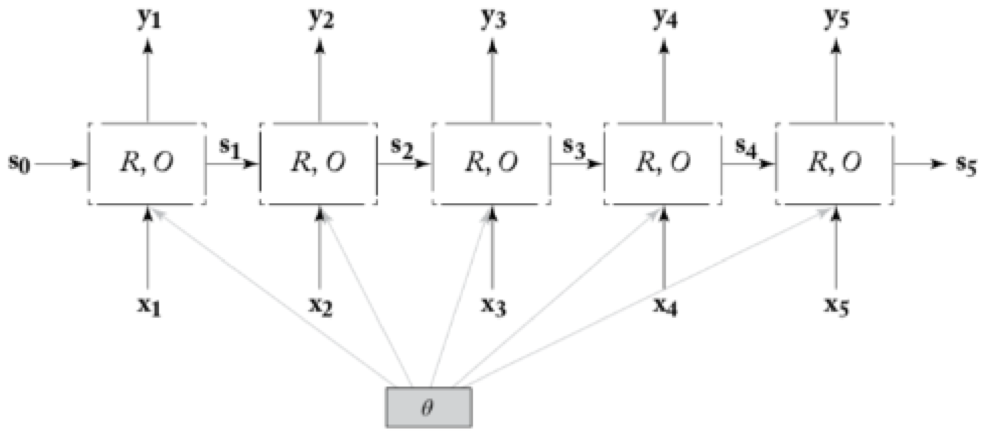
1. Sequence classification
   Put a dense layer on top of RNN to predict the desired class of the sequence after the whole sequence is processed

$$p(l_j|\boldsymbol{x}_{1:n}) = \texttt{softmax}(RNN(\boldsymbol{x}_{1:n}) \times W + b)_{[j]}$$

2. Sequence labelling
   Produce an output y_i for each input RNN reads in. Put a dense layer on top of each output to predict the desired class of the input

$$p(l_j|\boldsymbol{x}_j) = \texttt{softmax}(RNN(\boldsymbol{x}_{1:j}) \times W + b)_{[j]}$$

RNN unrolled

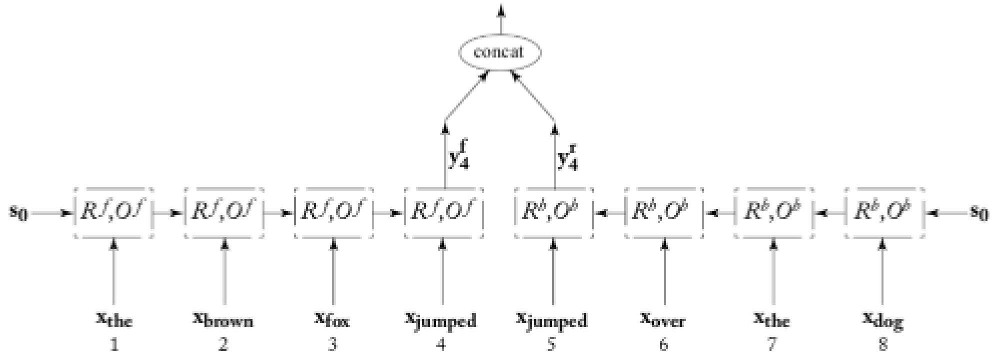


$$s_4 = R(s_3, x_4) = R(R(s_2, x_3), x_4) = R(R(R(s_1, x_2), x_3), x_4) =$$

$$= R(R(R(R(s_0, x1), x_2), x_3), x_4)$$

Bidirectional RNN (Bi-RNN)

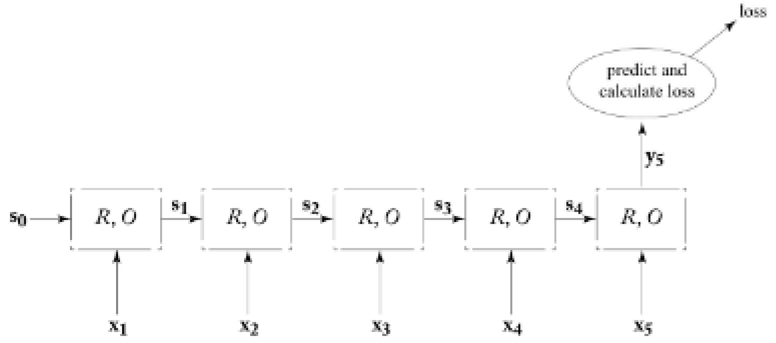
The input sequence can be read from left to right and from right to left.
Which direction is better?



$$biRNN(x_{1:n}, i) = y_i = [RNN^f(x_{1:i}); RNN^r(x_{n:i})]$$

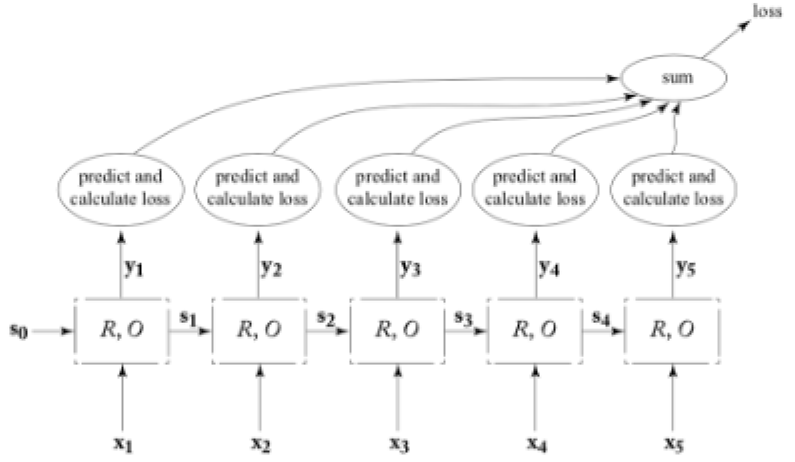
Sequence classification

- $\hat{y}_n = O(s_n)$
- $\texttt{prediction} = MLP(\hat{y}_n)$
- Loss: $L(\hat{y}_n, y_n)$
- L can take any form: cross entropy, hinge, margin, etc.

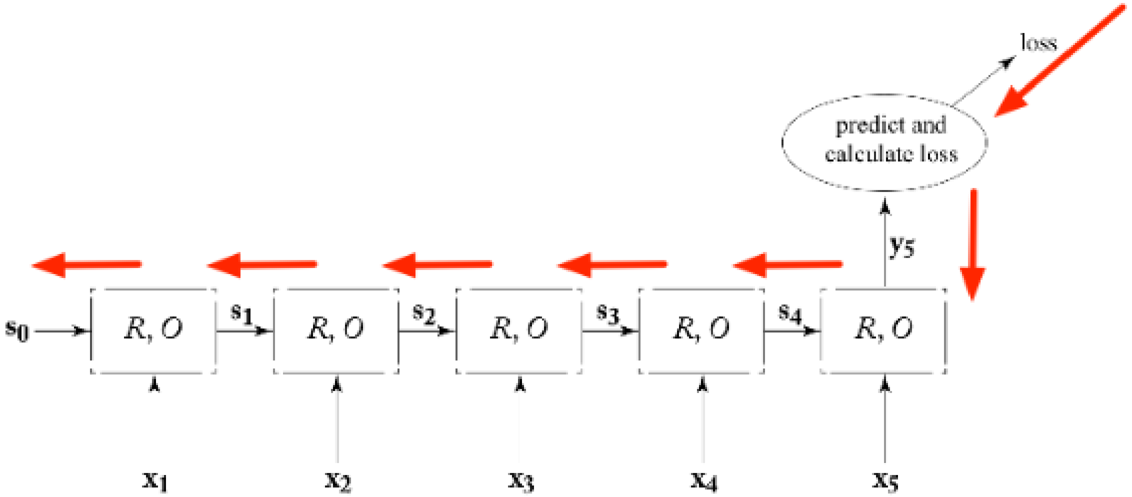


Sequence labelling

- Output $\hat{t}_i$ for each input $x_{1,i}$
- Local loss: $L_{local}(\hat{t}_i, t_i)$
- Global loss: $L(\hat{t}_n, t_n) = \sum_i L_{local}(\hat{t}_i, t_i)$
- L can take any form: cross entropy, hinge, margin, etc.



Backpropagation through time



$s_i = R(x_i, s_{i-1}) = g(s_{i-1}W^s + x_i W^x + b)$

Chain rule: $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial p(\hat{y}_5)} \frac{\partial p(\hat{y}_5)}{\partial s_4} (\frac{\partial s_4}{\partial w} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial w} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_w} + \ldots)$

Vanishing gradient problem

Chain rule: $\frac{\partial L}{\partial w} = \frac{\partial L}{\partial p(\hat{y_5})} \frac{\partial p(\hat{y_5})}{\partial s_4} (\frac{\partial s_4}{\partial w} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial w} + \frac{\partial s_4}{\partial s_3} \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_w} + \ldots)$
g – sigmoid

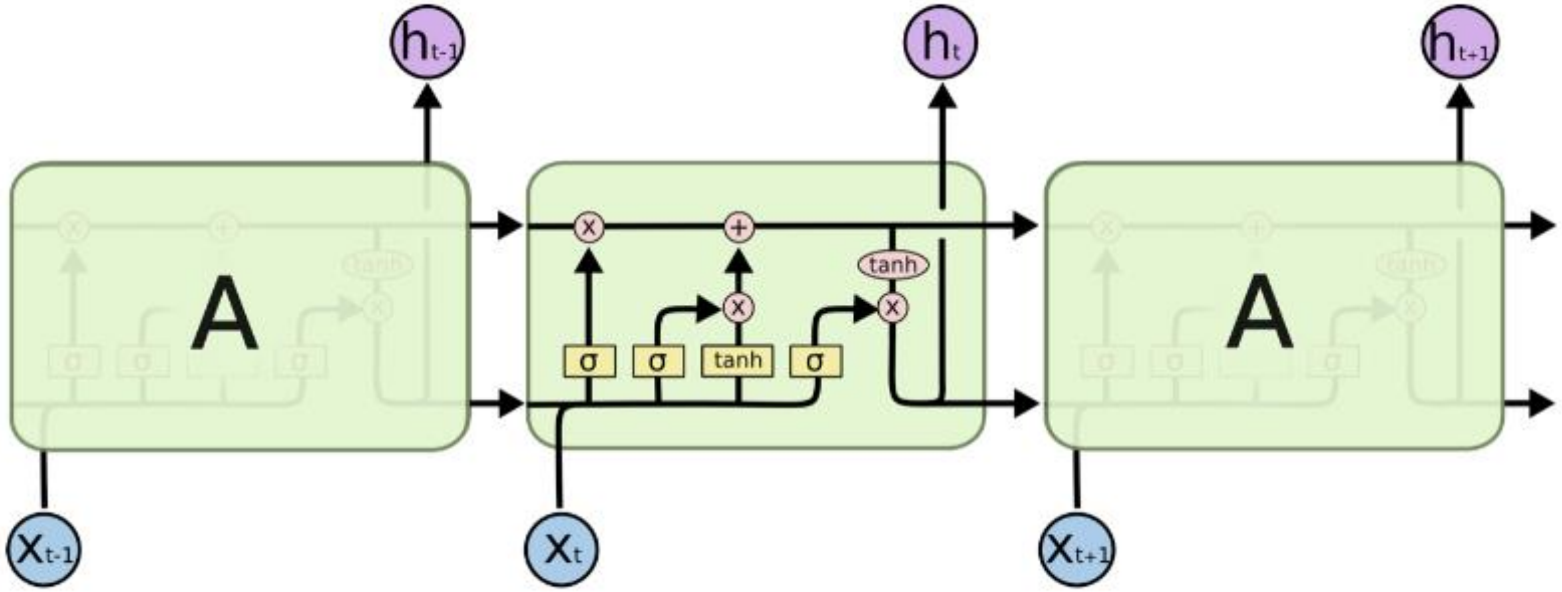
1. Many sigmoids near 0 and 1
   - Gradients $\rightarrow 0$
   - Not training for long term dependencies
2. Many sigmoids > 1
   - Gradients $\rightarrow +\inf$
   - Not training again

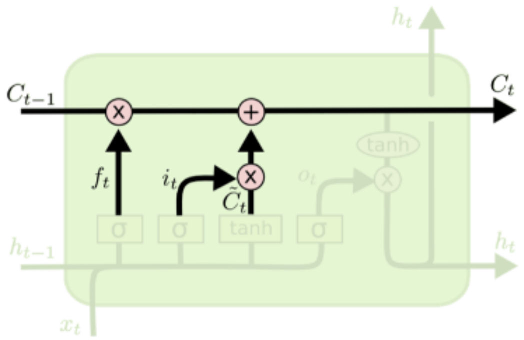
Solution: gated architectures (LSTM and GRU)

Controlled memory access

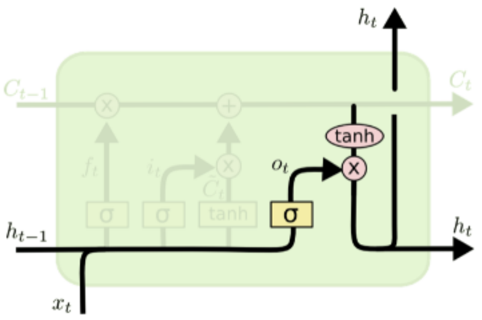
- Entire memory vector is changed: $s_{i+1} = R(x_i, s_i)$
- Controlled memory access: $s_{i+1} = g \odot R(x_i, s_i) + (1 - g)s_i$
  $g \in [0, 1]^d, s, x \in \mathbb{R}^d$
- Differential gates: $\sigma(g), g' \in \mathbb{R}^d$
- This controllable gating mechanism is the basis of the LSTM and the GRU architectures

Long short term memory

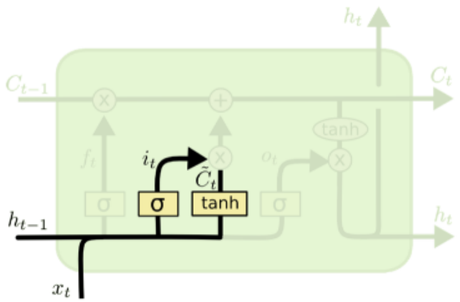




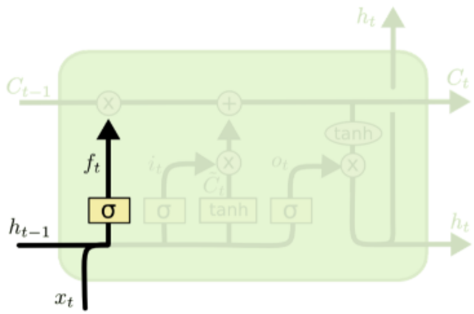
$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

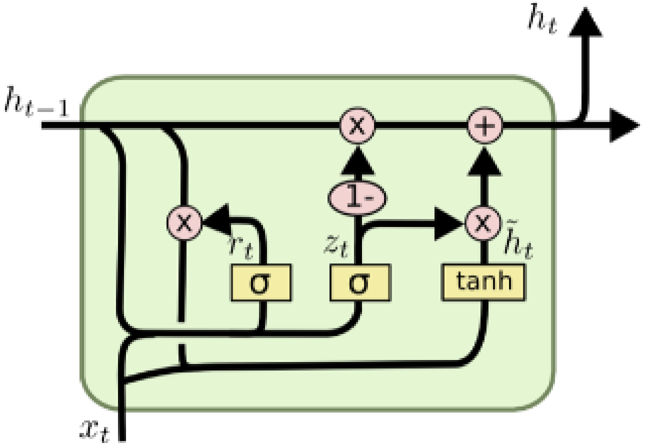


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t]\ +\ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t]\ +\ b_C)$$



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t]\ +\ b_f\right)$$

Gated recurrent unit



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$

$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$

$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$

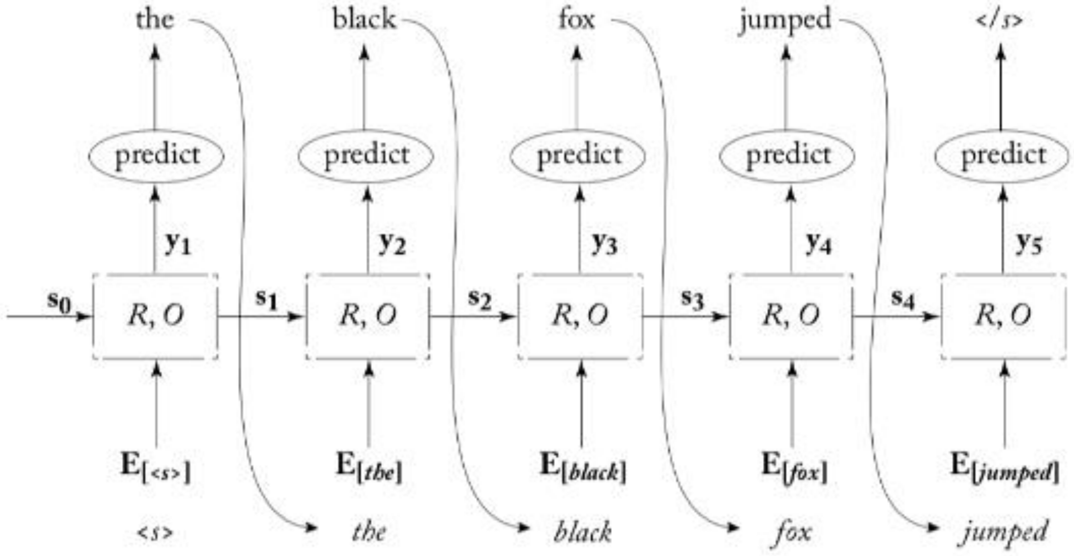
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Sequence generation

Teacher forcing: $x := <s> x, y := x </s>$

$x : <s> x_1 x_2 \dots x_n$

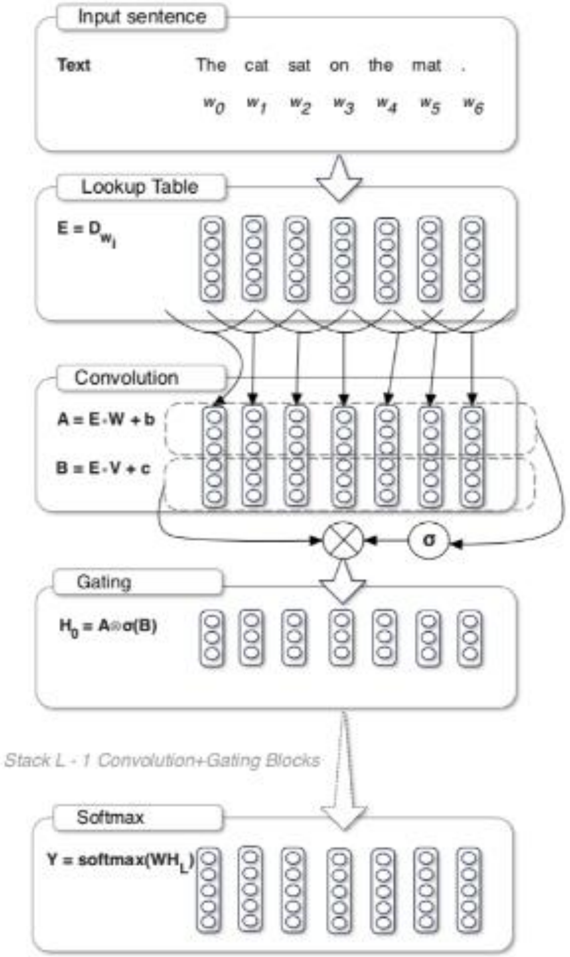
$y : x_1 x_2 \dots x_n </s>$



Pros and cons of RNNs

1. Advantages:
   - RNNs are popular and successful for variable-length sequences
   - The gating models such as LSTM are suited for long-range error propagation
2. Problems:
   - The sequentiality prohibits parallelization within instances
   - Long-range dependencies still tricky, despite gating

Language Modeling with Gated Convolutional Networks



- **Embeddings** $\in D^{|V|\times e}$
- **Input:** $w_0, \ldots, w_n \to E = [D_{w_0}, \ldots, D_{w_n}]$
- **Hidden layers**: $h_0, \ldots, h_n$:

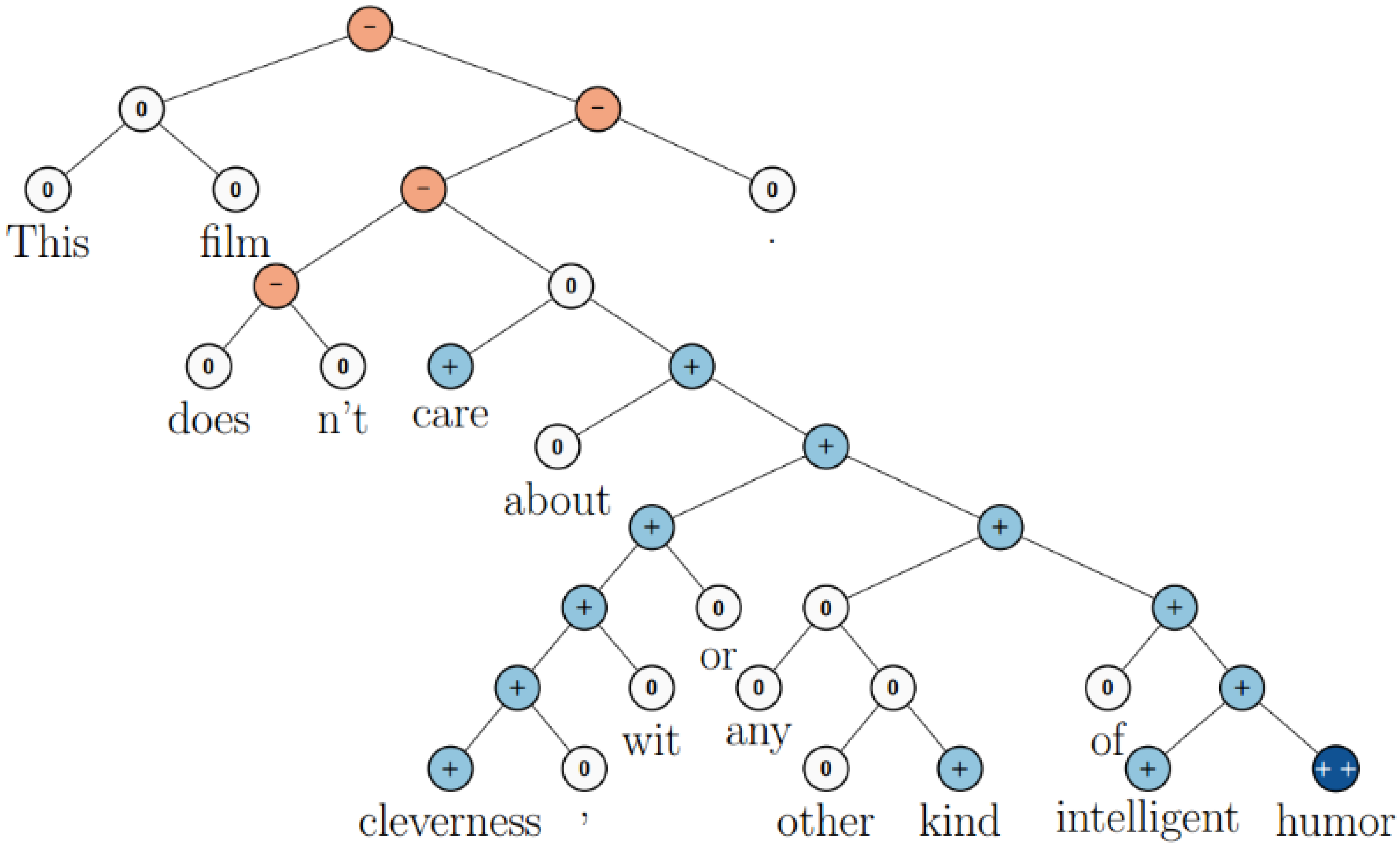
$$h_l(X) = (X \times W + b) \circ \sigma(X \times V + c)$$

- **Gated linear unit**: $X \circ \sigma(X)$
- **Output**: $Y = \mathrm{softmax}(WH_L)$

Modeling trees with Recursive NN

- Input: $x_1, x_2, \ldots, x_n$
- A binary tree T can be represented as a unique set of triplets (i, k, j), s.t. $i < k < j$, $x_{i:j}$ is parent of $x_{i:k}, i_{k+1,j}$
- RecNN takes as an input a binary tree and returns as output a corresponding set of inside state vectors $\boldsymbol{s_{i:j}^{A}} \in \mathbb{R}^d$
- Each state vector $\boldsymbol{s_{i:j}^{A}}$ represents the corresponding tree node $q_{i:j}^{A}$ and encodes the entire structure rooted at that node

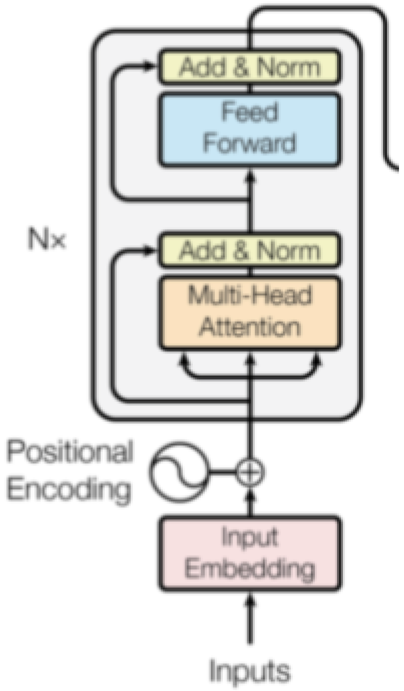
RecNN


This
film
.
does
n't
care
about
cleverness
,
wit
or
any
other
kind
of
intelligent
humor

The Transformer

An alternative architecture to RNN which allows of parallel and faster training

- Several layers of identical modules
- Each module consists of Multi-Head Attention and Feed Forward layers
- Input: embeddings. To get embeddings for numerical input, apply any dense layer
- Positional embeddings to make use of the order of the sequence

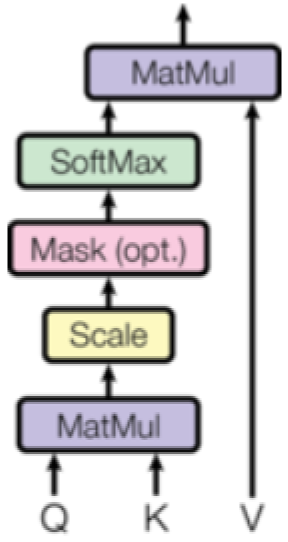


Scaled Dot-Product Attention

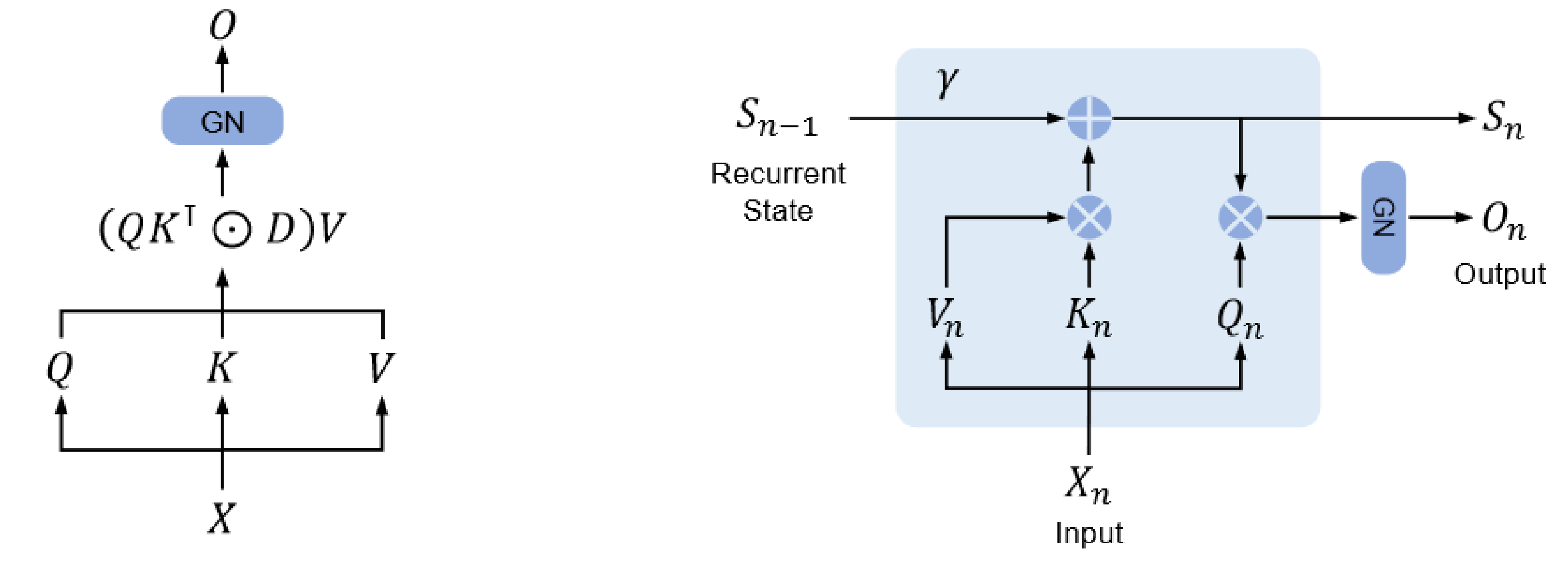
An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors:

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

where the input consists of queries Q and keys K of dimension d_k and values V of dimension d_v



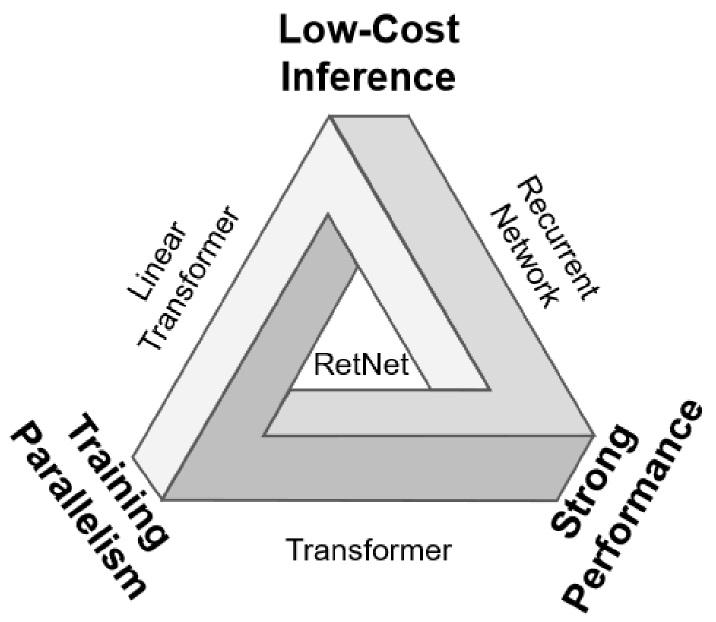
Modern RNNs: RetNet



(a) Parallel representation.

(b) Recurrent representation.

Figure 3: Dual form of RetNet. "GN" is short for GroupNorm.



The **RetNet model** (short for **Retention Network**, introduced by Microsoft Research in 2023) is a new sequence modeling architecture proposed as an alternative to Transformers.
Its main idea is to keep the parallelizability of Transformers while introducing recurrent-like efficiency for very long sequences.

Modern RNNs: RetNet

1. **Retention Mechanism**
    1. RetNet replaces the Transformer's self-attention with a new operator called **retention**.
    2. It computes token representations with a **decay-weighted recurrence**, which combines elements of attention (parallelizable) and RNNs (sequential memory).
    3. This allows it to **remember long-term dependencies** without quadratic complexity.
2. **Linear Time & Memory Complexity**
    1. Like linear-attention methods, RetNet has **O(n)** complexity in sequence length (vs. $O(n^2)$ for standard Transformers).
    2. Scales well for very long contexts (tens of thousands of tokens).
3. **Parallel + Recurrent Modes**
    1. RetNet can run in **parallel mode** (fast training, like Transformers) and **recurrent mode** (efficient autoregressive inference, like RNNs).
    2. This duality makes it especially practical for **deployment**.

Modern RNNs: Mamba

1. **State Space Model (SSM) Backbone**
    - Mamba is based on **selective state space models (SSMs)** rather than attention.
    - SSMs update a hidden state with each token, similar to RNNs, but with a **structured linear dynamical system** that allows efficient sequence modeling.
2. **Selectivity Mechanism**
    - Introduces a **data-dependent gating/selectivity** mechanism, letting the model decide which past information to keep or forget.
    - This makes memory usage **adaptive** rather than uniform (unlike RetNet's exponential decay).
3. **Linear Time Complexity**
    - Like RetNet, Mamba achieves **O(n)** complexity in sequence length.
    - Memory and compute scale linearly, making it suitable for **very long contexts (hundreds of thousands of tokens)**.
4. **Hardware Efficiency**
    - The architecture is designed to map efficiently to GPUs/TPUs.
    - Thanks to structured kernels, Mamba is **much faster than Transformers** for long sequences, both in training and inference.
5. **Continuous State Representation**
    - Maintains a **continuous hidden state** (like an RNN) that evolves over time.
    - This allows **constant-time per-token inference**, unlike attention models which need growing context windows.

Modern RNNs: Mamba

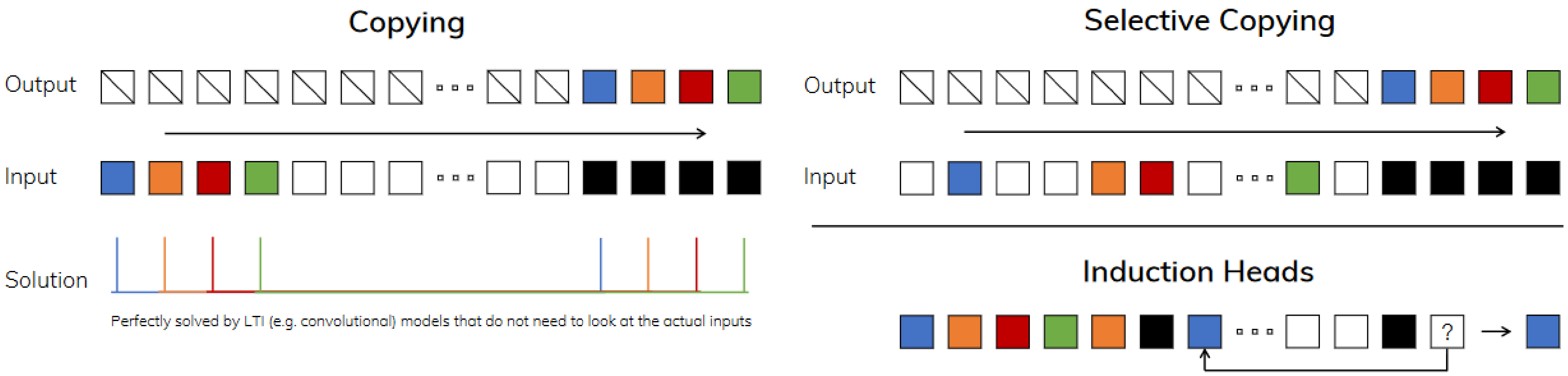
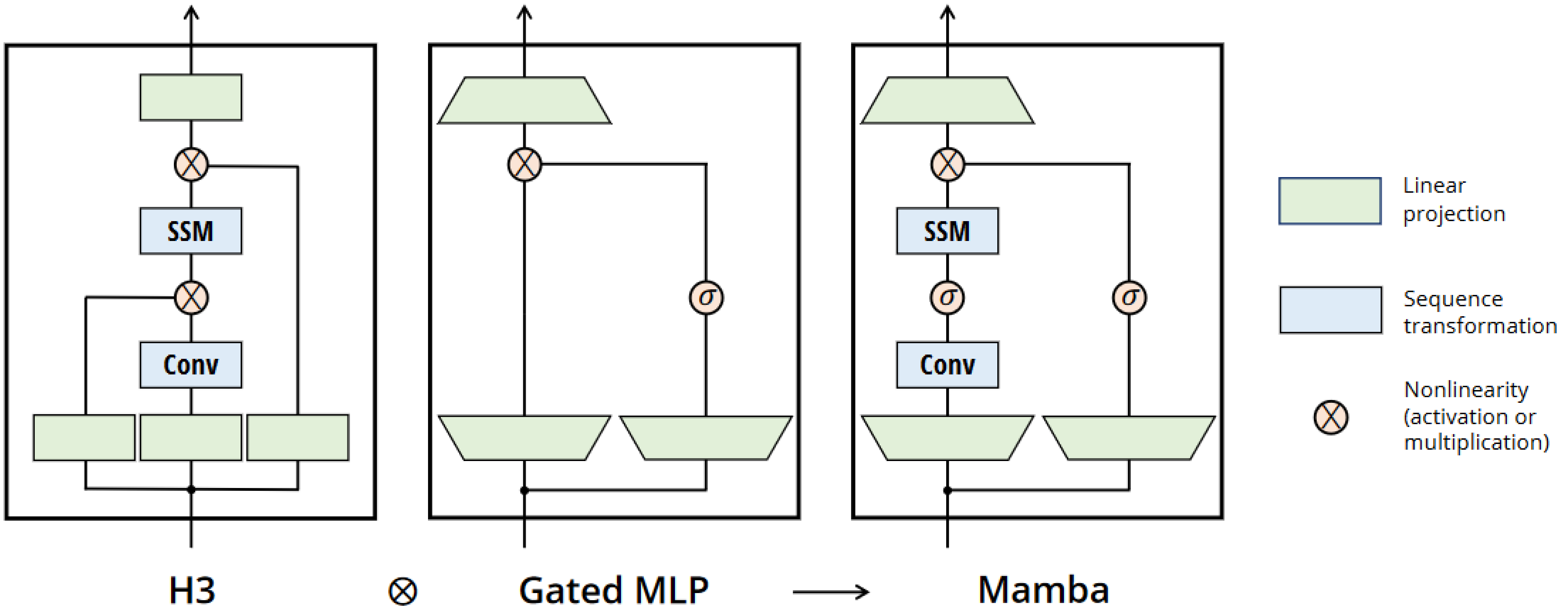
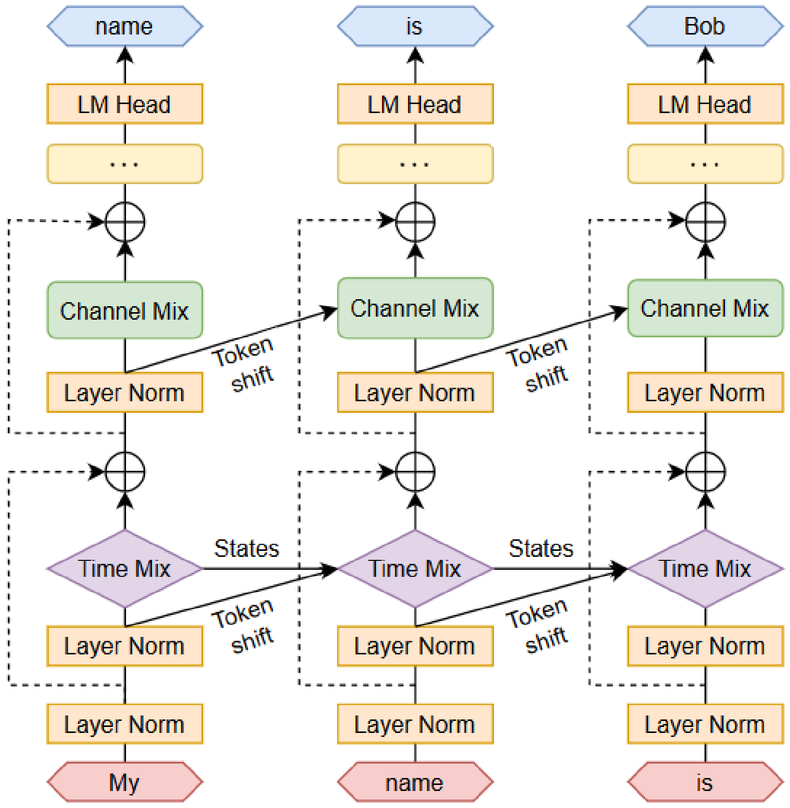


Figure 2: (*Left*) The standard version of the Copying task involves constant spacing between input and output elements and is easily solved by time-invariant models such as linear recurrences and global convolutions. (*Right Top*) The Selective Copying task has random spacing in between inputs and requires time-varying models that can *selectively* remember or ignore inputs depending on their content. (*Right Bottom*) The Induction Heads task is an example of associative recall that requires retrieving an answer based on context, a key ability for LLMs.

Modern RNNs: Mamba



Modern RNNs: RWKV



- R: The **Receptance** vector acts as the receiver of past information.
- W: The **Weight** signifies the positional weight decay vector, a trainable parameter within the model.
- K: The **Key** vector performs a role analogous to K in traditional attention mechanisms.
- V: The **Value** vector functions similarly to V in conventional attention processes.

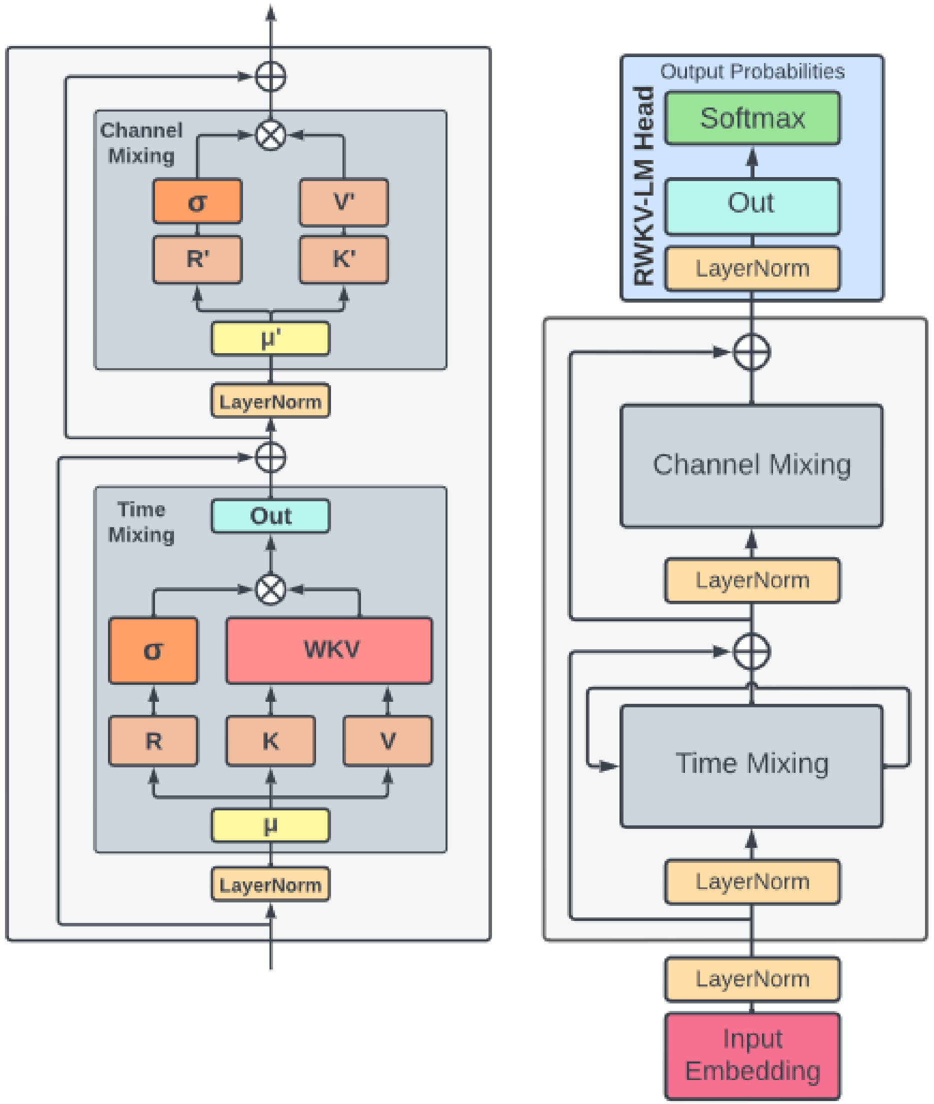


Figure 2: Elements within an RWKV block (left) and the complete RWKV residual block, equipped with a final head for language modeling (right).

Modern RNNs: RWKV

1. **RNN–Transformer Hybrid**
    1. RWKV is built as a **recurrent neural network** but trained with **Transformer-like weight sharing and architecture patterns**.
    2. It can process sequences one token at a time (like an RNN) while keeping some of the expressivity of attention models.
2. **Linear Complexity**
    1. Like RetNet and Mamba, RWKV has **O(n)** time and memory complexity.
    2. It avoids quadratic attention costs, making it efficient for **very long sequences**.
3. **Time-Mixing & Channel-Mixing Blocks**
    1. Instead of self-attention, RWKV uses **time-mixing** (temporal recurrence across tokens) and **channel-mixing** (feedforward across hidden dimensions).
    2. This plays a role similar to attention + MLP in Transformers, but in a recurrent-friendly way.
4. **Recurrent Inference Mode**
    1. RWKV can run **autoregressively like an RNN**: constant memory per token, only needing the hidden state.
    2. This makes inference **highly efficient** for deployment on CPUs and edge devices.
5. **Transformer-Like Training**
    1. Despite its RNN nature, RWKV can be trained with **parallelism similar to Transformers** using special weight formulations.
    2. This enables large-scale training with GPUs/TPUs.

Modern RNNs: ARMT

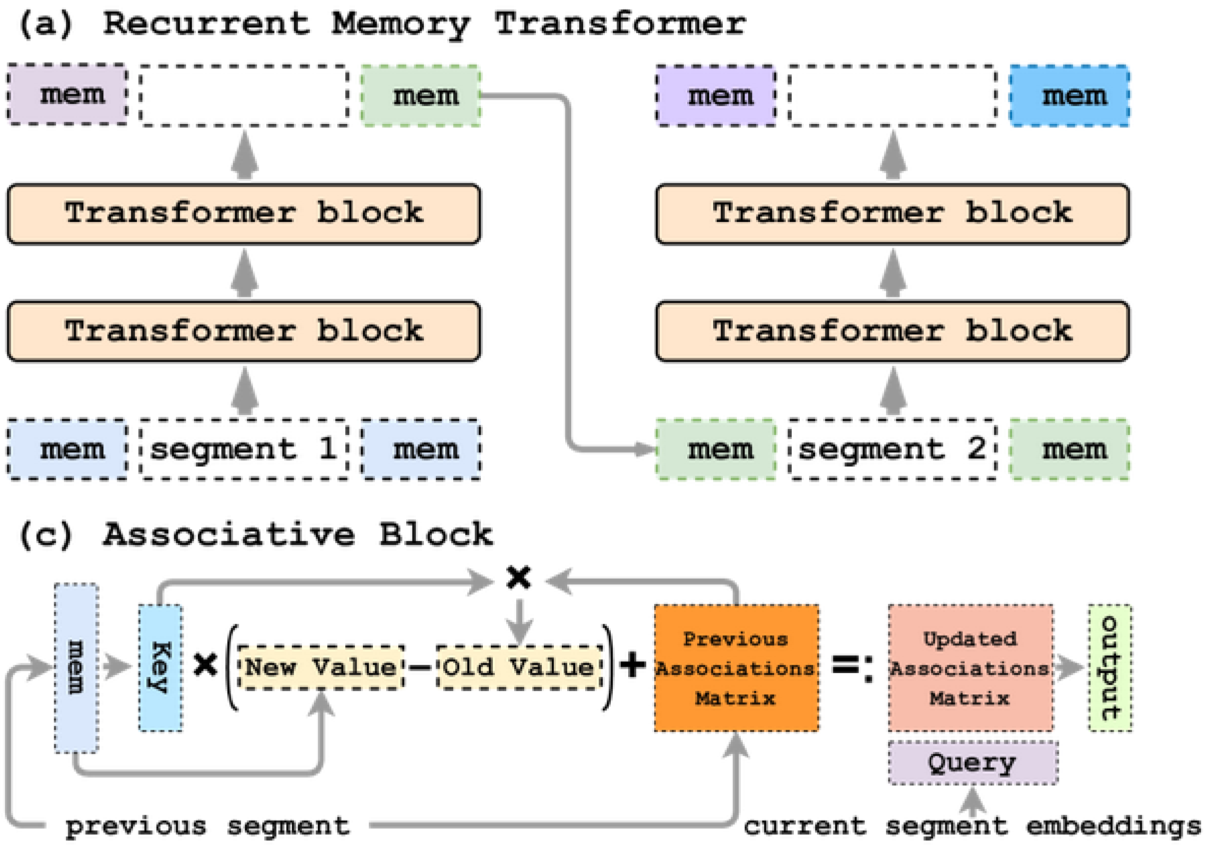


Figure 1: **ARMT augments the transformer's layers with associative memory.** (a) RMT architecture. (b) ARMT adds associative memory processing to each layer. (c) Associative memory is updated with layerwise memory representations.

Associative Recurrent Memory Transformer (ARMT) — 2024. Hybrid model combining local Transformer self-attention + **segment-level recurrence** to store and retrieve information across large contexts. Designed to handle very long sequences. "**Constant time for processing new information at each time step**" is claimed.