



VISIMAC 23

# Deep Learning with Memory

Federico Becattini  
[federico.becattini@unisi.it](mailto:federico.becattini@unisi.it)

# Course outline

- Introduction to Neural Networks
- Deep Learning with Memory
  - Internal Memory models
  - External Memory models
- Transformers

# Memory

«Memory is the power or process of reproducing or recalling what has been learned and retained especially through associative mechanisms»  
*Merriam-Webster dictionary*

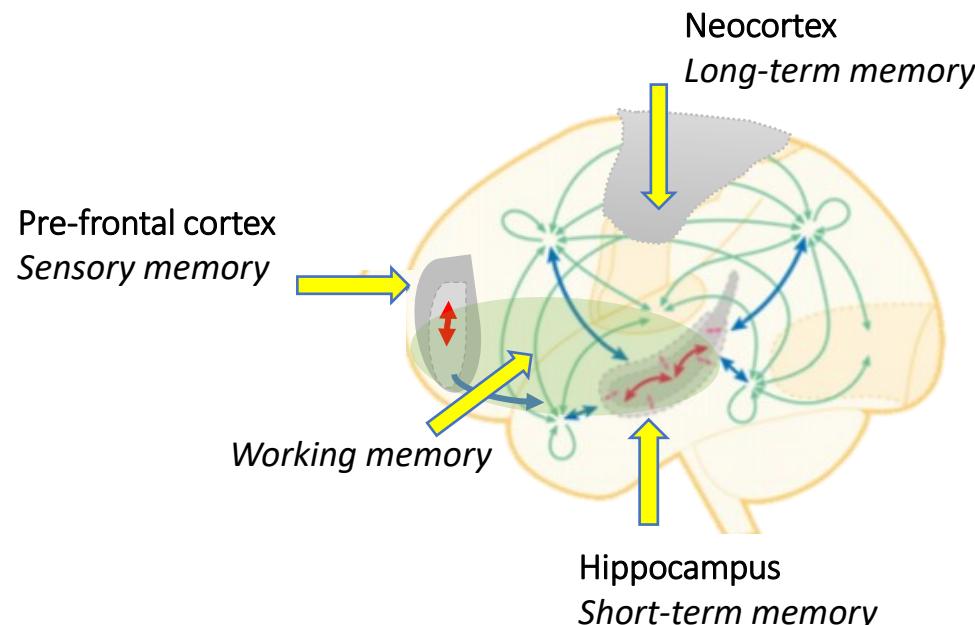
There are three major processes involved in memory:  
Encoding, Storage and Retrieval

# Human Memory

Human memory happens in many parts of the brain at once and some types of memories stick around longer than others:

- *Sensory Memory* (<1-4 secs): stores a snapshot of sensory experience in categorical stores outside of cognitive control.
- *Short-term Memory* (20-30 secs): deals with fresh information about which we have awareness. Has temporal decay and chunk capacity limits.
- *Working Memory* (20-30 secs): a short-term memory applied to cognitive tasks. Retains information in order to manipulate it using attention.
- *Long-term Memory* (extended period): information is largely outside of awareness but can be called into the working memory to be used when needed.

N. Cowan  
*Progress in Brain Research*, 2009



# Memory in AI

Two big challenges in AI:

- Define models that can capture long term dependencies in data sequences
- Define models that can make multiple computational steps to complete a task

In both cases this requires that networks have memory

PART1 – *Learning order dependency in sequence prediction problems*

RNN; LSTM; GRU  
*Memory Network models*

# Recurrent Neural Networks: RNN



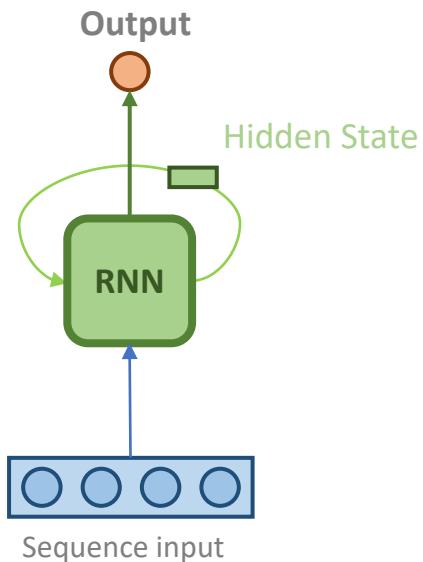
RNNs are a class of neural networks designed to interpret temporal or sequential information.

RNNs are called *recurrent* because they perform the same operation for every element of a sequence.



They have the capacity to memorize in a compact representation the result of previous operations and use it in the current step.

The **hidden state** acts as the **memory**.



The output of the hidden node is put back into the hidden node along with the input at next time step

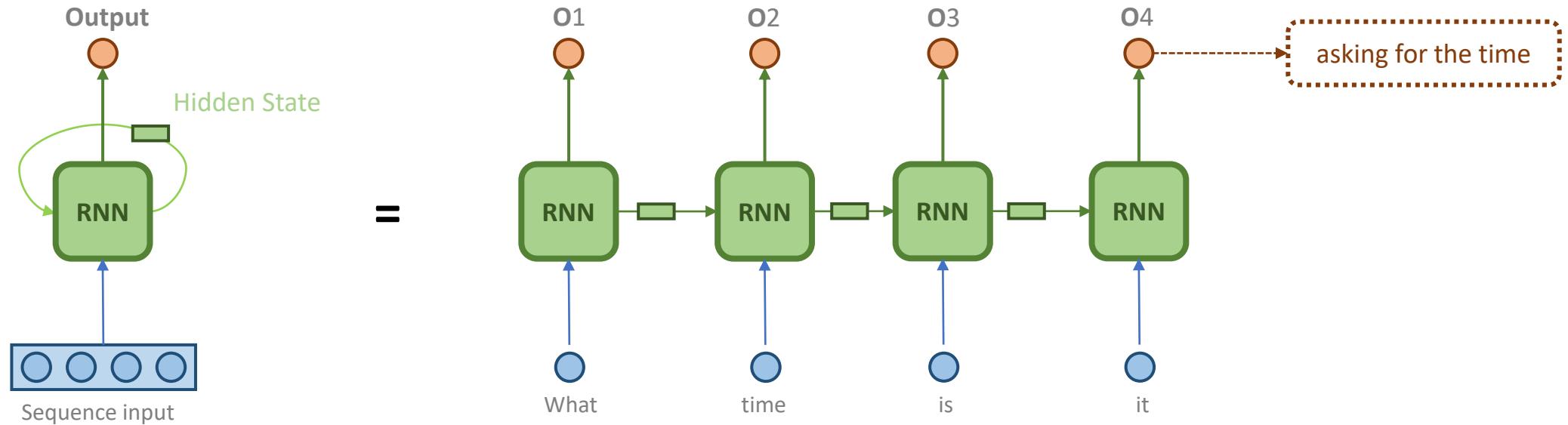
At every time step the internal state is updated and predictions can be made based on long term patterns

# Unrolled RNNs



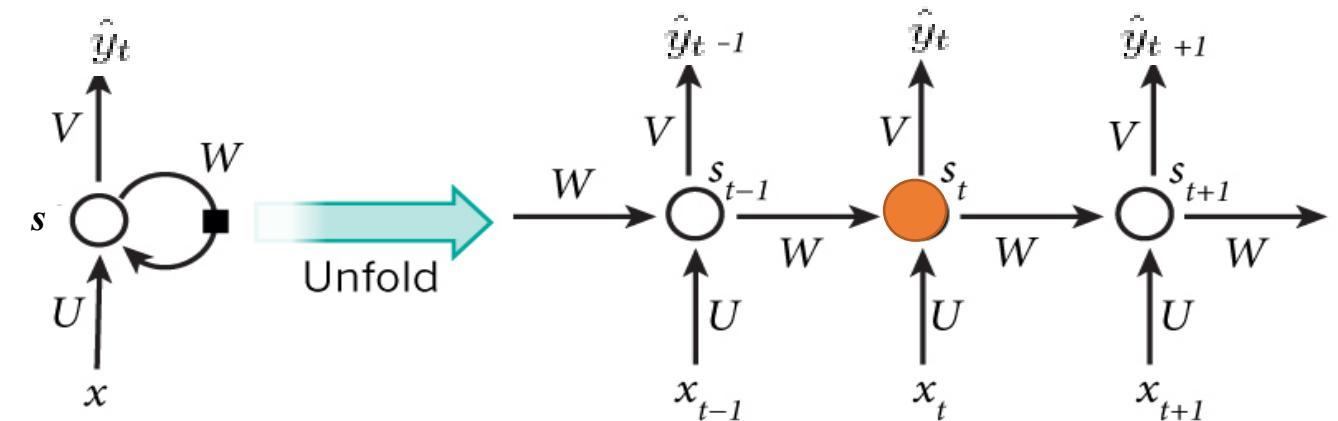
RNN can be unrolled into feed-forward models: at each time step the output becomes the next input.  
Like a deep neural network with indefinitely many layers.

The latent state of an RNN should give weights to samples in a time series, remembering important events.



# RNN equations

RNN's parameters are three weight matrices:  
W is applied to the previous hidden state  
U is applied to the current input  
V is applied between the hidden state and the output



Basic equations of RNN

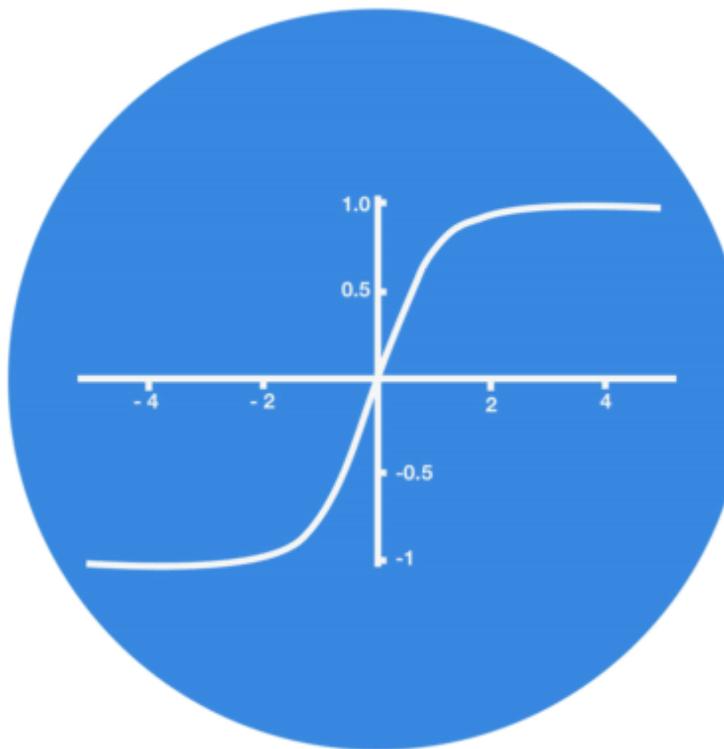
$$s_t = \tanh(Ux_t + Ws_{t-1})$$
$$\hat{y}_t = (Vs_t)$$

# Tanh activation

An *hyperbolic tangent* function ensures that the values stay between -1 and 1 thus regulating the output of the neural network.

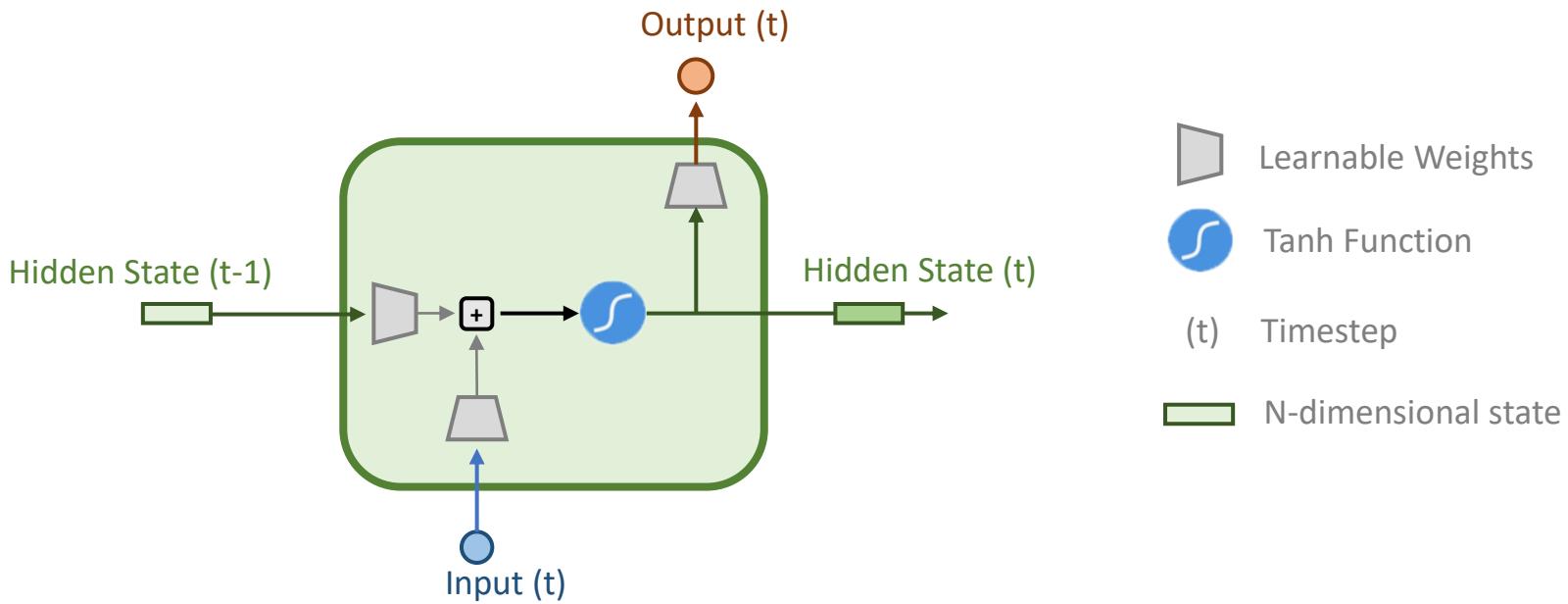
It also adds a non-linearity to the model.

5  
0.1  
-0.5



$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

# Recurrent Neural Network: RNN



Data: Time series, text, audio, videos

RNNs can process variable length sequences of inputs.

RNNs share the learnable weights across different time steps.



Speech recognition  
Sentiment analysis  
Language modeling  
Stock predictions

# RNN training

The full sequence is a single training example.

Each instance in the unfolded network shares the same parameters.

The error can be accumulated for all the time steps and the weights are updated depending on the whole sequence of hidden states.

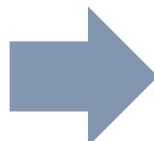
If the model generates an output after every time step, errors are summed across time steps.

Thus, weight updates in all time steps are summed together.

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

$$\hat{y}_t = (Vs_t)$$

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$



$$E(y, \hat{y}) = \sum_t E_t(y_t, \hat{y}_t) = - \sum_t y_t \log \hat{y}_t$$

# Backpropagation through time

RNNs consist entirely of differentiable operations. They can be trained using an extension of the backpropagation algorithm: *Backpropagation Through Time*

Each time step in a RNN is like a fully connected layer.

Weight updates are calculated by applying the chain rule gradients across the unfolded network.

EXAMPLE: at step 3, calculate the error wrt parameters V, W, U.

$$\frac{\partial E_3}{\partial V} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial V}$$

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W}$$

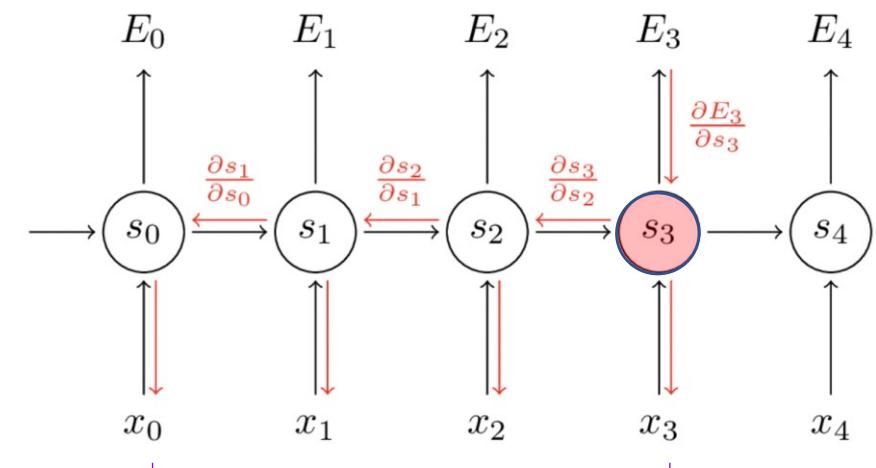
$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left( \prod_{j=k+1}^3 \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial W}$$

$$s_t = \tanh(Ux_t + Ws_{t-1})$$

$$\hat{y}_t = (Vs_t)$$

$$E_t(y_t, \hat{y}_t) = -y_t \log \hat{y}_t$$

$$\frac{\partial E_3}{\partial U} \quad similar \quad process \quad = \sum_{k=0}^3 \frac{\partial E_3}{\partial \hat{y}_3} \frac{\partial \hat{y}_3}{\partial s_3} \left( \prod_{j=k+1}^3 \frac{\partial s_j}{\partial s_{j-1}} \right) \frac{\partial s_k}{\partial U}$$



$$\frac{\partial s_3}{\partial s_0} = \frac{\partial s_3}{\partial s_2} \frac{\partial s_2}{\partial s_1} \frac{\partial s_1}{\partial s_0}$$

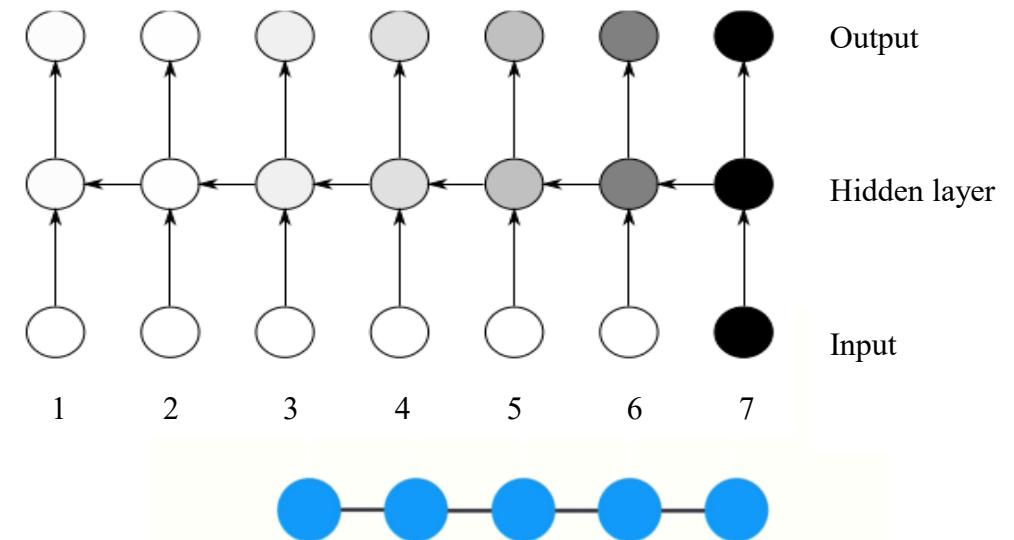
# Exploding /vanishing gradient

With multiple matrix multiplications gradient values might either grow or shrink exponentially.

Accumulation of large derivatives results in the model being very unstable and incapable of effective learning.

If derivatives are small, gradient contributions to learning from far away steps might become zero. Long-range dependencies are not learned.

In practice, the range of learned contextual information is limited to approximately 10 time steps.  
Inherently unstable over long timescales.



# Exploding /vanishing gradient

To avoid exploding gradients, gradient clipping at each timestamp can be used:  
*check if gradient > threshold and normalize it*

$$\mathbf{g} \leftarrow c \cdot \mathbf{g} / \|\mathbf{g}\|$$

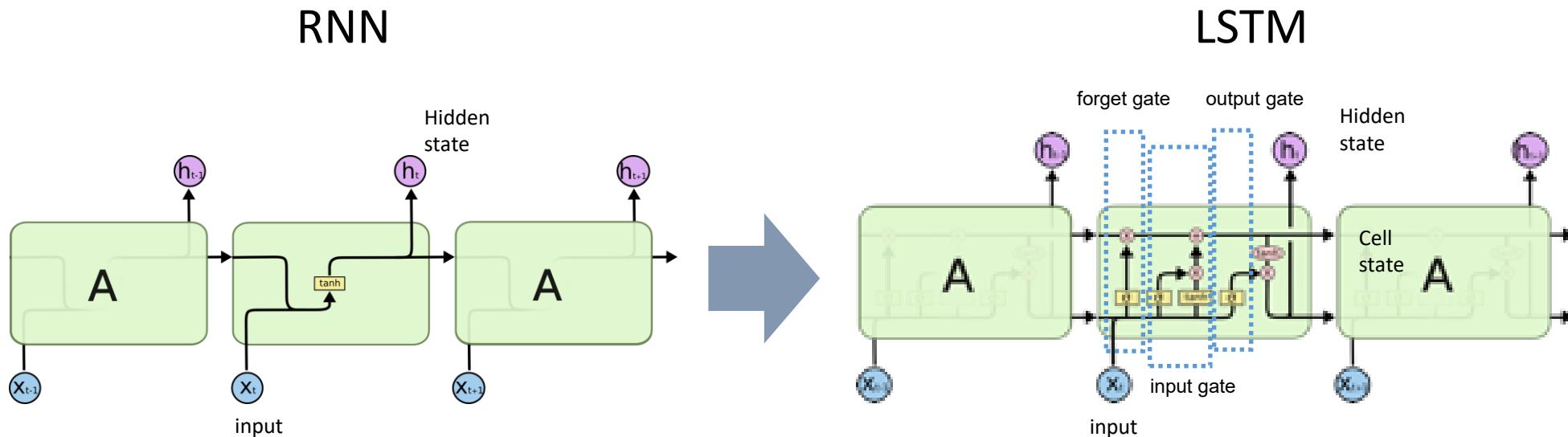
To tackle the vanishing gradient problem:

- use ReLU instead of *tanh* or *sigmoid* activation function
- initialize the W matrix with an identity matrix
- use gated cells such as LSTM or GRUs

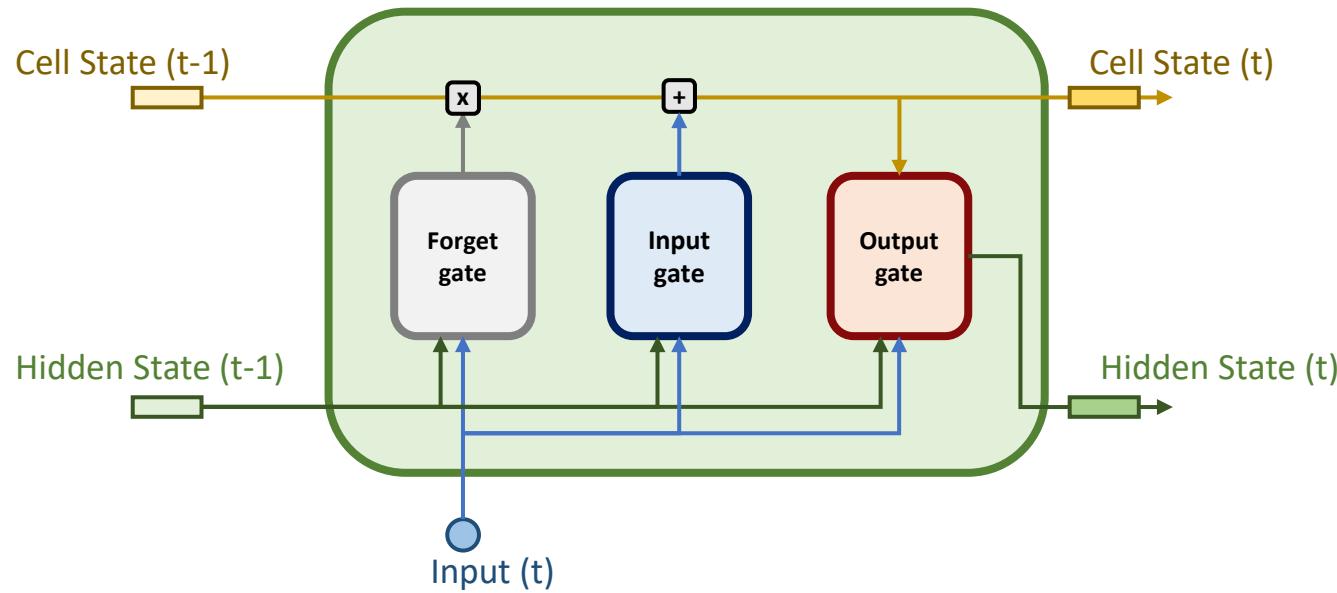
# Long Short Term Memory (LSTM)

Long Short-Term Memory (LSTM) to solve the problem of exploding/vanishing gradients.

LSTM memory cell: what an LSTM stores in its memory cell is regulated by layers with activations called gates that decide which information to forget or to remember.



# Long Short Term Memory (LSTM)



The cell state memorizes relevant information throughout the processing of the sequence and the gates learn which information to forget or to remember.

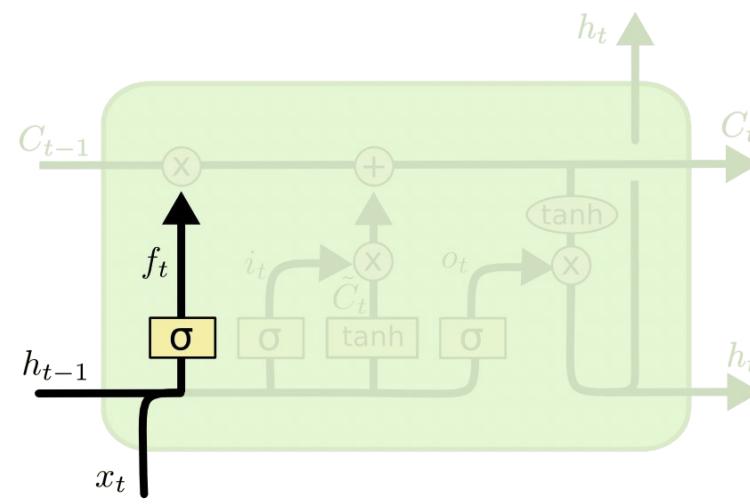


The gates are neural networks that contain learnable weights and activation functions. They generate values between 0 and 1.

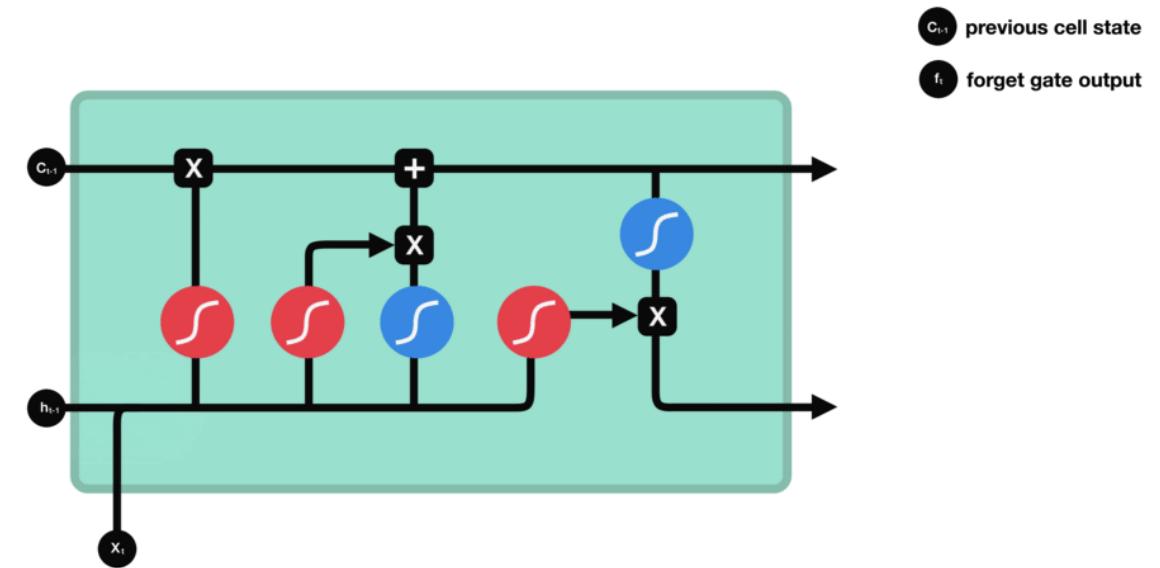
# LSTM gates – Forget Gate

Decides which information should be thrown away from the memory cell and whether to retain the memory or discard it depending on the current input.

Sigmoid outputs close to 0 mean that information is going to be forgotten. Sigmoid outputs close to 1 mean that information is going to be kept.



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



Credits C. Olah

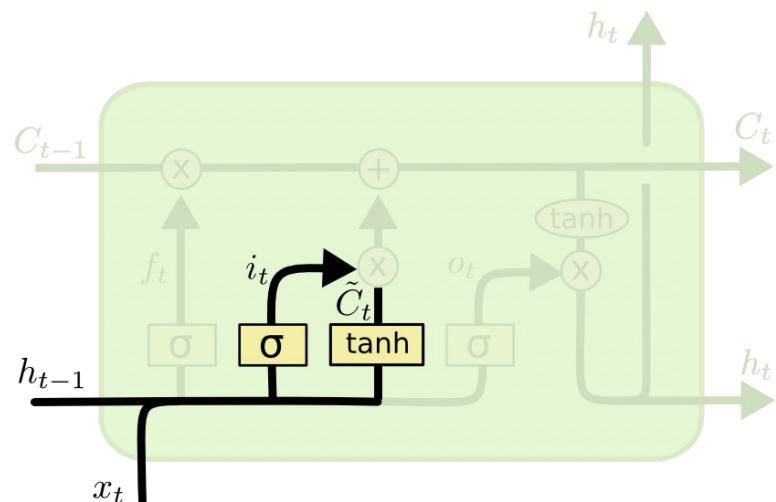
Credits M. Nguyen

# LSTM gates – Input Gate

Decides how much of the new information will be let through the memory cell and which values of the memory to update with the new observations.

*Sigmoid* output 0 means not important, 1 means important.

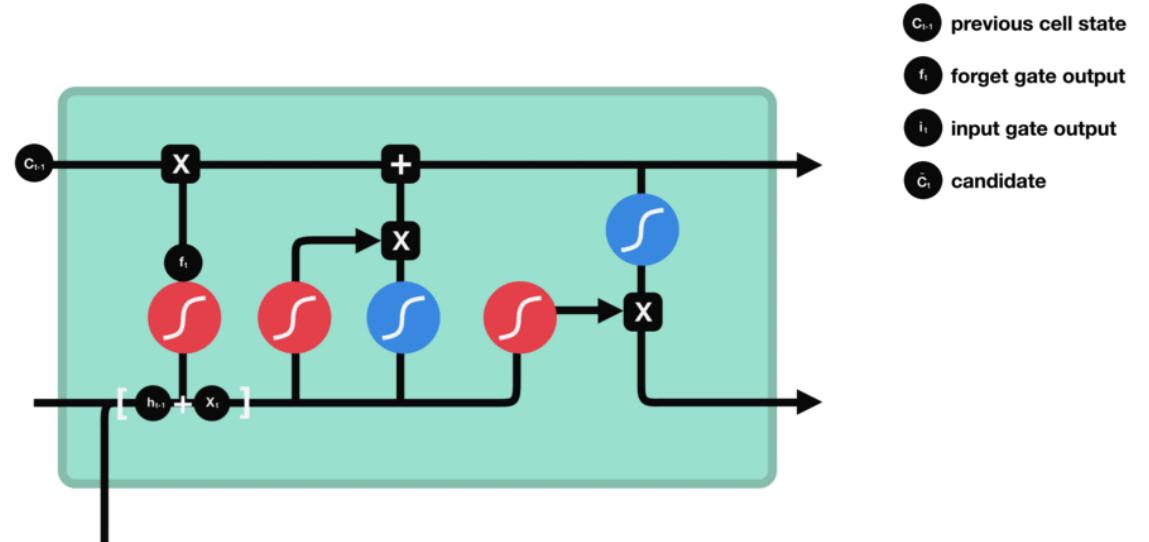
*tanh* function compresses the values between -1 and 1 to regularize the network.



$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$\tilde{C}_t$  provides change contents

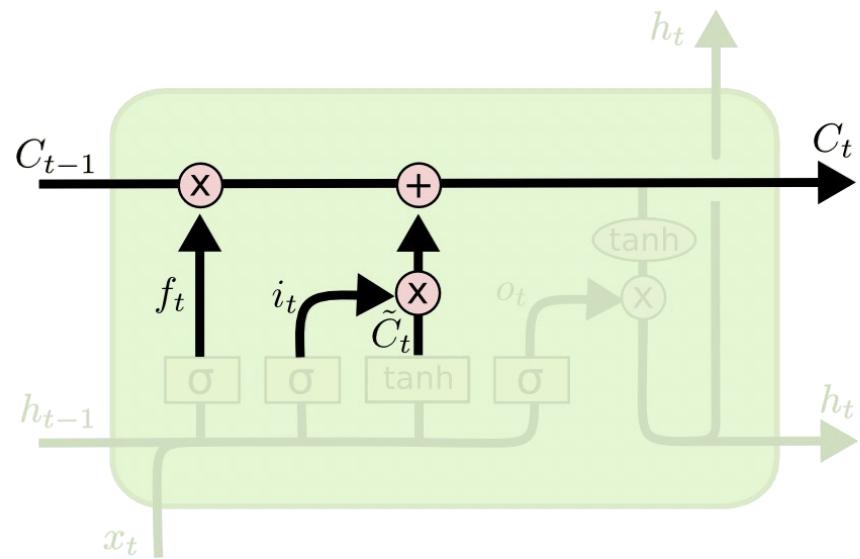


Credits C. Olah

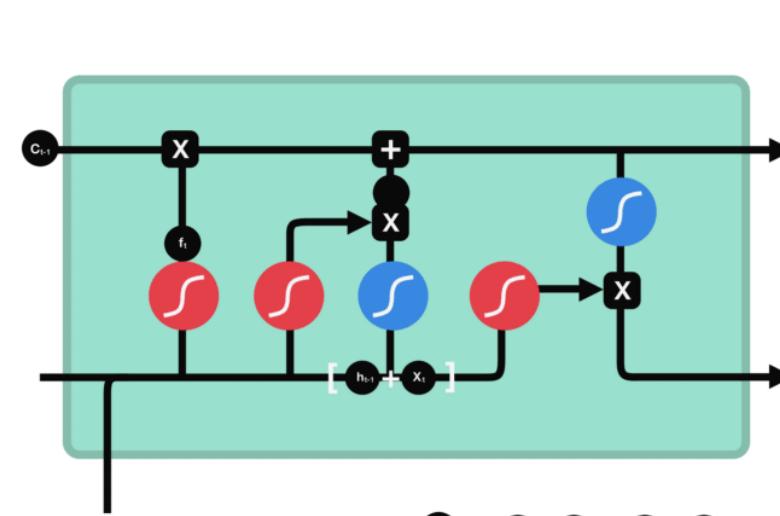
Credits M. Nguyen

# LSTM gates – State Update

Updates the state based on the previous state, subject to forget and input changes.



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



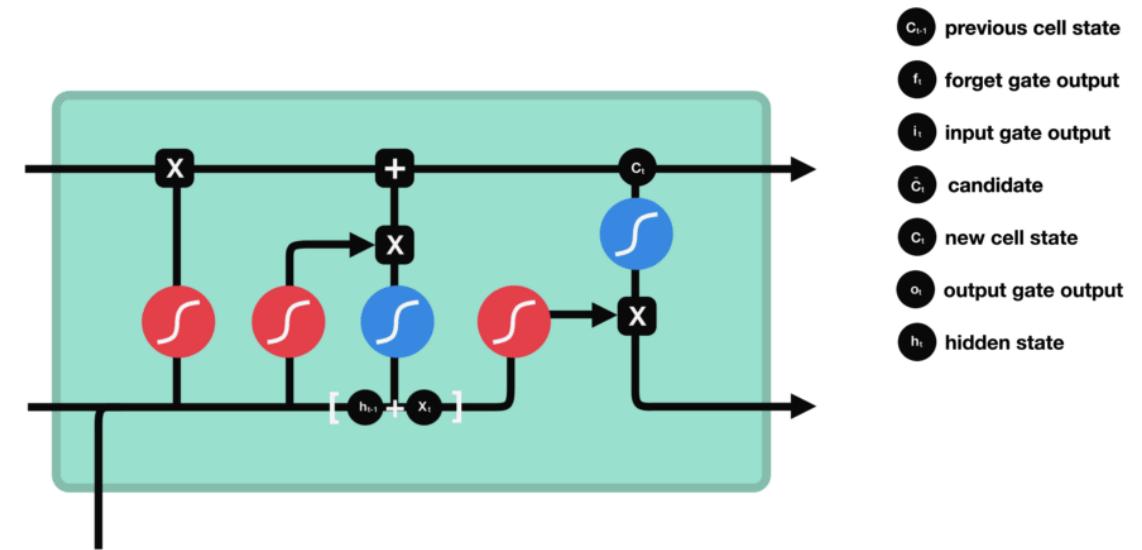
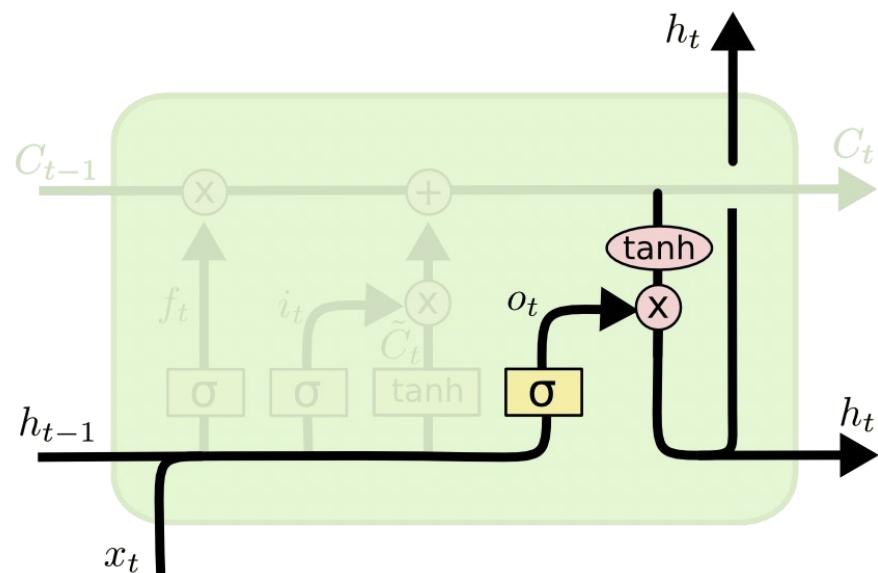
Credits C. Olah

Credits M. Nguyen

# LSTM gates – Output Gate

Decides how much information will be passed the next time step and what is going to be produced as output based on the current memory.

The hidden state contains information on previous inputs.



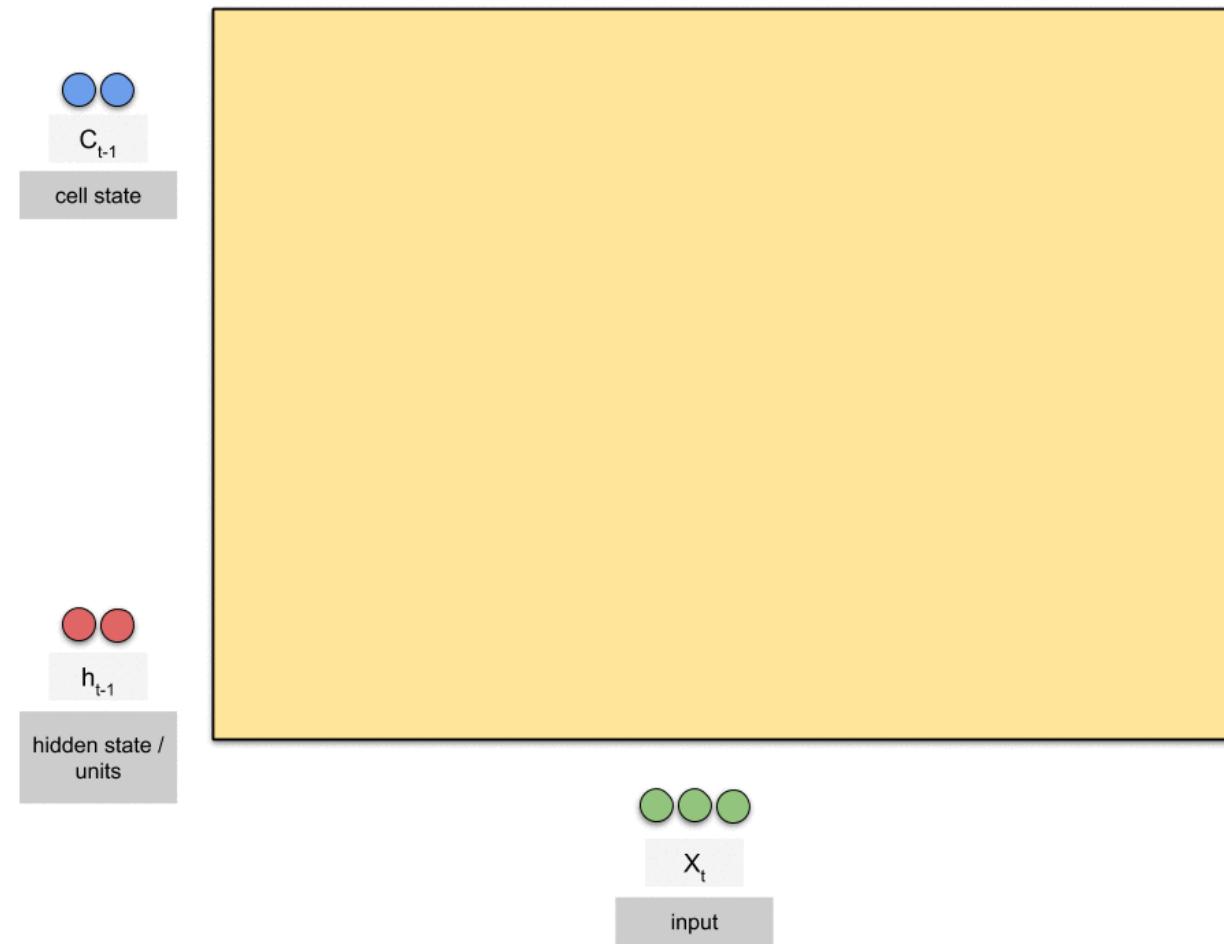
$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Credits C. Olah

Credits M. Nguyen

# LSTM

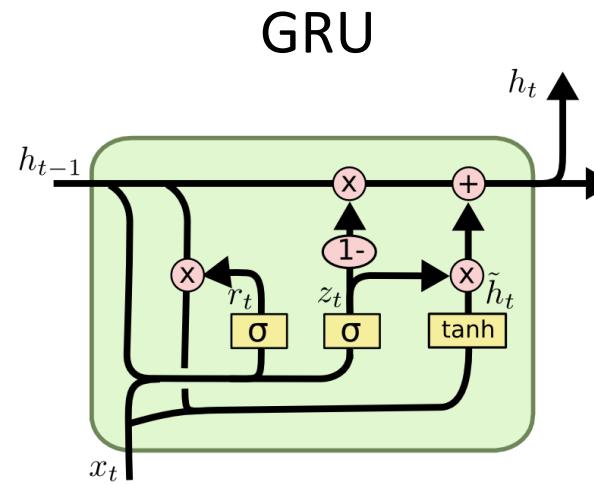
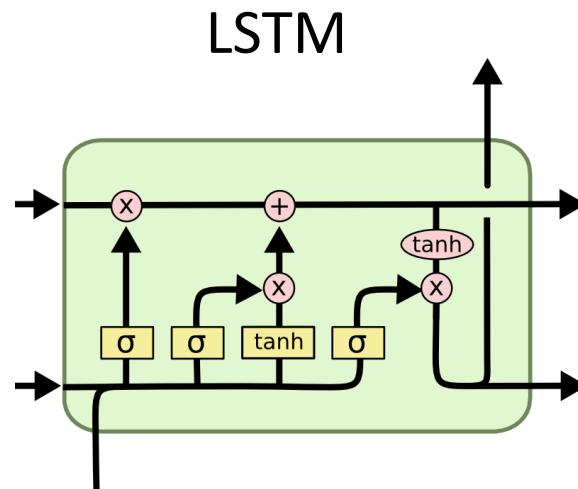


# Gated Recurrent Units

*Gated Recurrent Unit (GRU)* is a simplified version of LSTM which combines the Forget and Input gates into a single Update gate. Typically used instead of standard LSTM.

Cell state and hidden state are also merged making the model easier to train.

GRUs have less weights and therefore are lighter and faster. No groundbreaking performance difference.



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

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

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

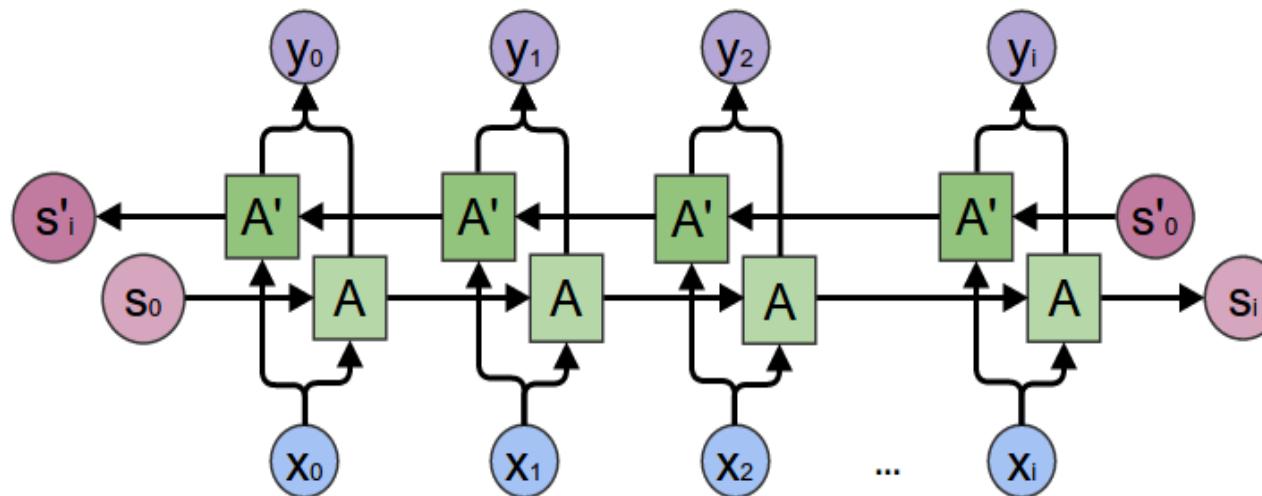
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

# Bidirectional LSTM

*Bidirectional* LSTMs train two models instead of one. Instead of observing only the past, the future is also taken into account.

Two processing branches: the data flows from the beginning to the end and vice versa.  
The two hidden states are combined.

This provides additional context to the network and results in faster and more effective learning.  
The whole sequence must be available. Not suitable for online learning.



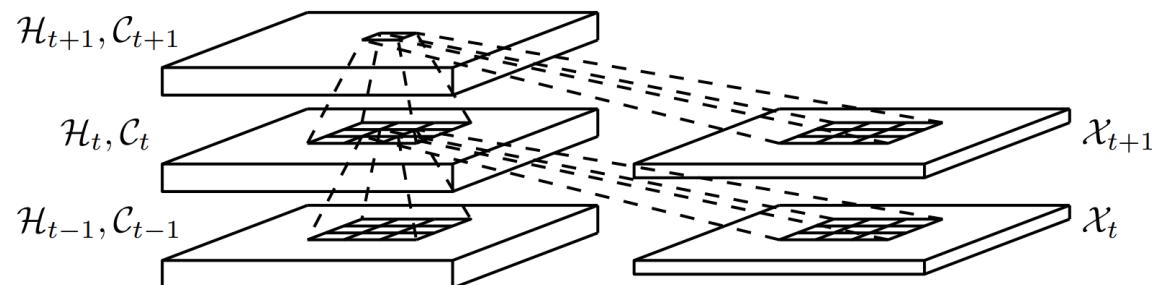
# Convolutional LSTM

Traditional LSTMs have vectors as inputs. *Convolutional LSTM* (ConvLSTM) is an extension of LSTM to tensorial data, e.g. images or convolutional feature maps.

Inputs, hidden states and gates of the ConvLSTM are 3D tensors.

Inputs and states are imagined as vectors standing on a spatial grid.

Multiplications with weights are replaced by the convolution operator ( \* )



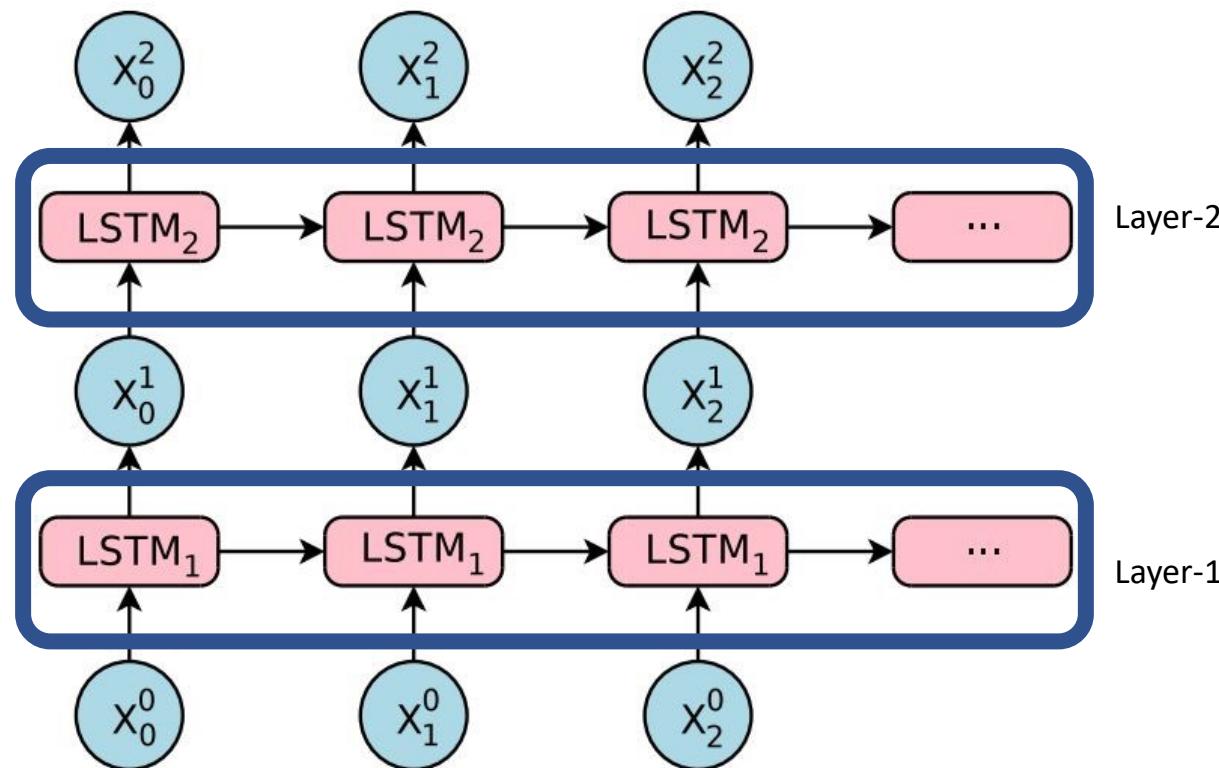
$$\begin{aligned} i_t &= \sigma(W_{xi} * \mathcal{X}_t + W_{hi} * \mathcal{H}_{t-1} + W_{ci} \circ \mathcal{C}_{t-1} + b_i) \\ f_t &= \sigma(W_{xf} * \mathcal{X}_t + W_{hf} * \mathcal{H}_{t-1} + W_{cf} \circ \mathcal{C}_{t-1} + b_f) \\ \mathcal{C}_t &= f_t \circ \mathcal{C}_{t-1} + i_t \circ \tanh(W_{xc} * \mathcal{X}_t + W_{hc} * \mathcal{H}_{t-1} + b_c) \\ o_t &= \sigma(W_{xo} * \mathcal{X}_t + W_{ho} * \mathcal{H}_{t-1} + W_{co} \circ \mathcal{C}_t + b_o) \\ \mathcal{H}_t &= o_t \circ \tanh(\mathcal{C}_t) \end{aligned}$$

# Stacked LSTM

*Stacked LSTM* is an extension of the LSTM model that has multiple hidden LSTM layers where each layer contains a different memory cells.

The first LSTM, rather than generating a single output, provides a sequence of outputs that are fed to the second LSTM and so on.

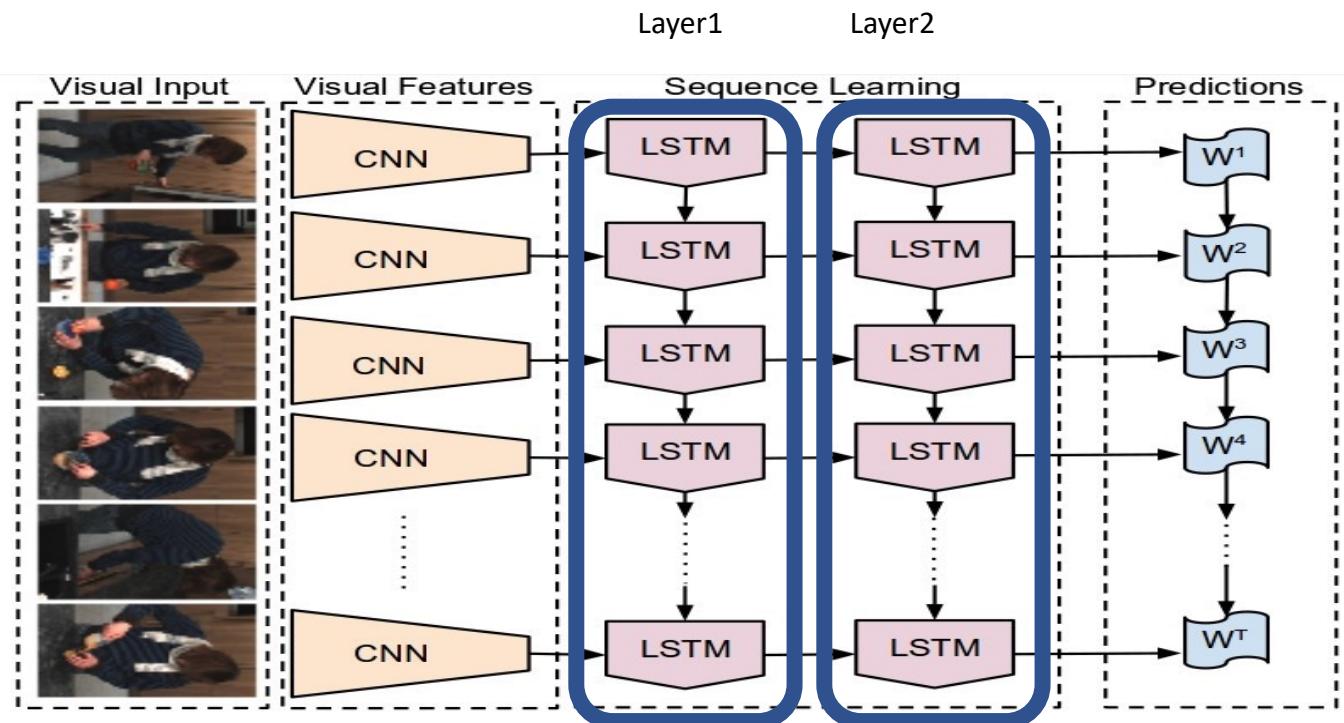
Given that LSTMs operate on sequential data, the addition of layers adds levels of abstraction of input observations over time.



# Stacked LSTM

A situation in which it is advantageous to stack LSTMs is when we want to learn hierarchical representations of time-sequence data e.g. a video sequence.

In stacked LSTMs, each LSTM layer outputs a sequence of vectors which will be used as an input to a subsequent LSTM layer. This hierarchy of hidden layers enables more complex representation of the sequence data, capturing information at different timescales.



# LSTM/GRU training

When training LSTM/GRU data must be organized in sequences

Input has different dimensionality with reference to traditional models such as CNNs:

convolutional layer : (Batch\_Size) x (Height) x (Width) x (Num\_Channels)

recurrent layer : (Batch\_Size) x (Num\_Time\_Steps) x (Feature\_Dimension)

Instead of tensorial data LSTM/GRUs usually work with feature vectors that are organized sequentially in time steps

The example is not fed to the layer all at once but one timestep at a time.

# LSTM/GRU training

- If an output is provided for each timestep then error and gradients can be obtained for each step and backpropagated through the previous ones.
- If the output is generated only at the end of the sequence there will only be a single error to backpropagate through all timesteps.

The weights that are updated are the same since the same LSTM/GRU is used for each timestep. What changes is the amount of gradients that are computed.

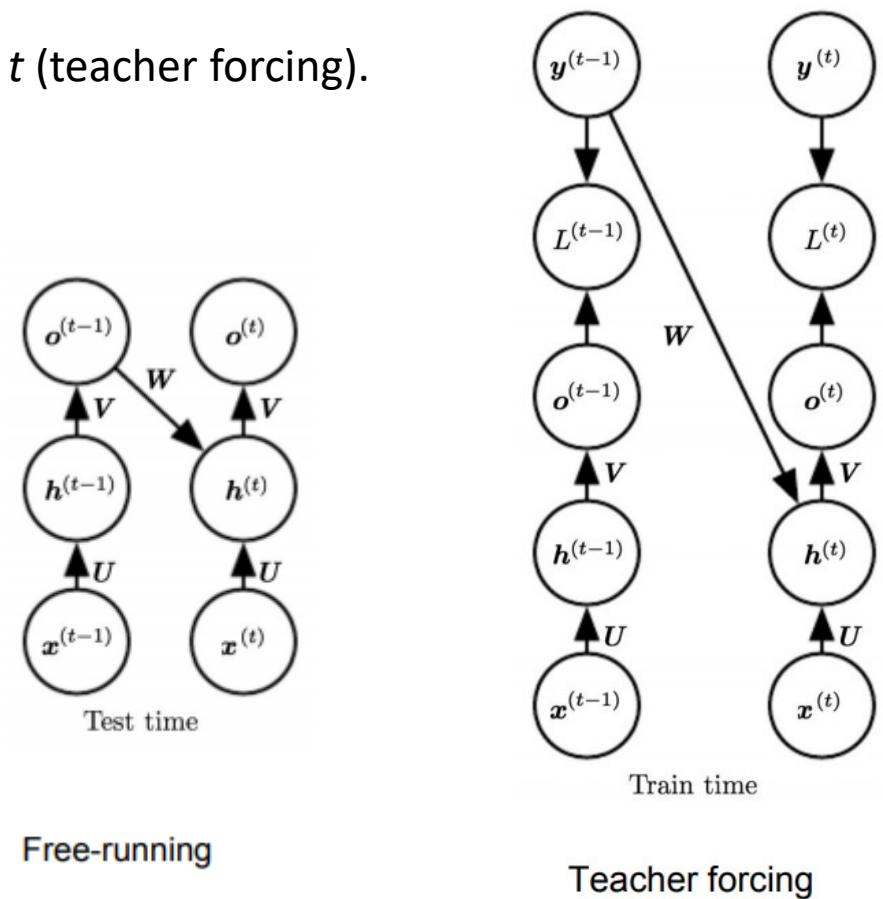
The cells learn when to allow data to enter, leave or be deleted through the iterative process of making guesses, backpropagating error and adjusting weights via gradient descent.

# Teacher Forcing

LSTMs can be trained autoregressively, i.e. the output at time  $t-1$  can be fed as input at time  $t$ . This is the case of LSTM for prediction and generation task.

Weights at early training stages might produce incorrect outputs. The LSTM will observe incorrect inputs and might diverge.

As a solution the ground truth at time  $t-1$  can be fed as input at time  $t$  (teacher forcing).



# Teacher Forcing

When using *teacher forcing* train and test data might not be distributed in the same way since the network will not generate perfect outputs.

At test time small errors might accumulate and lead to inputs that the network has never seen before.

A possible solution is to alternate between *teacher forcing* and standard training sampling i.e. using either the output of the LSTM or the ground truth as input for the next timestep.

# LSTM in practice

LSTM/GRU training issues to consider :

*Sequence length:* sequences of variable lengths need to be padded to a fixed length to enable batch processing. Data preprocessing can often be painful.

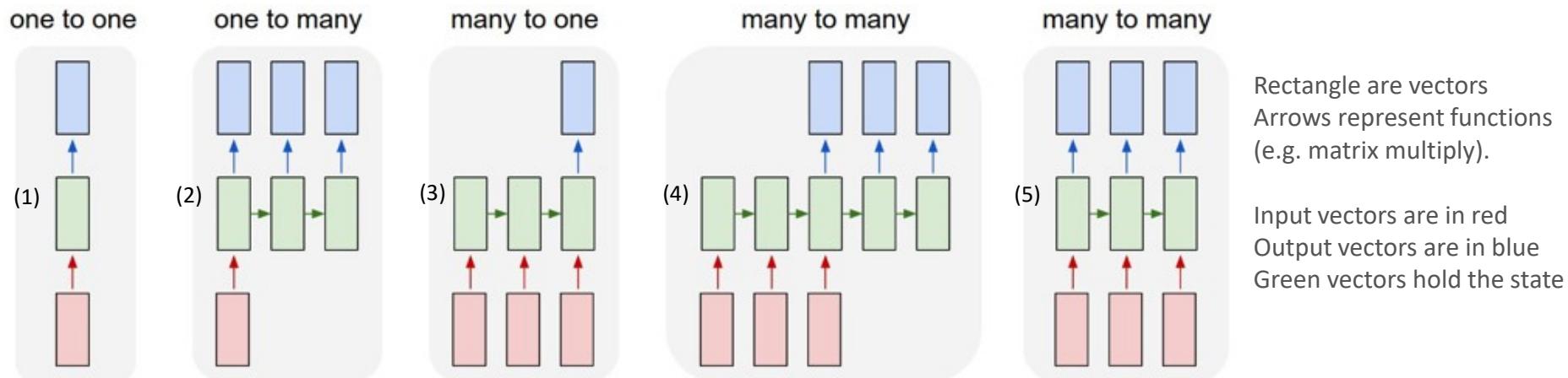
*Attention window:* in principle, LSTM/GRUs are able of processing longer sequences than vanilla RNNs. But for long sequences one may want to fix a window of attention, making the current sample depend only on a limited number of past observations.

*Interpretation vs temporal patterns:* evidence exists about difficulties of LSTM/GRUs to learn both the interpretation of the input and temporal patterns at the same time. It is better to feed LSTM/GRUs with meaningful features, e.g. pretrained convolutional features.

*Overfitting:* if the LSTM/GRU memorizes all the training data it is likely to overfit. Data augmentation and regularization are important.

# Memory Networks applications

LSTM/GRUs allow to operate over sequences of vectors: sequences in the input, sequences in the output, or in the most general case both.



- (1) Processing without RNN, from fixed-sized input to fixed-sized output (e.g. image classification)
- (2) Sequence output (image captioning takes an image and outputs a sentence of words)
- (3) Sequence input (sentiment analysis where a given sentence is classified as expressing positive or negative sentiment)
- (4) Sequence input and sequence output (machine translation: reads a sentence in French and then outputs a sentence in English)
- (5) Synced sequence input and output (video classification where we wish to label each frame of the video)

# Memory Networks applications

## Main applications examples

### Text

- Language Modeling
- Speech Recognition
- Machine Translation
- Question Answering
- .....

### Multimodal

- Image Captioning
- Video Captioning
- Visual Question Answering
- .....

### Video

- Visual Tracking
- Video Analysis
- Human Behavior Understanding
- .....

### Other

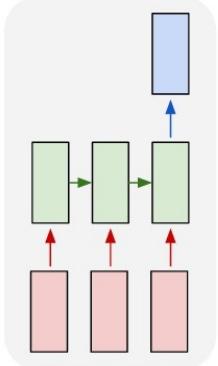
- Interaction
- Robotics
- Audio Analysis
- Automotive
- .....

# VQA: Visual Question Answering

A. Agrawal et al, ICCV 2015

many to one

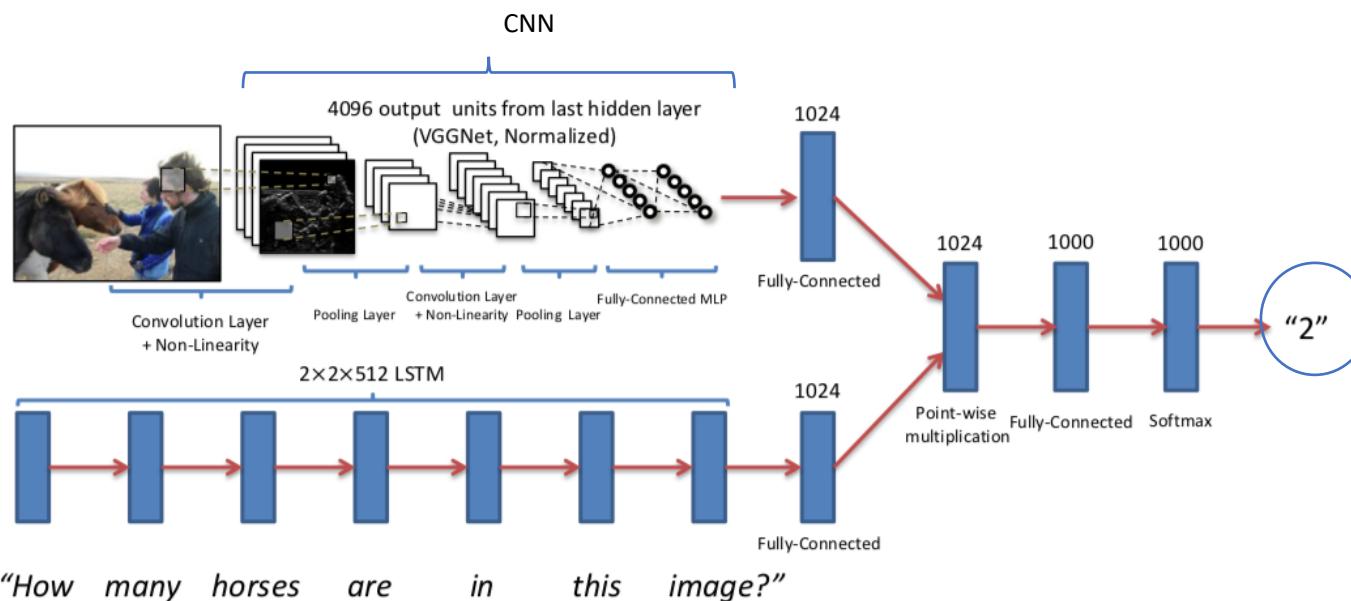
VQA on the global image content



*The LSTM is fed with features representing a textual question sentence.*

*The CNN fully connected layer provides content information about the image.*

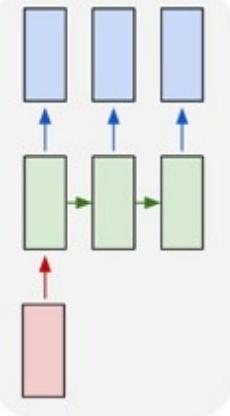
*The LSTM output is combined with the image content representation and produces the required answer*



# Deep Visual-Semantic Alignments for Generating Image Descriptions

A. Karpathy, L. Fei-Fei, CVPR2015

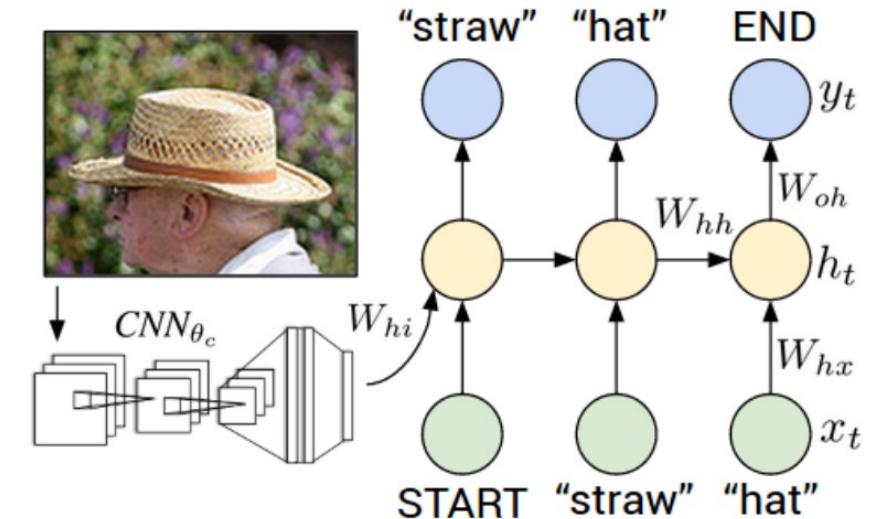
one to many



Multimodal Recurrent Neural Network generative model.

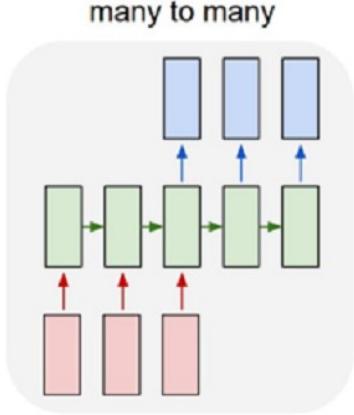
The RNN takes a word, the context from previous time steps and defines a distribution over the next word in the sentence.

The RNN is conditioned on the image information at the first time step.  
START and END are special tokens.



# Sequence to Sequence Learning with Neural Networks

I. Sutskever et al. Neurips 2014

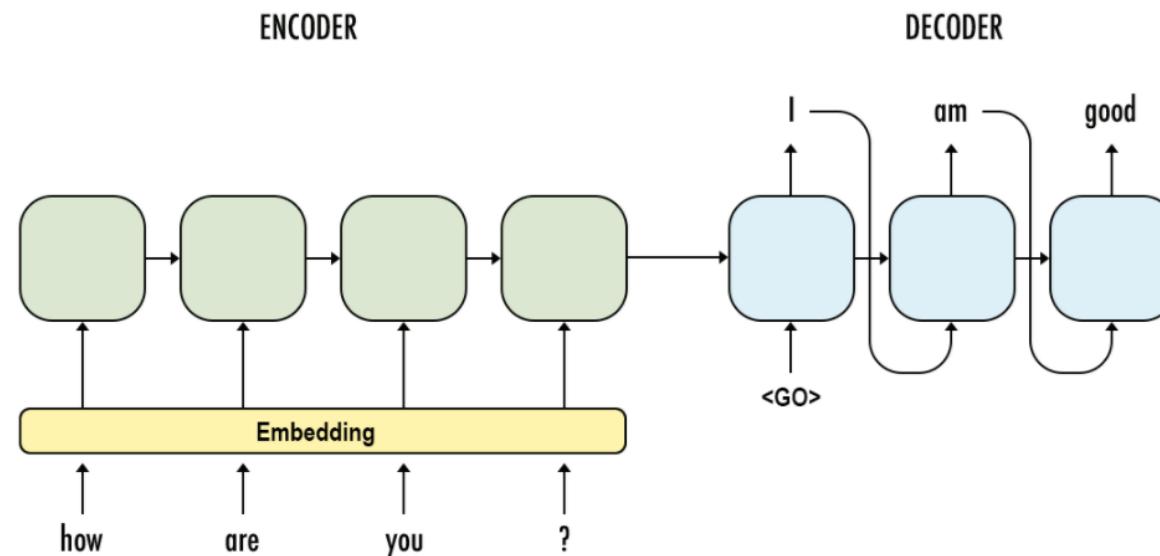


Encoder-Decoder structure.

The model reads an input sentence “ABC” and produces “WXYZ” as the output sentence. It stops after generating the end-of-sentence token.

Input and output lengths are decoupled.

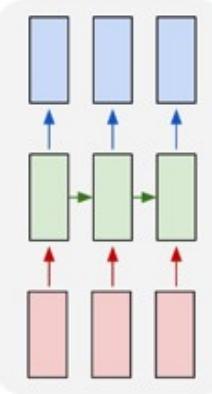
Output generation is performed autoregressively.



# Am I Done? Predicting Action Progress in Videos

F. Becattini et al. ACM TOMM 2020

many to many



## *Muhammad Ali boxing*

In 1977 Muhammad Ali was able to estimate the progress of his opponent Michael Dokes so well that he could dodge 21 punches in 10 seconds...



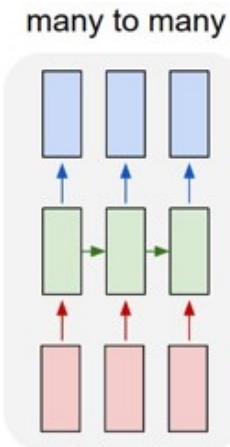
## *Young Frankenstein movie*

The monster is not able to correctly estimate the progress of the blind priest's actions and coordinate his movements accordingly...



# Am I Done? Predicting Action Progress in Videos

F. Becattini et al. ACM TOMM 2020

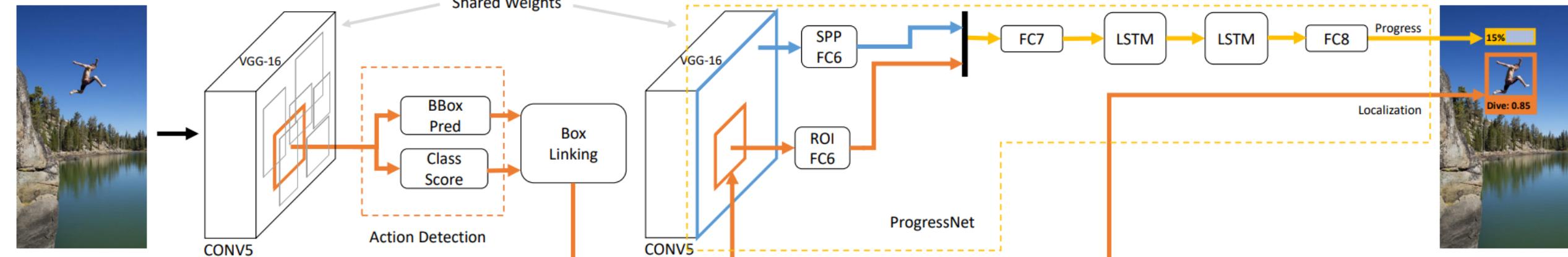


*ProgressNet predicts the progress of the ongoing action at each time step.*

*Predictions are made framewise, observing the whole past history.*

*Faster R-CNN applied framewise + box linking to predict action tubes.*

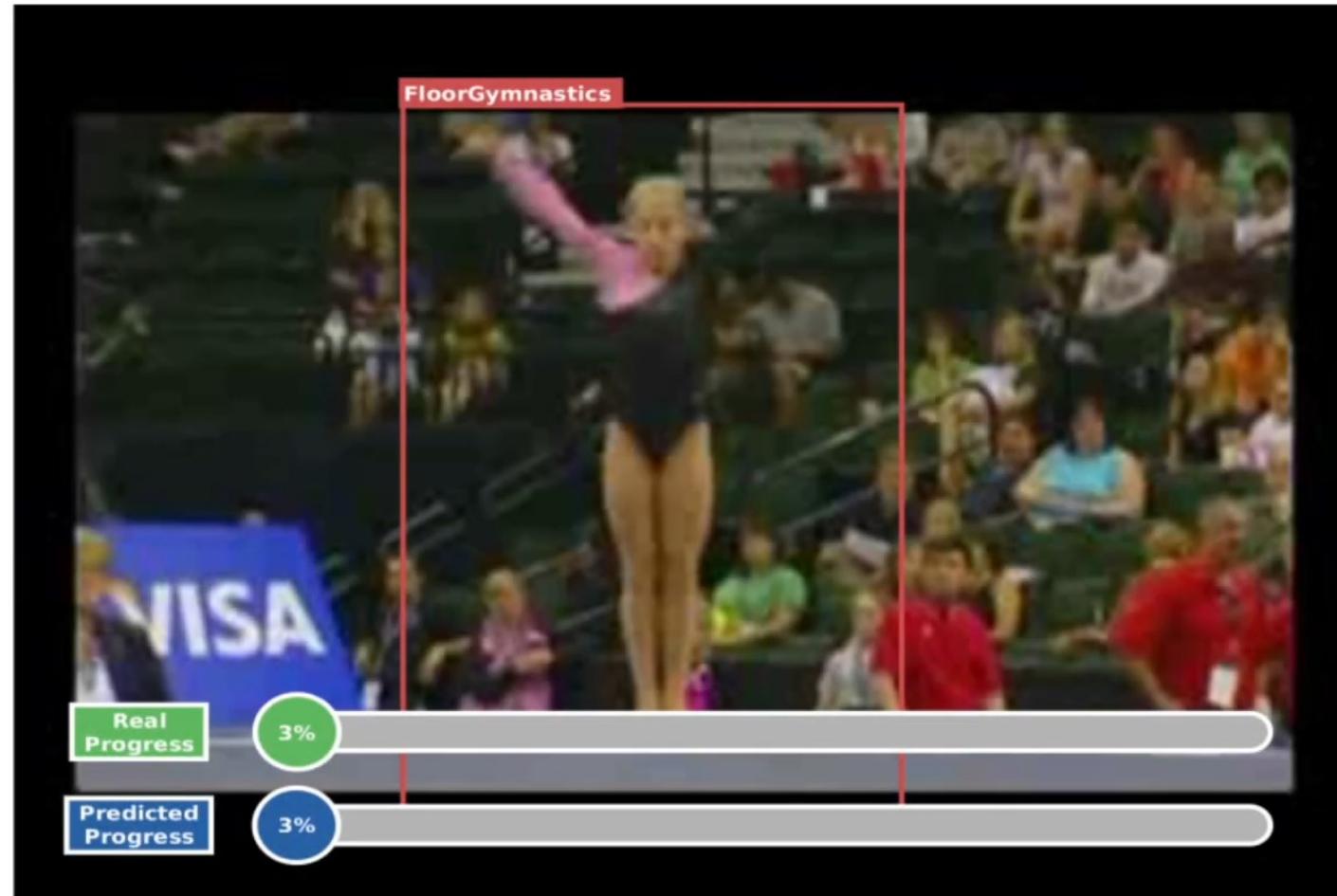
*Two stacked LSTM layers with 64 and 32 hidden units are fed with features representing the action region and global scene.*



$$p_i = \frac{t_i - t_S}{t_E - t_S} \in [0, 1]$$

# Am I Done? Predicting Action Progress in Videos

F. Becattini et al. ACM TOMM 2020



# Is a Cell State enough?



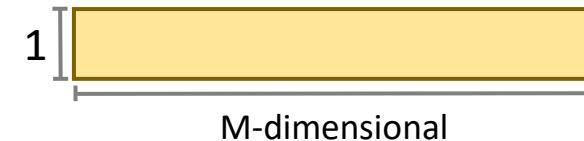
Knowledge is compressed into a single dense vector.

The memory is addressable as a whole.

It lacks the ability to address individual elements.

State to state transition is global: updating at each time step.

It lacks the ability to add new concepts from data streams  
without forgetting the previous learned ones



# Is Cell State enough?



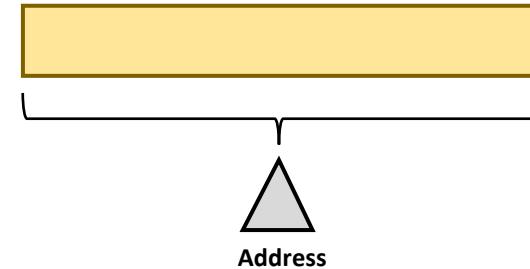
Knowledge is compressed into a single dense vector.

The memory is addressable as a whole.

It lacks the ability to address individual elements.

State to state transition is global: updating at each time step.

It lacks the ability to add new concepts from data streams  
without forgetting the previous learned ones



# Is Cell State enough?



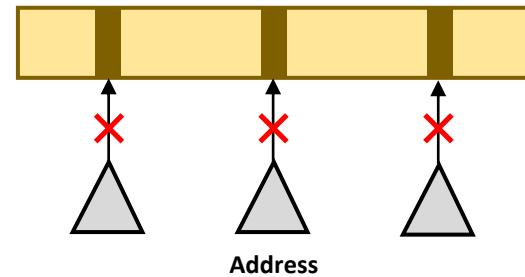
Knowledge is compressed into a single dense vector.

The memory is addressable as a whole.

**It lacks the ability to address individual elements.**

State to state transition is global: updating at each time step.

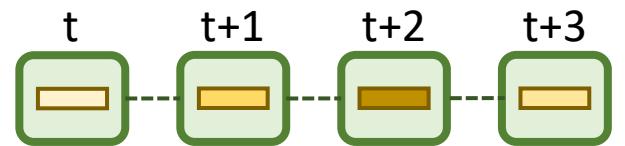
It lacks the ability to add new concepts from data streams without forgetting the previous learned ones



# Is Cell State enough?



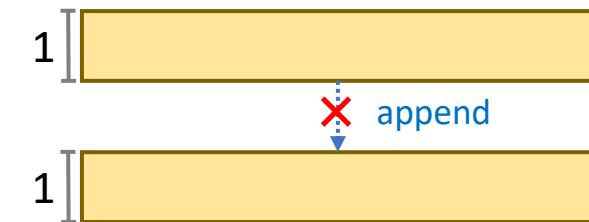
- Knowledge is compressed into a single dense vector.
- The memory is addressable as a whole.
- It lacks the ability to address individual elements.
- State to state transition is global: updating at each time step.**
- It lacks the ability to add new concepts from data streams without forgetting the previous learned ones



# Is Cell State enough?



- Knowledge is compressed into a single dense vector.
- The memory is addressable as a whole.
- It lacks the ability to address individual elements.
- State to state transition is global: updating at each time step.
- It lacks the ability to add new concepts from data streams without forgetting the previous learned ones



## PART 2 - *Learning algorithmic tasks*

NTM; MANN; End-to-End Memory Network; KV Memory Network; MANTRA  
*Memory Augmented Network models*

# Deeper AI tasks requirements

In complex tasks, rather than artificially increasing the size of the hidden state in the RNN or LSTM, we would like to arbitrarily increase the amount of knowledge we add to the model while making minimal changes to the model.

We can augment the model with an independent memory that acts as a *knowledge base* from which the network can read and write on demand. This benefits the following:

- Larger memory
- Memory increase should not require increasing number of parameters
- Memory compartmentalization
- Read/Write operations can follow algorithmic rules

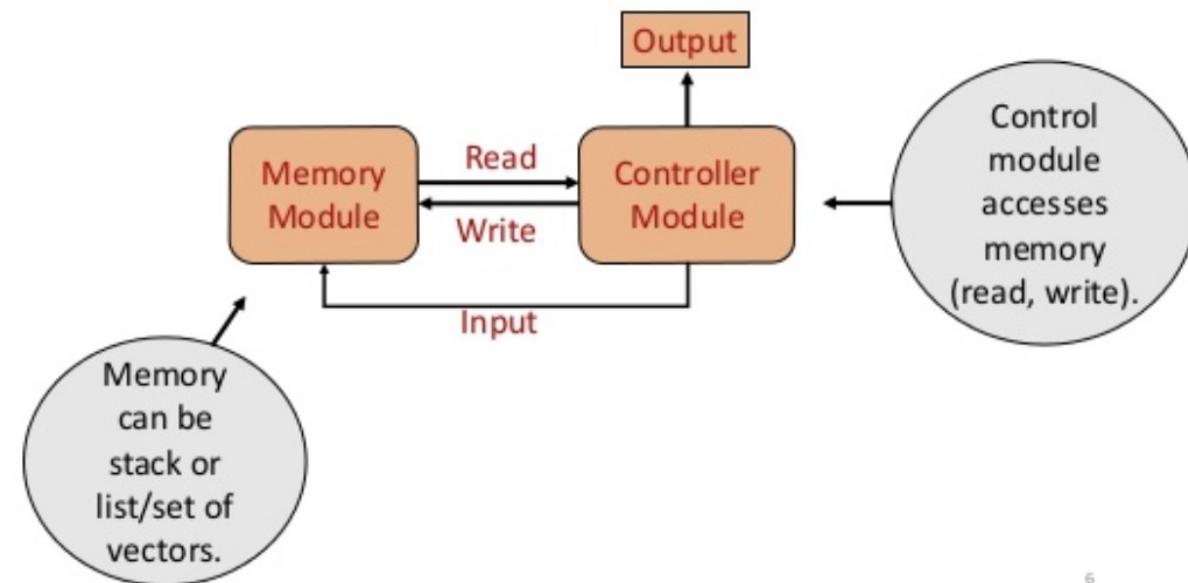
This is the Memory Augmented Neural Network (MAN/MANN) model.

# Memory Augmented Neural Networks

MANN model: *gives a neural network an external memory that acts as a knowledge base and the capacity to learn how to interact with it.*

The recurrence reads from a possibly large external memory multiple times before emitting a symbol.

Think of the Controller network as the CPU and the external memory as the RAM.



# Memory Augmented Neural Networks

In MANNs the Memory Controller is trained to write symbols in memory and to read what is necessary to produce the output.

Like an LSTM but, instead of incrementally creating a state, it keeps in memory a set of independent states.

Unlike LSTMs, MANNs encourage local changes in memory. This helps to find the structure in the training data and to generalize to sequences in algorithmic tasks.

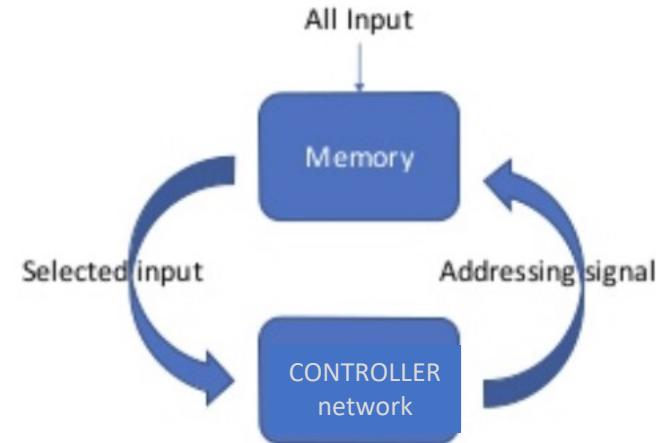
# Memory Augmented Neural Networks

RNN/LSTM



Inputs are fed to the LSTM one-by-one, in order. The LSTM has only one chance to look at an input symbol.

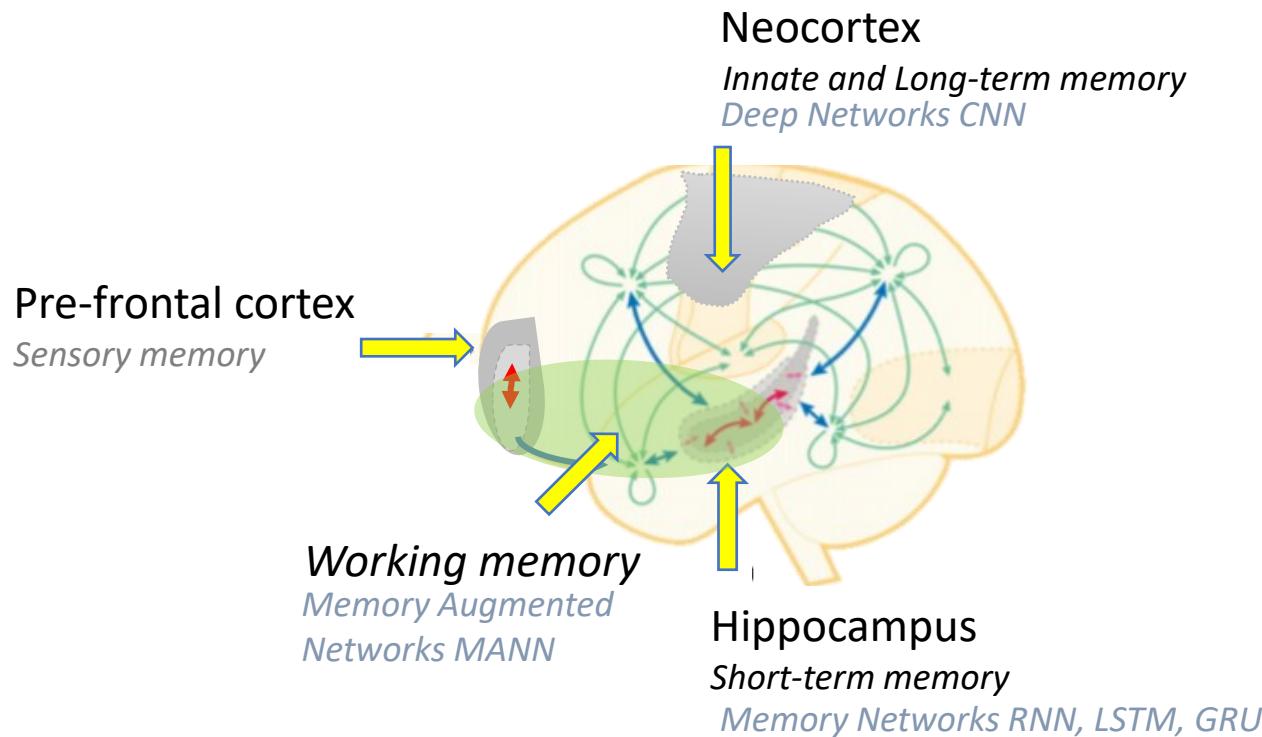
*Memory Augmented Neural Network*



Place all input symbols in memory and let the model decide which part to read next.

# Working Memory

While LSTM/GRU emulate human Short-term memory, MANNs emulate the process of the *Working Memory* in the human brain to some extent.



Working memory is the whole framework of retrieval structures and processes used for the temporary storage and manipulation of new information.

# Memory Augmented Neural Networks (MANN)



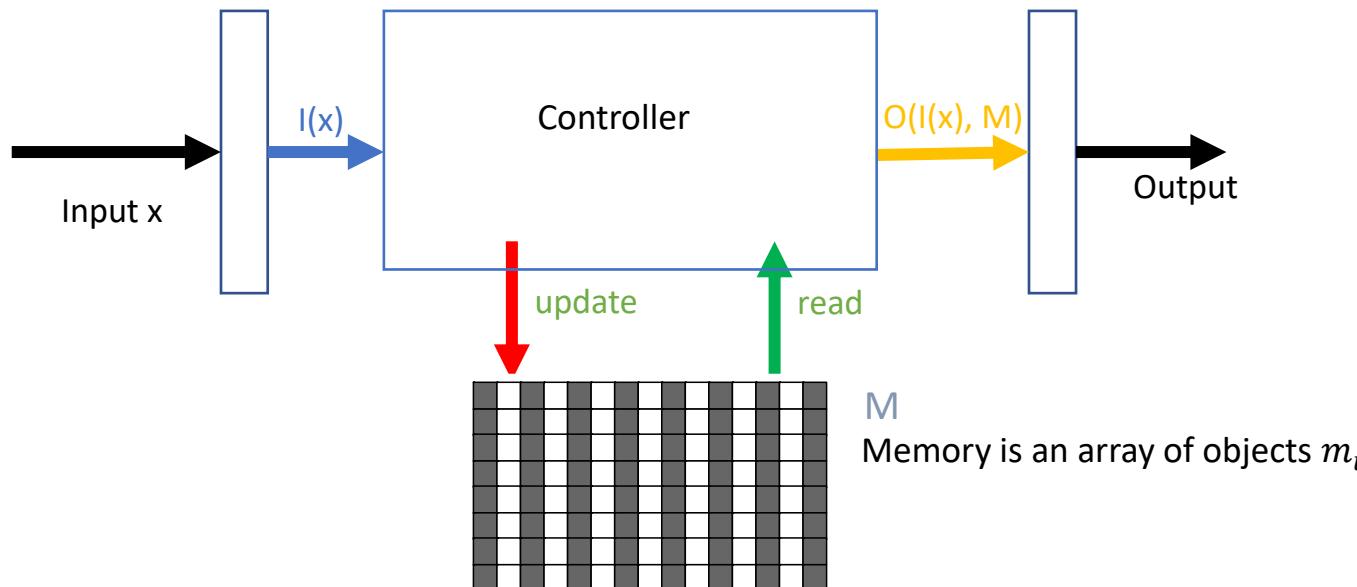
MANN models have an external memory and four components:

**I** *input feature map*: converts the input to the internal feature representation

**G** *generalization*: updates old memories given a new input

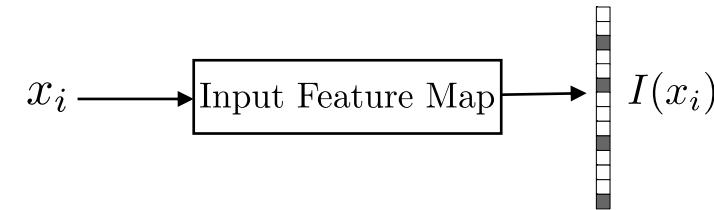
**O** *output feature map*: produces a new output given the current input and memory state

**R** *response*: converts the output into the desired response format

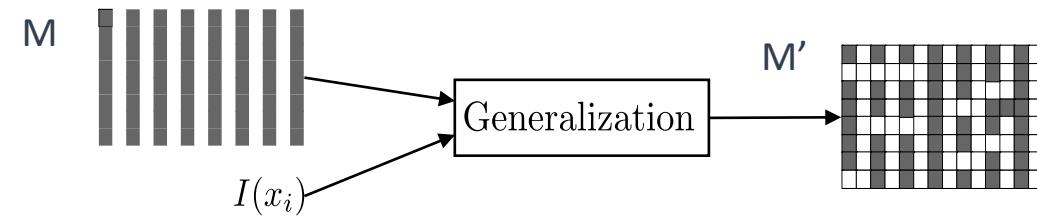


# The IGOR Model

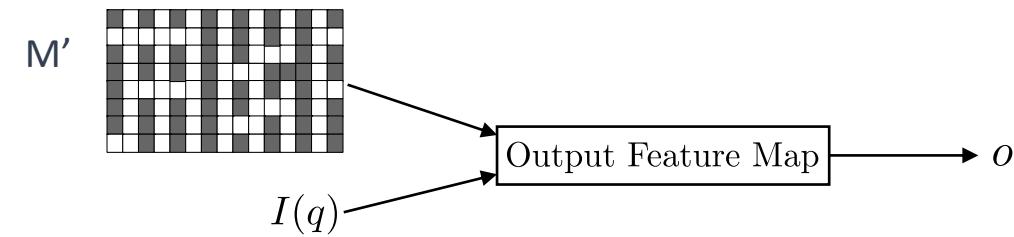
I      **Input:** converts a sequence of input sentences  $x_i$  into the internal feature representation.



G      **Generalization:** updates the old Memory content (compress and generalize memory for future use)



O      **Output:** given an input and the memory state, computes the output in the feature representation space



R      **Response:** decodes the output features to generate the answer in the desired format.



# Memory Network models

## Memory networks

Memory is a single hidden state vector that encodes all the temporal information.

Memory is addressable as a whole (all the past information is encoded in the state vector).

State to state transition is unstructured and global.

Find some structure in the training data.

The number of parameters is tied to the size of the hidden state.

## Memory Augmented Networks

Add an external memory matrix with increased storage capacity.

Memory is element-wise addressable (relevant items of information can be accessed selectively). Relies on attention to work.

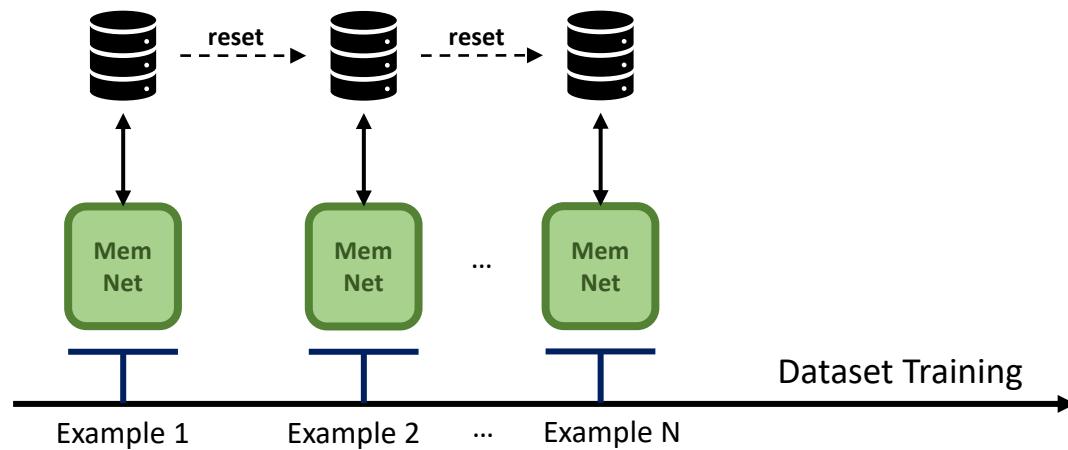
State to state transitions are obtained through read/write operations.

Find the structure in the training data, but also generalize to long sequences in algorithmic tasks.

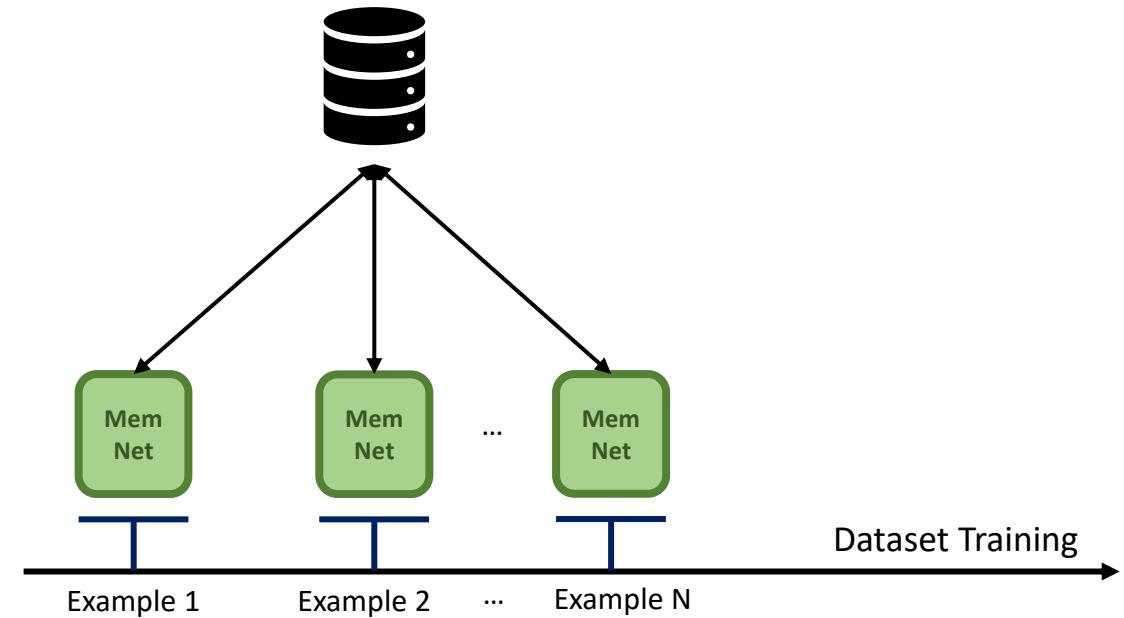
The number of parameters is not tied to the size of the memory. Increasing the number of memory slots will not increase the number of parameters.

# Episodic vs Persistent Memory

Episodic



Persistent



# Episodic memories

Typically in MANNs the memory is just episodic.

For each example presented to the network we start from an empty memory and perform a sequence of read / write steps.

The Controller is trained with the Task Loss.

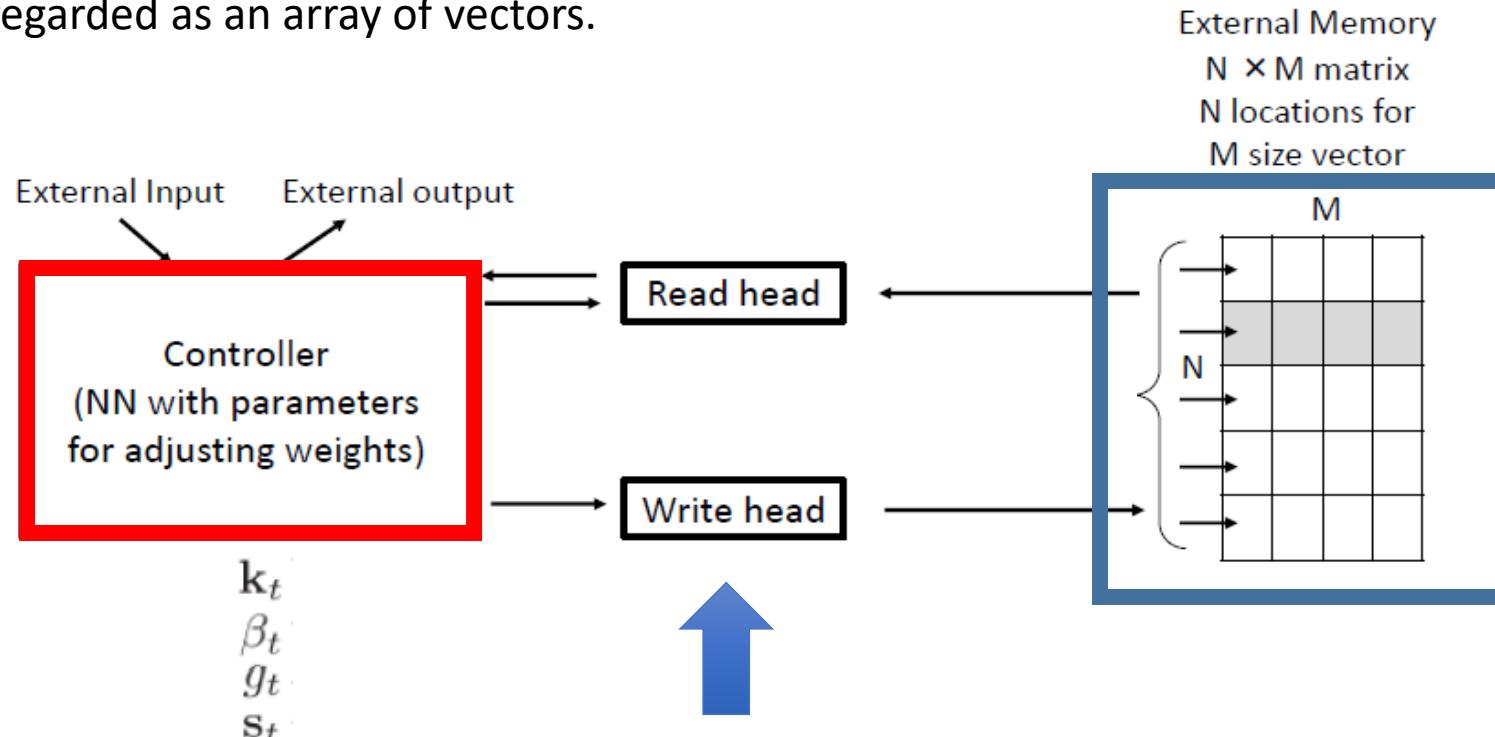
# Neural Turing Machine

A. Graves, G. Wayne, I. Danihelka, 2014

*Neural Turing Machine (NTM)* is a MANN that learns to read and write data from the external memory at different time steps to solve a given task.

NTM contains:

- a *Network Controller* that is responsible for making the interface between the input sequence and the output representation and the memory through read and write heads.
- a *Memory Bank* regarded as an array of vectors.



The heads, with the feature generated by the controller, compute the addressing to read/write in memory

Credits S. Malekmohammadi

# Neural Turing Machine

A.Graves, G. Wayne, I. Danihelka 2014

The goal of NTM is to learn an algorithmic task.

NTM learns:

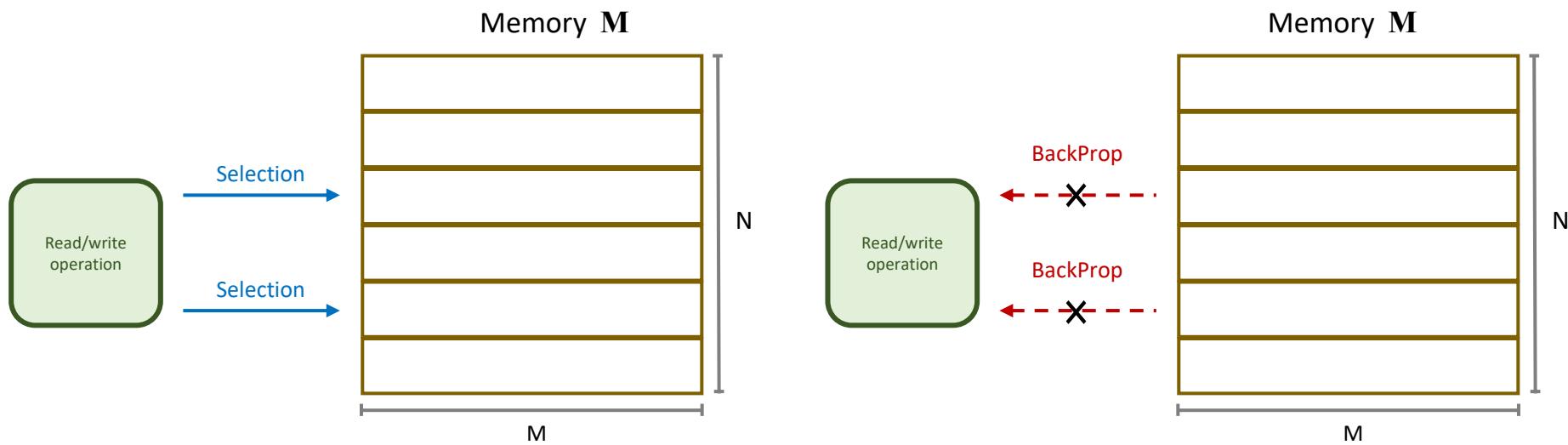
- What to write to memory
- When to write to memory
- When to stop writing
- Which memory cell to read from
- How to convert result of read into final output

# Neural Turing Machine

A.Graves, G. Wayne, I. Danihelka 2014

General issue: how to allow addressing in memory? Sometimes we want to select a single element, other times a small subset or other times the whole memory.

The operations *argmax* or *select index* are not differentiable so they cannot be exploited to train with gradient descent.

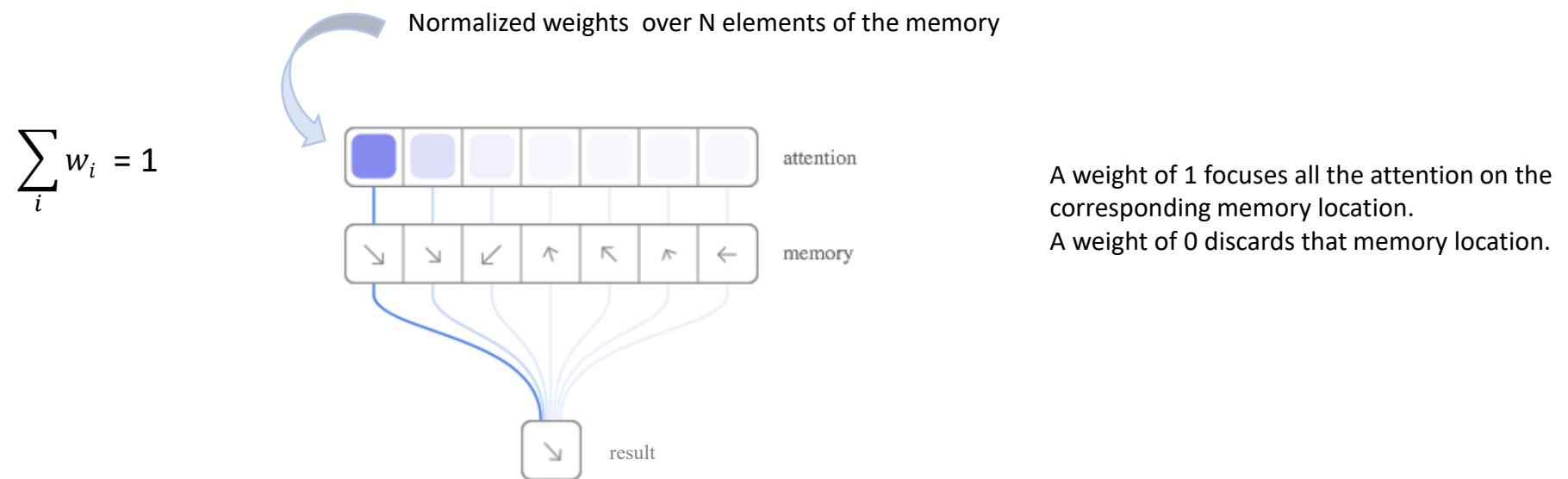


# NTM blurry operations

*Blurry operations:* interact to a greater or lesser degree with all the elements in memory rather than addressing a single or few elements directly.

The degree of blurriness is determined by an “attentional focus” mechanism that constrains each Read and Write operation to interact with a small portion of the memory while ignoring the rest.

The portion of memory brought into attentional focus is determined by normalized weights emitted by the heads.



# NTM addressing mechanisms

Attention weights are generated by the *Controller network* with two distinct addressing mechanisms with complementary facilities:

## *Content Based addressing*

Focuses attention on locations based on the similarity between the current memory values and values emitted by the controller.

Example of application: VQA - score sentences by similarity with a question.

Weights as softmax of similarity scores.

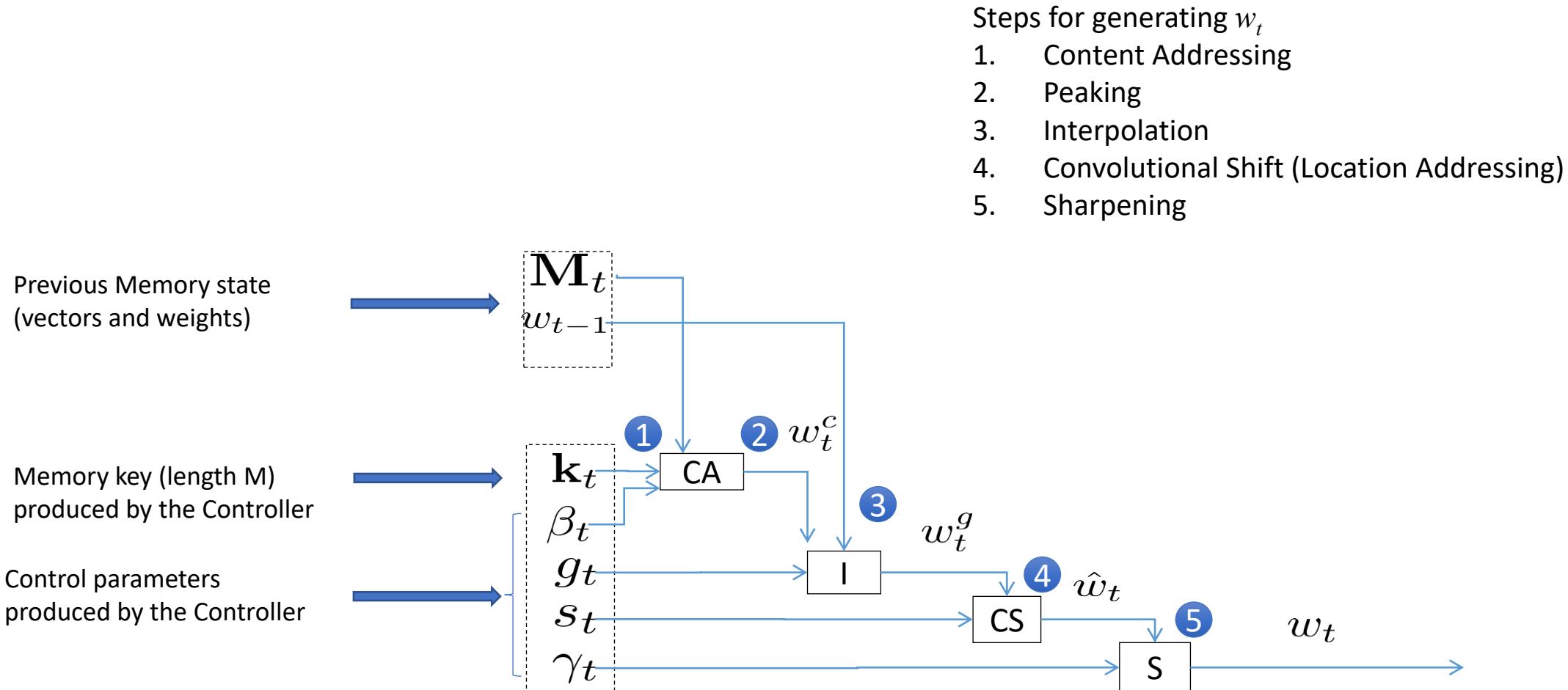
## *Location Based addressing*

Focuses attention on locations based on the address of the location.

Example of application: Copy Task - move to address ( $i+1$ ) after writing to index ( $i$ )

Weights  $\approx$  Transition probabilities.

# NTM addressing steps



Steps for generating  $w_t$

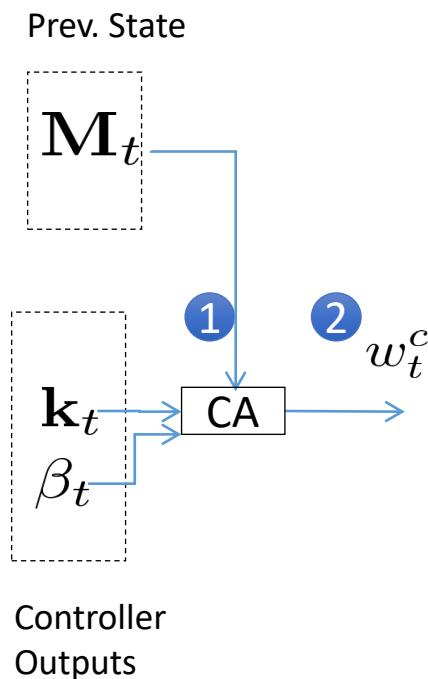
1. Content Addressing
2. Peaking
3. Interpolation
4. Convolutional Shift (Location Addressing)
5. Sharpening

# Content-based addressing and peaking

## Step 1, 2: Content Addressing and peaking

The Controller extracts feature  $\mathbf{k}_t$  from the input using a deep network (an LSTM or a feedforward network) and uses it to compute the addressing weights.

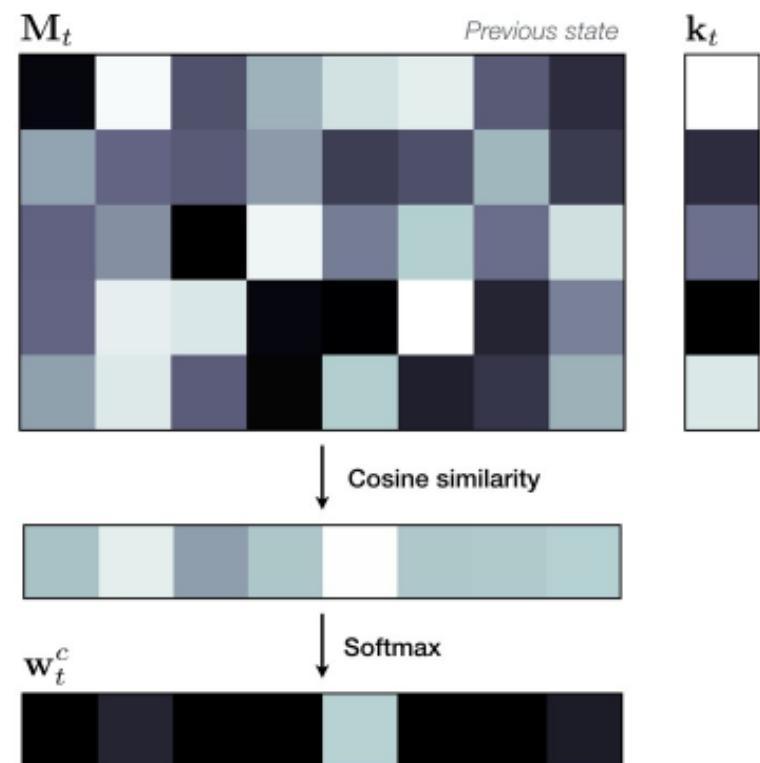
$\beta_t$  is the temperature of the softmax and is used to amplify or attenuate the difference in scores.



$$w_t^c(i) = \frac{\exp(\beta_t \cdot \langle \mathbf{M}_t(i), \mathbf{k}_t \rangle)}{\sum_i \exp(\beta_t \cdot \langle \mathbf{M}_t(i), \mathbf{k}_t \rangle)}$$

### Content addressing

$$w_t^c(i) \leftarrow \text{softmax}(\beta_t \cdot K[\mathbf{k}_t, \mathbf{M}_t(i)])$$



# Content-based addressing

## *Content-based addressing*

Each head (whether read or write) produces a length M *key vector*  $\mathbf{k}_t$  that is compared to each vector  $\mathbf{M}_t(i)$  by a cosine similarity measure  $K$  and produces a normalised weighting  $w_t$  applying a softmax function on the score  $K$ .

$$w_t^c(i) \leftarrow \frac{\exp\left(\beta_t K[\mathbf{k}_t, \mathbf{M}_t(i)]\right)}{\sum_j \exp\left(\beta_t K[\mathbf{k}_t, \mathbf{M}_t(j)]\right)}.$$

$$K[\mathbf{u}, \mathbf{v}] = \frac{\mathbf{u} \cdot \mathbf{v}}{\|\mathbf{u}\| \cdot \|\mathbf{v}\|}$$

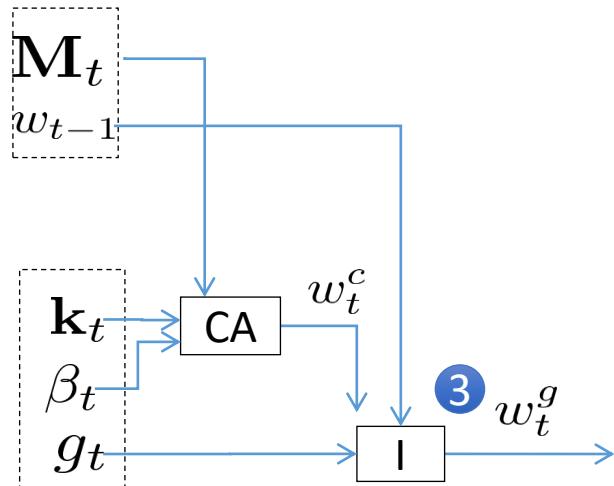
# Interpolation

## Step 3: Interpolation

In content-based addressing focus is only based on the current input. Interpolation also accounts for previous attention.

$$w_t^g = g_t w_t^c + (1 - g_t) w_{t-1}$$

Prev. State

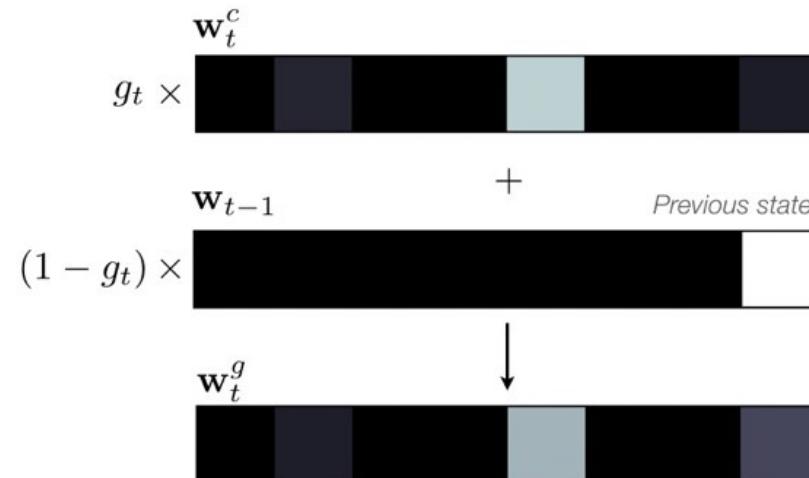


Controller  
Outputs

It results into a new weight that accounts for both the content-based focus and the focus in the last timestep.

### Interpolation

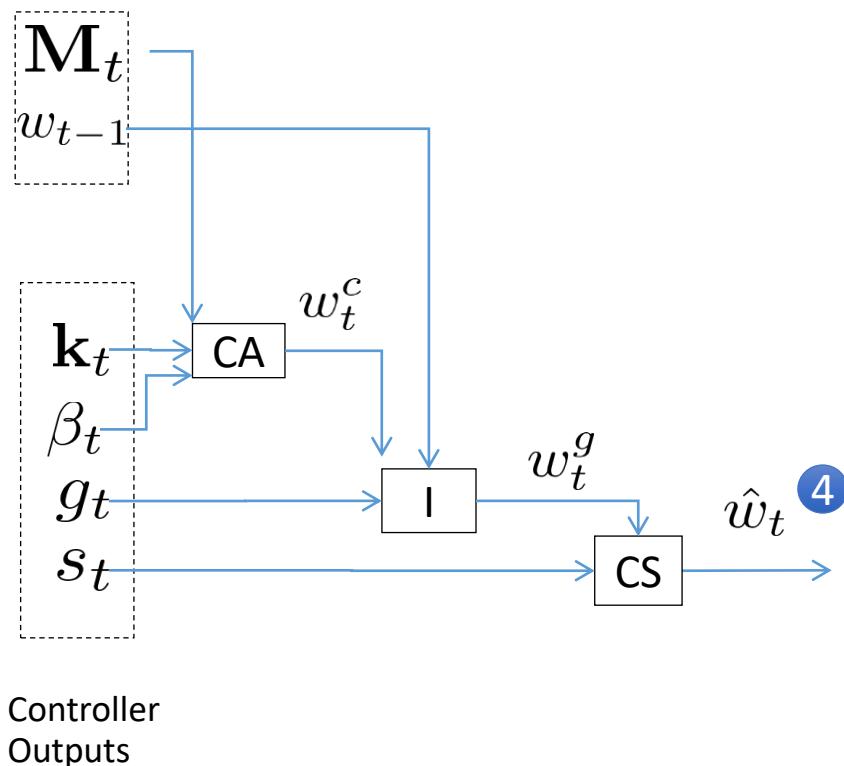
$$\mathbf{w}_t^g \leftarrow g_t \mathbf{w}_t^c + (1 - g_t) \mathbf{w}_{t-1}$$



# Convolutional Shift

$$\hat{w}_t(i) = \sum_{j=0}^{N-1} w_t^g(j) s_t(j - i)$$

Prev. State                             $s_t$  normalized shift distribution



## Step 4: Convolutional Shift

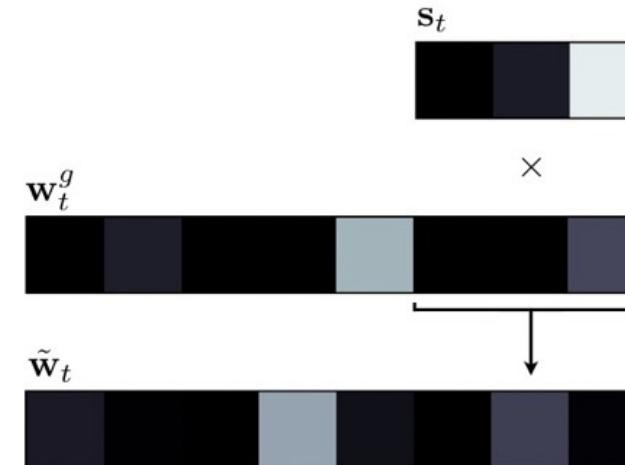
Convolutional shift handles a shift of focus. It smoothly shifts the weights left or right.

It creates a focus from a range of rows where the convolution function is a linear weighted sum of rows.

This mechanism allows NTM to perform basic algorithms like copy and sort. It is very close to the head shifting in a classical Turing Machine.

### Convolutional shift

$$\tilde{w}_t(i) \leftarrow \sum_j w_t^g(j) \cdot s_t(i - j)$$

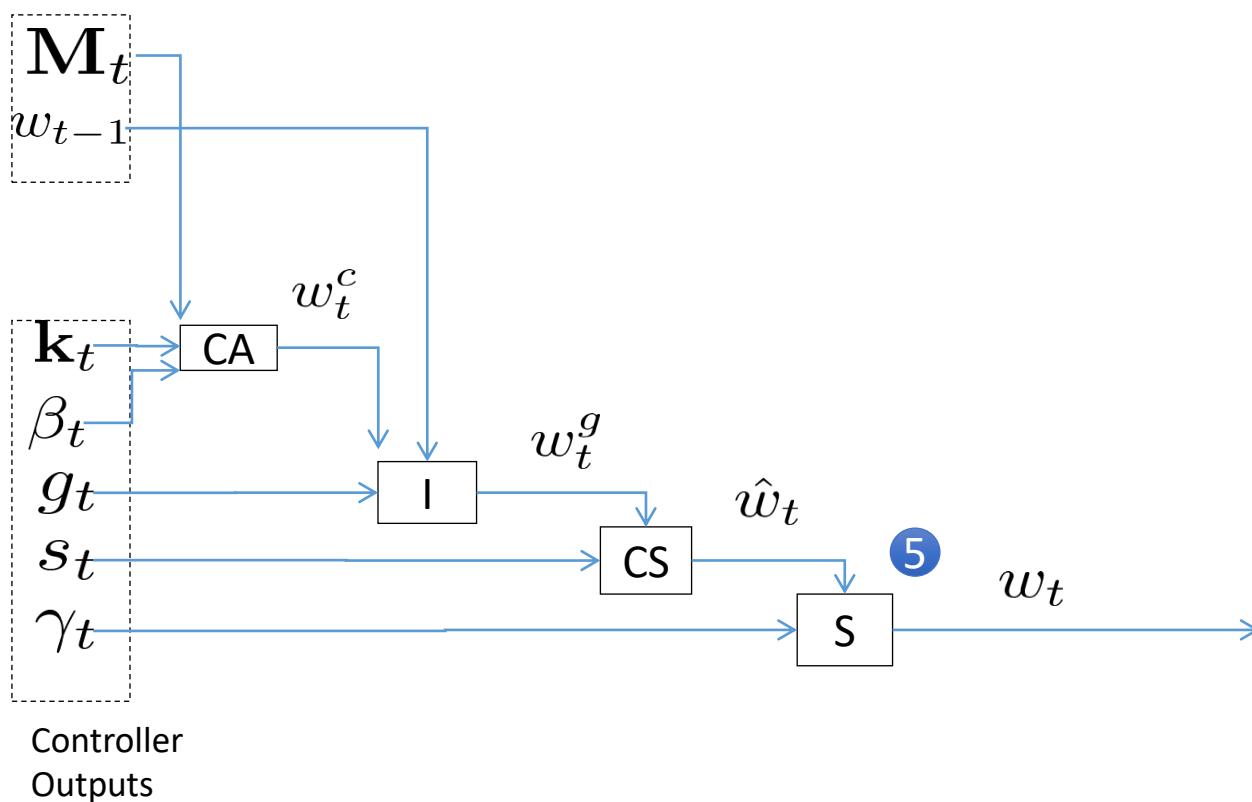


# Sharpening

Step 5: Sharpening

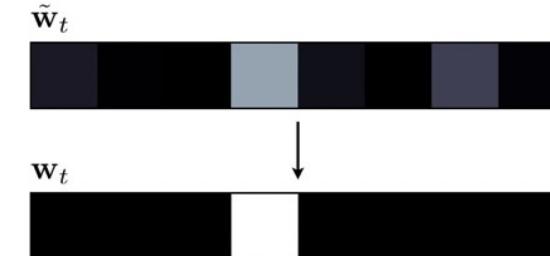
Uses  $\gamma_t$  to sharpen as:  $w_t(i) = \frac{\hat{w}(i)^{\gamma_t}}{\sum_i \hat{w}(i)^{\gamma_t}}$

Prev. State



Sharpening

$$w_t(i) \propto \tilde{w}_t(i)^{\gamma_t}$$



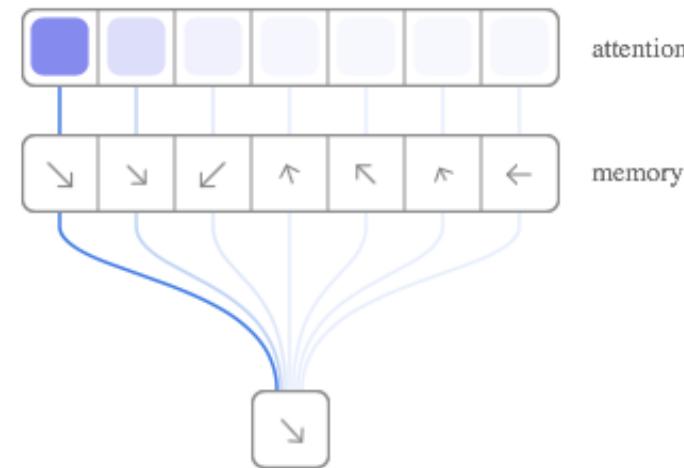
# Blurry Read Operation

Read result is a weighted sum

$$r_t = \sum_{i=0}^{N-1} w_t(i) \mathbf{M}_t(i)$$

Given:

- memory matrix  $\mathbf{M}_t$  of size  $N \times M$
- $w_t$  (weight vector) of length  $N$
- $t$  (time index)



# Blurry Write Operation

Write is the combination of erase and add operations:

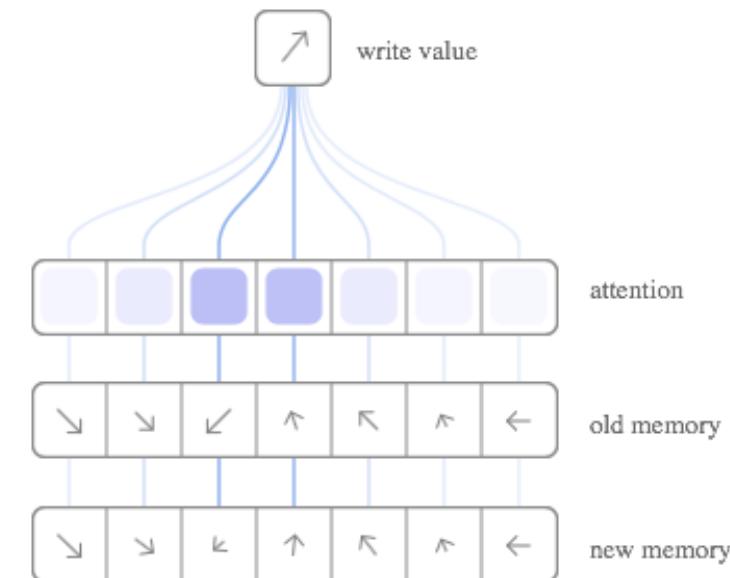
- $\mathbf{e}_t$  is an *erase vector*
- $\mathbf{a}_t$  is a length M *add vector*

$$\mathbf{M}_t(i) = \underbrace{\mathbf{M}_{t-1}(i)(1 - w_t(i)\mathbf{e}_t)}_{\text{Erase Component}} + \underbrace{w_t(i)\mathbf{a}_t}_{\text{Add Component}}$$

The memory writing process composes of previous state and new input and implements a similar mechanism as the forget and input gates in LSTM

Given:

- memory matrix  $\mathbf{M}_t$  of size  $N \times M$
- $w_t$  (weight vector) of length  $N$
- $t$  (time index)



# NTM Controller design

The attention mechanism decides which parts of the memory the model must focus on.

The controller extracts features  $k_t$  from the input using a deep network and uses it to compute the attention weights.

The NTM Controller can be either a Feed-forward network or an LSTM

- Feed-forward: faster, more transparency and interpretability about the learned function
- LSTM: more expressive power, does not limit the number of computations per time step

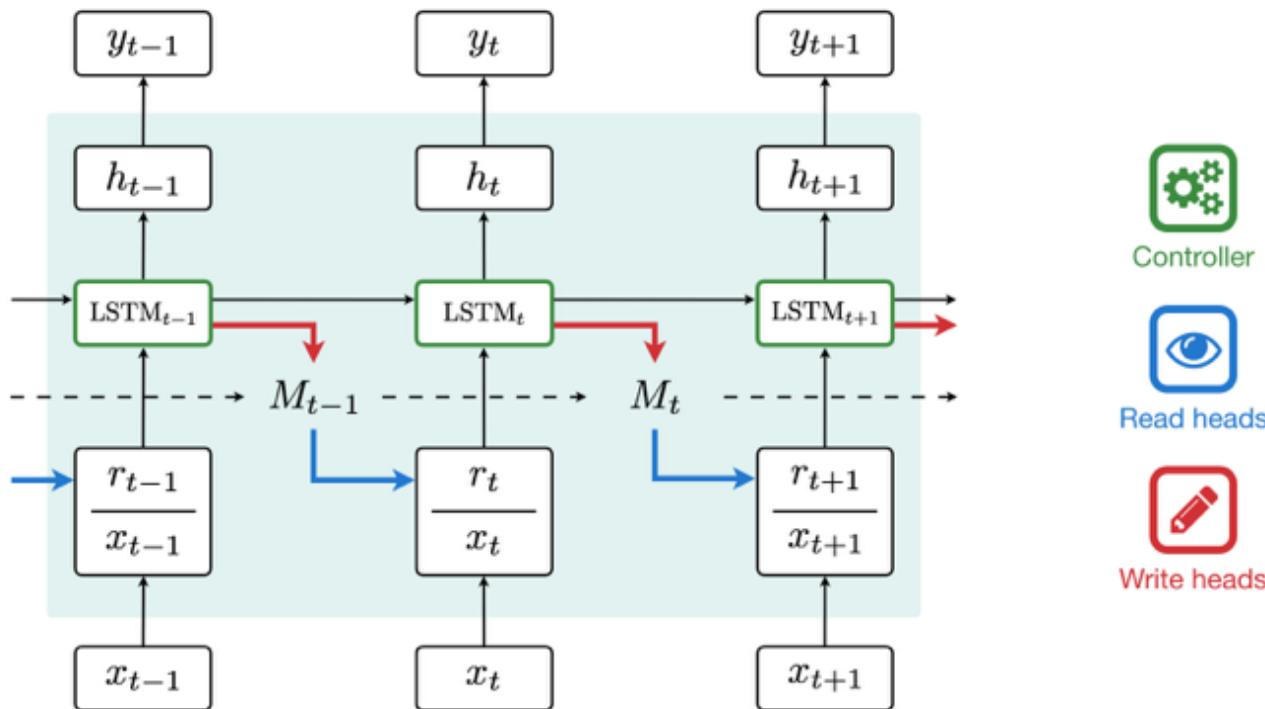
Comparing the controller to the CPU in a digital computer and the memory matrix to the RAM, then the hidden activations of the recurrent controller are akin to the registers in the processor.

# NTM Controller unfolded

Once the head has updated its weight vector, it is ready to operate on the memory.

If it is a read head, it outputs a weighted combination of the memory locations: the *read vector*. This is then fed back to the controller at the following time-step.

If it is a write head, the content of the memory is modified according to weights with an *erase* and an *add vector*, both produced by the controller.



$$\mathbf{r}_t \leftarrow \sum_i w_t(i) \mathbf{M}_t(i),$$
$$\mathbf{M}_t(i) = \mathbf{M}_{t-1}(i)(1 - w_t(i)\mathbf{e}_t) + w_t(i)\mathbf{a}_t$$

# End-to-end training

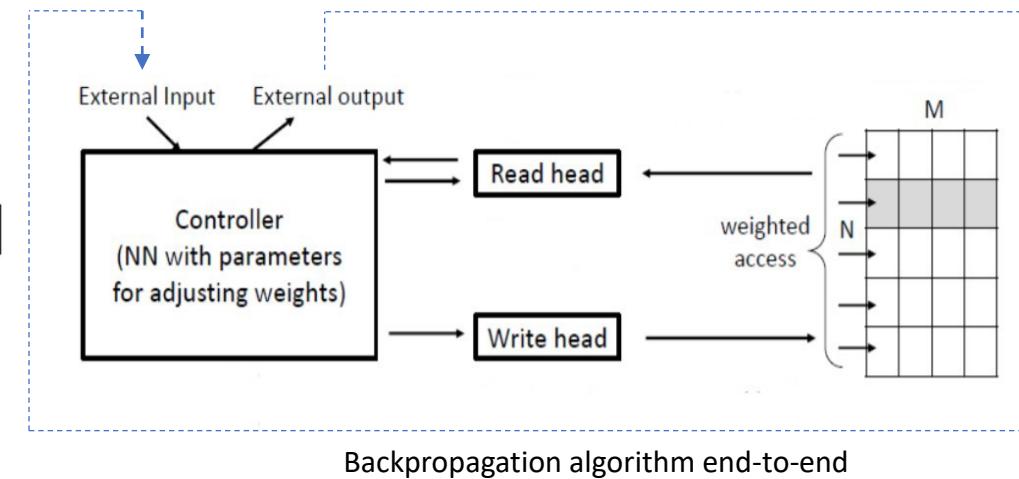
Read and write operations are made of products between attention weights and memory. Both are differentiable with reference to both the memory and the attention weights.

This leads to a fully differentiable model.

It is possible make an end-to-end training with SGD/RMSProp/Adam Optimizers.

Reading     $\mathbf{r}_t \leftarrow \sum_i w_t(i) \mathbf{M}_t(i),$

Writing     $\tilde{\mathbf{M}}_t(i) \leftarrow \mathbf{M}_{t-1}(i) [1 - w_t(i) \mathbf{e}_t]$   
 $\mathbf{M}_t(i) \leftarrow \tilde{\mathbf{M}}_t(i) + w_t(i) \mathbf{a}_t$



# NTM applications

The goal of NTM is to learn an algorithm: taking input and output and learning algorithms that map from one to the other, e.g. taking an input and copying it.

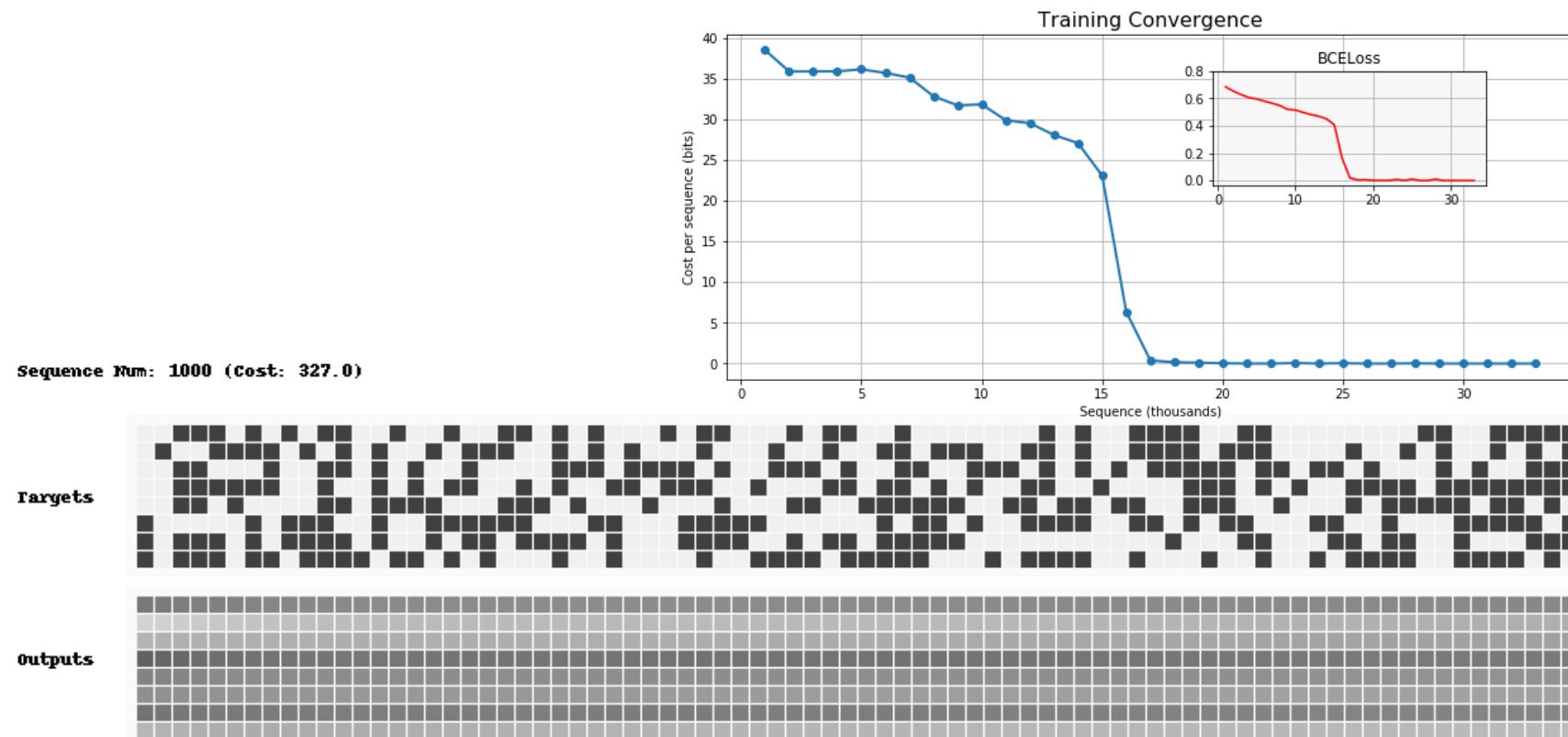
We feed the NTM with random inputs, with the corresponding expected outputs from the algorithm we intend to learn.

No prior knowledge on the nature of the algorithmic task is given to the NTM.

# NTM copy task

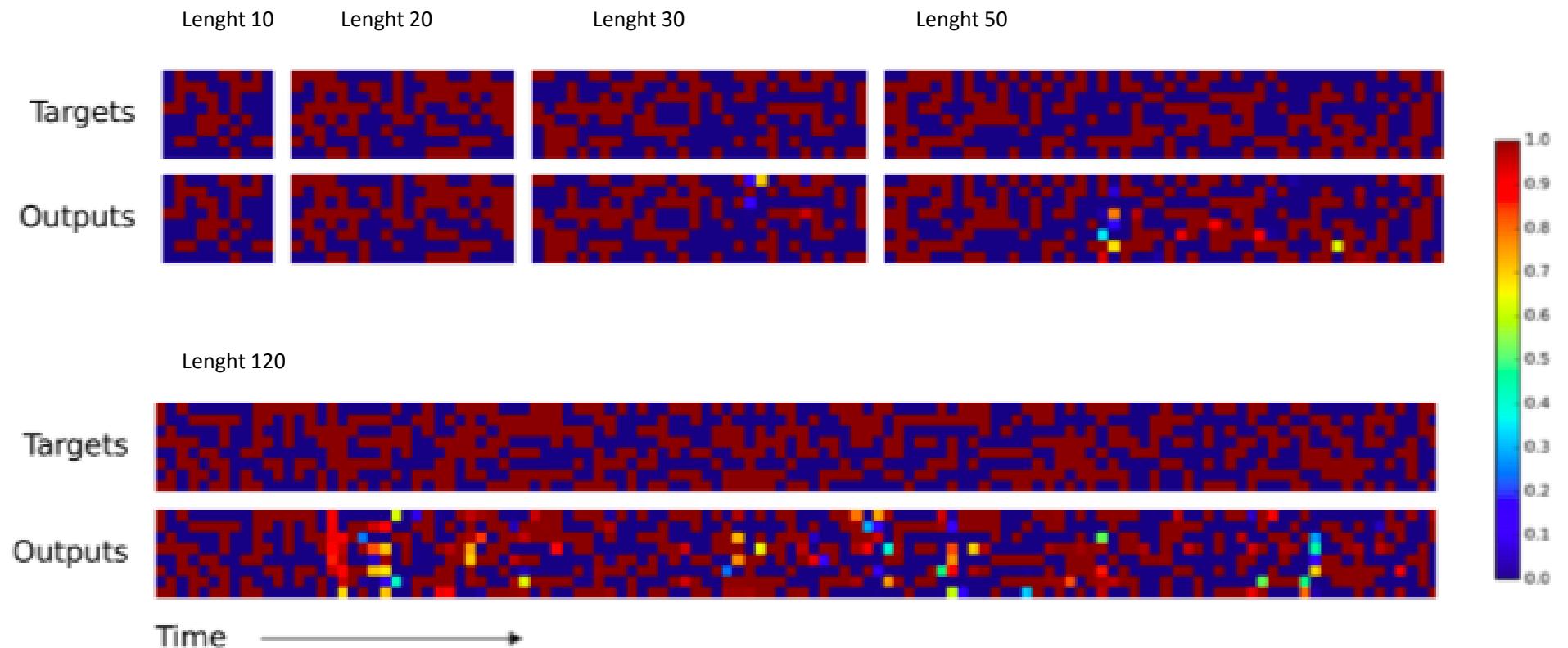
Copying task: the output should be the same 0, 1 pattern of the input. The network will store the input sequence in the memory and then it will read it back from the memory.

The network is trained with sequences of length 1 to 20 and performs well even if the input length is 80



# NTM copy task

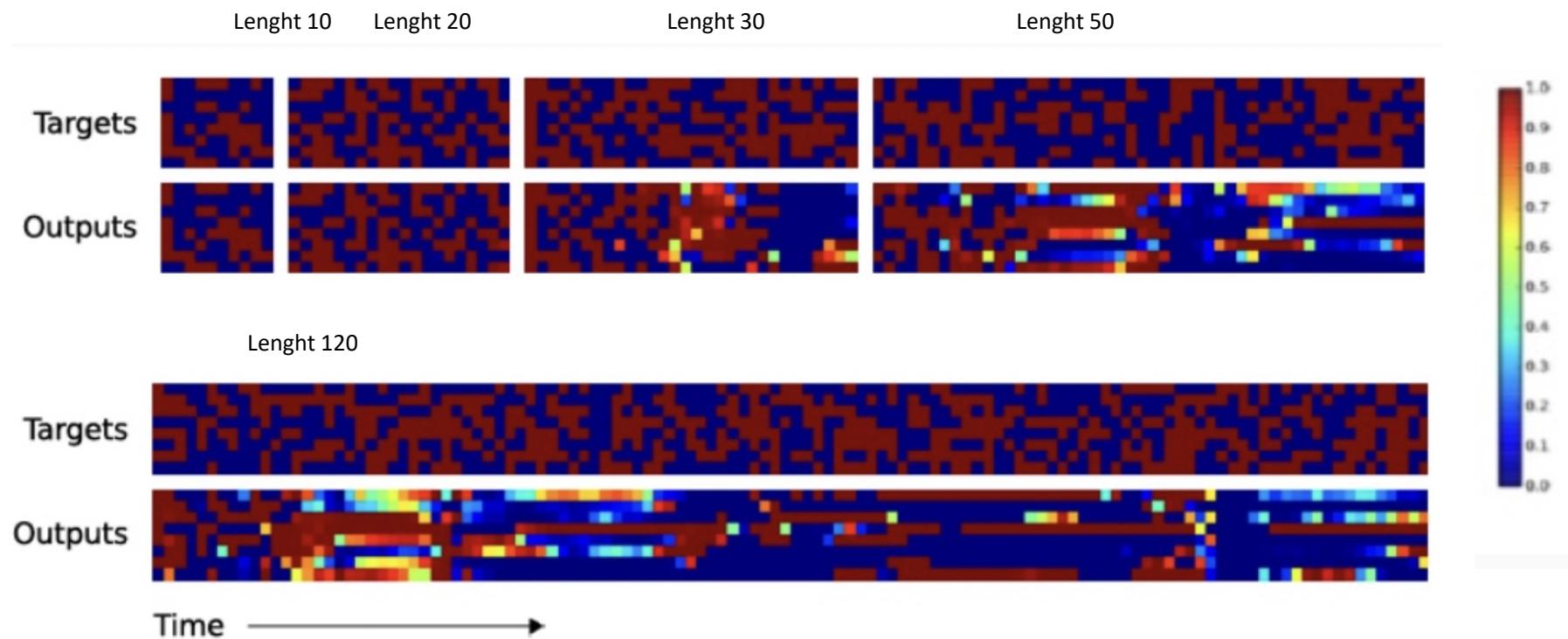
The NTM is able to generalize to sequences of any length, including sequences longer than what it saw during training.



Copy task: NTM trained on sequences of lenght 20

# LSTM copy task

The same task with LSTM



Copy task: NTM trained on sequences of length 20

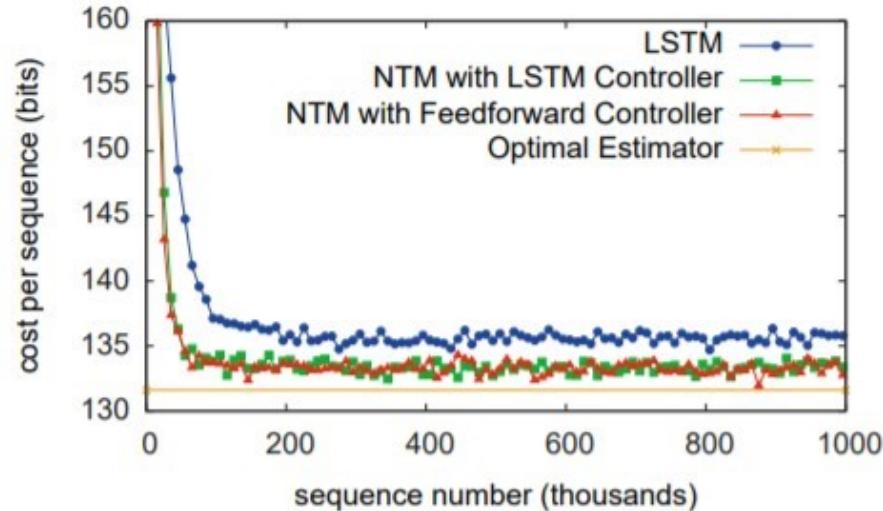
# NTM tasks

Task	Network Size		Number of Parameters	
	NTM with LSTM controller *	LSTM *	NTM with LSTM controller *	LSTM *
Copy	3 x 100	3 x 256	67K	1.3M
Repeat Copy	3 x 100	3 x 512	66K	5.3M
Associative Recall	3 x 100	3 x 256	70K	1.3M
N-grams	3 x 100	3 x 128	61K	330K
Priority Sort	2 x 100	3 x 128	269K	385K

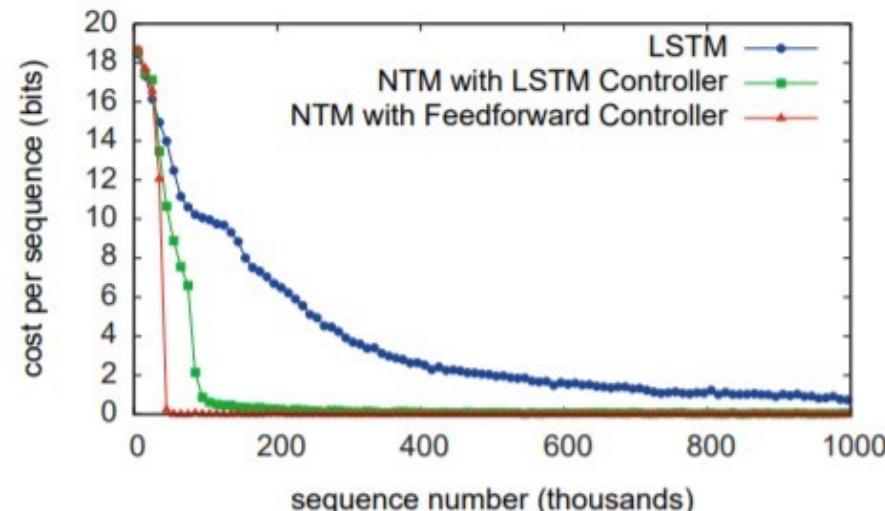
\* 3 stacked LSTM

# NTM tasks

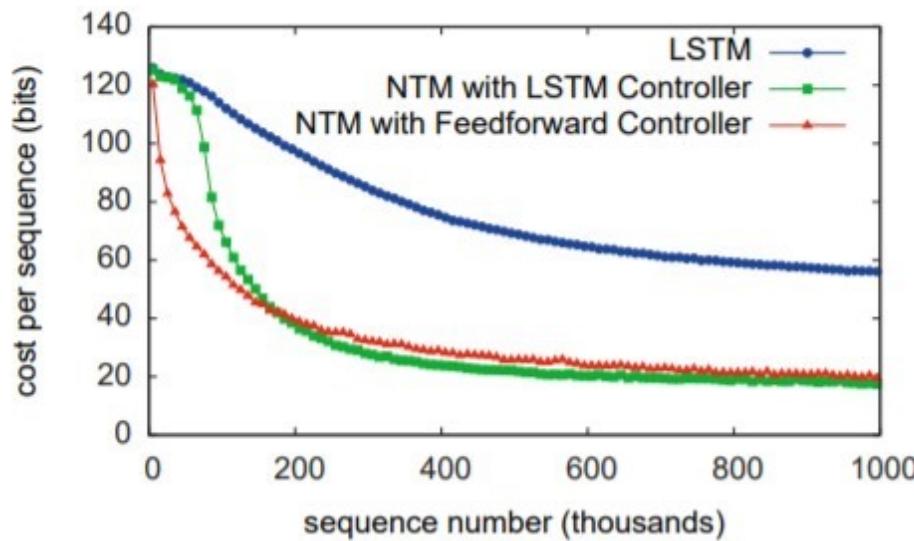
## Dynamic N-gram



## Associative recall



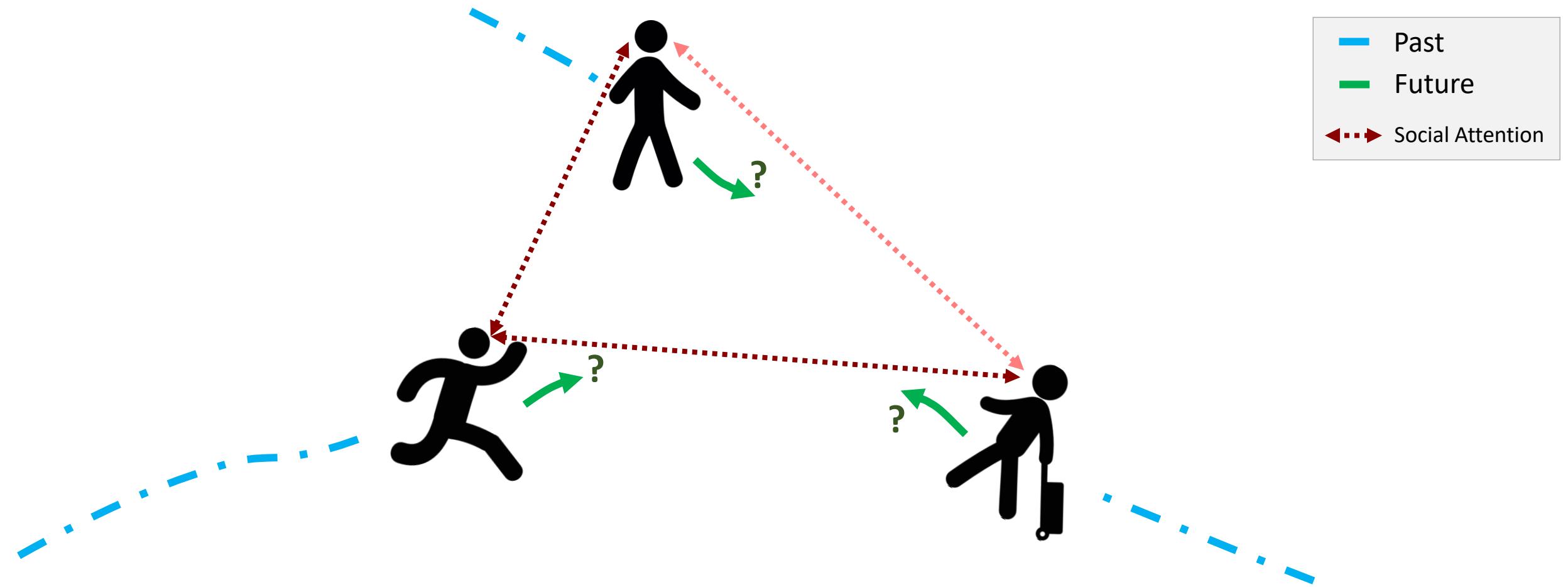
## Priority Call



# NTM in practice

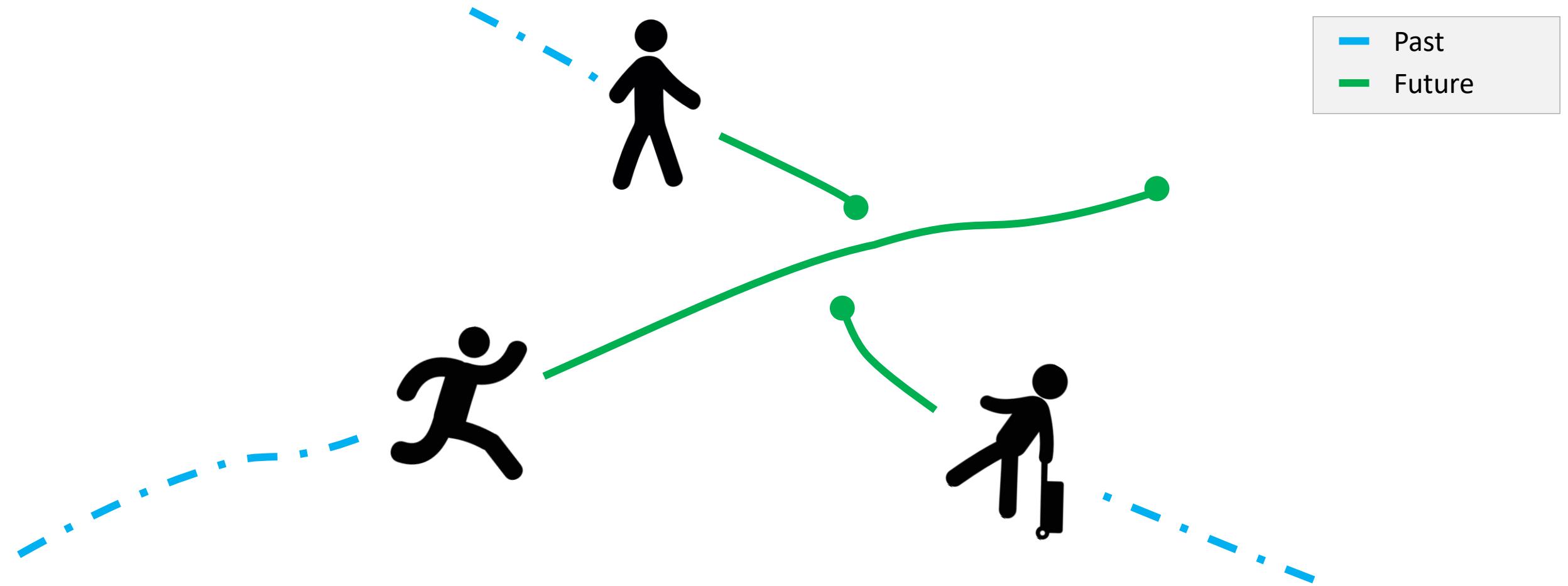
<https://colab.research.google.com/drive/1I7-va-TjO5BW39Savzba5q6uhTu-gZPA?usp=sharing>

# Application: Social Reasoning for Pedestrian Trajectory Prediction



Is understanding social interaction important?

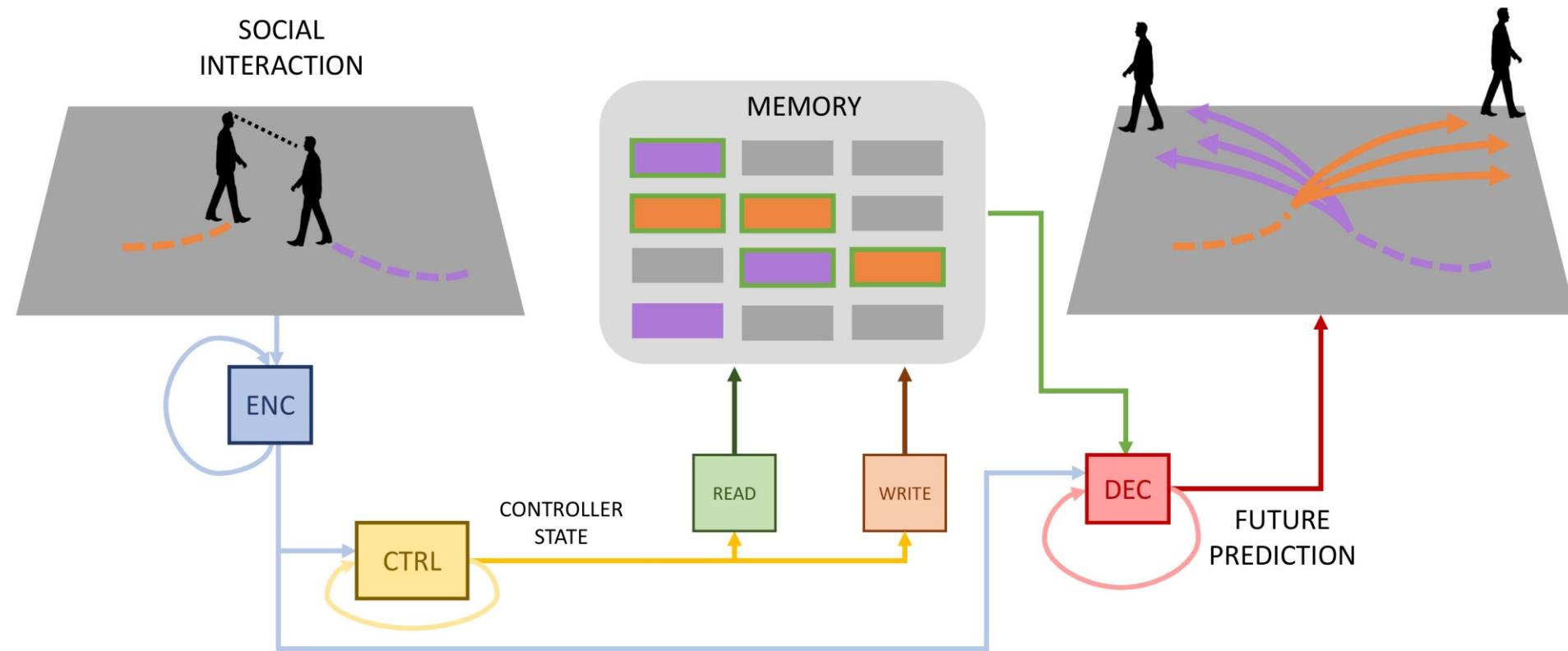
# Application: Social Reasoning for Pedestrian Trajectory Prediction



Yes, the pedestrians move observing also the behaviors of others around them.

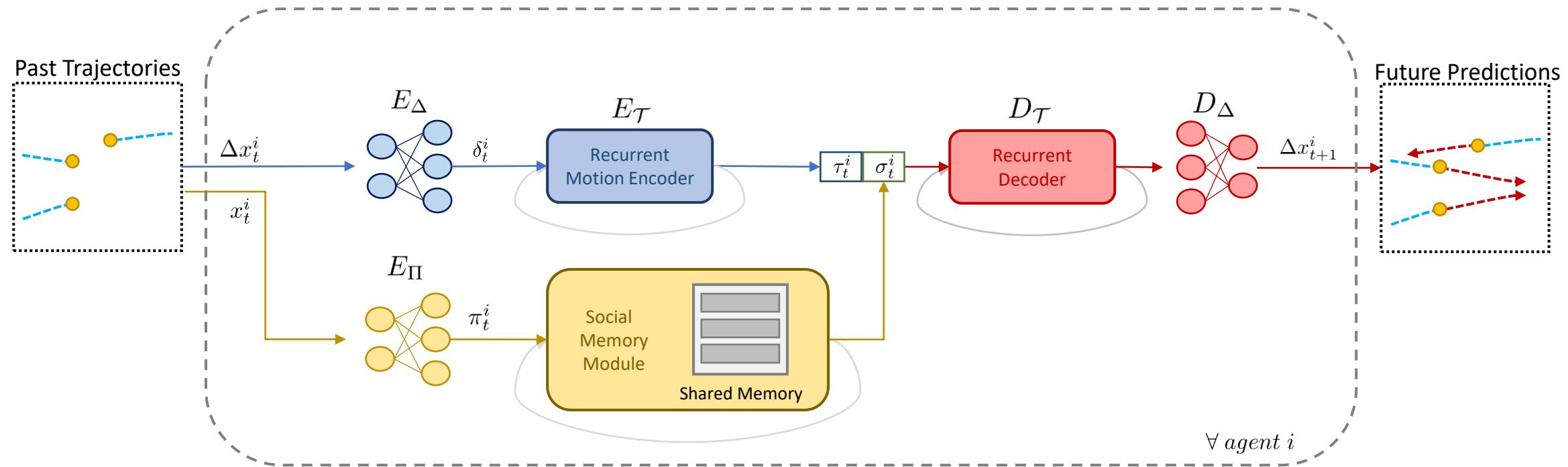
# SMEMO: Social MEMOry for pedestrian trajectory prediction

F. Marchetti *et al.*, 2022



SMEMO is equipped with an external storage that acts as a episodic working memory where the past information from multiple agents in a scene can be stored and later recalled to make predictions.

# SMEMO: Architecture

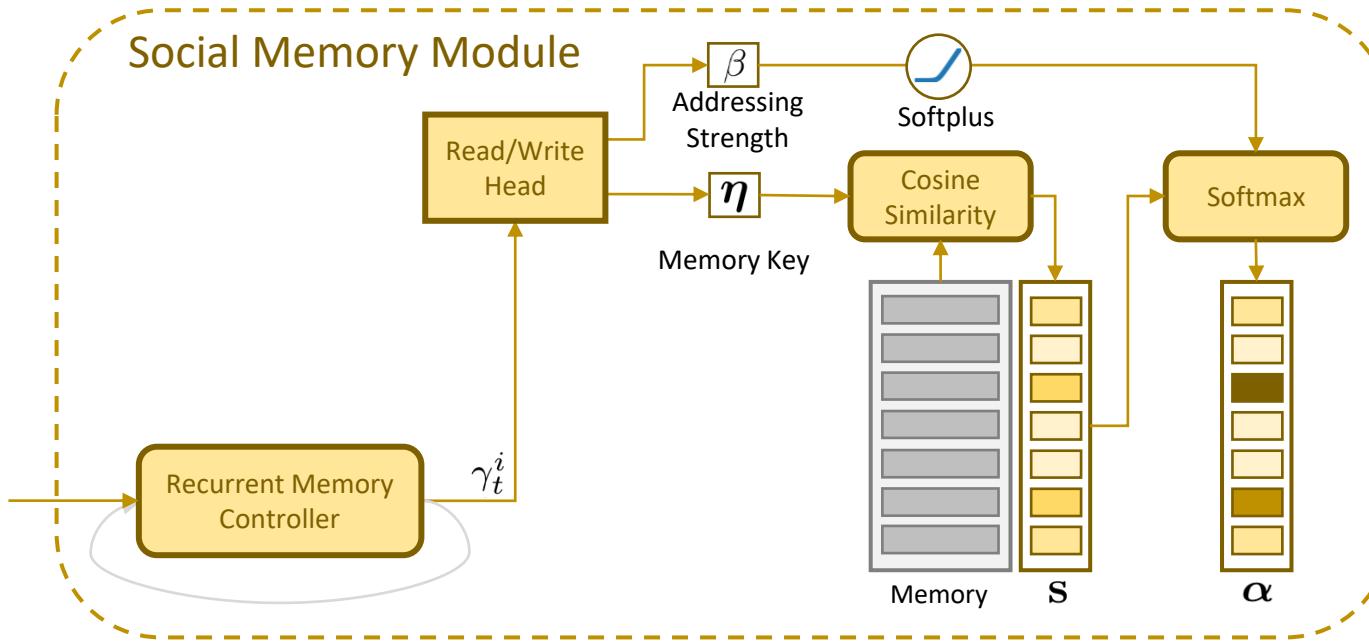


The upper stream models motion of single agents.

The lower stream interacts with the working memory and stores social information.

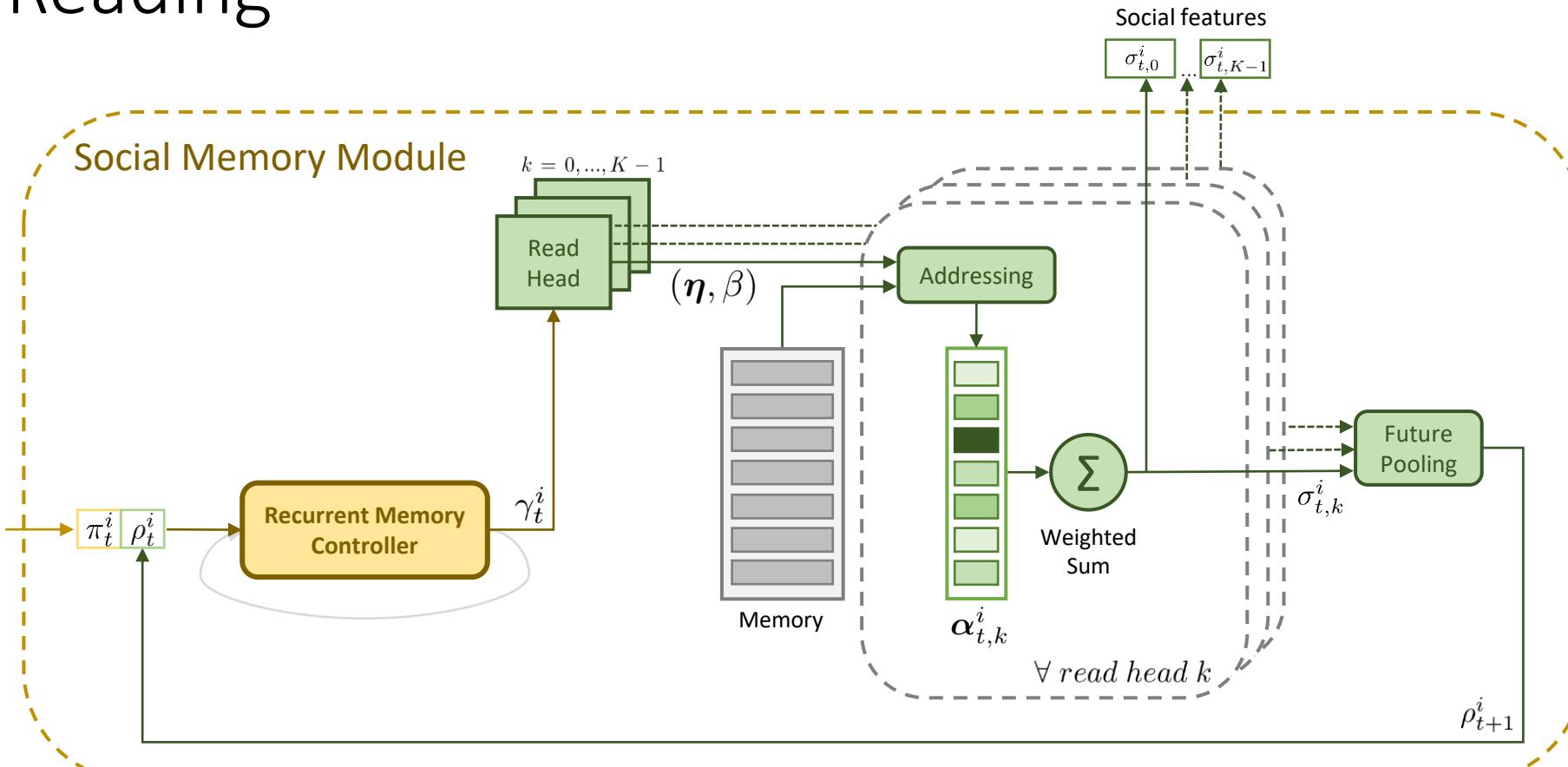
A social feature, read from memory, is used to condition the future generation.

# SMEMO: Addressing



- The memory controller outputs at each timestep a feature used by the read/write heads to generate a key.
- The key is used to find relevant locations in memory via cosine similarity.
- Access weights are obtained by softmax normalizing the similarities.

# SMEMO: Reading

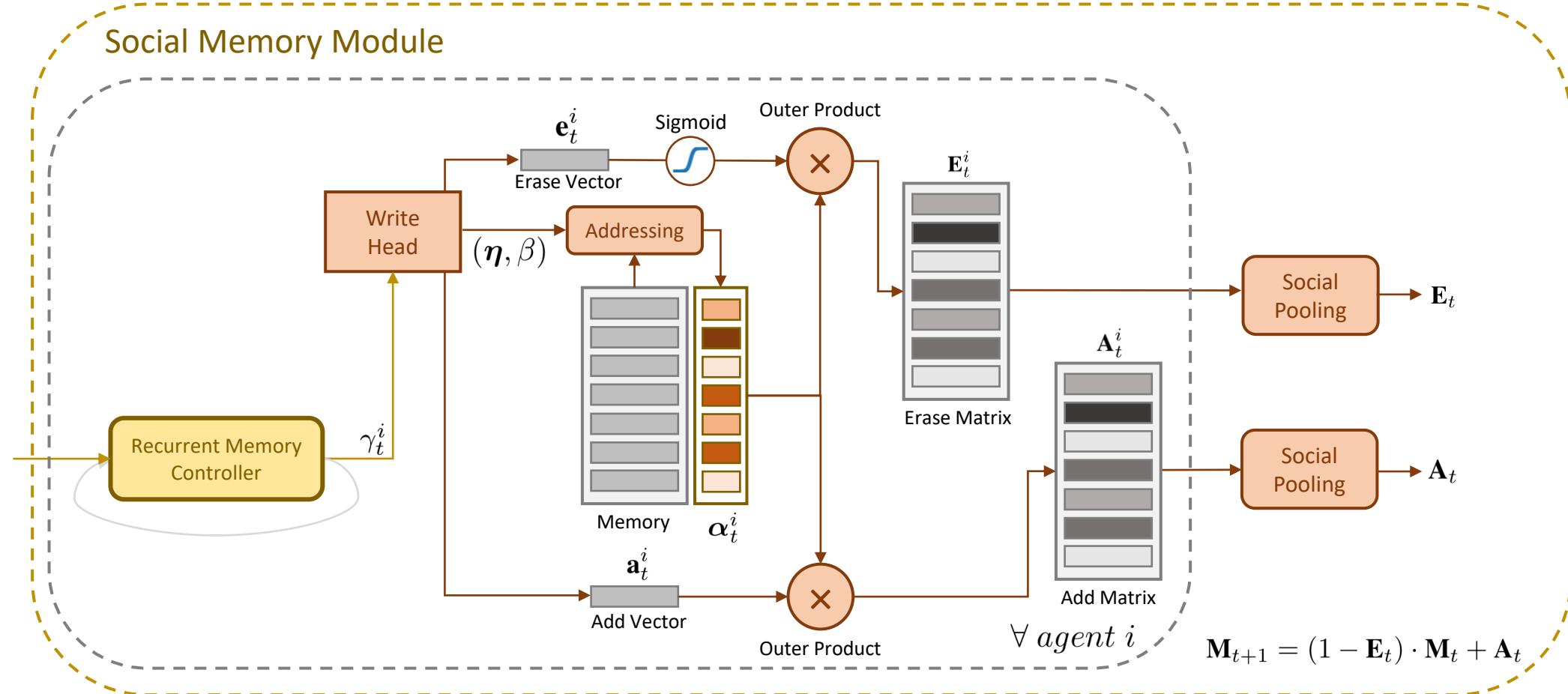


Separate read heads perform memory addressing to obtain  $K$  social features.

Each social feature will condition the decoder to output a separate future prediction.

The social features are pooled together and fed back autoregressively.

# SMEMO: Writing



A single write head performs memory addressing and generates erase and add vectors.

Using the addressed memory content, an Erase Matrix and an Add Matrix are generated.

Social Pooling guarantees invariance to agent ordering.

# Quantitative Results

K: number of predictions

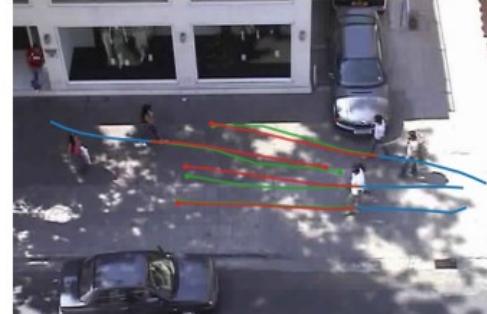
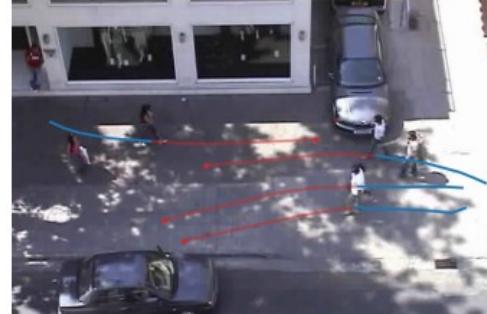
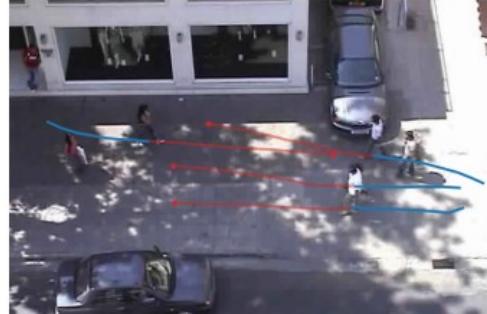
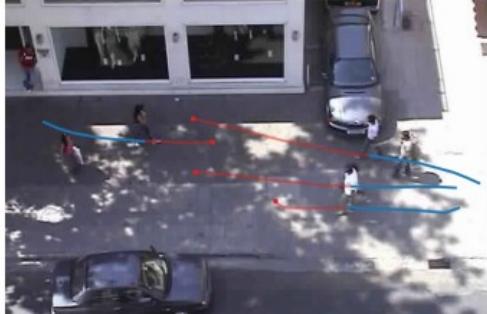
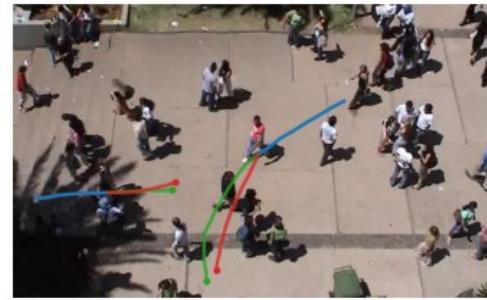
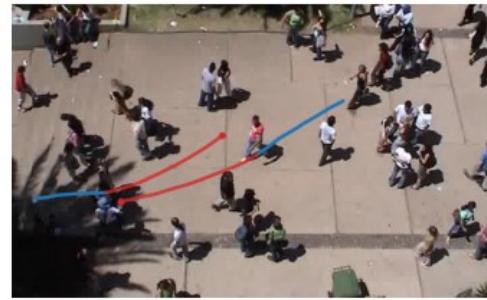
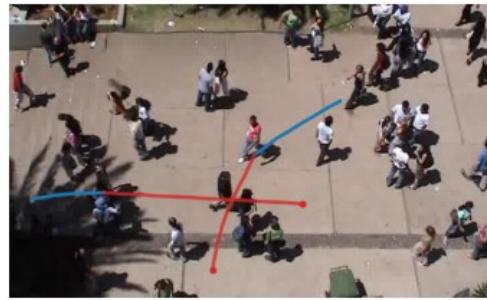
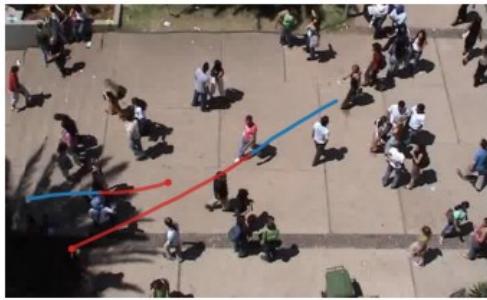
ETH/UCY

Method	ETH	HOTEL	UNIV	ZARA1	ZARA2	AVERAGE
SoPhie* [8]	0.70/1.43	0.76/1.67	0.54/1.24	0.30/0.63	0.38/0.78	0.54/1.15
Next* [67]	0.73/1.65	0.30/0.59	0.60/1.27	0.38/0.81	0.31/0.68	0.46/1.00
S-BiGAT* [33]	0.69/1.29	0.49/1.01	0.55/1.32	0.30/0.62	0.36/0.75	0.48/1.00
GOAL-GAN* [30]	0.59/1.18	0.19/0.35	0.60/1.19	0.43/0.87	0.32/0.65	0.43/0.85
Introvert* [23]	0.42/0.70	0.11/0.17	0.20/0.32	0.16/0.27	0.16/0.25	0.21/0.34
Social-GAN [6]	0.81/1.52	0.72/1.61	0.60/1.26	0.34/0.69	0.42/0.84	0.58/1.18
CGNS [68]	0.62/1.40	0.70/0.93	0.48/1.22	0.32/0.59	0.35/0.71	0.49/0.97
SR-LSTM [59]	0.63/1.25	0.37/0.74	0.51/1.10	0.41/0.90	0.32/0.70	0.45/0.94
MATF [69]	1.01/1.75	0.43/0.80	0.44/0.91	0.26/0.45	0.26/0.57	0.48/0.90
STGAT [36]	0.65/1.12	0.35/0.66	0.52/1.10	0.34/0.69	0.29/0.60	0.43/0.83
SGCN [34]	0.63/1.03	0.32/0.55	0.37/0.70	0.29/0.53	0.25/0.45	0.37/0.65
MANTRA [14]	0.48/0.88	0.17/0.33	0.37/0.81	0.27/0.58	0.30/0.67	0.32/0.65
Transformer [18]	0.61/1.12	0.18/0.30	0.35/0.65	0.22/0.38	0.17/0.32	0.31/0.55
PECNet [31]	0.54/0.87	0.18/0.24	0.35/0.60	0.22/0.39	0.17/0.30	0.29/0.48
PCCSNet [62]	<b>0.28/0.54</b>	0.11/0.19	0.29/0.60	0.21/0.44	0.15/0.34	0.21/0.42
Trajectron++ [28]	0.39/0.83	0.12/0.19	0.22/0.43	0.17/0.32	<b>0.12/0.25</b>	0.20/0.40
Social-NCE [70]	- /0.79	- /0.18	- /0.44	- /0.33	- /0.26	- /0.40
AgentFormer [25]	0.45/0.75	0.14/0.22	0.25/0.45	<b>0.18/0.30</b>	<b>0.14/0.24</b>	0.23/0.39
LB-EBM [64]	<b>0.30/0.52</b>	0.13/0.20	0.27/0.52	0.20/0.37	0.15/0.29	0.21/0.38
Expert-Goals [32]	0.37/0.65	<b>0.11/0.15</b>	<b>0.20/0.44</b>	<b>0.15/0.31</b>	<b>0.12/0.25</b>	<b>0.19/0.36</b>
<b>SMEMO</b>	0.39/0.59	0.14/0.20	<b>0.23/0.41</b>	0.19/0.32	0.15/0.26	<b>0.22/0.35</b>

SDD

K=5			K=20		
Method	ADE	FDE	Method	ADE	FDE
DESIRE [19]	19.25	34.05	Trajectron++ [28]*	19.30	32.70
Ridel et al. [61]	14.92	27.97	SoPhie [8]	16.27	29.38
MANTRA [14]	13.51	27.34	EvolveGraph [60]	13.90	22.90
PECNet [31]	12.79	25.98	CF-VAE [63]	12.60	22.30
PCCSNet [62]	12.54	-	Goal-GAN [30]	12.20	22.10
TNT [29]	12.23	21.16	P2TIRL [65]	12.58	22.07
<b>SMEMO</b>	<b>11.64</b>	<b>21.12</b>	SimAug [66]	10.27	19.71
			MANTRA [14]	8.96	17.76
			PCCSNet [62]	8.62	16.16
			PECNet [31]	9.96	15.88
			LB-EBM [64]	8.87	15.61
			Expert-Goals [32]	10.49	13.21
			<b>SMEMO</b>	<b>8.11</b>	<b>13.06</b>

# Qualitative Results



(a) Social Multimodal Prediction

(d) GT + best prediction

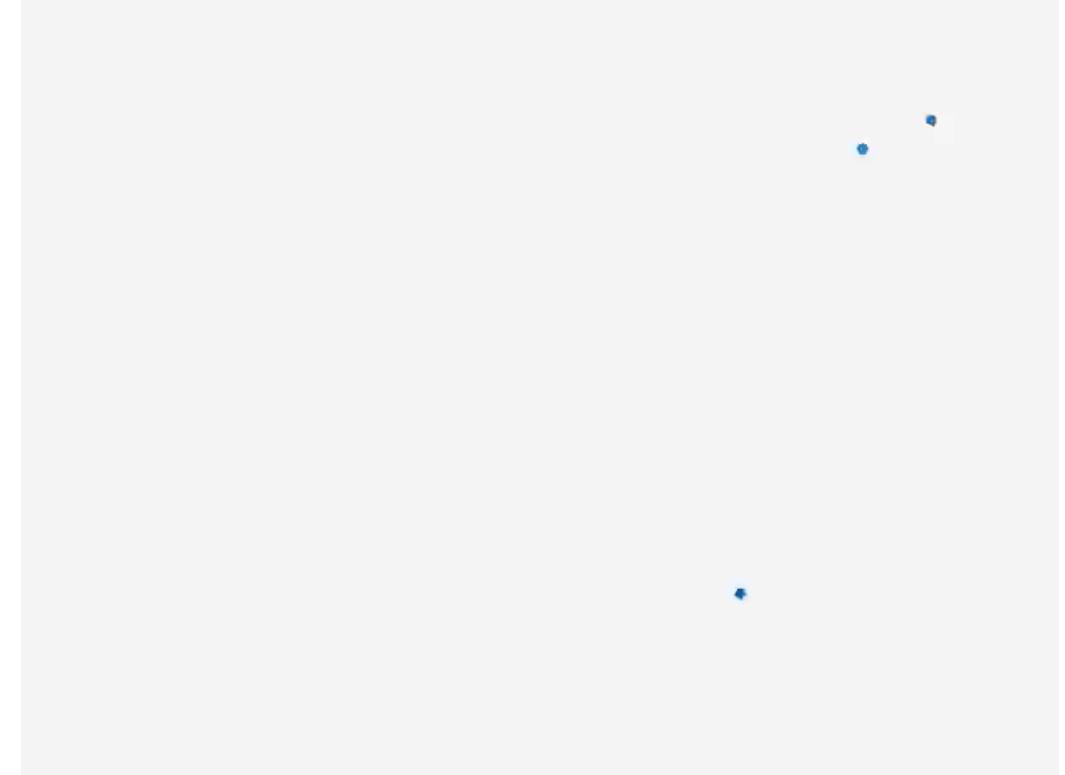
# Qualitative Results



# Qualitative Results: Real Dataset



Collision Avoidance



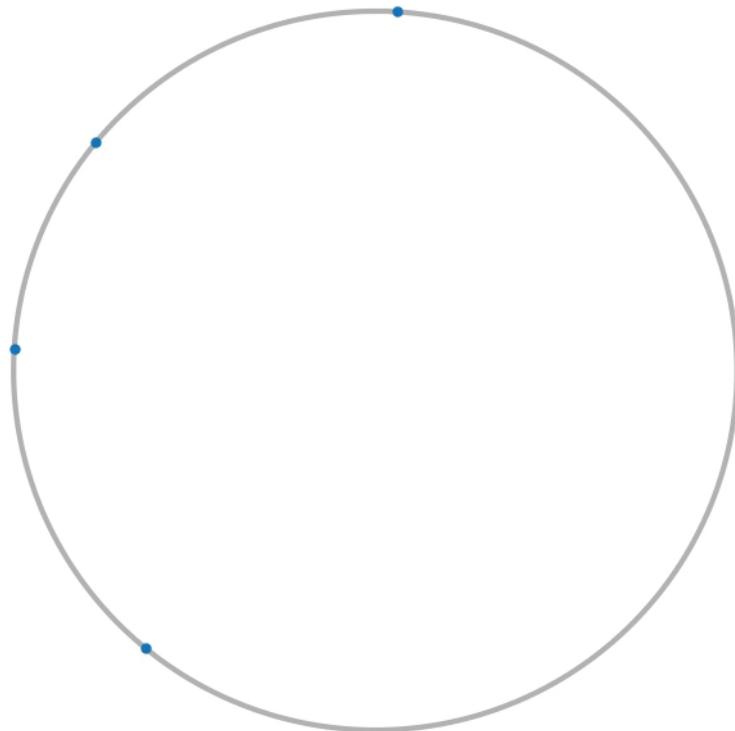
Group Following

# Synthetic Social Agents Dataset (SSA)



The agents (3 to 10)...

- start from one point on a unit circumference...
- ...at different speeds (constant)...
- ...all go towards the center...
- ...and must go to the opposite side



## Constraint

If two or more agents are close (below a certain distance threshold):

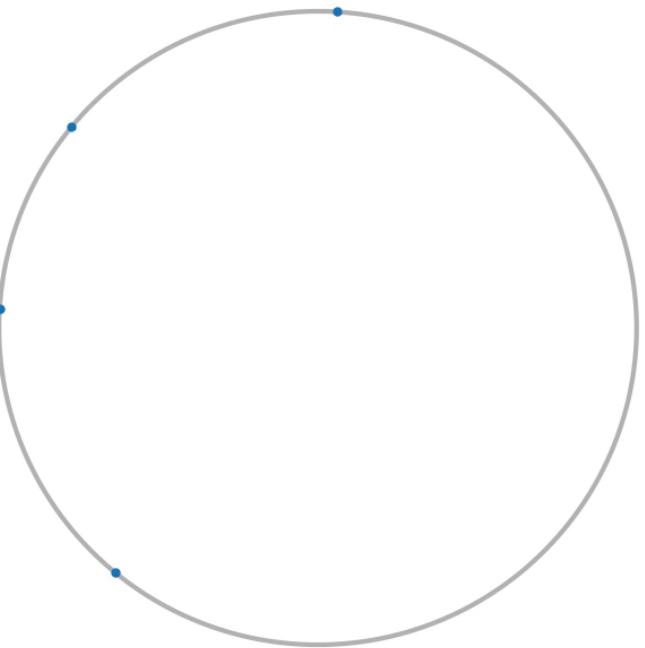
- the one with the highest speed passes
- and the others stop.



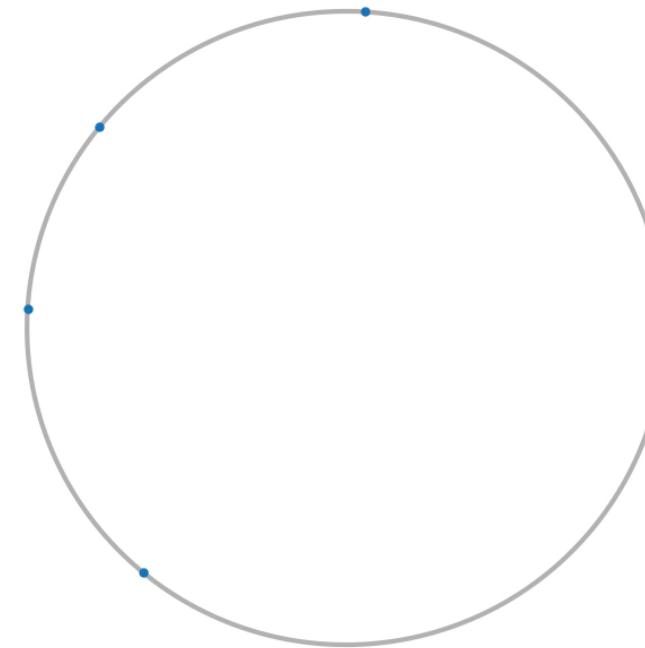
# Quantitative Results: SSA Datasets

<b>Method</b>	<b>ADE ↓</b>	<b>FDE ↓</b>	<b>Kendall ↑</b>
Linear	0.091	0.141	0.67
MLP	0.087	0.138	0.65
GRU ENC-DEC	0.087	0.138	0.64
Expert-Goals [32]	0.095	0.149	0.49
PECNet [31]	0.045	0.136	0.71
Trajectron++ [28]	0.084	0.132	0.59
Social-GAN [6]	0.051	0.085	0.67
AgentFormer [25]	0.040	0.064	0.70
SR-LSTM [59]	0.036	0.068	0.76
<b>SMEMO</b>	<b>0.027</b>	<b>0.038</b>	<b>0.83</b>

# Qualitative Results: SSA Dataset

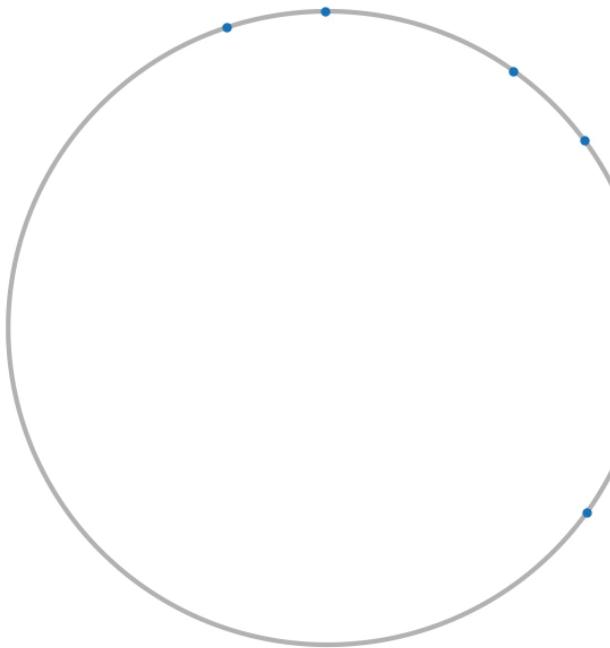


GROUND-TRUTH

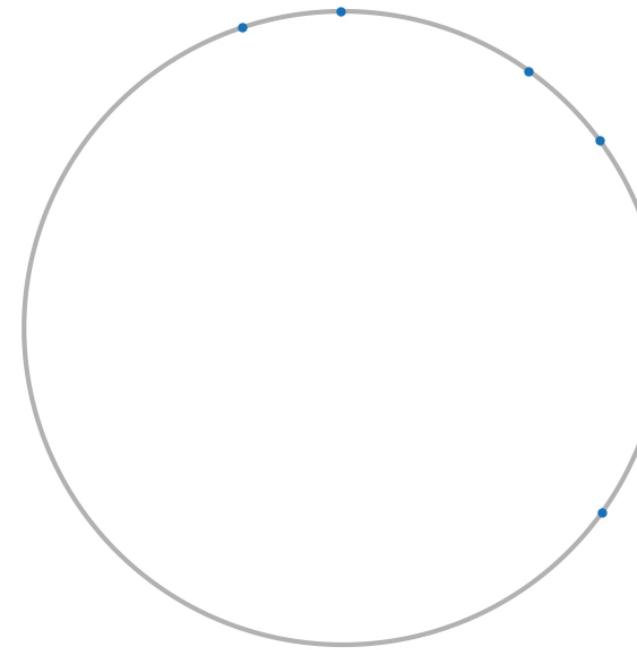


SMEMO

# Qualitative Results: SSA Dataset



GROUND-TRUTH

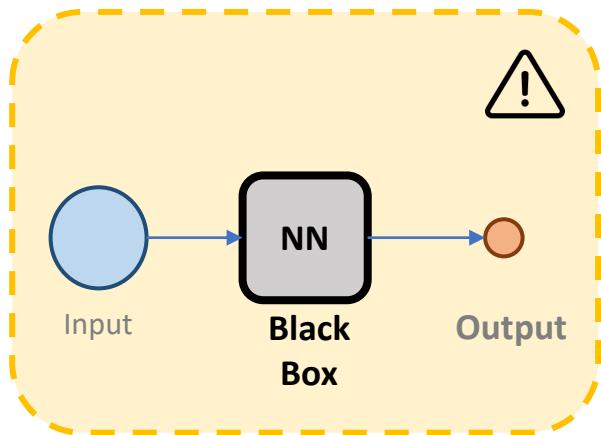


SMEMO

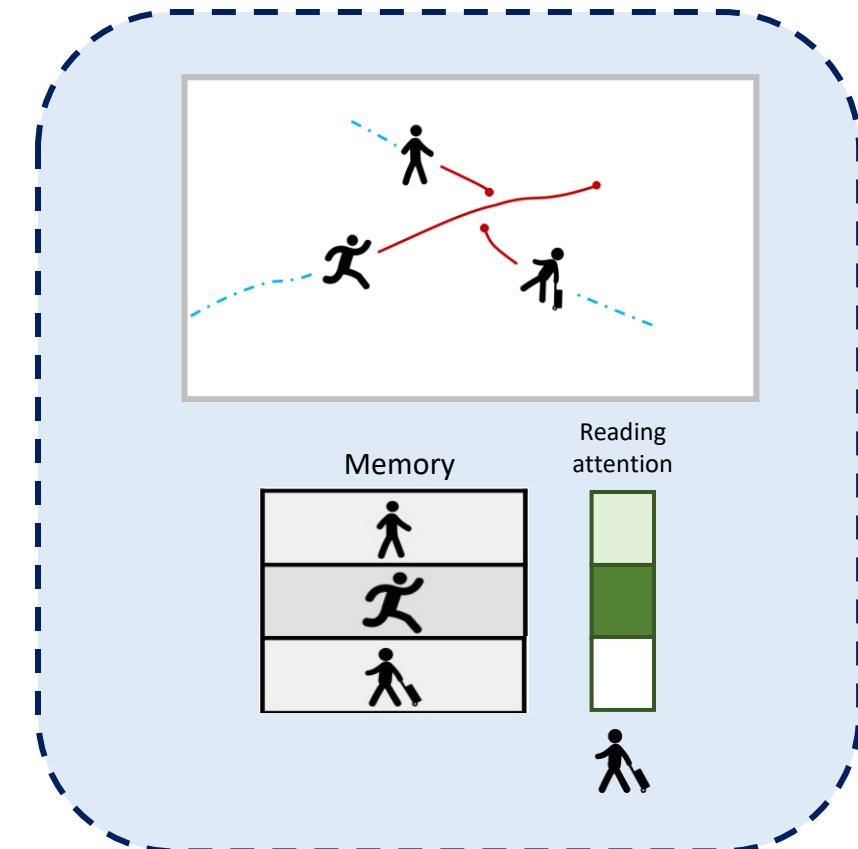
# SMEMO: Explainability



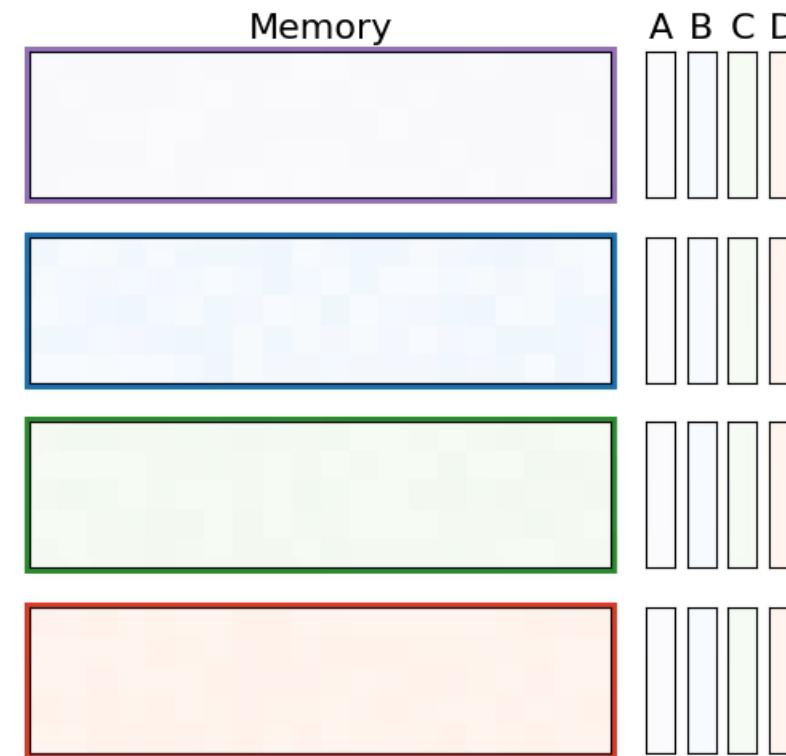
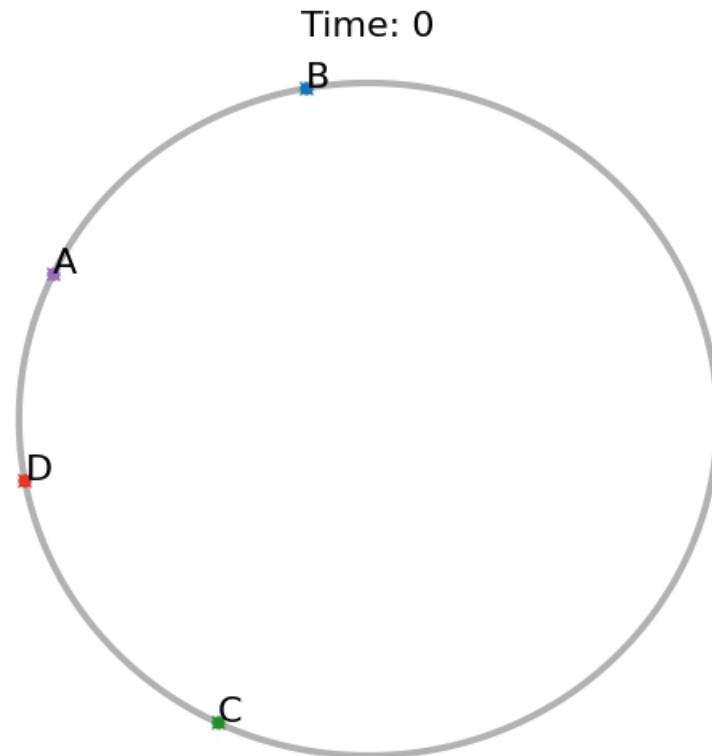
Deep learning methods lack the interpretability of the information learned from the network.



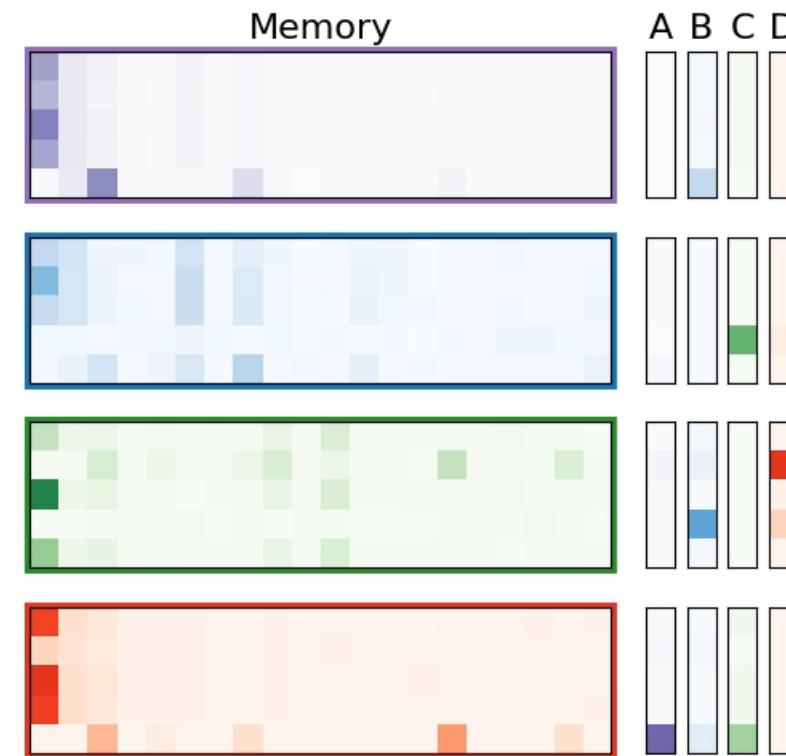
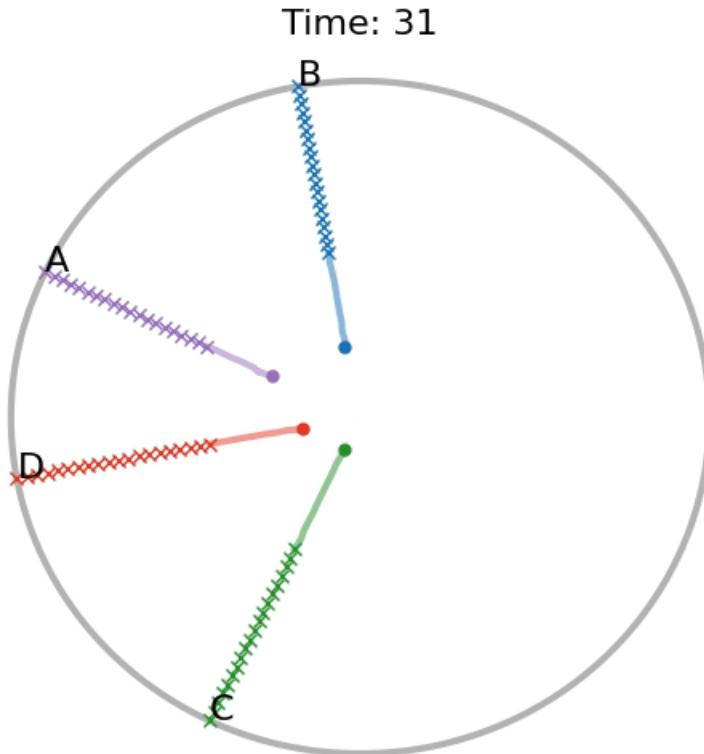
We want to understand which neighbors have contributed to the correct generation of each position of the future trajectory.



# Example explainability



# Example explainability



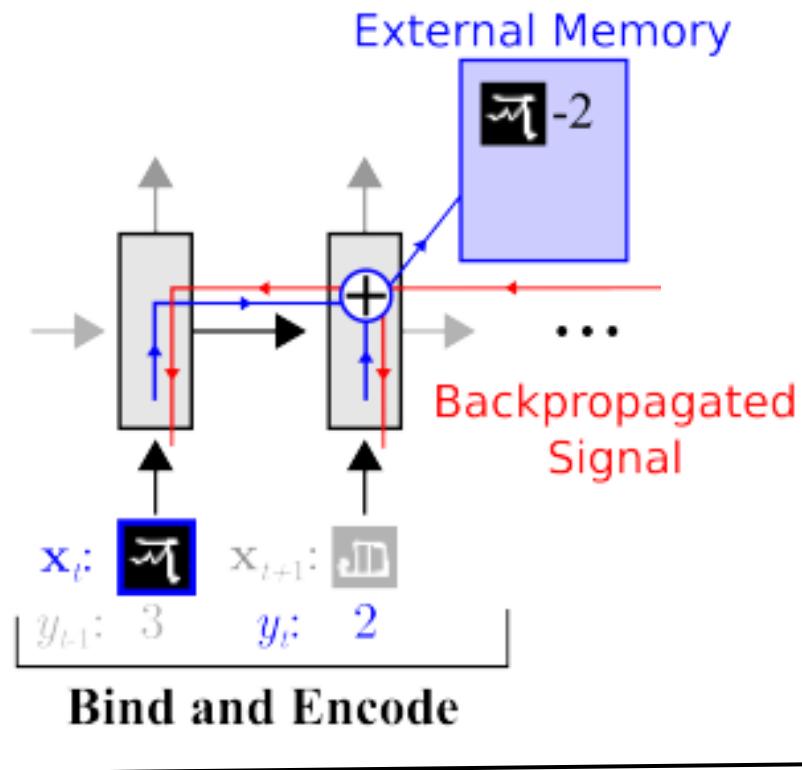
# Meta-Learning with Memory-Augmented Neural Networks

A. Santoro, et al. ICML 2016

MANN for *one-shot learning* task: situations when only a few training examples are presented one-by-one.

The model must learn to associate an image with a label presented with a time-offset

Must learn to hold data samples in memory until the appropriate labels are presented at the next time-step, after which sample-class information can be bound and stored for later use



At each time  $t$  the correct label for the previous sample image is presented  
The network must learn the association between the image and the last label given

# Reading

Reading is the same as NTM: when retrieving, a memory  $\mathbf{M}_t$  is addressed using the cosine similarity measure which is used to produce a read-weight vector.

$$\mathbf{r}_t \leftarrow \sum_i w_t^r(i) \mathbf{M}_t(i)$$

$$w_t^r(i) \leftarrow \frac{\exp(K(\mathbf{k}_t, \mathbf{M}_t(i)))}{\sum_j \exp(K(\mathbf{k}_t, \mathbf{M}_t(j)))}$$

$$K(\mathbf{k}_t, \mathbf{M}_t(i)) = \frac{\mathbf{k}_t \cdot \mathbf{M}_t(i)}{\| \mathbf{k}_t \| \| \mathbf{M}_t(i) \|}$$

# LRUA Writing

A new memory access method by content with reference to NTM.

*Least Recently Used Access* criterion:

LRUA writes new information into rarely-used locations, preserving recently encoded information or into the least used location (update of the memory with newer, possibly more relevant information).

The distinction between these two options is accomplished with an interpolation between the *previous read weights* and *weights scaled according to usage*.

$$\mathbf{w}_t^u \leftarrow \gamma \mathbf{w}_{t-1}^u + \mathbf{w}_t^r + \mathbf{w}_t^w$$

Updated by decaying the previous usage weights and adding the current read and write weights.

$$\mathbf{w}_t^w \leftarrow \sigma(\alpha) \mathbf{w}_{t-1}^r + (1 - \sigma(\alpha)) \mathbf{w}_{t-1}^{lu}.$$

Convex combination of the previous read weights and previous least-used weights.

$$w_t^{lu}(i) = \begin{cases} 0 & \text{if } w_t^u(i) > m(\mathbf{w}_t^u, n) \\ 1 & \text{if } w_t^u(i) \leq m(\mathbf{w}_t^u, n) \end{cases}$$

Least-used weights

# Training

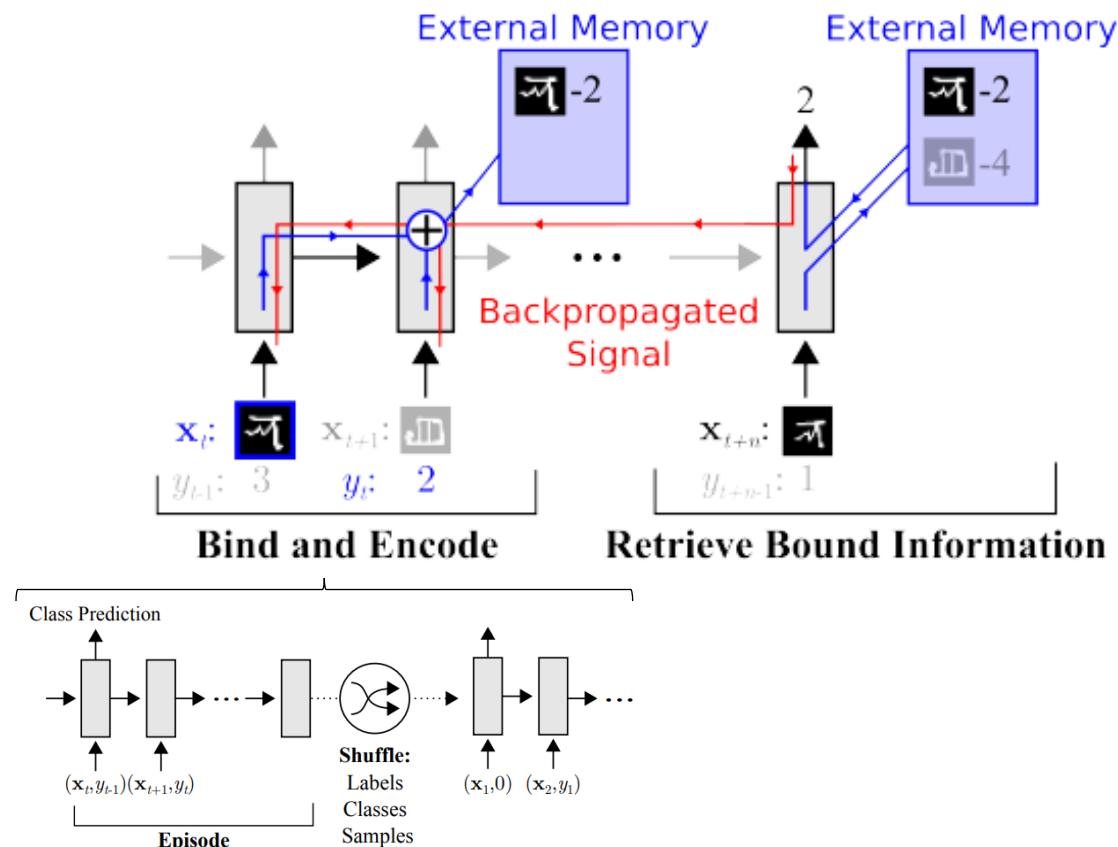
Data samples are held in memory until the label is presented.

Then sample-class information is bound into an encoding and stored for later use.

At every episode labels are shuffled across datasets to avoid learning image-to-label fixed associations.

At test time, when the input is presented again, the information is retrieved from memory and decoded to predict the correct label.

The network receives the sequence of inputs  
 $(x_1, \text{null}), (x_2, y_1), \dots, (x_T, y_{T-1})$



# Dataset



Omniglot: over 1600 separate classes with only a few examples per class.

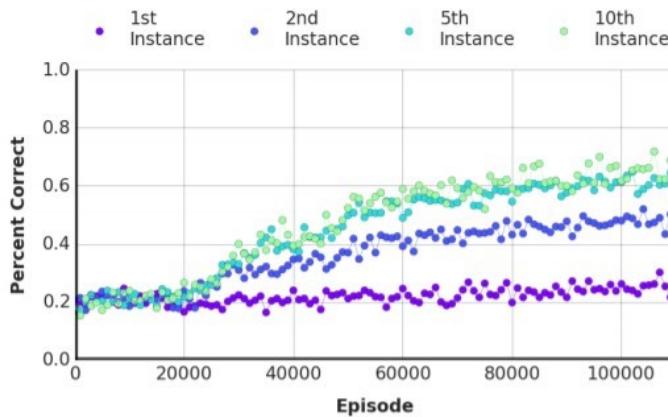
After training on 100k episodes with five randomly chosen classes with a randomly chosen label, there are a series of test episodes.

The model must predict the class labels for never-before-seen classes of a disjoint test set.

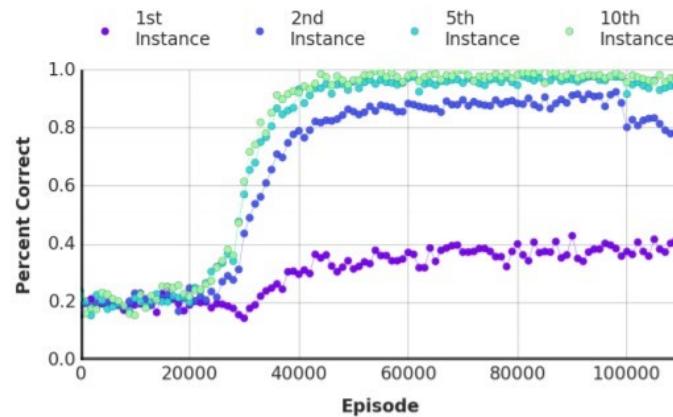
# Result: Human vs Machine

MODEL	INSTANCE (% CORRECT)					
	1 <sup>ST</sup>	2 <sup>ND</sup>	3 <sup>RD</sup>	4 <sup>TH</sup>	5 <sup>TH</sup>	10 <sup>TH</sup>
HUMAN	34.5	57.3	70.1	71.8	81.4	92.4
FEEDFORWARD	24.4	19.6	21.1	19.9	22.8	19.5
LSTM	24.4	49.5	55.3	61.0	63.6	62.5
MANN	<b>36.4</b>	<b>82.8</b>	<b>91.0</b>	<b>92.6</b>	<b>94.9</b>	<b>98.1</b>

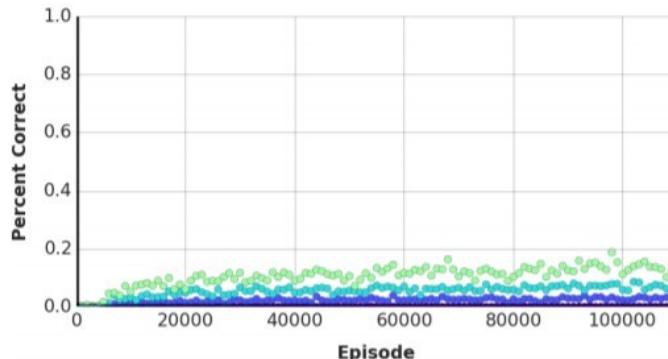
Test-set classification using one-hot encodings of labels and five classes presented per episode.



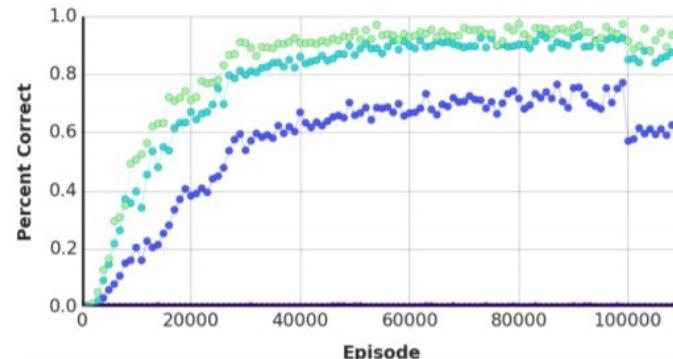
(a) LSTM, five random classes/episode, one-hot vector labels



(b) MANN, five random classes/episode, one-hot vector labels



(c) LSTM, fifteen classes/episode, five-character string labels



(d) MANN, fifteen classes/episode, five-character string labels

# End-To-End Memory Networks

Sukhbaatar et. al. NeurIPS 2015

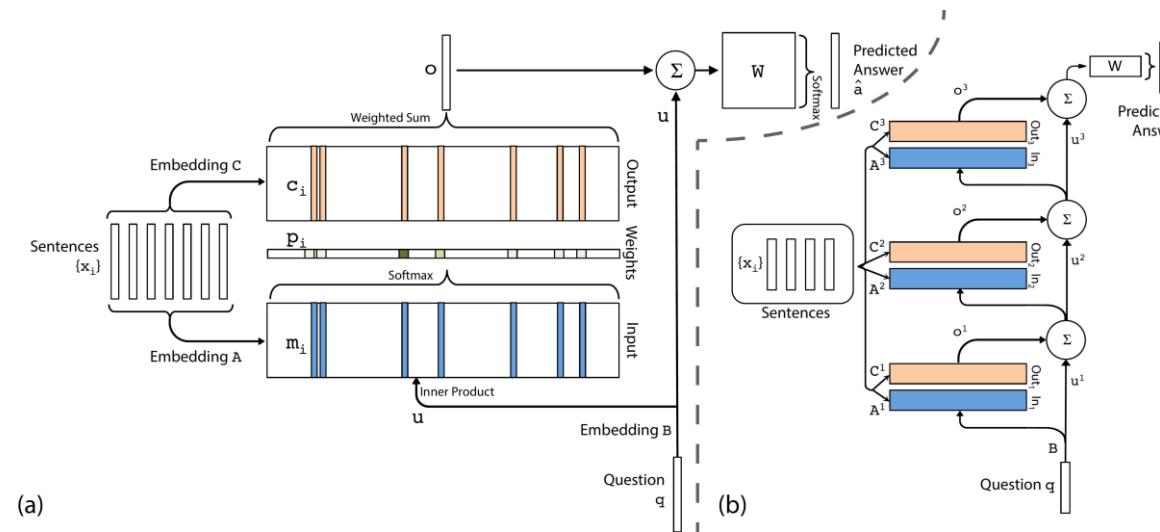
Uses a network with external memory to perform *Question Answering*.

All sentences are stored in the memory in separate slots.

To answer a query, the sentence which is most relevant is found in the memory through an attention mechanism.

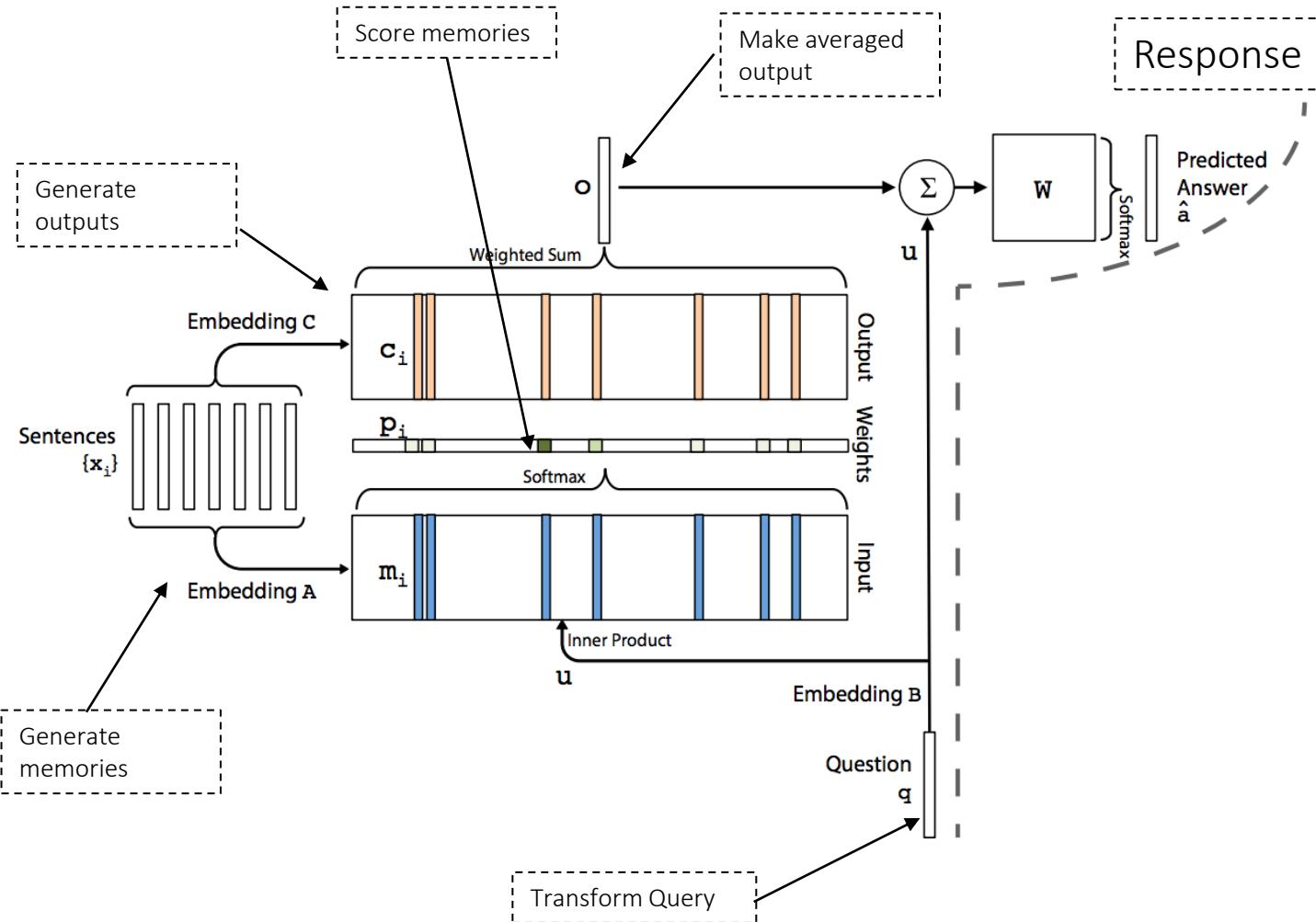
The retrieved sentence is then concatenated with the question to form a new query and look for a new relevant sentence.

The process is iterated until the model can generate an appropriate answer to the original query.



# End-To-End Memory Networks

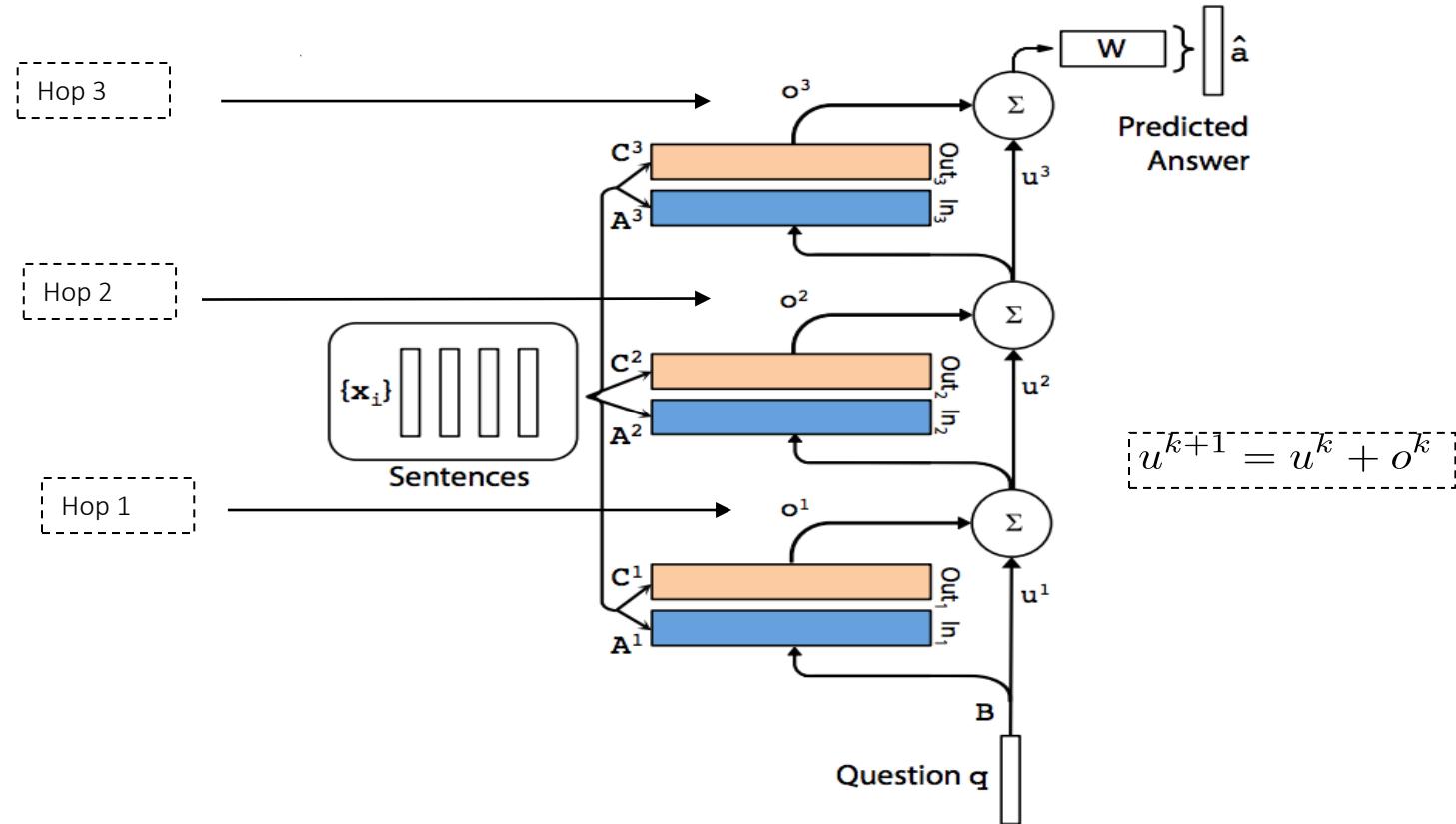
Sukhbaatar et. al. NeurIPS 2015



# End-To-End Memory Networks

Sukhbaatar et. al. NeurIPS 2015

Multi hop: different Memories and Outputs for each hop



# Results

## Question Answering (QA)

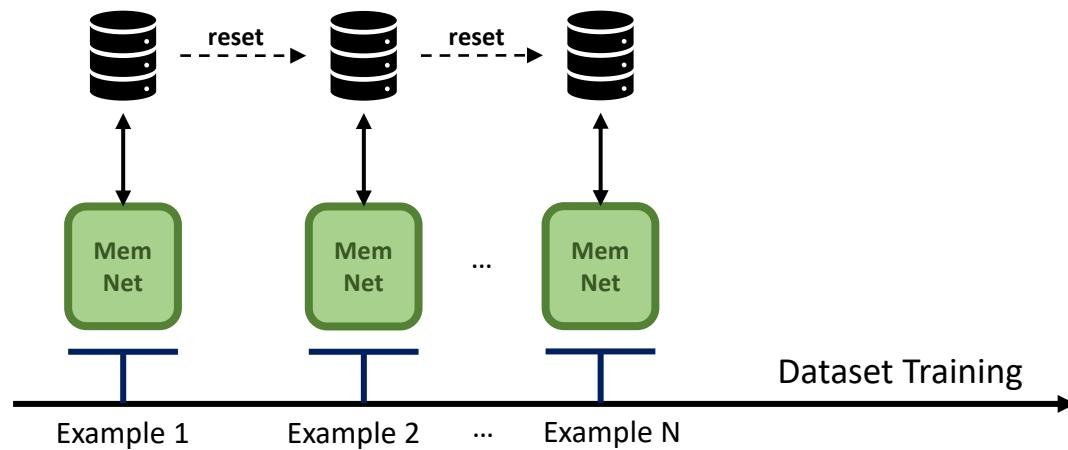
Mean error (%)	LSTM	MemN2n 1 layer	MemN2n 2 layers	MemN2n 3 layers
bAbI dataset	51.3	25.8	15.6	<b>13.3</b>

## Language Modeling

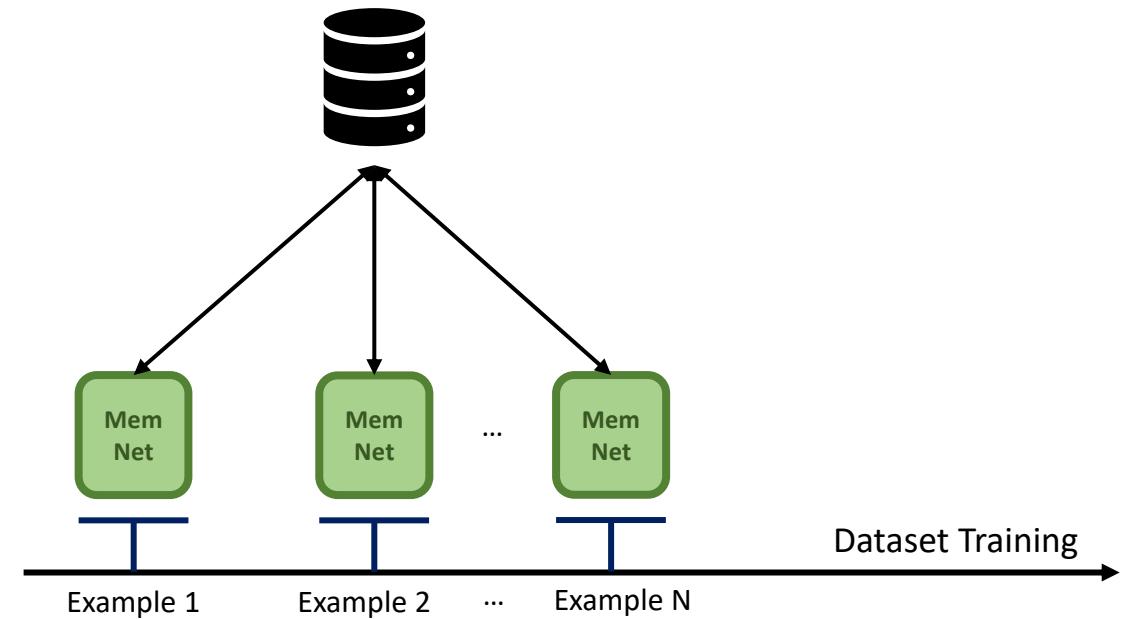
Error (Perplexity)	RNN	LSTM	MemN2n 3 layers	MemN2n 5 layers	MemN2n 7 layers
Penn Treebank Dataset	129	115	122	118	<b>111</b>
Text8 Dataset	184	154	178	154	<b>147</b>

# Episodic vs Persistent Memory

Episodic



Persistent



# MANNs with persistent memory

Alternatively to the case of episodic memory MANNs can have *persistent memory*

- With NTM, memory is continuously written to and read from, with network learning when to perform memory read and write operations.
- Using memory as a persistent storage means focusing on retrieving information. The *Controller* is trained to write in memory only the examples meaningful for the task.

# Visual Question Answering with Memory-Augmented Networks

Ma et al.; CVPR2018



Given an input question and a reference image, the task is to predict the most accurate answer.



SOTA methods learn to respond to the majority of training data rather than specific scarce exemplars.



**Q:** What is the dark green vegetable?

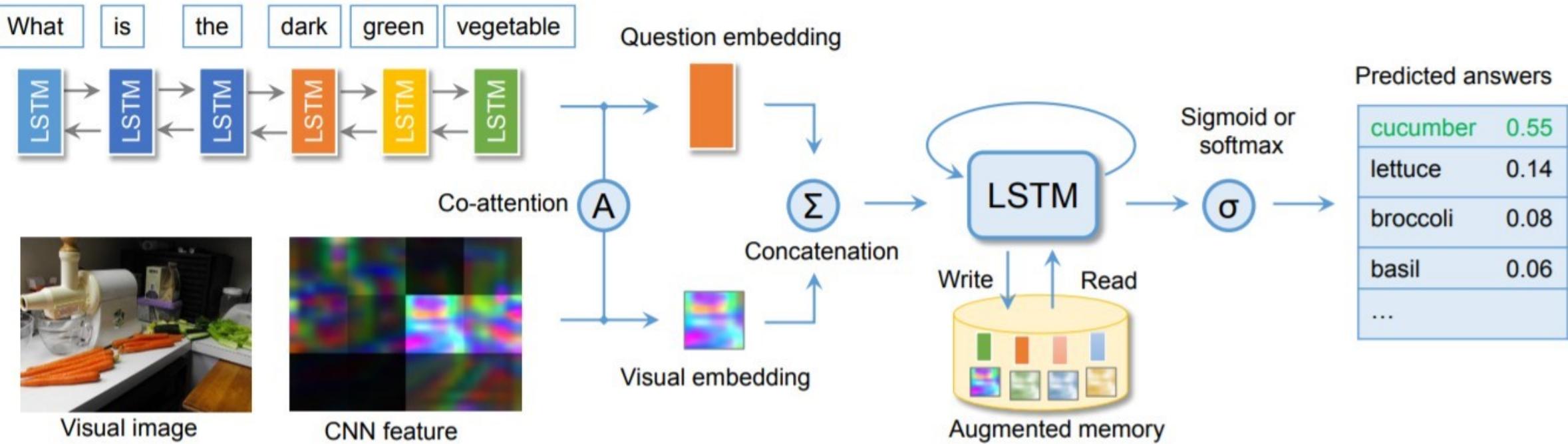
**A:** Cucumber (Ours)    **A:** Broccoli [21]    **A:** Lettuce [2]

VQA systems [2, 21] exclude the rare answer “cucumber” from the training set.

[2]: S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. L. Zitnick, and D. Parikh. VQA: Visual Question Answering. In Proc. IEEE Int. Conf. Comp. Vis., 2015

[21]: J. Lu, J. Yang, D. Batra, and D. Parikh. Hierarchical question-image co-attention for visual question answering. In Proc. Advances in Neural Inf. Process. Syst., 2016.

# Architecture



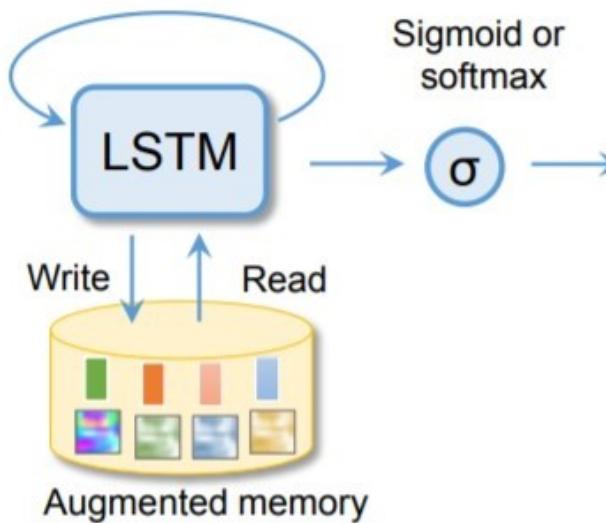
**Co-attention** attends to the most relevant image regions as well as textual word.

**Augmented memory** maintains a long-term memory of scarce and uncommon question and answer pairs.

**The controller LSTM** determines when to write or read from the external augmented memory.

# Memory details: reading

The module learns a mechanism that can selectively pay more attention to scarce training items whose effect is always neglected during a huge amount of training iterations.



$$\mathbf{h}_t = \text{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

Hidden state

$$D(\mathbf{h}_t, \mathbf{M}_t(i)) = \frac{\mathbf{h}_t \cdot \mathbf{M}_t(i)}{\|\mathbf{h}_t\| \|\mathbf{M}_t(i)\|}$$

Cosine Distance

$$w_t^r(i) = \text{softmax}(D(\mathbf{h}_t, \mathbf{M}_t(i)))$$

Read weight vector

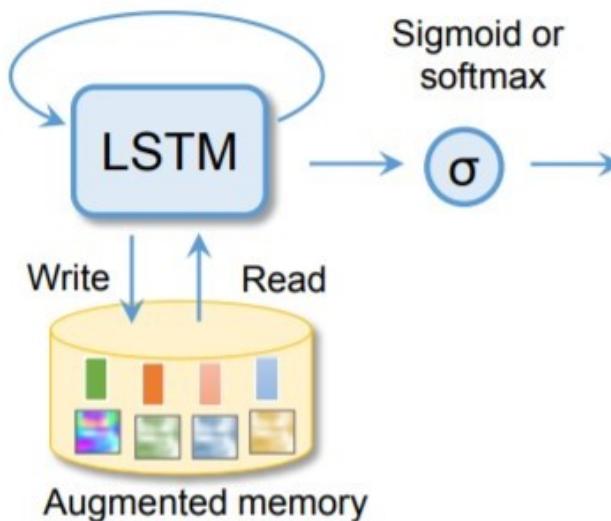
$$\mathbf{r}_t = \sum_i w_t^r(i) \mathbf{M}_i$$

Retrieved values from memory

# Memory details: writing

Similar to *One-shot Learning with Memory Augmented Neural Networks*.

Usage weights to control writing to memory.



$$\mathbf{h}_t = \text{LSTM}(\mathbf{x}_t, \mathbf{h}_{t-1})$$

Hidden state

$$\mathbf{w}_t^u = \gamma \mathbf{w}_{t-1}^u + \mathbf{w}_t^r + \mathbf{w}_t^w$$

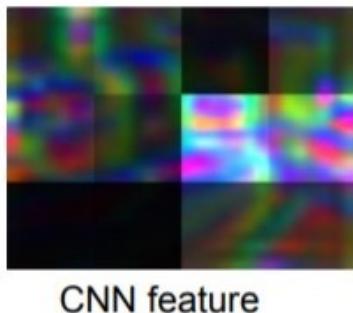
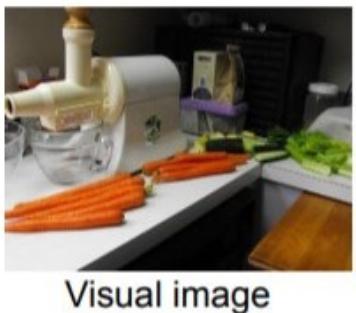
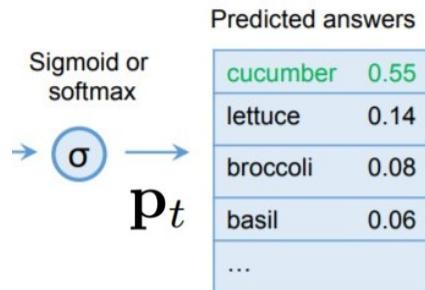
Usage weights

$$\mathbf{w}_t^w = \sigma(\alpha) \mathbf{w}_{t-1}^r + (1 - \sigma(\alpha)) \mathbb{1}(\mathbf{w}_{t-1}^u \leq m(\mathbf{w}_{t-1}^u, n))$$

write weights

$$\mathbf{M}_t^i = \mathbf{M}_{t-1}(i) + w_t^w(i) \mathbf{h}_t$$

# Training



Given the output distribution, the network is optimized by minimizing the Cross-Entropy loss over the input one-hot encoded label vector.

$$\mathcal{L}(\theta) = - \sum_t \mathbf{y}_t^\top \log \mathbf{p}_t$$

Pre-trained VGGNet-16 and ResNet-101 to extract CNN features

# Results

Method	Test-dev								Test-standard								<b>best</b> <u>second best</u>	
	Multiple-choice				Open-ended				Multiple-choice				Open-ended					
	Y/N	Num	Other	All	Y/N	Num	Other	All	Y/N	Num	Other	All	Y/N	Num	Other	All		
iBOWIMG [44]	76.7	37.1	54.4	61.7	76.6	35.0	42.6	55.7	76.9	37.3	54.6	62.0	76.8	35.0	42.6	55.9		
DPPnet [27]	80.8	38.9	52.2	62.5	80.7	37.2	41.7	57.2	80.4	38.8	52.8	62.7	80.3	36.9	42.2	57.4		
VQA team [2]	80.5	38.2	53.0	62.7	80.5	36.8	43.1	57.8	80.6	37.7	53.6	63.1	80.6	36.4	43.7	58.2		
SAN [43]	-	-	-	-	79.3	36.6	46.1	58.7	-	-	-	-	-	-	-	58.9		
NMN [1]	-	-	-	-	80.5	37.4	43.1	57.9	-	-	-	-	-	-	-	58.0		
ACK [39]	-	-	-	-	81.0	38.4	45.2	59.2	-	-	-	-	81.1	37.1	45.8	59.4		
SMem [41]	-	-	-	-	80.9	37.3	43.1	58.0	-	-	-	-	80.8	37.3	43.1	58.2		
DMN+ [40]	-	-	-	-	80.5	36.8	48.3	60.3	-	-	-	-	-	-	-	60.4		
MRN-ResNet [17]	<b>82.4</b>	39.7	57.2	65.6	<u>82.4</u>	38.4	49.3	61.5	<b>82.4</b>	39.6	58.4	66.3	<b>82.4</b>	<b>38.2</b>	49.4	61.8		
Re-Ask-ResNet [23]	-	-	-	-	78.4	36.4	46.3	58.4	-	-	-	-	78.2	36.3	46.3	58.4		
HieCoAtt-ResNet [21]	79.7	40.0	59.8	65.8	79.7	38.7	51.7	61.8	-	-	-	66.1	-	-	-	62.1		
RAU-ResNet [26]	<u>81.9</u>	<u>41.1</u>	61.5	67.7	81.9	<b>39.0</b>	53.0	63.3	<u>81.7</u>	<u>40.0</u>	61.0	67.3	<u>81.7</u>	<b>38.2</b>	52.8	63.2		
MCB-ResNet [7]	-	-	-	<u>69.1</u>	<b>82.5</b>	37.6	<b>55.6</b>	<b>64.7</b>	-	-	-	-	-	-	-	-		
MLP-ResNet [14]	-	-	-	-	-	-	-	-	80.8	17.6	62.0	65.2	-	-	-	-		
VQA-Mac-ResNet [37]	81.5	40.0	62.2	67.7	81.5	38.4	53.0	63.1	81.4	39.8	<u>62.3</u>	<u>67.8</u>	81.4	<b>38.2</b>	<u>53.2</u>	<u>63.3</u>		
Ours-VGG	81.1	41.0	<u>62.5</u>	67.8	81.2	37.8	50.7	61.8	81.2	39.3	61.7	67.4	81.2	36.4	51.7	62.3		
Ours-ResNet	81.6	<b>42.1</b>	<b>65.2</b>	<b>69.5</b>	81.5	<b>39.0</b>	<u>54.0</u>	<u>63.8</u>	81.6	<b>40.9</b>	<b>65.1</b>	<b>69.4</b>	<u>81.7</u>	37.6	<b>54.7</b>	<b>64.1</b>		

MSCOCO dataset: 200K real images with three questions each.

Each question has ten answers collected from human subjects.

$$\text{Acc}(\hat{a}) = \min \left\{ \frac{\#\text{humans that labeled } \hat{a}}{3}, 1 \right\}$$

# Trajectory Prediction

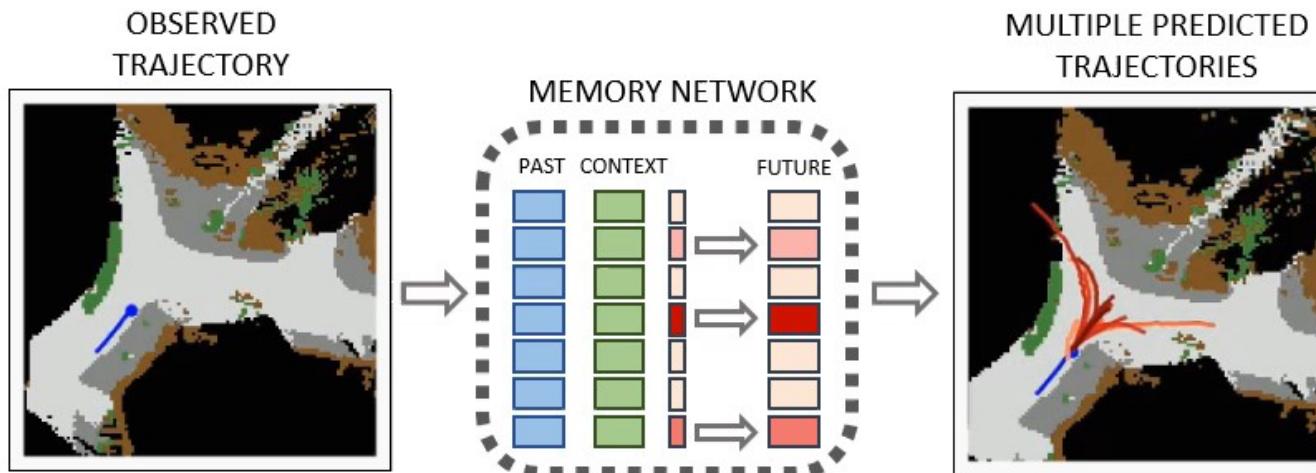


# Trajectory Prediction



Multiple futures are possible

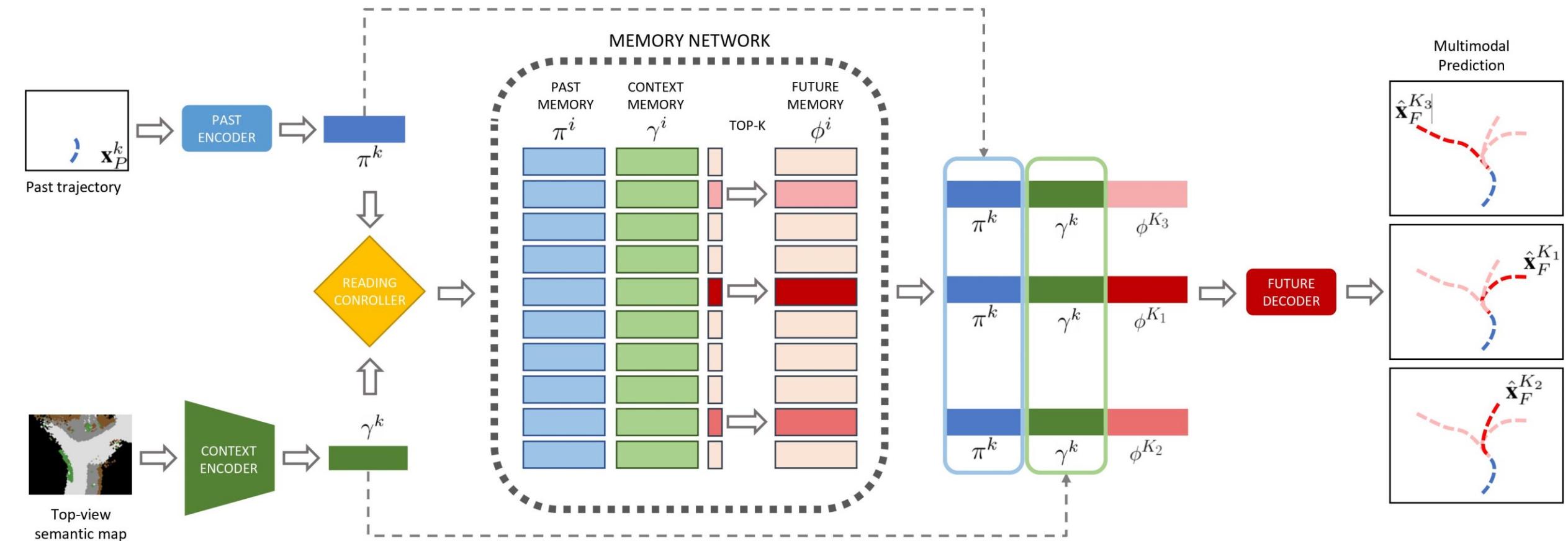
# MANTRA: Memory Augmented Neural TRAjectory predictor



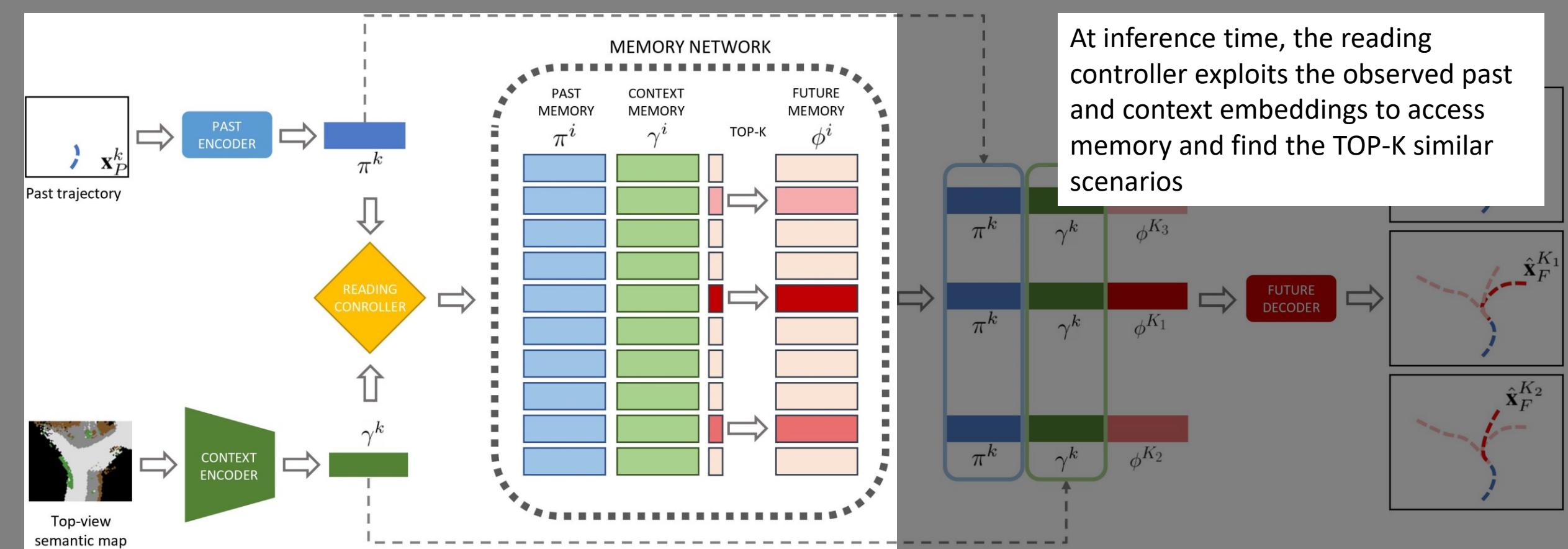
F. Marchetti et al., "Mantra: Memory augmented networks for multiple trajectory prediction", CVPR 2020

F. Marchetti et al., "Multiple Trajectory Prediction of Moving Agents with Memory Augmented Networks", TPAMI 2020

# MANTRA: Overview



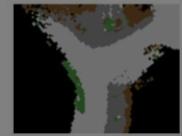
# MANTRA: Inference Time



# MANTRA: Decoding

Each future read from memory is combined with the current past and context and is decoded into a future prediction

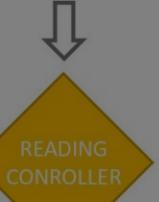
Past trajectory



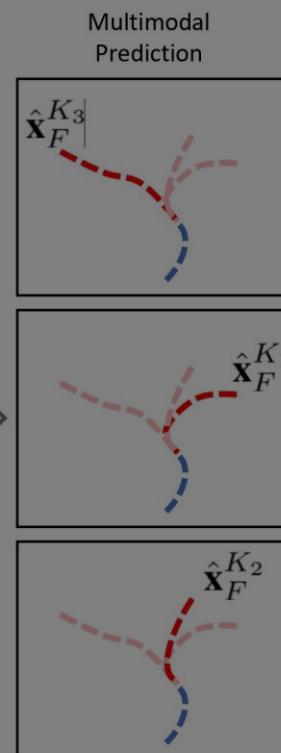
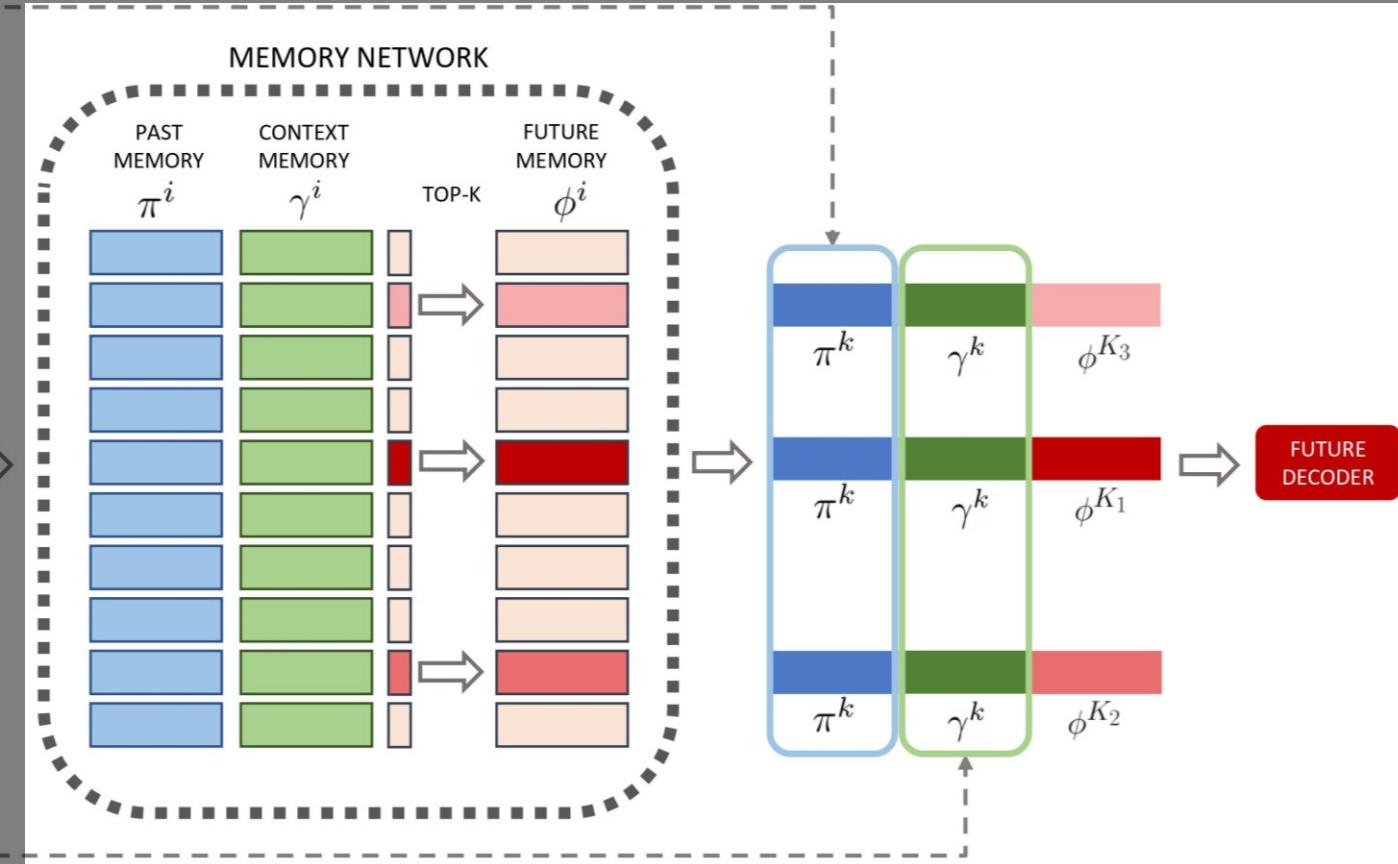
Top-view  
semantic map



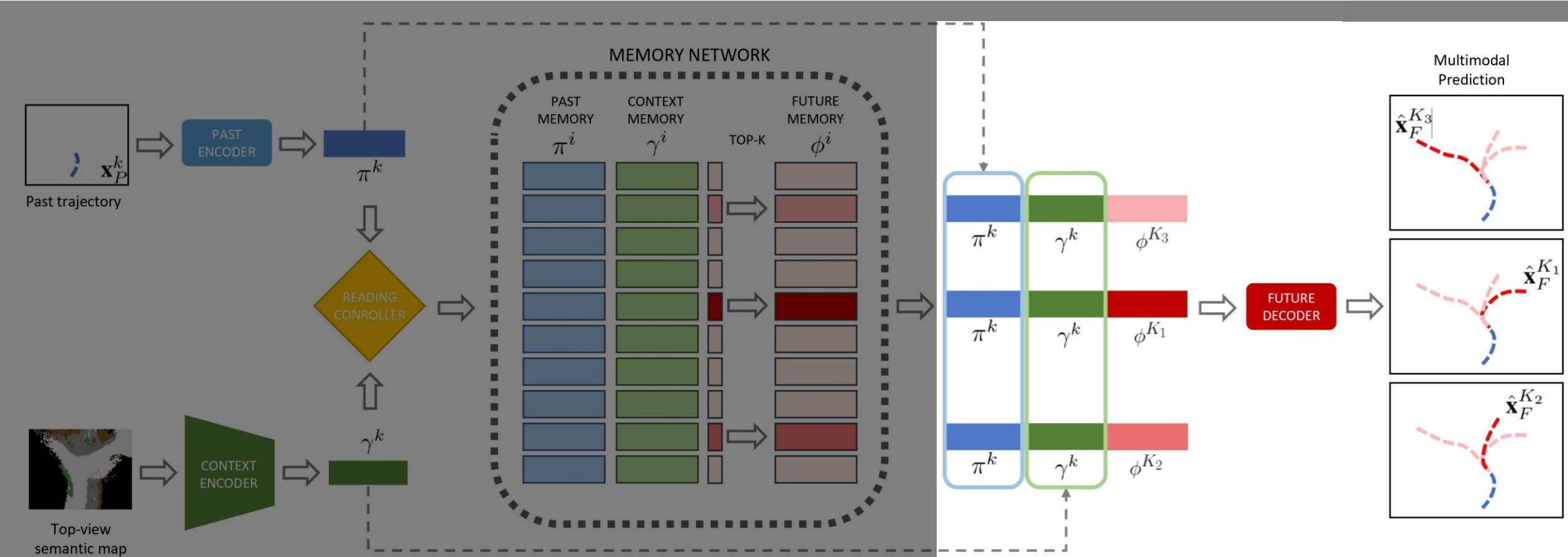
CONTEXT  
ENCODER



$\gamma^k$

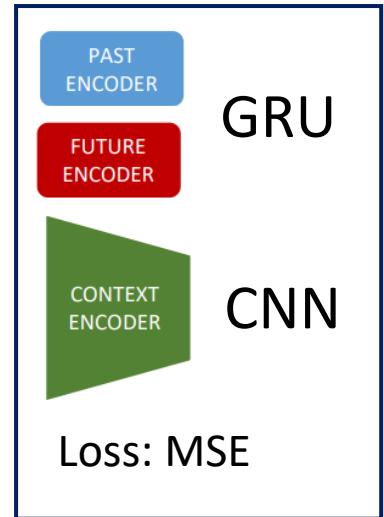
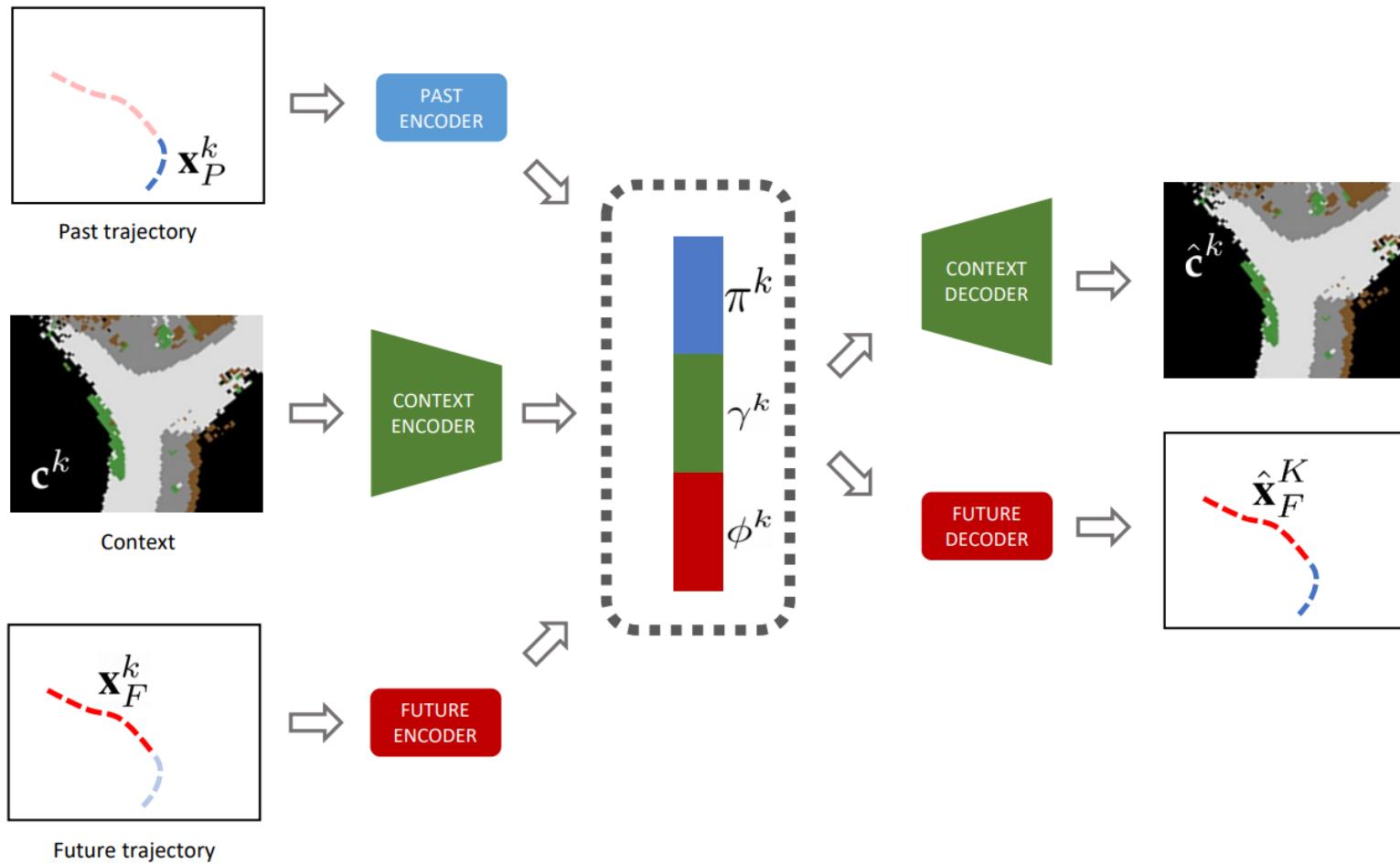


# MANTRA: Multimodal prediction

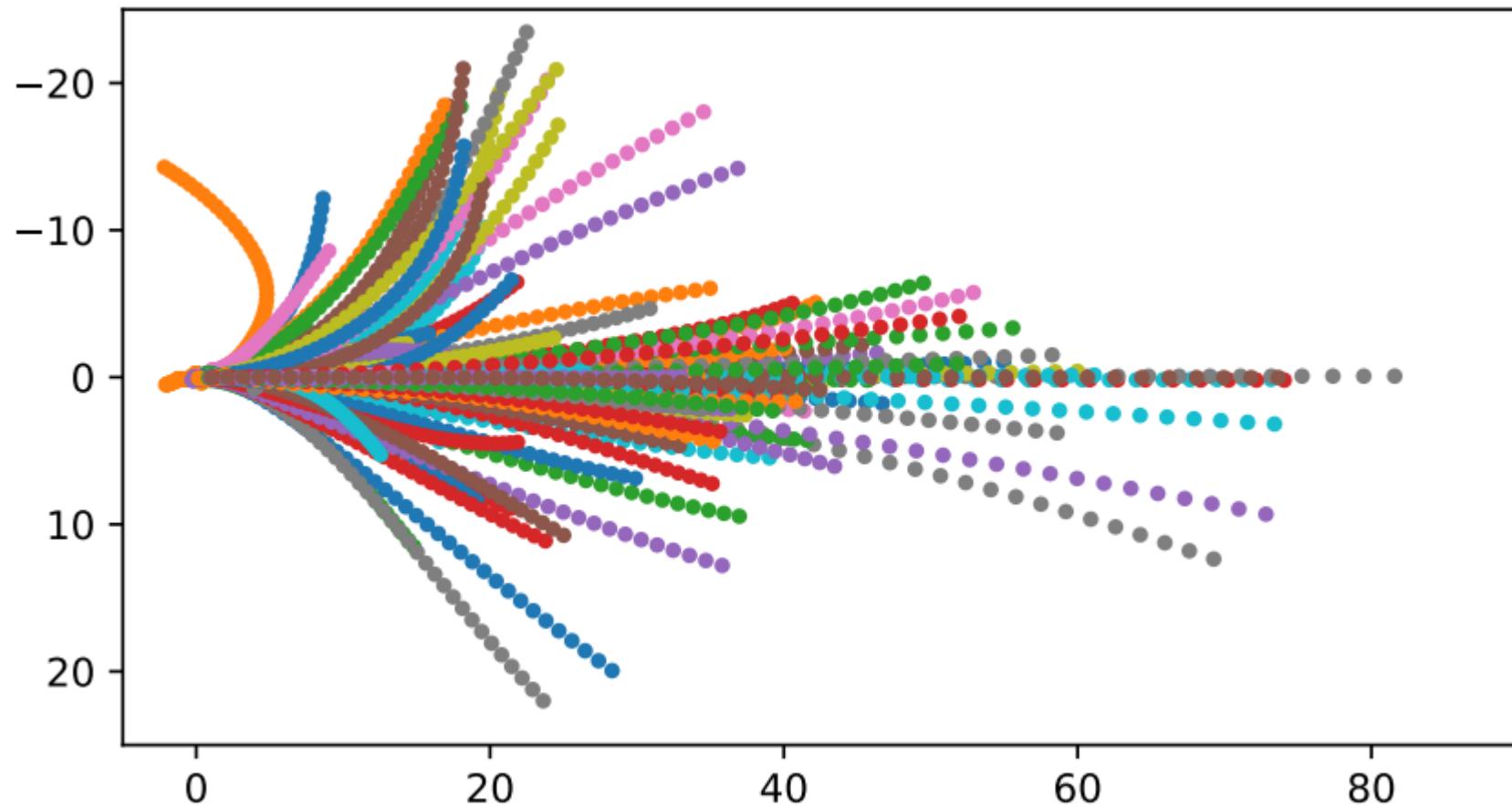


# AutoEncoder for Feature Representation

- To learn effective feature representations, we train encoding and decoding functions as an autoencoder.

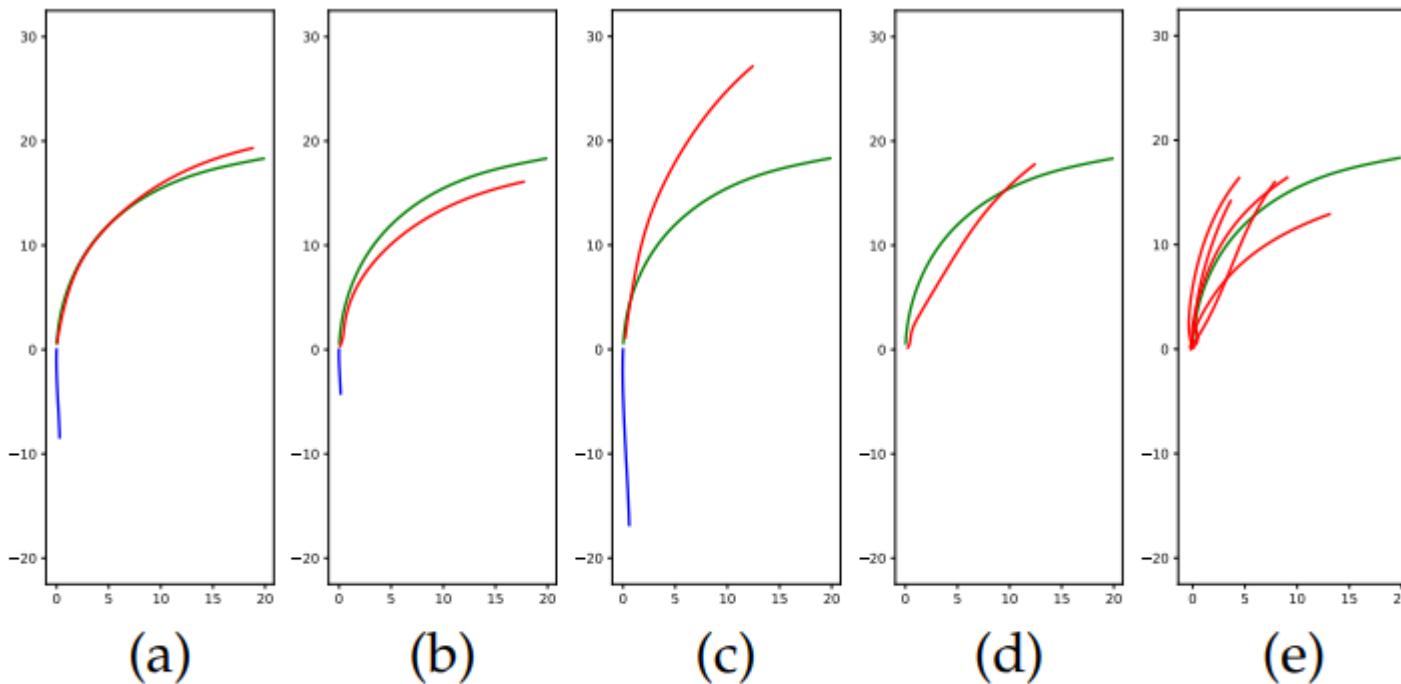


# Memory Inspection



# AutoEncoder for Feature Representation

Influence of past



Past: 2s  
Future: 4s

Fig. 11. Influence of past in the decoder. (a) observed past; (b) slower past; (c) faster past; (d) past embedding zeroed; (e) multiple randomized past embeddings. Blue: past trajectory. Red: future reconstruction, Green: original future.

# AutoEncoder for Feature Representation

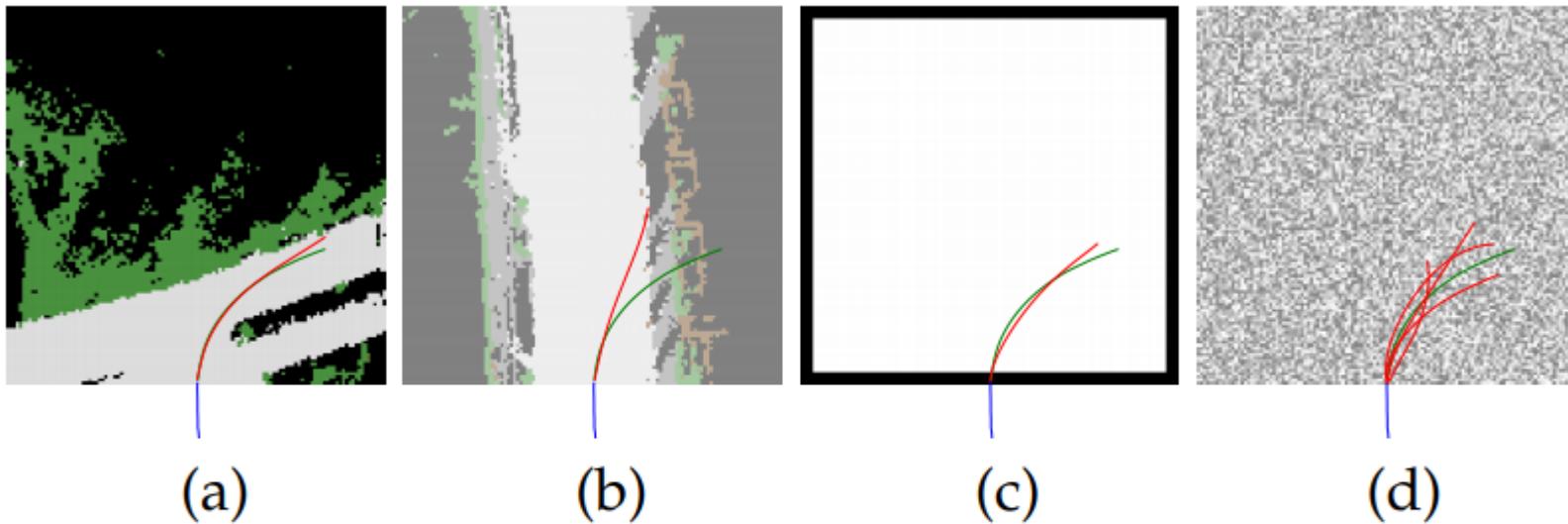
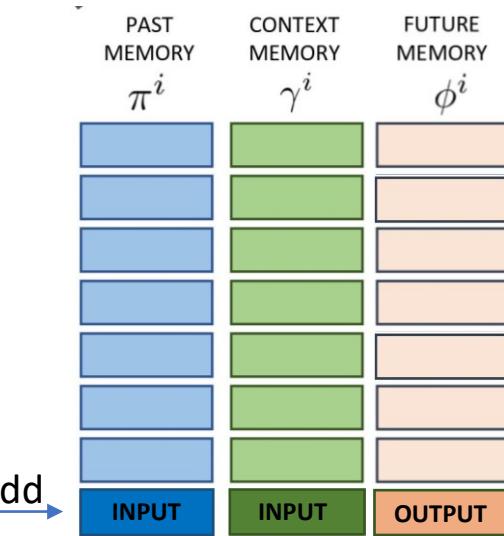
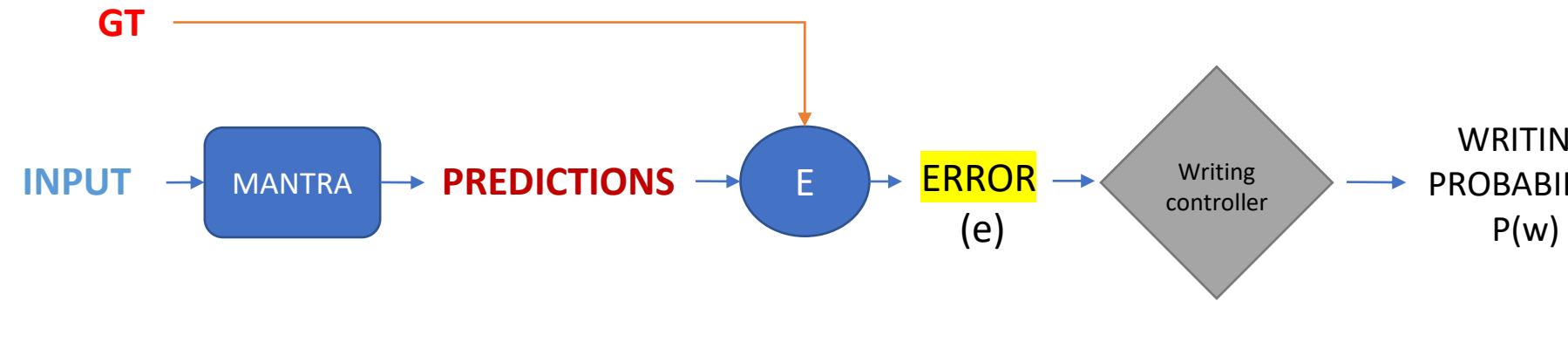


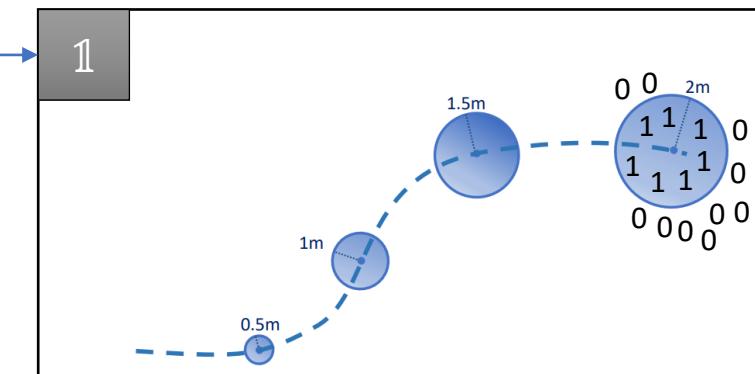
Fig. 12. Influence of context in the decoder. (a) original context; (b) different context; (c) context embedding zeroed; (e) multiple randomized context embeddings. Blue: past trajectory used for decoding. Red: future reconstruction. Green: original future.

# Training: writing controller



adaptive miss rate error

$$E = 1 - \frac{1}{N} \sum_{i=1}^N \mathbb{1}_i(\hat{\mathbf{x}}_F, \mathbf{x}_F)$$



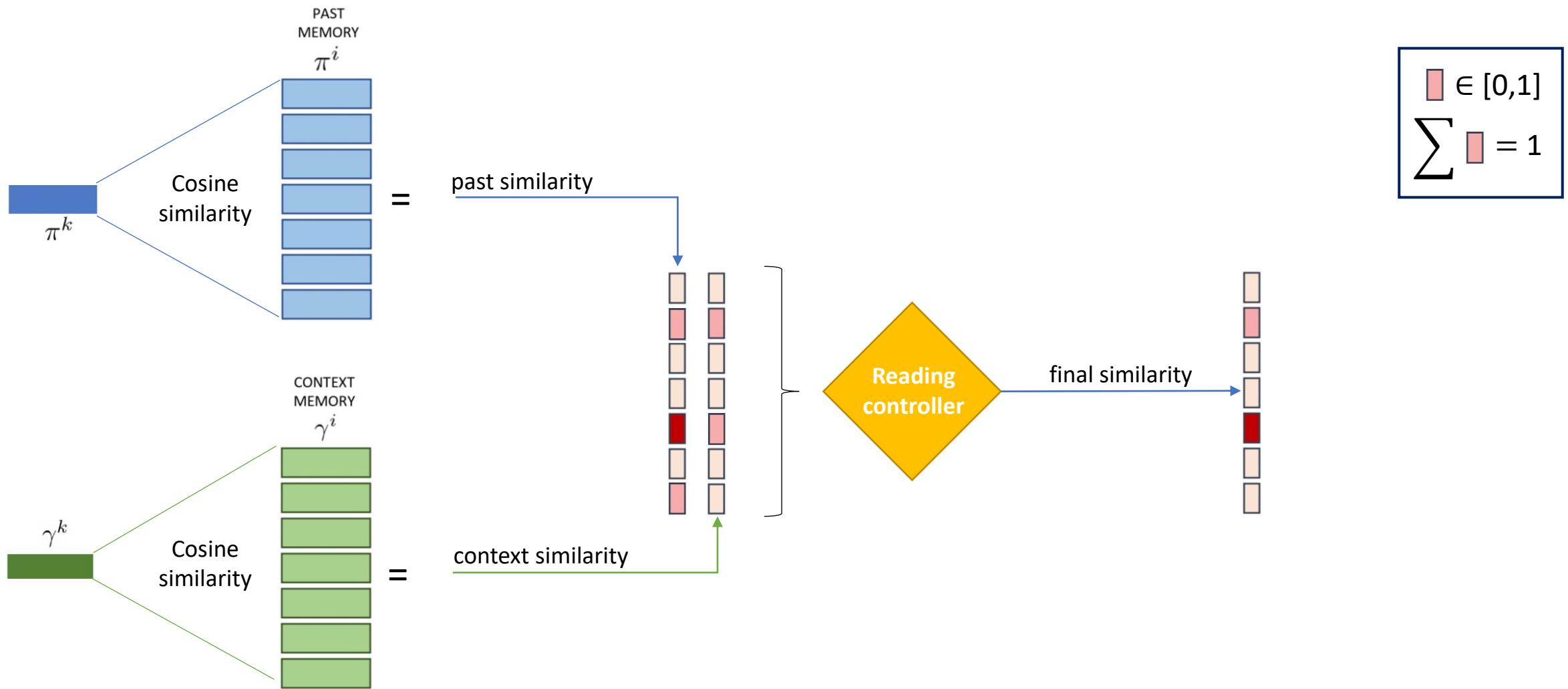
Reconstruction Loss

$$\mathcal{L}_w = e \cdot (1 - P(w)) + (1 - e) \cdot P(w)$$

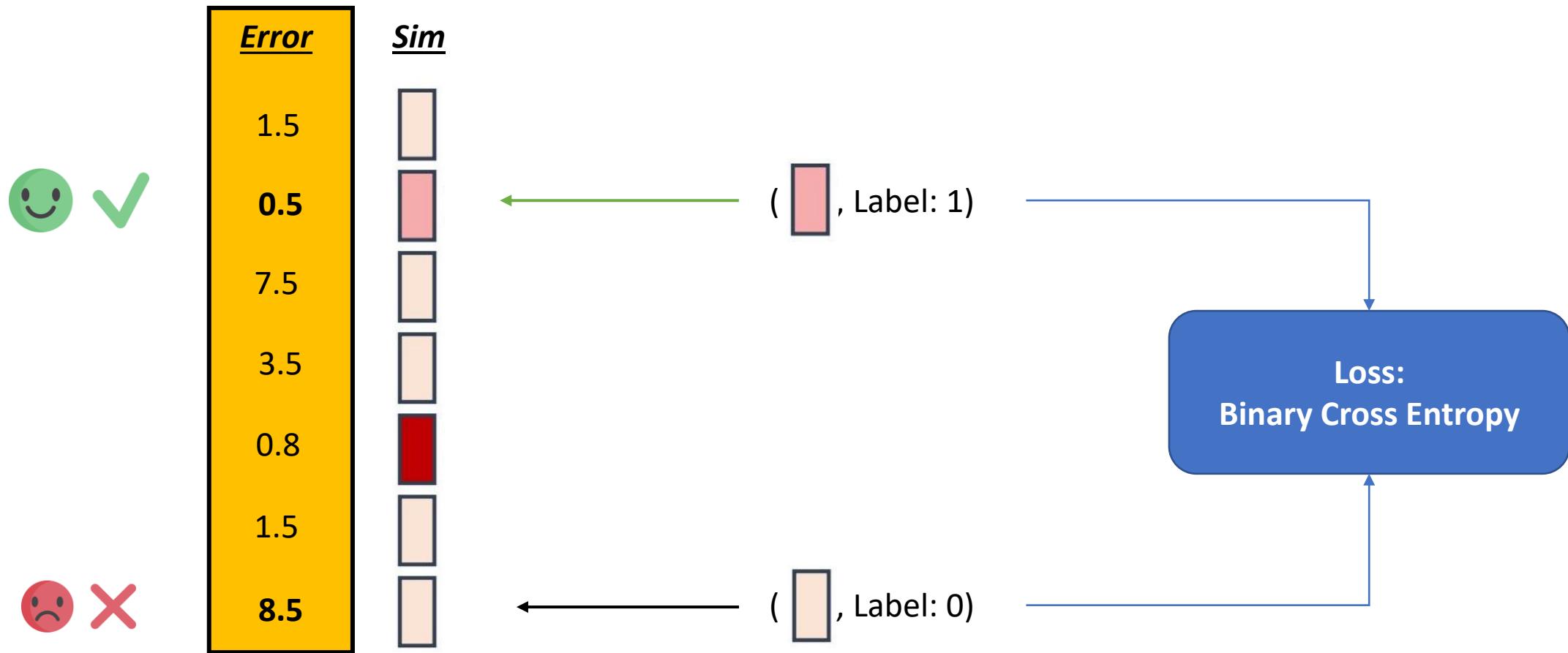
Low error  $\longrightarrow \mathcal{L}_w \approx P(w)$

High error  $\longrightarrow \mathcal{L}_w \approx 1 - P(w)$

# Training: reading controller



# Training: reading controller



# Quantitative Results

Method	ADE				FDE			
	1s	2s	3s	4s	1s	2s	3s	4s
Kalman	0.33	0.54	0.93	1.4	0.46	1.18	2.18	3.32
Linear	0.31	0.56	0.89	1.28	0.47	1.13	1.94	2.87
MLP	0.30	0.54	0.88	1.28	0.46	1.12	1.94	2.88
RNN Enc-Dec [78]	0.68	1.94	3.20	4.46	-	-	-	-
Markov [52]	0.70	1.41	2.12	2.99	-	-	-	-
Conv-LSTM (top 5) [52]	0.76	1.23	1.60	1.96	-	-	-	-
INFER (top 1) [52]	0.75	0.95	1.13	1.42	1.01	1.26	1.76	2.67
INFER (top 5) [52]	0.56	0.75	0.93	1.22	0.81	1.08	1.55	2.46
MANTRA (top 1)	0.37	0.67	1.07	1.55	0.60	1.33	2.32	3.50
MANTRA (top 5)	0.33	0.48	0.66	0.90	0.45	0.78	1.22	2.03
MANTRA (top 10)	0.31	0.43	0.57	0.78	0.43	0.67	1.04	1.78
MANTRA (top 20)	<b>0.29</b>	<b>0.41</b>	<b>0.55</b>	<b>0.74</b>	<b>0.41</b>	<b>0.64</b>	<b>1.00</b>	<b>1.68</b>

KITTI

Method	ADE		FDE	
	1s	3s	1s	3s
Kalman (top 1)	0.72	2.70	1.29	6.56
Linear (top 1)	0.58	1.95	0.98	4.58
MLP (top 1)	0.53	1.68	0.87	3.90
NN [1] (top 1)	0.75	2.46	1.28	5.60
NN + map [1] (top 6)	0.72	2.28	1.33	4.80
LSTM ED [1] (top 1)	0.68	2.27	1.78	5.19
LSTM ED + map [1] (top 6)	0.80	2.25	1.35	4.67
MFP [9] (top 6)	-	1.39	-	-
MANTRA (top 1)	0.72	2.36	1.25	5.31
MANTRA (top 6)	0.56	1.22	0.84	2.30
MANTRA (top 10)	0.53	1.00	0.77	1.69
MANTRA (top 20)	<b>0.52</b>	<b>0.84</b>	<b>0.73</b>	<b>1.16</b>

ARGOVERSE

# Qualitative Results

KITTI



ARGOVERSE



(a) Linear

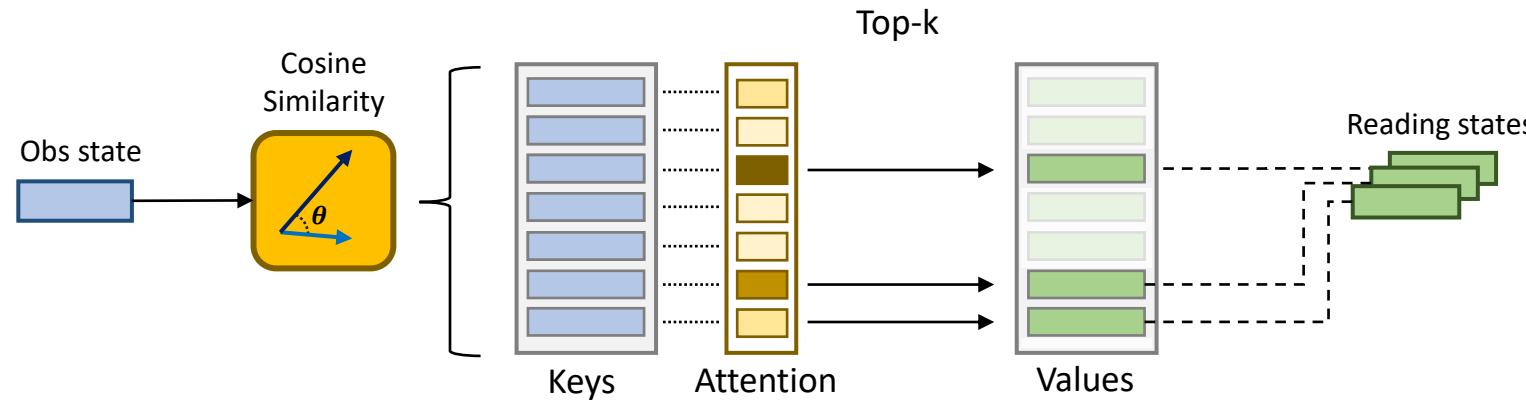
(b) Kalman

(c) MANTRA

# DEMO: Carla simulator



# Reading Controller



Similarity function to retrieve the top-K samples which are used to **individually** generate K different futures.



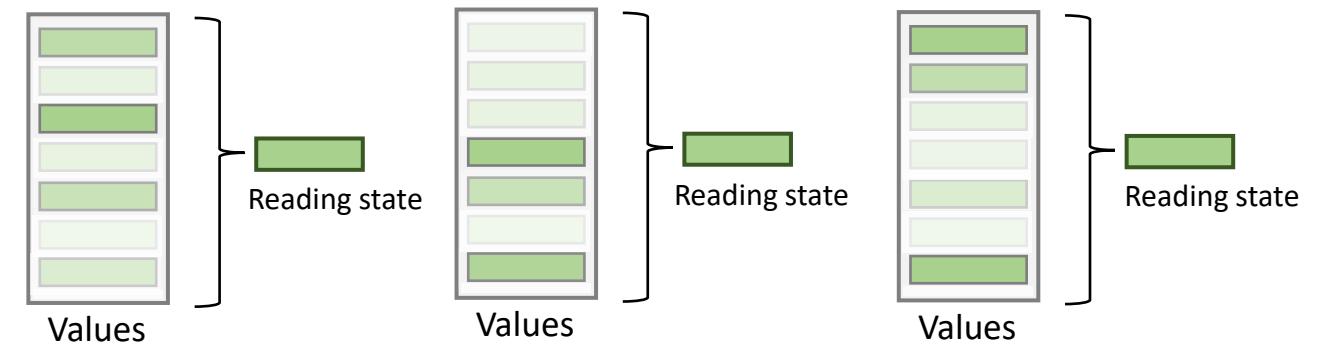
Only one element is responsible for generating a future trajectory.



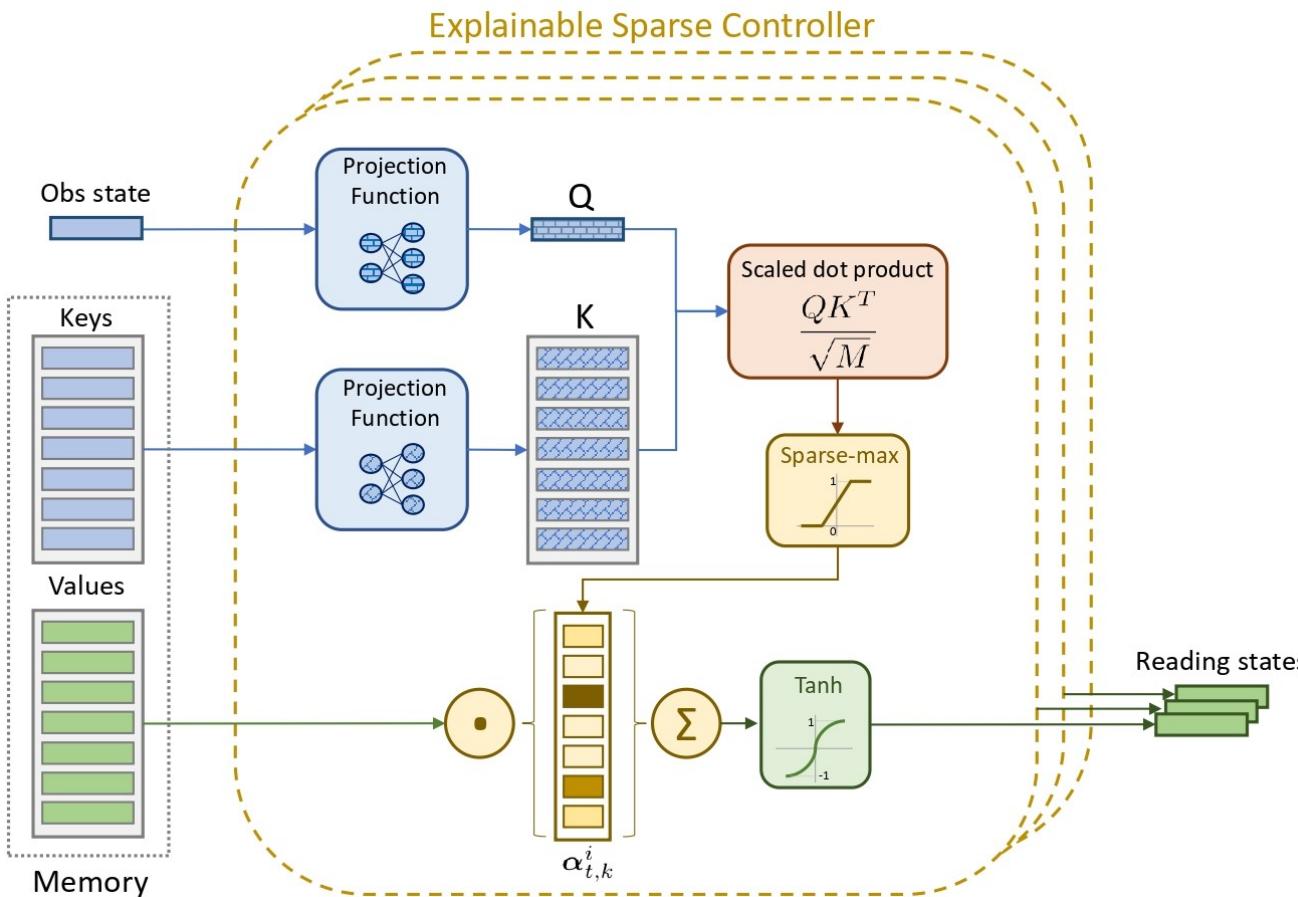
**Combination** of features from **different** samples in memory for each prediction.



Better **generalization** capabilities.  
Robust to outliers or corrupted samples.



# Explainable Sparse Attention Controller (ESA)



Multiple read heads extract different information from memory using different learnable **projection function**.



Query  $Q$  and Key matrix  $K$  are compared using **scaled dot product**.



**Attention** is created by a sparse-max function.



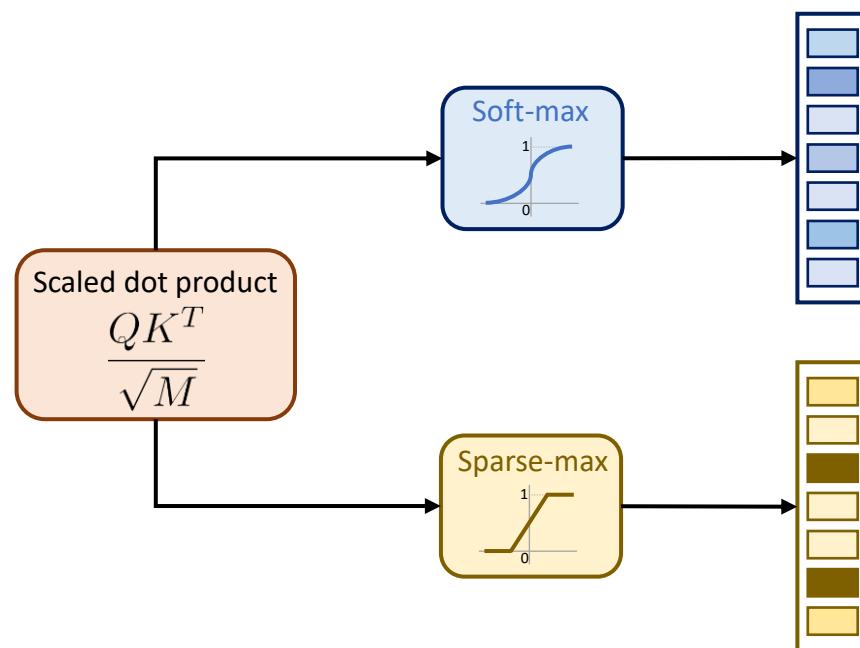
**Reading state** is a weighted sum of memory Values with attention scores.

# Sparse-max Attention<sup>[4]</sup>

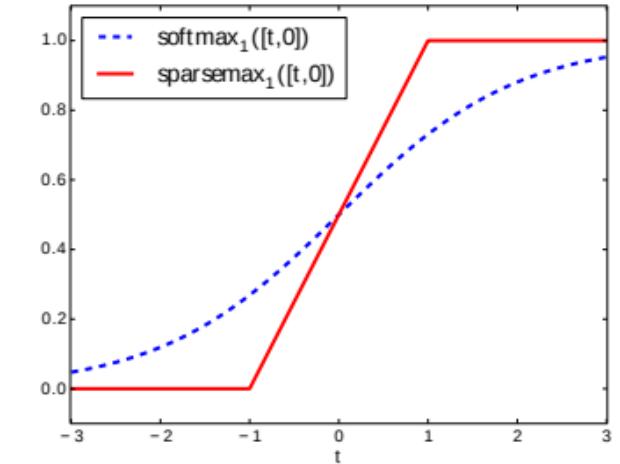


Euclidean projection of the input vector onto the **probability simplex**.

The projected input hits the boundary of simplex making the **output sparse**.



Focus only on few relevant elements to the current observation.  
**Small subset of elements** enables a better model explainability.



# Experiments: Dataset & Metrics

## Mantra on KITTI

Hours of navigation in rural areas and highways of mid-size city, Karlsruhe.

**Past: 2.0s – Future: 4.0s**

## Mantra-M on ARGOVERSE

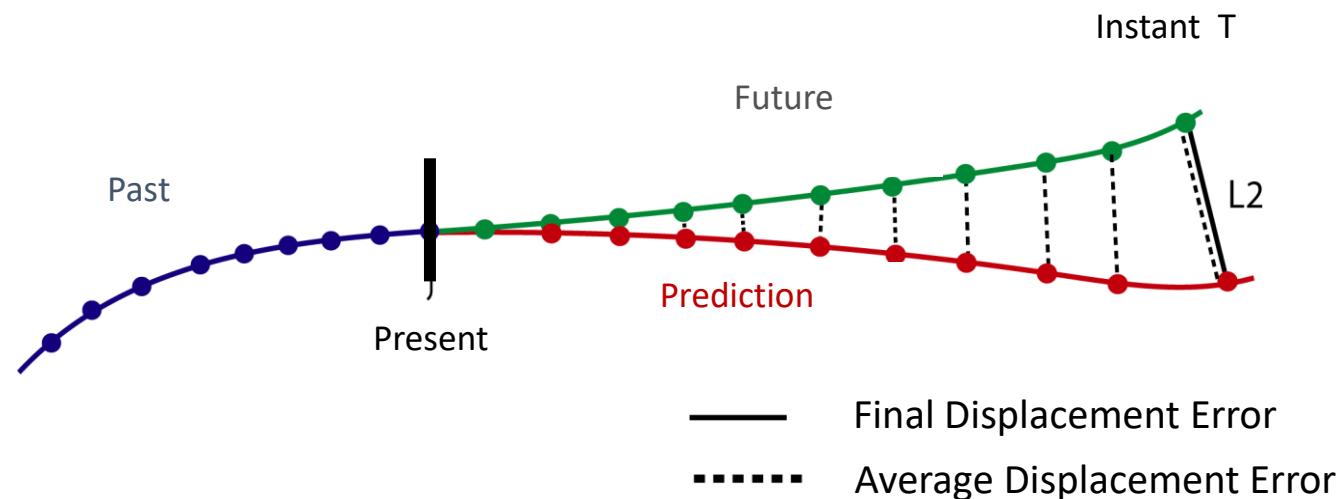
Trajectories acquired in an area of  $1000\text{km}^2$  in the cities of Pittsburgh and Miami.

**Past: 2.0s – Future: 3.0s**

## Memonet on SDD

Pedestrians and bicycles acquired by a bird's eye view drone on university campus.

**Past: 3.2s – Future: 4.8s**



In multi-modal setting, we take the best out of K predictions.

# Quantitative Results

KITTI

	Method	ADE				FDE			
		1s	2s	3s	4s	1s	2s	3s	4s
K=1	Kalman [32]	0.51	1.14	1.99	3.03	0.97	2.54	4.71	7.41
	Linear [32]	0.20	0.49	0.96	1.64	0.40	1.18	2.56	4.73
	MLP [32]	<b>0.20</b>	<b>0.49</b>	0.93	1.53	<b>0.40</b>	1.17	2.39	4.12
	DESIRE [25]	-	-	-	-	0.51	1.44	2.76	4.45
	MANTRA [32]	0.24	0.57	1.08	1.78	0.44	1.34	2.79	4.83
	MANTRA+ESA	0.24	0.50	<b>0.91</b>	<b>1.48</b>	0.41	<b>1.13</b>	<b>2.30</b>	<b>4.01</b>
K=5	SynthTraj [3]	0.22	0.38	0.59	0.89	0.35	0.73	1.29	2.27
	DESIRE [25]	-	-	-	-	<b>0.28</b>	0.67	1.22	<b>2.06</b>
	MANTRA [32]	<b>0.17</b>	0.36	0.61	0.94	0.30	0.75	1.43	2.48
	MANTRA+ESA	0.21	<b>0.35</b>	<b>0.55</b>	<b>0.83</b>	0.31	<b>0.66</b>	<b>1.20</b>	2.11
K=20	DESIRE [25]	-	-	-	-	-	-	-	2.04
	MANTRA [32]	<b>0.16</b>	0.27	0.40	0.59	0.25	0.49	0.83	1.49
	MANTRA+ESA	0.17	0.27	<b>0.38</b>	<b>0.56</b>	<b>0.24</b>	<b>0.47</b>	<b>0.76</b>	<b>1.43</b>

Unit of measurement: meters

↓ 17.18% FDE@4s

↓ 14.91% FDE@4s

ARGOVERSE

	Method	ADE		FDE		Off-Road (%)	Memory Size
		1s	3s	1s	3s		
K=1	MANTRA-M [33]	0.72	2.36	1.25	5.31	<b>1.62%</b>	75,424
	MANTRA-M+ESA	<b>0.58</b>	<b>1.76</b>	<b>0.96</b>	<b>3.95</b>	1.84%	<b>9,701</b>
K=6	MANTRA-M [33]	0.56	1.22	0.84	2.30	3.27%	12,467
	MANTRA-M+ESA	<b>0.47</b>	<b>0.93</b>	<b>0.68</b>	<b>1.57</b>	<b>2.32%</b>	<b>2337</b>
K=10	MANTRA-M [33]	0.53	1.00	0.77	1.69	4.17%	6,566
	MANTRA-M+ESA	<b>0.44</b>	<b>0.80</b>	<b>0.63</b>	<b>1.20</b>	<b>2.98%</b>	<b>1,799</b>
K=20	MANTRA-M [33]	0.52	0.84	0.73	1.16	7.93%	2,921
	MANTRA-M+ESA	<b>0.45</b>	<b>0.73</b>	<b>0.65</b>	<b>0.88</b>	<b>3.14%</b>	<b>1,085</b>

↓ 81.25% Memory Size

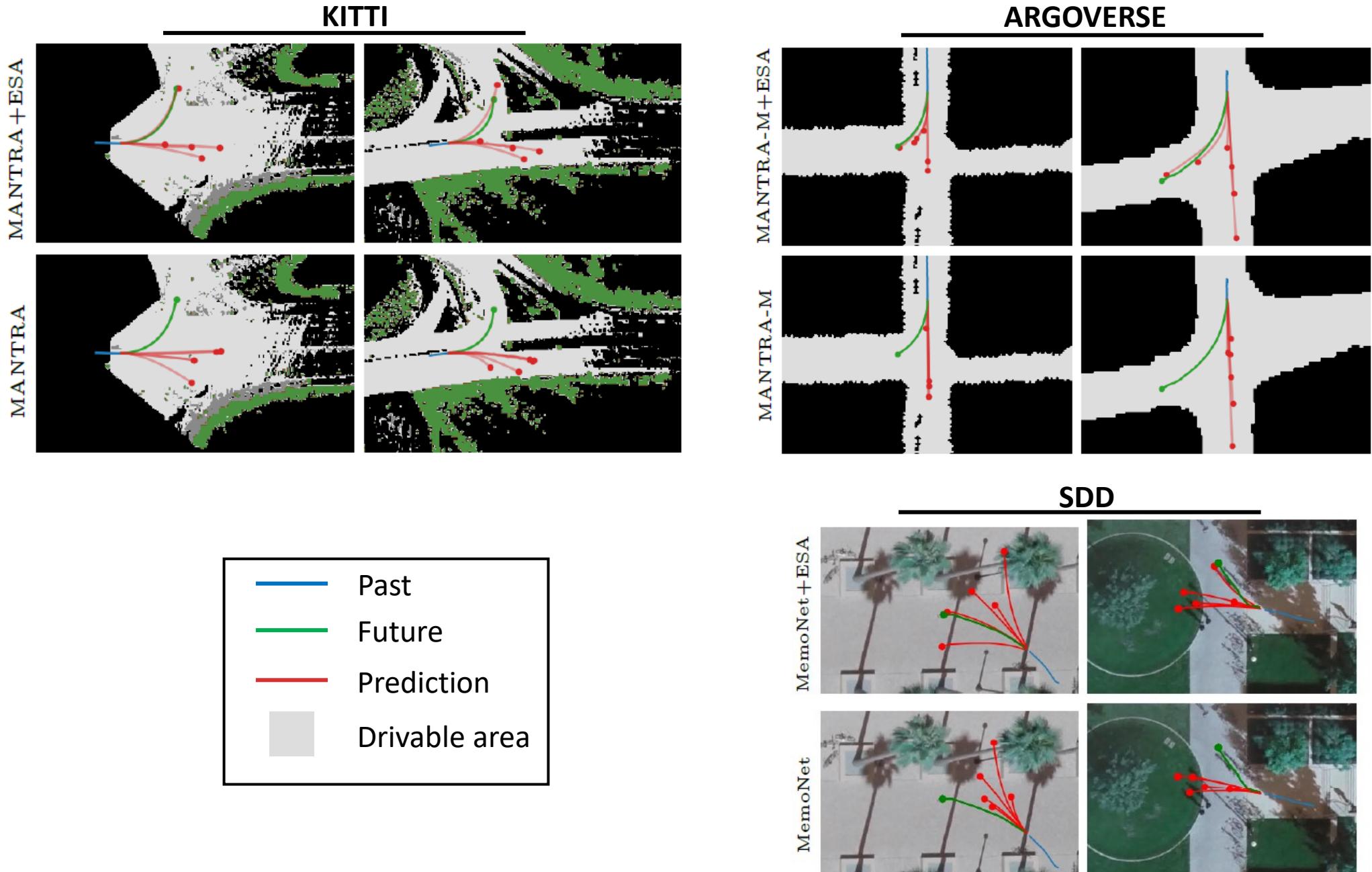
↓ 60.40% Off-Road

# Quantitative Results

## Unit of measurement: pixels

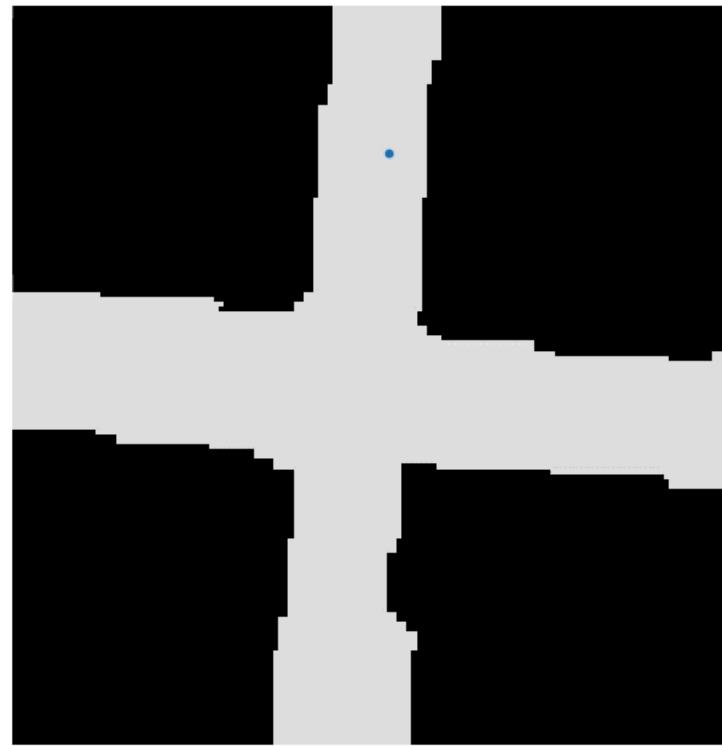
	K=20						K=5						
	Method		ADE		FDE			Method		ADE		FDE	
SDD	Social-STGCNN [36]		20.60		33.10		SimAug [27]		10.27		19.71		
	Trajectron++ [45]		19.30		32.70		MANTRA [32]		8.96		17.76		
	SoPhie [44]		16.27		29.38		PCCSNet [51]		8.62		16.16		
	NMMP [36]		14.67		26.72		PECNet [31]		9.96		15.88		
	EvolveGraph [26]		13.90		22.90		LB-EBM [37]		8.87		15.61		
	EvolveGraph [26]		13.90		22.90		Expert-Goals [19]		<b>7.69</b>		14.38		
	CF-VAE [4]		12.60		22.30		SMEMO [34]		8.11		13.06		
	Goal-GAN [9]		12.20		22.10		MemoNet [54]		8.56		<b>12.66</b>		
	P2TIRL [11]		12.58		22.07		MemoNet+ESA		<b>8.02</b>		12.97		

# Qualitative Results



# Explainability

Plot of semantic maps and trajectories weighed by ESA attention.



Argoverse example

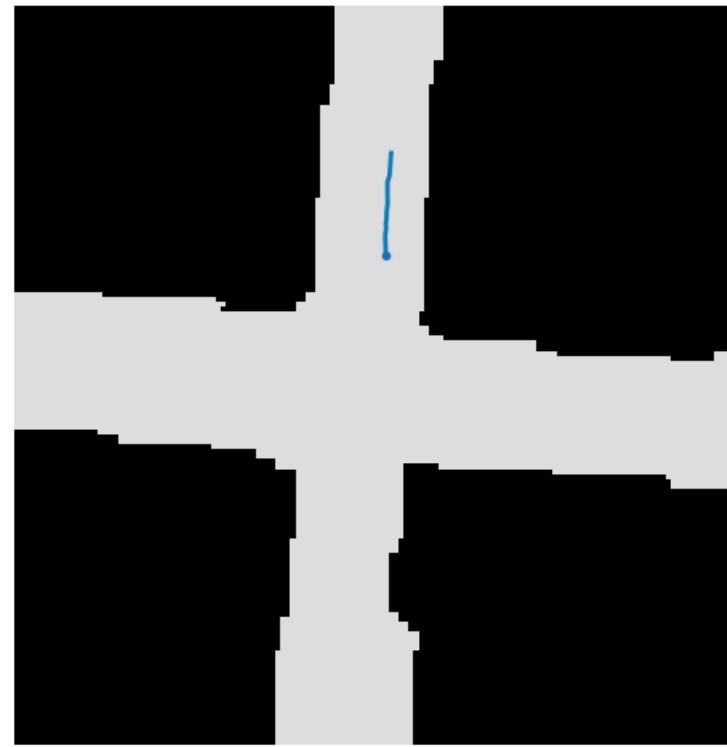
Evident **cause-effect relationship** between memory reading and the generated output



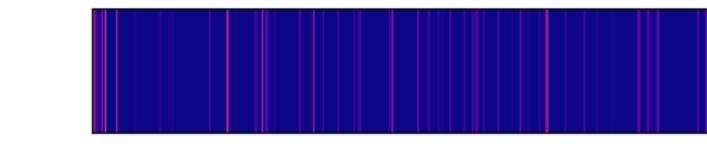
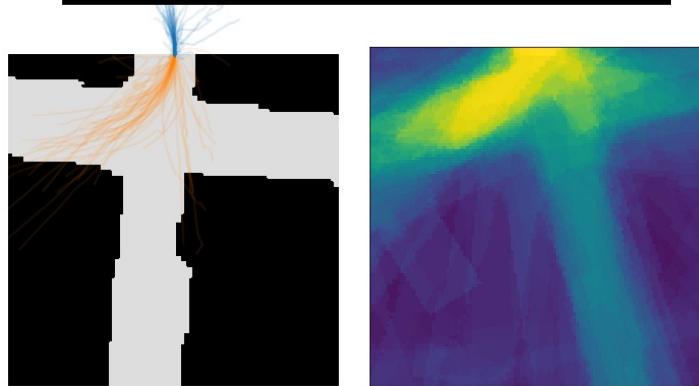
Attention vector over memory locations



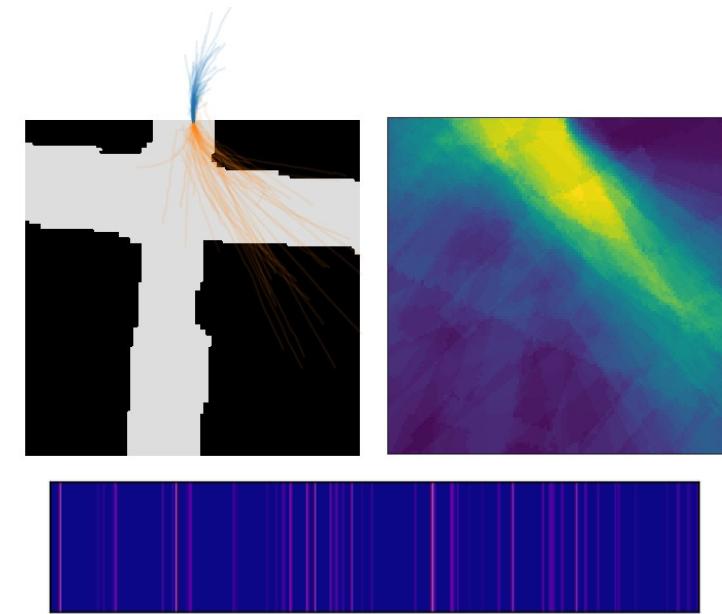
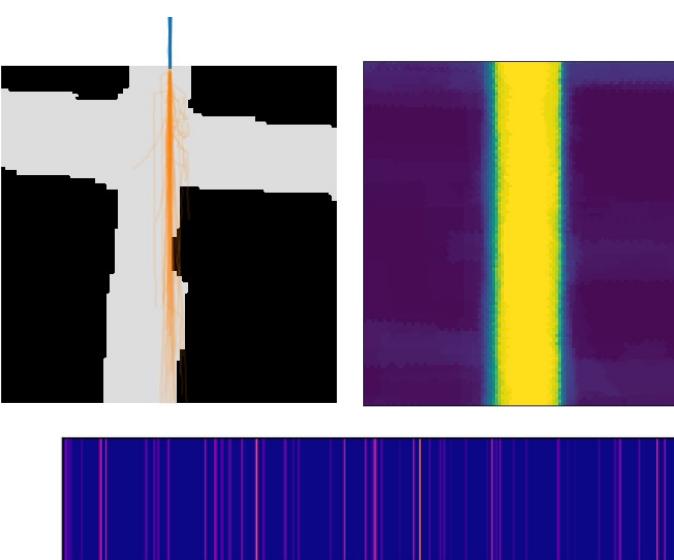
# Explainability



Plot of semantic maps and trajectories weighed by ESA attention.



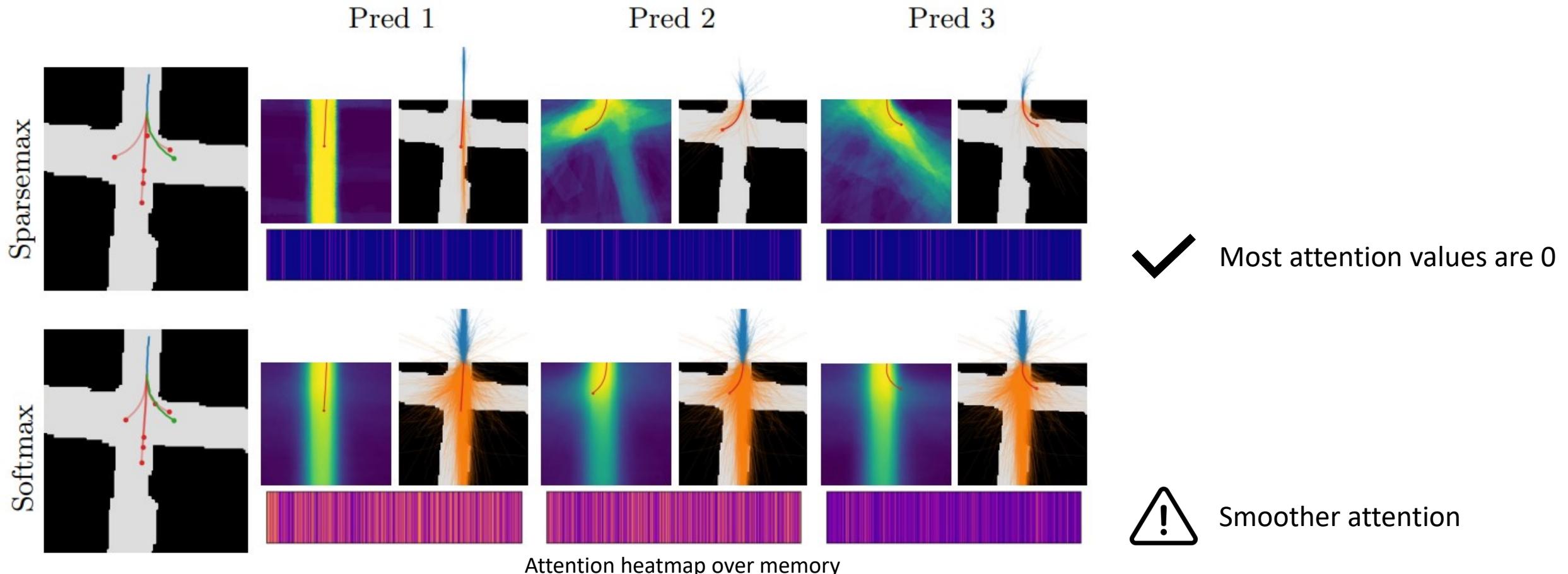
Attention vector over memory locations



Argoverse example

Evident **cause-effect relationship** between memory reading and the generated output

# Explainability

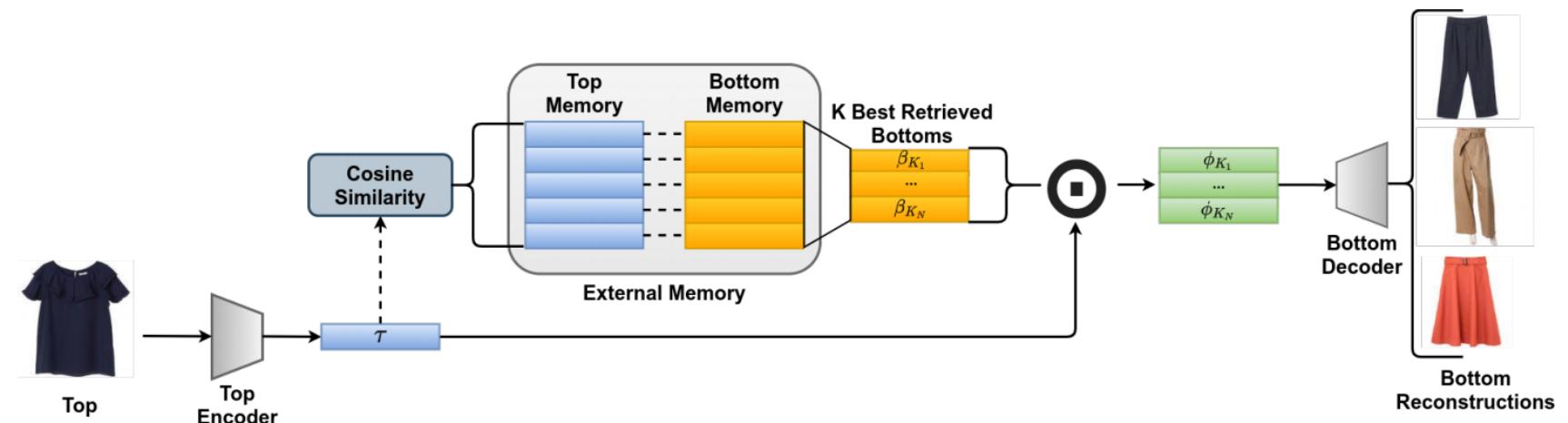
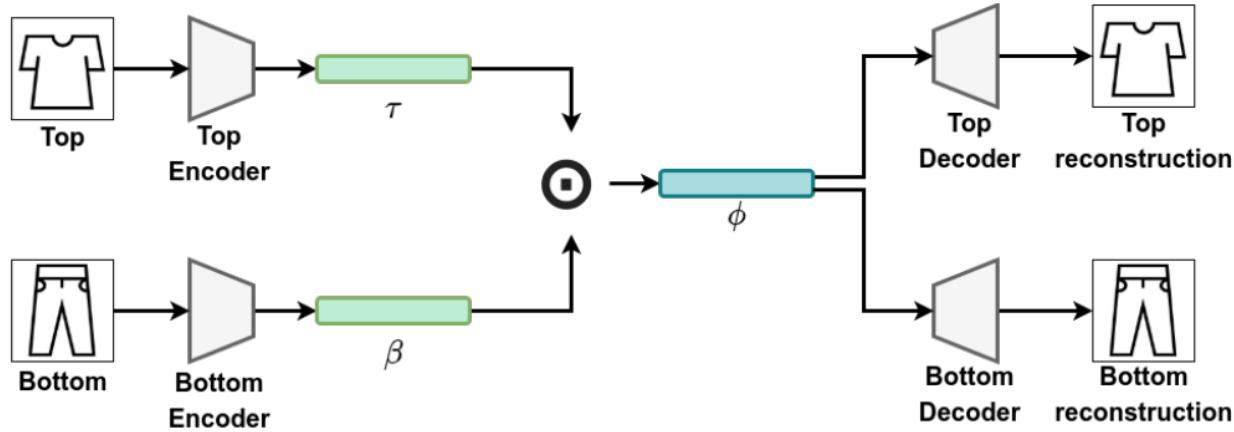


**Sparse-max:** Maps and tracks with positive attentions can be interpreted as a scenario consistent with prediction.

**Soft-max:** Soft attention vector, no element in memory is clearly identifiable as responsible of the prediction.

# Garment Recommendation with Memory Augmented Neural Networks

L. De Divitiis, ICPR 2020



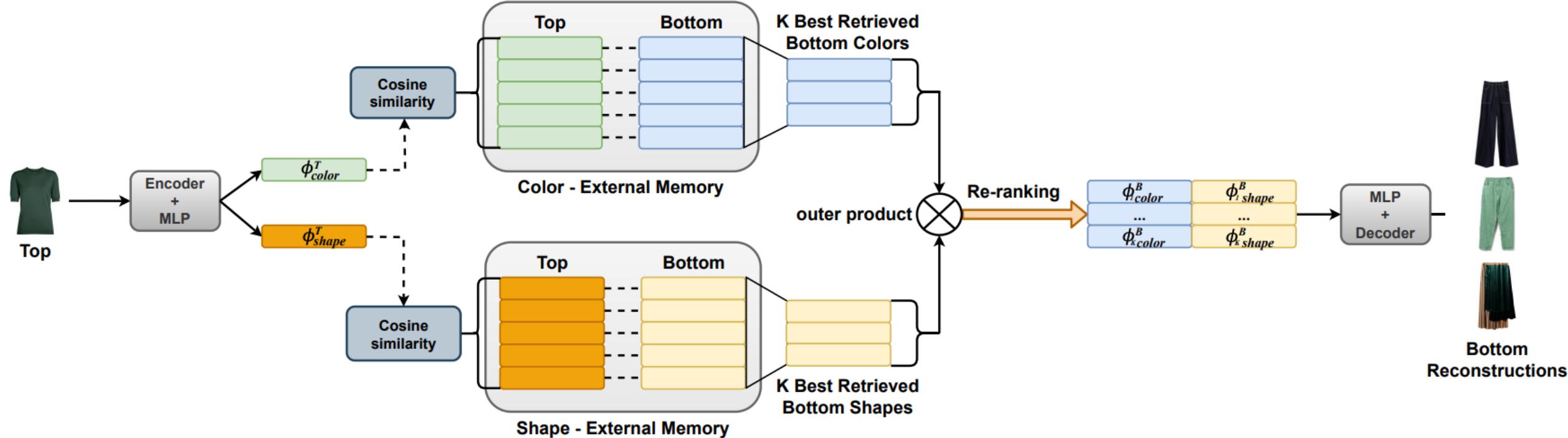
# Garment Recommendation with Memory Augmented Neural Networks

Ground Truth	Retrieved bottoms									
										
										
										
										

# Garment Recommendation with Memory Augmented Neural Networks

L. De Divitiis et al., TOMM 2022

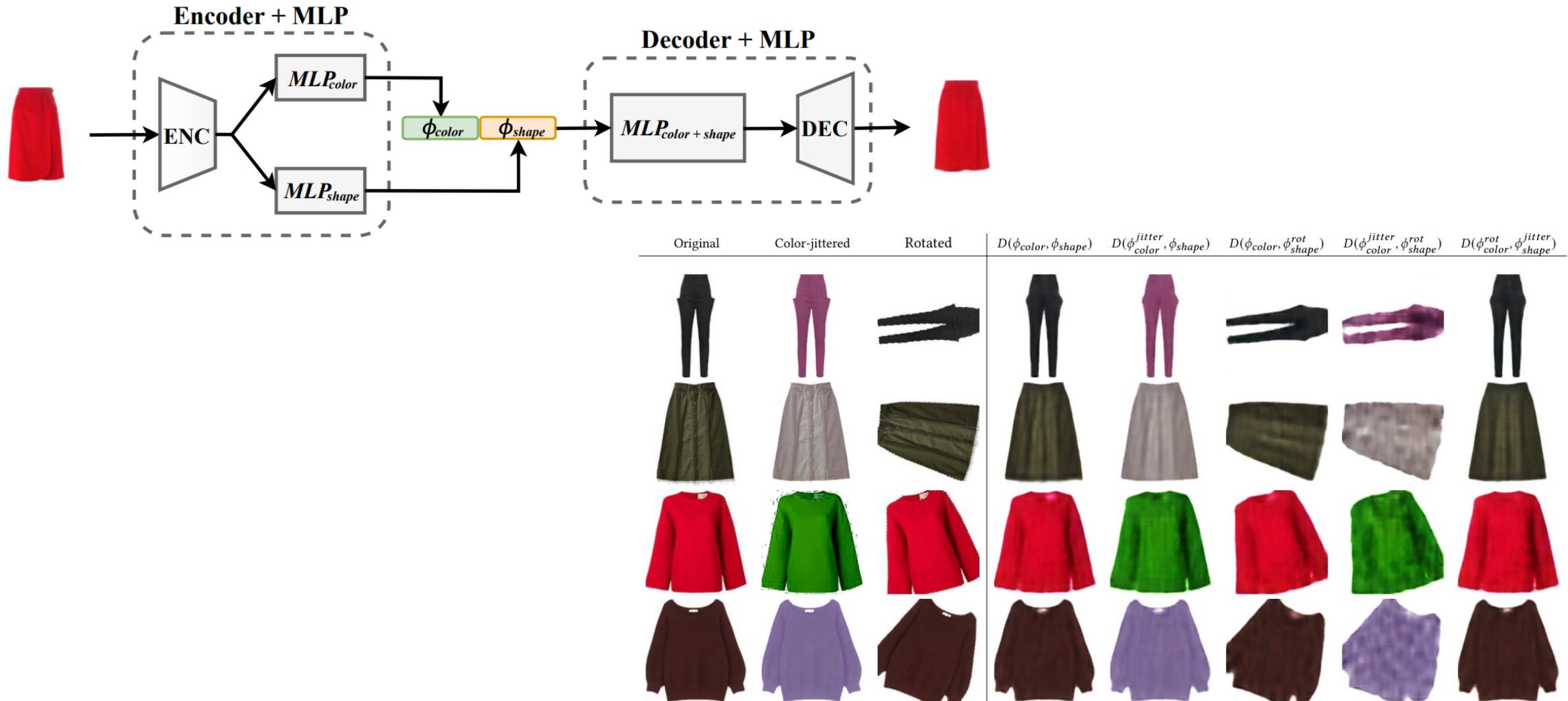
Disentangled features corresponding to color and shape are stored in two different memory banks.



# Garment Recommendation with Memory Augmented Neural Networks

L. De Divitiis et al., TOMM 2022

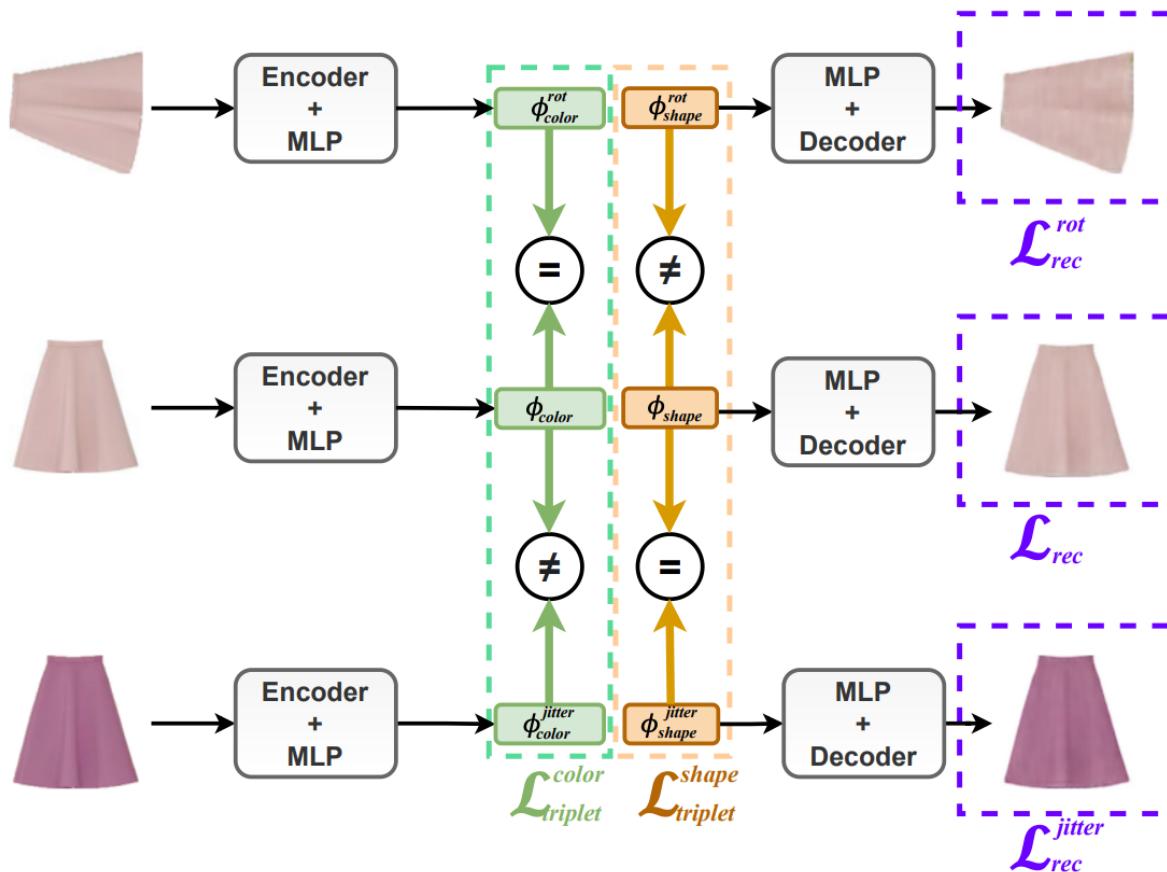
Disentangled features are learned with an autoencoder with two internal states plus a triplet loss.



# Garment Recommendation with Memory Augmented Neural Networks

L. De Divitiis et al., TOMM 2022

Disentangled features are learned with an autoencoder with two internal states plus a triplet loss.



$$\mathcal{L} = \mathcal{L}_{rec} + \mathcal{L}_{rec}^{rot} + \mathcal{L}_{rec}^{jitter} + \lambda(\mathcal{L}_{triplet}^{color} + \mathcal{L}_{triplet}^{shape})$$

Reconstruction losses                                           Triplet losses

# Garment Recommendation with Memory Augmented Neural Networks

L. De Divitiis et al., TOMM 2022

The following memory controller loss is used:  $\mathcal{L}_{controller} = d^* \cdot (1 - p_w) + (1 - d^*) \cdot p_w$

When the reconstruction error  $d^*$  is not normally distributed and does not cover the entire [0, 1] range, the writing controller can converge to trivial solutions such as «write everything» or «write nothing».

The loss is regularized by normalizing  $d^*$  and by adding a penalty that enforces the following behaviour:

- samples with errors higher than the Nth-percentile of  $d^*$  should be written in memory
- samples with errors lower than the Nth-percentile of  $d^*$  should not be written in memory

$$\mathcal{L}_{penalty} = \begin{cases} th_w - p_w + m & \text{if } p_w < th_w \quad \& \quad d^* > perc_N \\ p_w - th_w + m & \text{if } p_w > th_w \quad \& \quad d^* < perc_N \end{cases} \quad th_w = \frac{perc_N}{d_{max}^*}$$

$$\mathcal{L}^*_{controller} = \frac{d^*}{d_{max}^*} \cdot (1 - p_w) + \left(1 - \frac{d^*}{d_{max}^*}\right) \cdot p_w + \alpha \cdot \mathcal{L}_{penalty}$$

## PART3 - *Transformers*

# Attention and memory

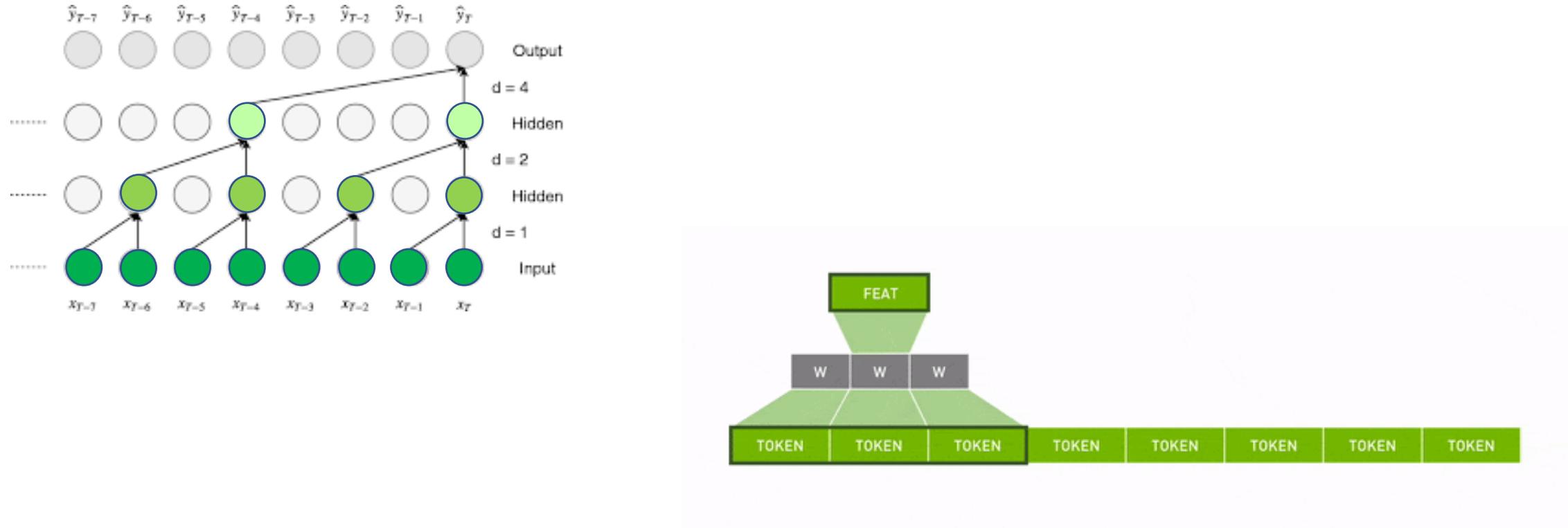
Talking of memory in computer systems we refer to their storage capacity. In this sense computers have much better memory than people as they are able to store everything.

Memory in humans is different. Human memory has a limited capacity, and thus attention determines what will be encoded. Human memory is rather the ability to select information and attend to that.

Memory is attention over time. Attention and memory are important features of human cognition. They cannot operate without each other.

These intertwined concepts have been used differently in deep learning systems.

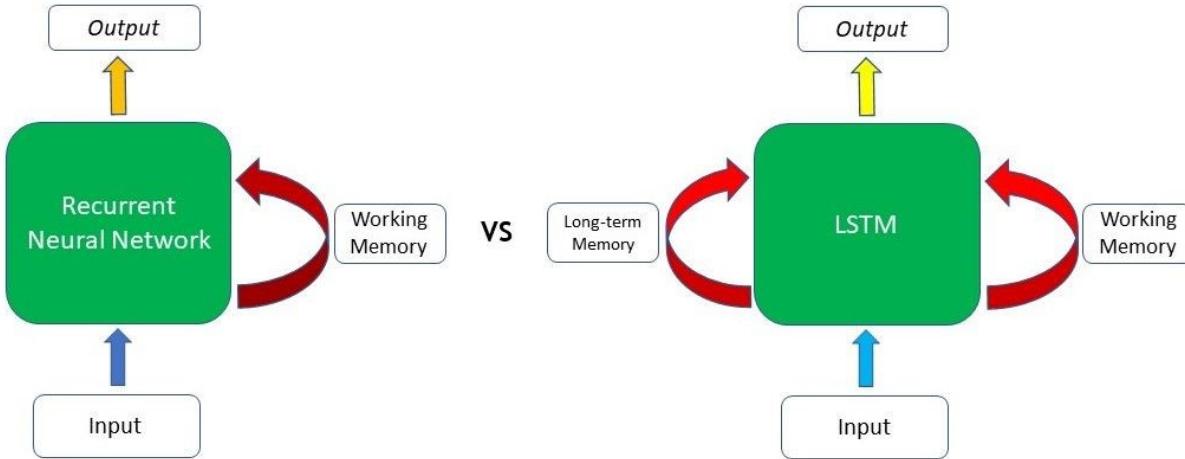
# Attention in CNNs



In a CNN, parallelization is trivial per layer. Convolutional filters extract local correlations within the filter window. The maximum range of a dependency that can be learnt in a layer is the size of the convolutional filter.

Long range dependencies of length  $n$  will require **O(log n)** layers (distance between positions is logarithmic)

# Attention in RNNs

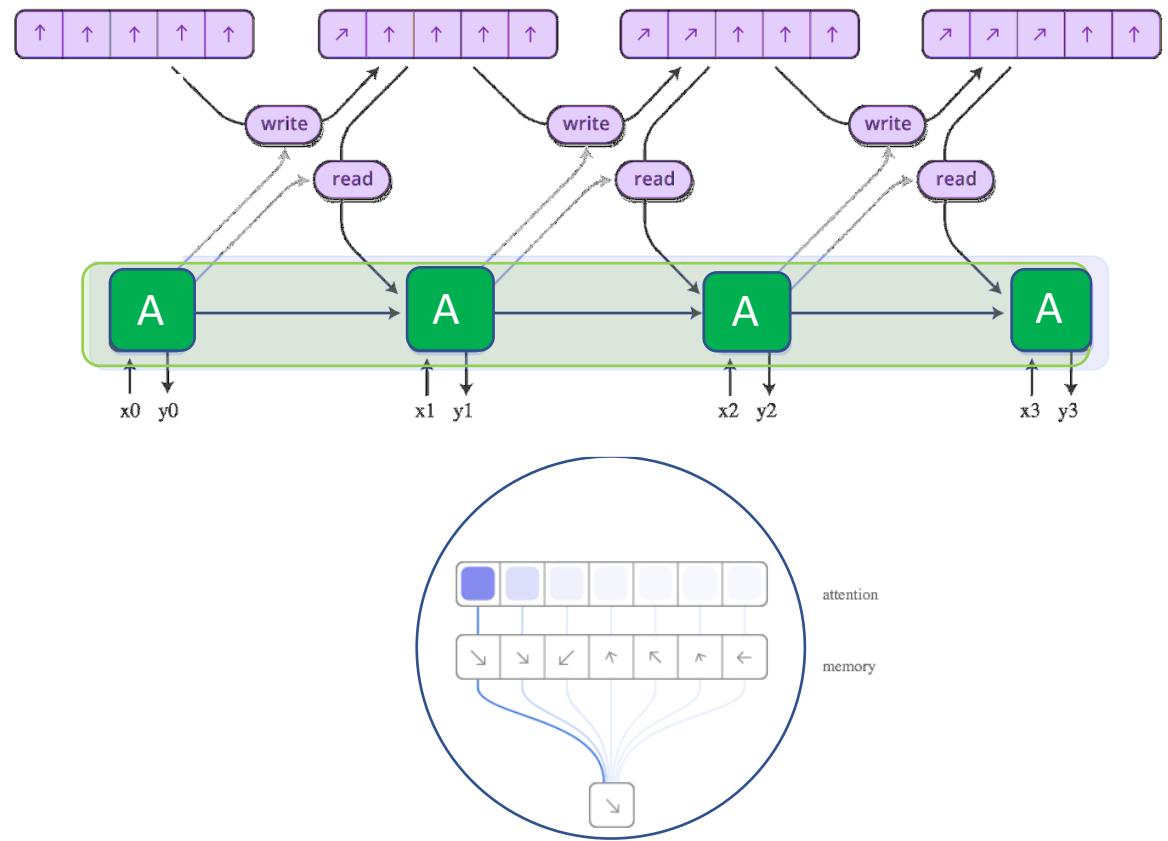
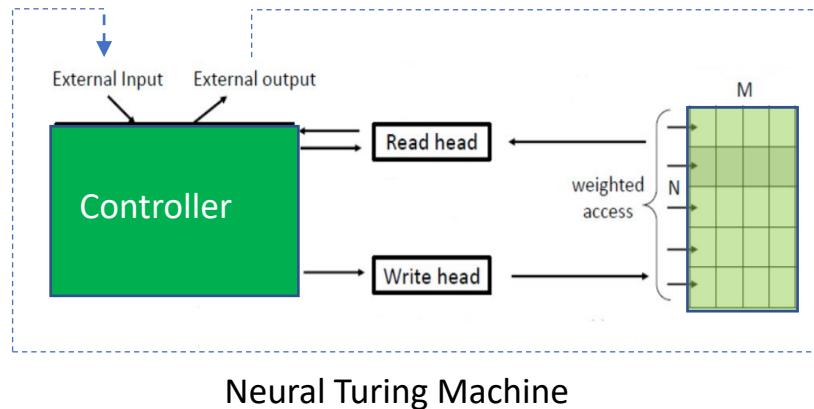


In RNNs and LSTMs sequential computation inhibits parallelization.

RNNs and LSTMs do not have a mechanism for modelling item-specific time dependencies.

The entire information is modified, and there is no consideration of what is important and what is not.

# Attention in MANNs



In MANNs, the Controller executes sequentially.

Memory is an array of vectors individually addressable. Memory entries can be modified.

An attention vector specifies the magnitude of each memory state that should be extracted.

# Transformers

Transformers are a type of Encoder-Decoder model. Transformers were developed to solve the problem of sequence transduction, or neural machine translation.

For models to perform sequence transduction, it is necessary to have some sort of memory.

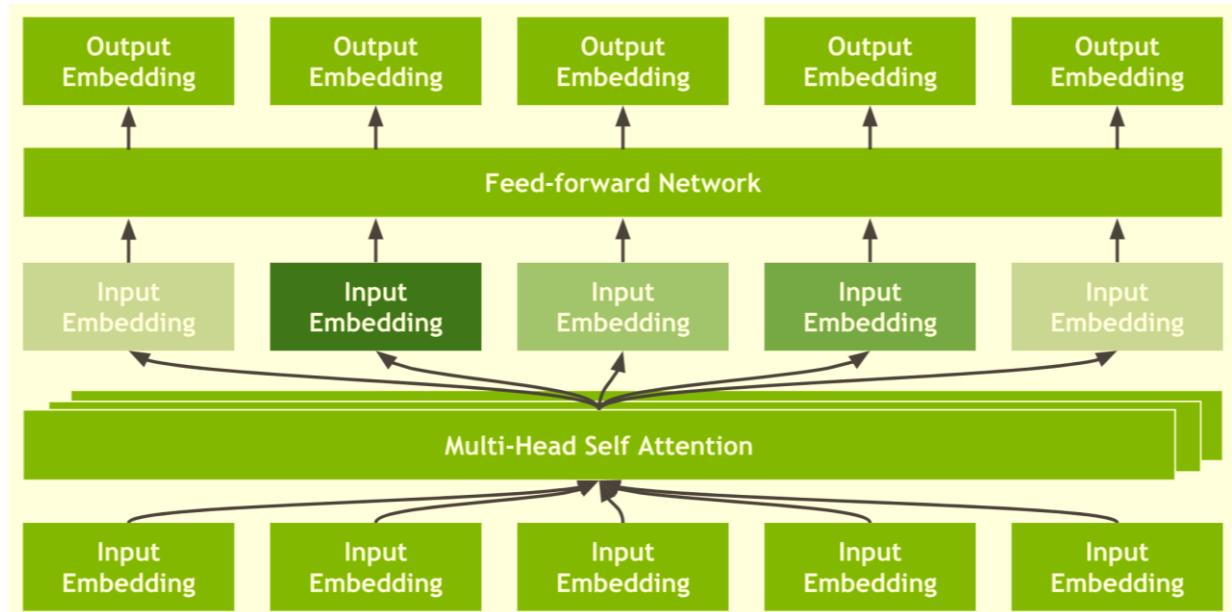


# Transformers

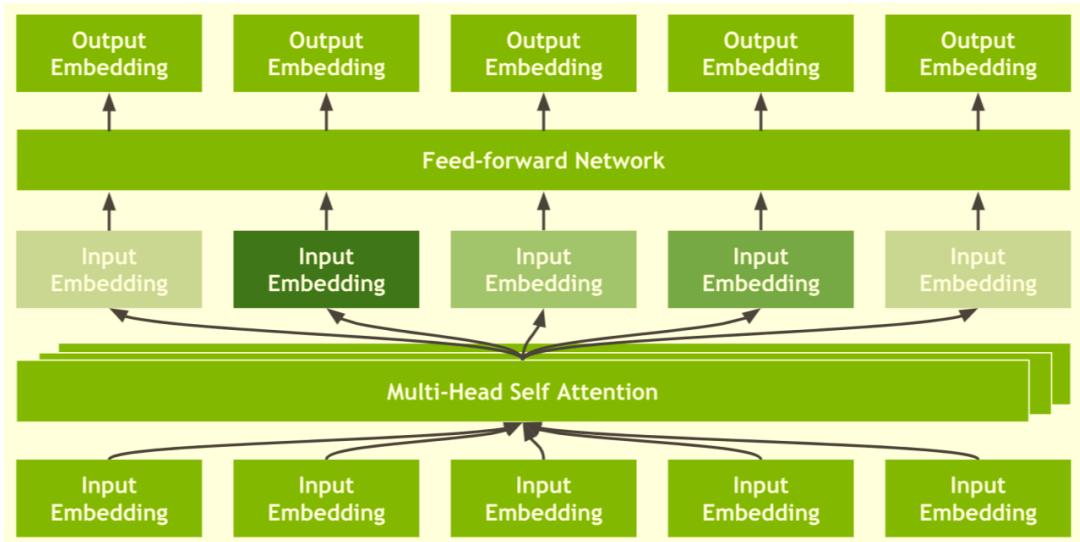
Transformers make use of *multi-headed self-attention* to perform sequence to sequence tasks such as language modelling and machine translation.

Self-attention is used to learn long-range dependencies between the elements in a sequence.

*Multi-head self-attention* is the combination of several attention heads. This is conceptually similar to how a convolutional layer can consist of multiple convolution filters, with each filter independently extracting different types of features.

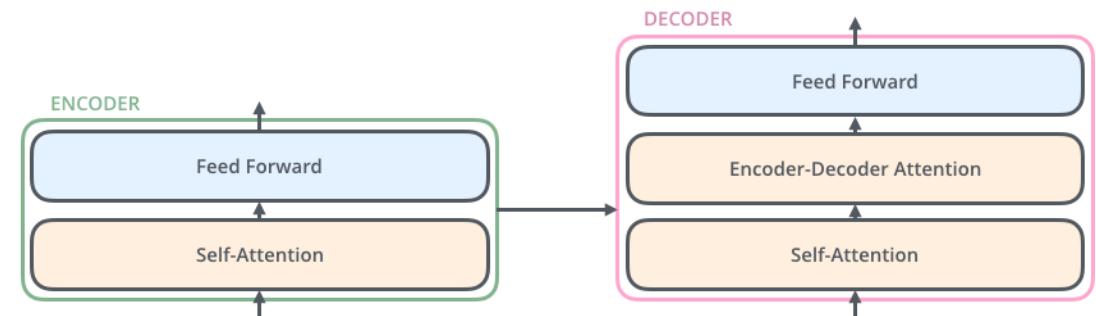
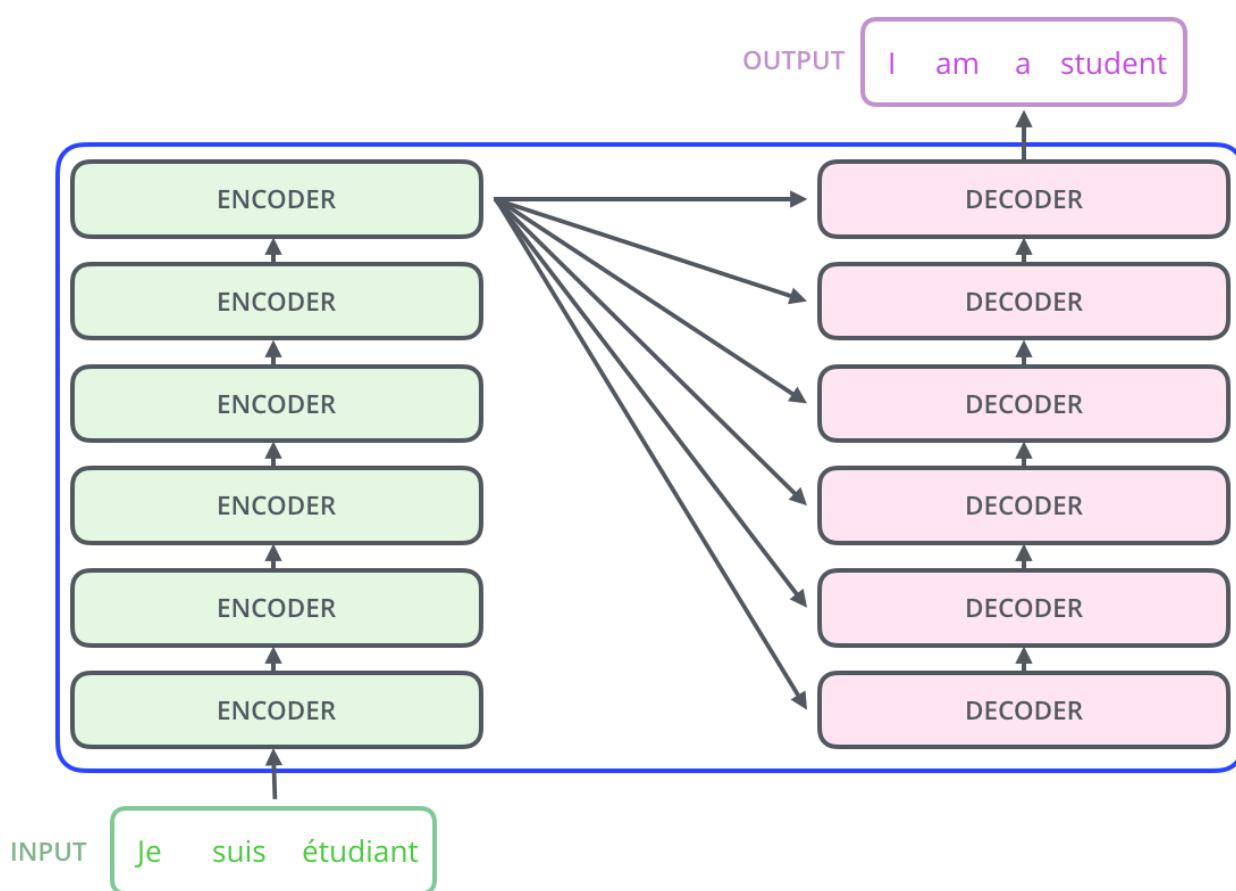


# Attention in transformers



The attention mechanism performs a lookup producing a set of weights for each element.  
The most relevant elements have the highest attention scores.

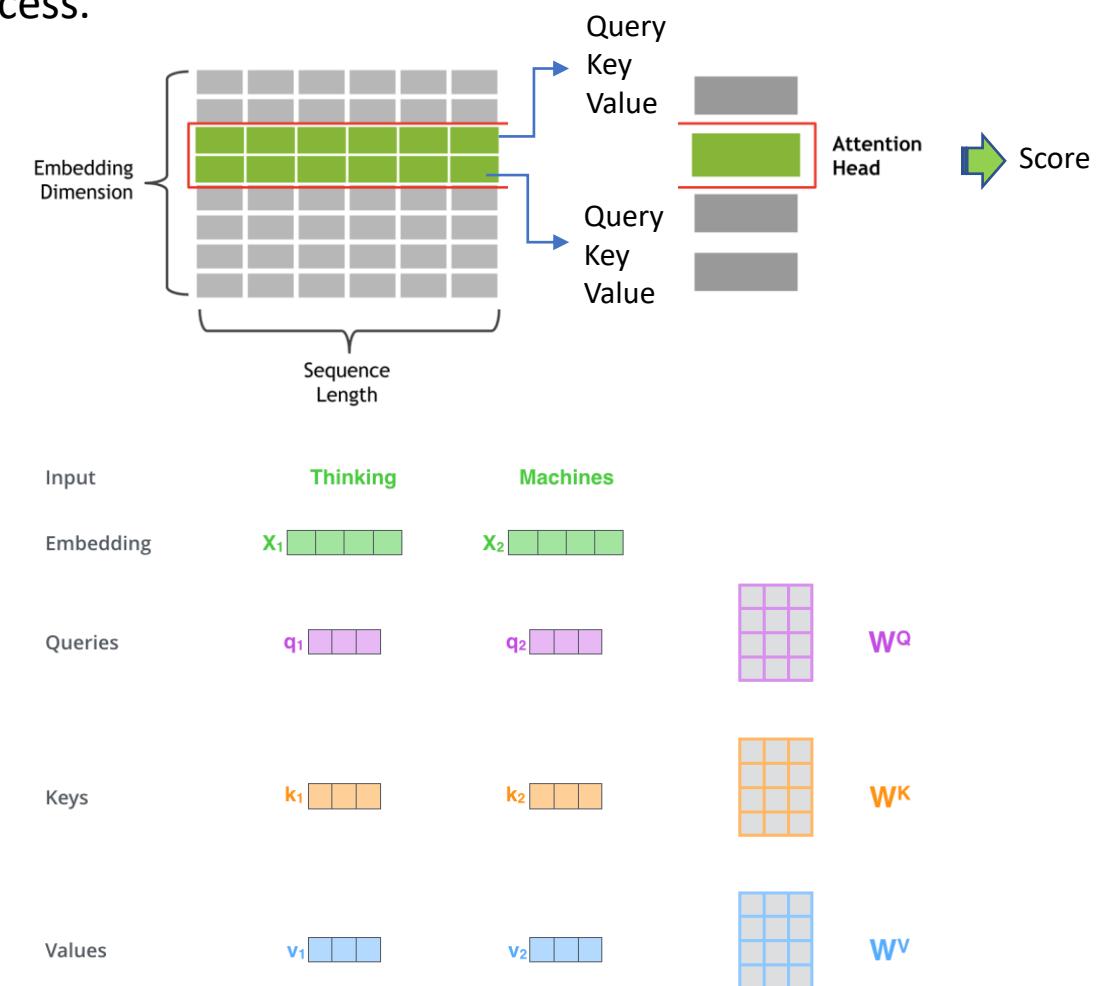
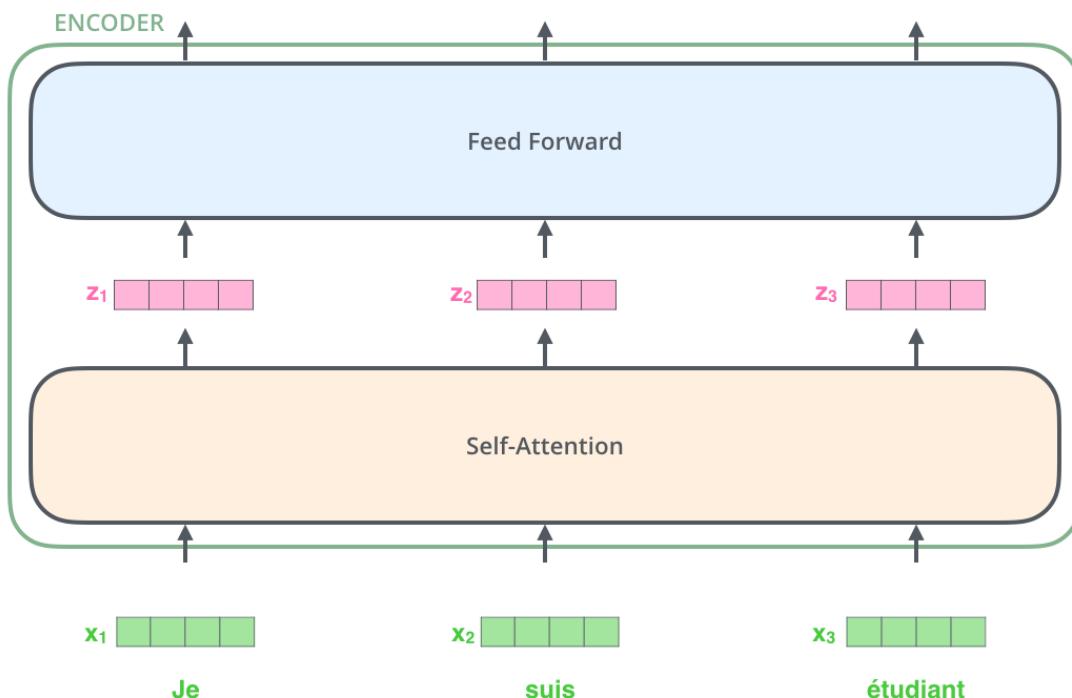
# Transformer encoder-decoder architecture



# Self-attention mechanism

Self-attention is a mechanism used to build representations based on the pair-wise correlations between the elements in a sequence. Each layer has a complexity of  $O(n^2)$  for sequences of length  $n$ . Outputs are weighted sums of the inputs.

From each of the encoder's input vectors a Query vector, a Key vector, and a Value vector are created by multiplying the embedding by three matrices learned during the training process.

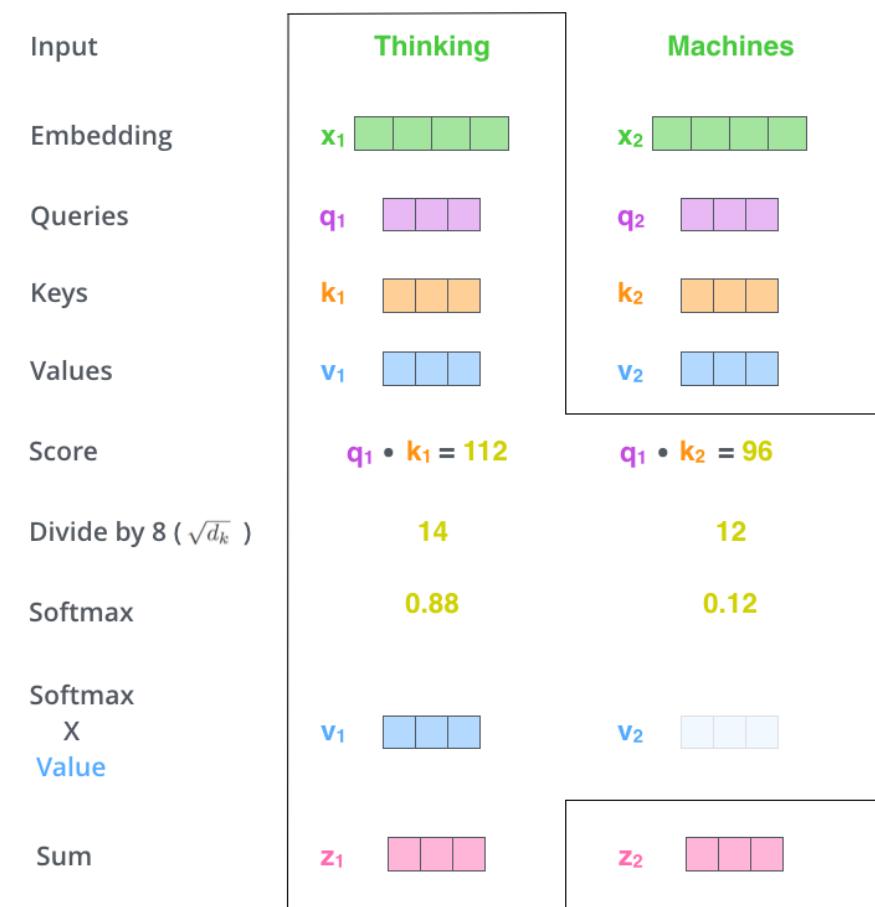


# Self-attention mechanism

The attention scores are calculated between the query vector Q of each element and the key vector K of every other element in the sequence. The computed attention scores are softmax-normalized and used as weights.

Multiplying each value vector by the softmax score and summing them up let us know how much another item is relevant to the current item.

$$\text{softmax}\left(\frac{\begin{matrix} \mathbf{Q} & \mathbf{K}^T \\ \begin{matrix} \times \end{matrix} & \begin{matrix} \mathbf{K}^T \end{matrix} \end{matrix}}{\sqrt{d_k}}\right) \mathbf{V} = \mathbf{Z}$$



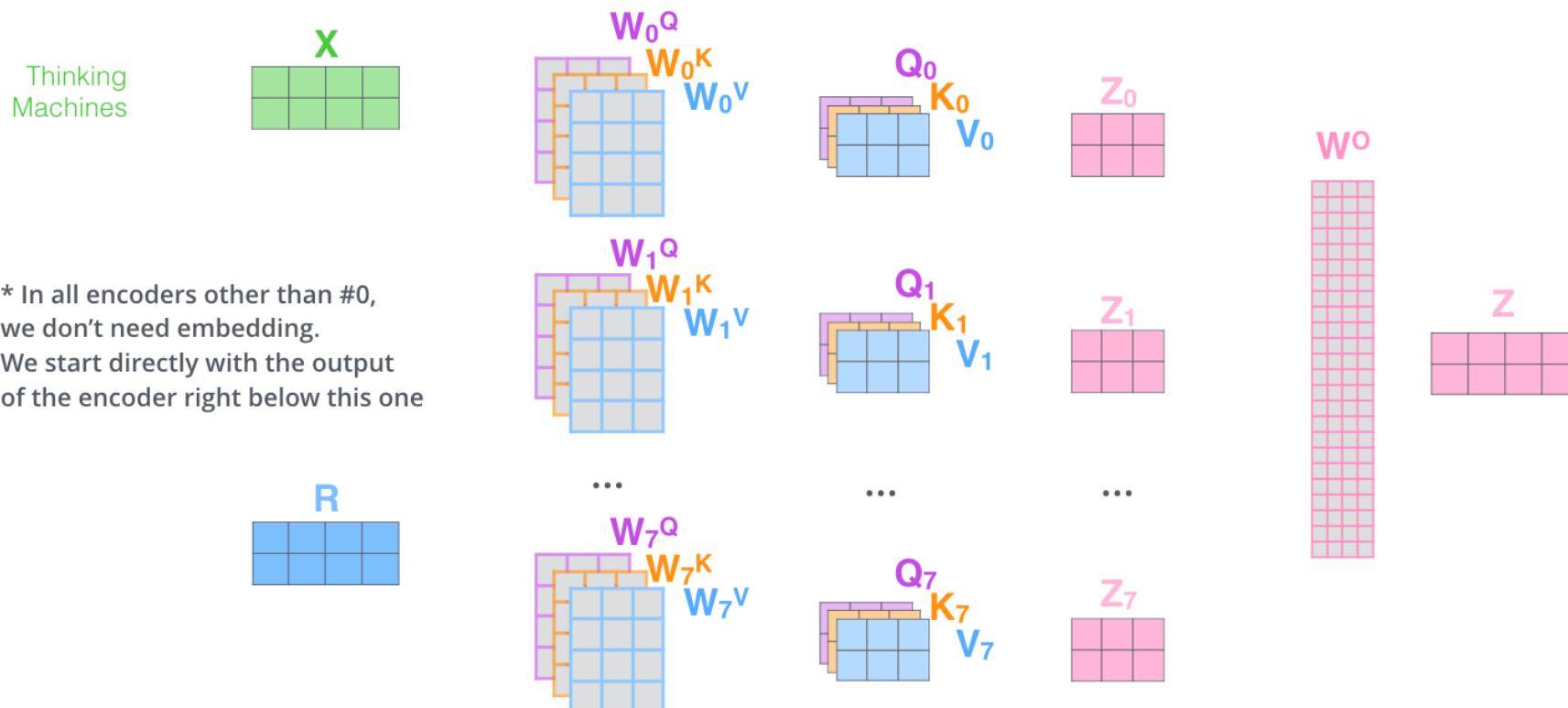
# Multi-head self-attention

With *multi-headed self-attention*, the network learns different semantic meanings of attention (e.g., one for vocabulary, one for grammar...)

Separate Q/K/V weight matrices are maintained for each head.

As in the single-head case, we multiply X by the  $W_Q, W_K, W_V$  matrices to produce Q, K, V.

- 1) This is our input sentence\*      2) We embed each word\*
- 3) Split into 8 heads.  
We multiply  $X$  or  $R$  with weight matrices
- 4) Calculate attention using the resulting Q/K/V matrices
- 5) Concatenate the resulting Z matrices, then multiply with weight matrix  $W^O$  to produce the output of the layer

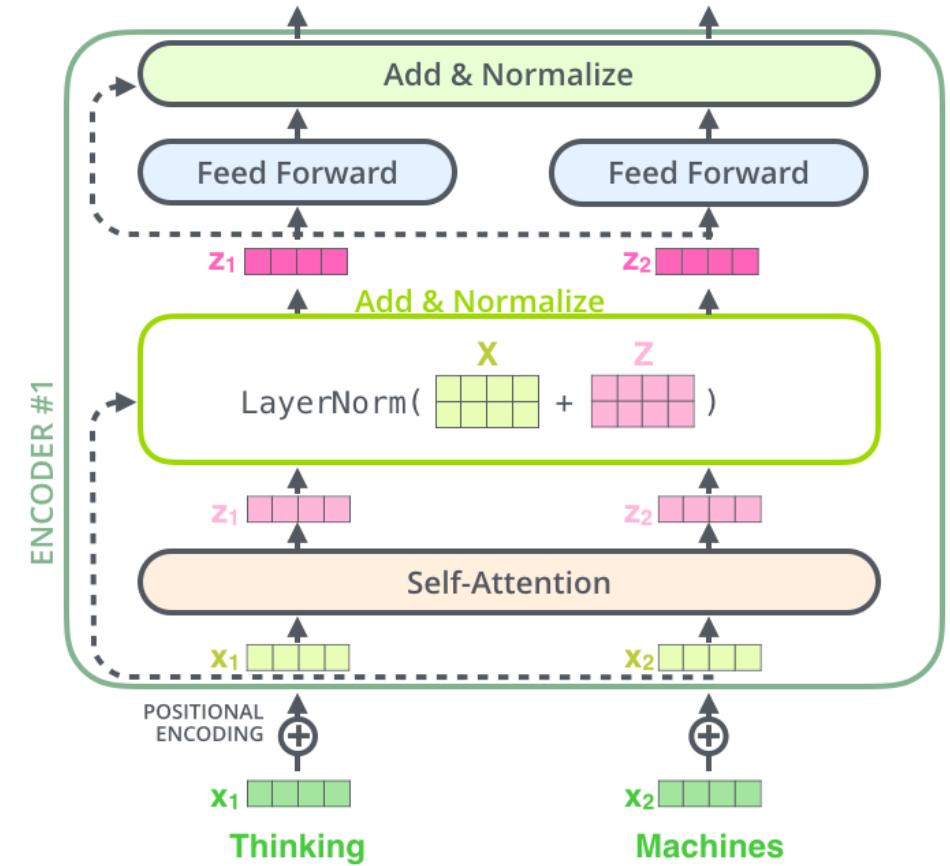


# Normalization and skip connections

Each block in the encoder has an Add&Normalize operation after the self attention.

Here the input X is added with the attended feature Z and the result is normalized with Layer Normalization.

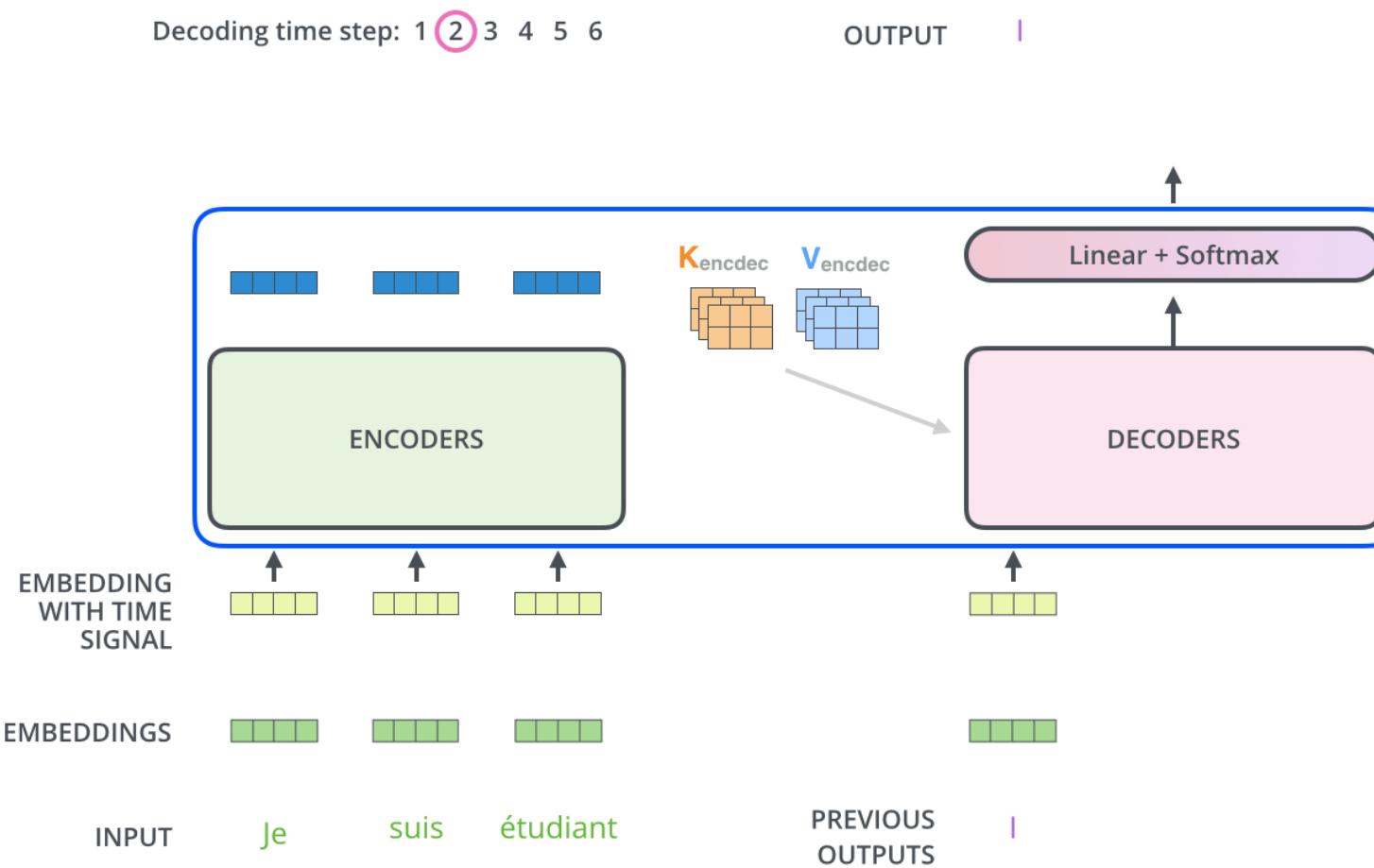
The skip connection that sums X and Z has the benefit of easing training.



# Decoding

Decoding is performed one step at a time.

The decoder receives as input both the output of the encoder and the generated output sequence.



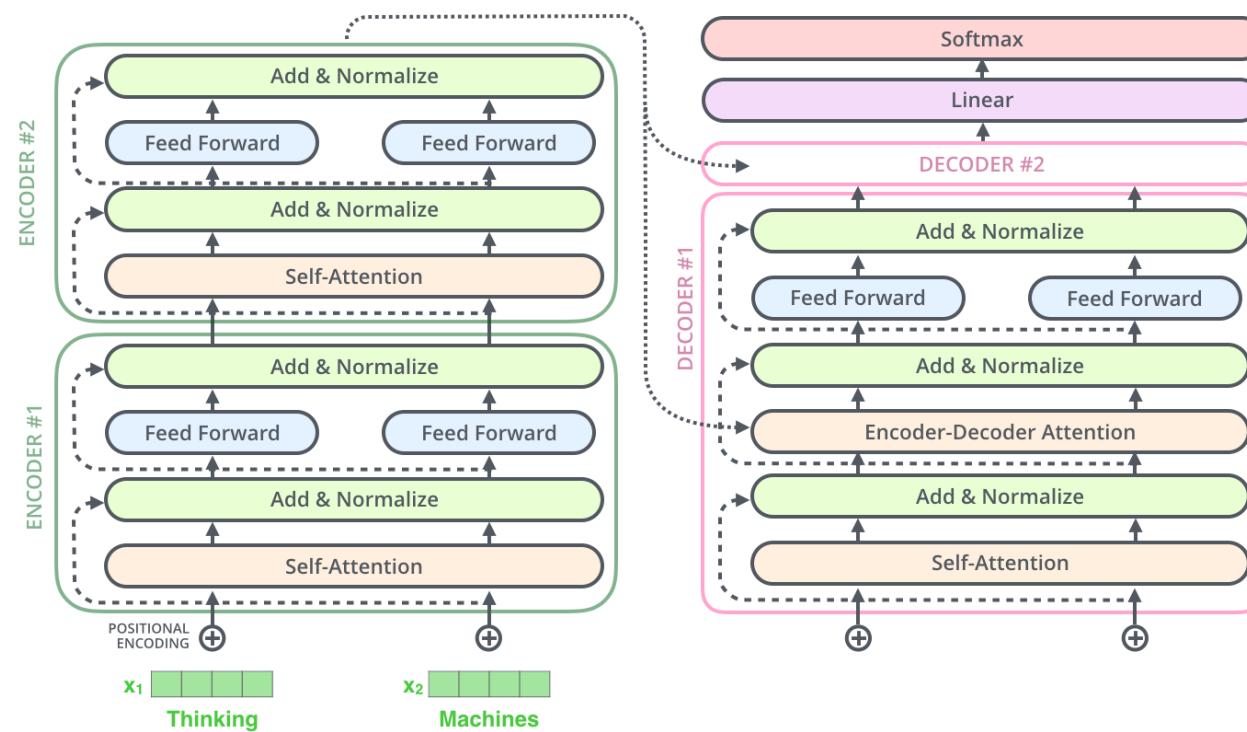
# Decoder

The structure of the decoder is similar to the one of the encoder.

Two attentions are present in each decoder block:

- At first a self-attention between the generated output tokens is performed
- Then an encoder-decoder attention is performed between the embedded input tokens and the embedded output tokens

This makes the model explainable with reference to both input and output tokens.

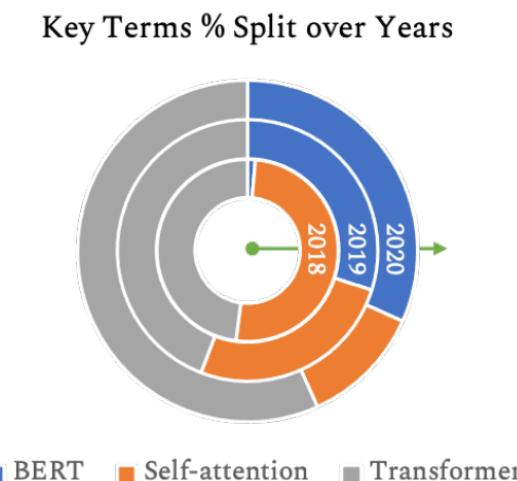
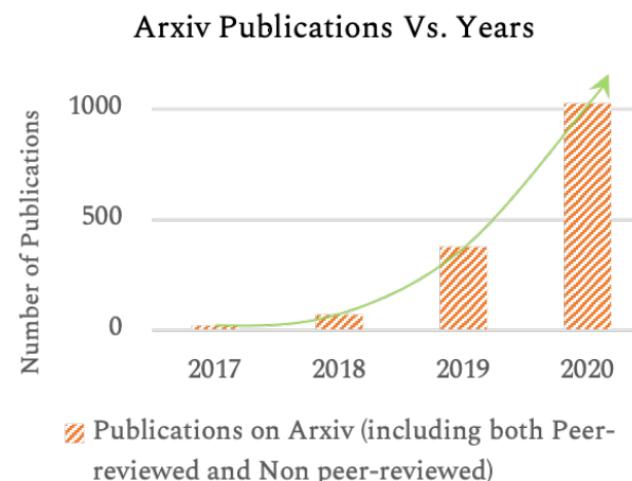
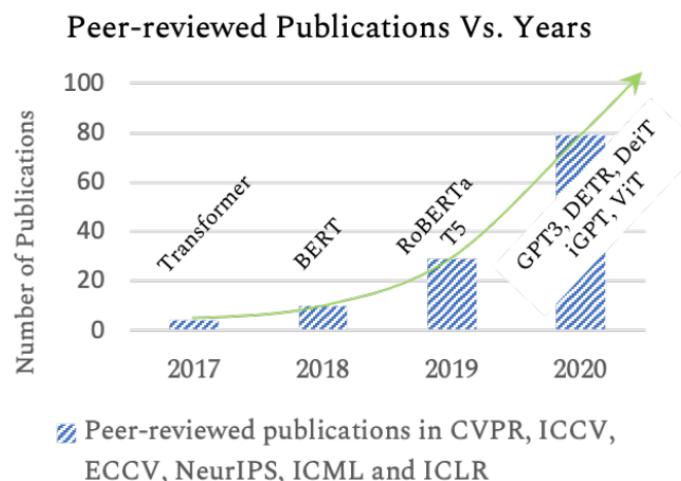
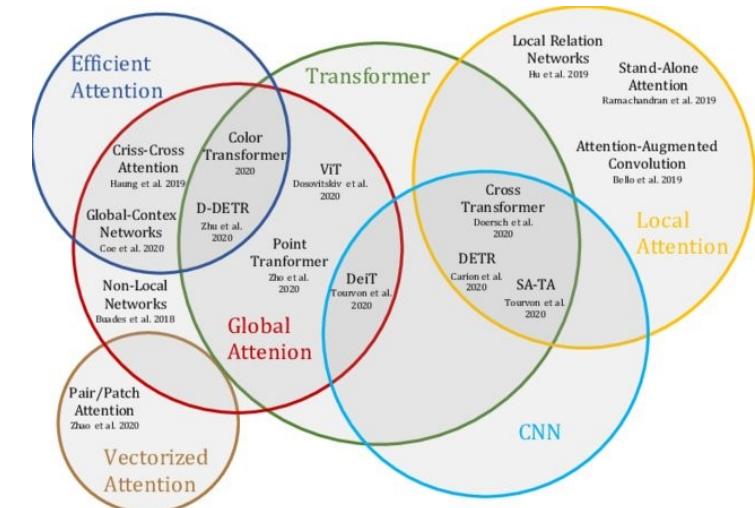


# Transformer Applications

Transformers have been developed to address Natural Language Processing tasks and overcome the limitations of RNNs.

In principle, Transformers can tackle any problem based on sequences/sets.

Rapid application explosion in the past few years covering NLP but also Computer Vision and Machine Learning in general.



# BERT - Bidirectional Encoder Representations from Transformers

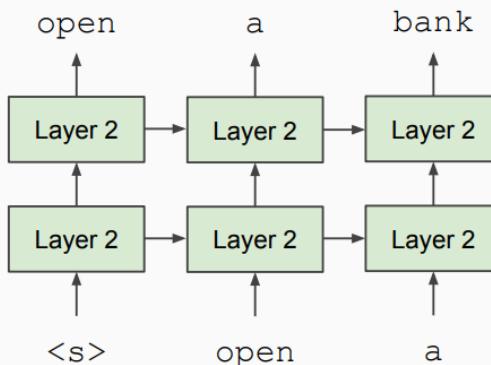
Devlin et al., 2018

Language models usually use only a left or right context, treating the problem in an unidirectional way.

BERT achieved a breakthrough in NLP by performing a pretraining that exploits bidirectional contexts. The resulting model can be then finetuned for a variety of NLP tasks.

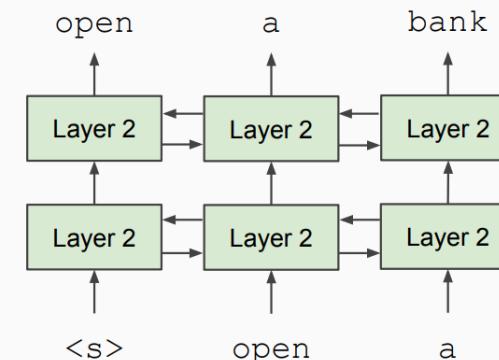
## Unidirectional context

Build representation incrementally



## Bidirectional context

Words can “see themselves”



# BERT - Bidirectional Encoder Representations from Transformers

Devlin et al., 2018

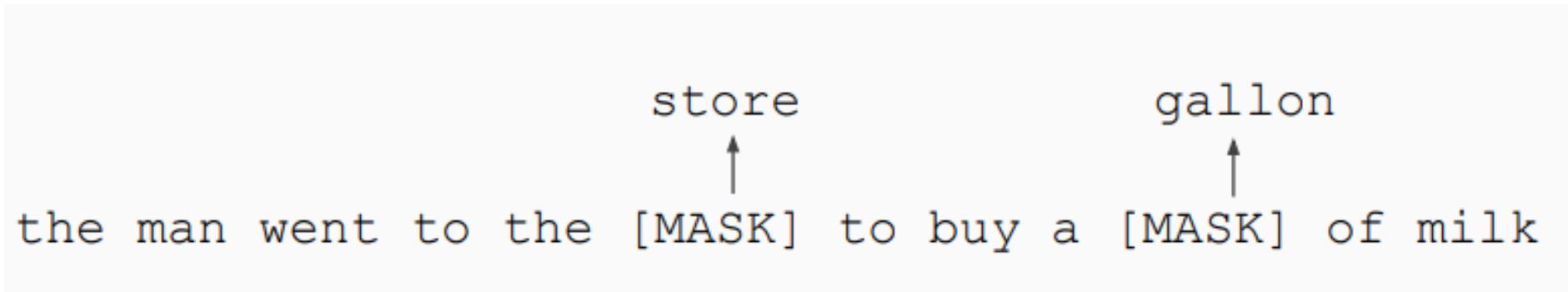
The idea is to mask a percentage of the words in a sentence and then predict the masked words.

BERT uses a masking of 15%.

Too little masking makes the model too expensive to train.

Too much masking does not provide sufficient context.

Training strategy: randomly replace the [MASK] token with a random word or with the original word 10% of the time.  
Makes word distribution more similar between training and testing time.



# BERT - Bidirectional Encoder Representations from Transformers

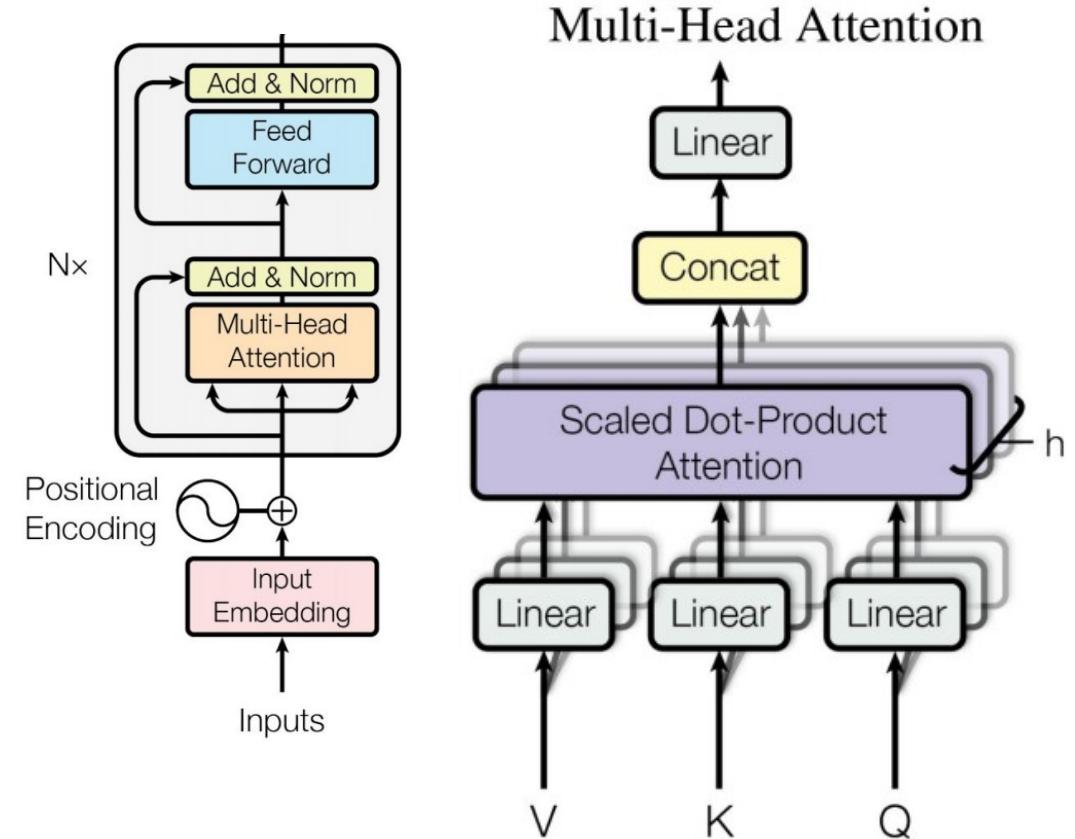
Devlin et al., 2018

BERT uses a Transformer encoder.

A multi-headed self attention provides an effective way to model context.

Feed-forward layers capture the non-linearity of the data and layer norm and residuals help the convergence of the model.

Positional embeddings allow the model to learn relative positioning.



# Positional embeddings

In order to add position information (order of the sequence)

$$PE_{(pos,2i)} = \sin(pos/10000^{2i/d_{\text{model}}})$$

$$PE_{(pos,2i+1)} = \cos(pos/10000^{2i/d_{\text{model}}})$$

Each dimension of the positional encoding corresponds to a sinusoid.

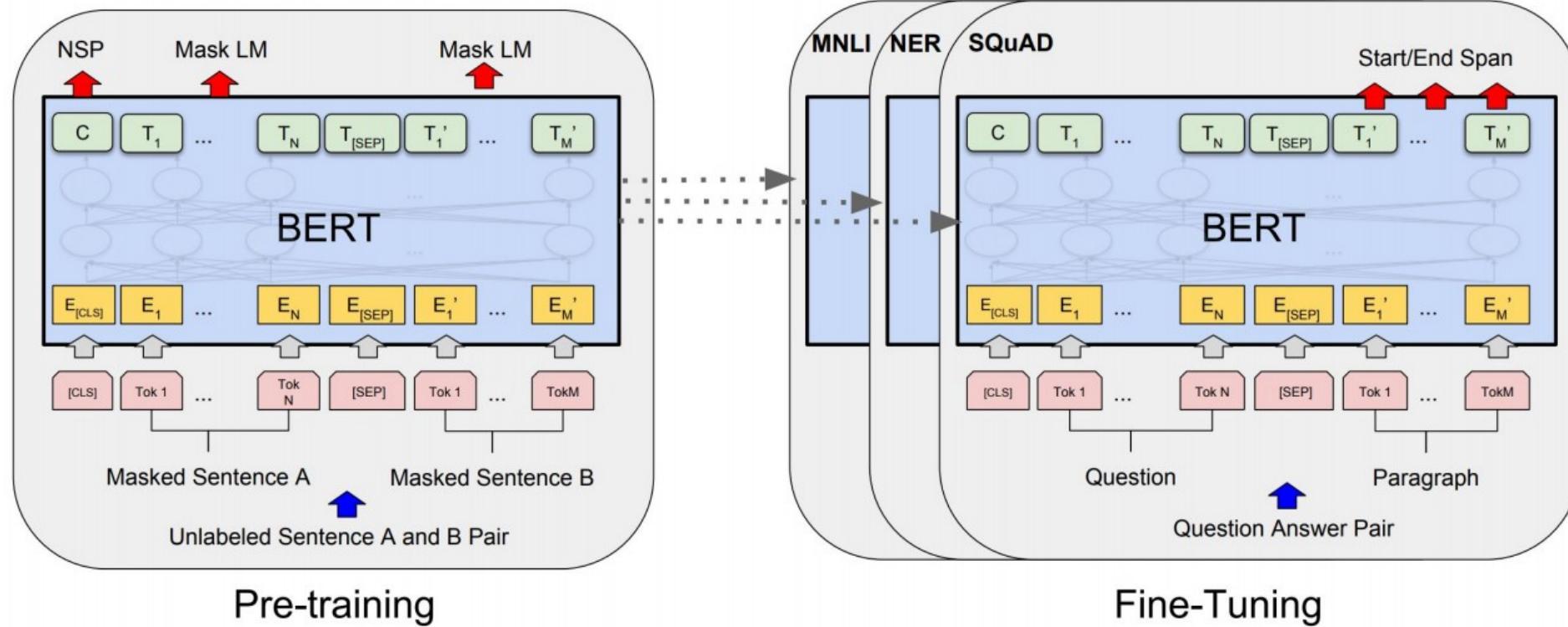
For any fixed offset  $k$ ,  $PE_{pos+k}$  can be represented as a linear transformation of  $PE_{pos}$ . This would allow the model to easily learn to attend by relative positions.

# BERT - Bidirectional Encoder Representations from Transformers

Devlin et al., 2018

BERT acts as an effective pretraining for several NLP tasks.

The model can be simply finetuned on specific tasks by changing the classification head.

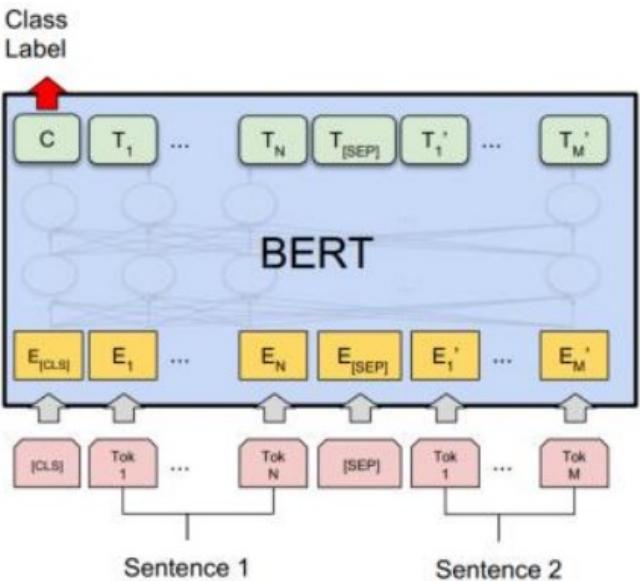


# Fine-tuning with BERT

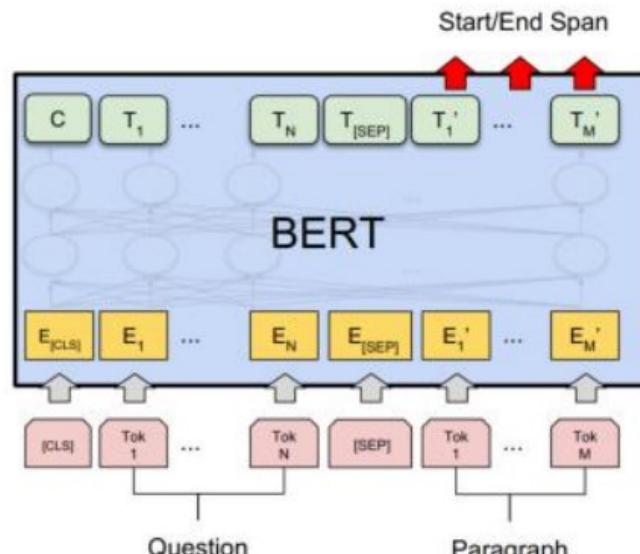
- Context vector  $C$ : Take the final hidden state corresponding to the first token in the input: [CLS].
- Transform to a probability distribution of the class labels:

$$P = \text{softmax}(CW^T)$$

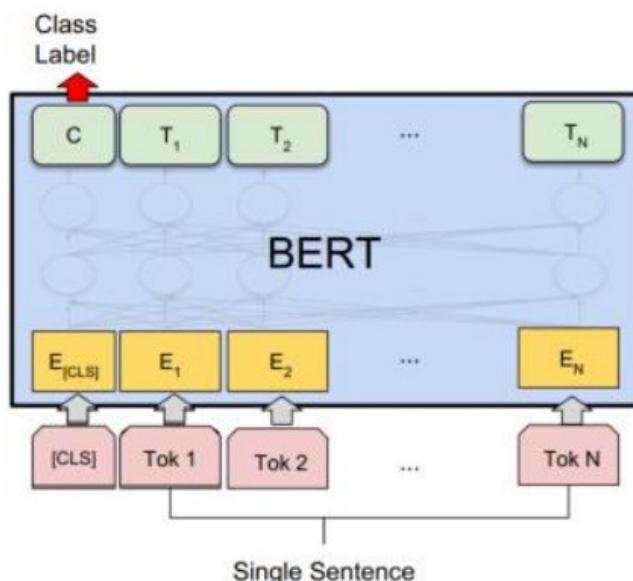
- **Batch size:** 16, 32
- **Learning rate (Adam):** 5e-5, 3e-5, 2e-5
- **Number of epochs:** 3, 4



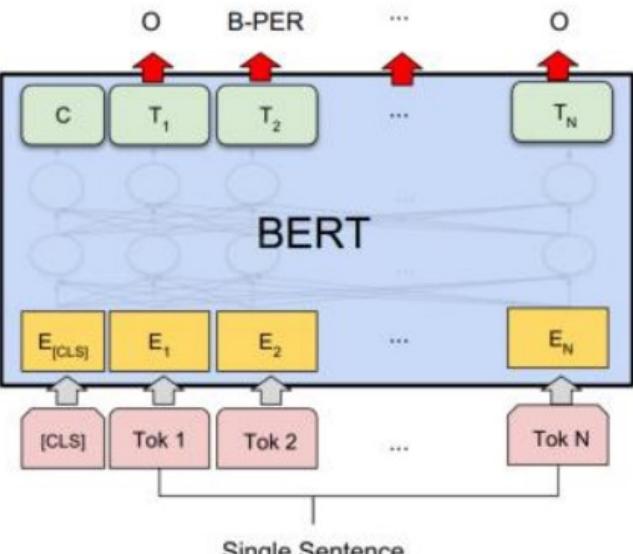
(a) Sentence Pair Classification Tasks:  
MNLI, QQP, QNLI, STS-B, MRPC,  
RTE, SWAG



(c) Question Answering Tasks:  
SQuAD v1.1



(b) Single Sentence Classification Tasks:  
SST-2, CoLA

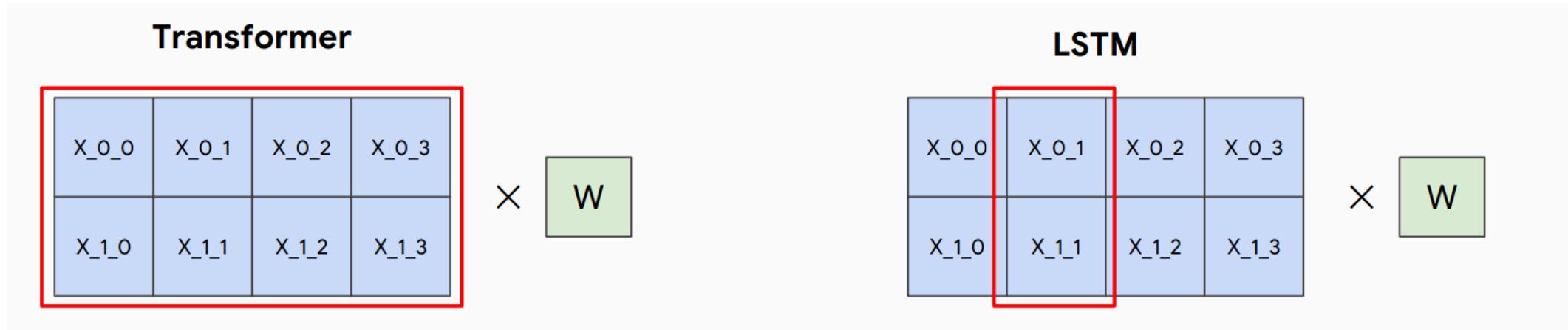


(d) Single Sentence Tagging Tasks:  
CoNLL-2003 NER

# Transformers vs LSTM

In Transformers, self attention is not influenced by distance between tokens (no long-term forgetting).

Transformers are more efficient since matrix multiplication is performed between weights and whole sentences instead of tokens. Made possible by high parallelization in modern GPUs.



# Transformers beyond NLP

Transformers have been exploited also to process different kinds of data than text.

Suitable for any input/output that can be represented as a set of tokens.

Since the inputs are processed in parallel, there is no need for sequential data.

What about images? Images can be seen as a set of pixels/patches.

Image patches in computer vision can be encoded separately into a set of tokens just as words in NLP.

# Vision Transformer (ViT)

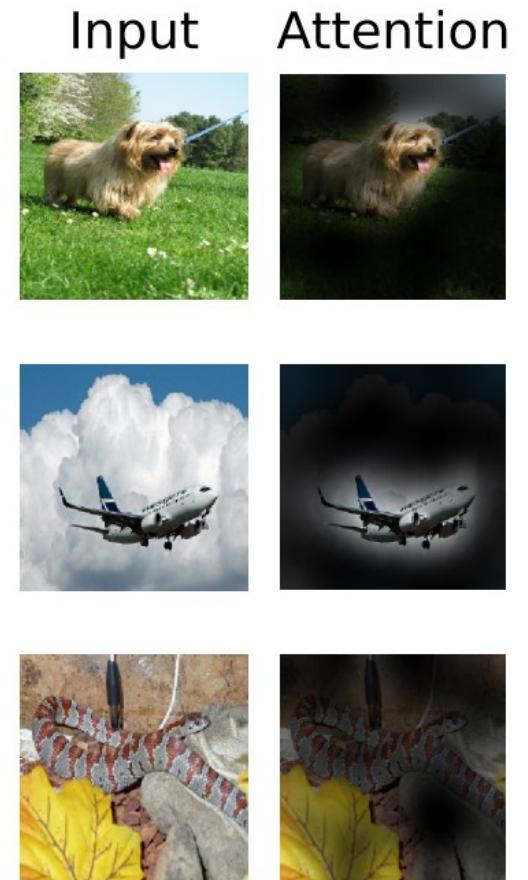
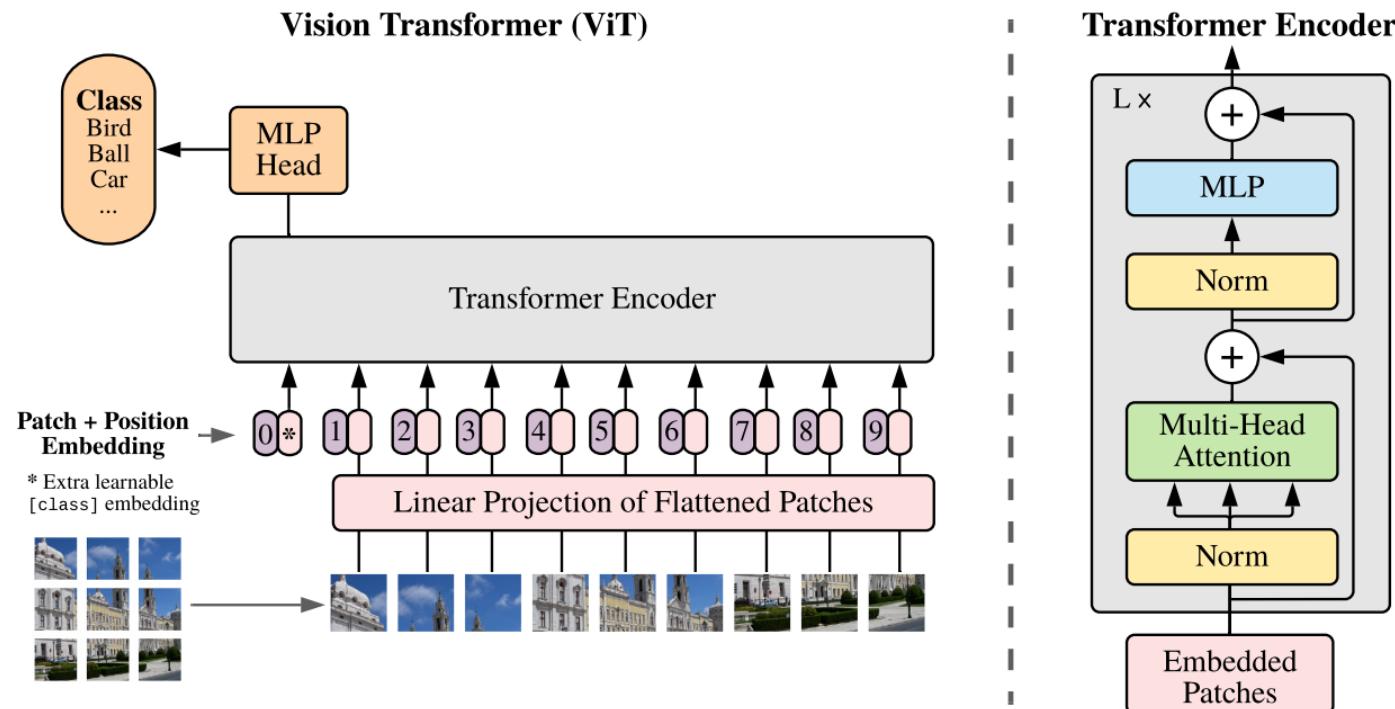
A. Dosovitskiy et al. 2020

The image is split into fixed-size patches, linearly embedded and concatenated with position embeddings.

The sequence of “tokens” is fed to a standard Transformer encoder.

An MLP head is used for classifying images.

Results are explainable thanks to attention.

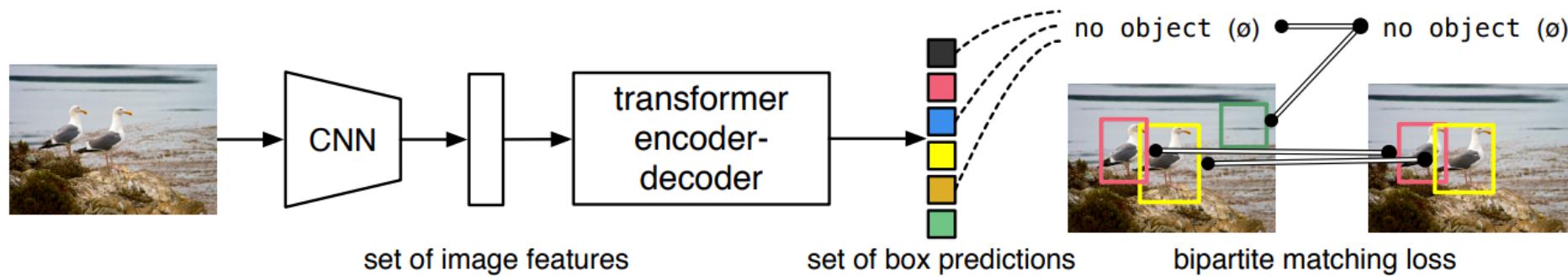


# DETR – Object Detection with Transformers

N. Carion et al, 2020

DETR addresses the problem of object detection as bipartite matching problem, uniquely assigning predictions with ground truth boxes.

CNN feature maps are considered as a set of vectors describing each pixel and are fed to a transformer. The output is a set of detections.



# DETR – Object Detection with Transformers

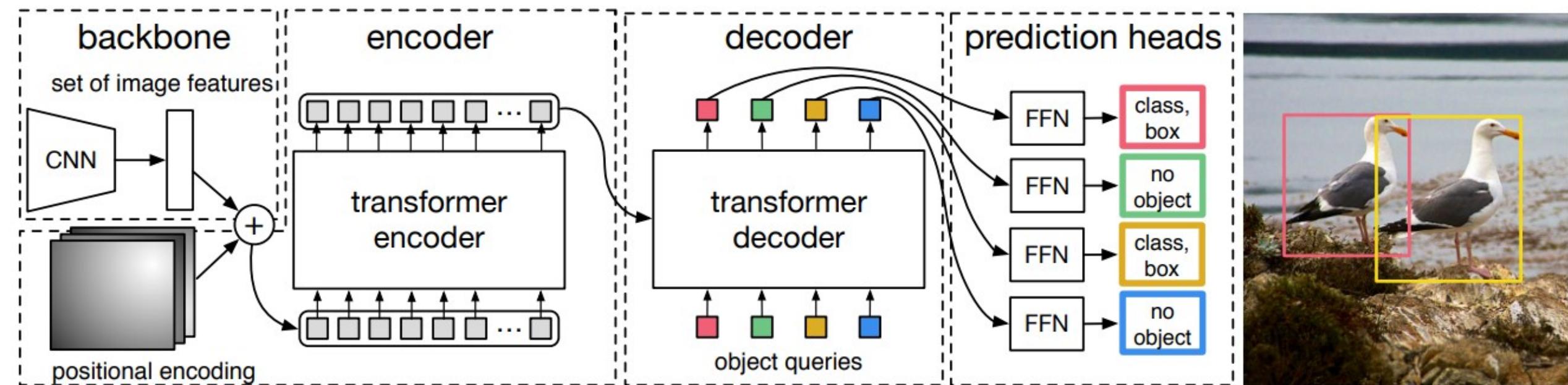
N. Carion et al, 2020

DETR uses a conventional CNN backbone to learn a 2D representation of an input image.

The model flattens it and supplements it with a positional encoding before passing it into a transformer encoder.

A transformer decoder then takes as input a small fixed number of learned positional embeddings, called queries, and additionally attends to the encoder output.

Each output embedding of the decoder is passed to a shared feed forward network (FFN) predicting either a detection (class and bounding box) or a “no object” class.



# DETR – Object Detection with Transformers

N. Carion et al, 2020

Interpretability with reference to input pixels thanks to self-attention.

self-attention(430, 600)



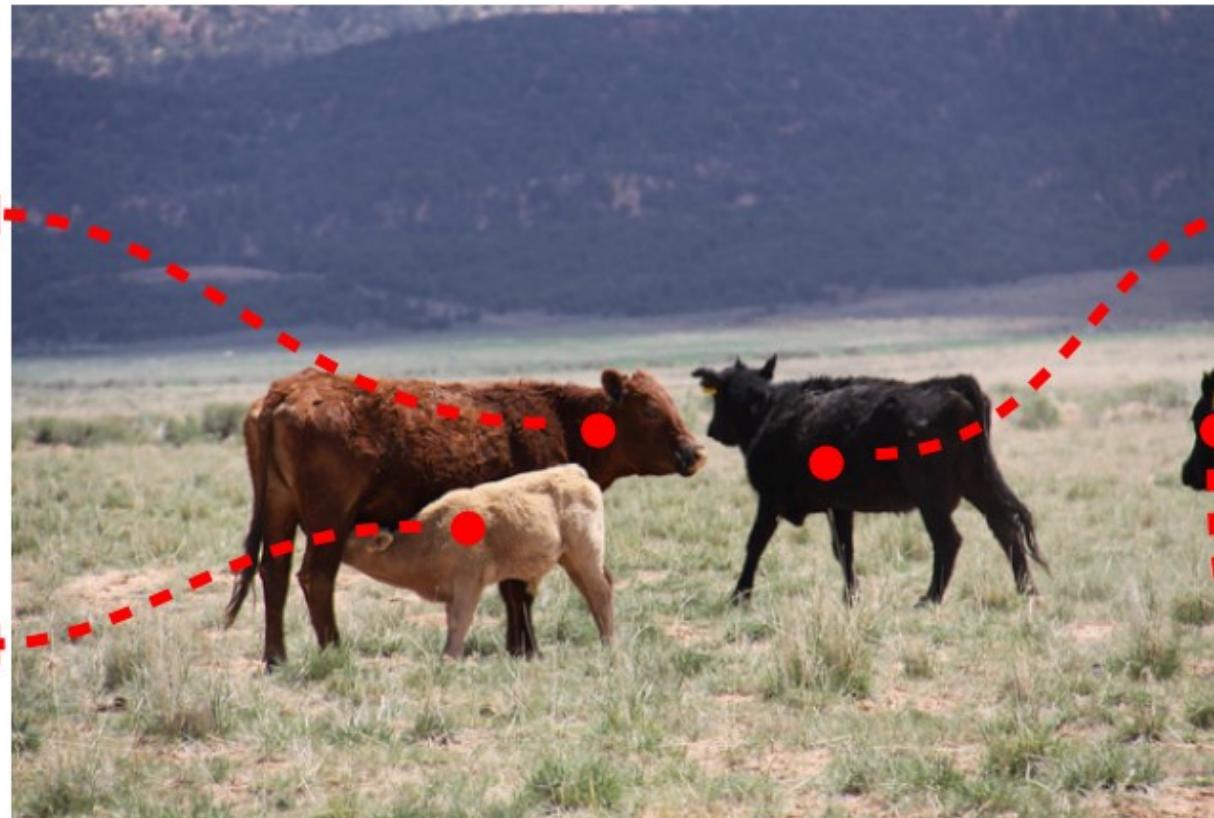
self-attention(520, 450)



self-attention(450, 830)



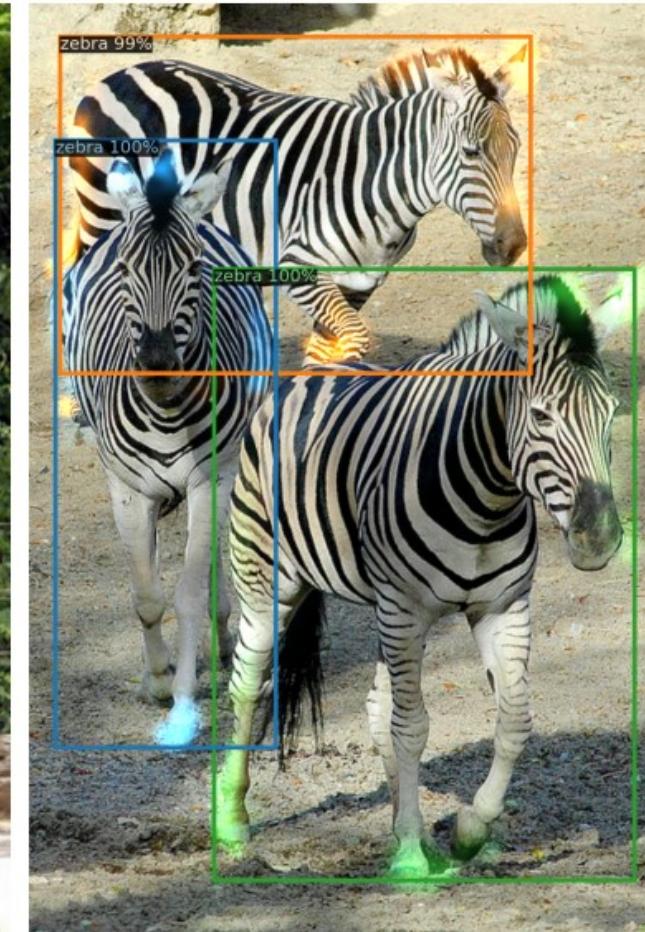
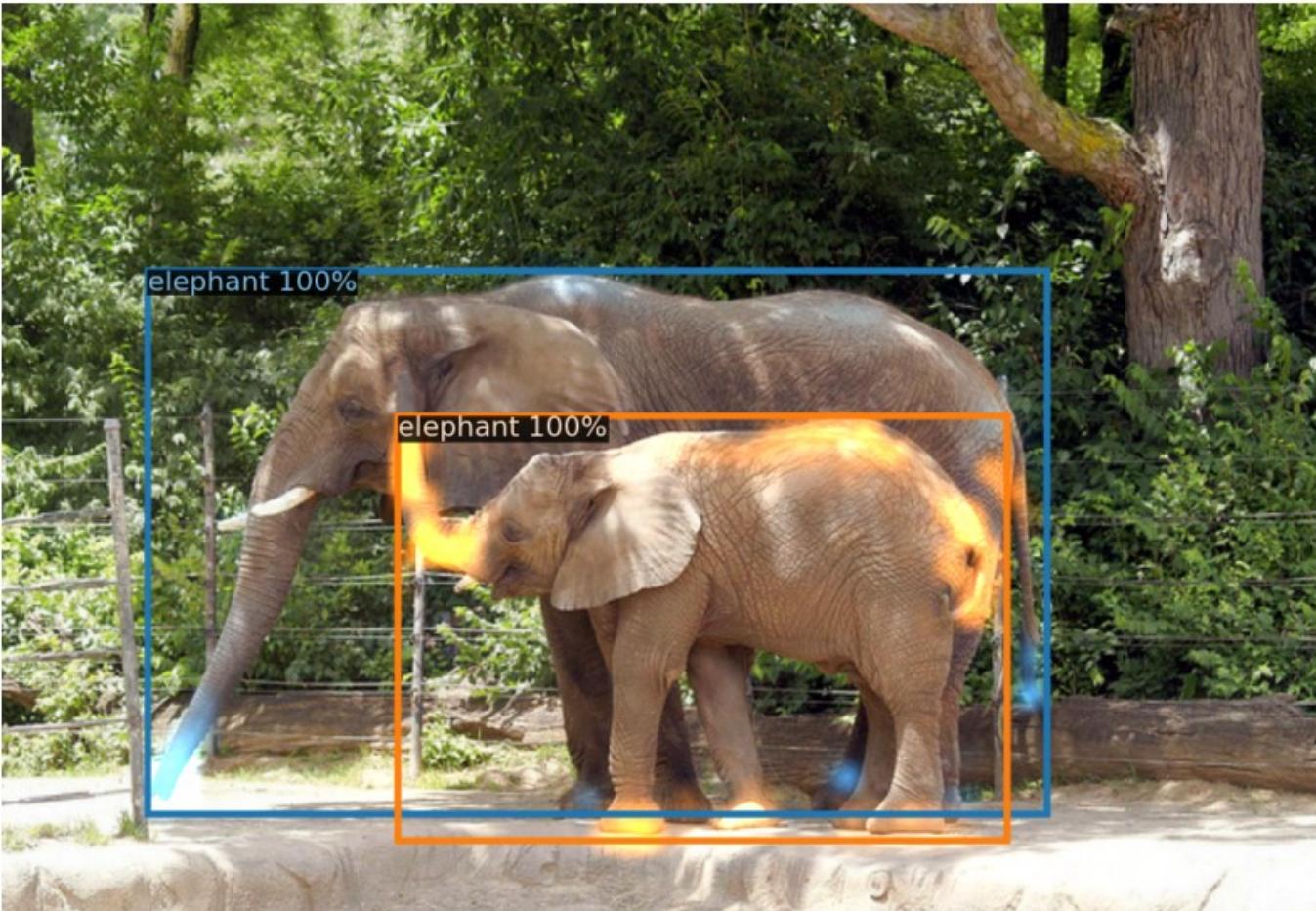
self-attention(440, 1200)



# DETR – Object Detection with Transformers

N. Carion et al, 2020

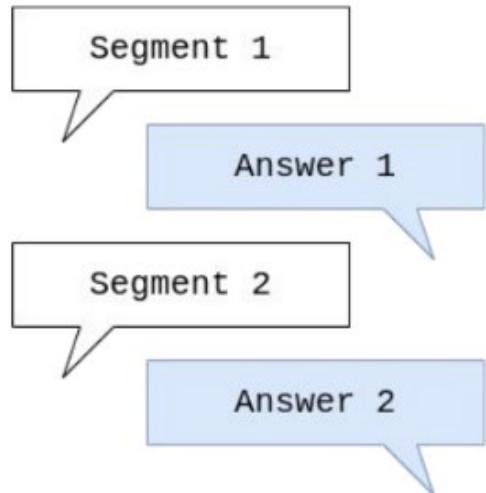
Interpretability wrt output boxes thanks to input-query attention.



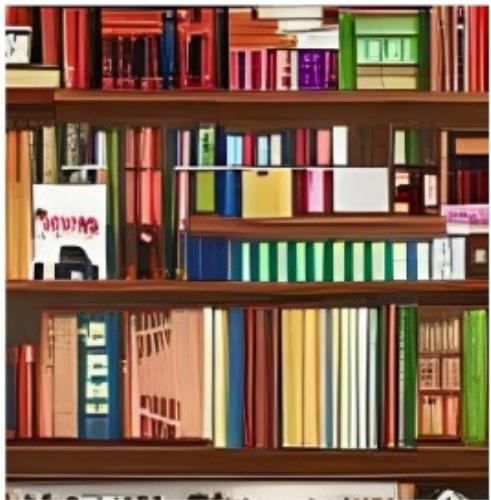
# Limitations of transformers

- Parallelization makes transformers computationally efficient, however it restricts a full exploitation of the sequential nature of the input.
- Temporal information must be manually added to the input using positional encodings.
- Hidden representations only accesses the past representations of lower layers, even though higher-level representations of the past have already been computed as an autoregressive model.
- At inference time, Transformers generate one token at a time, so they could access these representations for better performance. Such information is not exploited at training time due to parallelization.
- Lack of recursive computation: the number of possible transformations on the input is bounded by the model depth.

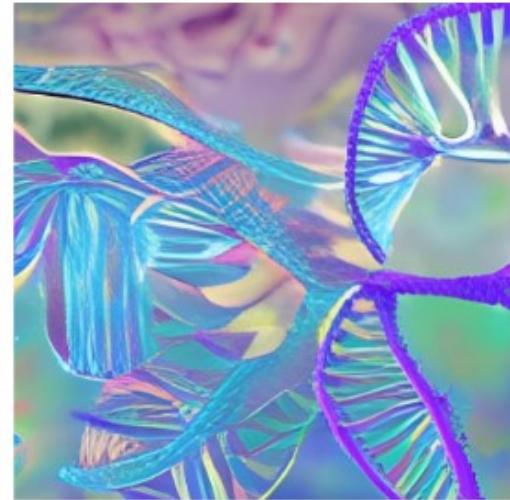
# Sequential tasks with longer sequences



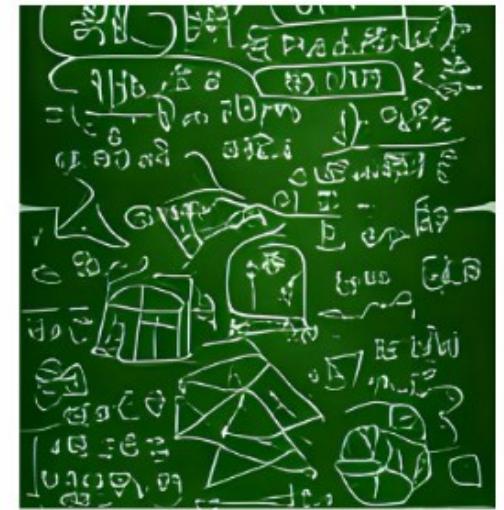
Dialogs



Long texts



Bio-sequences  
related



Step-by-step  
solutions

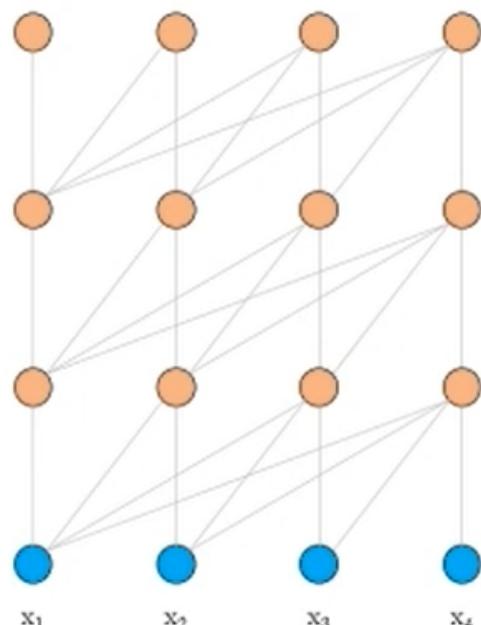
# Language Modeling

Given a corpus of  $T$  tokens  $\mathbf{x} = [x_1, x_2, \dots, x_t, \dots, x_T]$ , model

$$\begin{aligned} P_{\theta}(\mathbf{x}) &= \prod_{t=1}^T P_{\theta}(x_t \mid \mathbf{x}_{<t}) \\ &= \prod_{t=1}^T P(x_t \mid f_{\theta}(\mathbf{x}_{<t})) \\ &= \prod_{t=1}^T \frac{\exp(f_{\theta}(\mathbf{x}_{<t})^\top e_{x_t})}{\sum_{x \in \mathcal{X}} \exp(f_{\theta}(\mathbf{x}_{<t})^\top e_x)} \end{aligned}$$

# Standard Transformers

**Step 1:** use causal attention masks



- Remove any backward connection

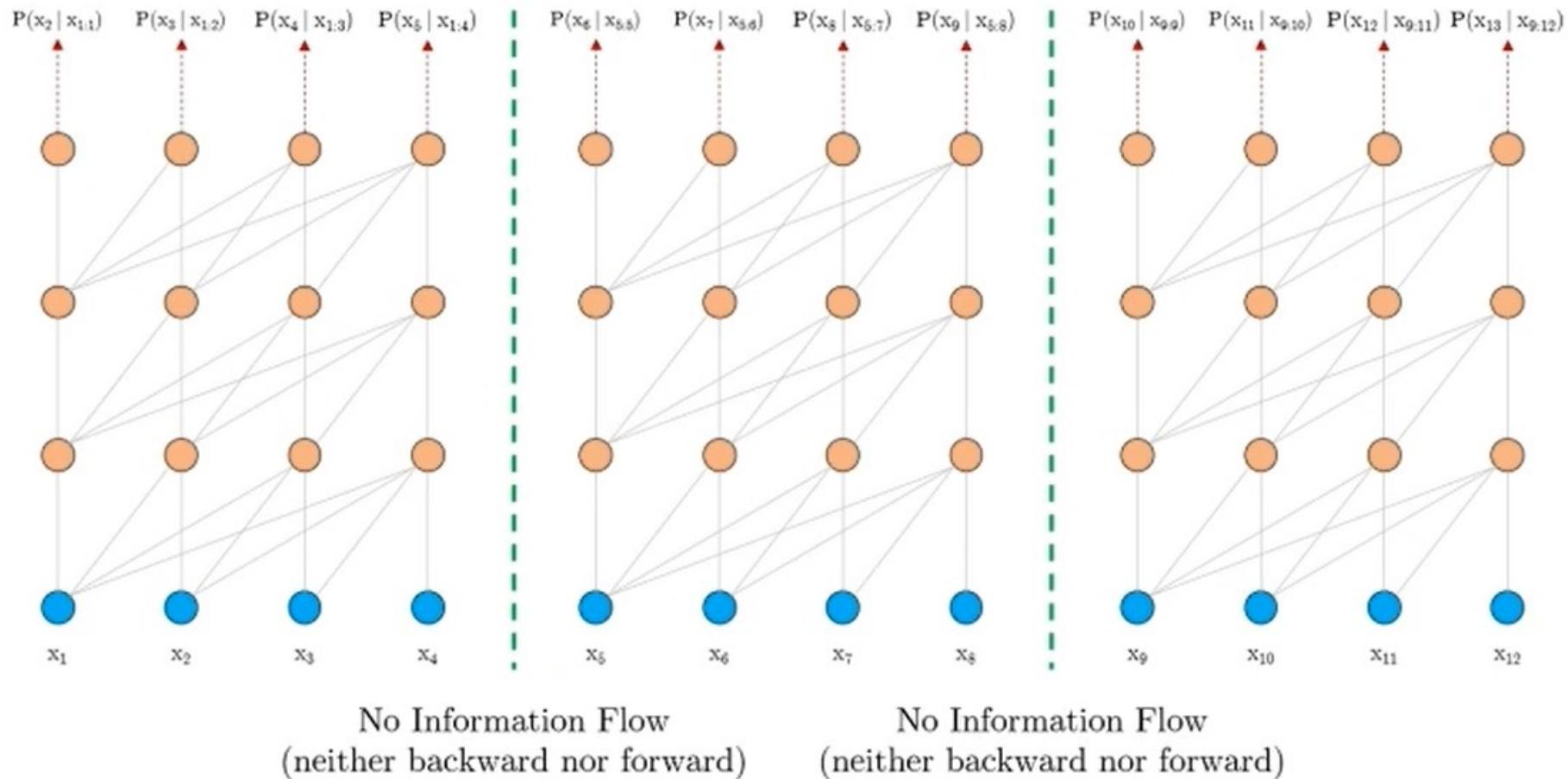
**Step 2:** break the corpus into segments of equal lengths

$$\mathbf{x} = \left[ \underbrace{(x_1, x_2, \dots, x_L), \dots, (x_{(\tau-1)L+1}, x_{(\tau-1)L+2}, \dots, x_{\tau T}), \dots}_{\text{segment 1}} \right]$$
$$= \left[ \underbrace{(x_{1,1}, x_{1,2}, \dots, x_{1,L}), \dots, (x_{\tau,1}, x_{\tau,2}, \dots, x_{\tau,L}), \dots}_{\mathbf{s}_1} \right]$$

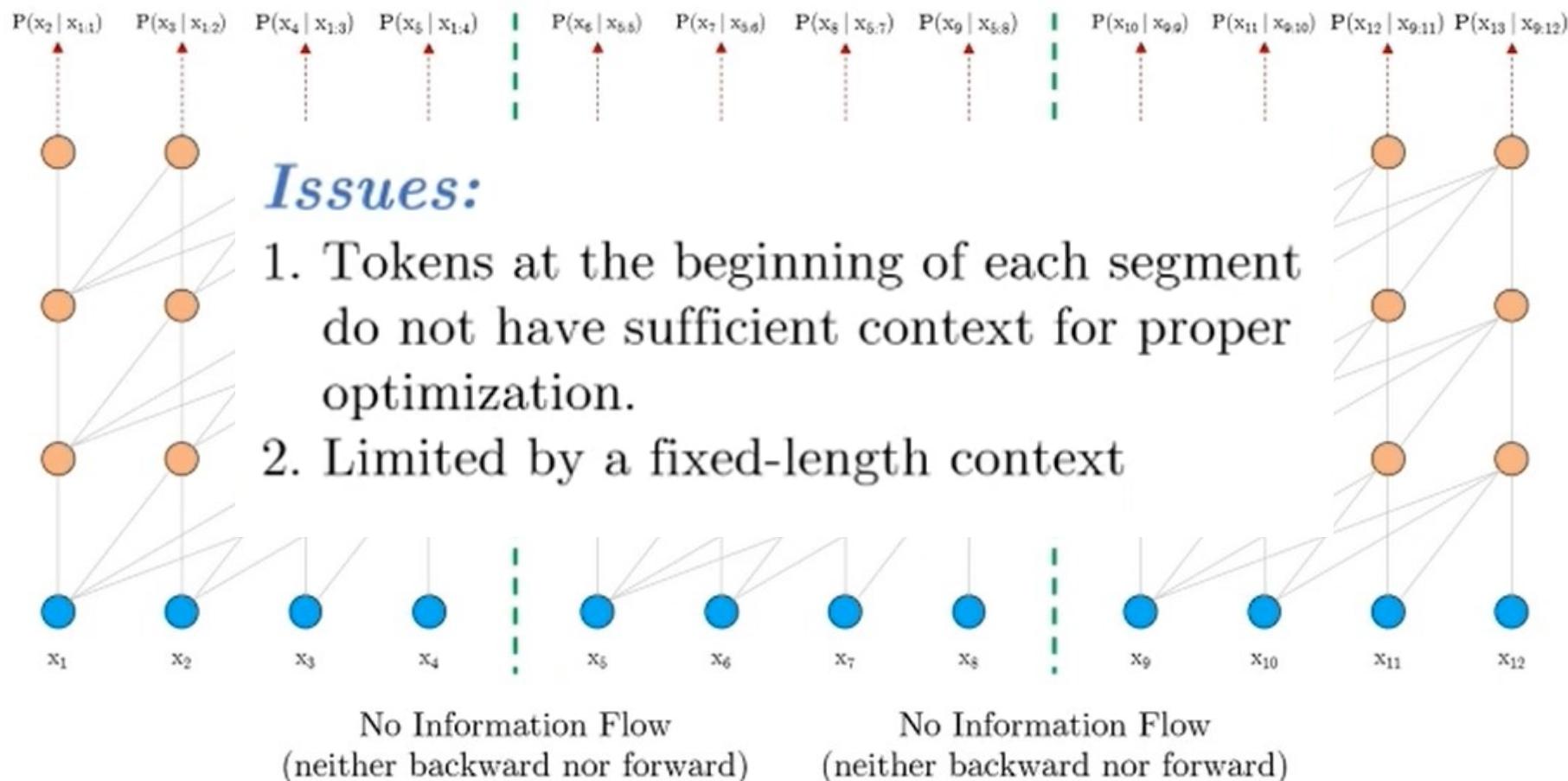
**Step 3:** Model each segment **independently** (limited memory)

$$P(\mathbf{x}) = \prod_{\tau} P(\mathbf{s}_{\tau} \mid \mathbf{s}_{<\tau}) \approx \prod_{\tau} P(\mathbf{s}_{\tau}) \quad (\text{independence assumption})$$
$$= \prod_{\tau} \prod_{i=1}^L P(x_{\tau,i} \mid \mathbf{x}_{\tau,< i}) = \prod_{\tau} \prod_{I=1}^L P(x_{\tau,i} \mid f(\mathbf{x}_{\tau,< I}))$$

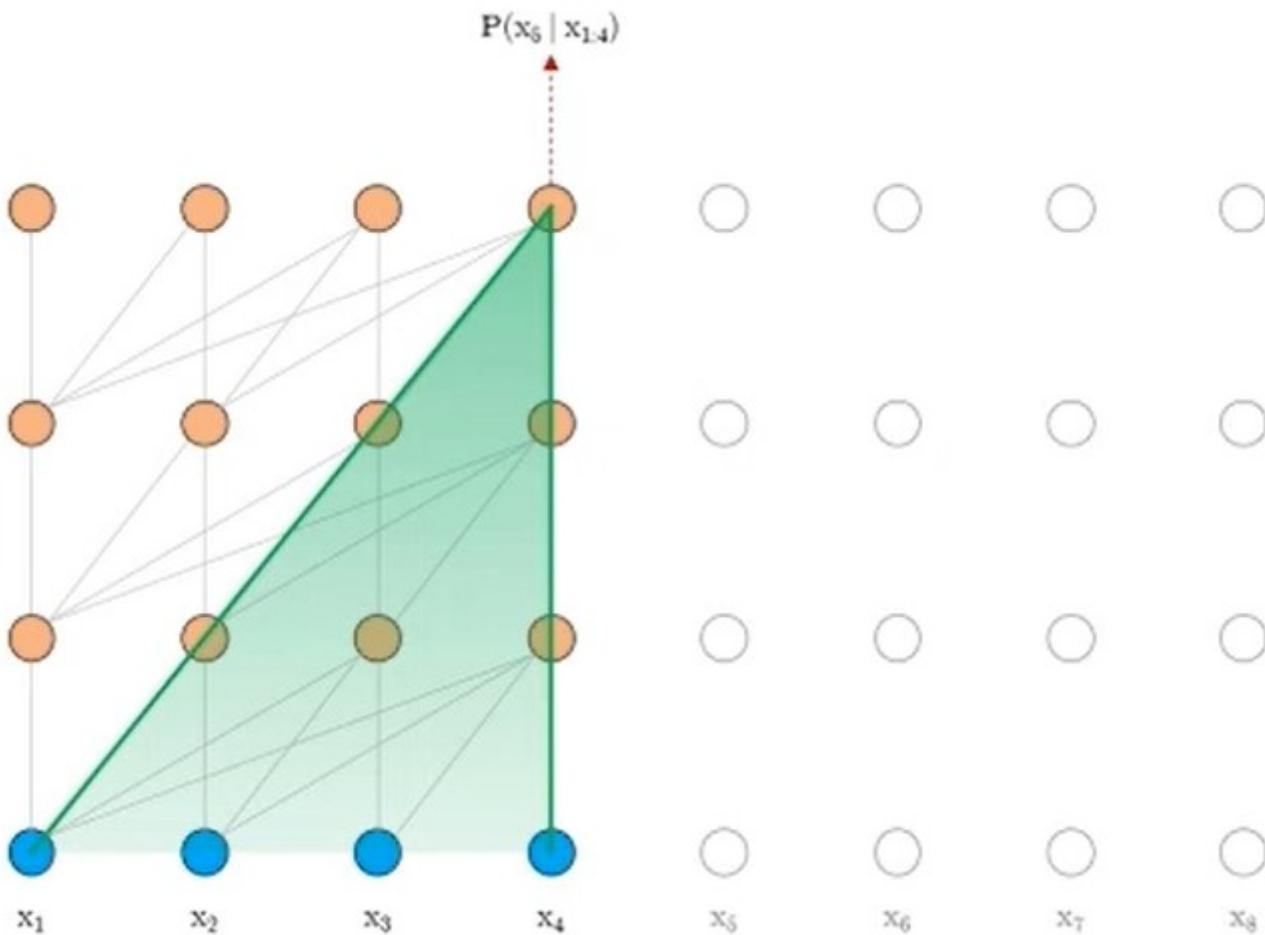
# Standard Transformers



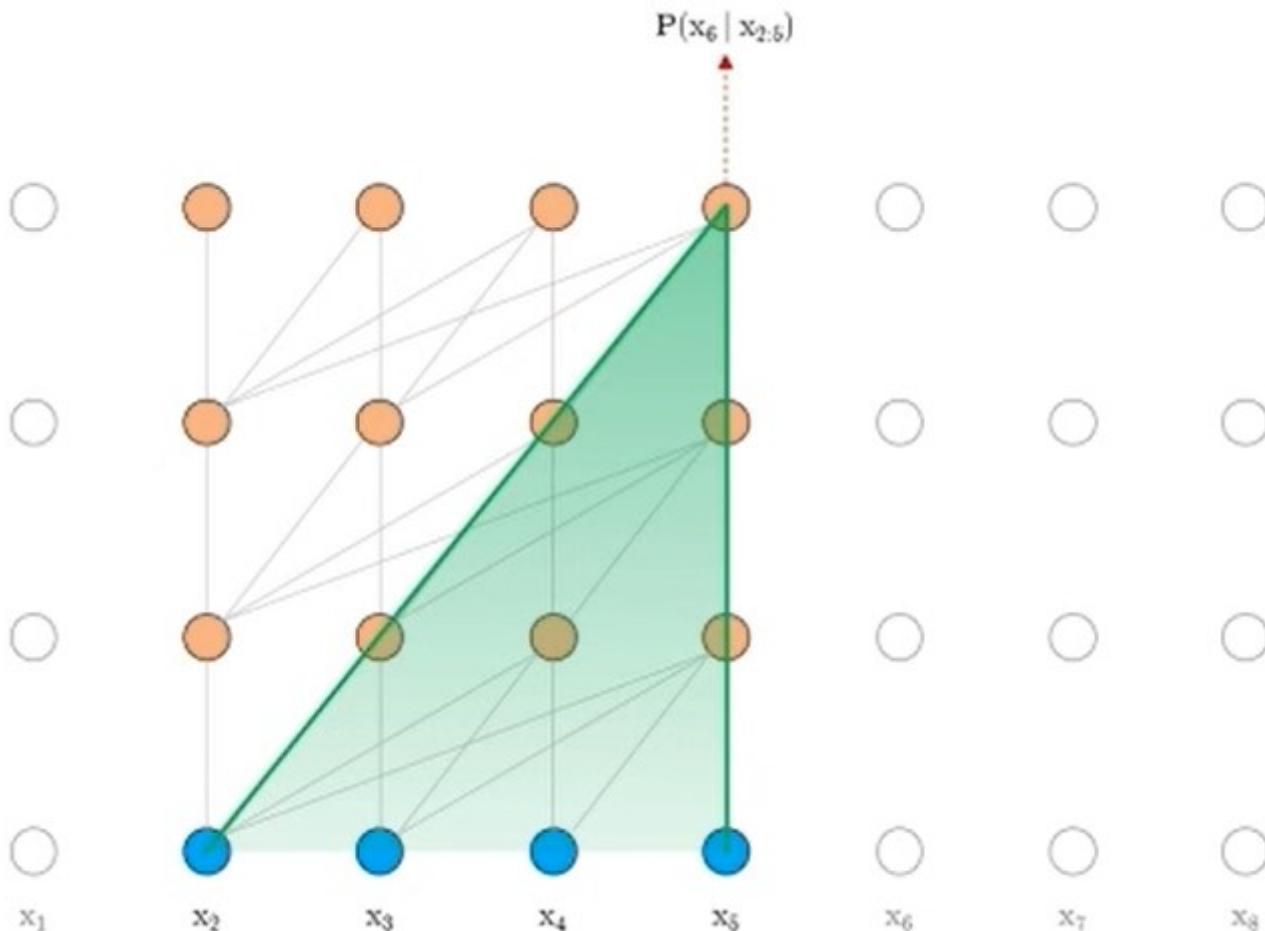
# Standard Transformers



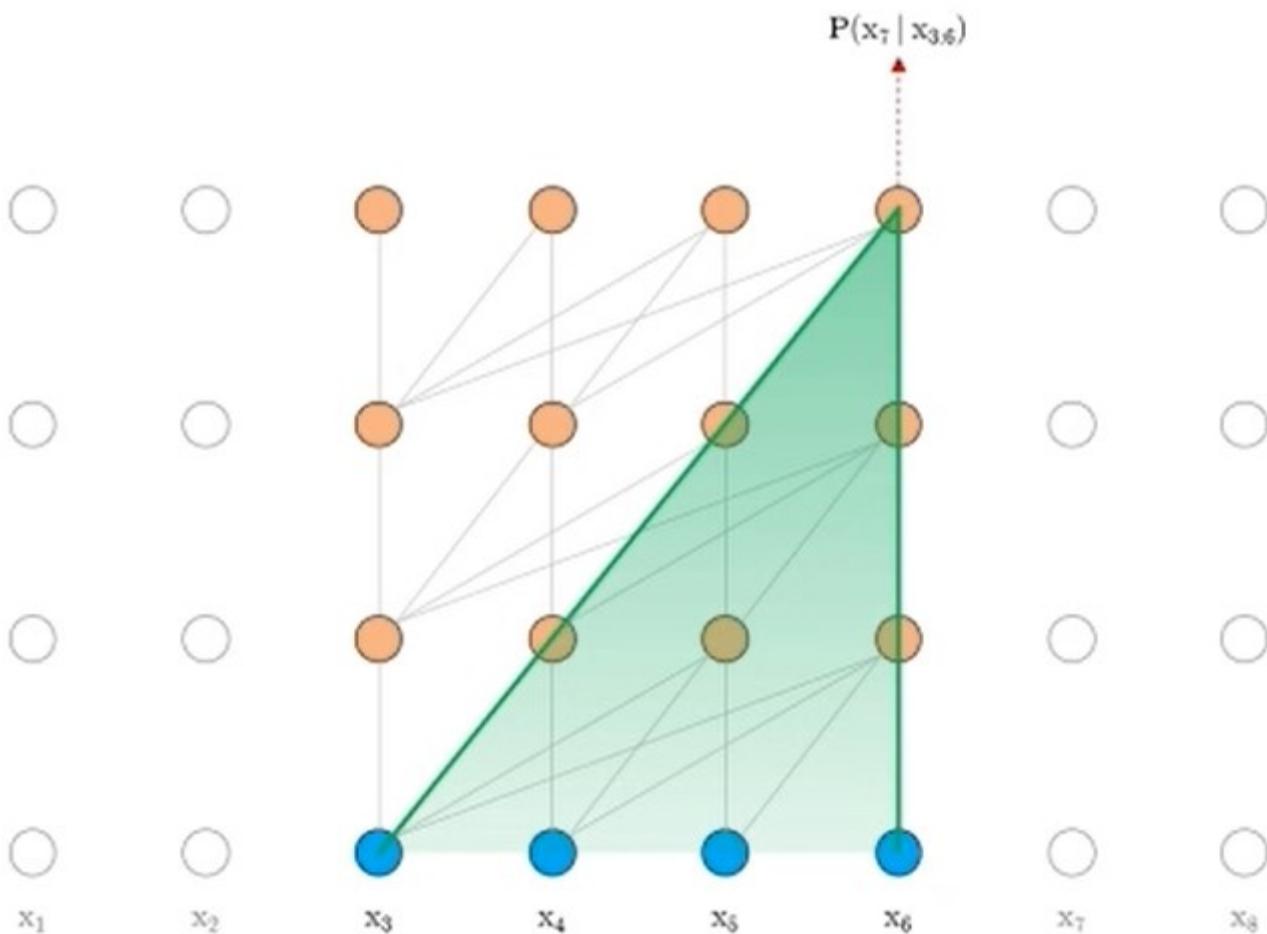
# Standard Transformers: evaluation



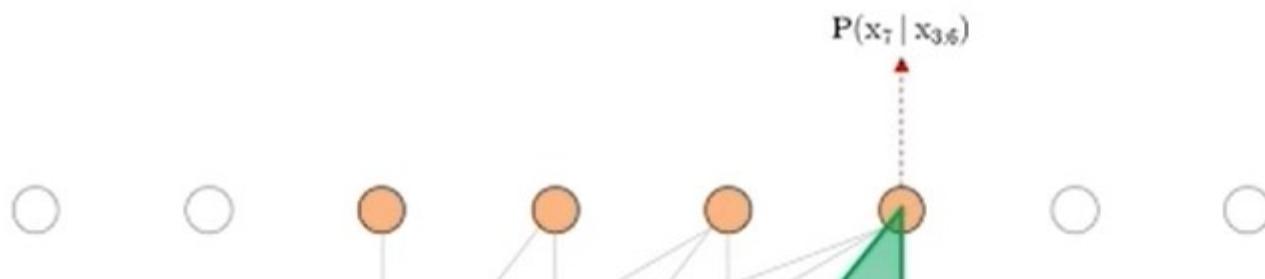
# Standard Transformers: evaluation



# Standard Transformers: evaluation

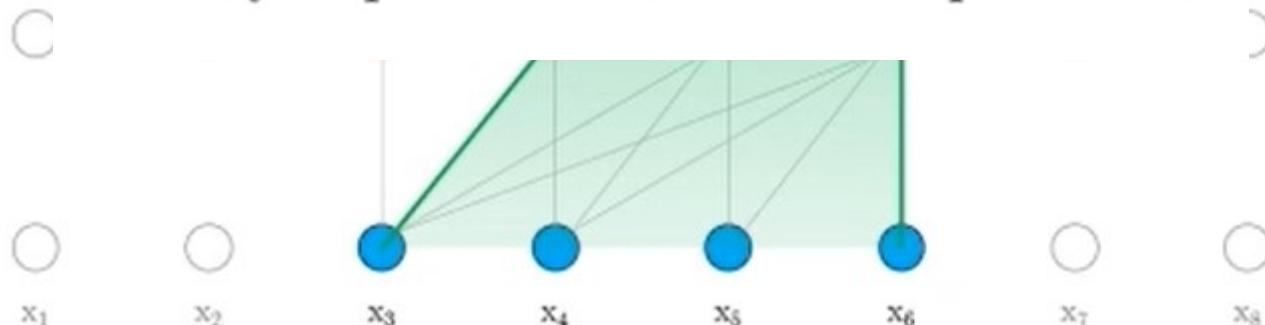


# Standard Transformers: evaluation



## *Issues:*

- 1. Longest context limited by segment length.
- 2. Very expensive due to recomputation.



# Transformer XL

## Key Ideas

### (1) Segment-Level Recurrence

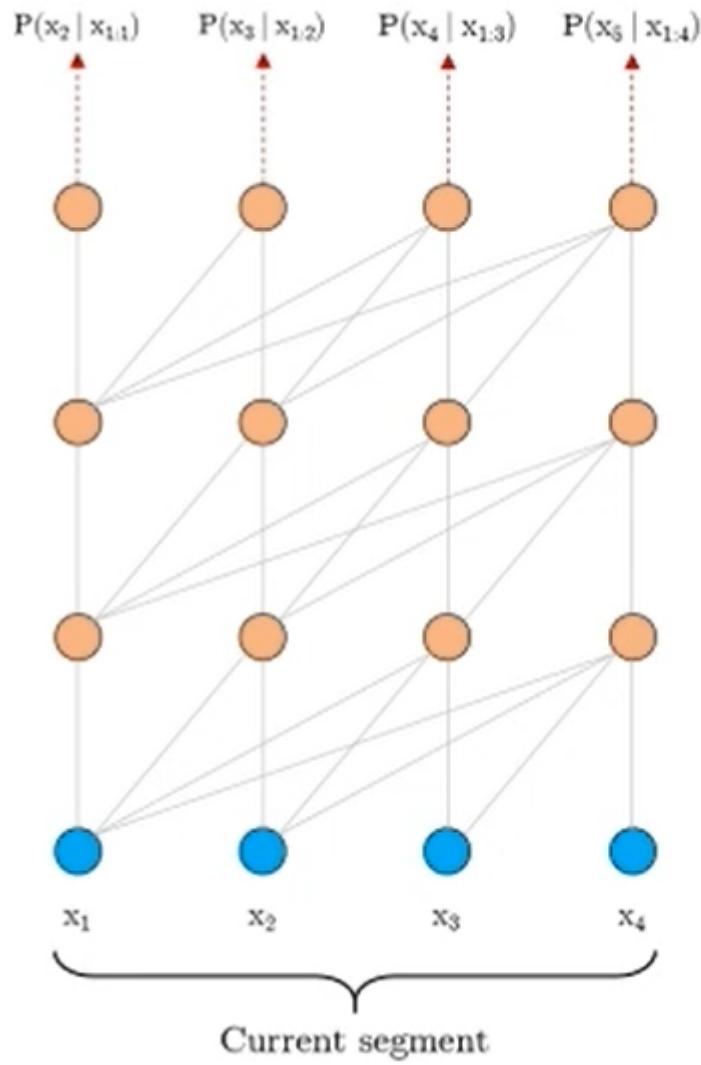
- *Cache* and *reuse* hidden states from last batch
- Analogous to Truncated BPTT for RNN: pass the last hidden state to the next segment as the initial hidden

### (2) Keep temporal information **coherent**

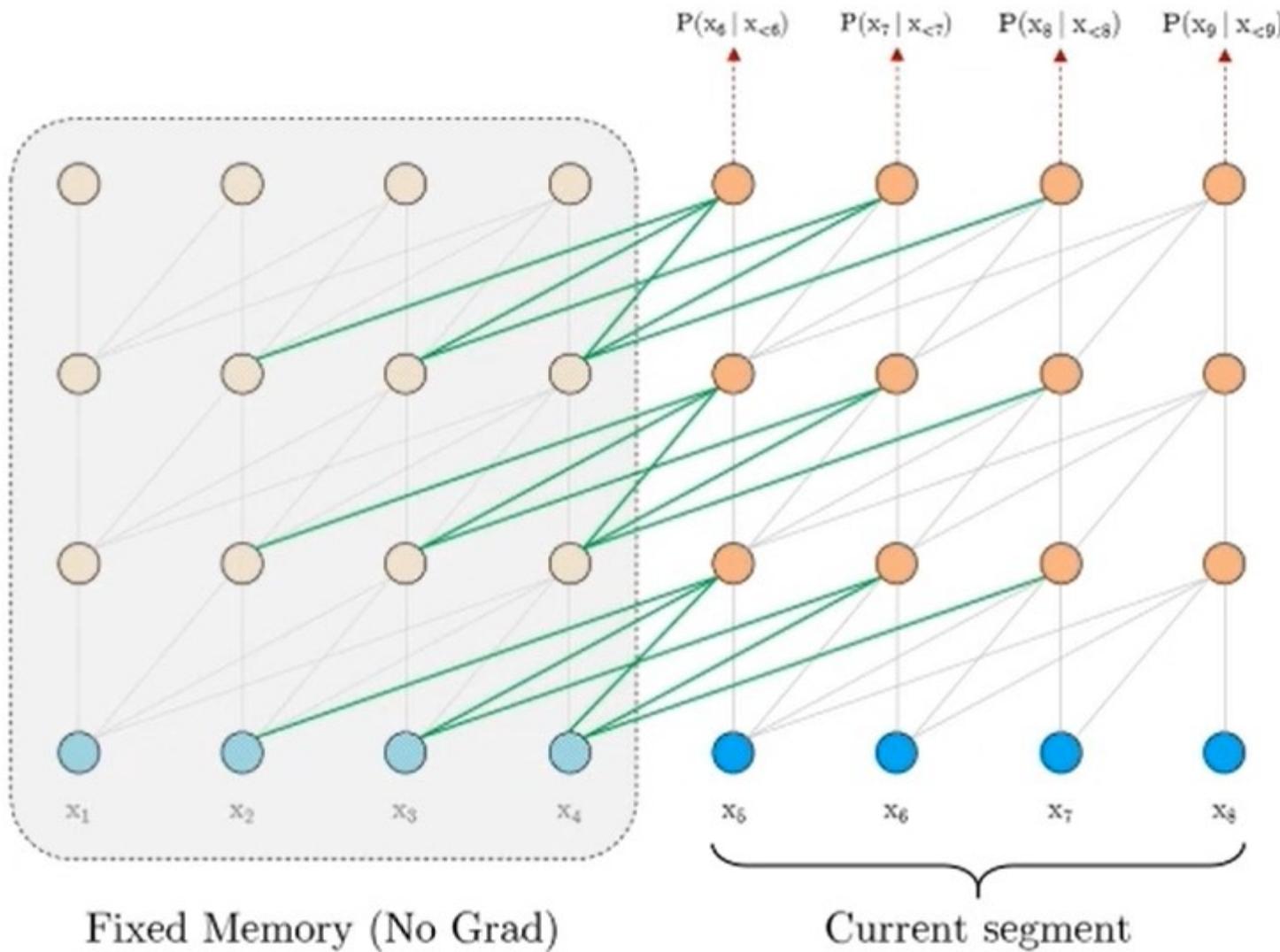


Transformer-XL = Transformer Extra Long

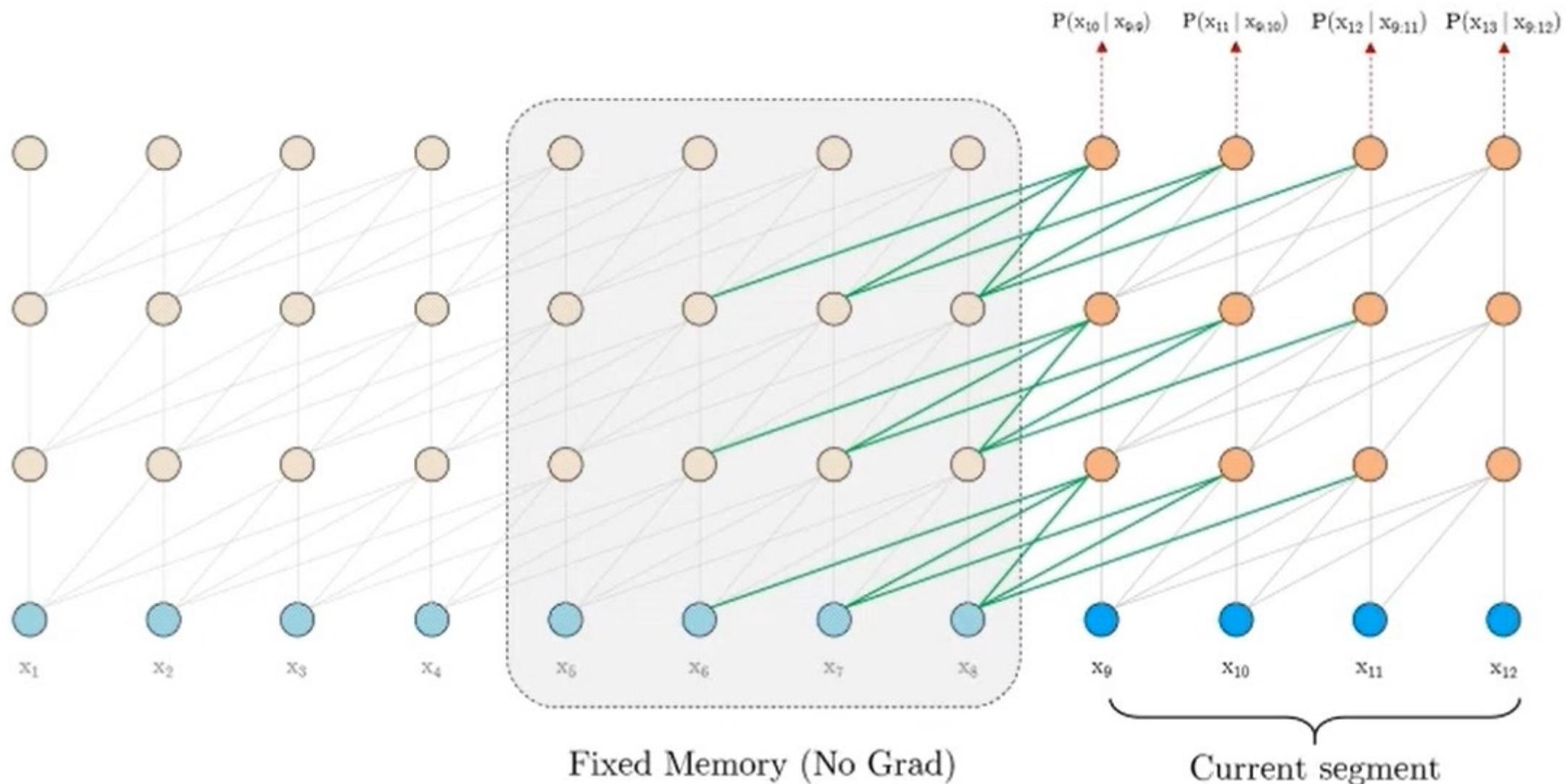
# Transformer XL training



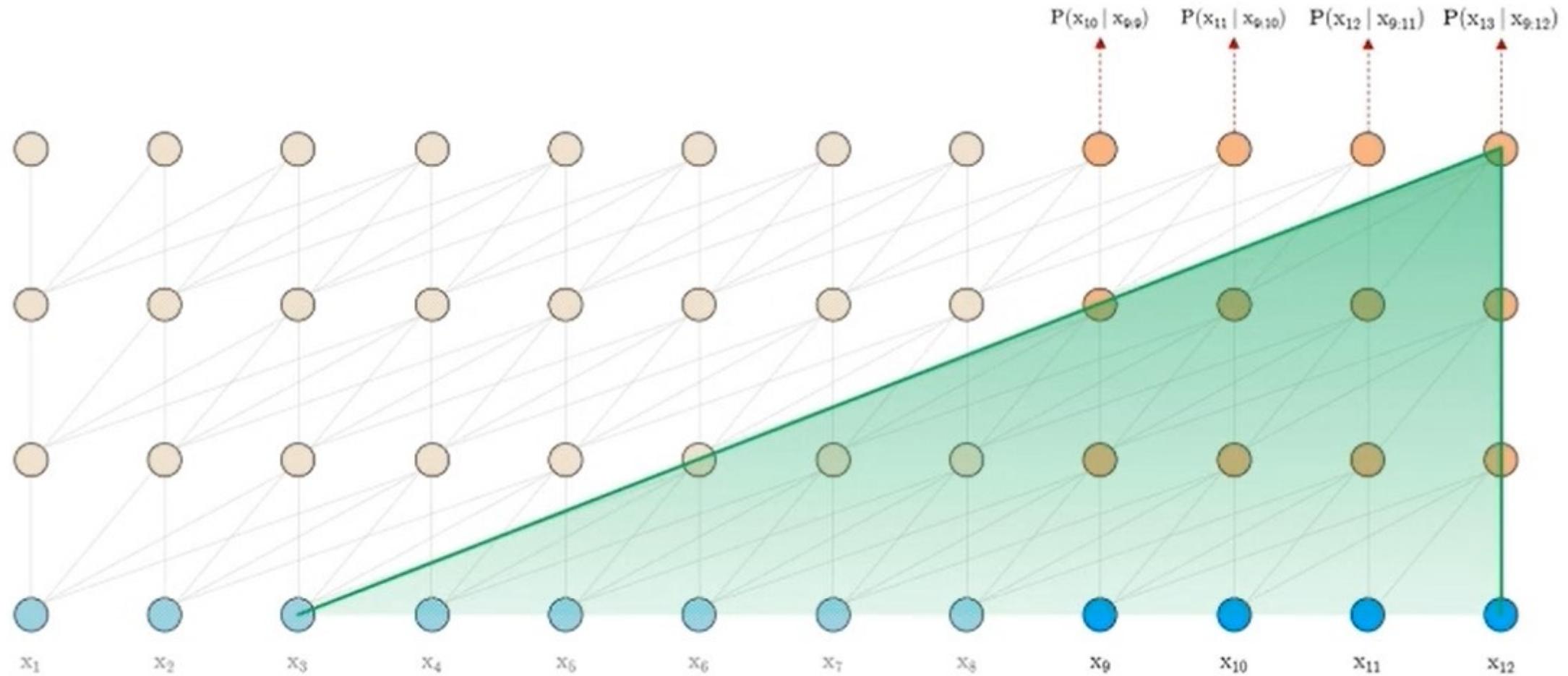
# Transformer XL training



# Transformer XL training

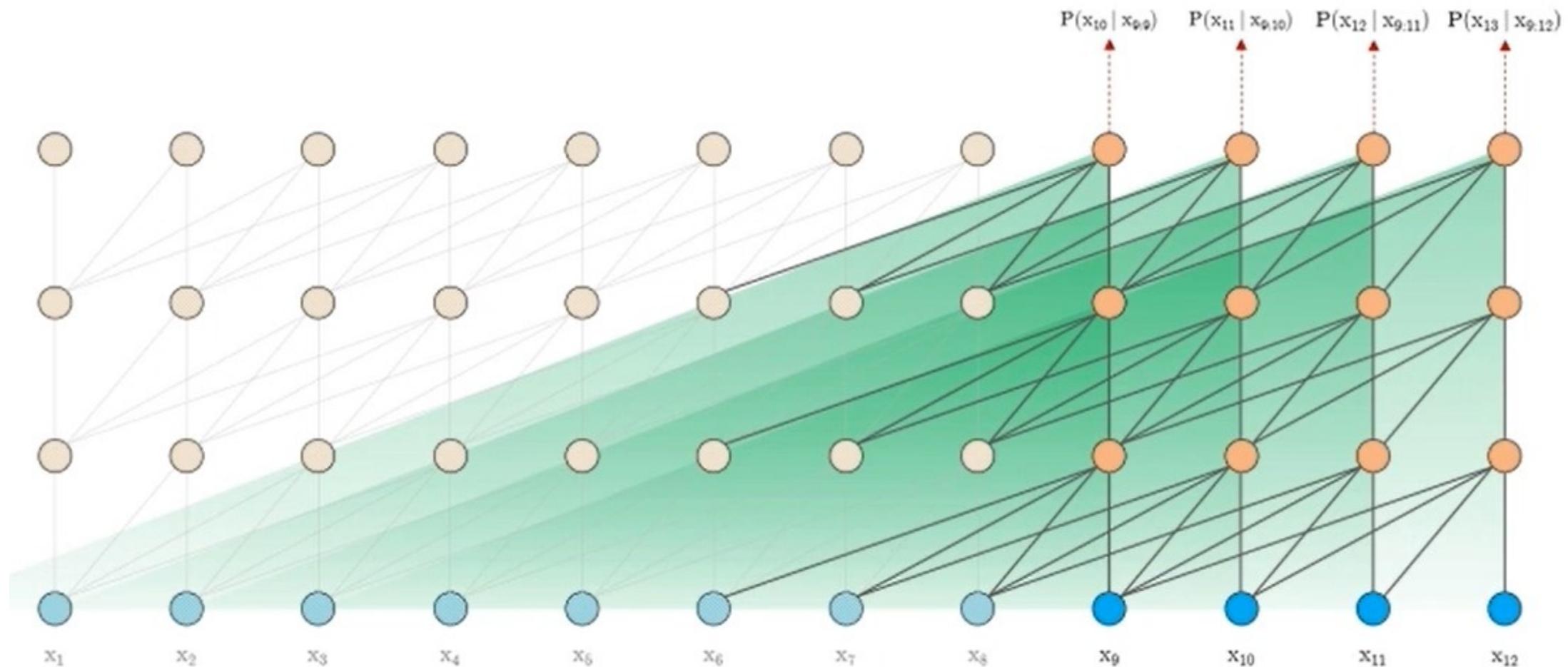


# Transformer XL: much longer context



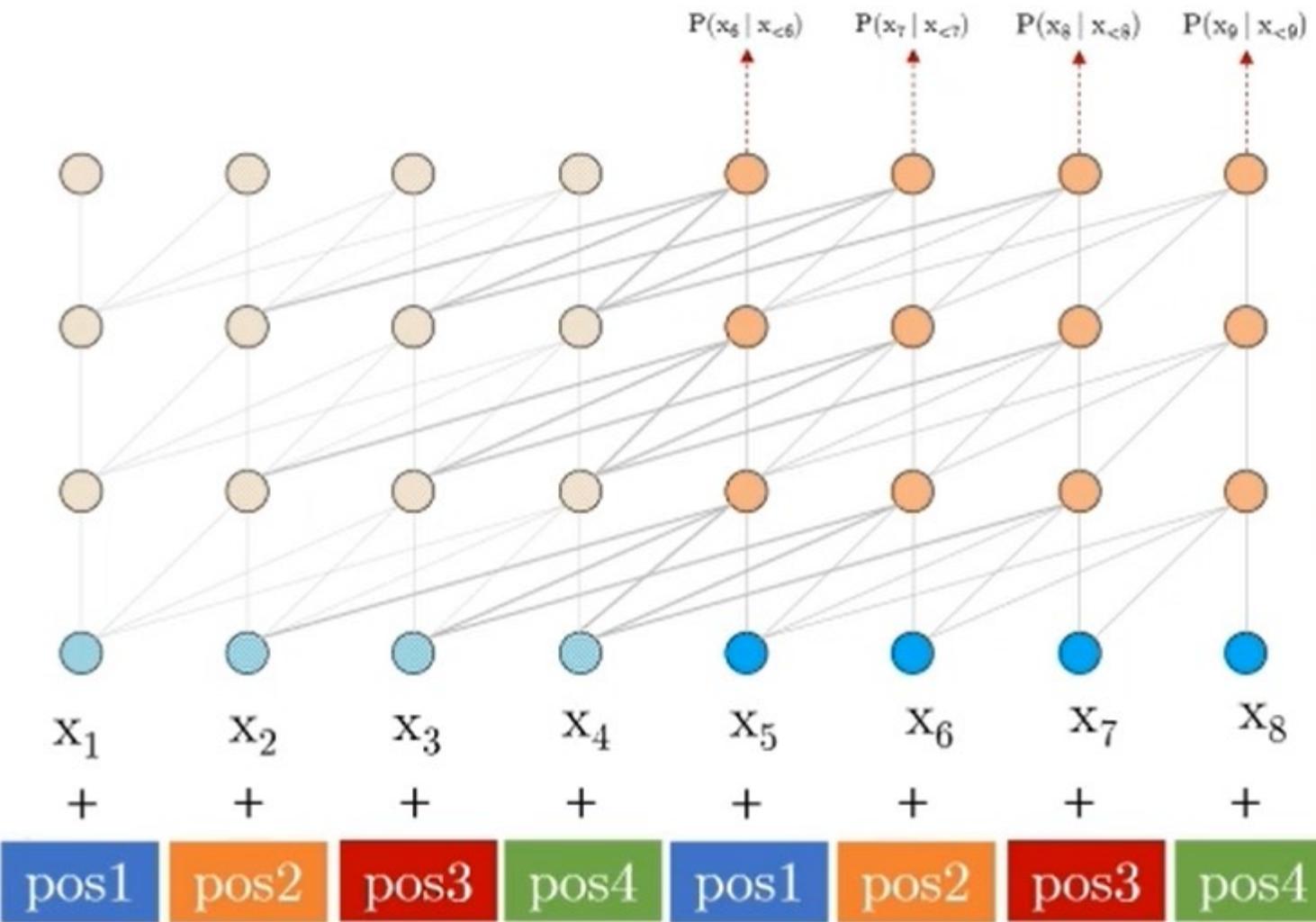
- **Extra-Long context span:** num\_layers  $\times$  segment\_len

# Transformer XL: evaluation



- No recomputation → much faster

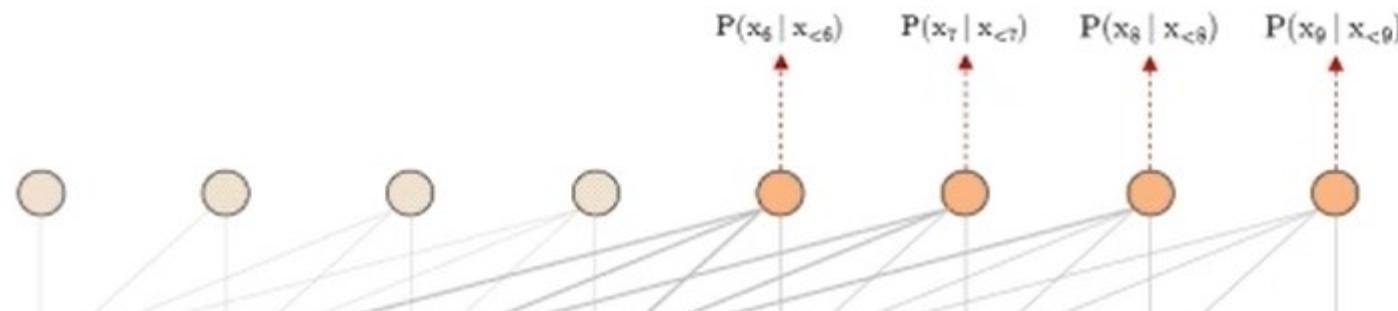
# A caveat: temporal incoherence



*Incoherent! 😞*

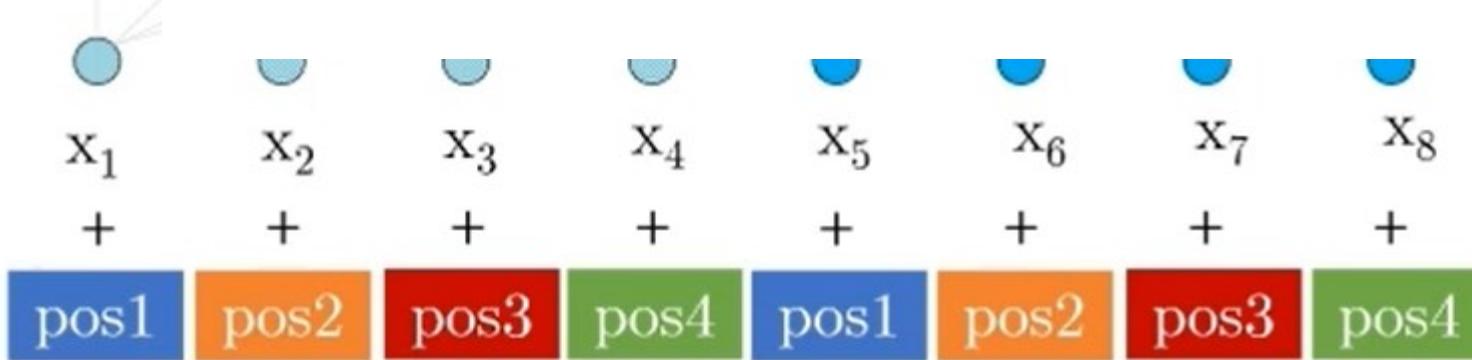
Different tokens have same position encoding.

# A caveat: temporal incoherence

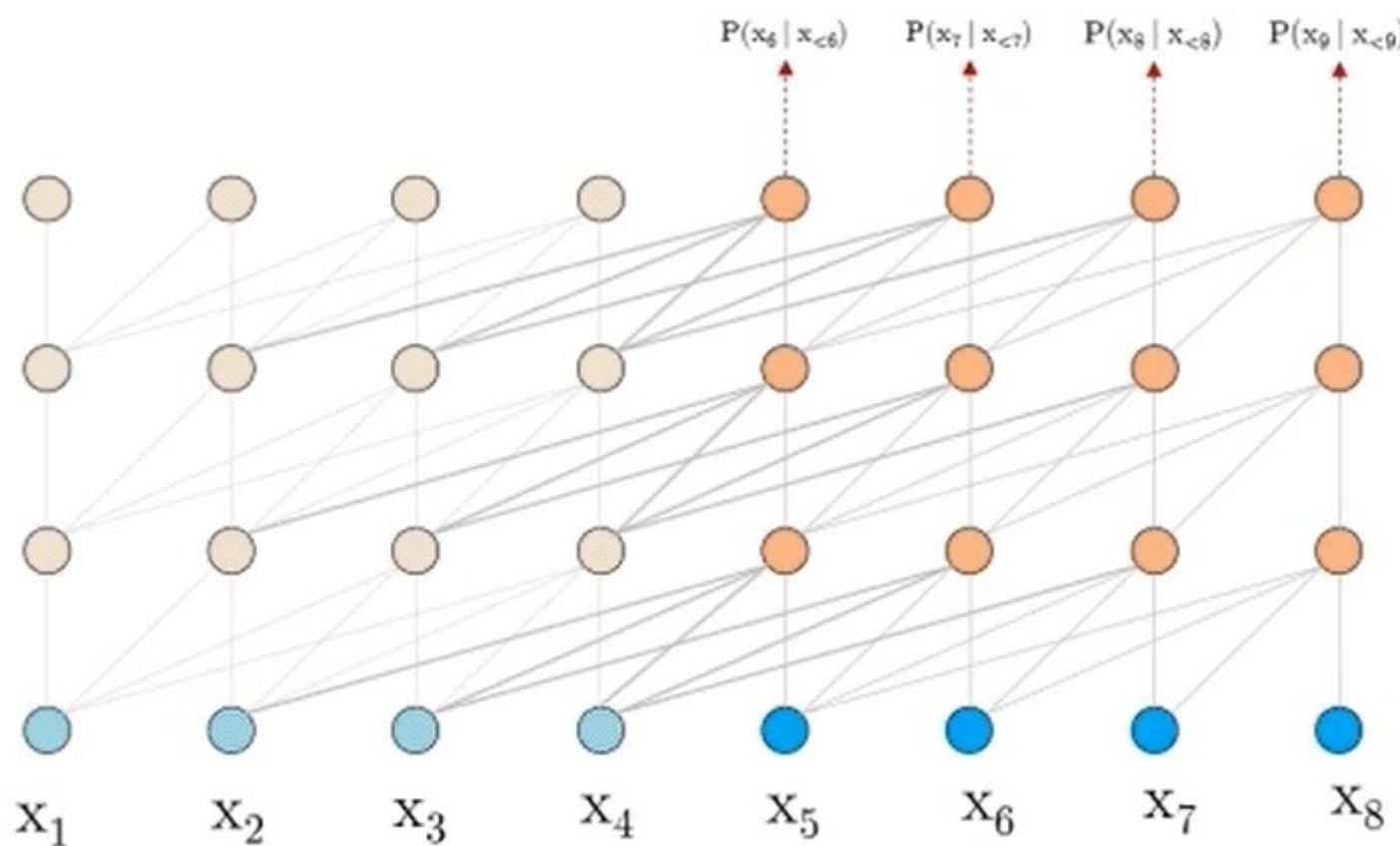


## Dilemma:

1. Positional encodings are important for performance
2. Positional encodings do not work with recurrence mechanism

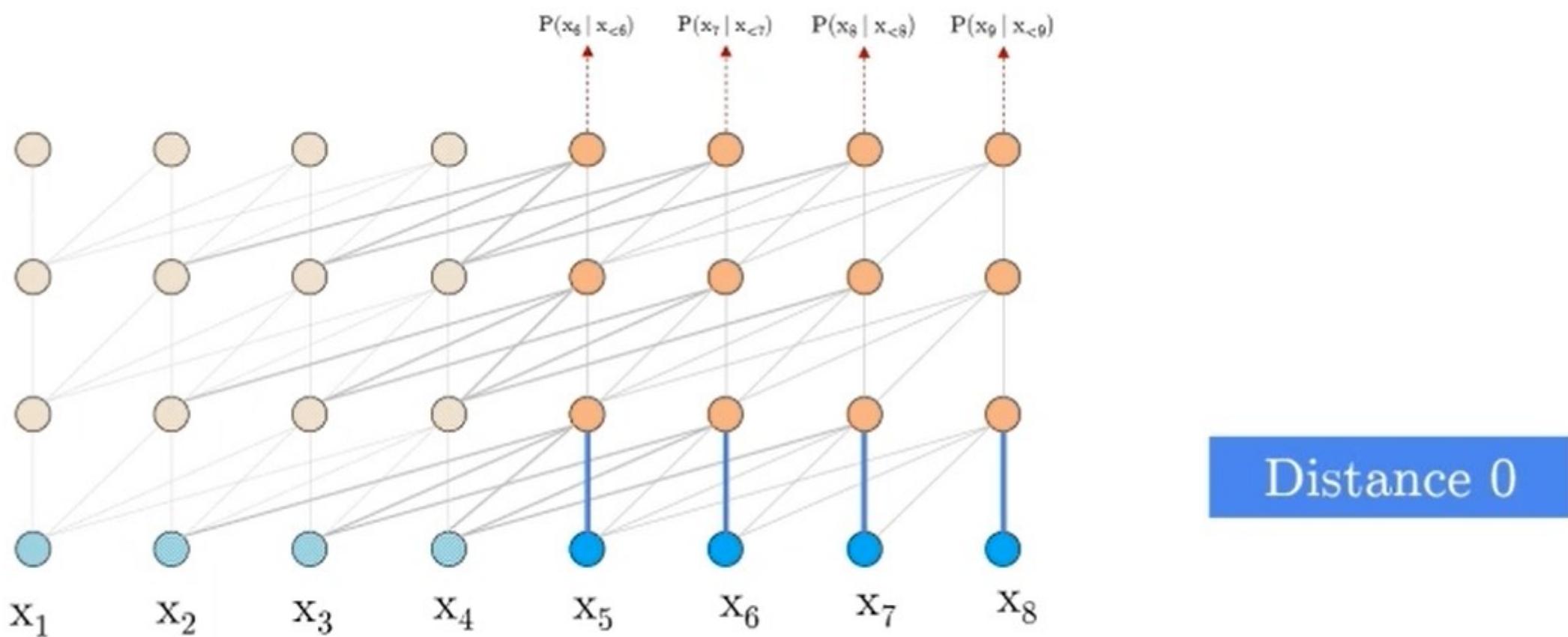


# Solution: relative positional embedding



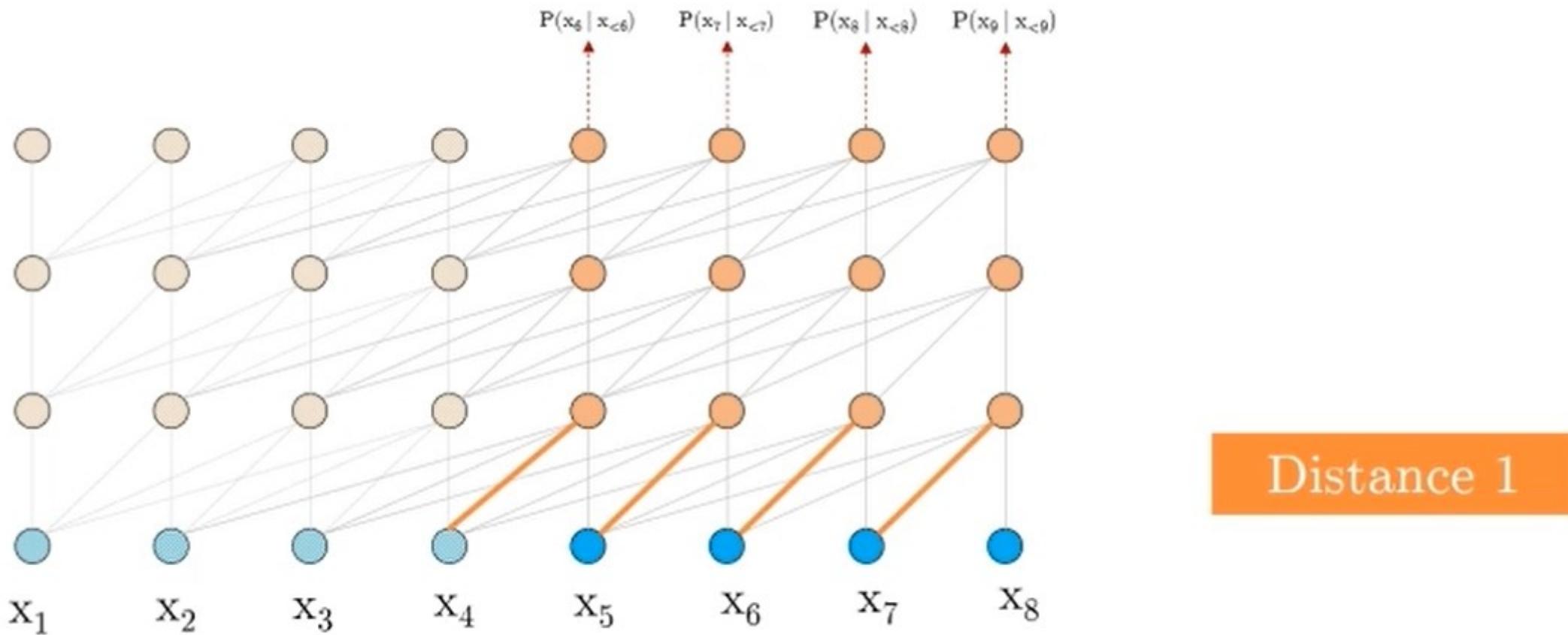
*Encode distances on edges!*

# Solution: relative positional embedding



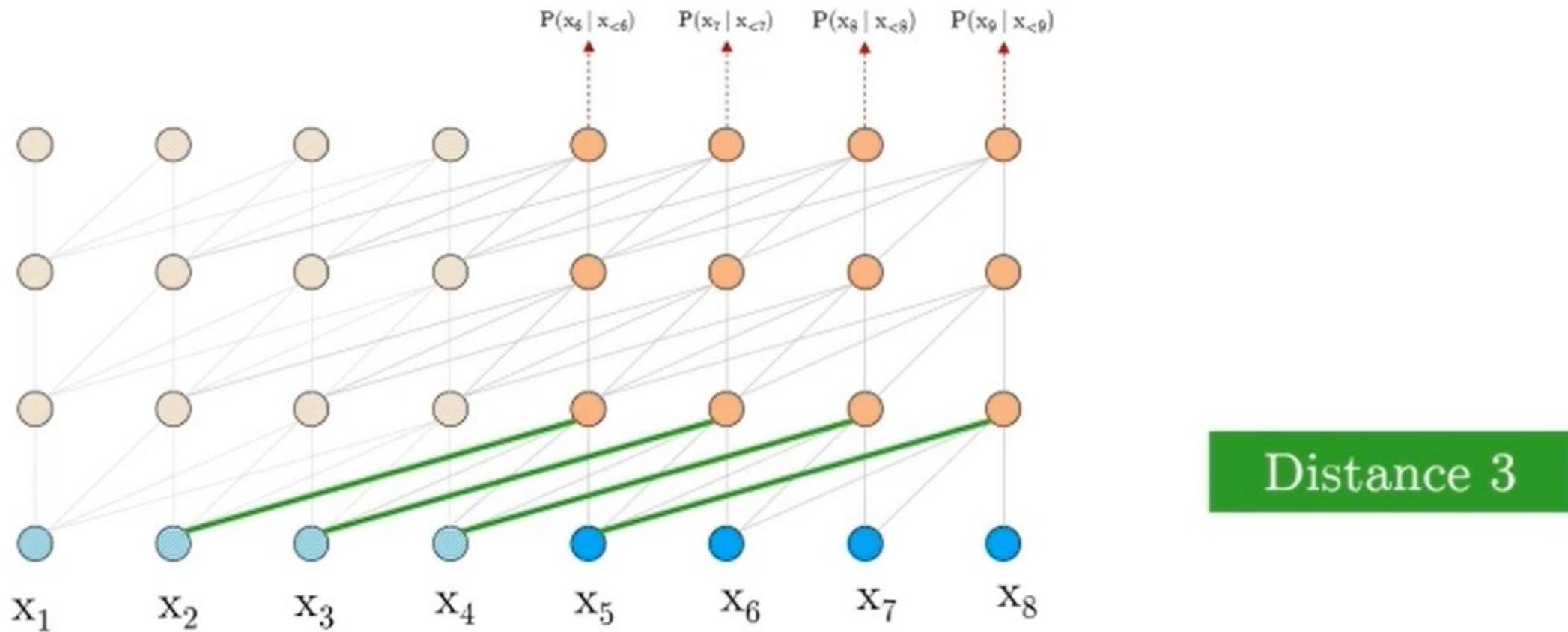
*Encode distances on edges!*

# Solution: relative positional embedding



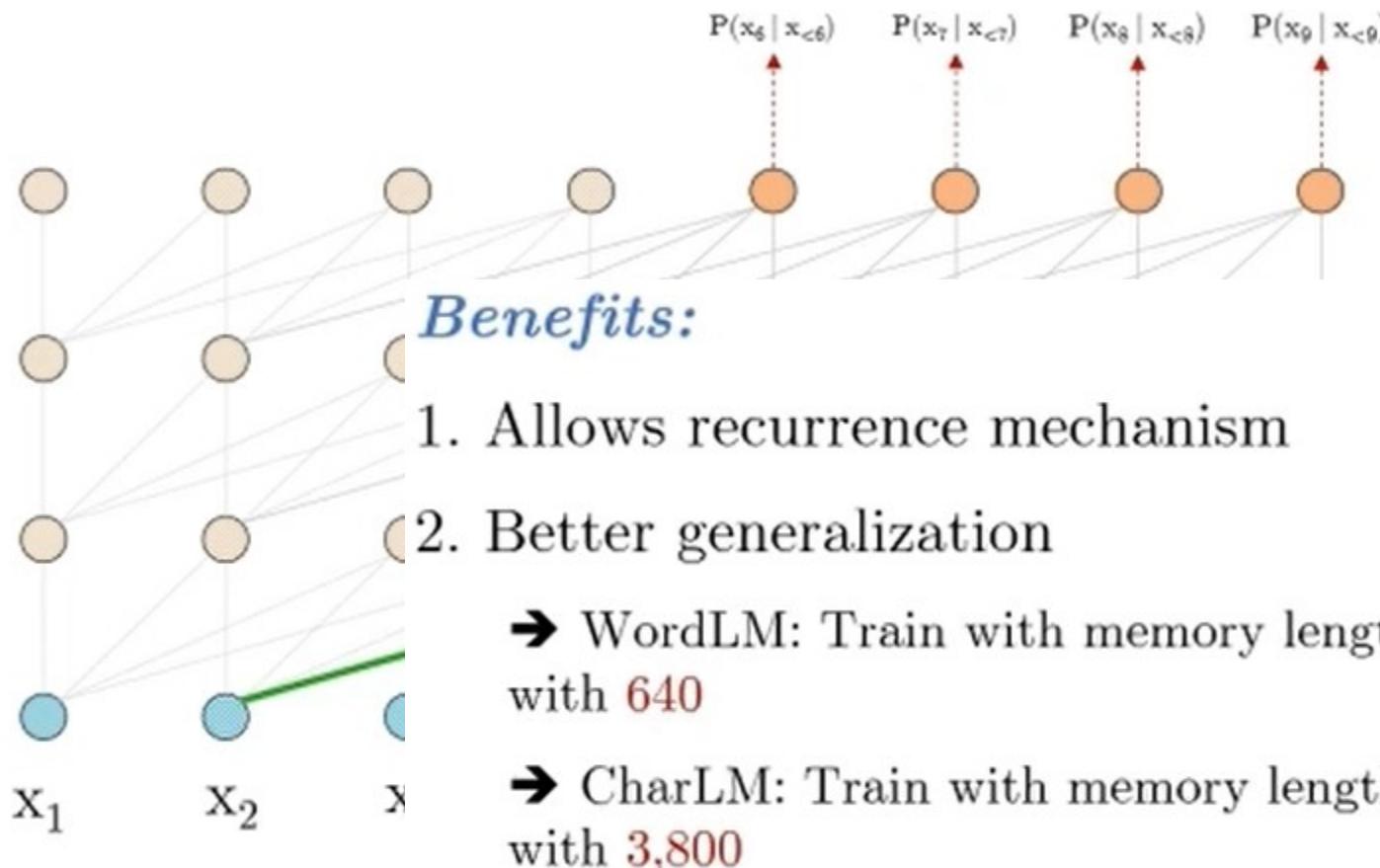
*Encode distances on edges!*

# Solution: relative positional embedding



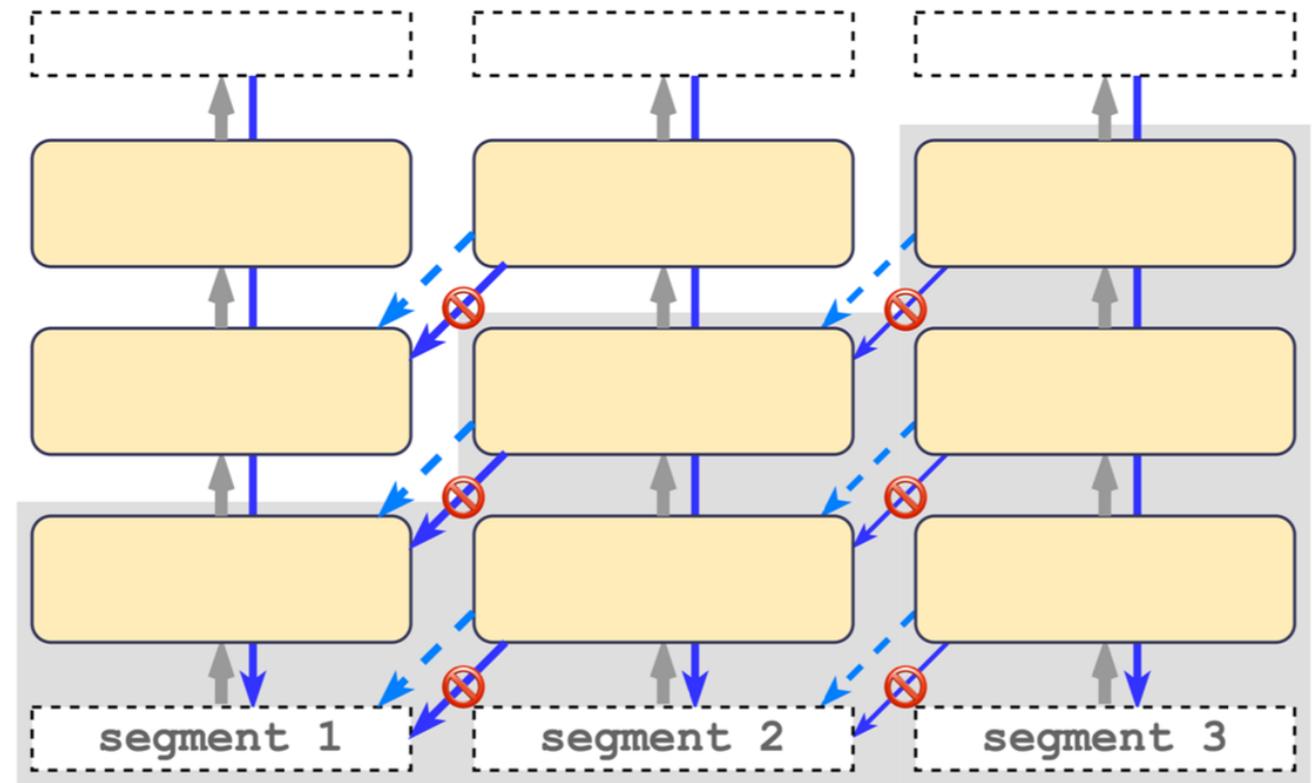
*Encode distances on edges!*

# Solution: relative positional embedding



*Encode distances on edges!*

# Transformer-XL recap



Transformer layer

Attention to  
cached states

Effective  
context

Forward pass

Gradients

Stop Gradient

# Transformer-XL recap

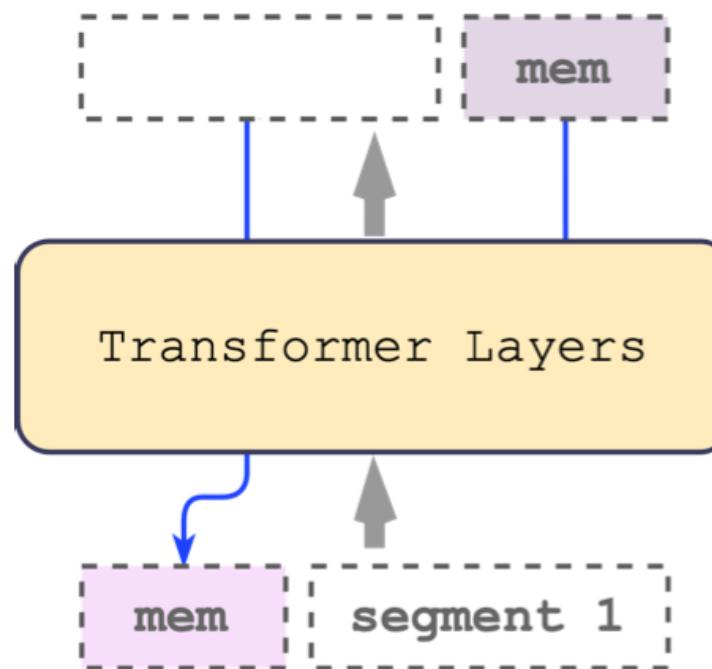
## Pros:

- As a result, Transformer-XL learns dependency that is 80% longer than RNNs and 450% longer than vanilla Transformers

## Cons:

- It's a completely different architecture!
- You need to store previous representation from each layer, and add them in the self attention at every layer!

# Recurrent Memory Transformer

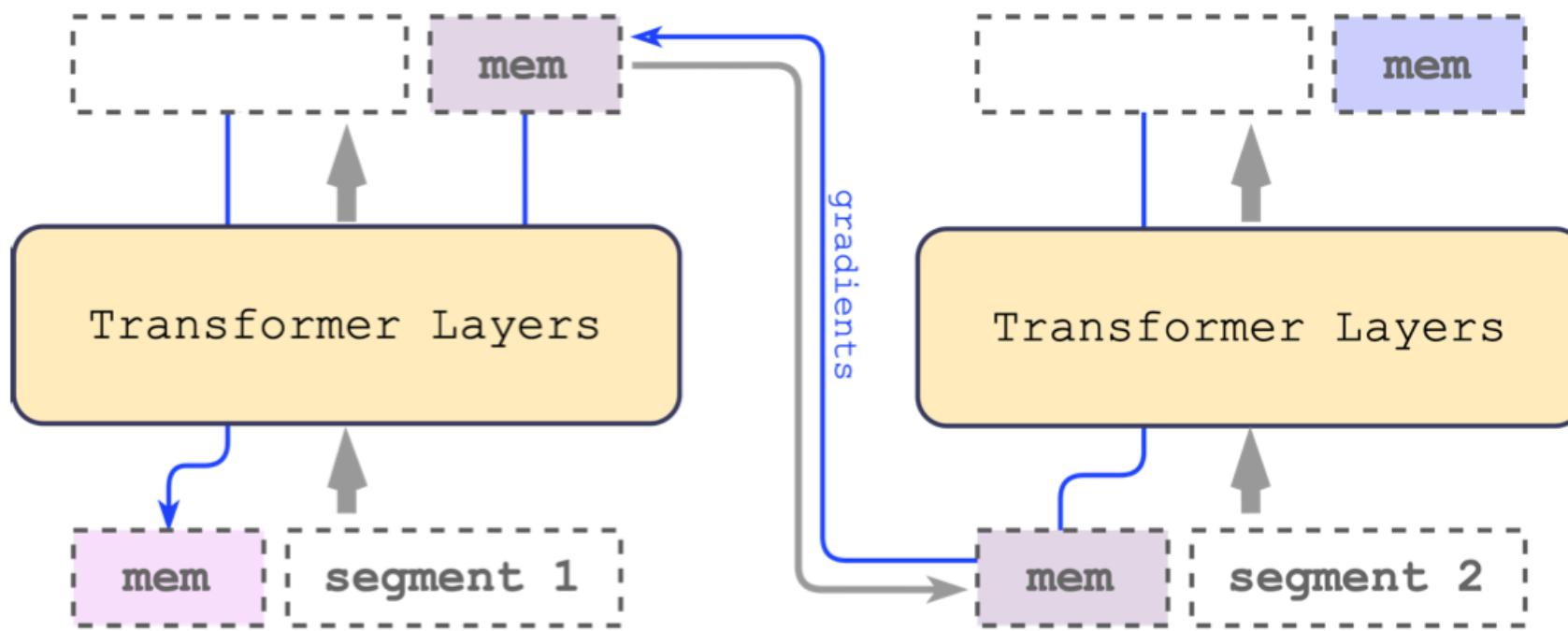


# Recurrent Memory Transformer

## Where to put memory + Why recurrency

- Memory tokens are **added to the beginning** of the input sequence, BUT
  - in decoder-only architectures** the causal attention mask makes impossible for memory tokens at the start of the sequence to collect information from the subsequent tokens.
- Memory tokens are **placed at the end** of the sequence, BUT
  - preceding tokens unable to access their representations. To solve this problem we add a recurrence to the sequence processing.

# Recurrent Memory Transformer

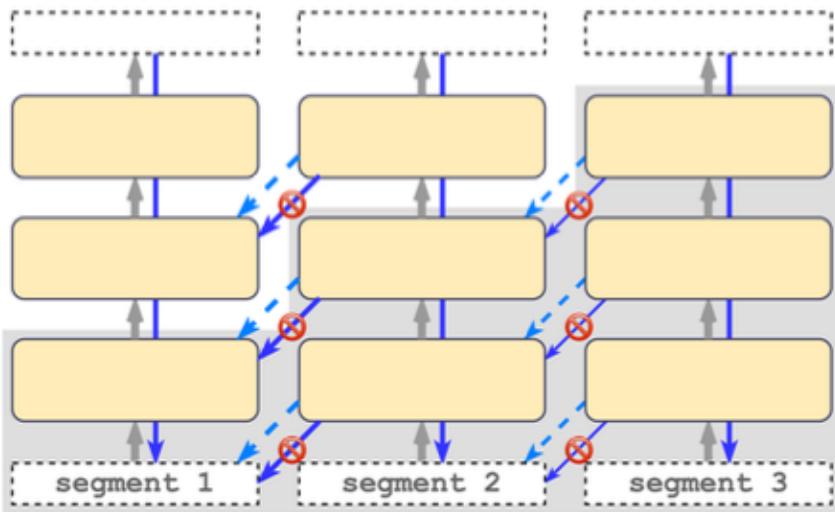


# Recurrent Memory Transformer

- Augmented Transformer with **token based memory storage**
- Outperform Transformer-XL on **memory-intensive tasks (copy, reverse, associate retrieval)**

# Differences with Transformer-XL

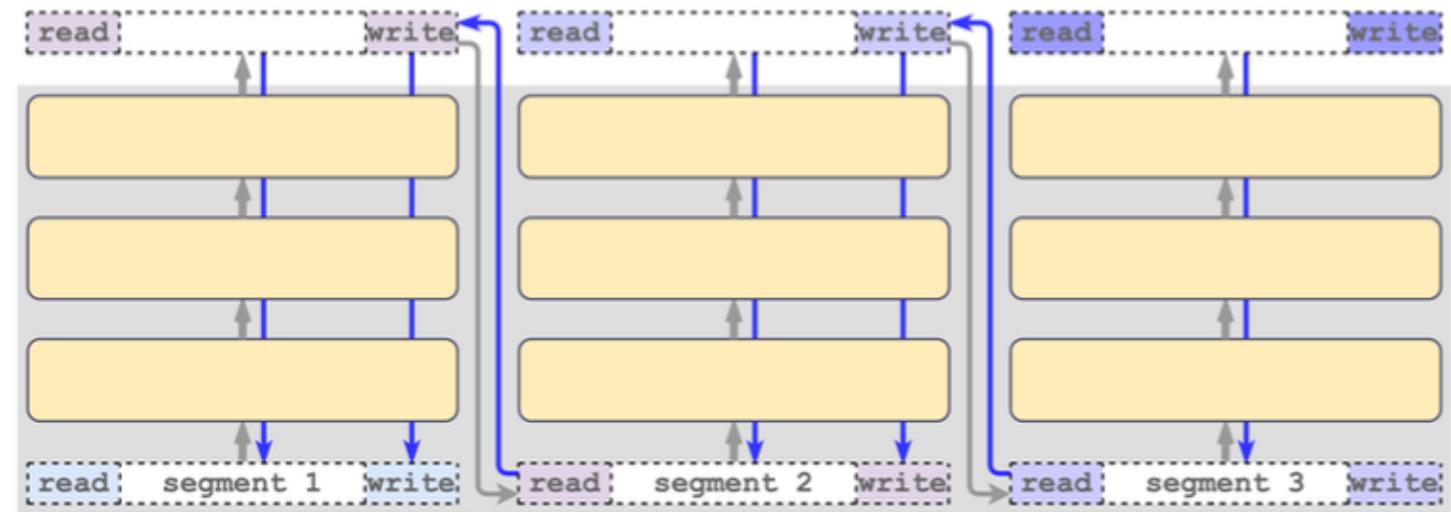
Transformer-XL



Transformer layer

Attention to  
cached states

Recurrent Memory Transformer



Effective  
context

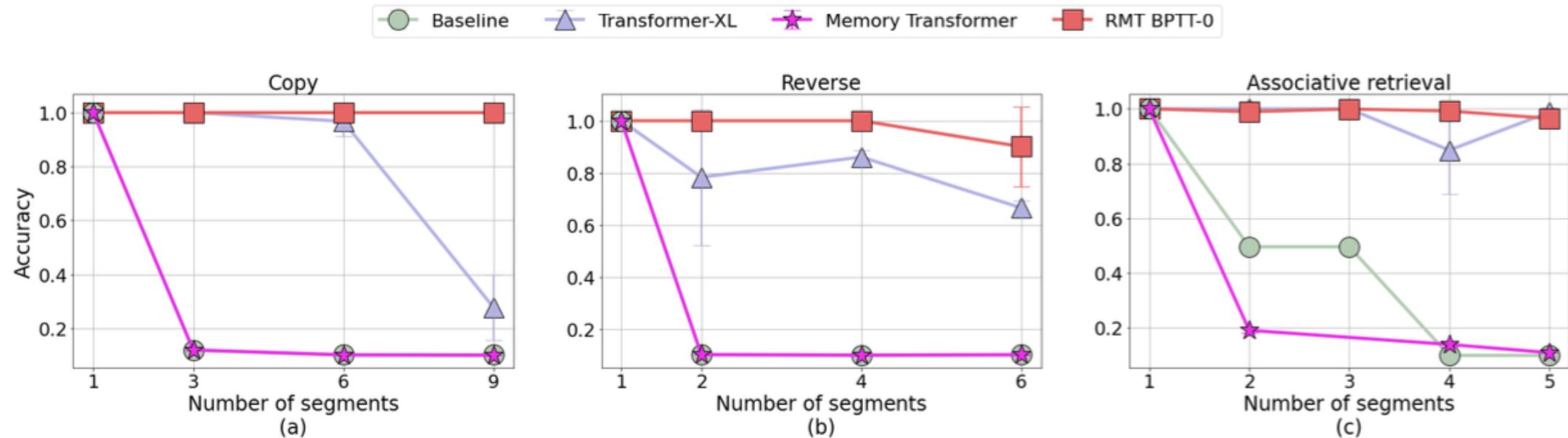
Forward pass

Gradients

Stop Gradient

# Memory Intensive Tasks

A baseline is used exploiting a Transformer-XL without memory



reproduce input sequence

reverse the input sequence

N key-value pairs. One key is randomly selected, and the task is to produce an appropriate value for the selected key.

# Scaling Transformer to 1M tokens and beyond with RMT

**Aydar Bulatov<sup>1</sup>**  
bulatov@deeppavlov.ai

**Yuri Kuratov<sup>1,2</sup>**  
kuratov@airi.net

**Mikhail S. Burtsev<sup>1,3</sup>**  
mbur@lims.ac.uk

<sup>1</sup>DeepPavlov

<sup>2</sup>Artificial Intelligence Research Institute (AIRI)

<sup>3</sup>London Institute for Mathematical Sciences

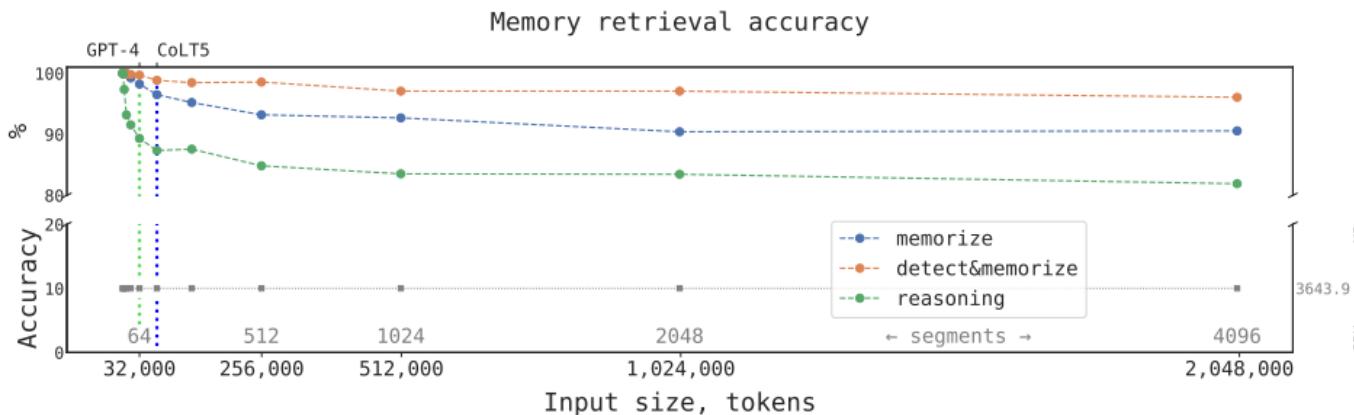


Figure 1: **Recurrent Memory Transformer retains information across up to  $2 \times 10^6$  tokens.** By augmenting a pre-trained BERT model with recurrent memory (Bulatov et al., 2022), we enabled it to store task-specific information across 7 segments of 512 tokens each. During inference, the model effectively utilized memory for up to 4,096 segments with a total length of 2,048,000 tokens—significantly exceeding the largest input size reported for transformer models (64K tokens for CoLT5 (Ainslie et al., 2023), and 32K tokens for GPT-4 (OpenAI, 2023)). This augmentation maintains the base model’s memory size at 3.6 GB in our experiments.

# Thank you for attending

## *Memory Networks*

### *Schedule*

PART 0 - *Introduction to Neural Networks*

PART 1 - *Memory Networks*

RNN; LSTM; GRU. Applications.

PART 2 - *Memory Augmented Neural Networks*

NTM; MANN; End-to-End Memory Network;  
KV Memory Network; Episodic vs Persistent Memory  
Networks; MANTRA; SMEMO. Applications.

PART 3 - *Memory and Attention*

Transformers; Introductory issues; Transformers and  
Memory

# References

## DEEP NETWORKS

- Deep Learning  
I.Goodfellow, Y. Bengio, A. Courville, MIT Press 2016 (online free)
- *Neural Networks and Deep Learning*  
M. Nielsen, 2017. (online free)
- *Deep Learning with Python*  
F. Chollet, Manning Pubs, 2017
- *Hands-On Machine Learning with Scikit-Learn and TensorFlow*  
A. Géron, O'Reilly Media, 2017

## MEMORY NETWORKS

- *Long short-term memory*  
S. Hochreiter; J. Schmidhuber, Neural Computation 1997
- *Long-term Recurrent Convolutional Networks for Visual Recognition and Description*  
J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, T. Darrell, CVPR 2015
- *Stacked Attention Networks for Image Question Answering*  
Z. Yang, X. He, J. Gao2, L. Deng, A. Smola, CVPR 2017
- *Image Captioning with Semantic Attention*  
Q. You, H. Jin, Z. Wang, C. Fang, J. Luo, CVPR 2016

- *From captions to visual concepts and back*  
H. Fang, S. Gupta, F. Iandola, R. Srivastava, L. Deng, P. Dollar, J. Gao, X. He, M. Mitchell, J. Platt, et al. CVPR 2015
  - *Bottom-Up and Top-Down Attention for Image Captioning and Visual Question Answering*  
P. Anderson, X. He, C. Buehler, D. Teney, M. Johnson, S. Gould L. Zhang, CVPR 2018
  - *Am I Done? Predicting Action Progress in Videos*  
F. Becattini, T. Uricchio, L. Seidenari, L. Ballan, A. Del Bimbo, TOMM 2020
  - *Recurrent Neural Network Tutorial*  
D. Blitz, 2015
- 
- MEMORY AUGMENTED NETWORKS
    - *iCaRL: Incremental Classifier and Representation Learning*  
S. Rebuffi, A. Kolesnikov, G. Sperl, C. H. Lampert, CVPR 2017
    - *End-To-End Memory Networks,*  
S. Sukhbaatar, A. Szlam, J. Weston, R. Fergus, NIPS 2015
    - *Neural Turing Machines*  
A. Graves, G. Wayne, I. Danihelka, arXiv 2014
    - *Meta-Learning with Memory-Augmented Neural Networks*  
A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, T. Lillicrap, ICML 2016
    - *Mantra: Memory augmented networks for multiple trajectory prediction*  
F. Marchetti, F. Becattini, L. Seidenari, A. Del Bimbo, CVPR 2020
    - *A review on Neural Turing Machine*  
S. Malekmohammadi F. Faramarz Safi-Esfahani , ArXiv 2019

- TRANSFORMERS

- *Attention is all you need*

- A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. Gomez, L. Kaiser, I. Polosukhin, NIPS 2017

- *Bert: Pre-training of deep bidirectional transformers for language understanding.*

- J. Devlin, M. Chang, K. Lee, K. Toutanova, NAACL 2019

- *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*

- A. Dosovitskiyand, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly and others, ICLR 2021

- *End-to-End Object Detection with Transformers*

- N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, S. Zagoruyko, ECCV 2020

- *Transformers in Vision: A Survey*

- S. Khan, M. Naseer, M. Hayat, S. Waqas Zamir, F. Shahbaz Khan, M. Shah, arxiv 2021