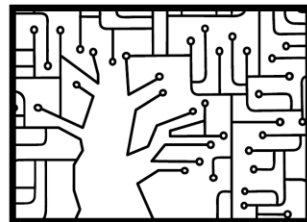


Deep Learning for Computer Vision: Sequences and Attention

Shai Bagon

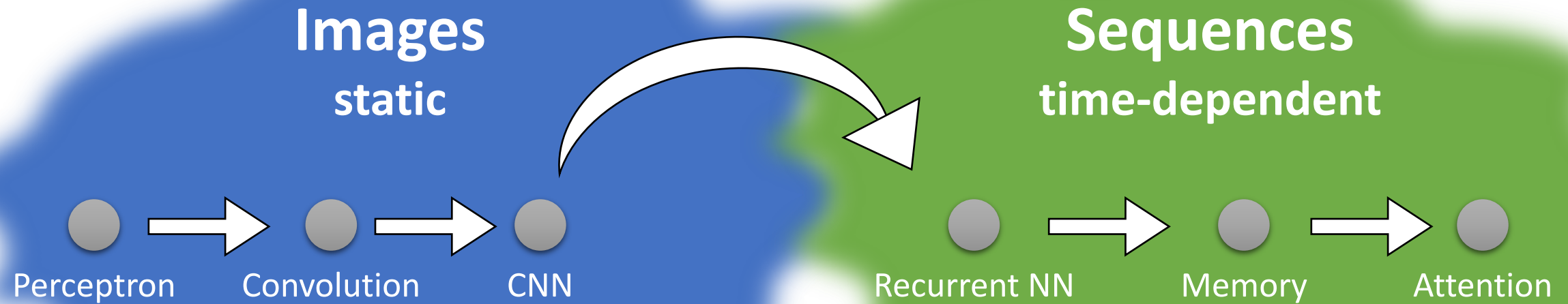


WAIC

100
DAYS IN HELL

Bring Them Home Now!

Agenda

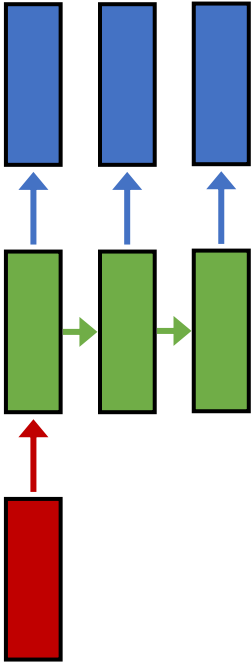


Deep Learning for Sequences

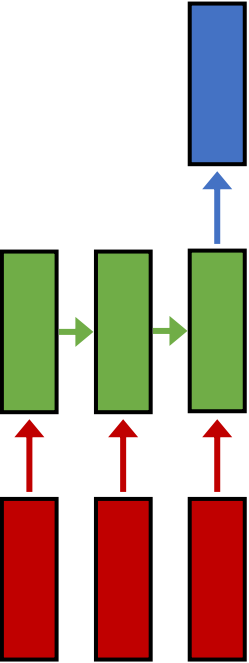
One to one



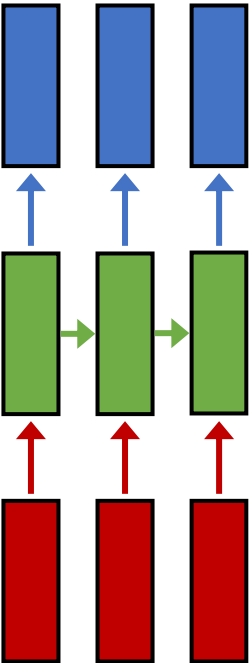
One to many



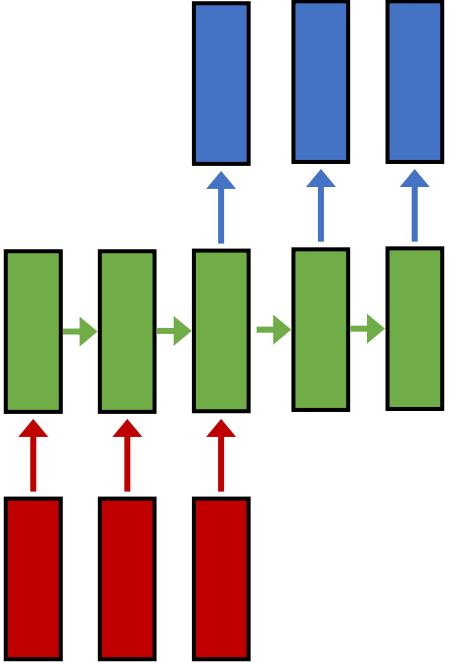
Many to one



Many to many



Many to many



Feed-for

e.g., **image classification**

e.g., **video classification**

e.g., **video frames classification**

e.g., **Machine translation**

e.g., **classification**

image -> sequence

sequence of frames

sequence of frames -> sequence

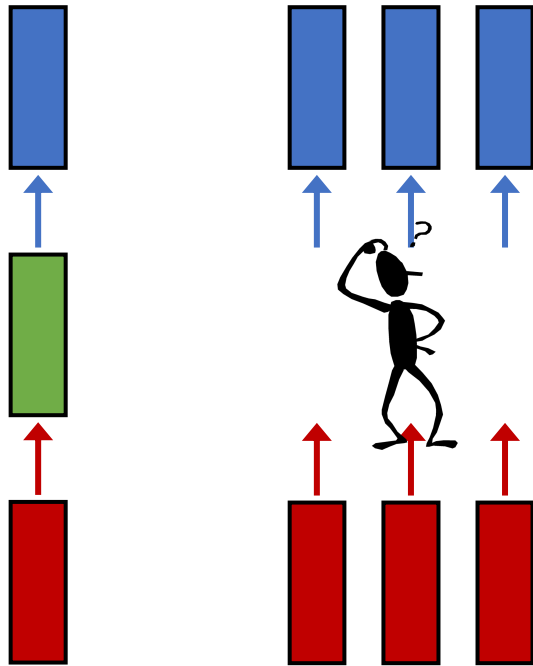
sequence of words -> sequence of words

image -> label



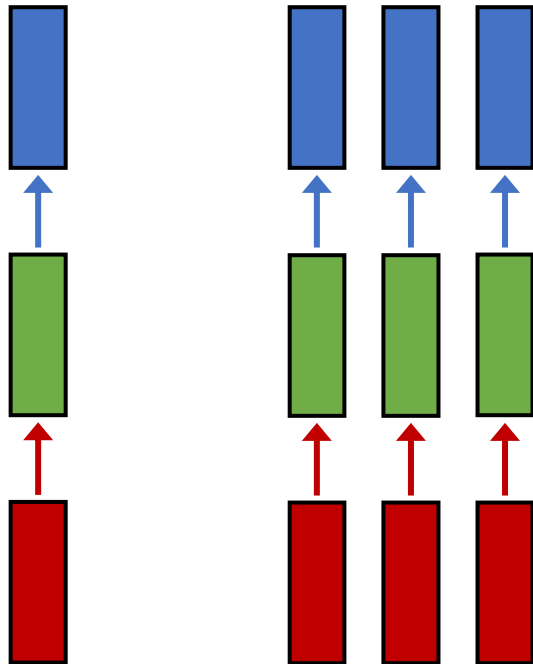
Recurrent Neural Networks

One to one Many to many



Recurrent Neural Networks

One to one Many to many



```
class ManyToManyV0(nn.Module):
    def __init__(self, number_of_time_steps):
        super(ManyToMany, self).__init__()
        # SAME instance of SingleTimeStep for each time step
        self.time_steps = SingleTimeStep(...)

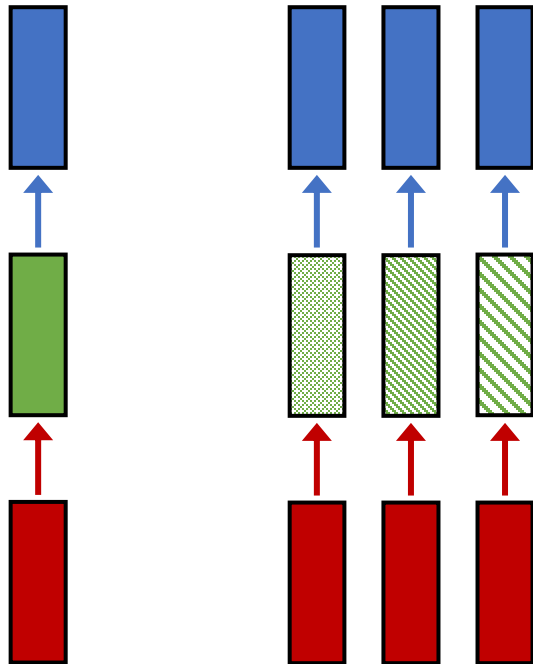
    def forward(self, in_seq):
        out_pred = []
        for t, x_t in enumerate(in_seq):
            p_t = self.time_steps(x_t)
            out_pred.append(p_t)
        return out_pred
```

(+) Parameter efficient

(-) No temporal dependency

Recurrent Neural Networks

One to one Many to many



```
class ManyToManyV1(nn.Module):
    def __init__(self, number_of_time_steps):
        super(ManyToMany, self).__init__()
        # DIFFERENT instance of SingleTimeStep for each time step
        self.time_steps = nn.ModuleList([SingleTimeStep(...)
                                         for _ in range(number_of_time_steps)])

    def forward(self, in_seq):
        out_pred = []
        for t, x_t in enumerate(in_seq):
            p_t = self.time_steps[t](x_t)
            out_pred.append(p_t)
        return out_pred
```

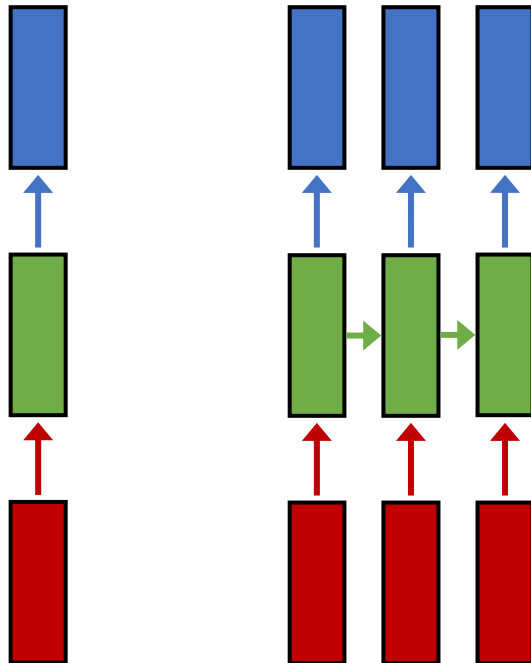
(+?) Temporal dependency (via trained parameters)

(-) Parameter inefficiency

(-) Fixed sequence length

Recurrent Neural Networks

One to one Many to many



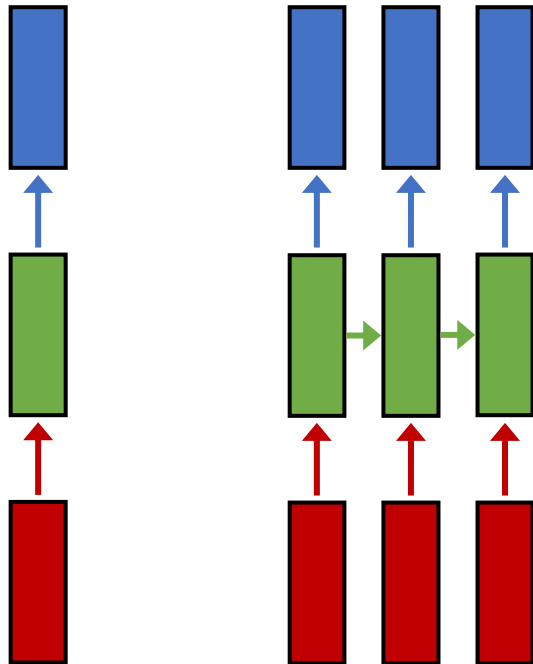
```
class ManyToMany(nn.Module):
    def __init__(self):
        super(ManyToMany, self).__init__()
        # SHARE a single instance of SingleTimeStep
        self.time_step = SingleTimeStep(...)

    def forward(self, in_seq):
        out_pred = []
        state = self.init_state
        for t, x_t in enumerate(in_seq):
            p_t, state = self.time_step(x_t, state)
            out_pred.append(p_t)
        return out_pred
```

- (+) Temporal dependency (via “hidden state”)
- (+) Parameter efficiency
- (+) Arbitrary sequence length

Recurrent Neural Networks

One to one Many to many



```
class ManyToMany(nn.Module):
    def __init__(self):
        super(ManyToMany, self).__init__()
        # SHARE a single instance of SingleTimeStep
        self.time_step = SingleTimeStep(...)

    def forward(self, in_seq):
        out_pred = []
        state = self.init_state
        for t, x_t in enumerate(in_seq):
            p_t, state = self.time_step(x_t, state)
            out_pred.append(p_t)
        return out_pred
```

$$h_t = f(h_{t-1}, x_t; W)$$

Example: Language Modeling

Task:

Given characters c_0, c_1, \dots, c_{t-1}

Predict c_t

Training sequence: "hello"

Vocabulary: ['h', 'e', 'l', 'o']

Embedding Layer:

Input sequence:

1
0
0
0

0
1
0
0

0
0
1
0

0
0
1
0

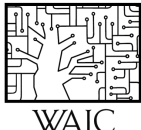
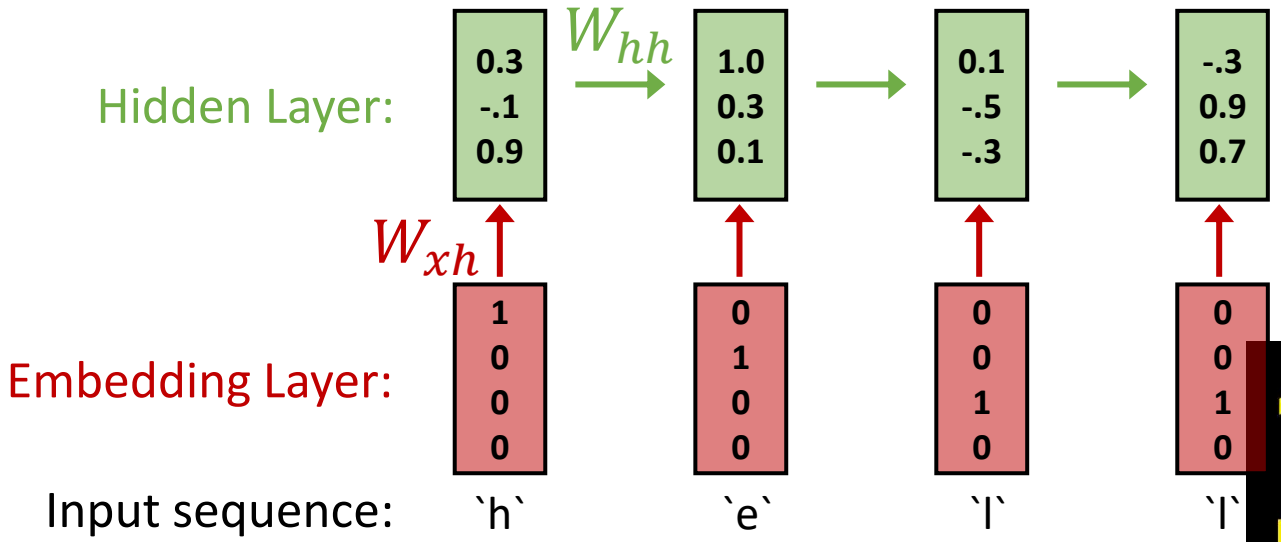
Slide credit: [Justin Johnson \(EECS-498-007, UMich\)](#)

Example: Language Modeling

Task:
Given characters c_0, c_1, \dots, c_{t-1}
Predict c_t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: "hello"
Vocabulary: ['h', 'e', 'l', 'o']

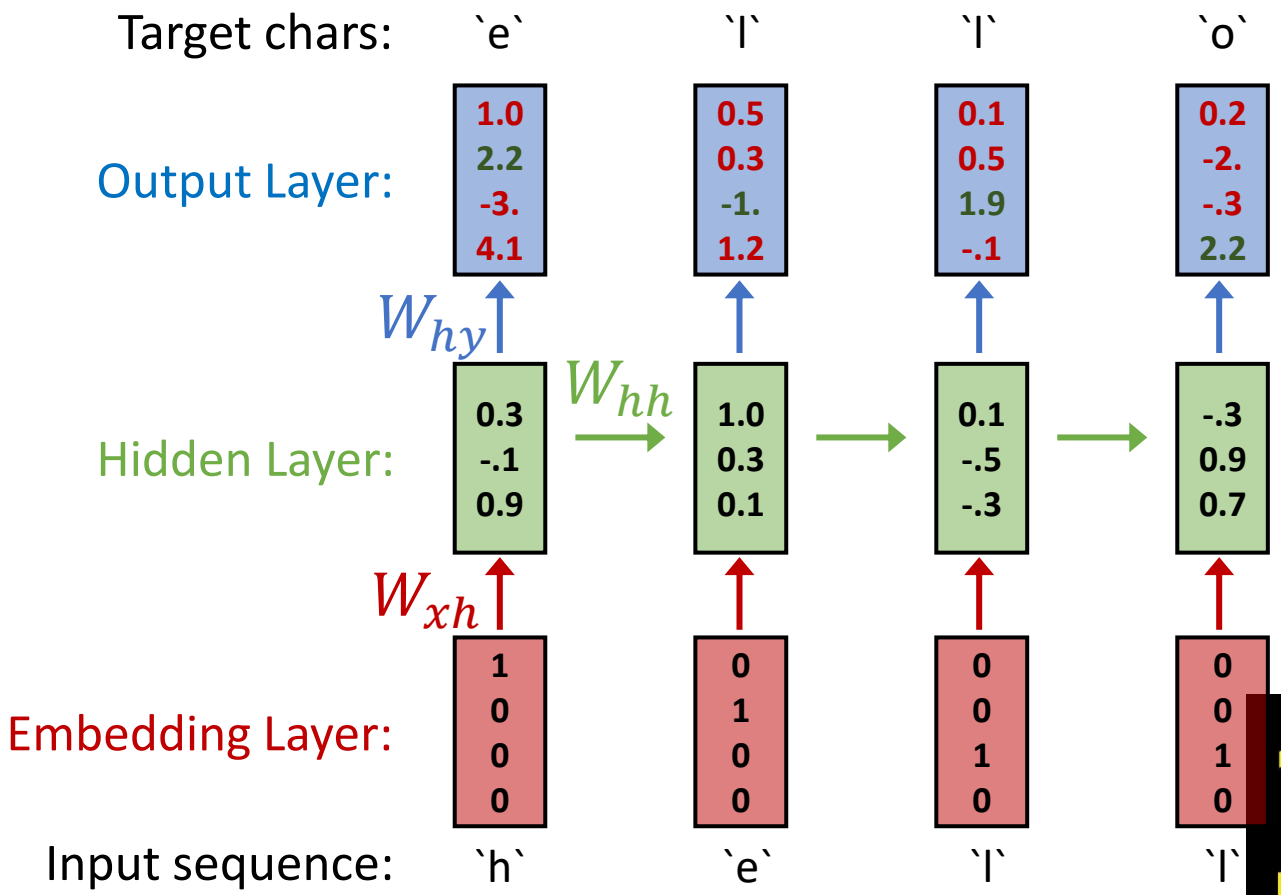


Example: Language Modeling

Task:
 Given characters c_0, c_1, \dots, c_{t-1}
 Predict c_t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Training sequence: "hello"
 Vocabulary: ['h', 'e', 'l', 'o']



Example: Language Modeling

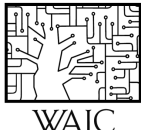
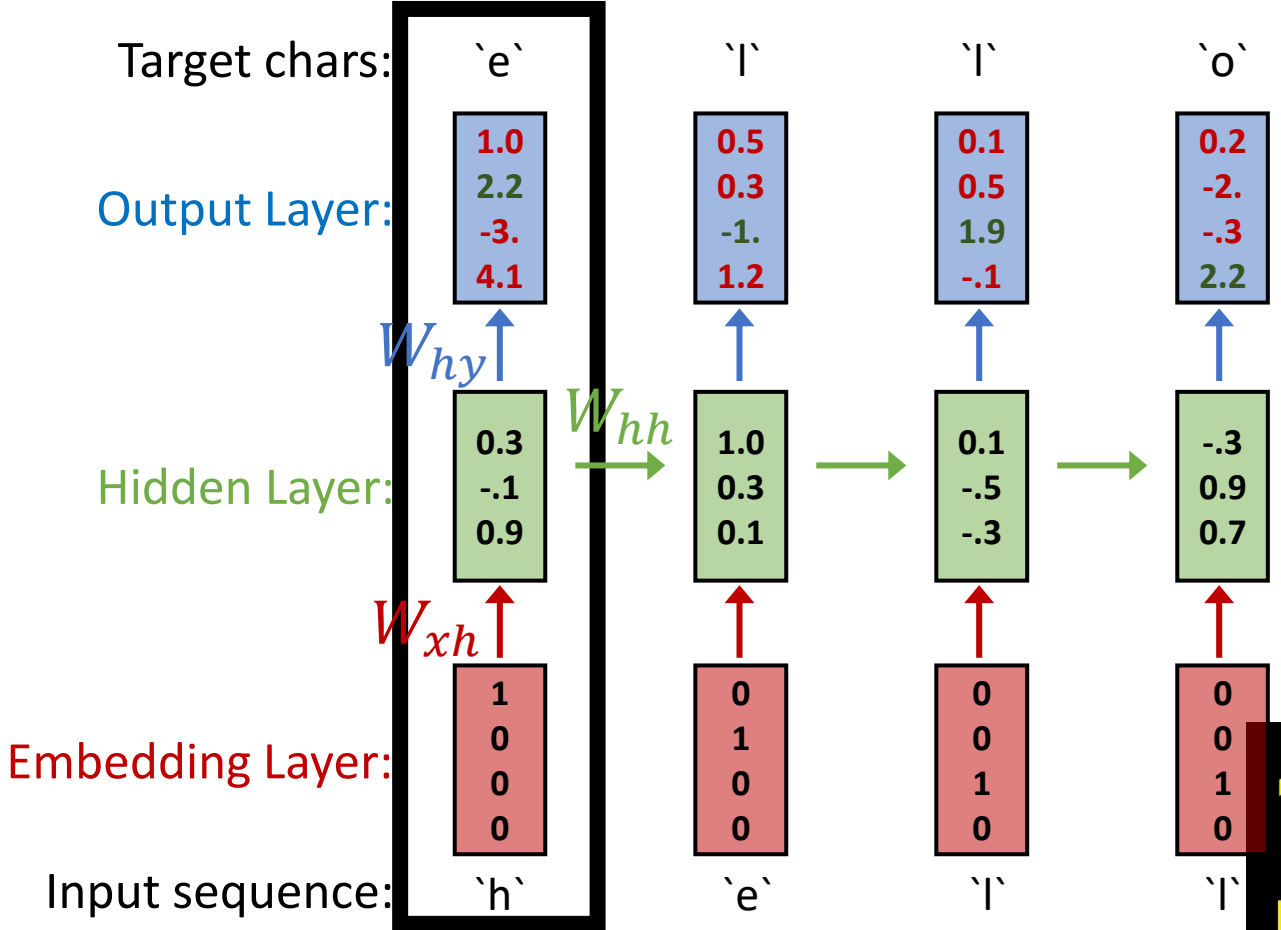
Task:
 Given characters c_0, c_1, \dots, c_{t-1}
 Predict c_t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Training sequence: "hello"
 Vocabulary: ['h', 'e', 'l', 'o']

Given "h" predict "e"



Example: Language Modeling

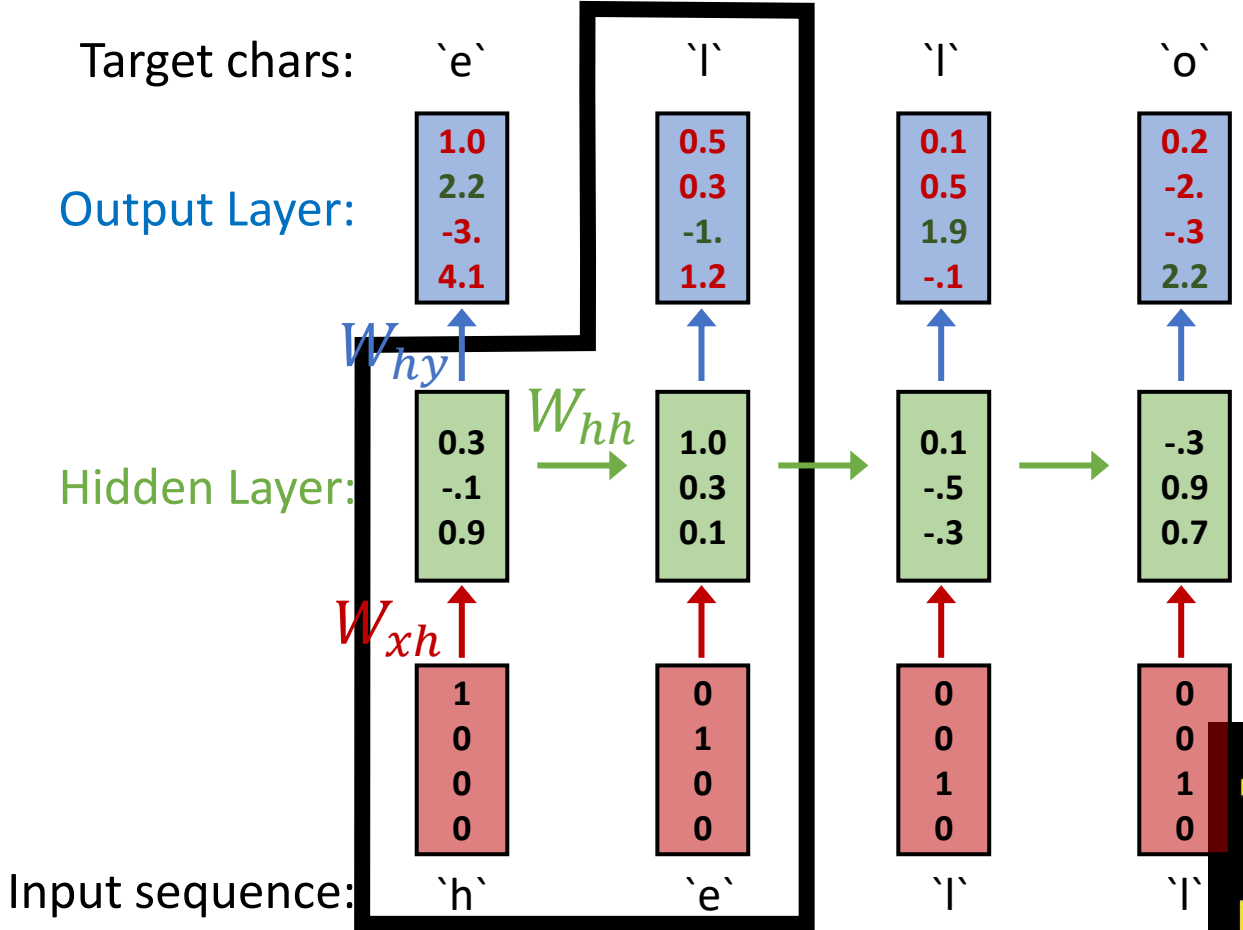
Task:
 Given characters c_0, c_1, \dots, c_{t-1}
 Predict c_t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Training sequence: "hello"
 Vocabulary: ['h', 'e', 'l', 'o']

Given "he" predict "l"



Example: Language Modeling

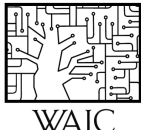
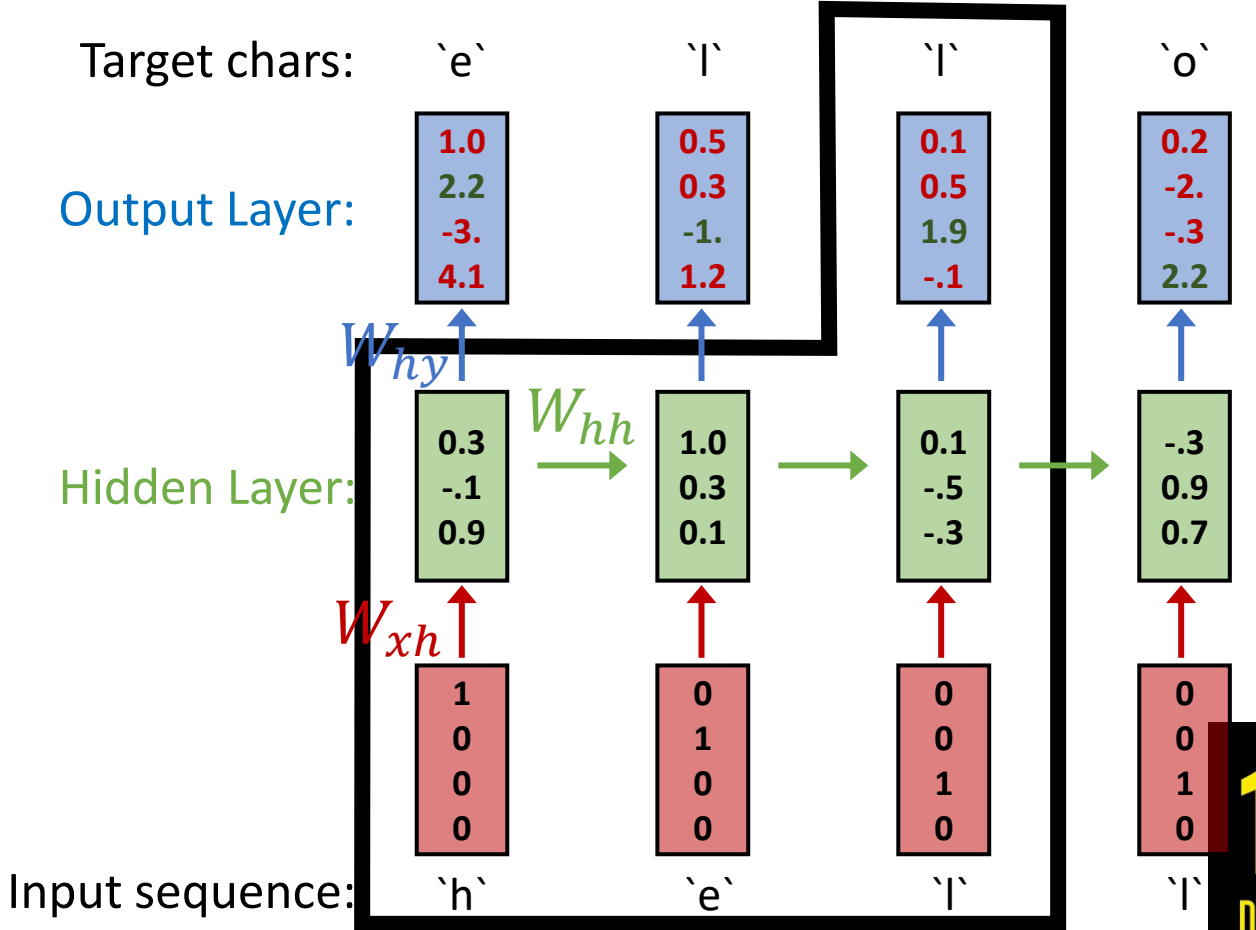
Task:
 Given characters c_0, c_1, \dots, c_{t-1}
 Predict c_t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Training sequence: "hello"
 Vocabulary: ['h', 'e', 'l', 'o']

Given "hel" predict "l"



Example: Language Modeling

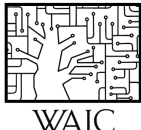
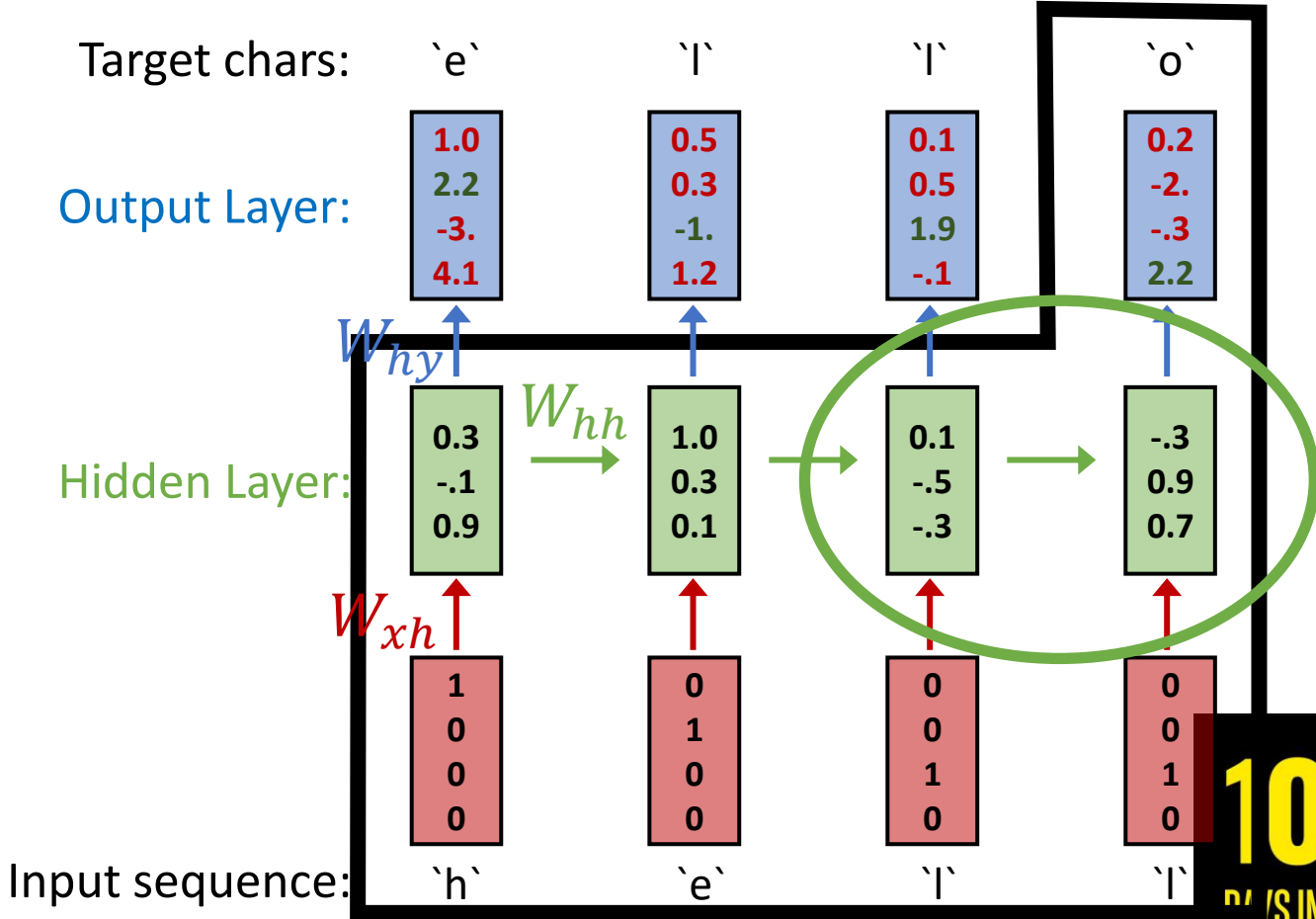
Task:
 Given characters c_0, c_1, \dots, c_{t-1}
 Predict c_t

$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$y_t = W_{hy}h_t$$

Training sequence: "hello"
 Vocabulary: ['h', 'e', 'l', 'o']

Given "hell" predict "o"

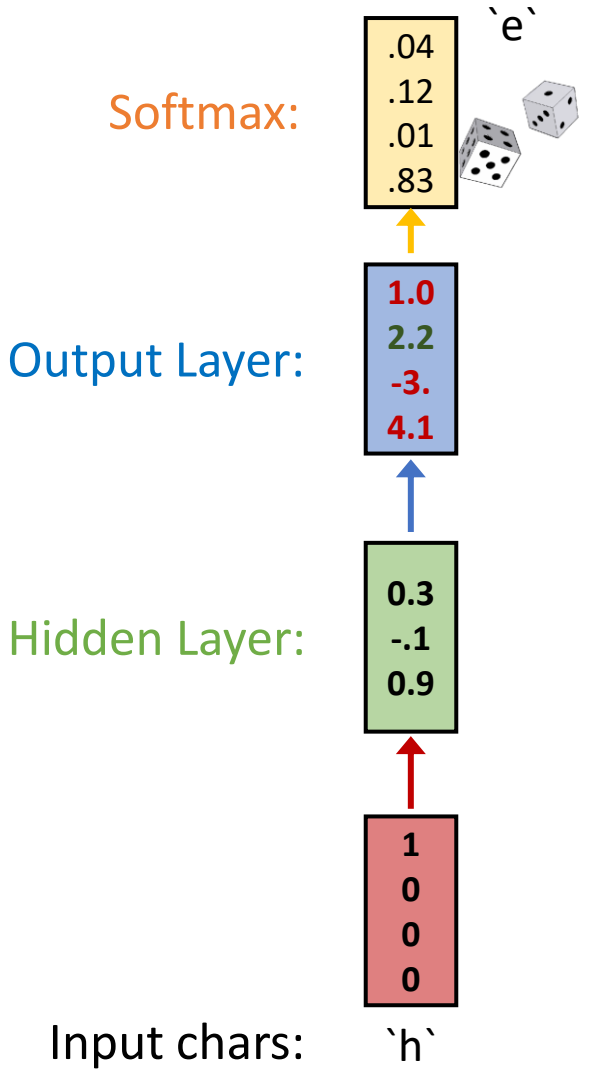


Example: Language Modeling

Task:
Given characters c_0, c_1, \dots, c_{t-1}
Predict c_t

At test time: **generate** new text
Sample one char at a time

Training sequence: "hello"
Vocabulary: ['h', 'e', 'l', 'o']

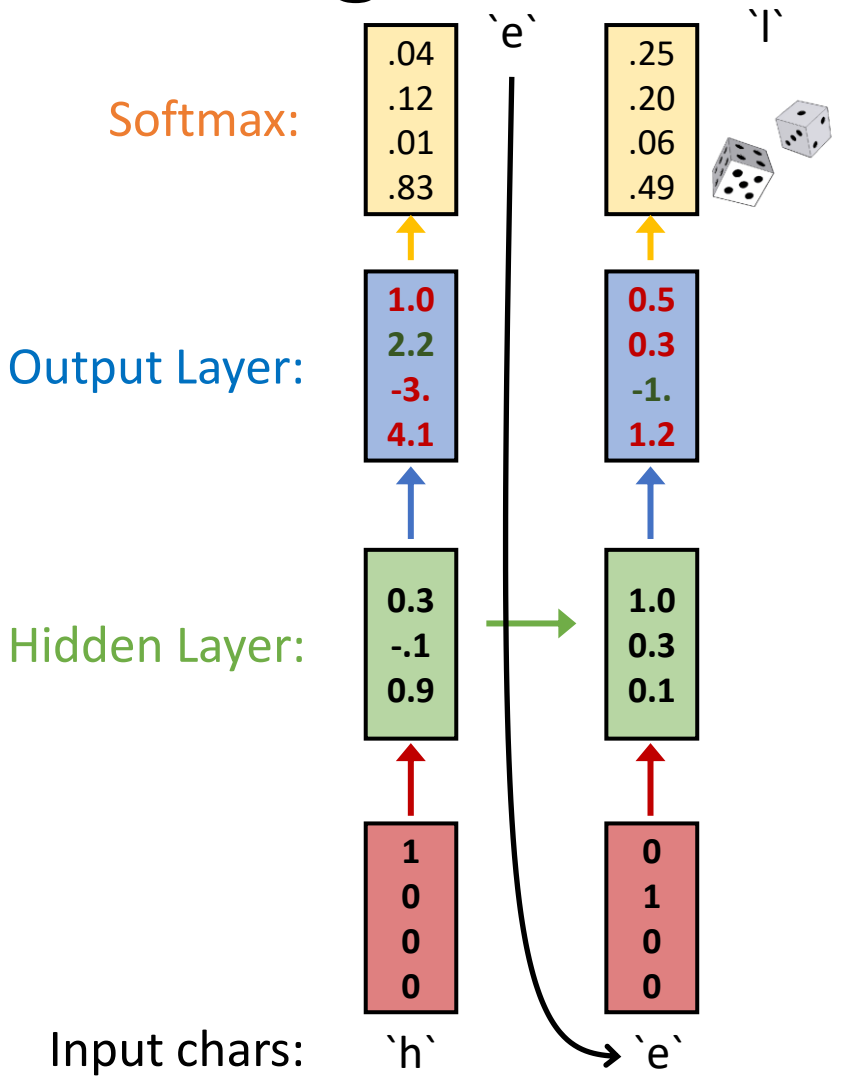


Example: Language Modeling

Task:
 Given characters c_0, c_1, \dots, c_{t-1}
 Predict c_t

At test time: **generate** new text
 Sample one char at a time

Training sequence: "hello"
 Vocabulary: ['h', 'e', 'l', 'o']

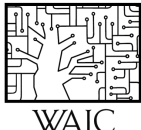
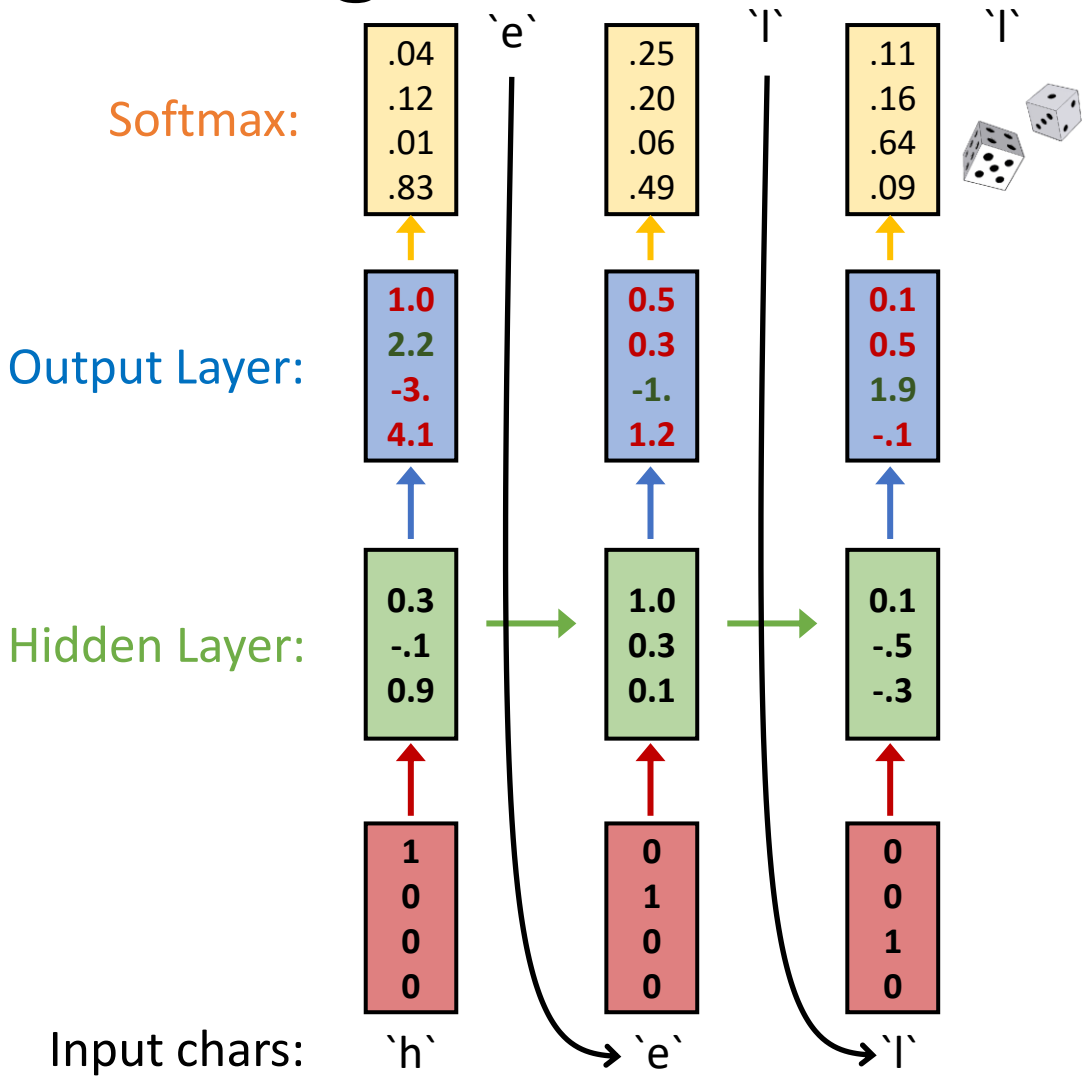


Example: Language Modeling

Task:
 Given characters c_0, c_1, \dots, c_{t-1}
 Predict c_t

At test time: **generate** new text
 Sample one char at a time

Training sequence: "hello"
 Vocabulary: ['h', 'e', 'l', 'o']

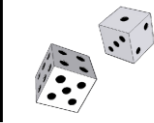
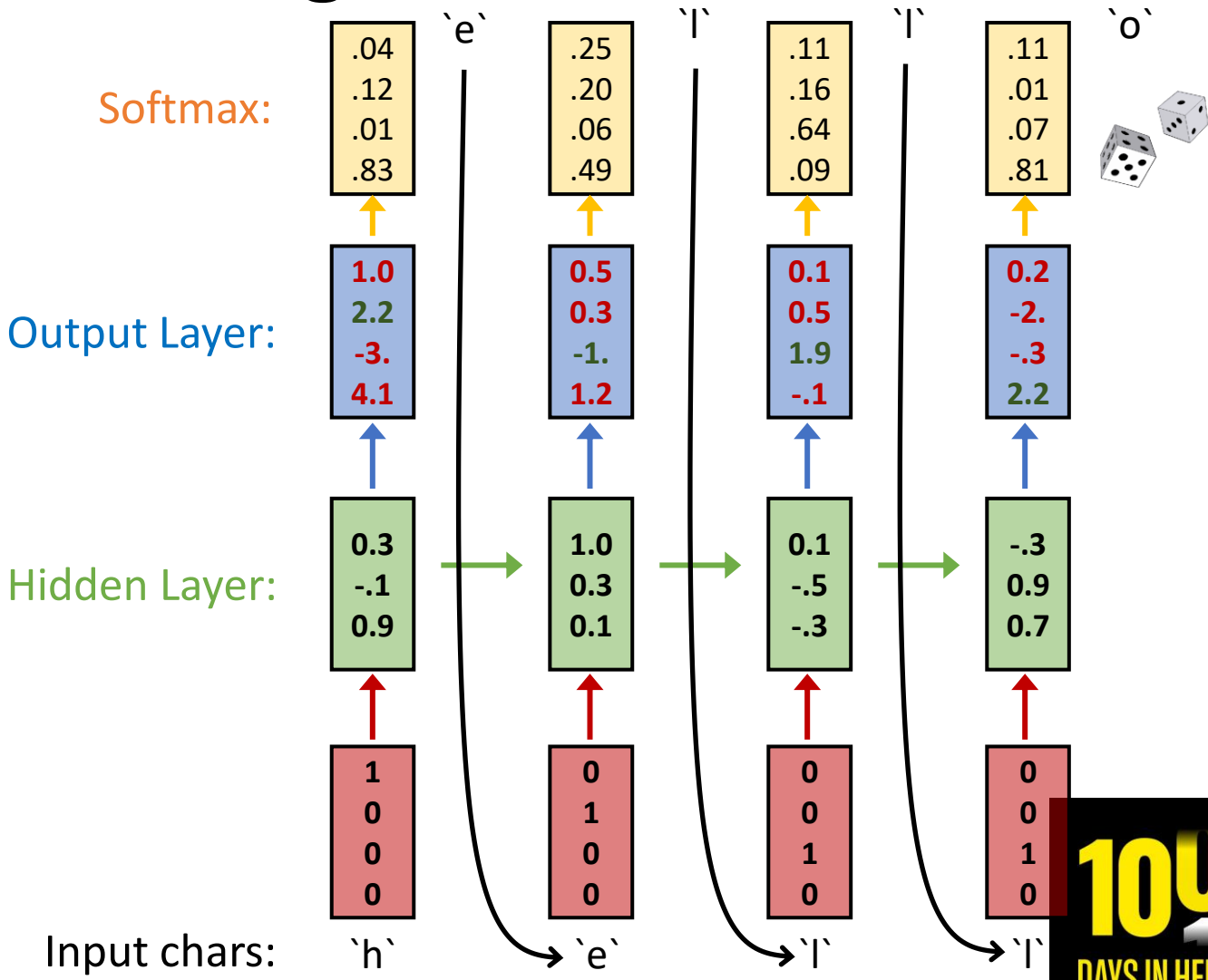


Example: Language Modeling

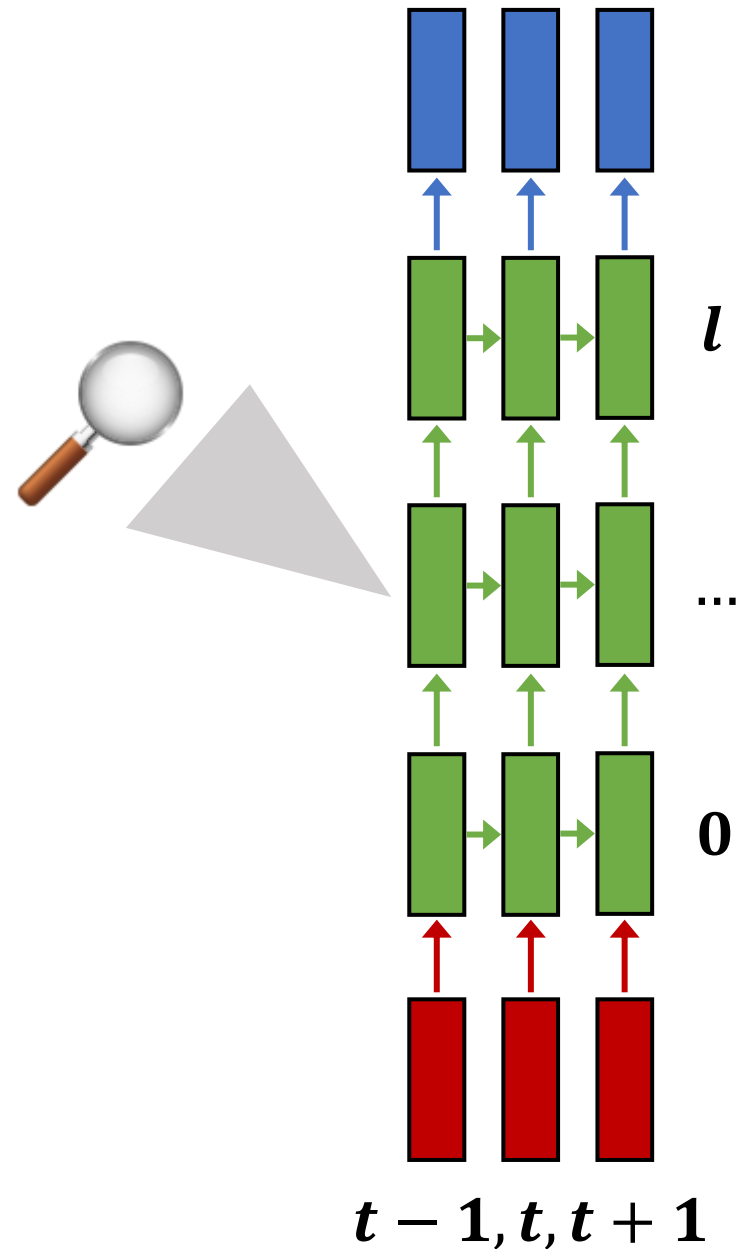
Task:
 Given characters c_0, c_1, \dots, c_{t-1}
 Predict c_t

At test time: **generate** new text
 Sample one char at a time

Training sequence: "hello"
 Vocabulary: ['h', 'e', 'l', 'o']



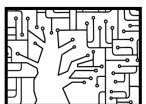
Searching for Interpretable Cells



Searching for Interpretable Cells

A large portion of cells are not easily interpretable. Here is a typical example:

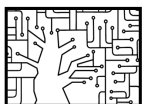
```
/* Unpack a filter field's string representation from user-space
 * buffer. */
char *audit_unpack_string(void **bufp, size_t *remain, size_t len)
{
    char *str;
    if (!*bufp || (len == 0) || (len > *remain))
        return ERR_PTR(-EINVAL);
    /* Of the currently implemented string fields, PATH_MAX
     * defines the longest valid length.
     */
}
```



Searching for Interpretable Cells

Cell sensitive to position in line:

The sole importance of the crossing of the Berezina lies in the fact that it plainly and indubitably proved the fallacy of all the plans for cutting off the enemy's retreat and the soundness of the only possible line of action--the one Kutuzov and the general mass of the army demanded--namely, simply to follow the enemy up. The French crowd fled at a continually increasing speed and all its energy was directed to reaching its goal. It fled like a wounded animal and it was impossible to block its path. This was shown not so much by the arrangements it made for crossing as by what took place at the bridges. When the bridges broke down, unarmed soldiers, people from Moscow and women with children who were with the French transport, all--carried on by vis inertiae--pressed forward into boats and into the ice-covered water and did not, surrender.

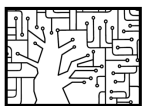


Searching for Interpretable Cells

Cell that turns on inside quotes:

"You mean to imply that I have nothing to eat out of.... On the contrary, I can supply you with everything even if you want to give dinner parties," warmly replied Chichagov, who tried by every word he spoke to prove his own rectitude and therefore imagined Kutuzov to be animated by the same desire.

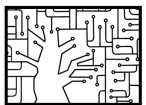
Kutuzov, shrugging his shoulders, replied with his subtle penetrating smile: "I meant merely to say what I said."



Searching for Interpretable Cells

Cell that robustly activates inside if statements:

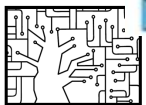
```
static int __dequeue_signal(struct sigpending *pending, sigset_t *mask,
    siginfo_t *info)
{
    int sig = next_signal(pending, mask);
    if (sig) {
        if (current->notifier) {
            if (sigismember(current->notifier_mask, sig)) {
                if (!(current->notifier)(current->notifier_data)) {
                    clear_thread_flag(TIF_SIGPENDING);
                    return 0;
                }
            }
        }
        collect_signal(sig, pending, info);
    }
    return sig;
}
```



Searching for Interpretable Cells

Cell that turns on inside comments and quotes:

```
/* Duplicate LSM field information. The lsm_rule is opaque, so
 * re-initialized. */
static inline int audit_dupe_lsm_field(struct audit_field *df,
                                     struct audit_field *sf)
{
    int ret = 0;
    char *lsm_str;
    /* our own copy of lsm_str */
    lsm_str = kstrdup(sf->lsm_str, GFP_KERNEL);
    if (unlikely(!lsm_str))
        return -ENOMEM;
    df->lsm_str = lsm_str;
    /* our own (refreshed) copy of lsm_rule */
    ret = security_audit_rule_init(df->type, df->op, df->lsm_str,
                                  (void **)&df->lsm_rule);
    /* Keep currently invalid fields around in case they
     * become valid after a policy reload. */
    if (ret == -EINVAL) {
        pr_warn("audit rule for LSM \'%s\' is invalid\n",
              df->lsm_str);
        ret = 0;
    }
    return ret;
}
```

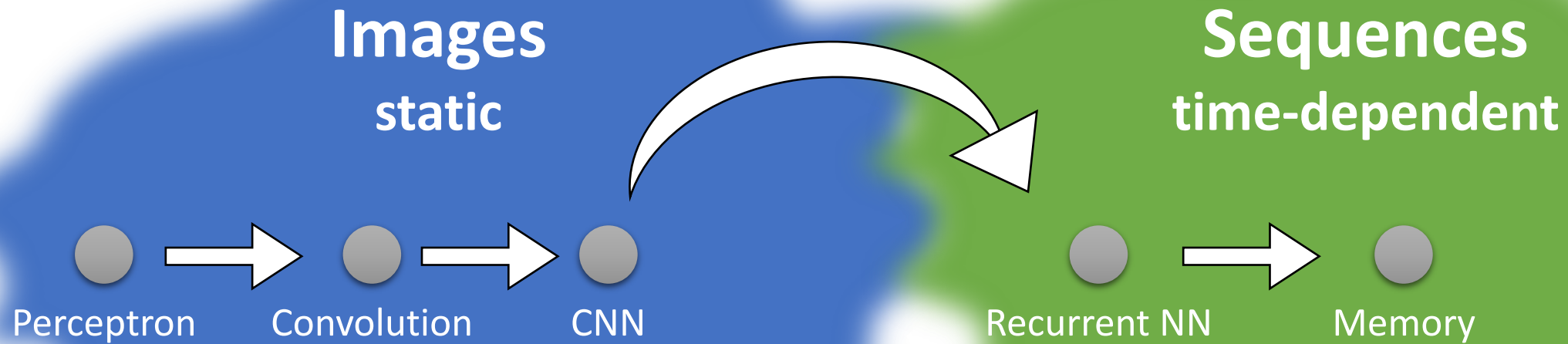


Searching for Interpretable Cells

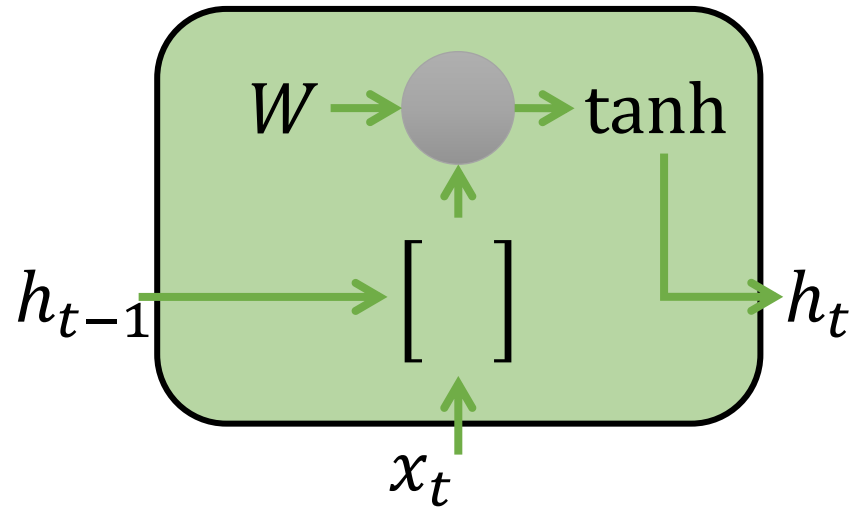
Cell that is sensitive to the depth of an expression:

```
#ifdef CONFIG_AUDITSYSCALL
static inline int audit_match_class_bits(int class, u32 *mask)
{
    int i;
    if (classes[class]) {
        for (i = 0; i < AUDIT_BITMASK_SIZE; i++)
            if (mask[i] & classes[class][i])
                return 0;
    }
    return 1;
}
```

Agenda



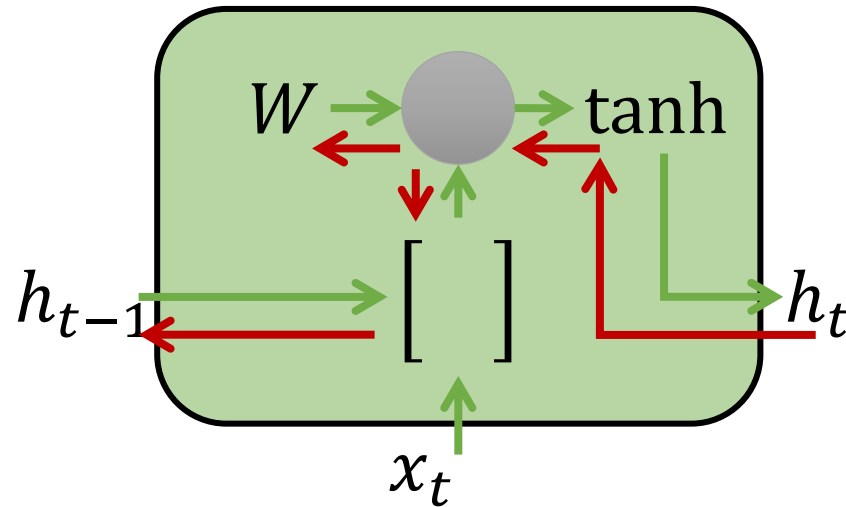
RNN: Gradient Flow



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

$$= \tanh\left(W \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}\right)$$

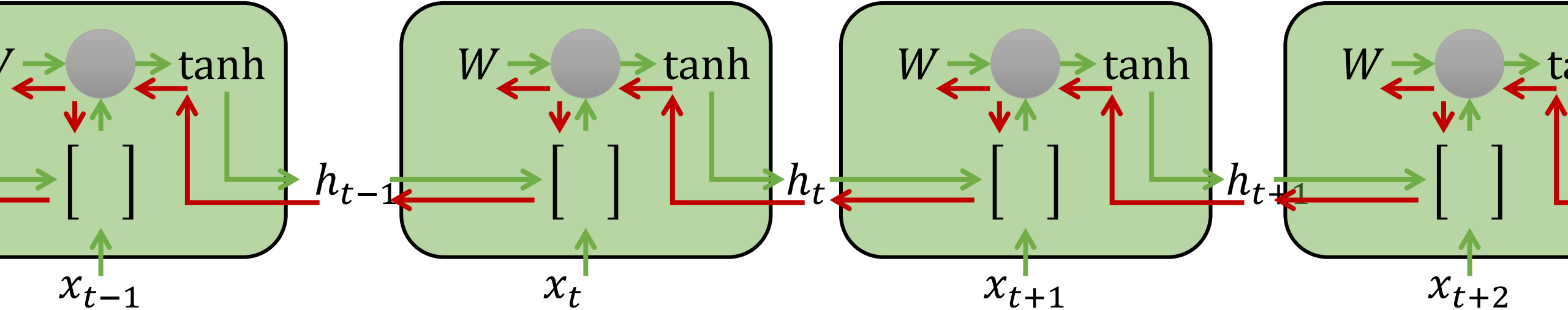
RNN: Gradient Flow



$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

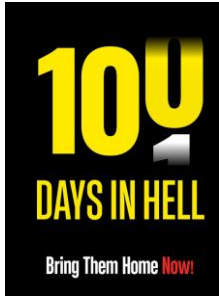
$$= \tanh\left(W \cdot \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix}\right)$$

RNN: Gradient Flow

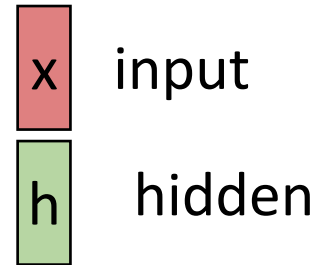
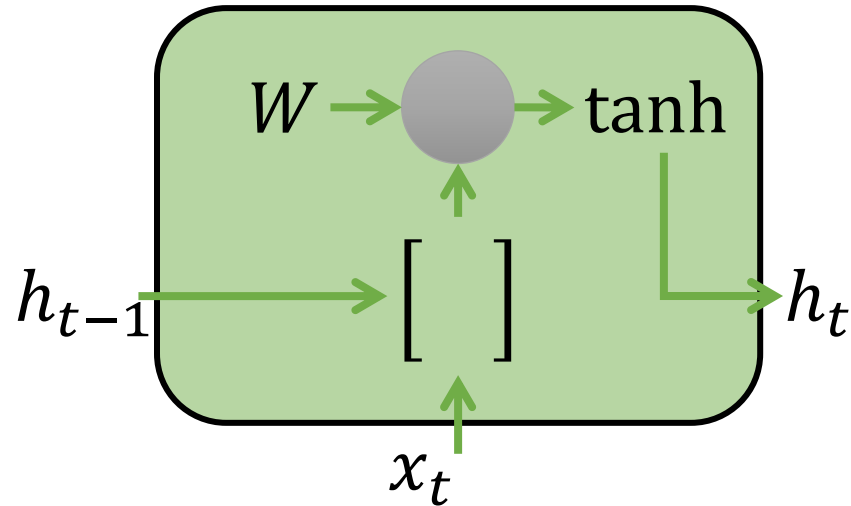
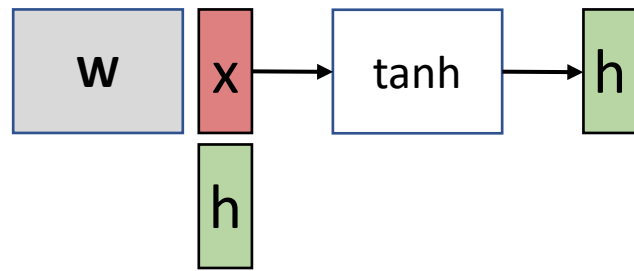


Forwarding/backwarding in time for long sequences = very deep NN

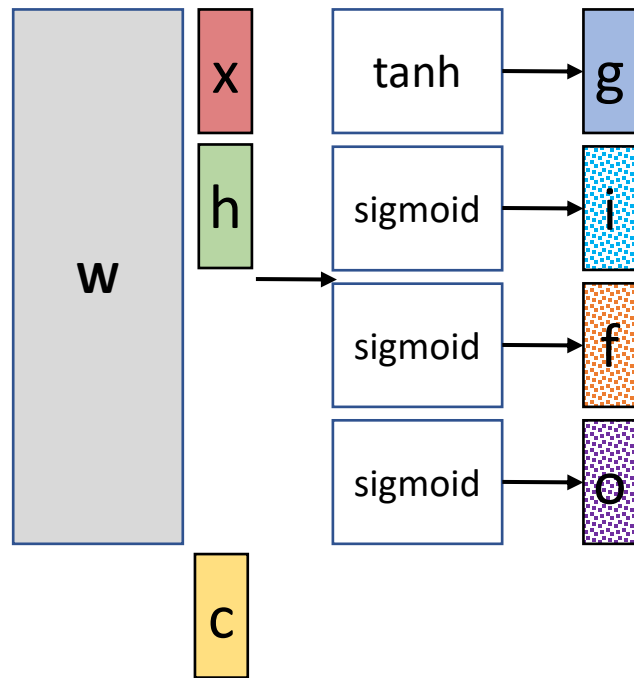
Easily leads to exploding/vanishing gradients



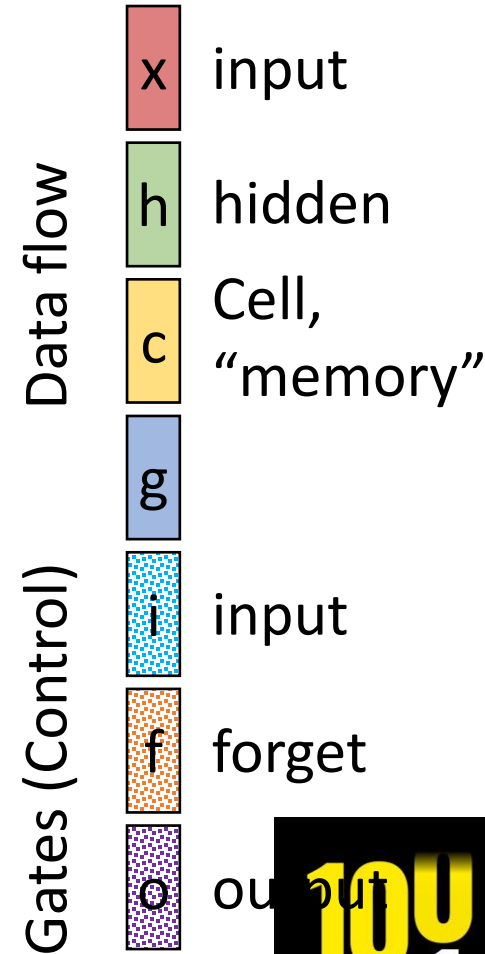
Long Short-Term Memory (LSTM)



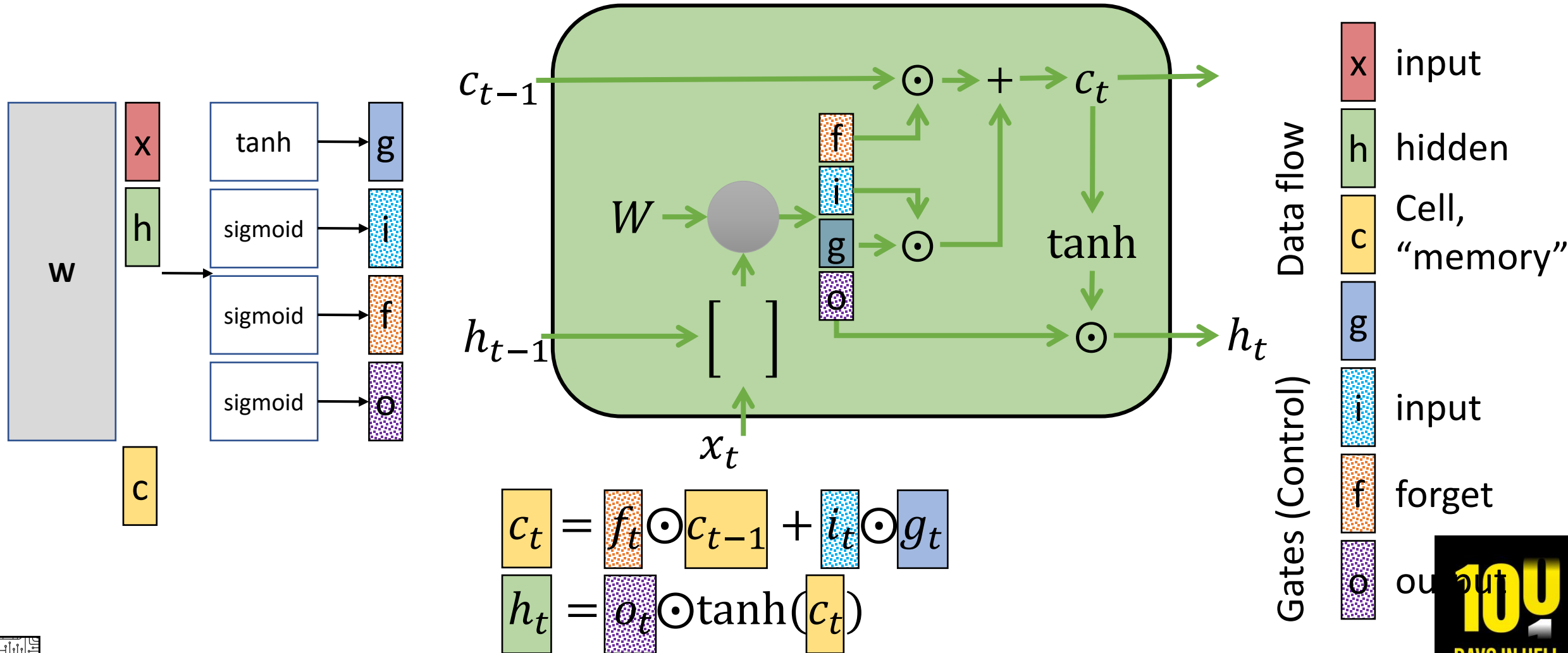
Long Short-Term Memory (LSTM)



$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$

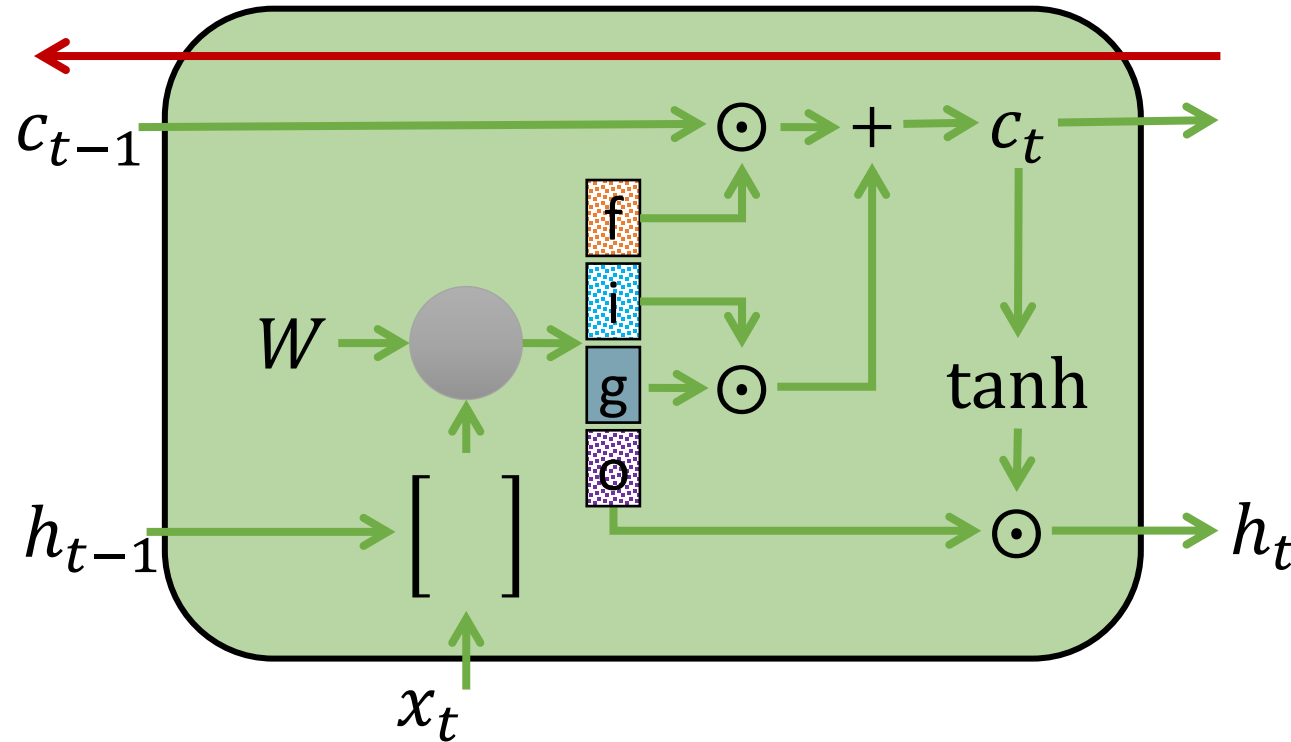


Long Short-Term Memory (LSTM)



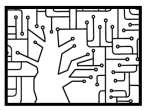
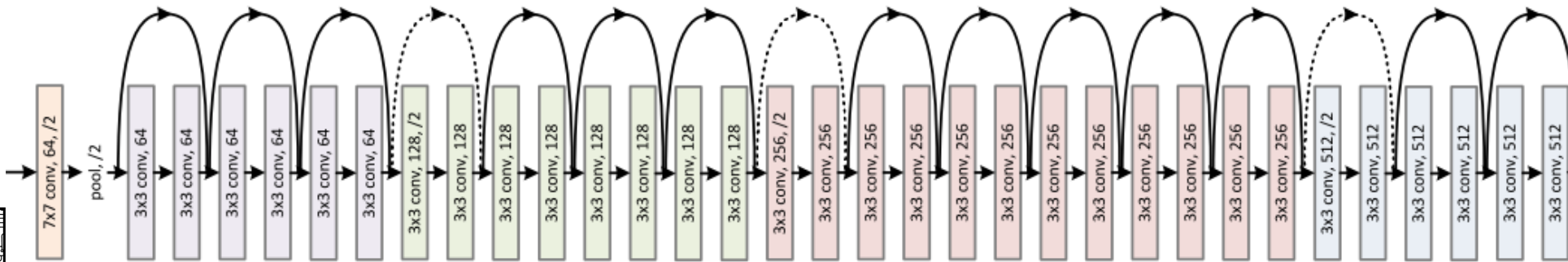
Long Short-Term Memory (LSTM)

A gradient path
w/o matrix
multiplication

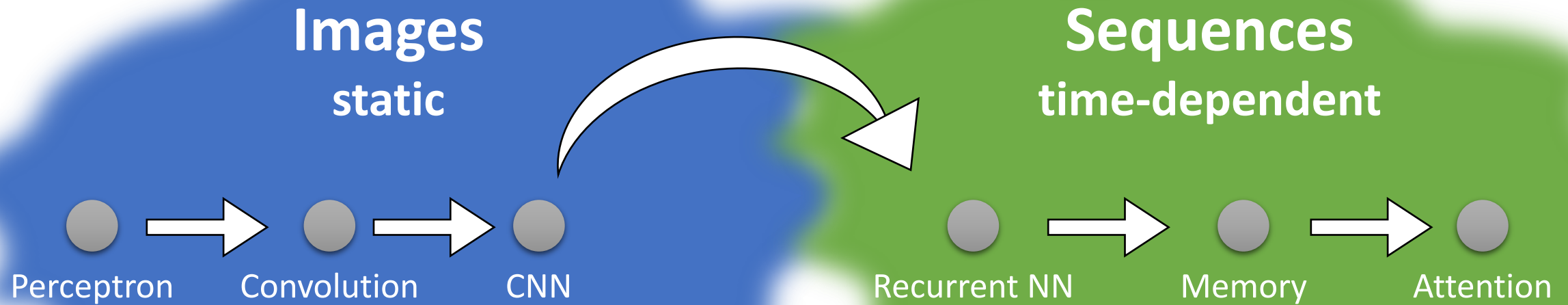


Data flow
Gates (Control)

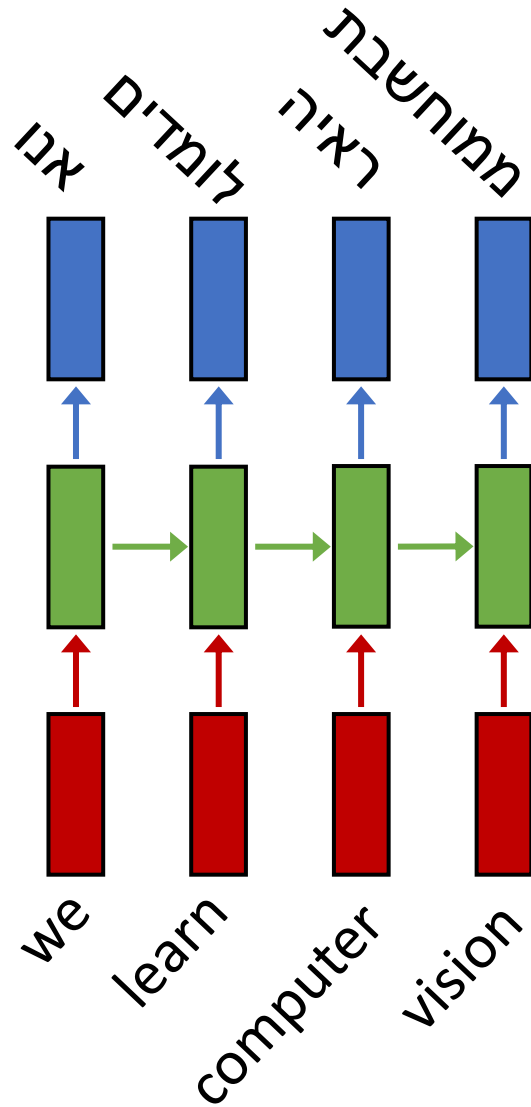
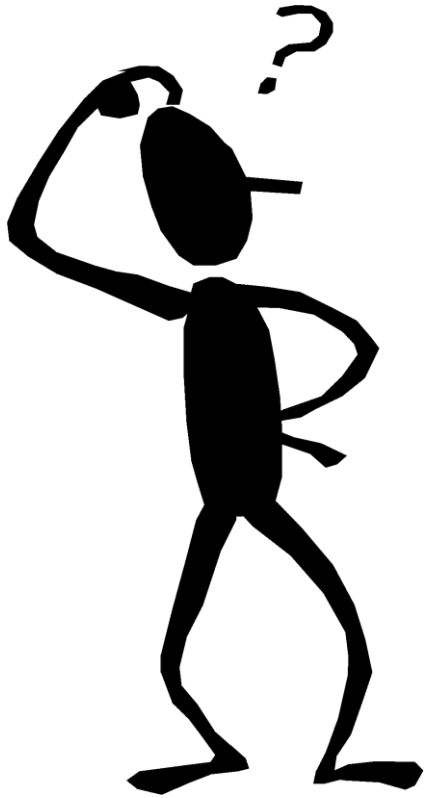
- x input
- h hidden
- c Cell, "memory"
- g gate
- i input
- f forget
- o output



Agenda



Sequence to Sequence



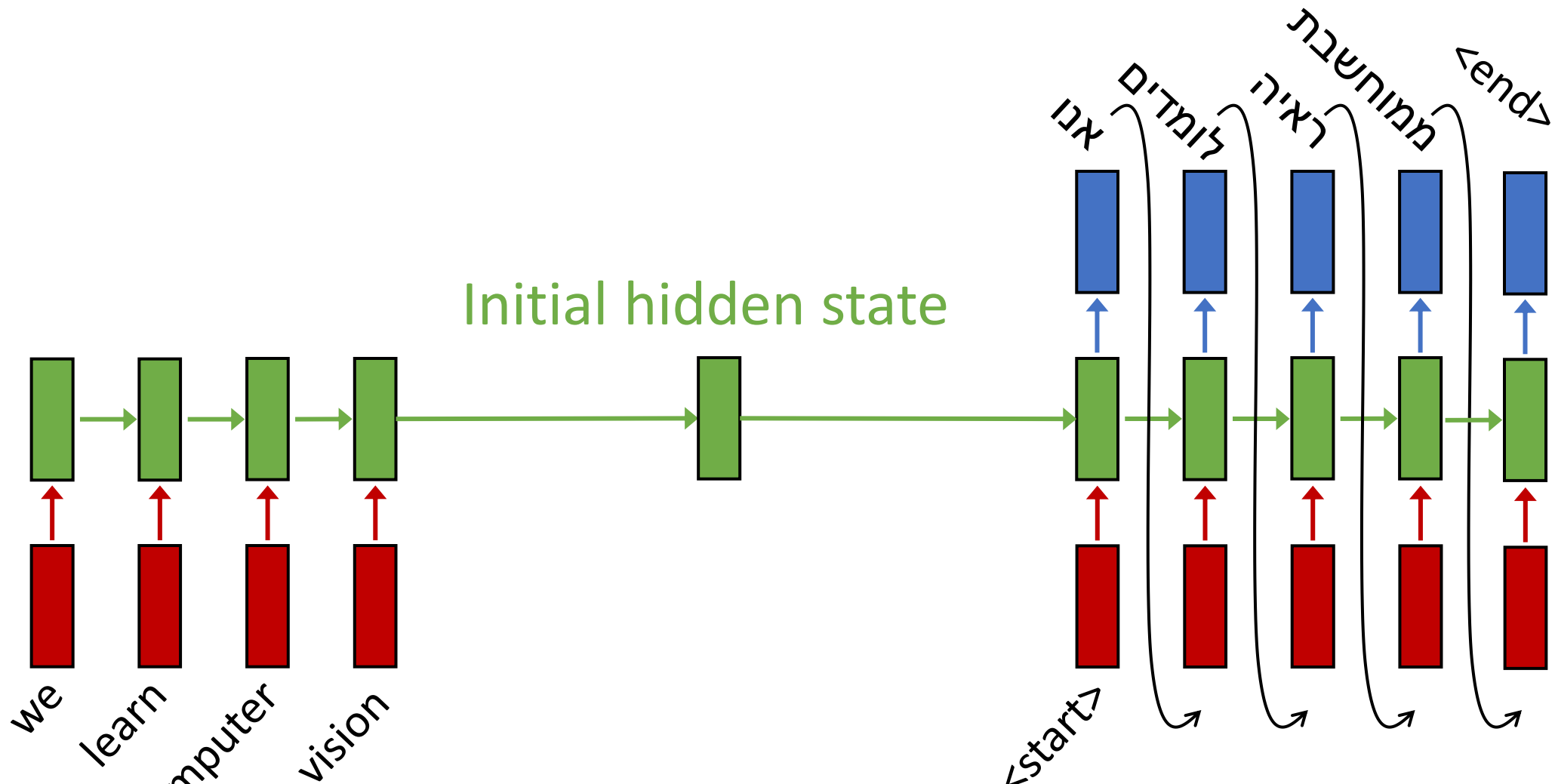
We
Learn
Computer
Vision

אנו (ANU)
לומדים (LOMDIM)
ממוחשבת (MEMUCHSHEVET)
ראיה (RE'EYA)

Sequence to Sequence: RNN

Encoder RNN

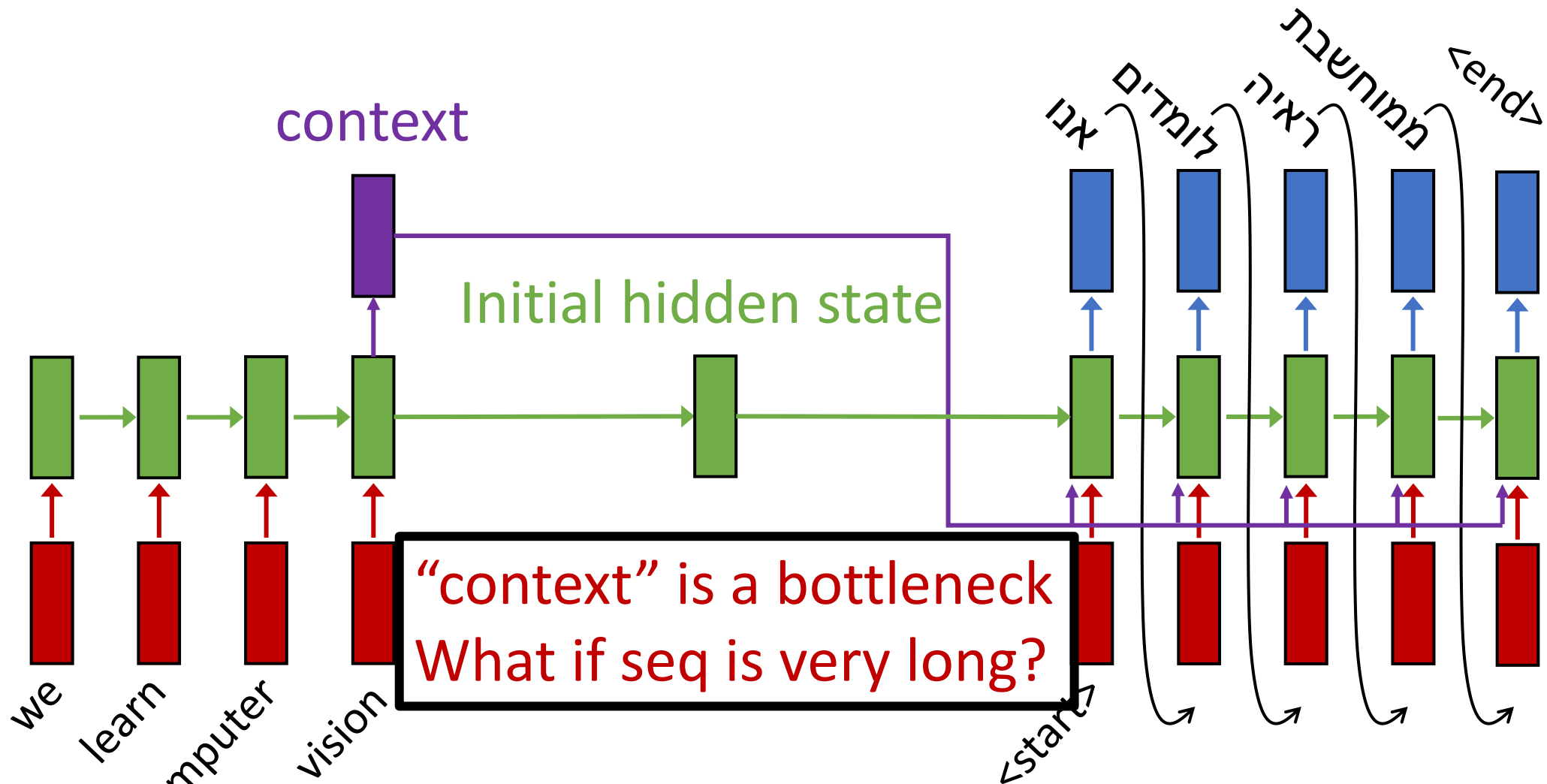
Decoder RNN



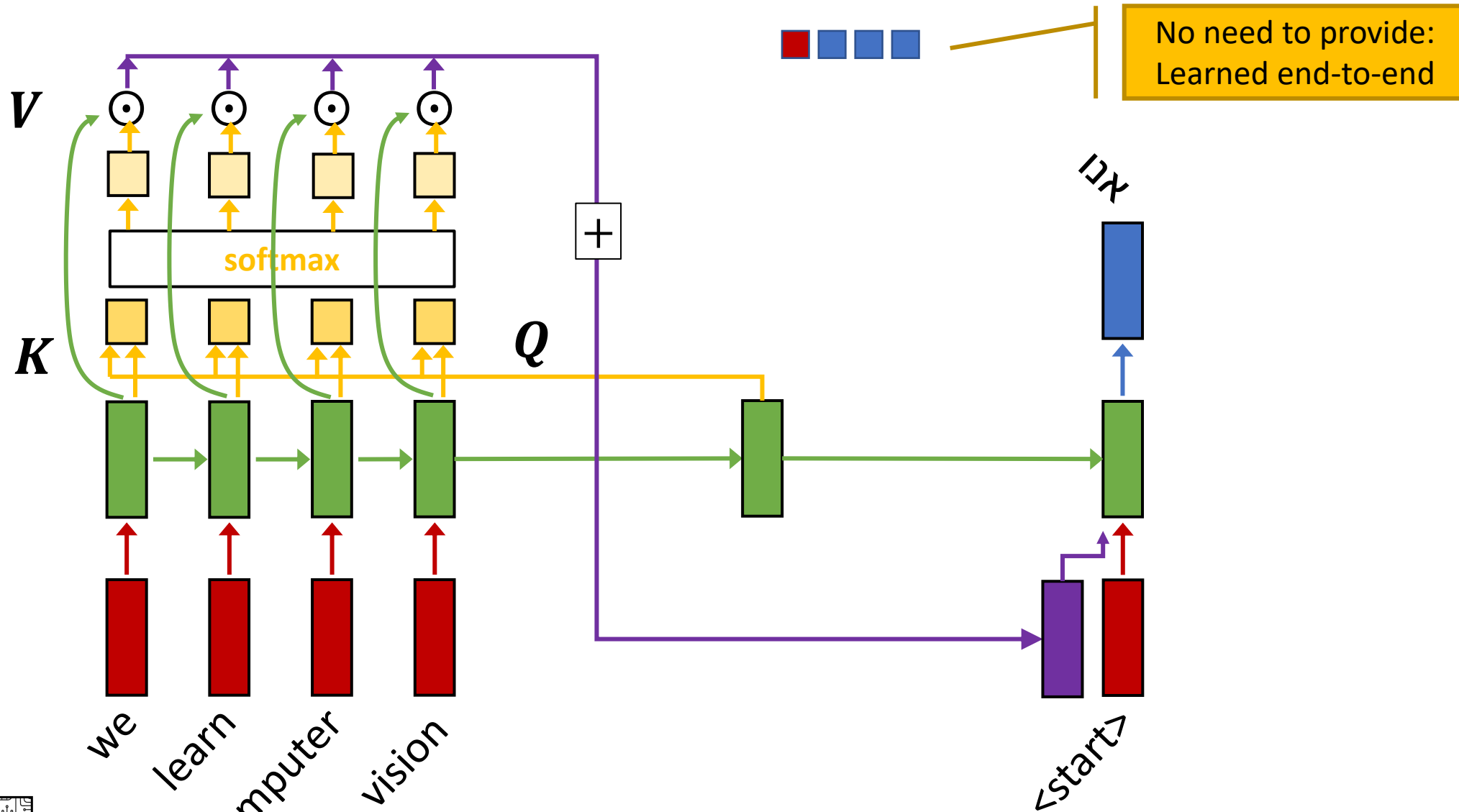
Sequence to Sequence: RNN

Encoder RNN

Decoder RNN

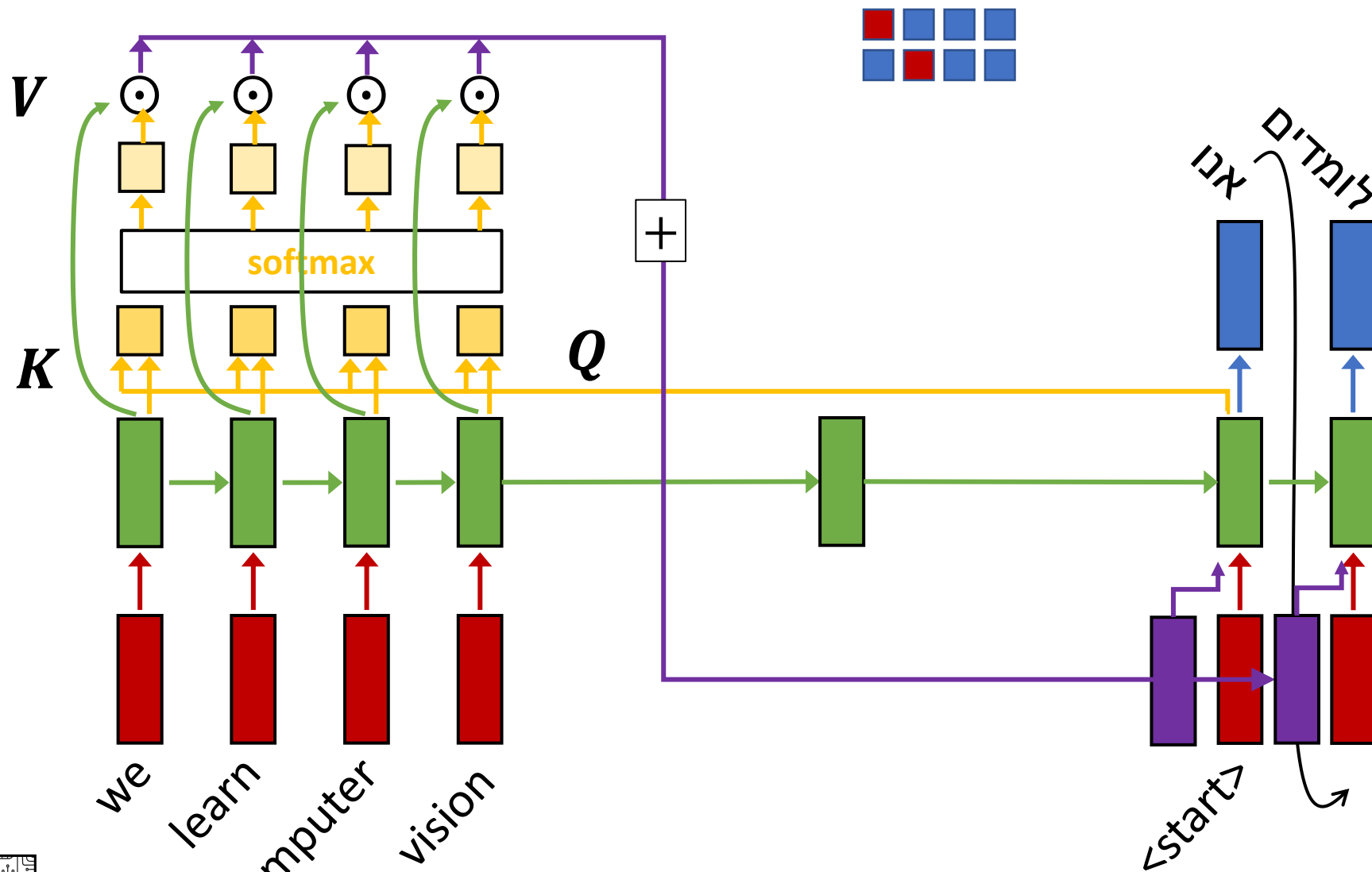


Sequence to Sequence: RNN & Attention



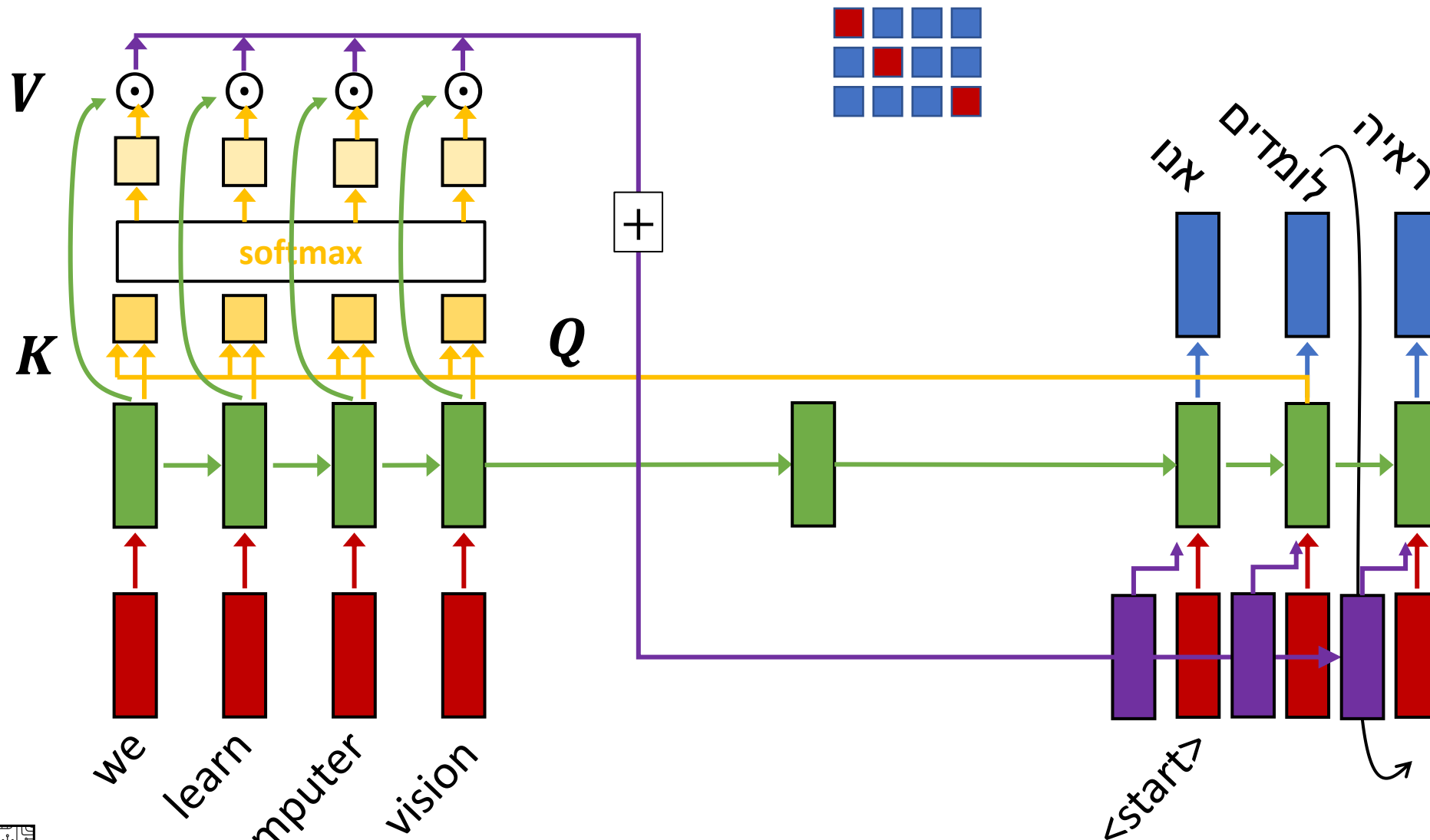
No need to provide:
Learned end-to-end

Sequence to Sequence: RNN & Attention

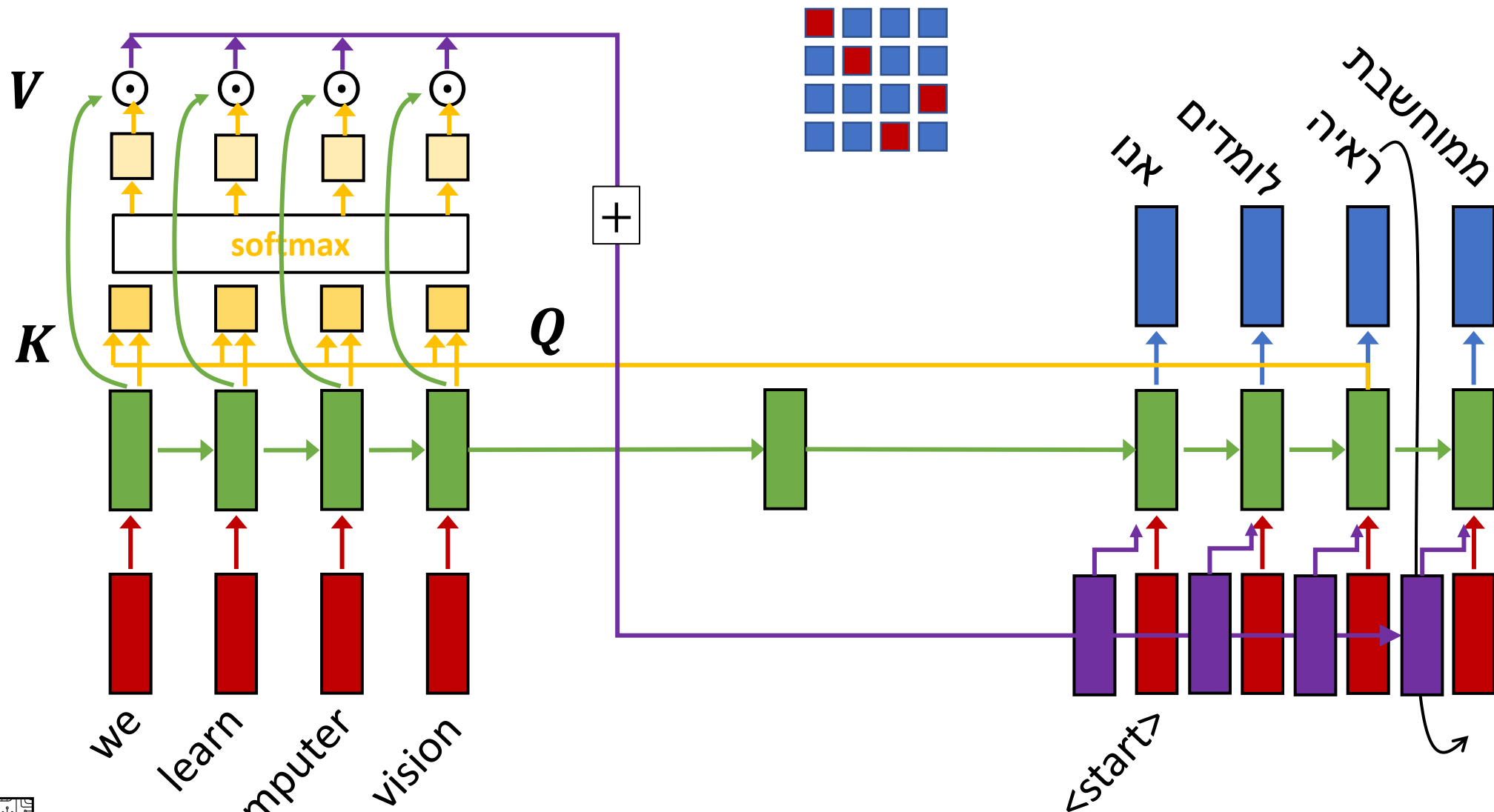


Slide credit: [Justin Johnson \(EECS-498-007, UMich\)](#)

Sequence to Sequence: RNN & Attention



Sequence to Sequence: RNN & Attention



Sequence to Sequence: RNN & Attention

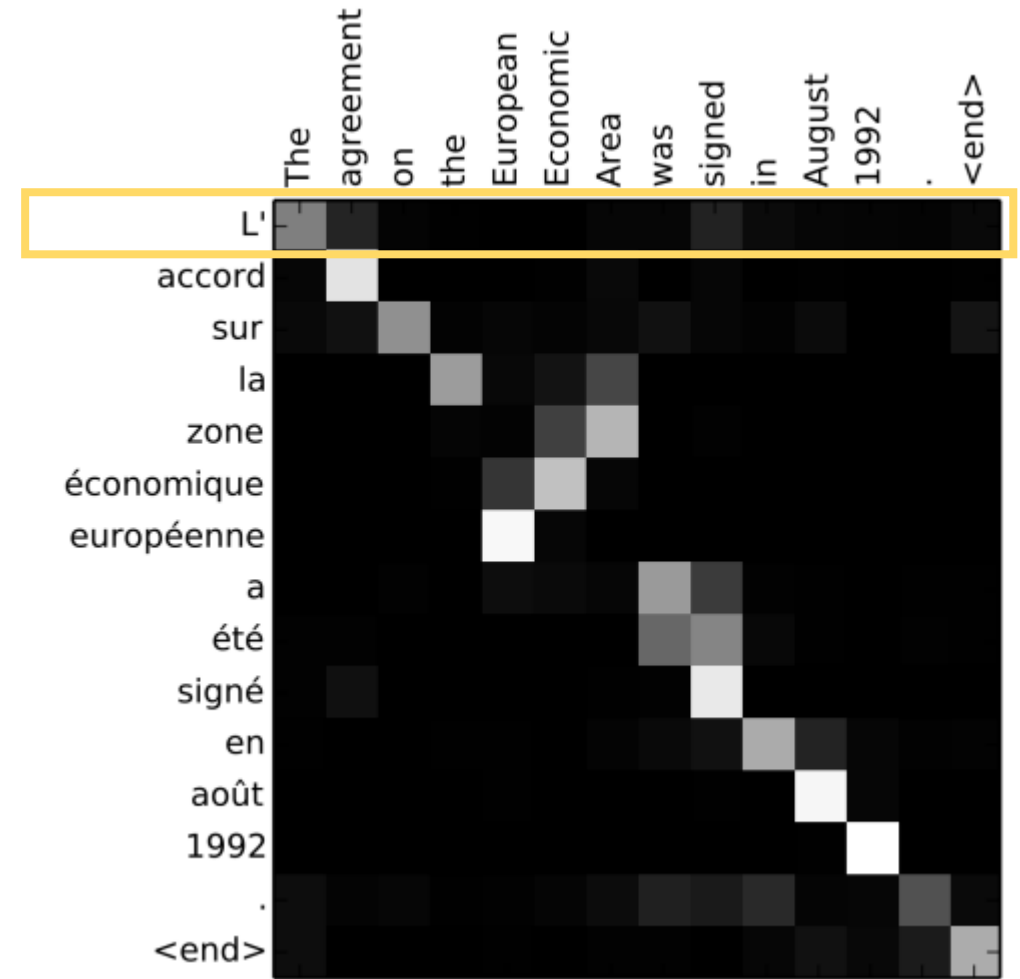
Example: English to French translation

Input (English):

The agreement on the European Economic Area was signed in August 1992.

Output (French):

L'accord sur la zone économique européenne a été signé en août 1992.



Bahdanau et al, "Neural machine translation by jointly learning to align and translate",

Slide credit: [Justin Johnson \(EECS-498-007, UMich\)](#)

Sequence to Sequence: RNN & Attention

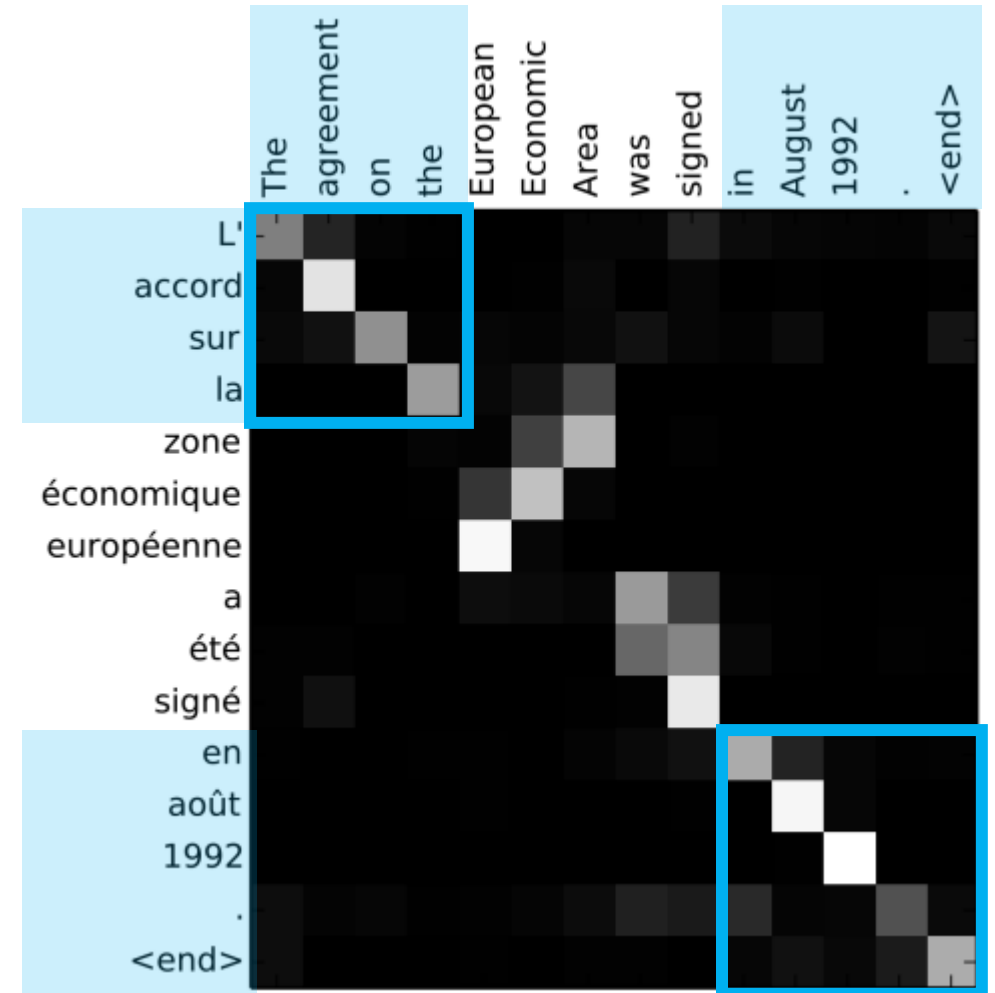
Example: English to French translation

Input (English):

The agreement on the European Economic Area was signed in August 1992.

Output (French):

L'accord sur la zone économique européenne a été signé en août 1992.



Sequence to Sequence: RNN & Attention

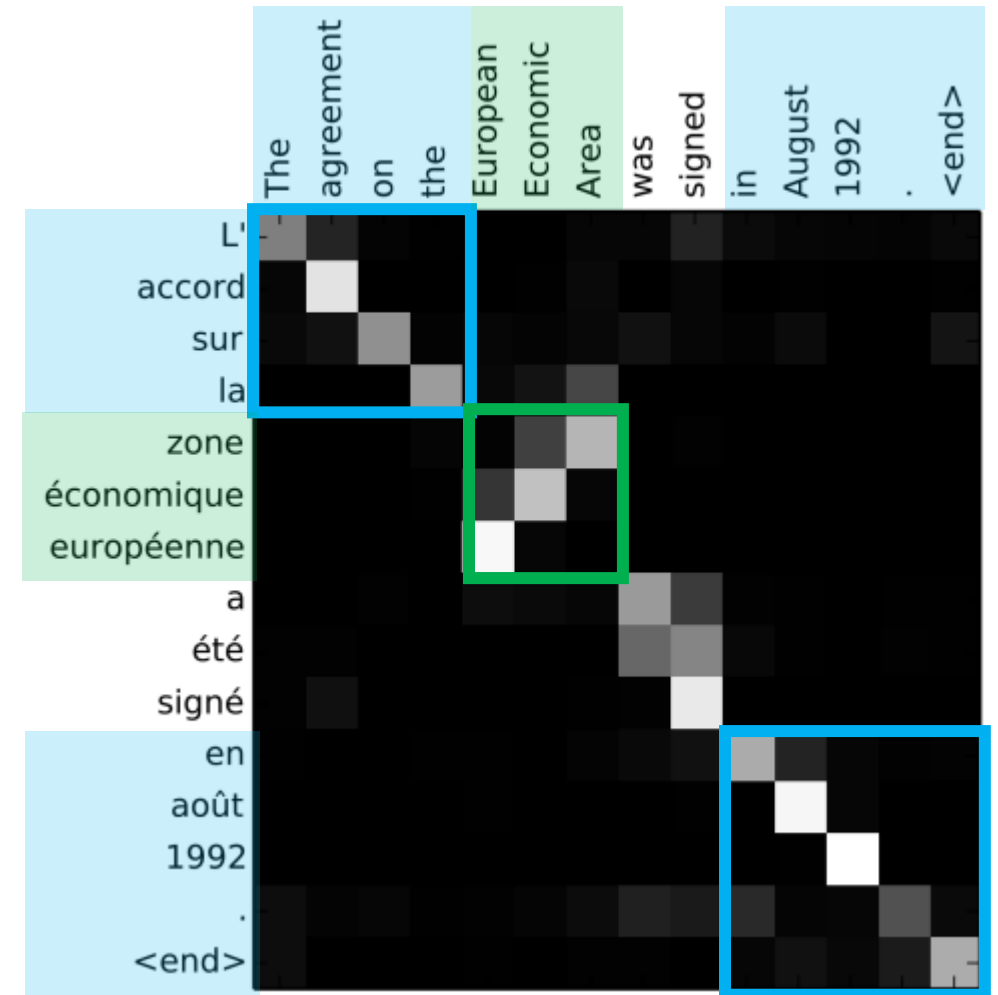
Example: English to French translation

Input (English):

The agreement on the **European Economic Area** was signed in August 1992.

Output (French):

L'accord sur la **zone économique européenne** a été signé en août 1992.



Sequence to Sequence: RNN & Attention

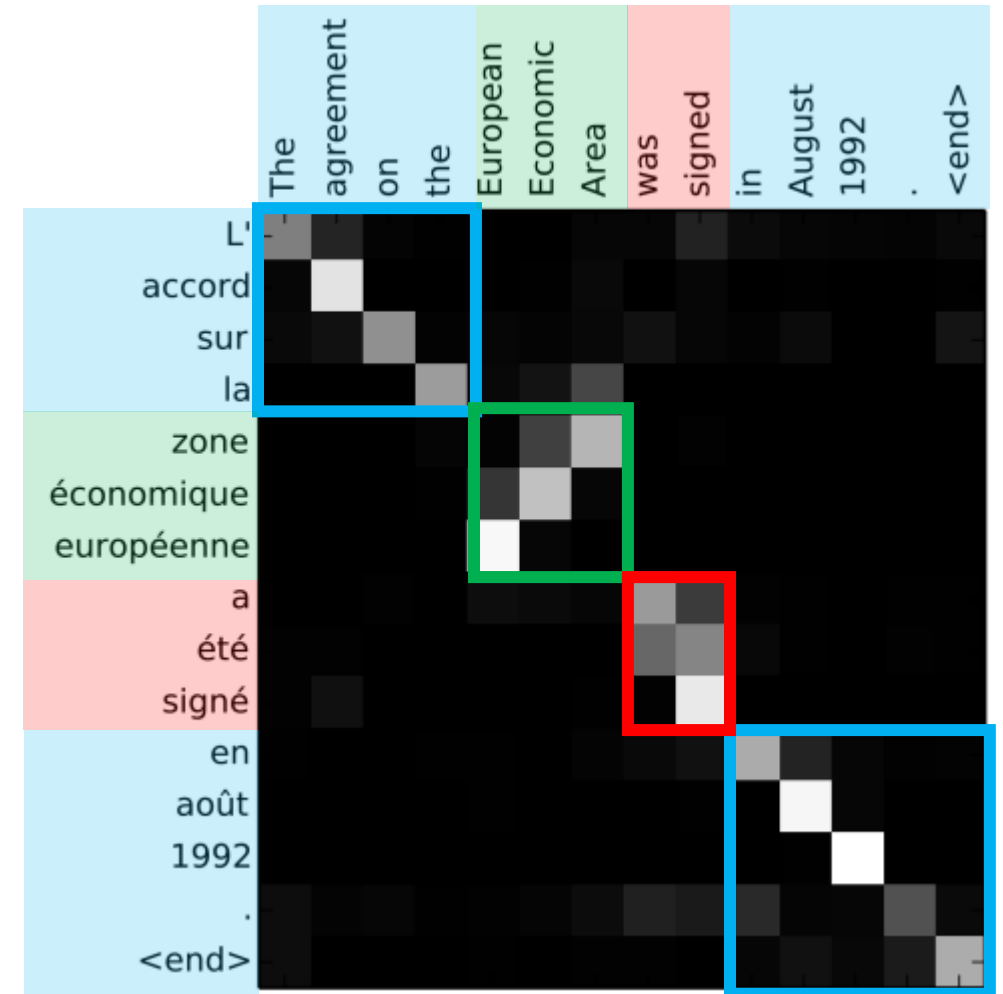
Example: English to French translation

Input (English):

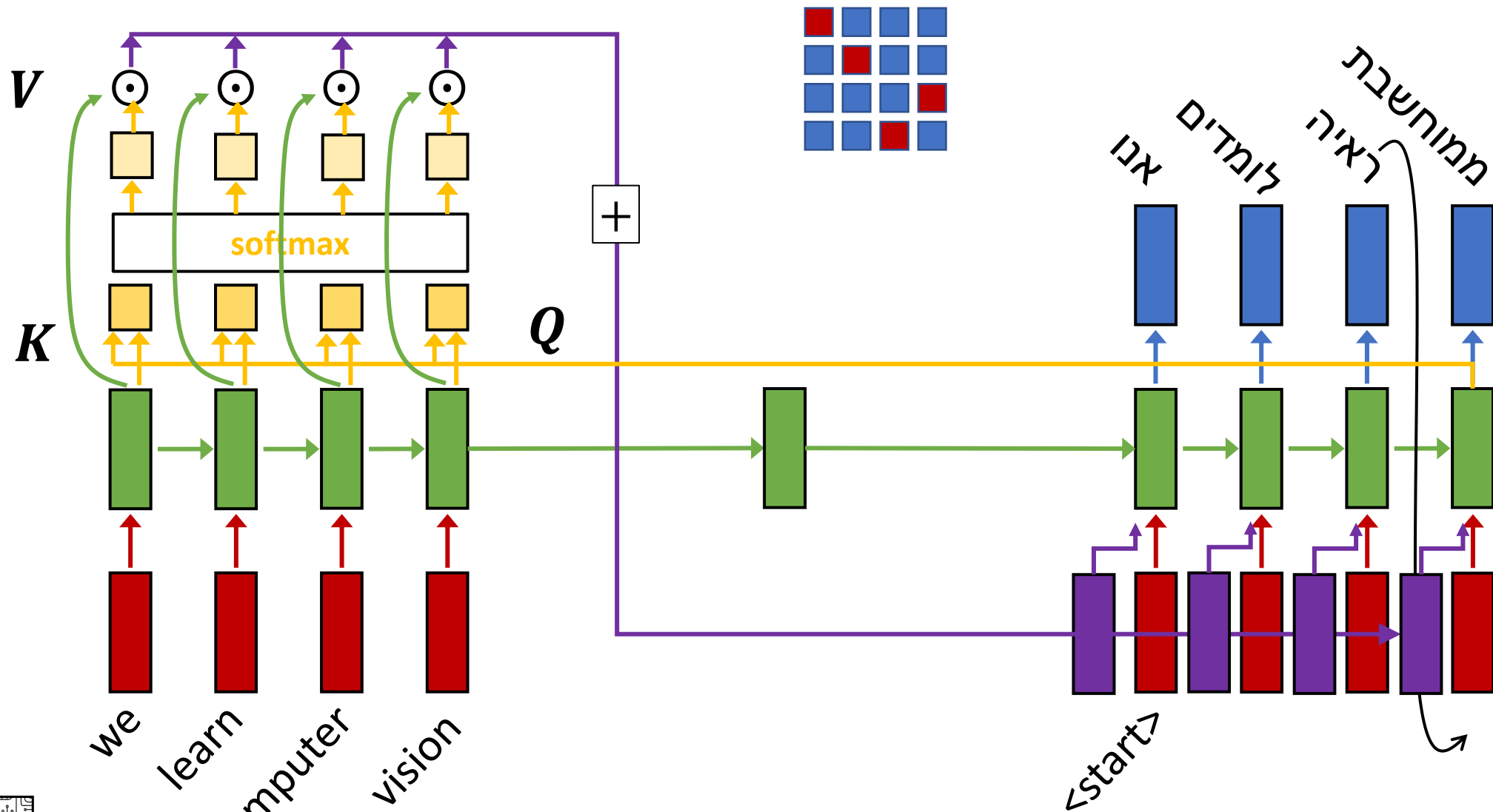
The agreement on the European Economic Area
was signed in August 1992.

Output (French):

L'accord sur la zone économique européenne
a été signé en août 1992.



Attention Layer



Attention Layer

Inputs:

Query: Q (shape: $N_q \times D_q$)

Input: X (shape: $N_x \times D_x$)

Layer's Parameters:

$X \rightarrow K$: W_k (shape: $D_x \times D_q$)

$X \rightarrow V$: W_v (shape: $D_x \times D_v$)

Compute:

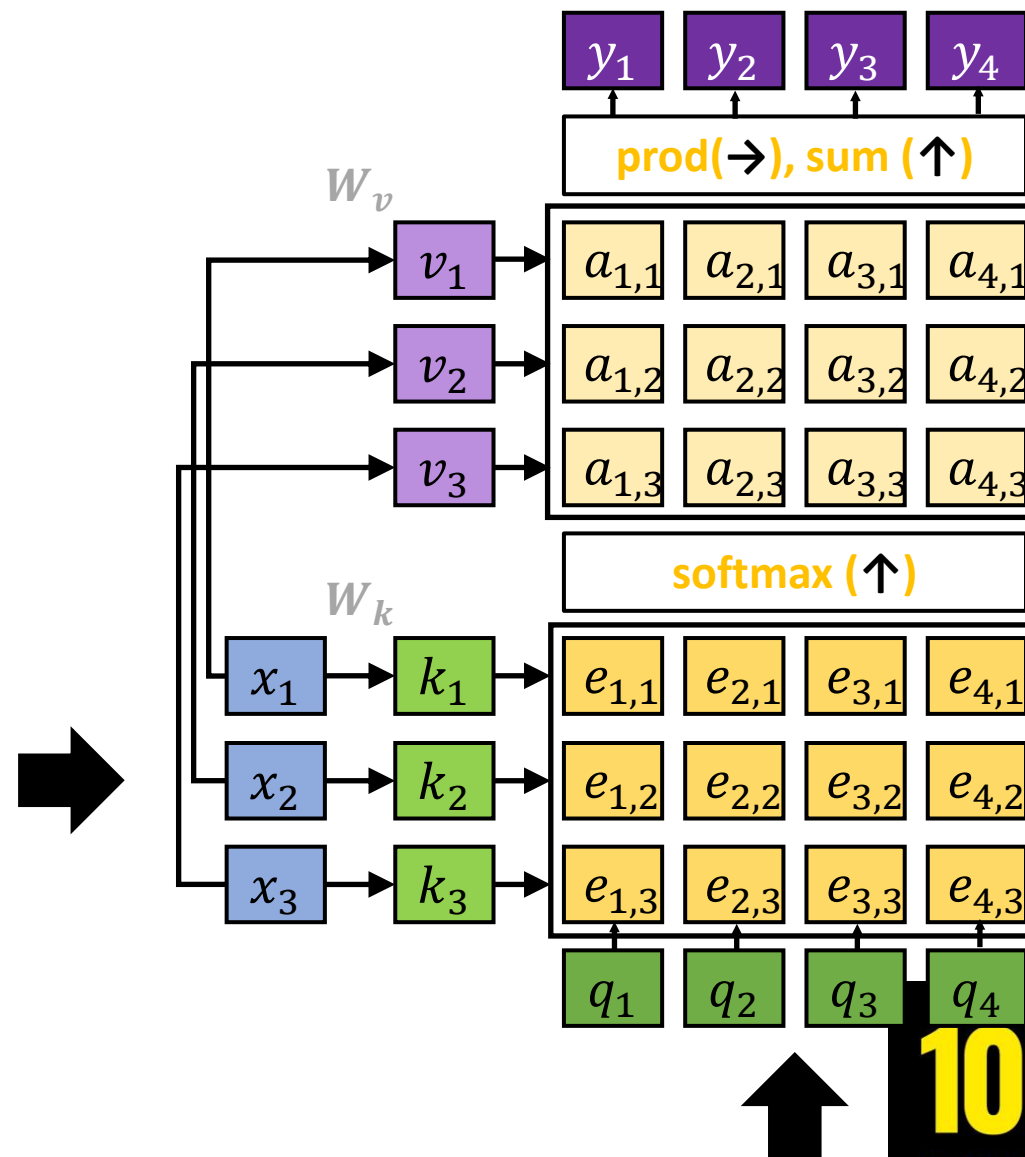
Keys: $K = XW_k$ (shape: $N_x \times D_q$)

Values: $V = XW_v$ (shape: $N_x \times D_v$)

Similarities: $E = QK^T / \sqrt{D_q}$ (shape: $N_q \times N_x$)

Attention: $A = \text{softmax}(E; \uparrow)$ (shape: $N_q \times N_x$)

Outputs: $Y = AV$ (shape: $N_q \times D_v$)



Self-Attention Layer

Input:

Input: X (shape: $N_x \times D_x$)

Layer's Parameters:

$X \rightarrow Q$: W_q (shape: $D_x \times D_q$)

$X \rightarrow K$: W_k (shape: $D_x \times D_q$)

$X \rightarrow V$: W_v (shape: $D_x \times D_v$)

Compute:

Query: $Q = XW_q$ (shape: $N_x \times D_q$)

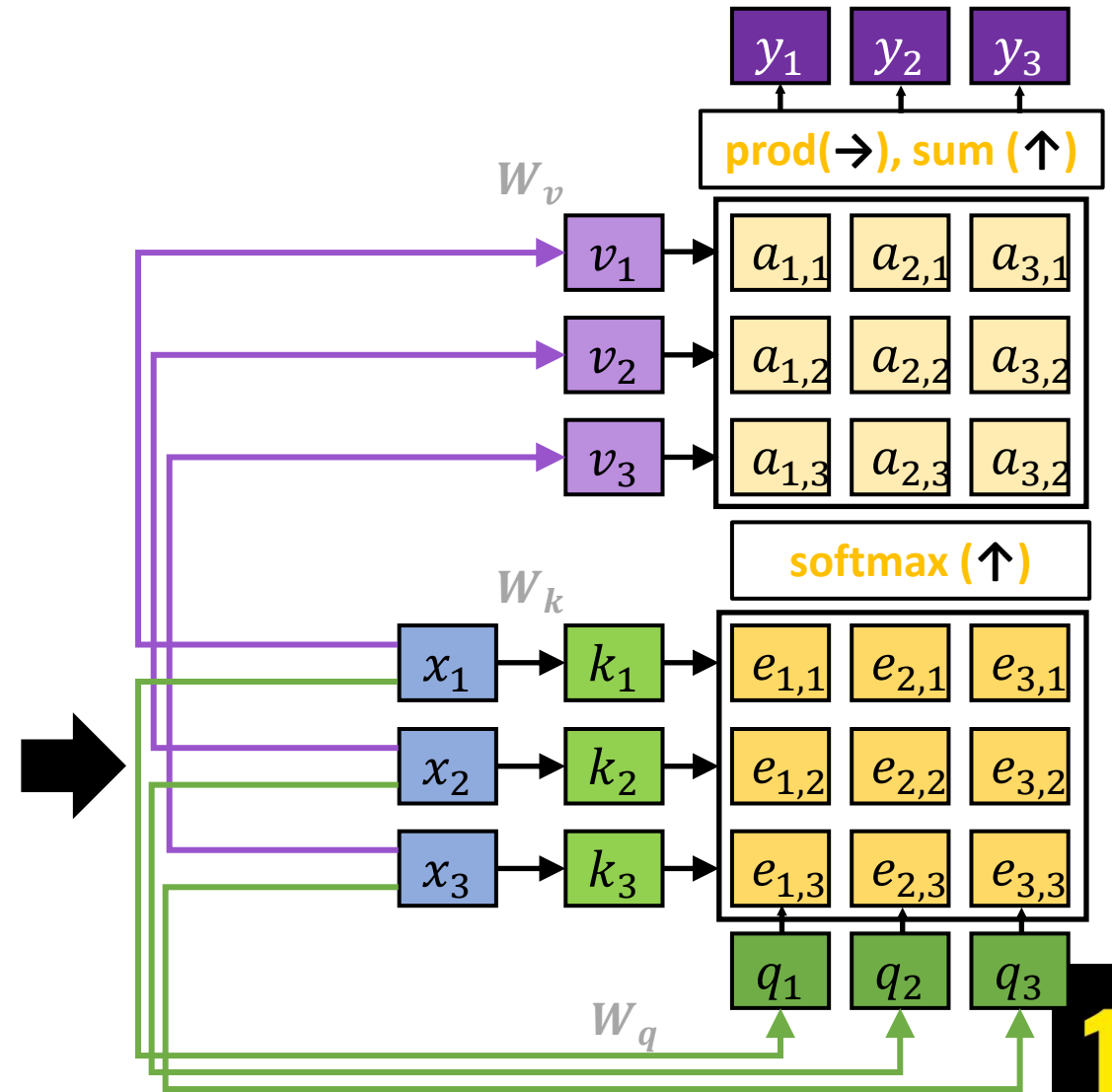
Keys: $K = XW_k$ (shape: $N_x \times D_q$)

Values: $V = XW_v$ (shape: $N_x \times D_v$)

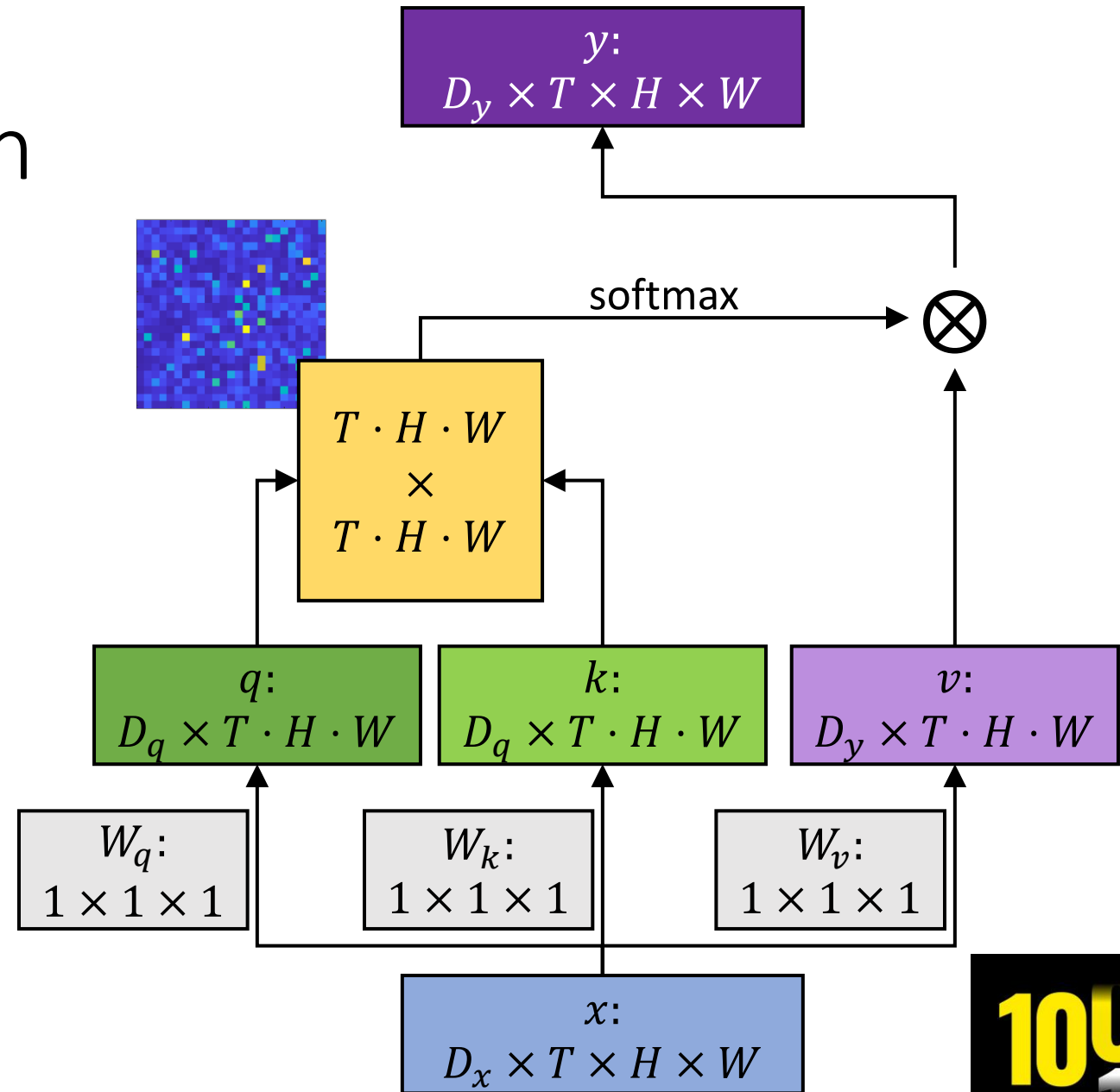
Similarities: $E = QK^T / \sqrt{D_q}$ (shape: $N_x \times N_x$)

Attention: $A = \text{softmax}(E; \uparrow)$ (shape: $N_x \times N_x$)

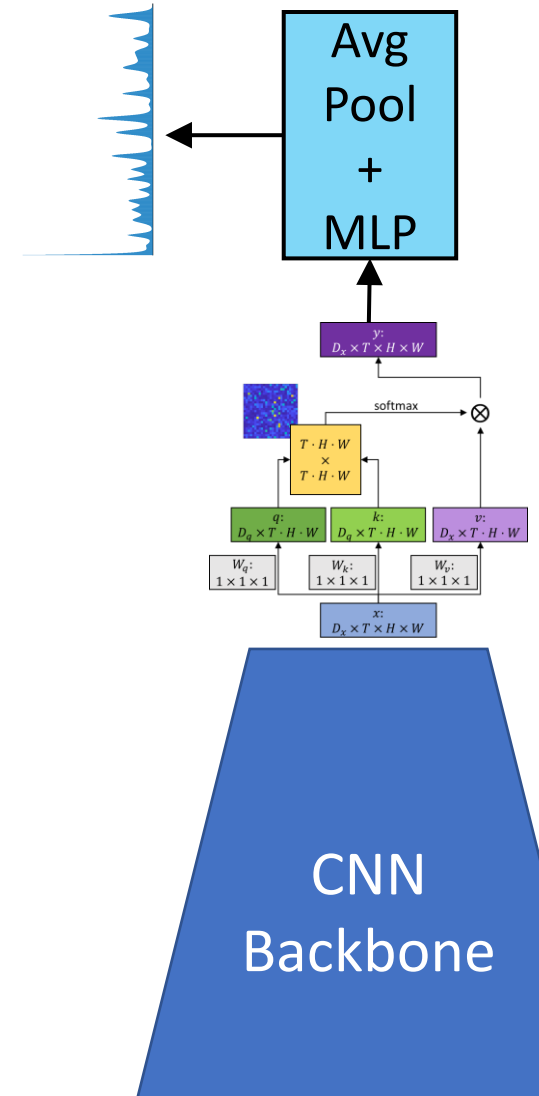
Outputs: $Y = AV$ (shape: $N_x \times D_v$)



Self Attention in Vision



Self Attention in Vision

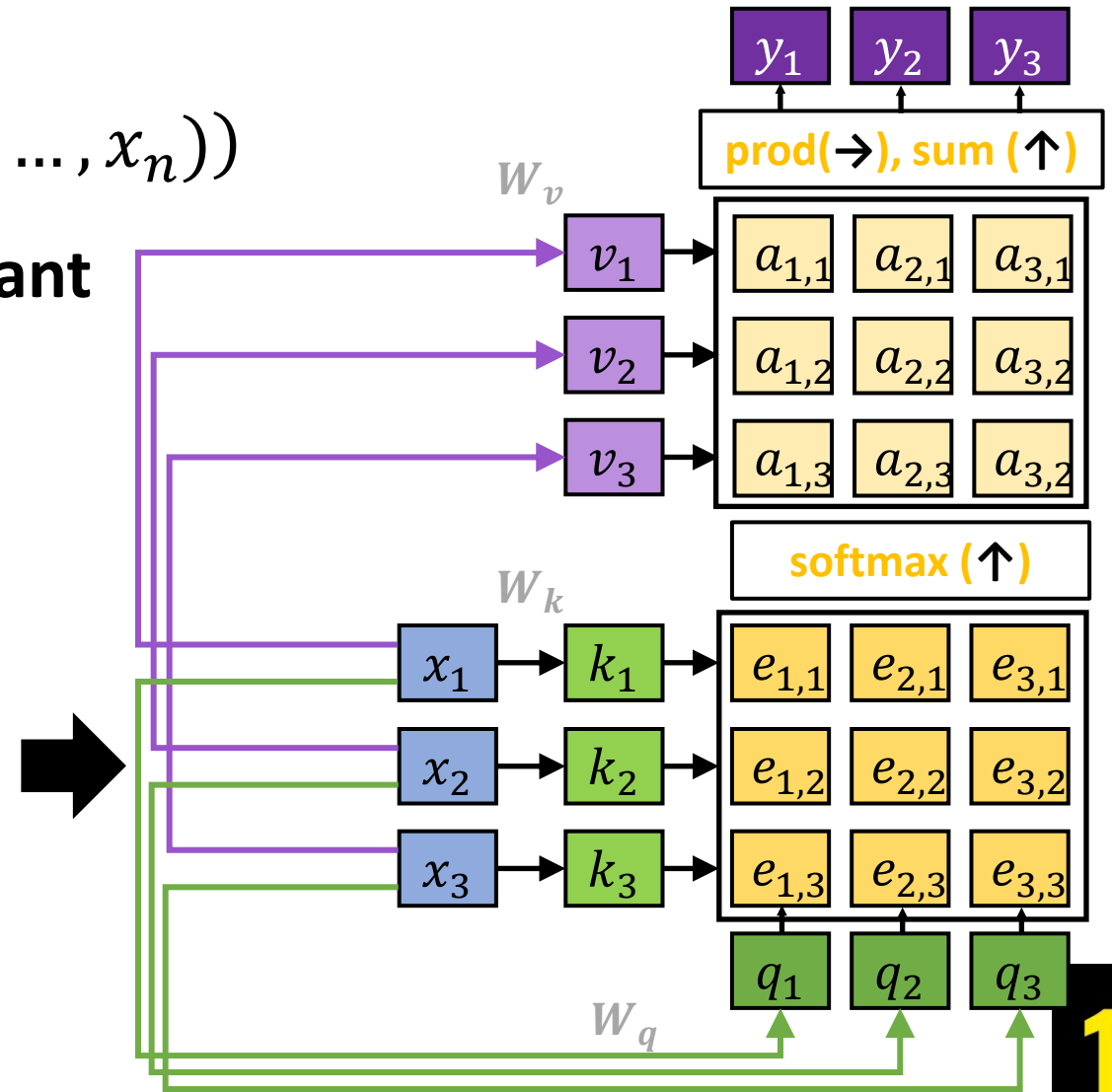


Self-Attention Layer: Properties

$$\text{SelfAtt}(\pi(x_1, \dots, x_n)) = \pi(\text{SelfAtt}(x_1, \dots, x_n))$$

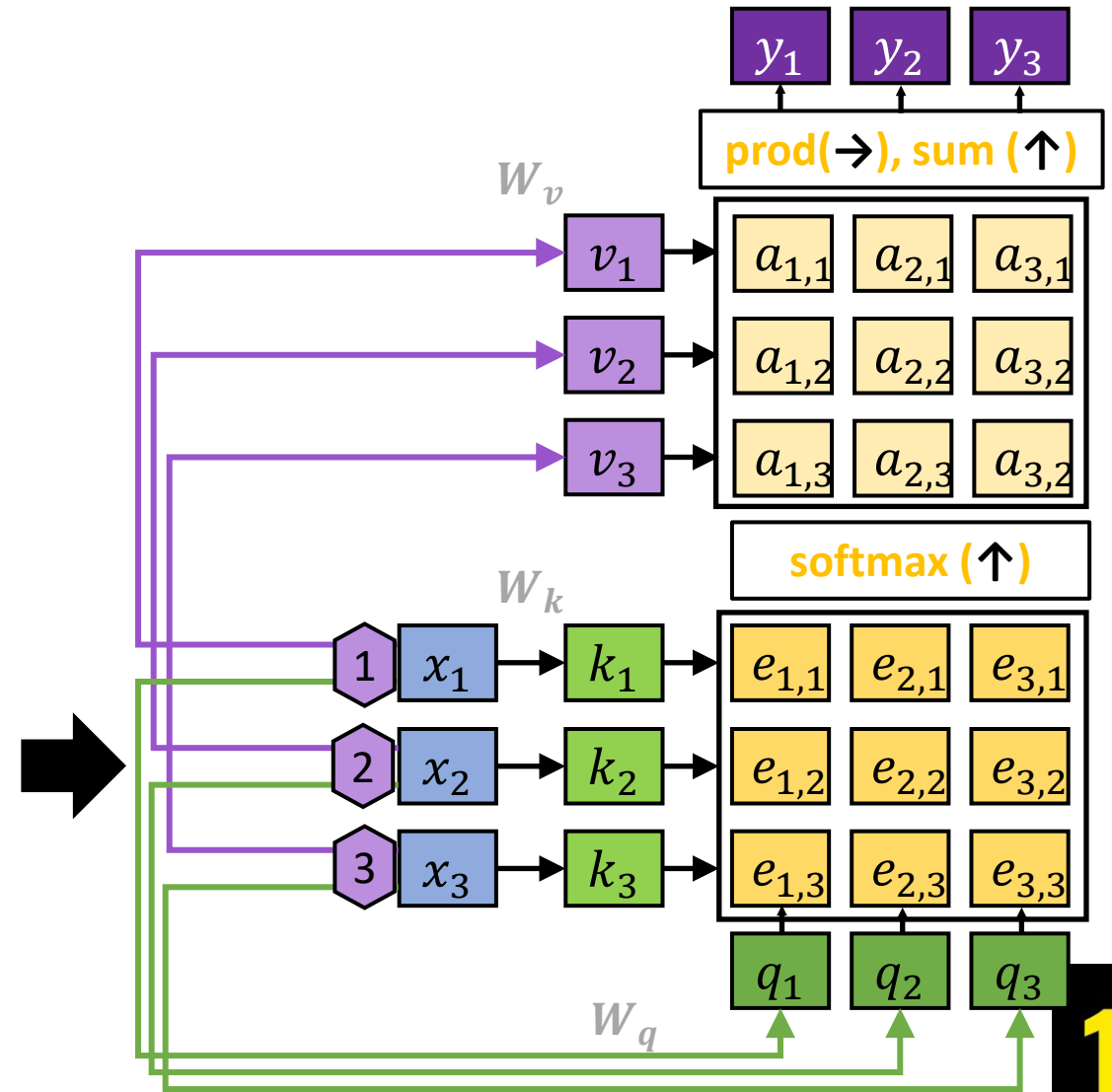
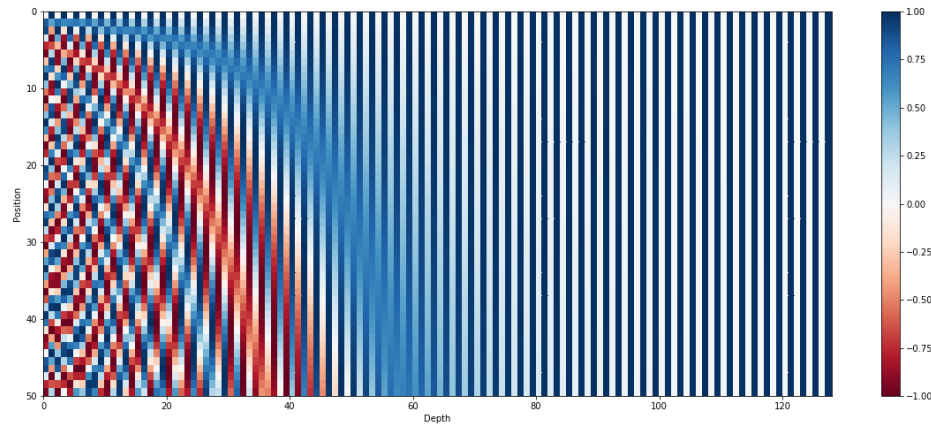
Self-Attention is **permutation equivariant**

“I am studying” ? “Am I studying”

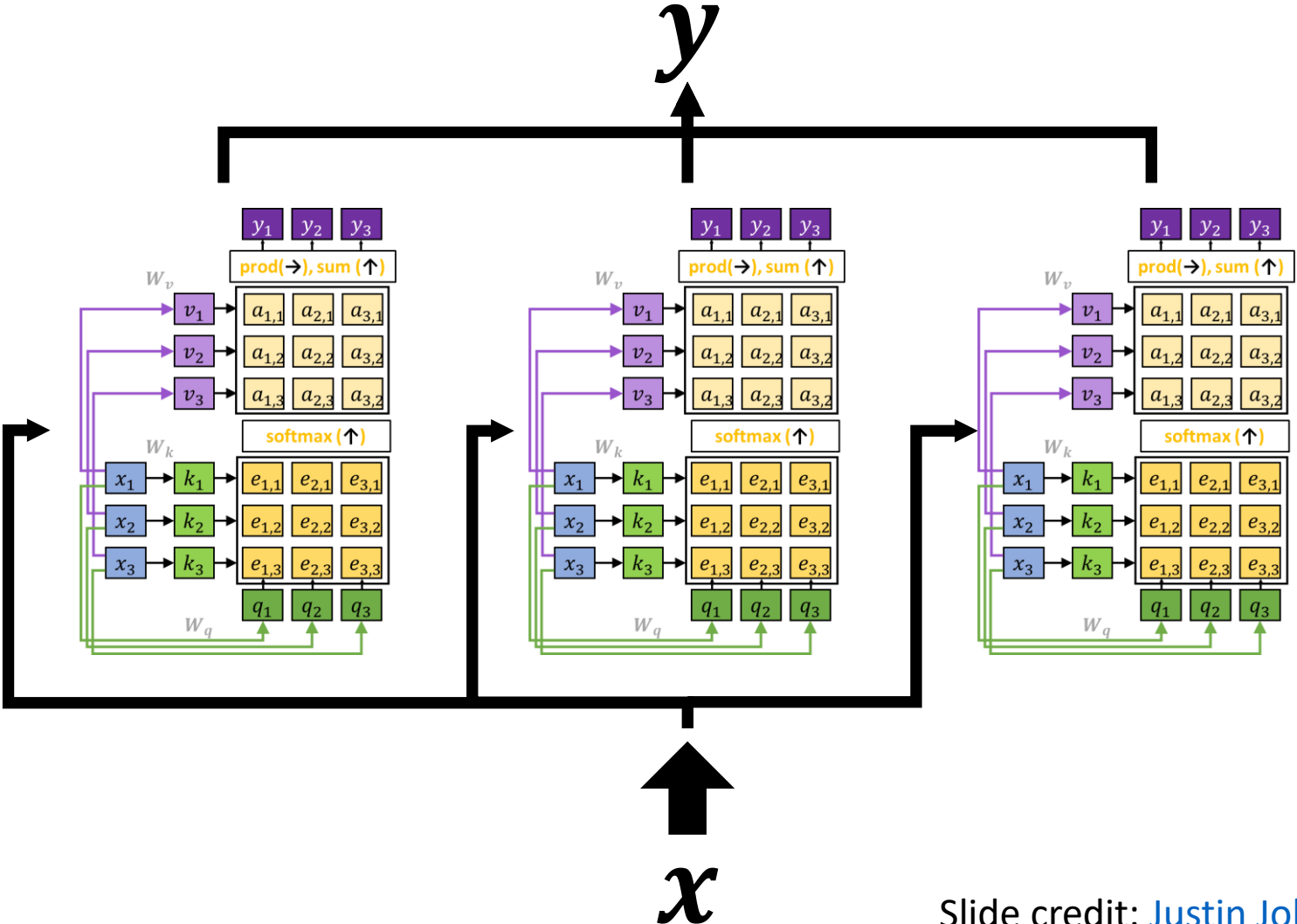


Self-Attention Layer: Properties

Positional Encoding

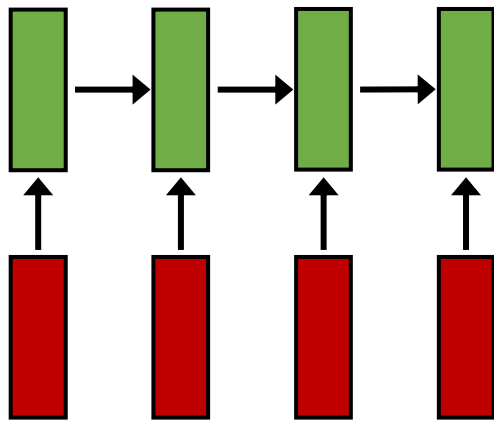


Multi-head Self-Attention



Three Ways of Processing Sequences

Recurrent Neural Network

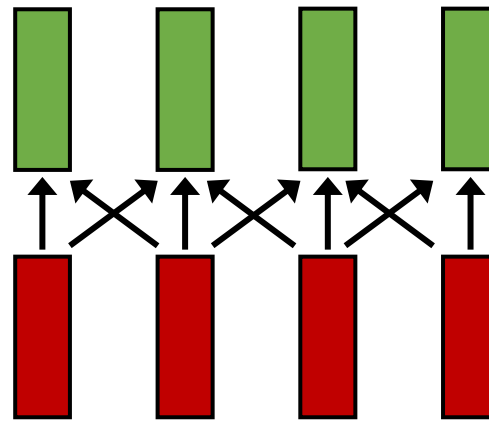


Works on **Ordered Sequences**

(+) large and adaptive receptive field via hidden state

(-) Not parallelizable: need to process states sequentially

1D Convolution

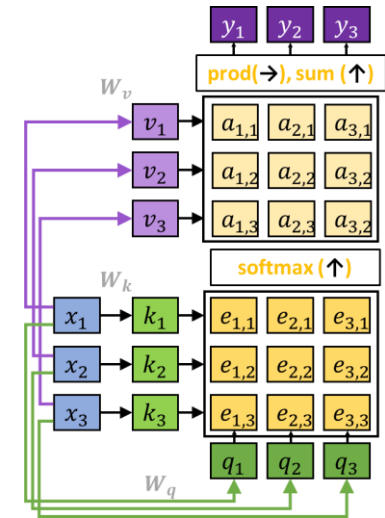


Works on **Multidimensional Grids**

(-) Fixed receptive field. Need to stack many layers to have a decent one

(+) Highly parallelizable

Self-Attention



Works on **Sets**

(+) receptive field = entire sequence

(+) parallelizable

(-) Very memory intensive

Three Ways of Processing Sequences

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

Aidan N. Gomez* †
University of Toronto
aidan@cs.toronto.edu

Łukasz Kaiser*
Google Brain
lukaszkaizer@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

NeurIPS 2017



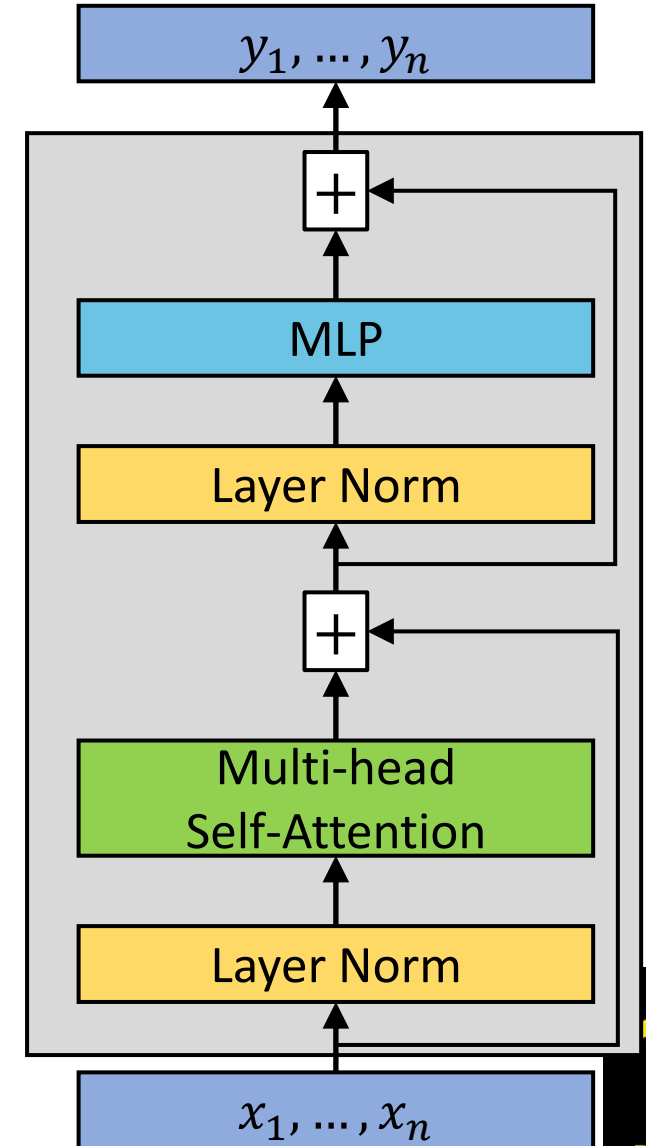
Transformer Layer

Input: x_1, \dots, x_n (n tokens in D dimensions)

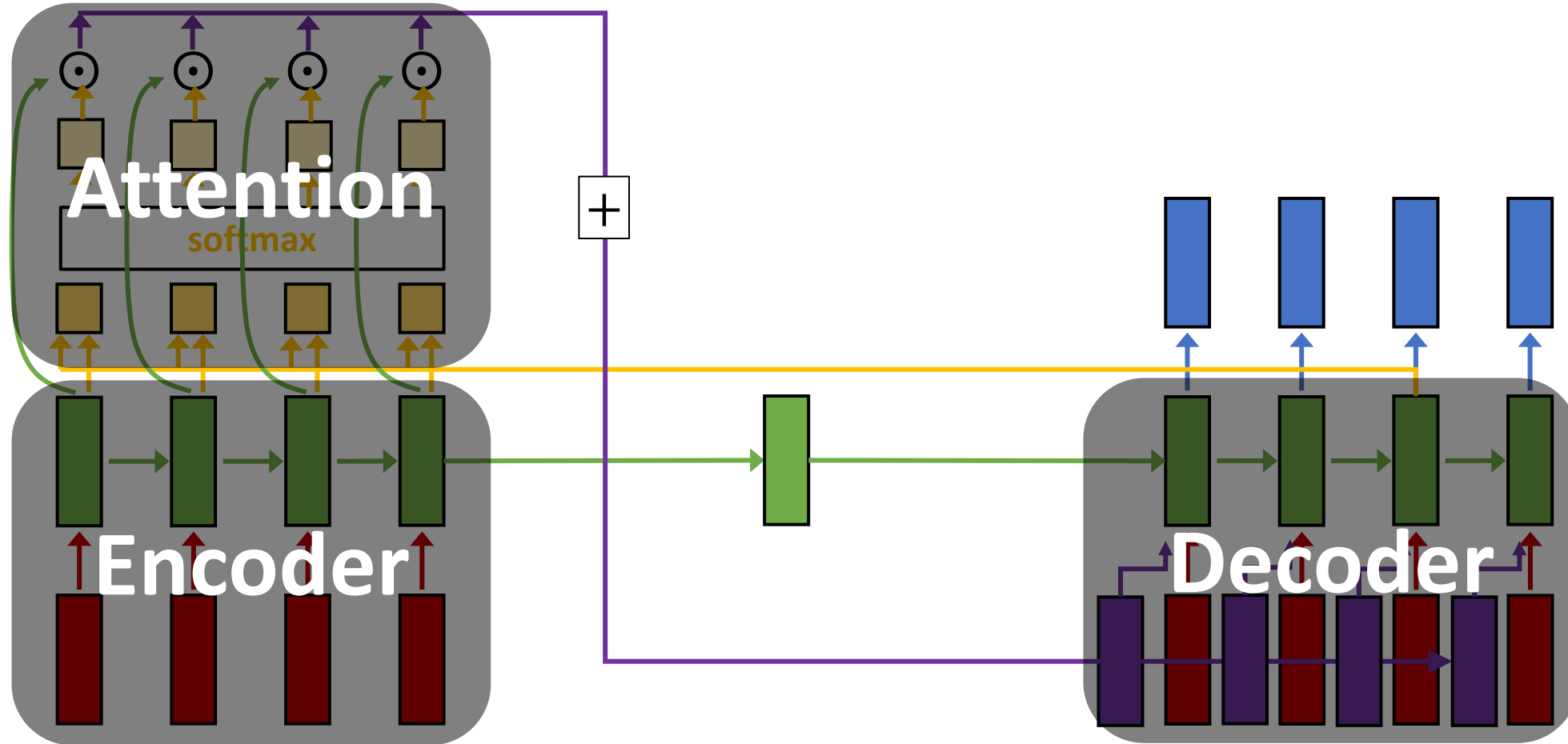
Output: y_1, \dots, y_n (n tokens in D dimensions)

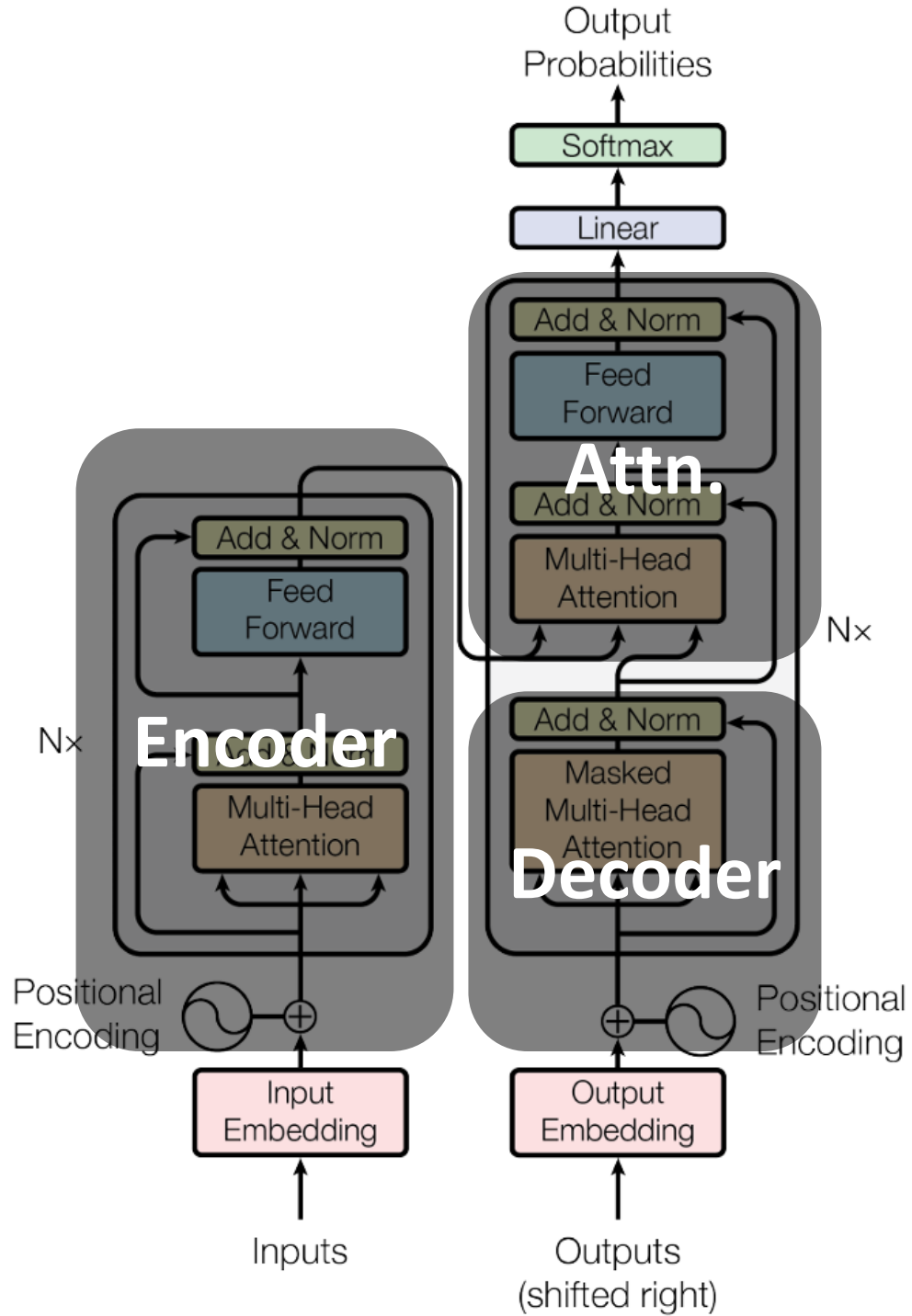
Highly scalable

Highly parallelizable



Sequence to Sequence





Transformers Network

Pretraining:

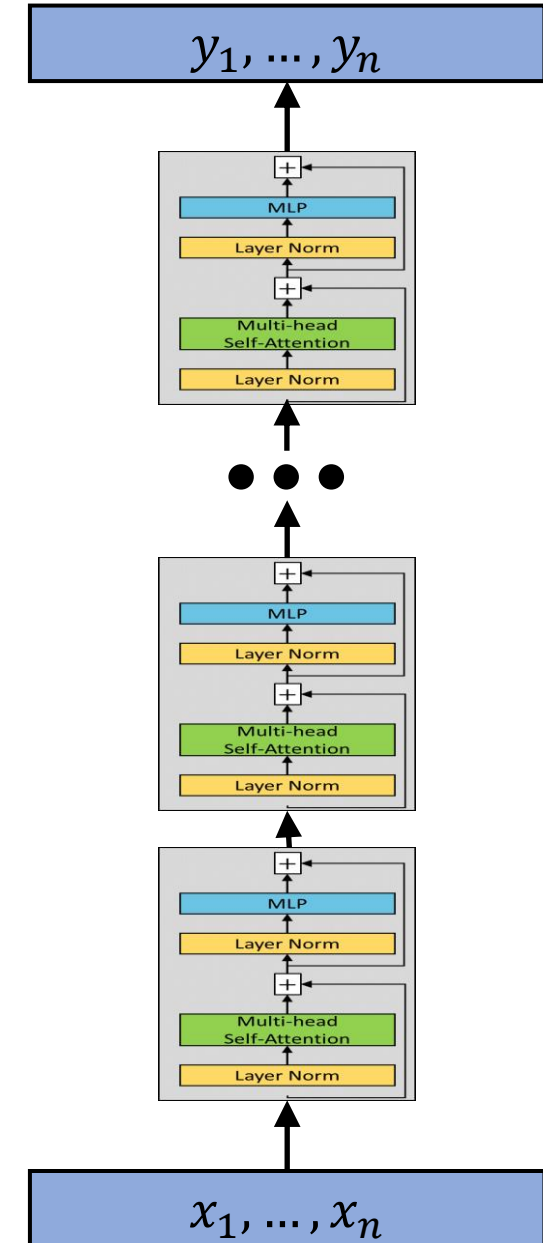
Download a LOT of text from the internet

Train a transformers network using self-supervision

Finetuning:

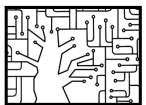
Fine-tune the transformer to specific NLP task at hand

Model	Layers	Width (D)	#Heads	#Params	Data
BERT-base	12	768	12	110M	13GB
BERT-large	24	1,024	16	340M	13GB
GPT-2	48	1,600	?	1.5B	40GB
GPT-3	96	12,288	96	175B	694GB

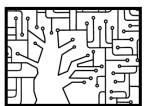


Example of GPT-3 generated text

chat.openai.com



Final Project – team up deadline



What's next?

Tomorrow:

AI & Robotics Seminar (not for credit)
Assaf Shocher

Next lecture:

Vision Transformers – ViT (Shai)

