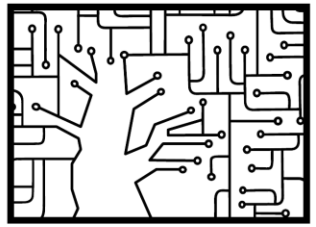
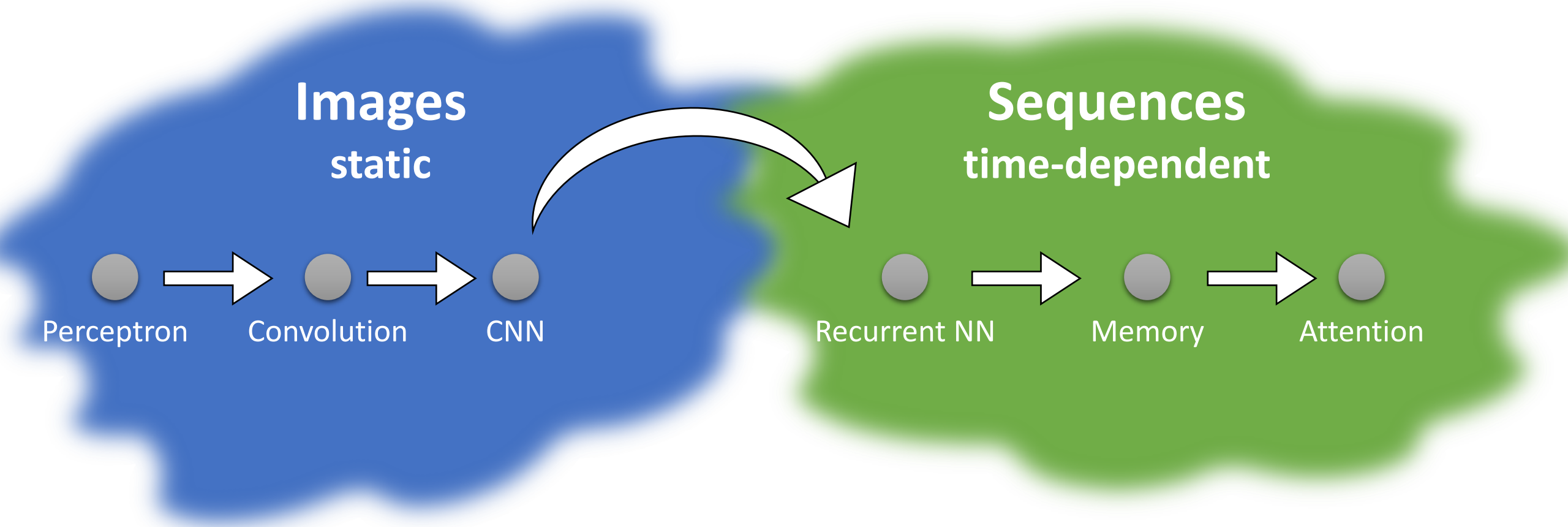
Deep Learning for Computer Vision:
Sequences and Attention

Shai Bagon



Agenda

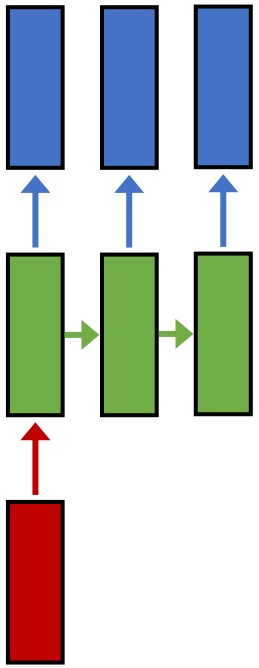

Images
static
Sequences
time-dependent
Perceptron
Convolution
CNN
Recurrent NN
Memory
Attention

Deep Learning for Sequences

One to one

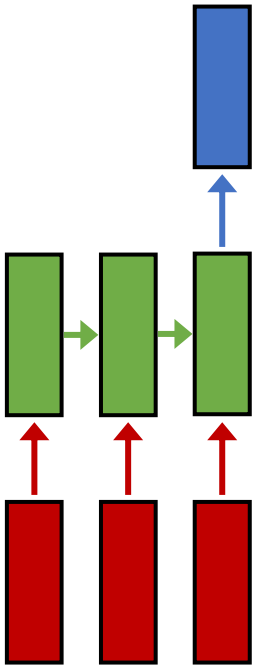
Feed-for
e.g., **classif**
image -> label

One to many



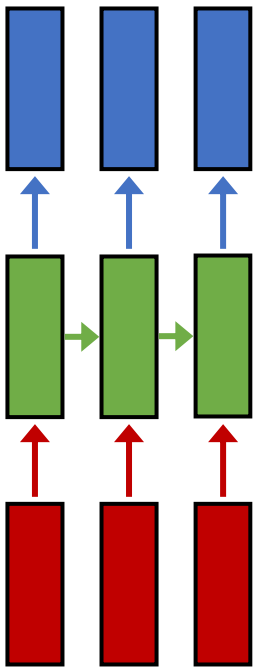
e.g., **image** ca
image -> sequenc

Many to one



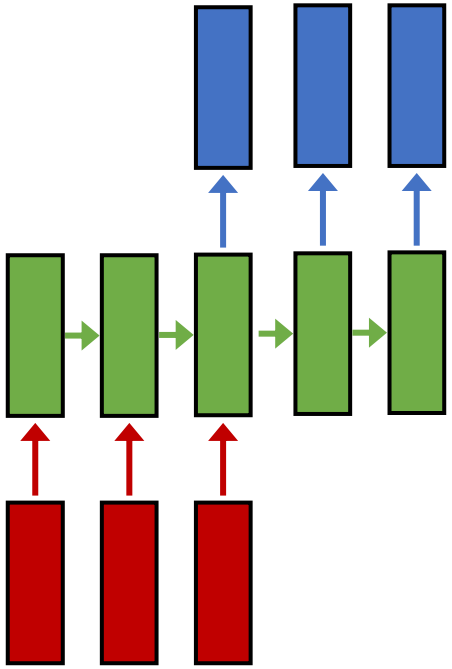
e.g., **video c**
sequence of f

Many to many



e.g., **video frames classi**
sequence of frames -> sec

Many to many



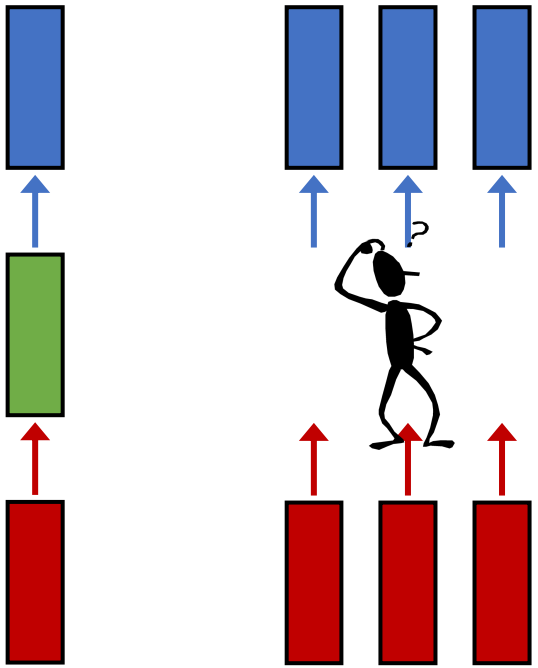
e.g., **Machine translation**
sequence of words -> seq. of words

Slide credit: Justin Johnson (EECS-498-007, UMich)

Recurrent Neural Networks

One to one

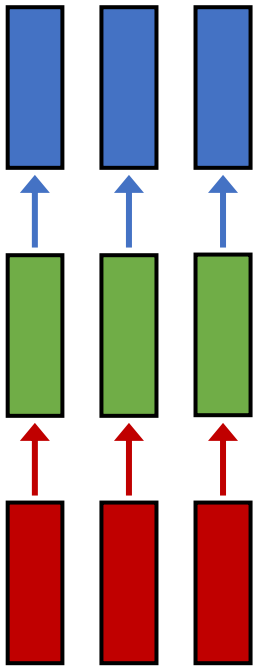
Many to many



Recurrent Neural Networks

One to one

Many to many



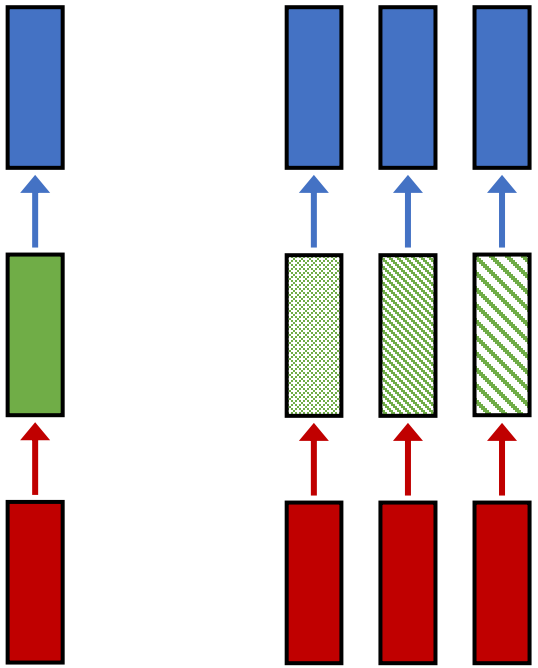
```python
class ManyToManyV0(nn.Module):
    def __init__(self, number_of_time_steps):
        super(ManyToMany, self).__init__()
        # SAME instance of SingleTimeStep for each time step
        self.time_steps = SingleTimeStep(...)

    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



```python
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
```

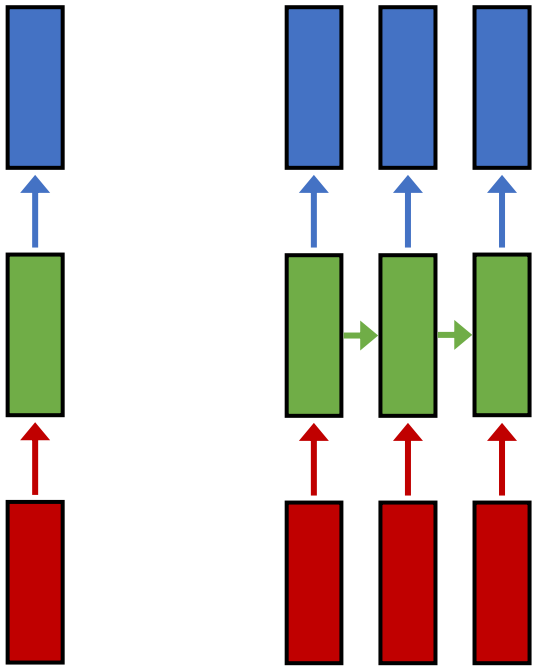
(+?) Temporal dependency (via trained parameters)

(-) Parameter inefficiency

(-) Fixed sequence length

Recurrent Neural Networks

One to one Many to many



```python
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”)

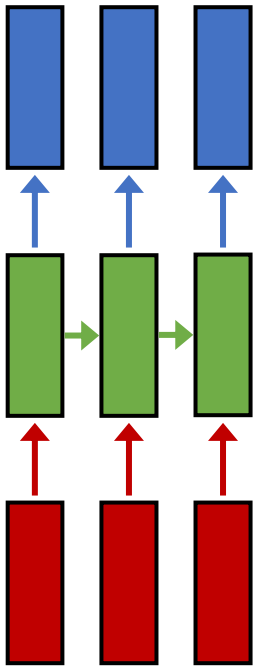
(+) Parameter efficiency

(+) Arbitrary sequence length

Recurrent Neural Networks

One to one

Many to many



```python
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
```

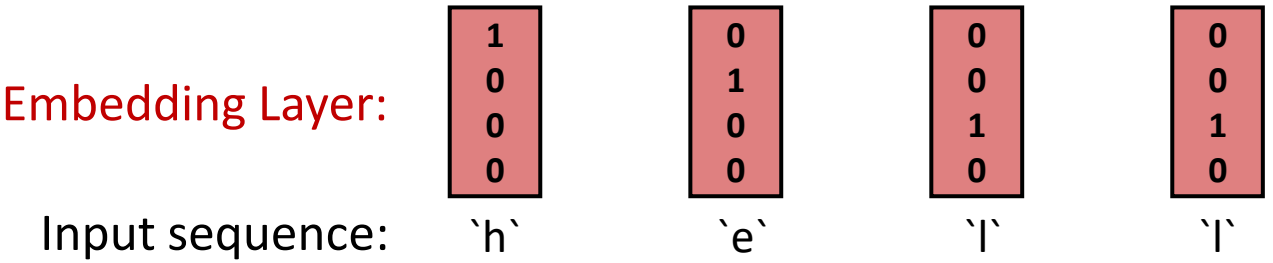
$$h_t = f(h_{t-1}, x_t; \boldsymbol{W})$$

Example: Language Modeling

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

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]



Slide credit: Justin Johnson (EECS-498-007, UMich)

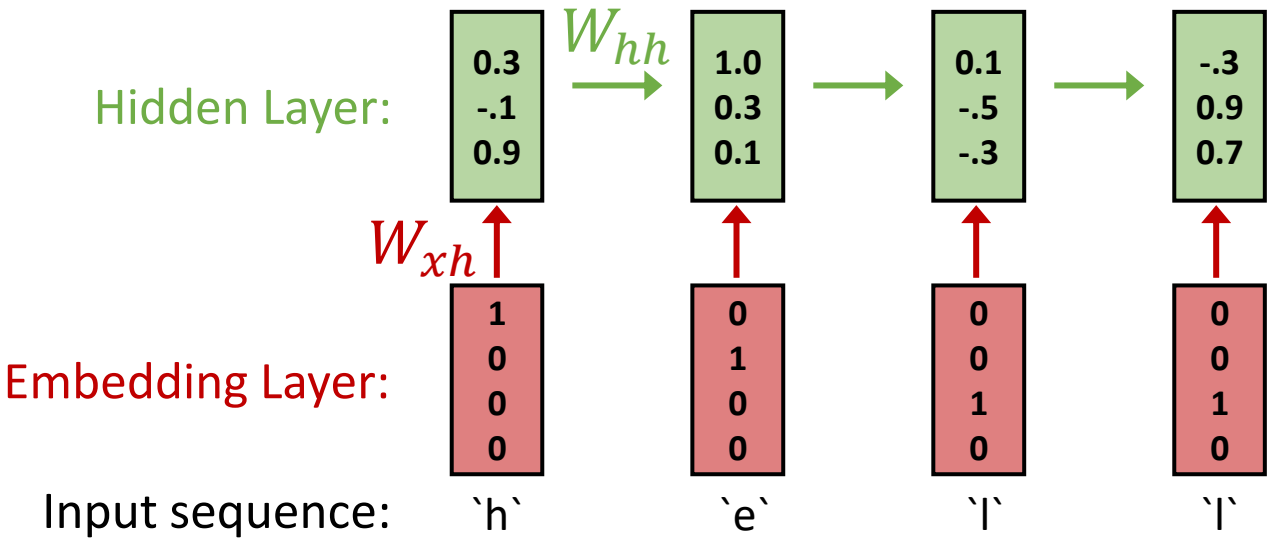
Example: Language Modeling

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

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

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]



Slide credit: Justin Johnson (EECS-498-007, UMich)

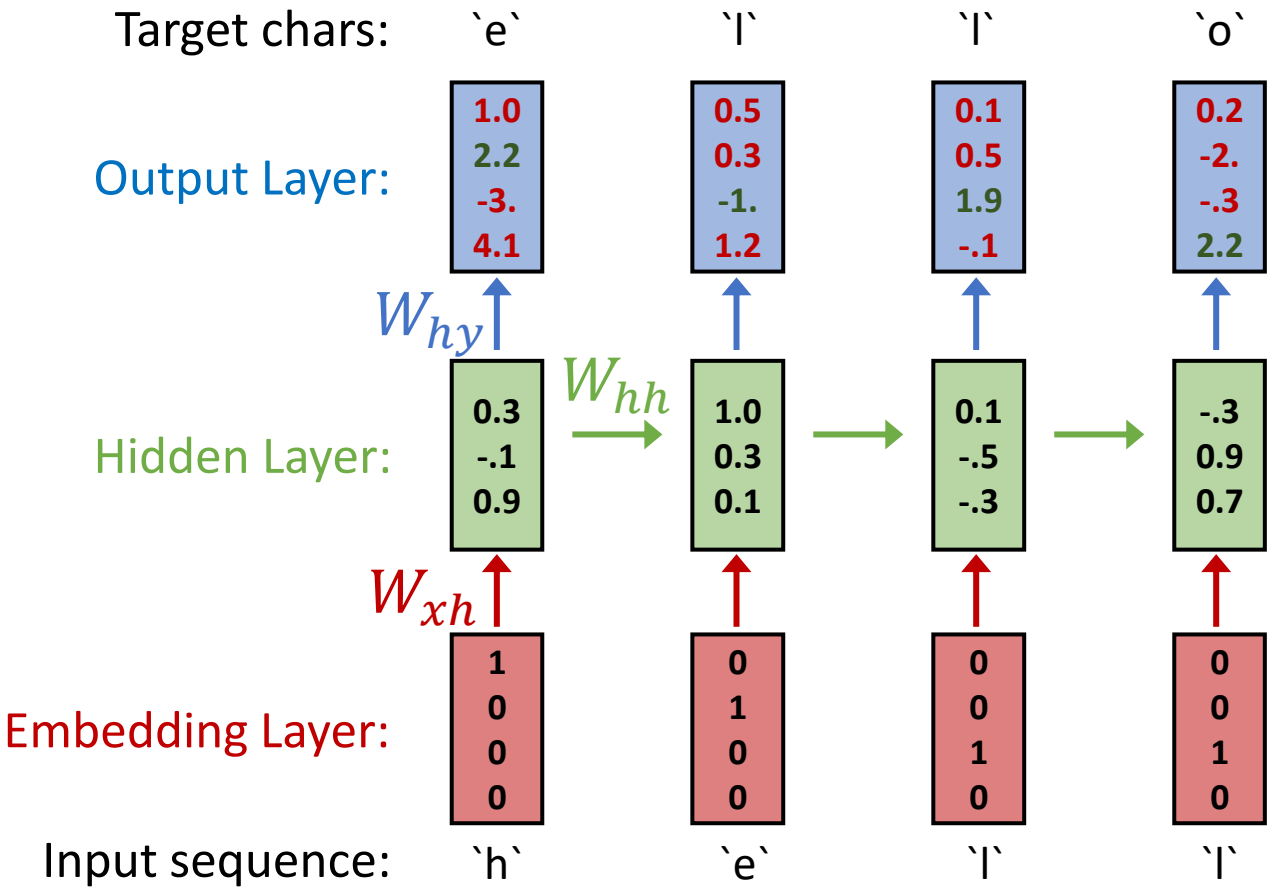
Example: Language Modeling

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

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

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]



Slide credit: Justin Johnson (EECS-498-007, UMich)

Example: Language Modeling

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

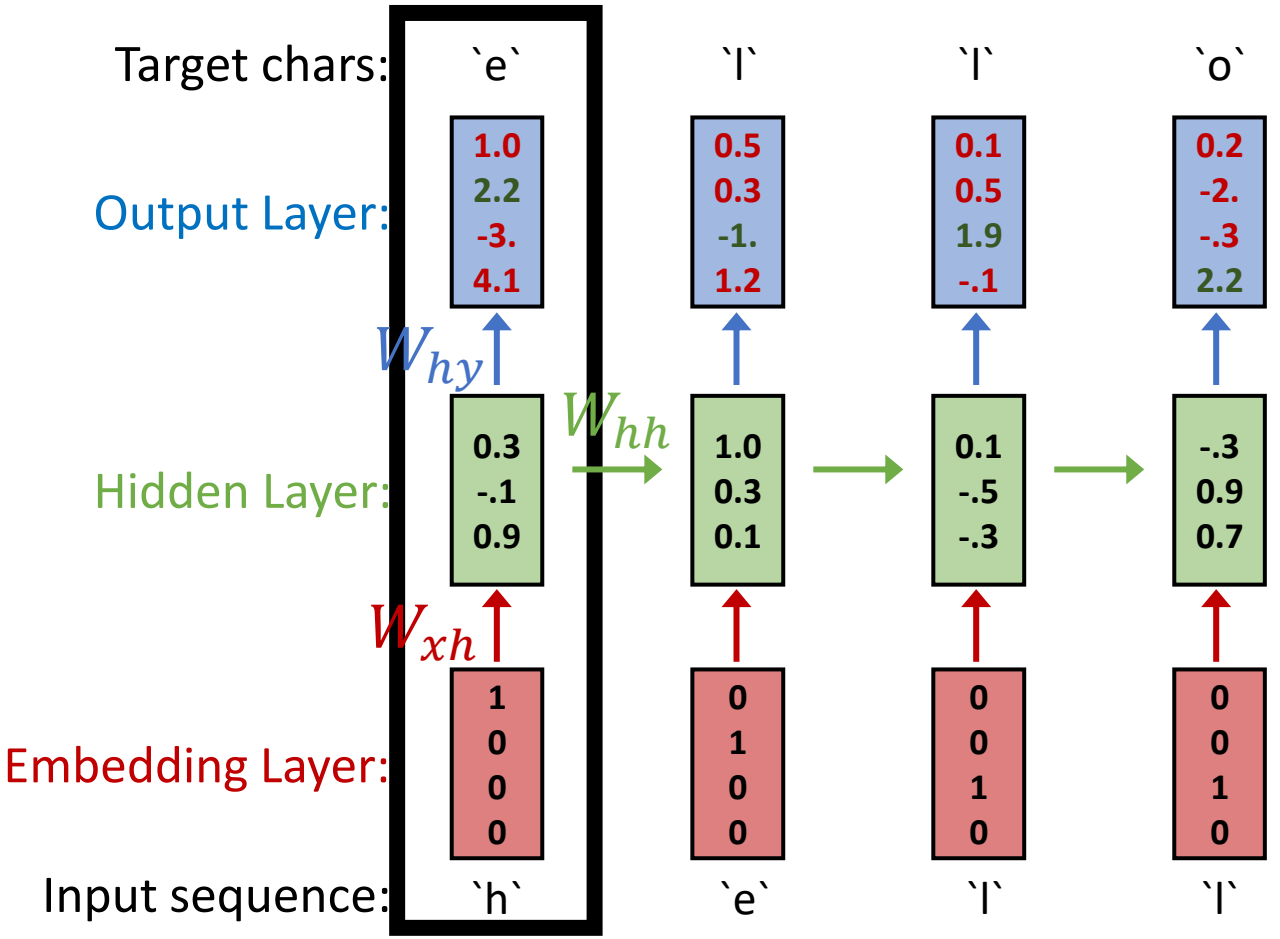
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]

Given “h” predict “e”



Slide credit: Justin Johnson (EECS-498-007, UMich)

Example: Language Modeling

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

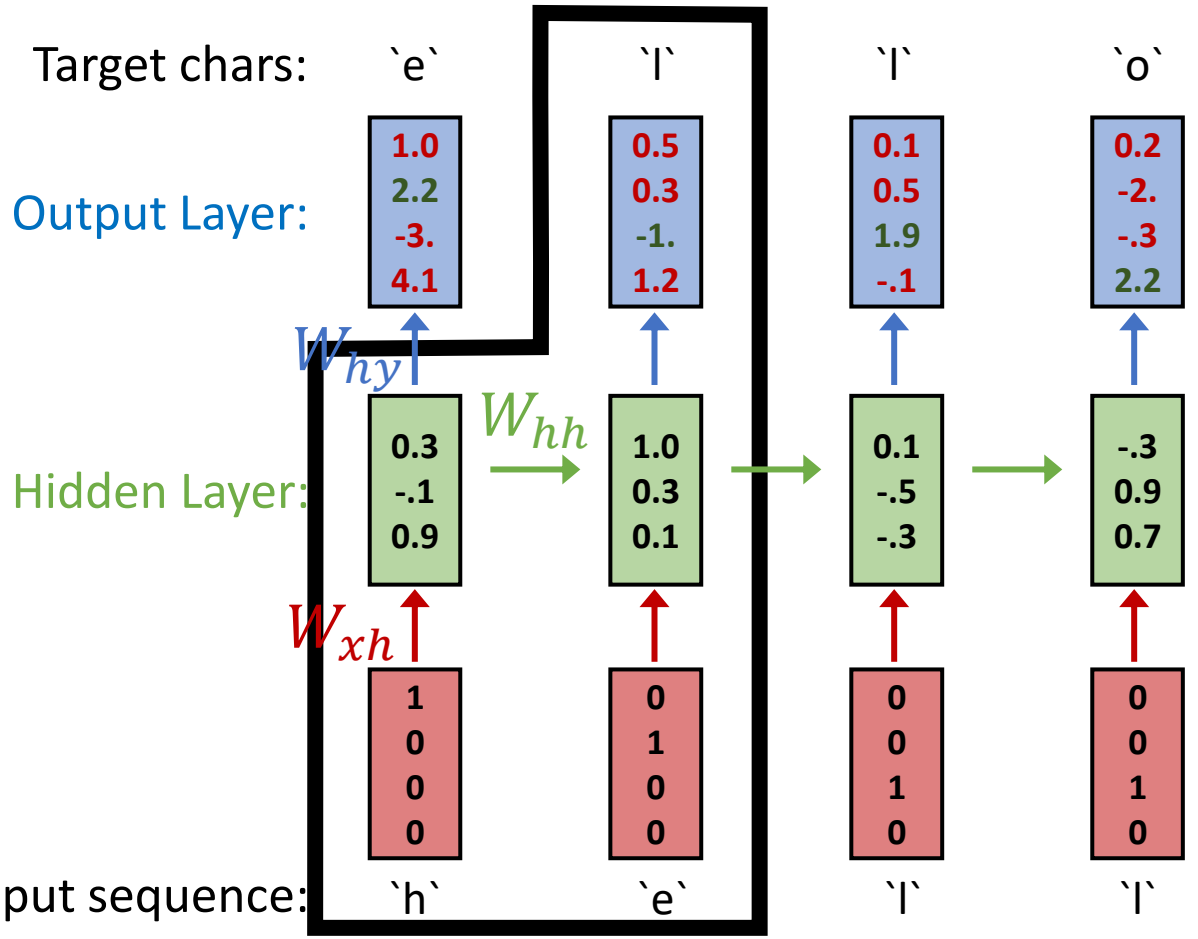
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]

Given “he” predict “l”



Slide credit: Justin Johnson (EECS-498-007, UMich)

Example: Language Modeling

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

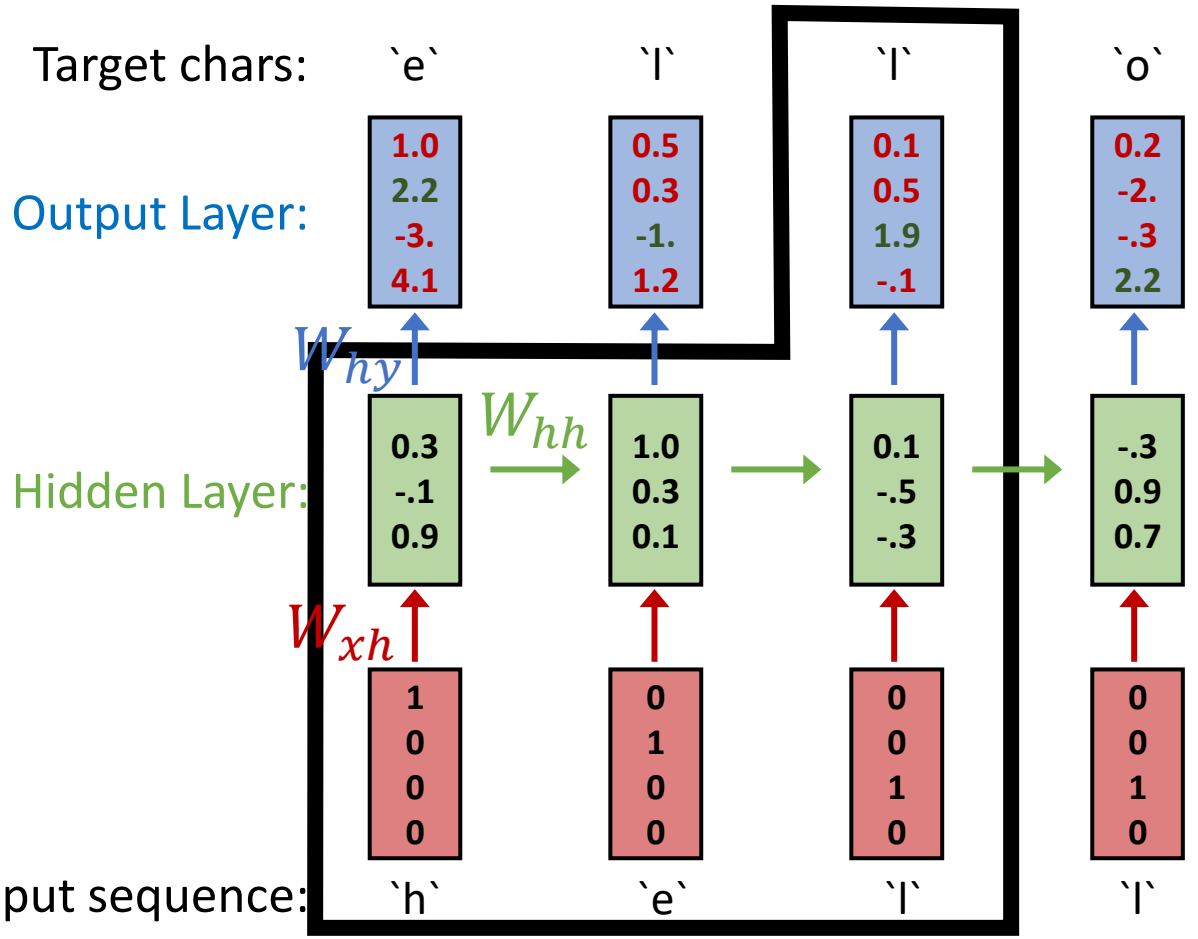
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]

Given “hel” predict “l”



Slide credit: Justin Johnson (EECS-498-007, UMich)

Example: Language Modeling

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

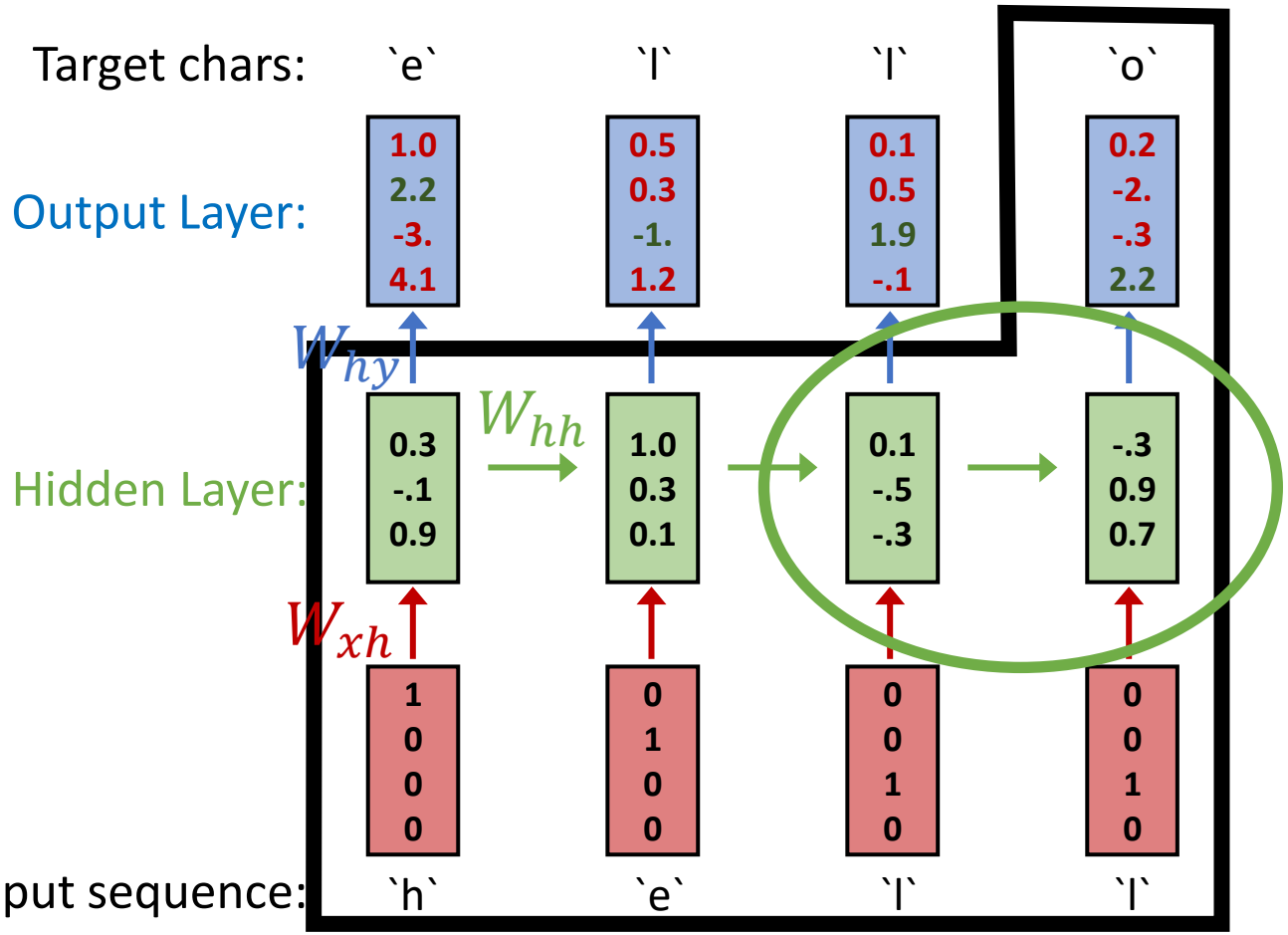
$$h_t = \tanh(W_{hh} h_{t-1} + W_{xh} x_t)$$

$$y_t = W_{hy} h_t$$

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]

Given “hell” predict “o”



Slide credit: Justin Johnson (EECS-498-007, UMich)

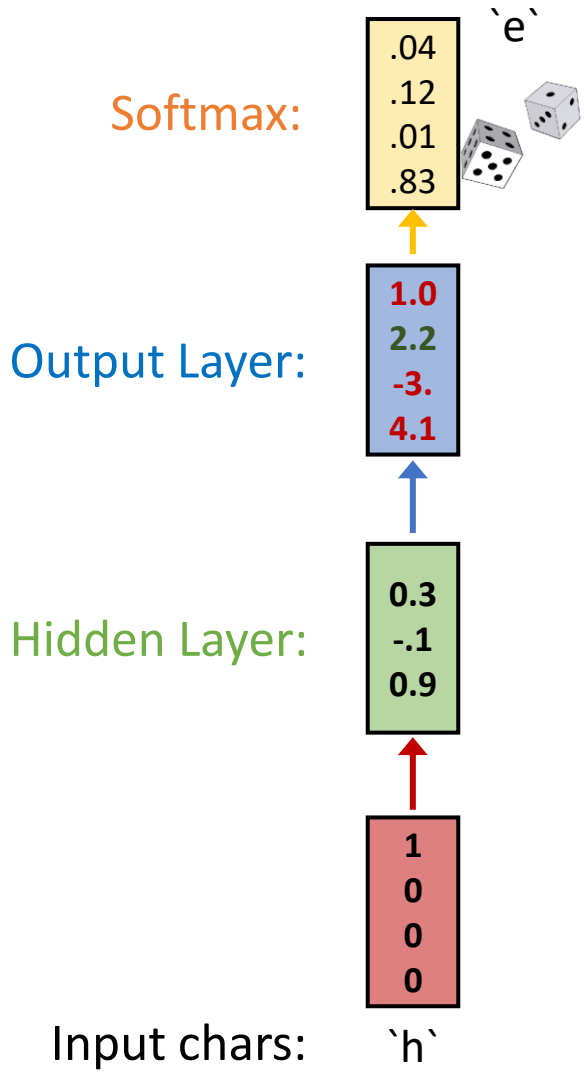
Example: Language Modeling

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

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

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]



Slide credit: Justin Johnson (EECS-498-007, UMich)

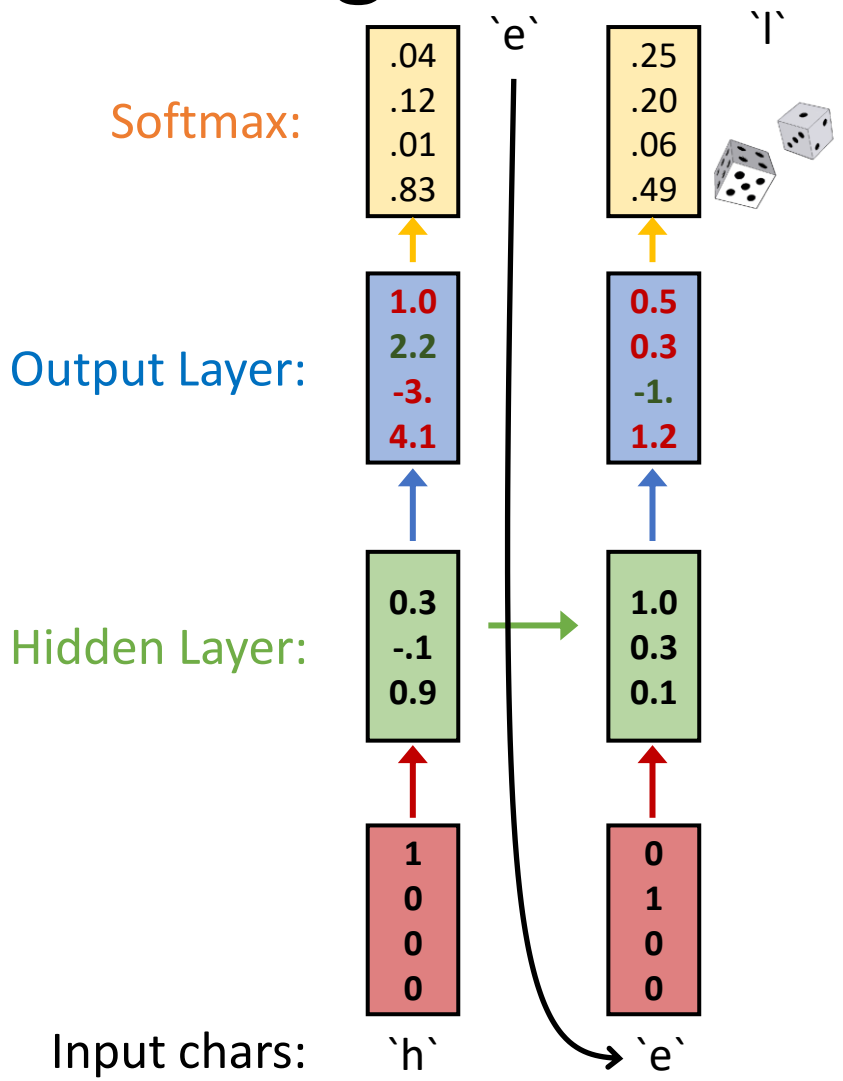
Example: Language Modeling

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

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

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]



Slide credit: Justin Johnson (EECS-498-007, UMich)

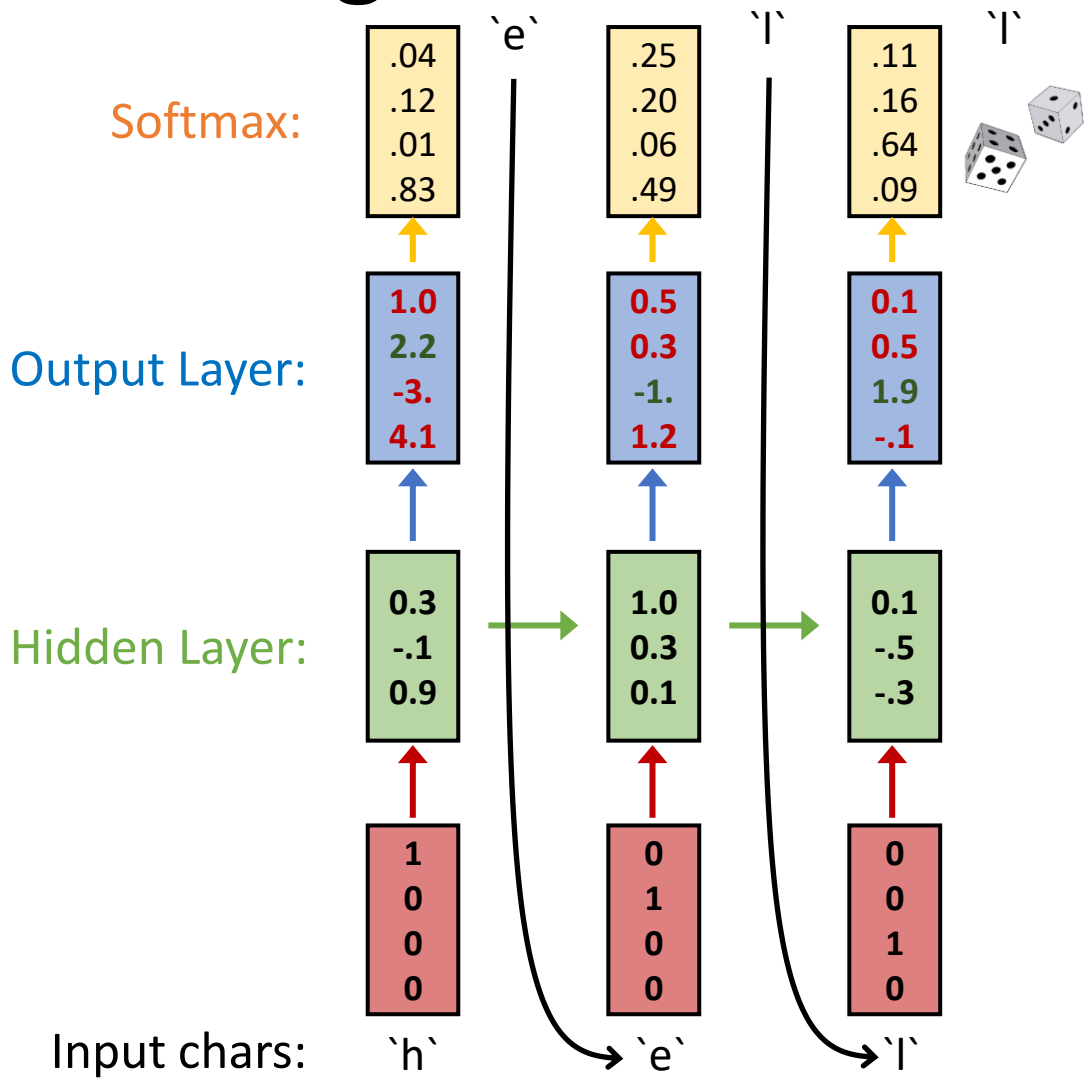
Example: Language Modeling

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

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

Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]



Slide credit: Justin Johnson (EECS-498-007, UMich)

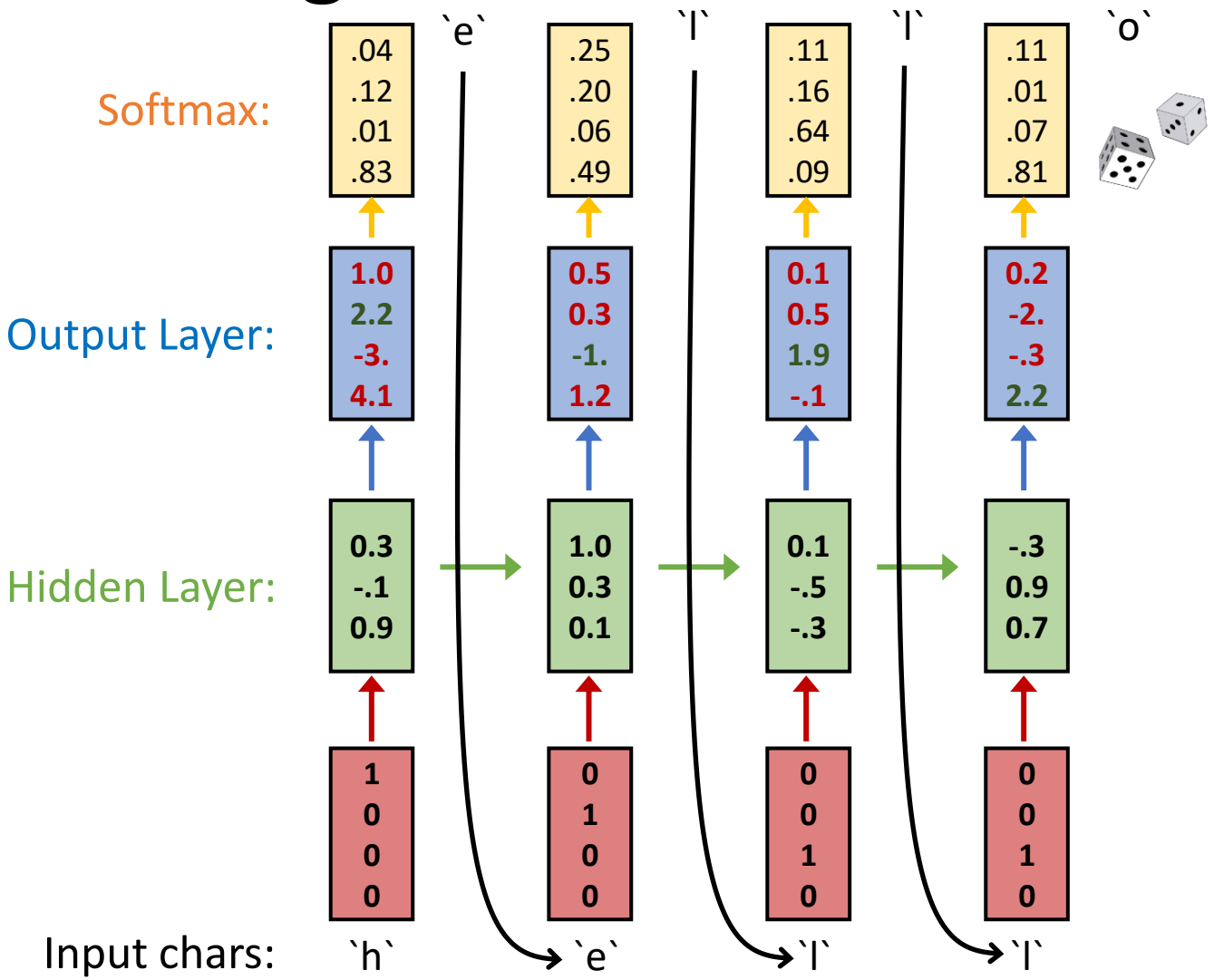
Example: Language Modeling

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

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

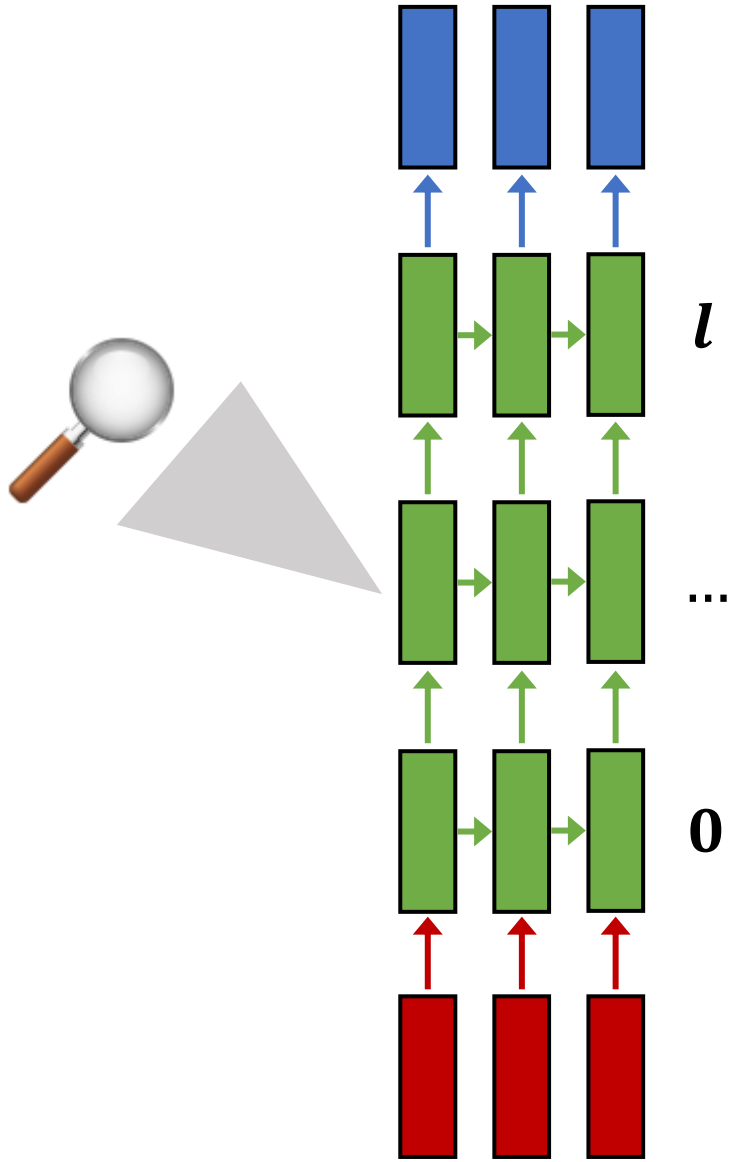
Training sequence: “hello”

Vocabulary: [‘h’, ‘e’, ‘l’, ‘o’]



Slide credit: Justin Johnson (EECS-498-007, UMich)

Searching for Interpretable Cells



Searching for Interpretable Cells

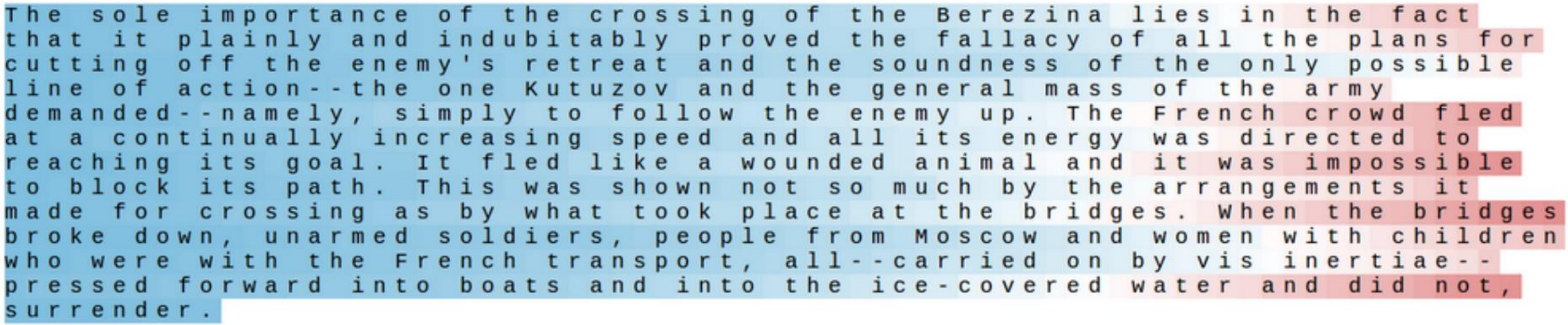
A large portion of cells are not easily interpretable. Here is a typical example:

```
/*  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.
   */
```


Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Searching for Interpretable Cells

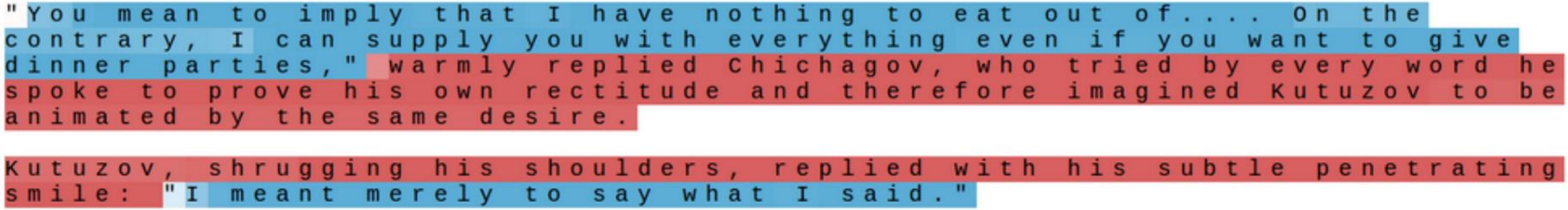
Cell sensitive to position in line:



Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

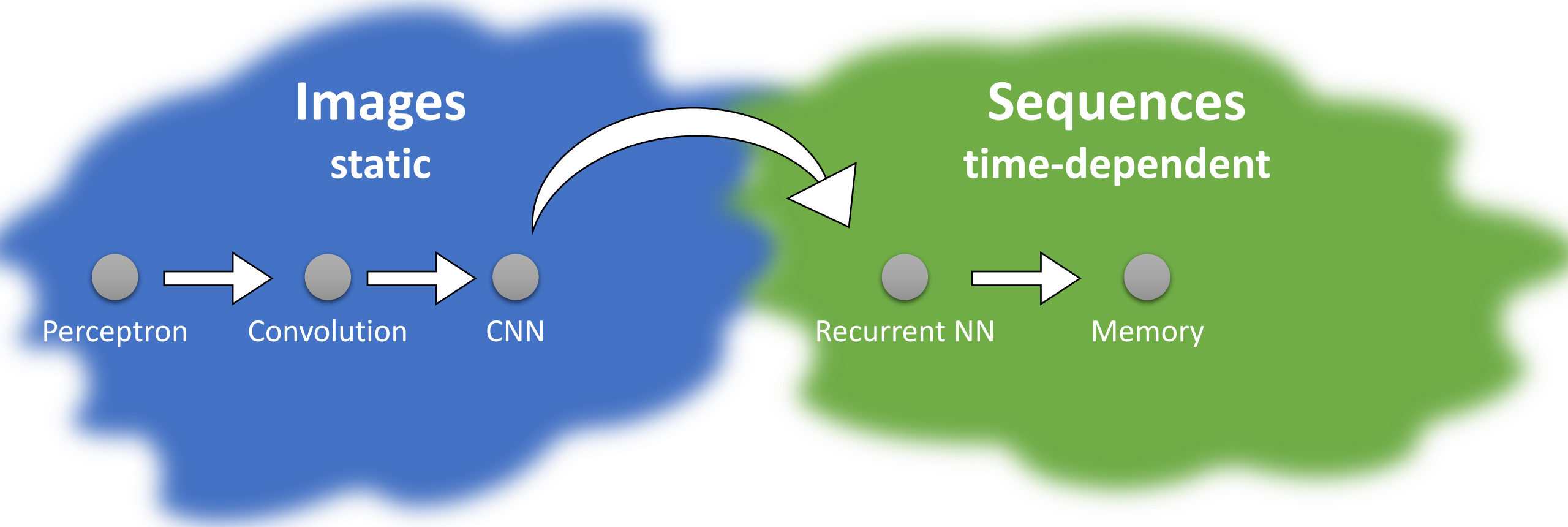
Searching for Interpretable Cells

Cell that turns on inside quotes:

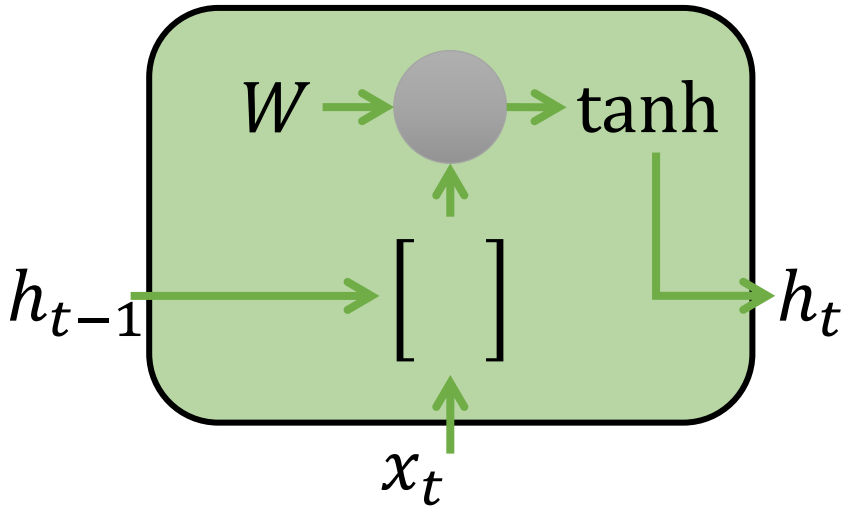


Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016

Agenda


Images
static
Sequences
time-dependent
Perceptron
Convolution
CNN
Recurrent NN
Memory

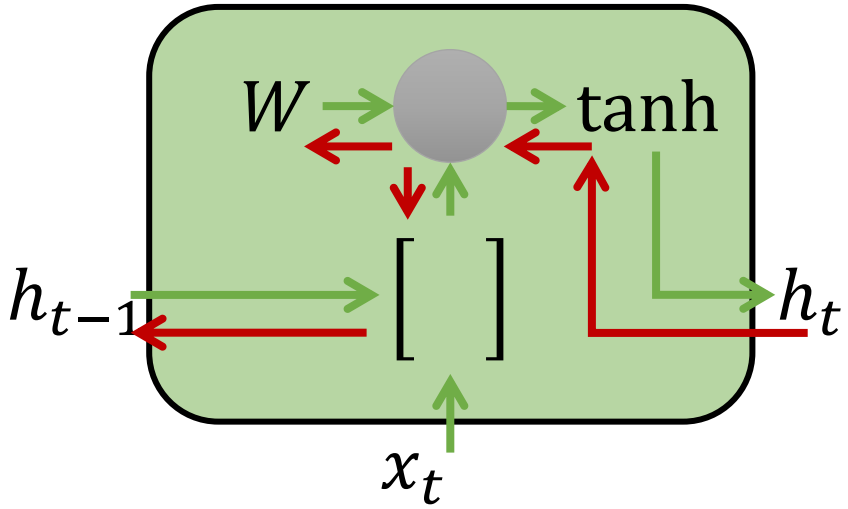
RNN: Gradient Flow



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

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

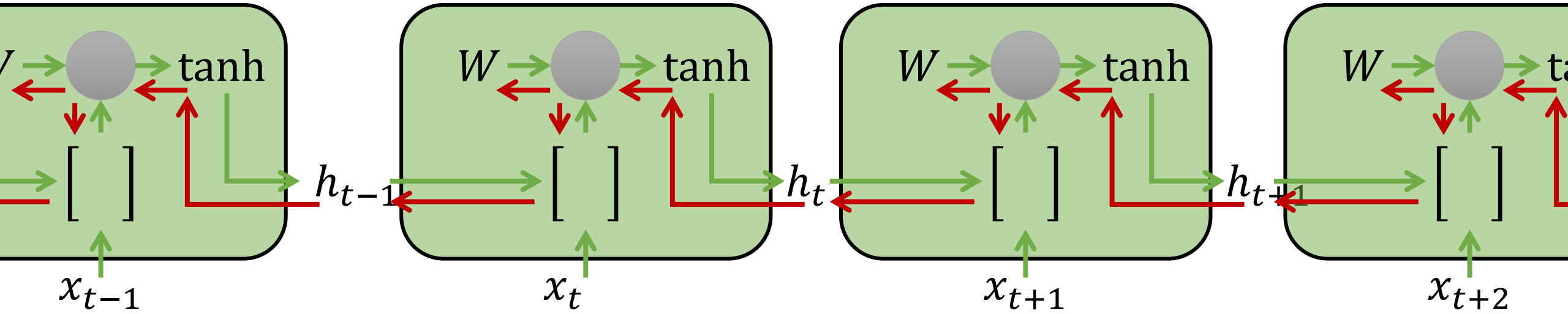
RNN: Gradient Flow



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

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

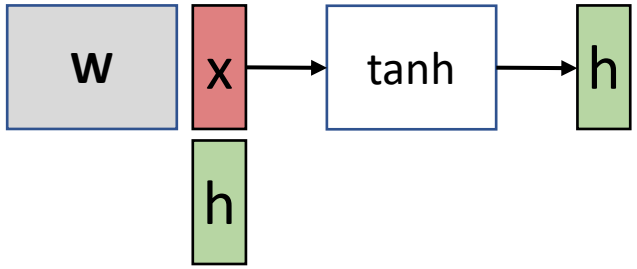
RNN: Gradient Flow

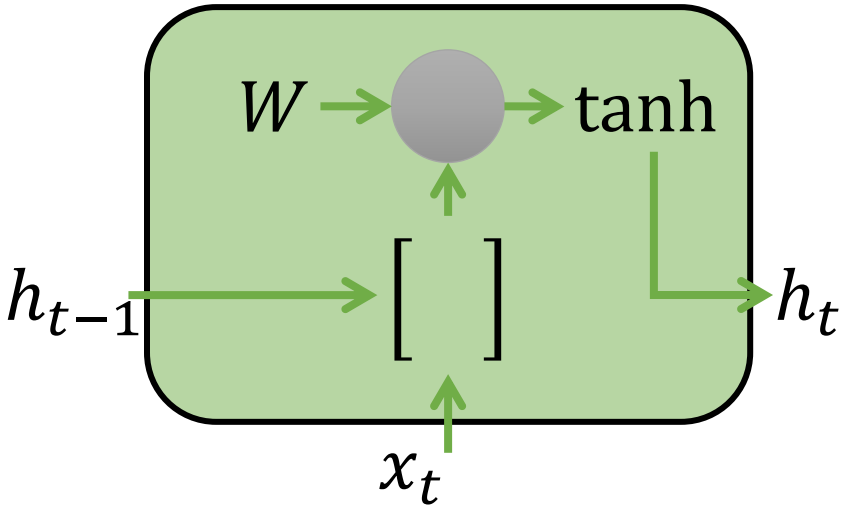


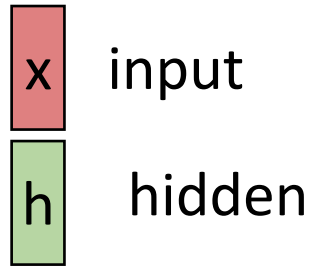
Forwarding/backwarding in time for long sequences = very deep NN

Easily leads to exploding/vanishing gradients

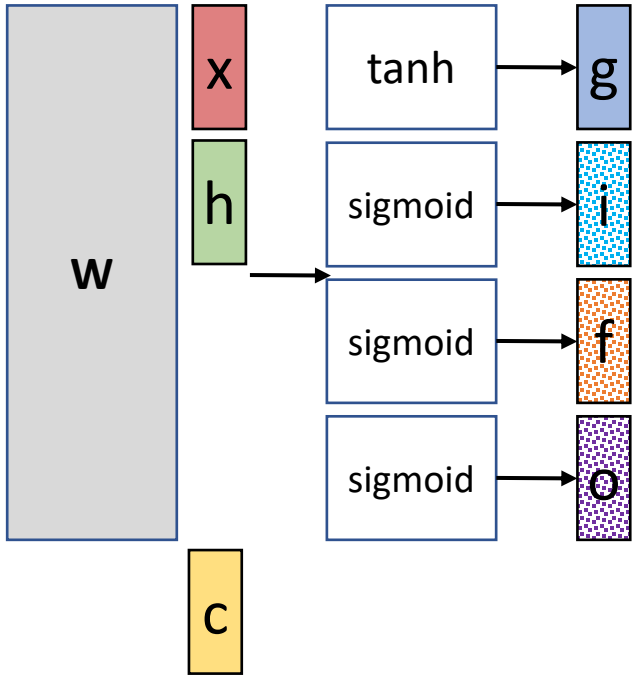
Long Short-Term Memory (LSTM)



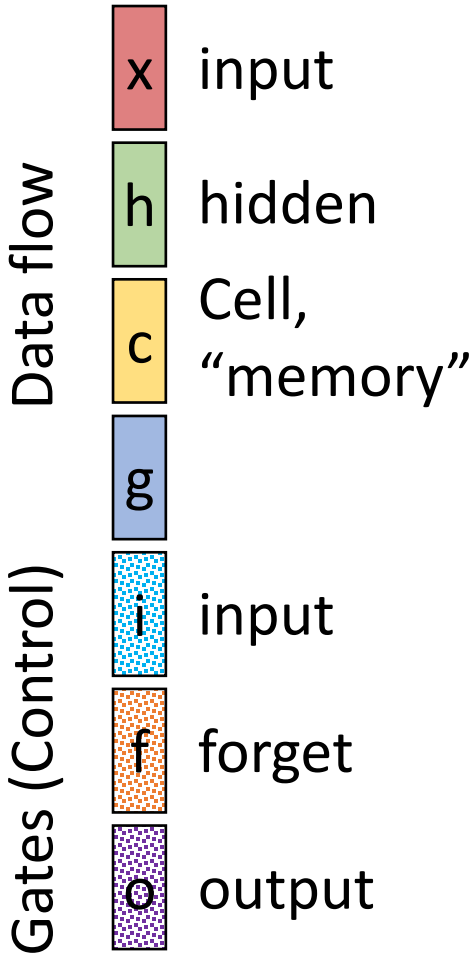




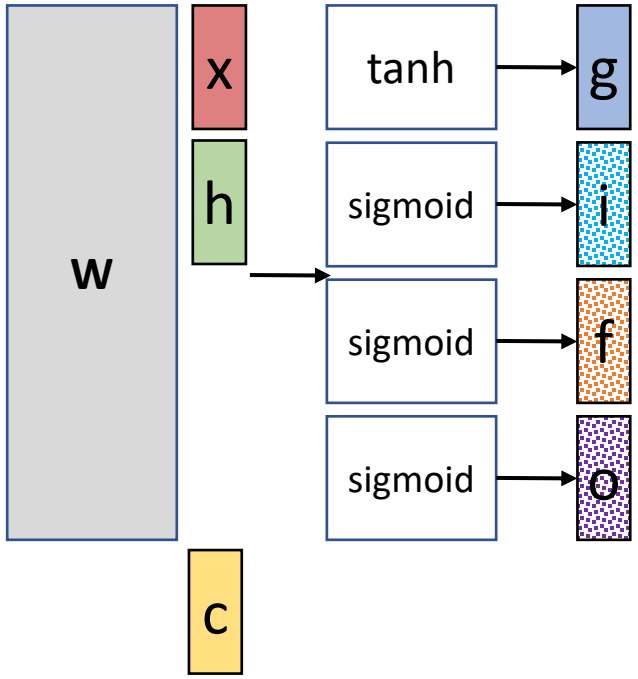
Long Short-Term Memory (LSTM)

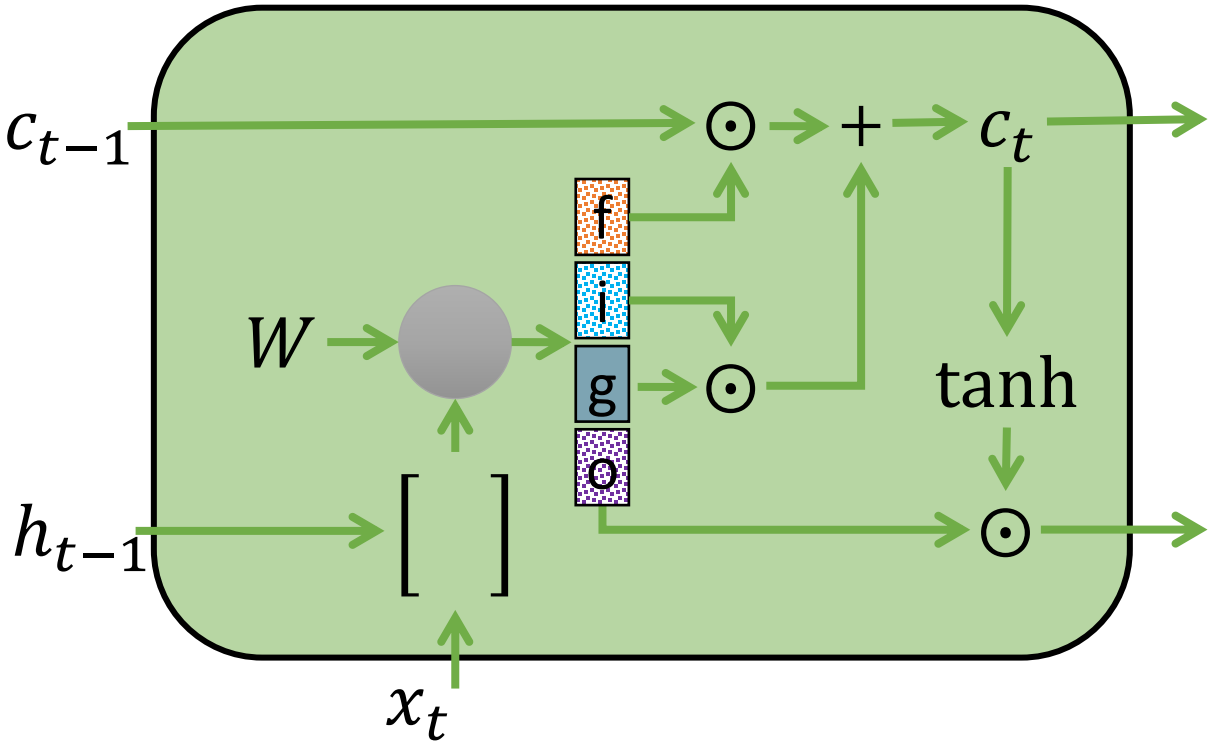


$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$



Long Short-Term Memory (LSTM)





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

$$h_t = o_t \odot \tanh(c_t)$$

Data flow

Gates (Control)

x input

h hidden

c Cell, "memory"

g

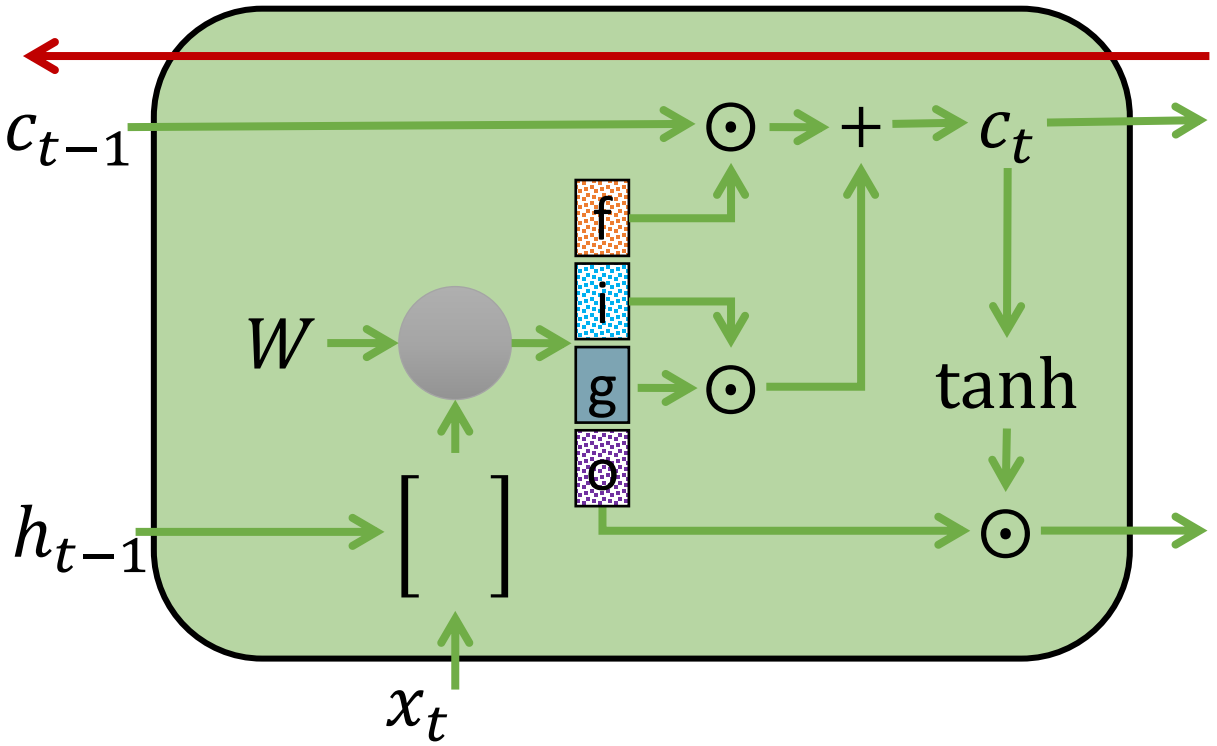
i input

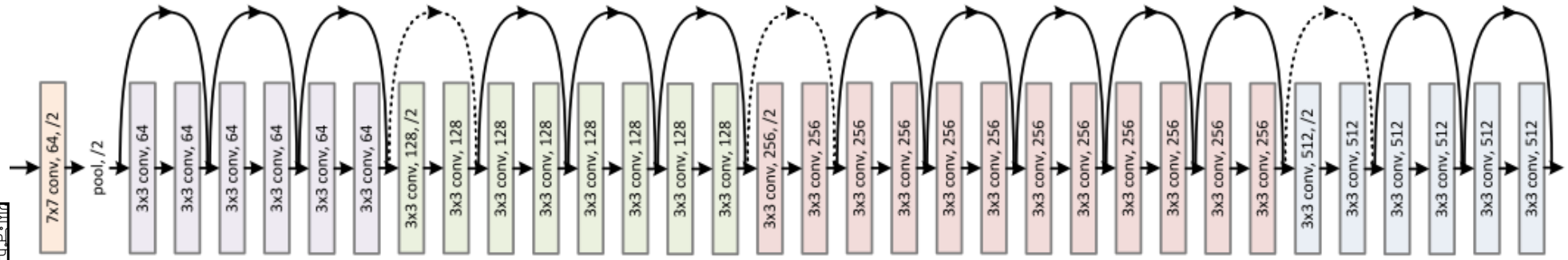
f forget

o output

Long Short-Term Memory (LSTM)

A gradient path w/o matrix multiplication





Data flow

Gates (Control)

x input

h hidden

c Cell, "memory"

g

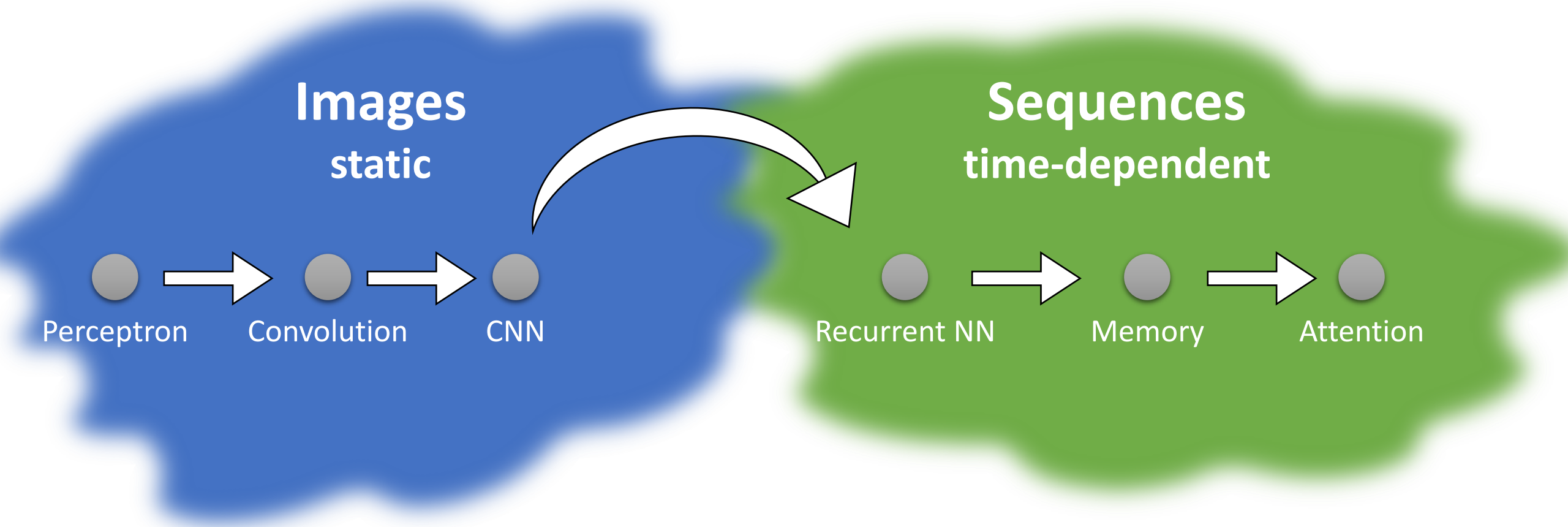
i input

f forget

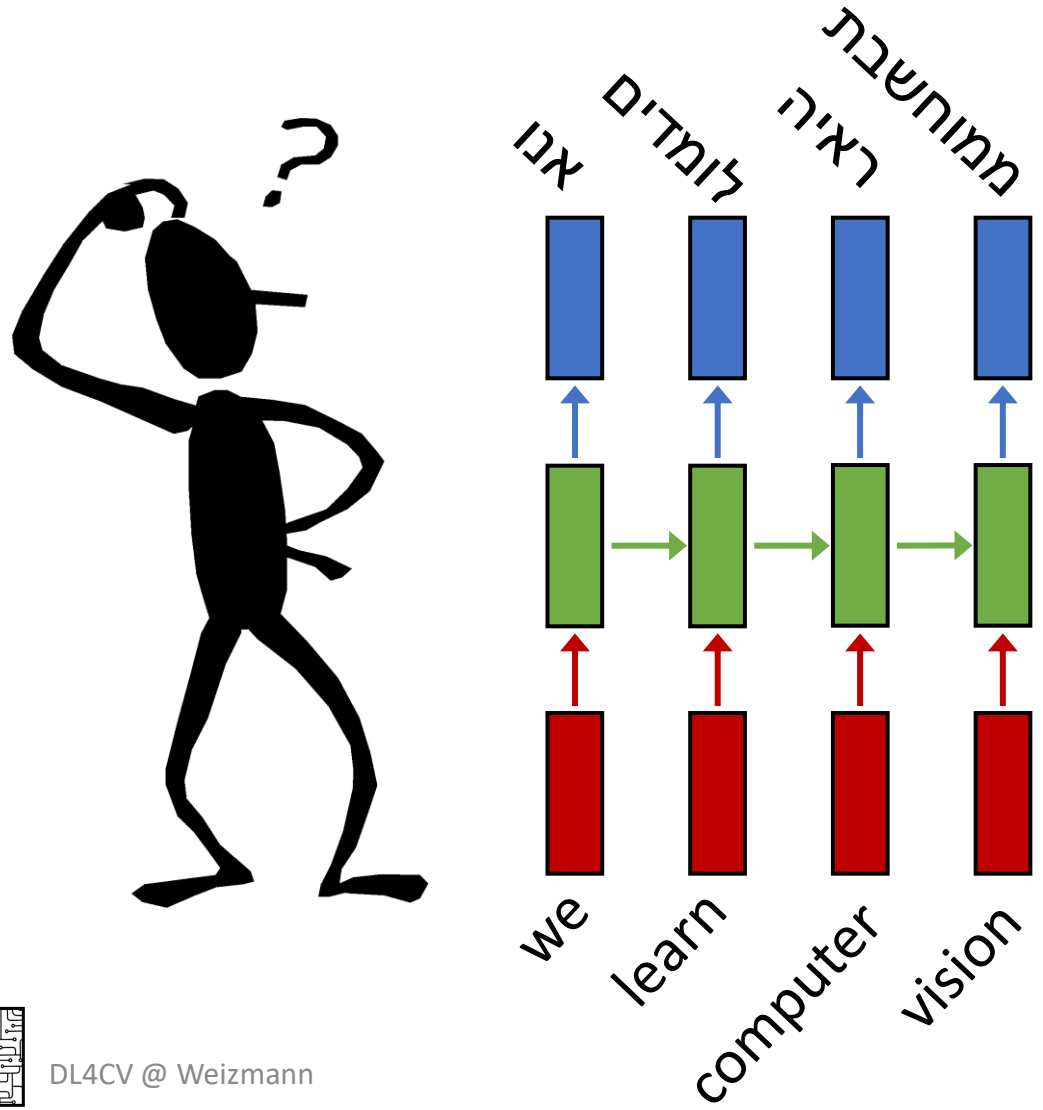
o output

WAIC

Agenda

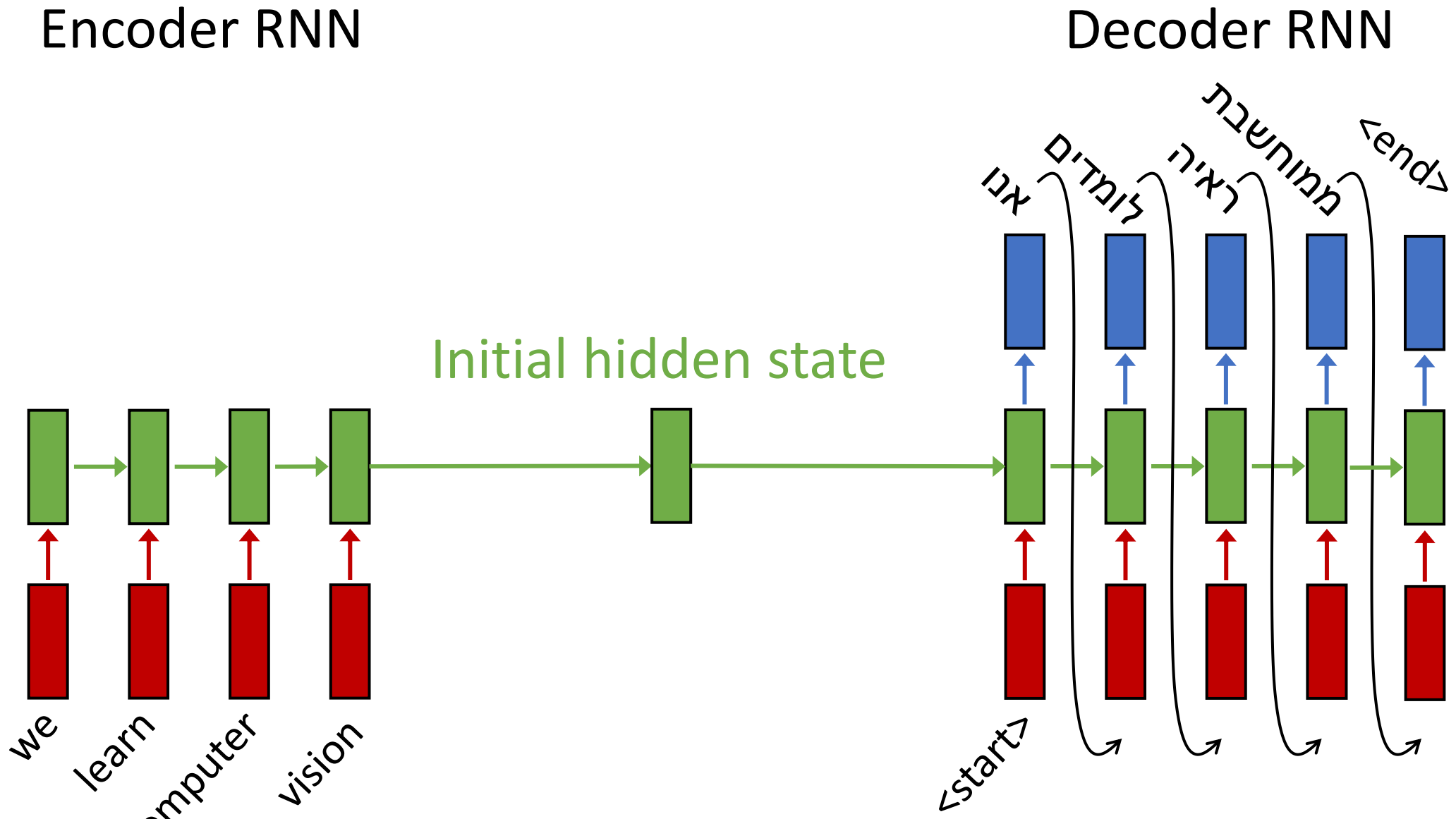

Images
static
Sequences
time-dependent
Perceptron
Convolution
CNN
Recurrent NN
Memory
Attention

Sequence to Sequence



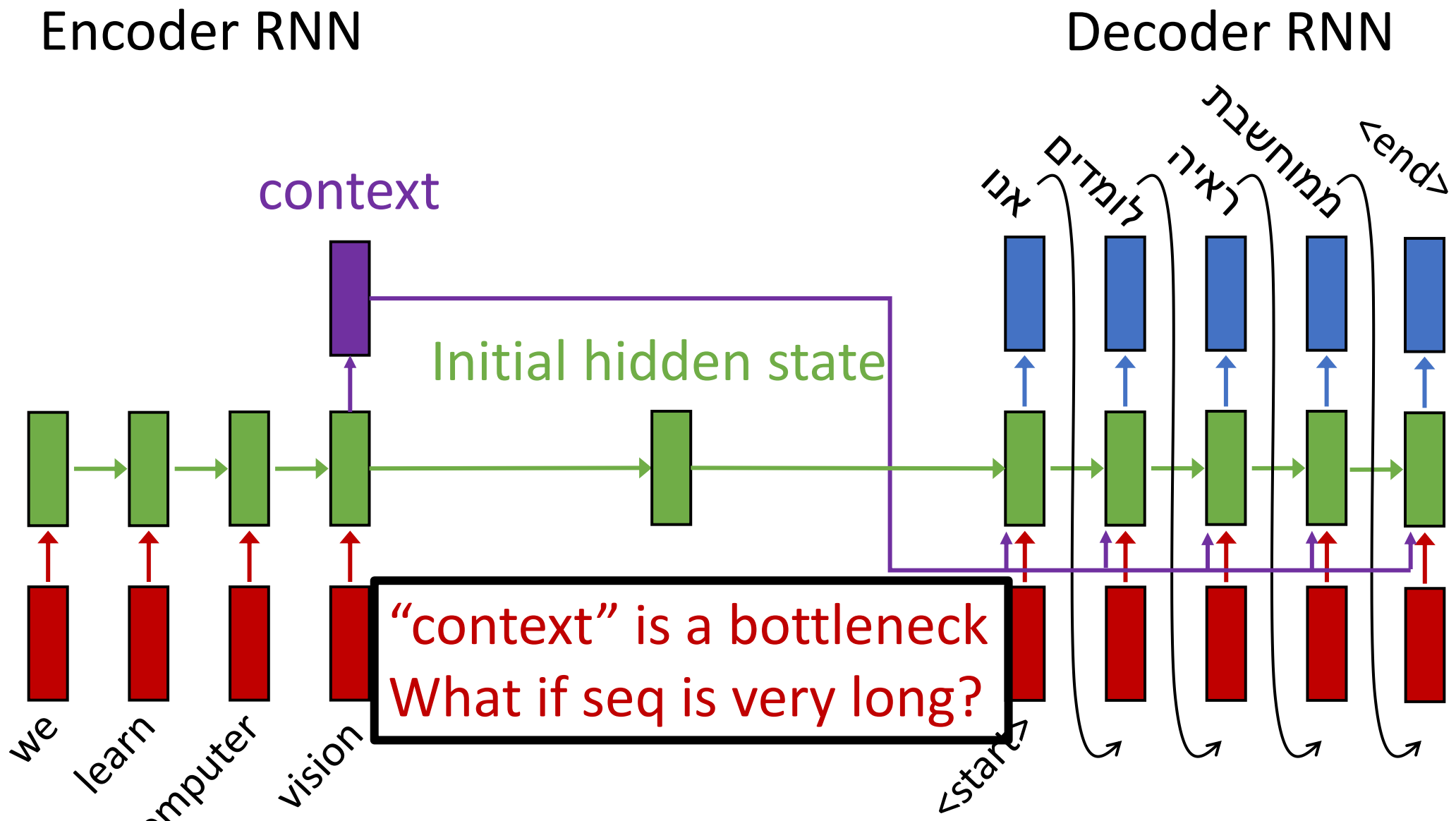
We	אנו (ANU)
Learn	לומדים (LOMDIM)
Computer	ממוחשבת (MEMUCHSHEVET)
Vision	ראיה (RE’EYA)

Sequence to Sequence: RNN



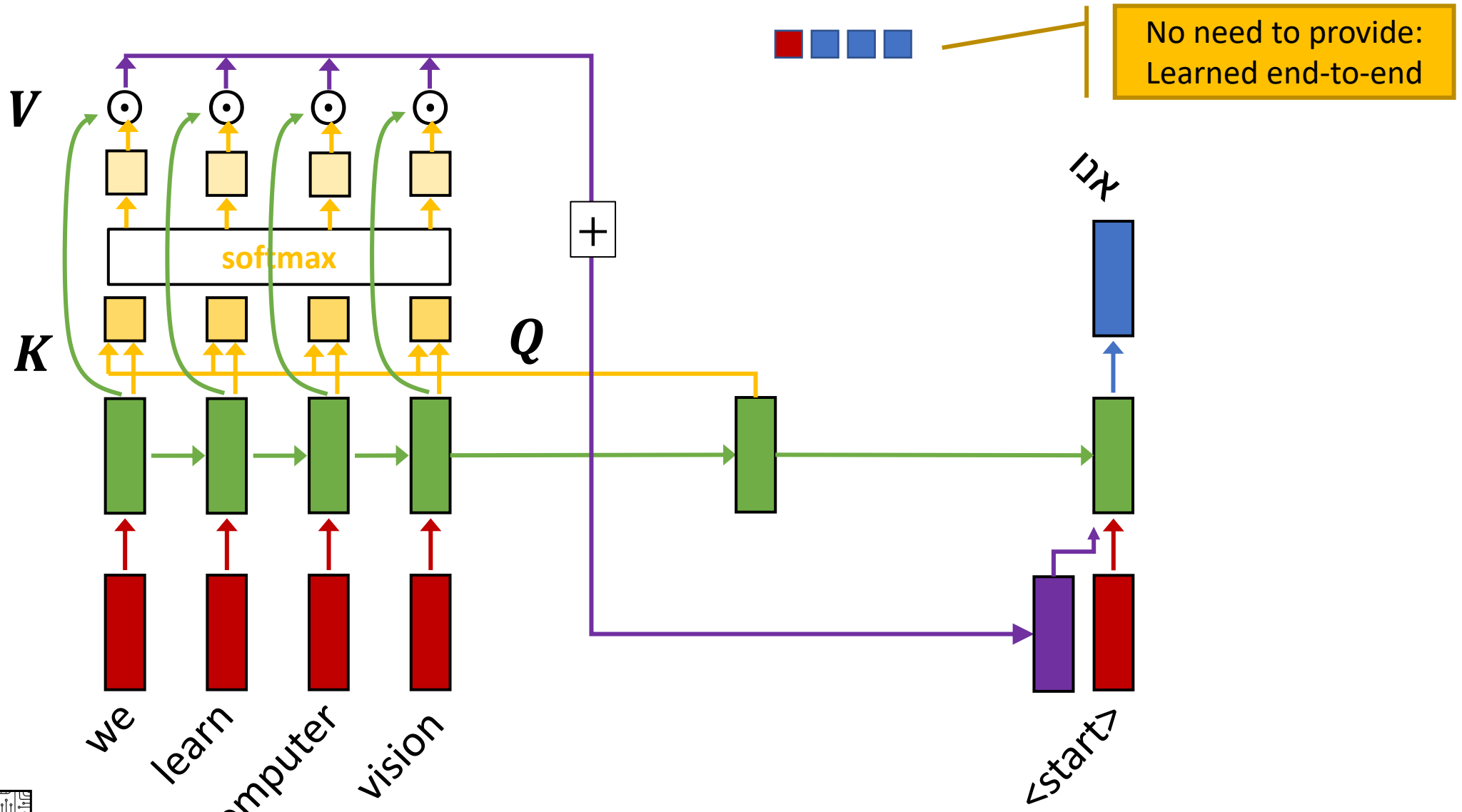
Slide credit: Justin Johnson (EECS-498-007, UMich)

Sequence to Sequence: RNN



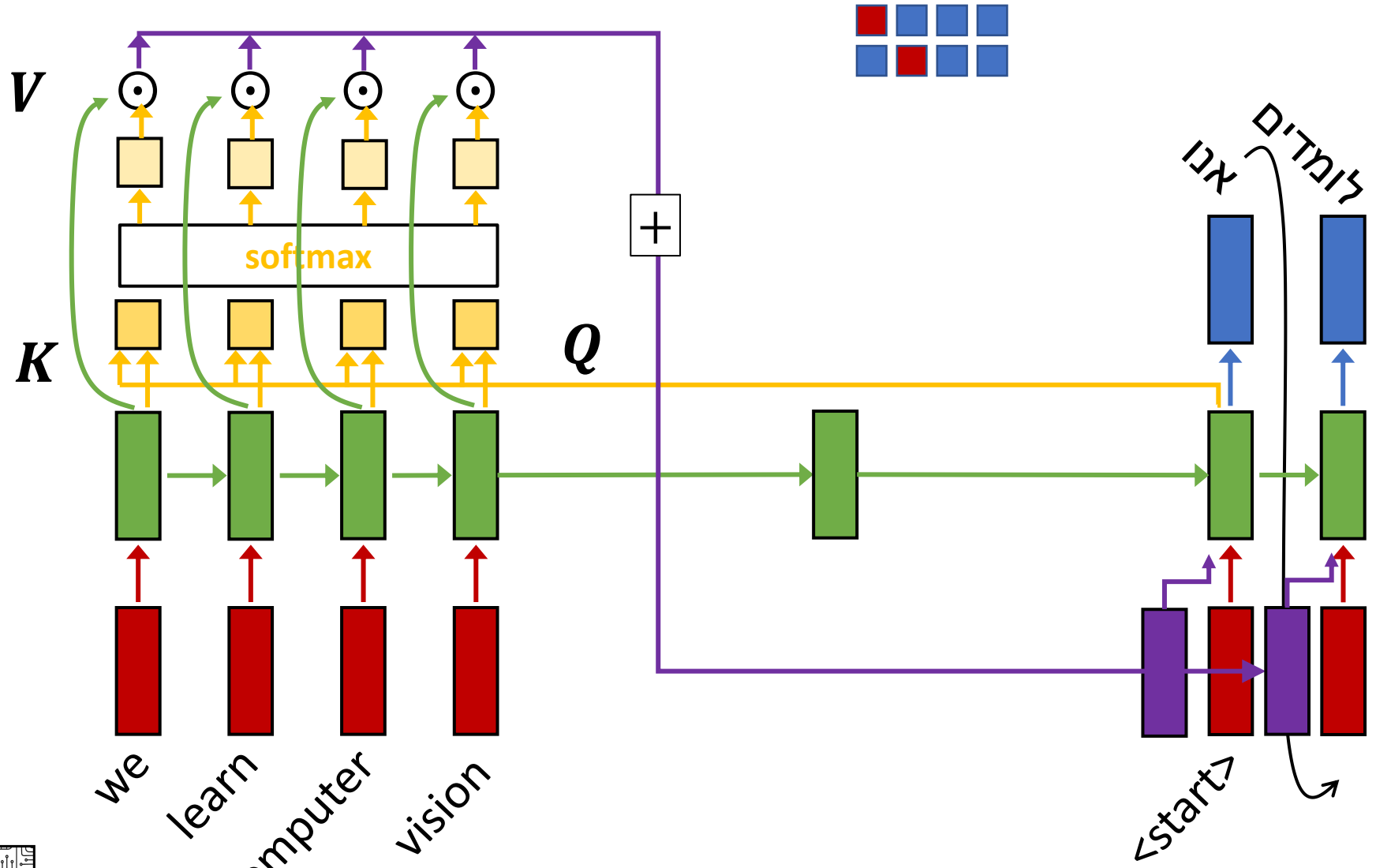
Slide credit: Justin Johnson (EECS-498-007, UMich)

Sequence to Sequence: RNN & Attention



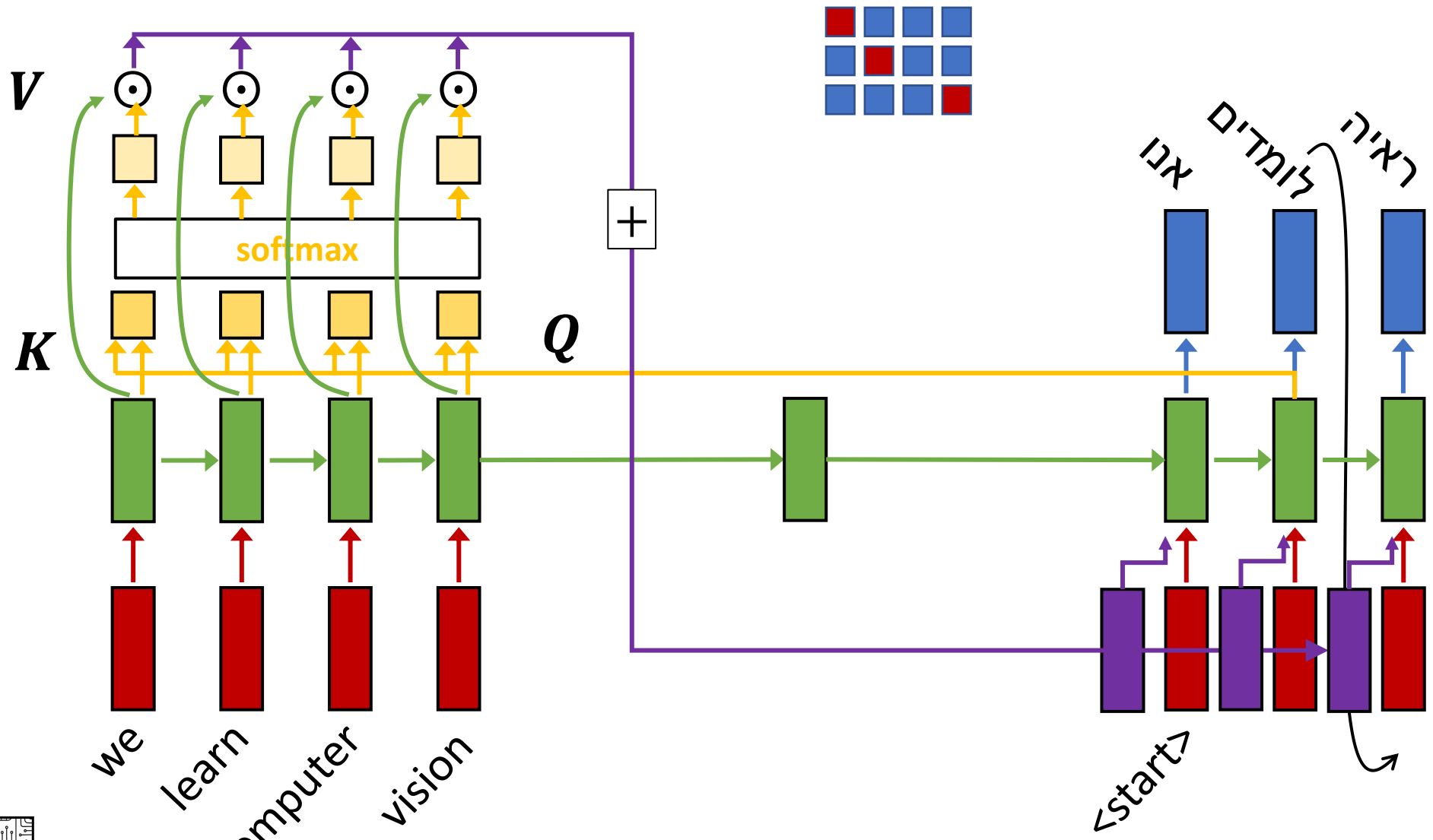
Slide credit: Justin Johnson (EECS-498-007, UMich)

Sequence to Sequence: RNN & Attention



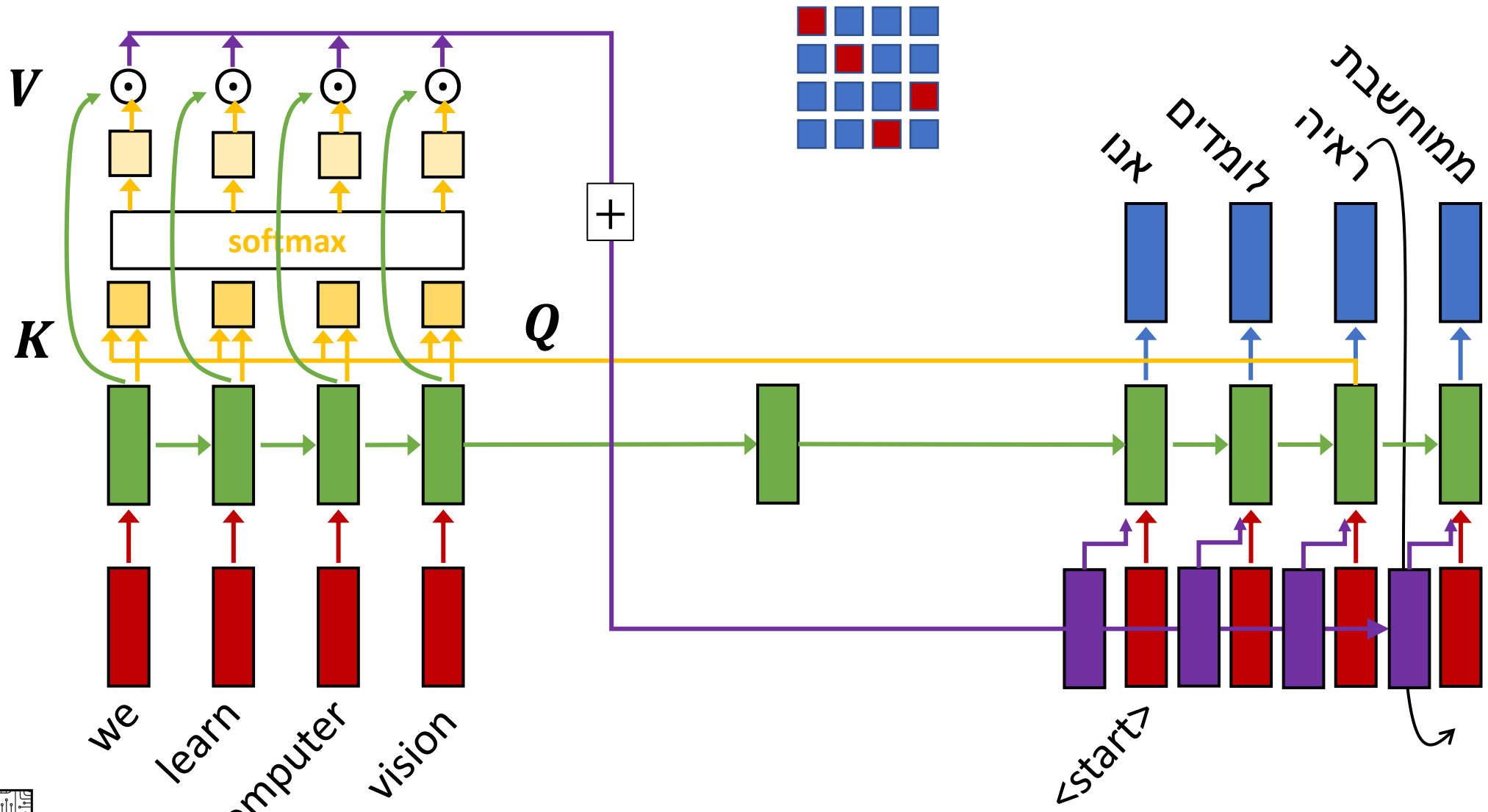
Slide credit: Justin Johnson (EECS-498-007, UMich)

Sequence to Sequence: RNN & Attention



Slide credit: Justin Johnson (EECS-498-007, UMich)

Sequence to Sequence: RNN & Attention



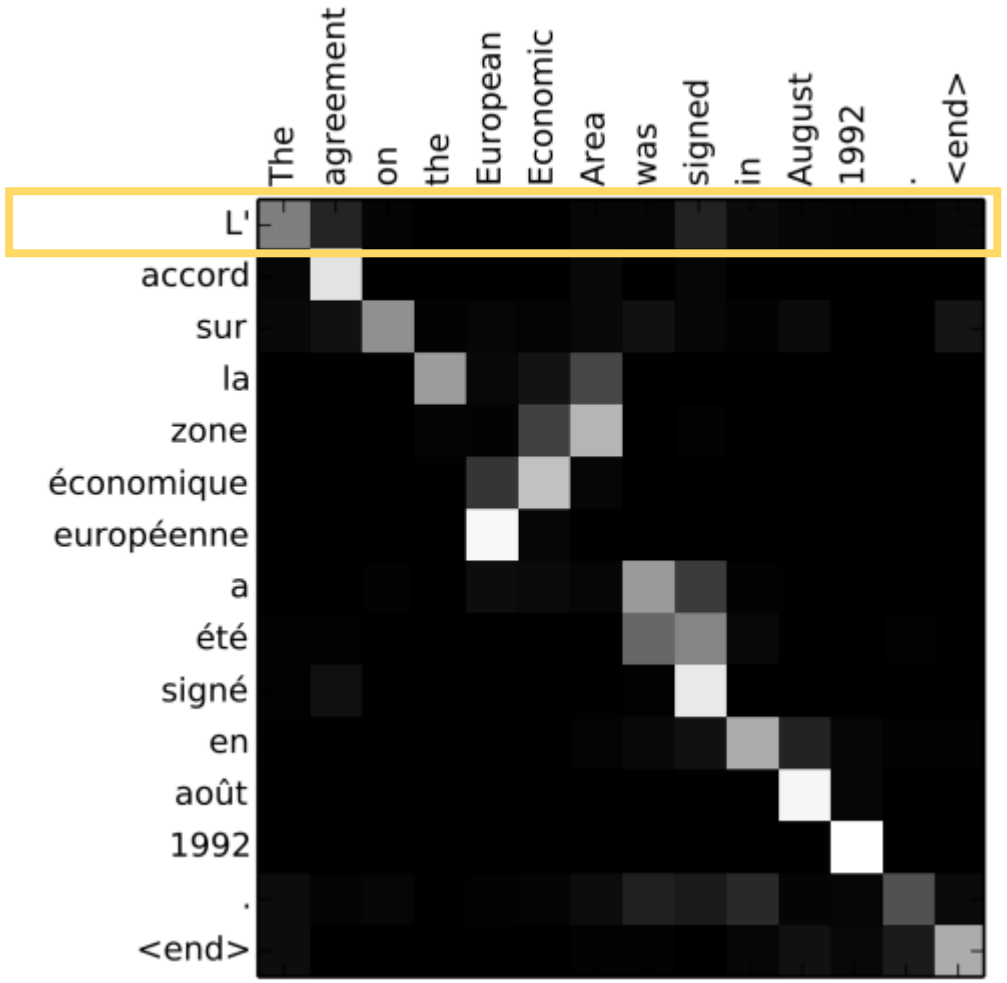
Slide credit: Justin Johnson (EECS-498-007, UMich)

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.



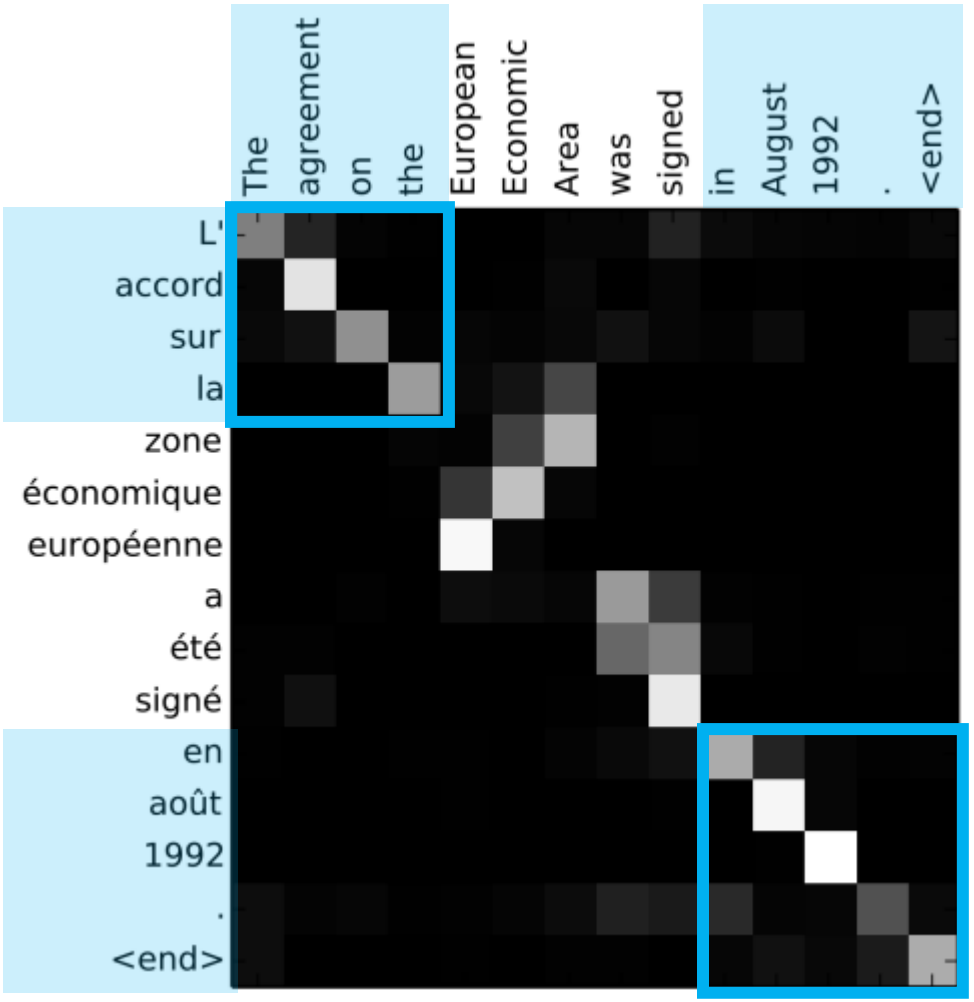
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015
Slide credit: Justin Johnson (EECS-498-007, UMich)

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.**



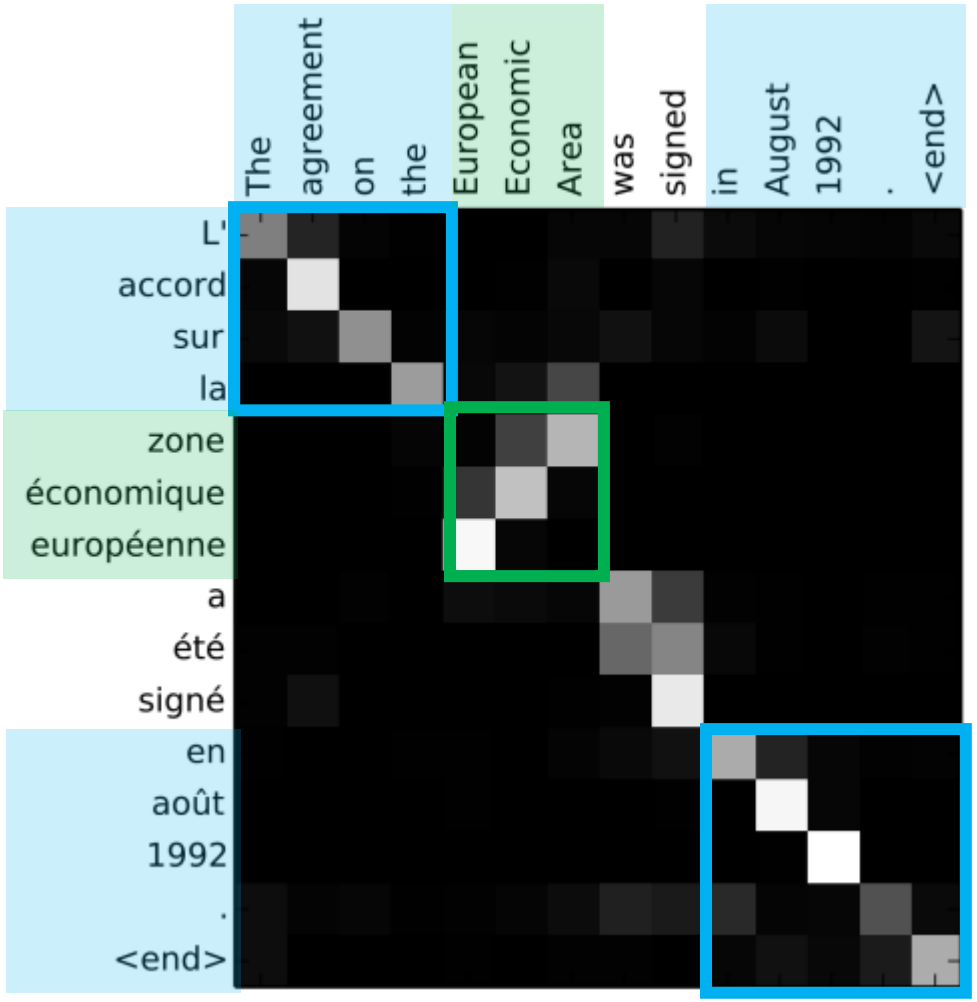
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015
Slide credit: Justin Johnson (EECS-498-007, UMich)

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.



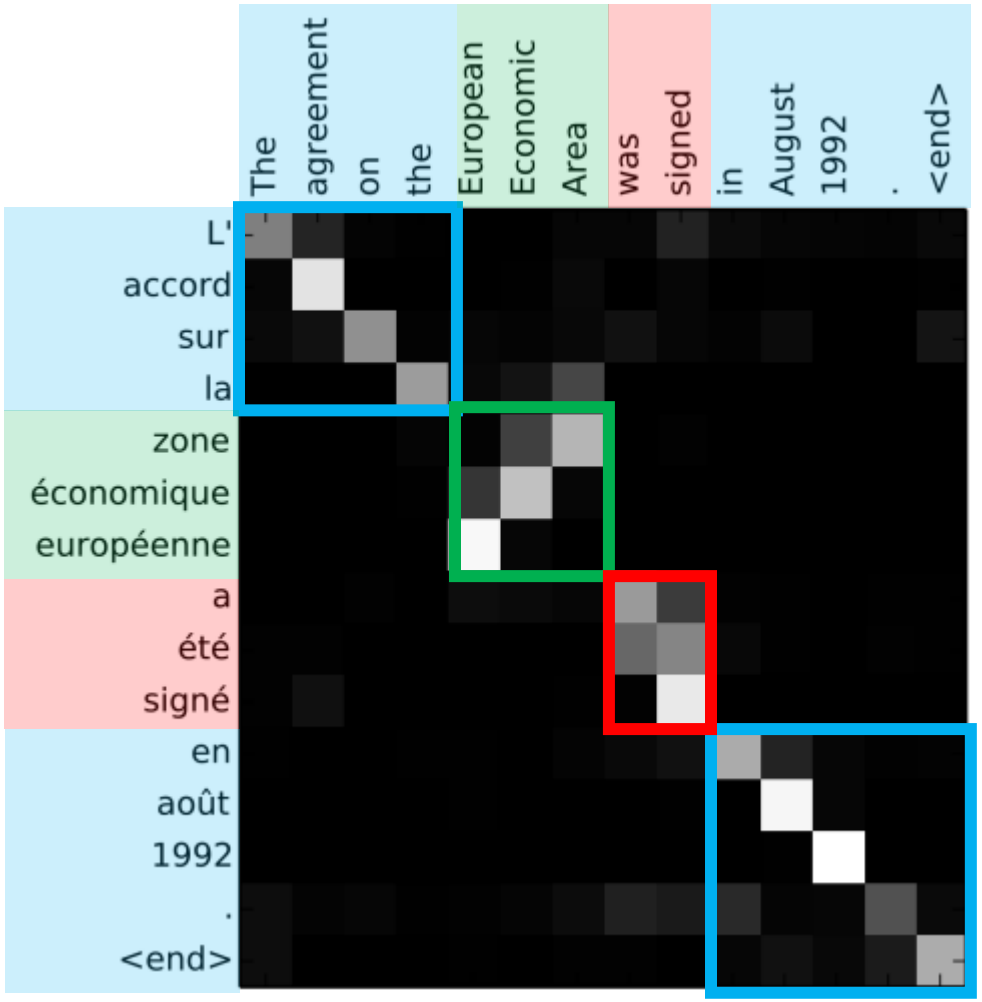
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015
Slide credit: Justin Johnson (EECS-498-007, UMich)

Sequence to Sequence: RNN & Attention

Example: English to French translation

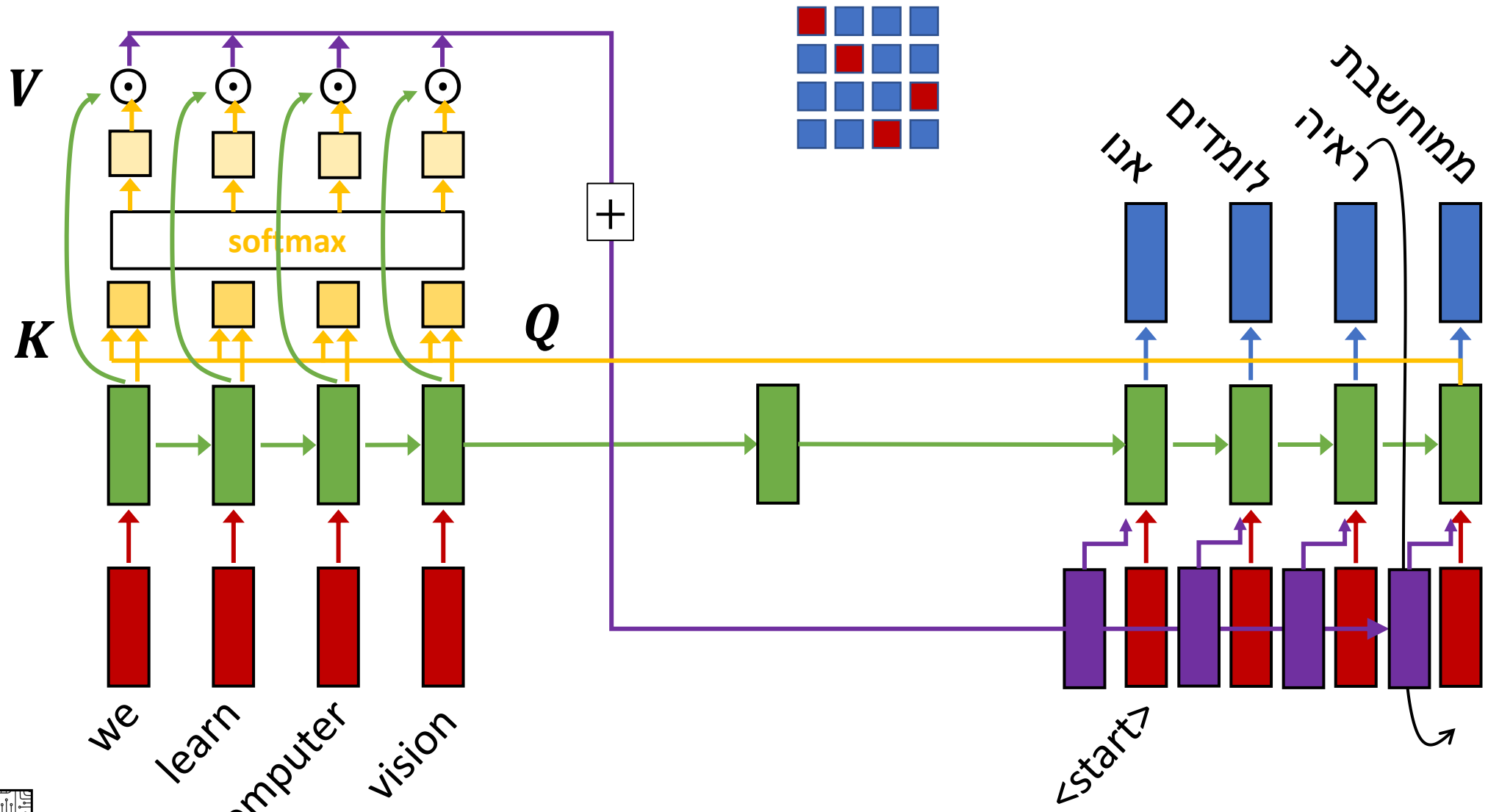
Input (English):
The agreement on the European Economic Area **was signed** in August 1992.

Output (French):
L'accord sur la zone économique européenne **a été signé** en août 1992.



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015
Slide credit: Justin Johnson (EECS-498-007, UMich)

Attention Layer



Slide credit: Justin Johnson (EECS-498-007, UMich)

Attention Layer

Inputs:

Query: $\boldsymbol{Q}$ (shape: $N_q \times D_q$)

Input: $\boldsymbol{X}$ (shape: $N_x \times D_x$)

Layer's Parameters:

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

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

Compute:

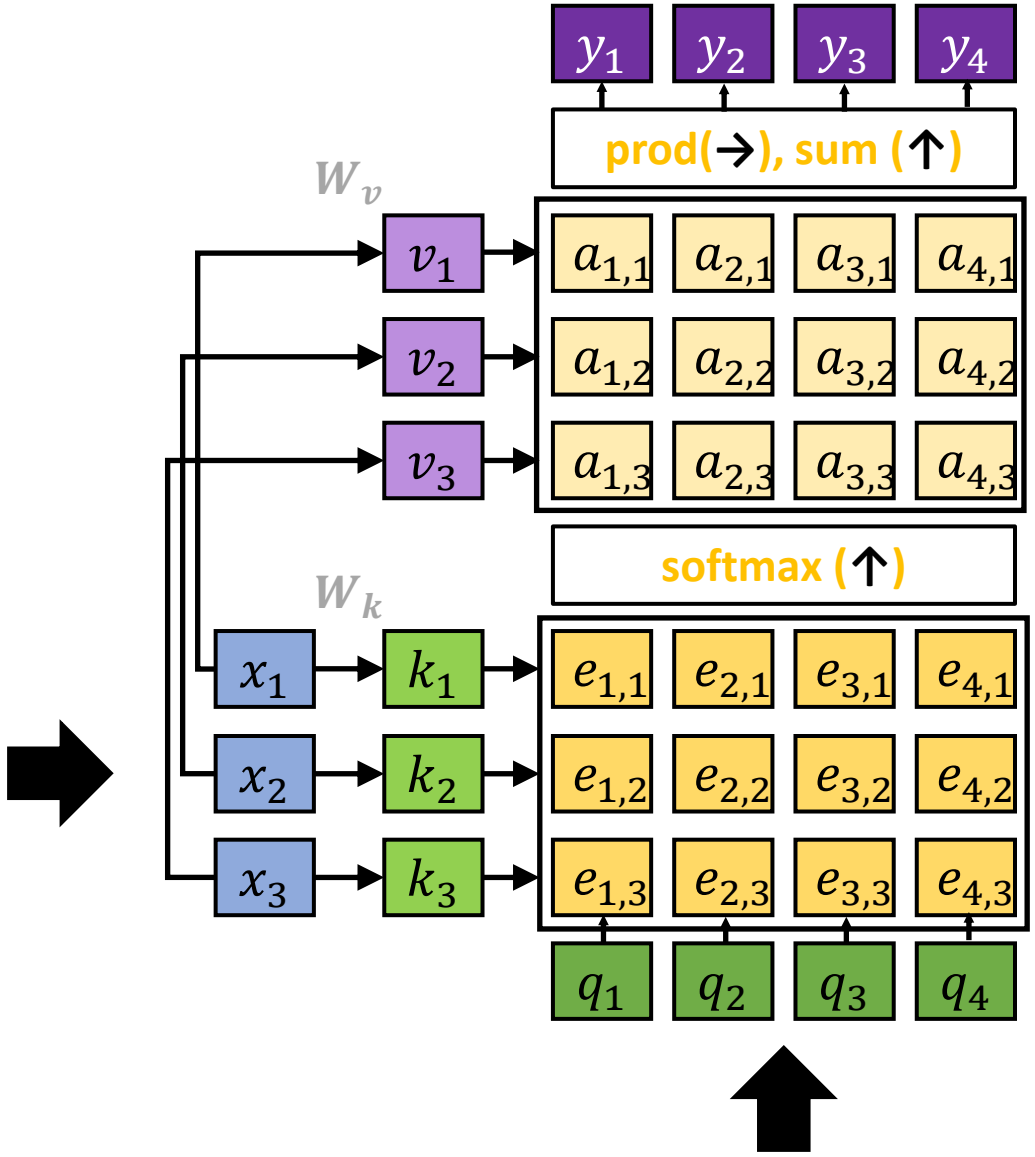
Keys: $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W_k}$ (shape: $N_x \times D_q$)

Values: $\mathbf{V} = \mathbf{X}\boldsymbol{W_v}$ (shape: $N_x \times D_v$)

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

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

Outputs: $\mathbf{Y} = \mathbf{A}\mathbf{V}$ (shape: $N_q \times D_v$)



Slide credit: Justin Johnson (EECS-498-007, UMich)

Self-Attention Layer

Input:

Input: $\boldsymbol{X}$ (shape: $N_x \times D_x$)

Layer's Parameters:

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

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

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

Compute:

Query: $\boldsymbol{Q} = \boldsymbol{X}\boldsymbol{W_q}$ (shape: $N_x \times D_q$)

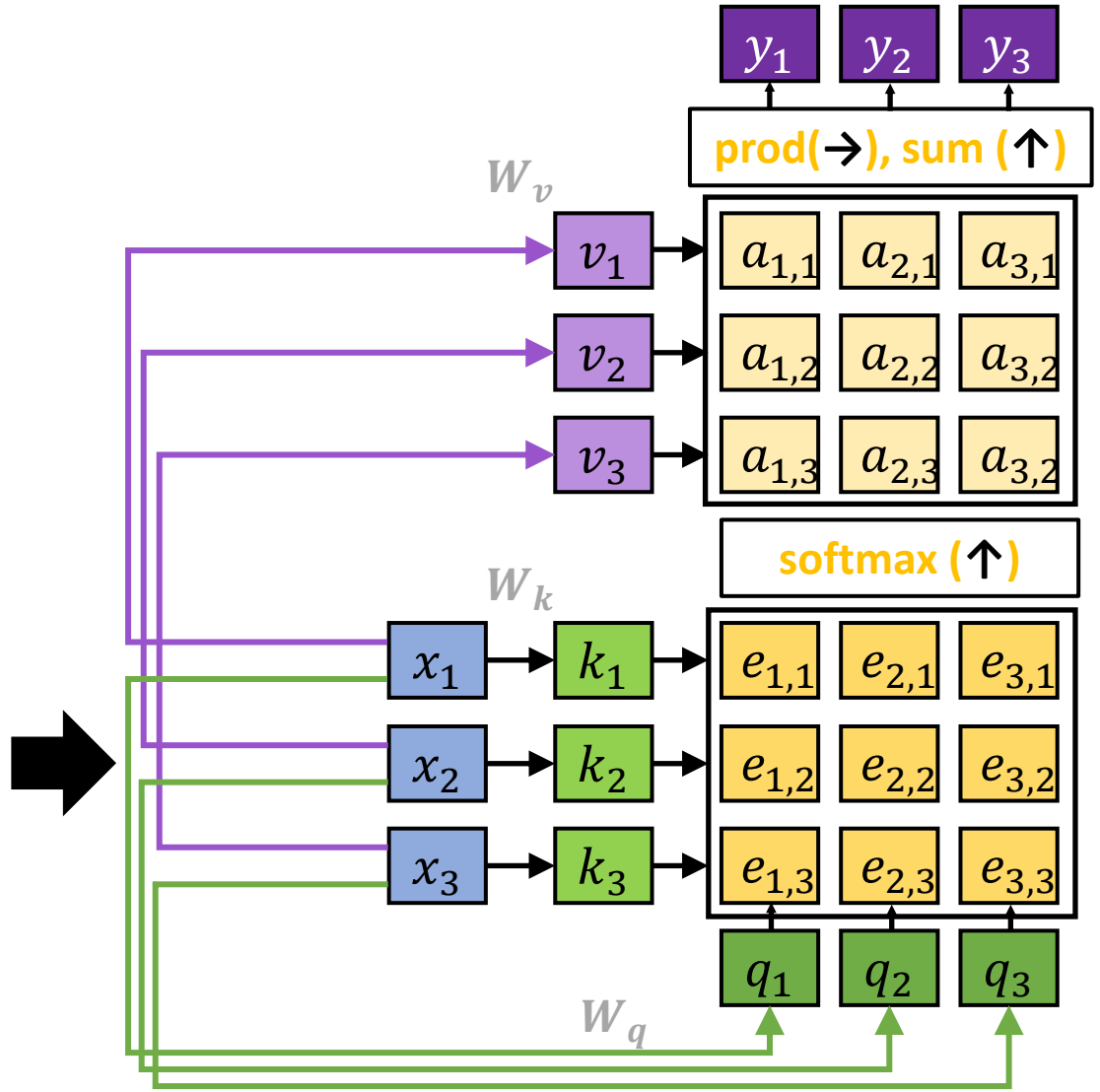
Keys: $\boldsymbol{K} = \boldsymbol{X}\boldsymbol{W_k}$ (shape: $N_x \times D_q$)

Values: $\mathbf{V} = \mathbf{X}\boldsymbol{W_v}$ (shape: $N_x \times D_v$)

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

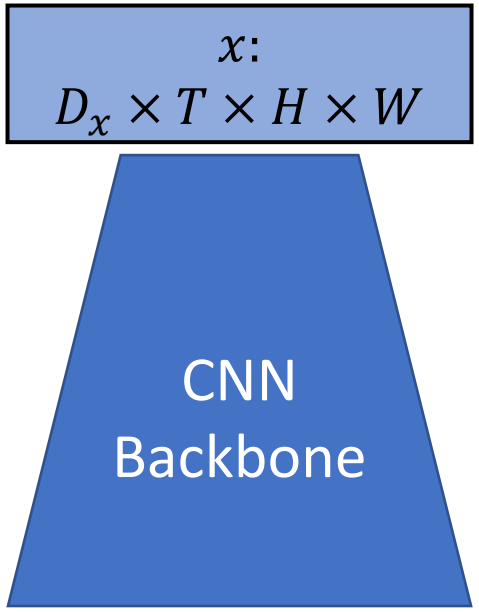
Attention: $\boldsymbol{A} = \mathrm{softmax}(\boldsymbol{E}; \uparrow)$ (shape: $N_q \times N_x$)

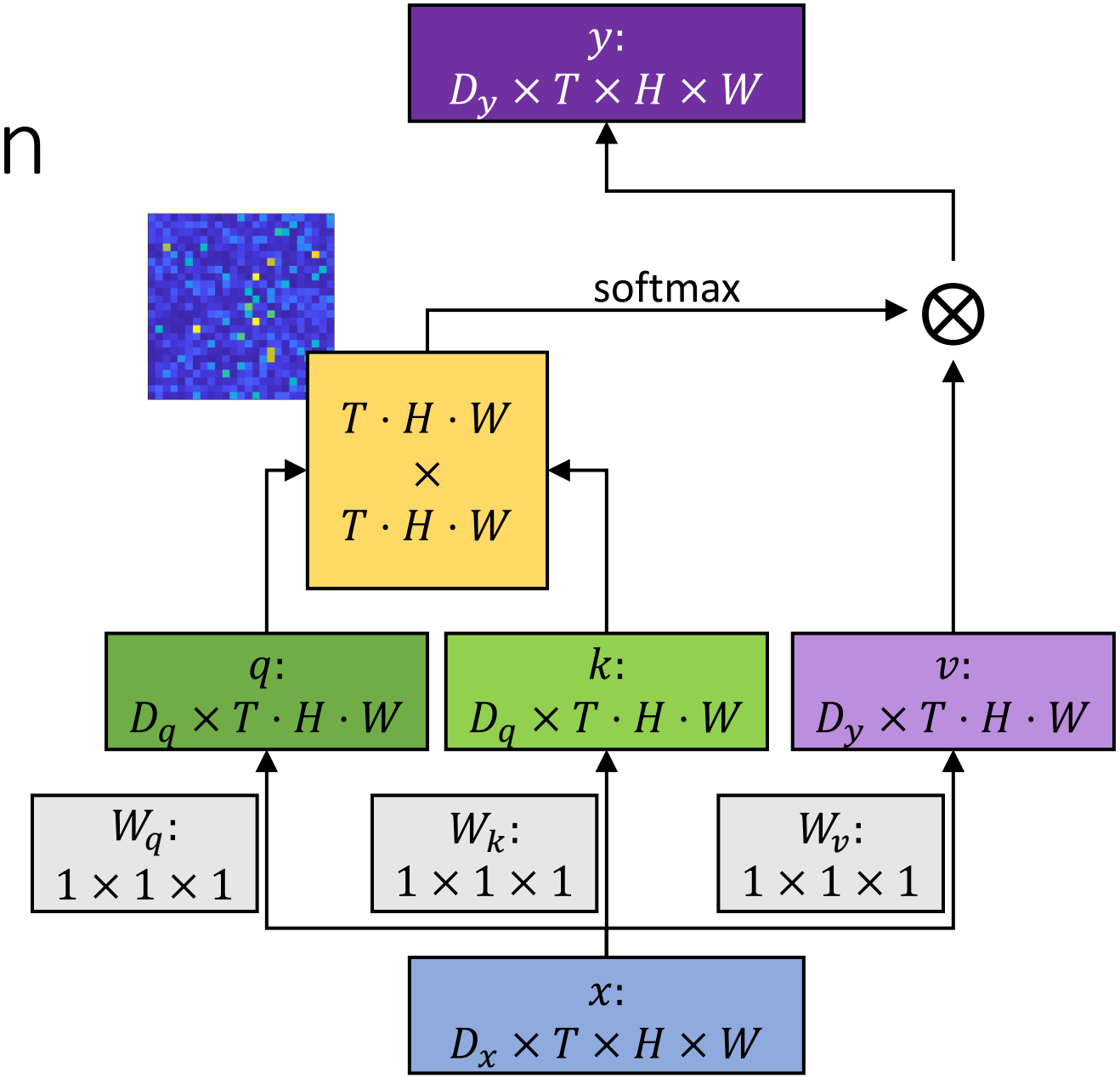
Outputs: $\mathbf{Y} = \mathbf{A}\mathbf{V}$ (shape: $N_q \times D_v$)



Slide credit: Justin Johnson (EECS-498-007, UMich)

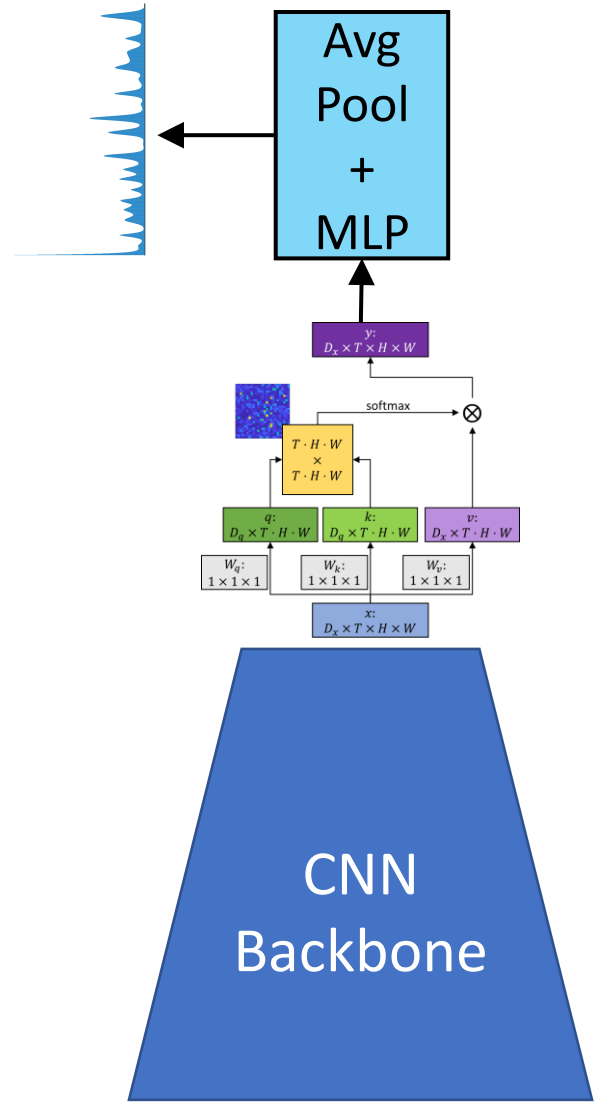
Self Attention in Vision





Wang, X., Girshick, R., Gupta, A. and He, K., "Non-local neural networks" CVPR (2018)

Self Attention in Vision



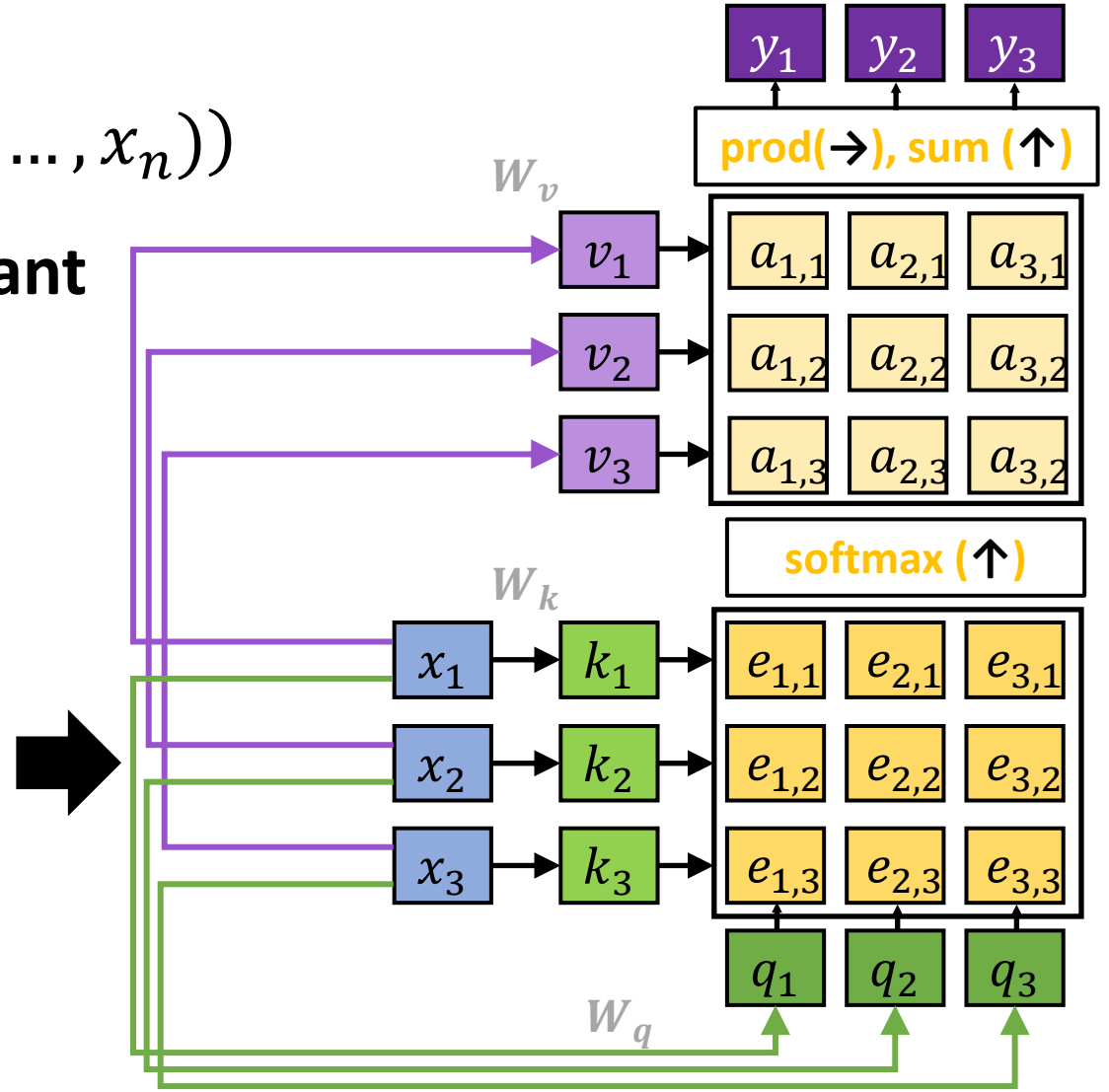
Wang, X., Girshick, R., Gupta, A. and He, K., "Non-local neural networks" CVPR (2018)

Self-Attention Layer: Properties

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

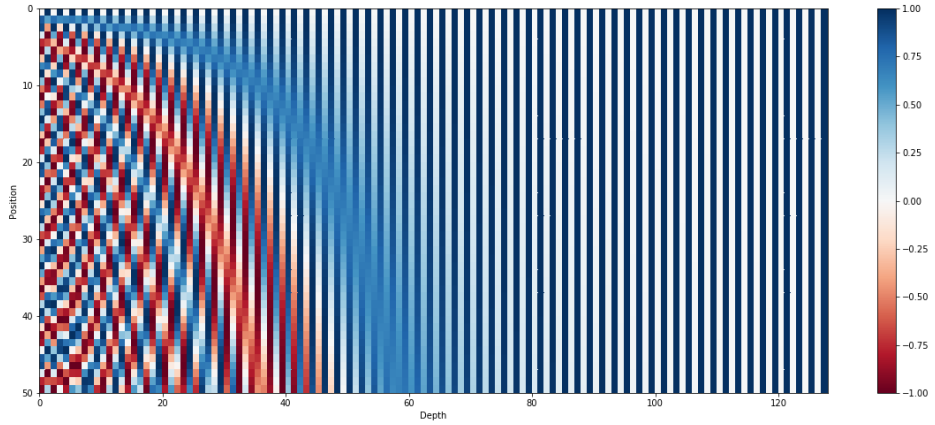
Self-Attention is **permutation equivariant**

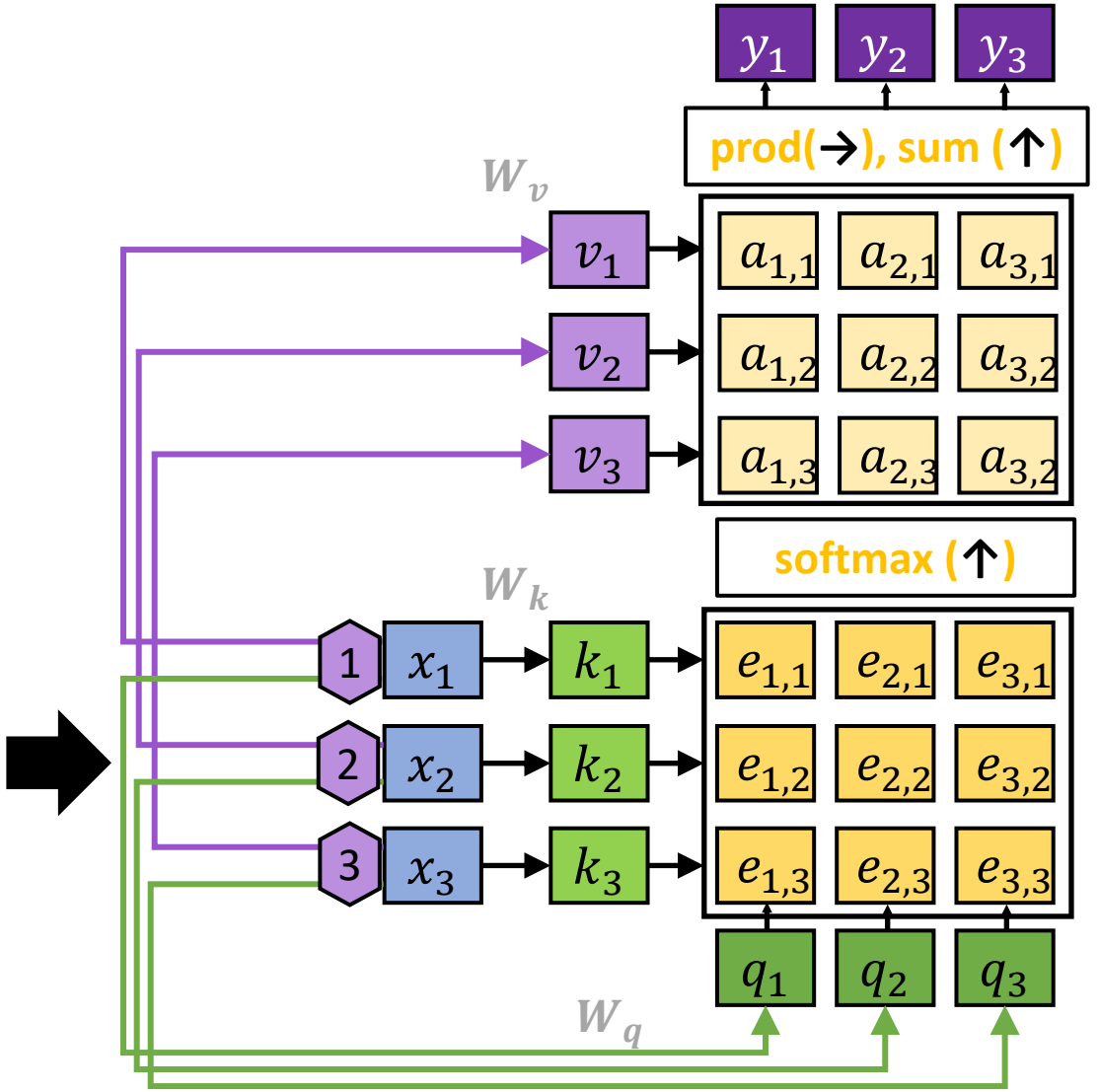
"I am studying" ? "Am I studying"



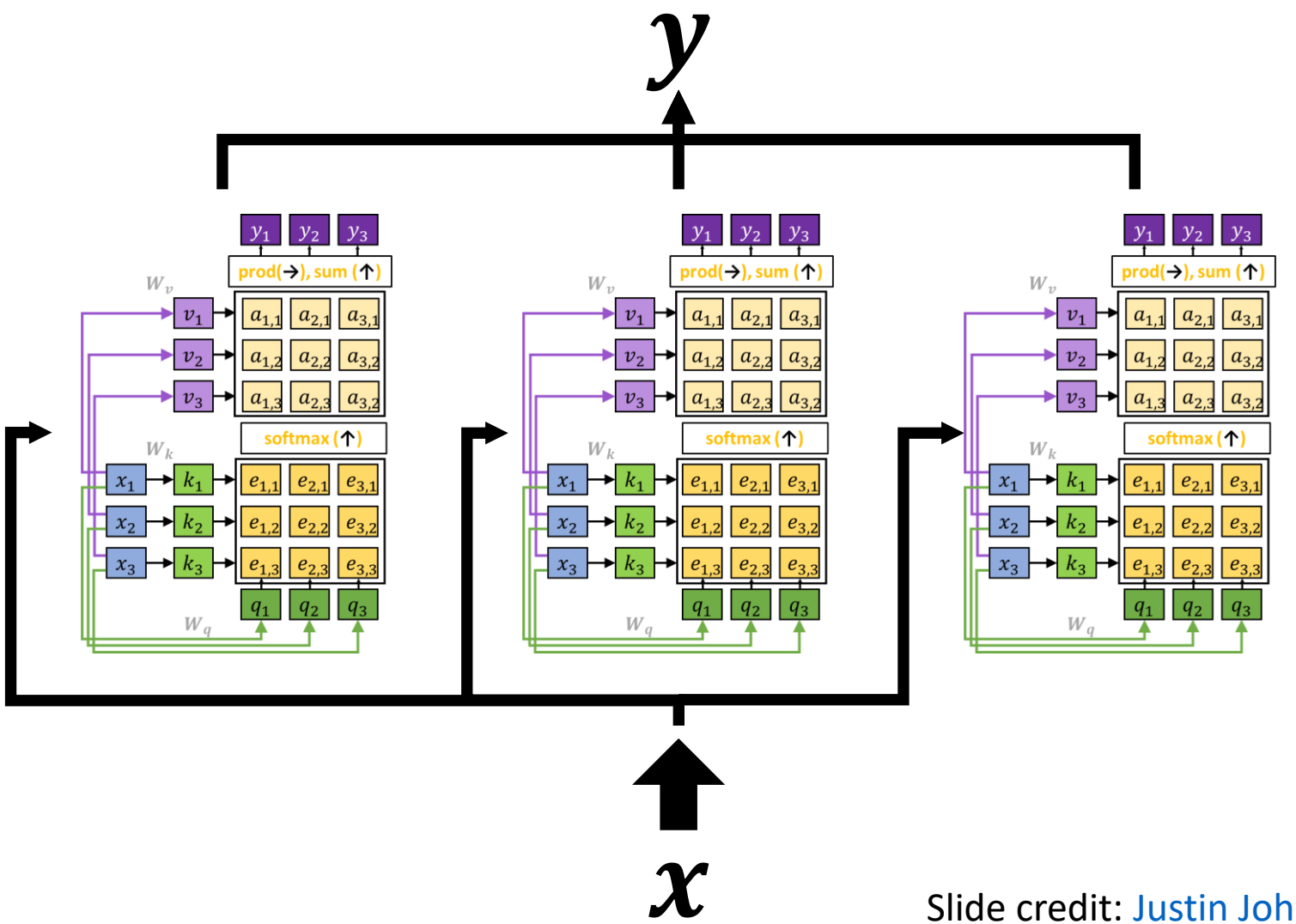
Self-Attention Layer: Properties

Positional Encoding





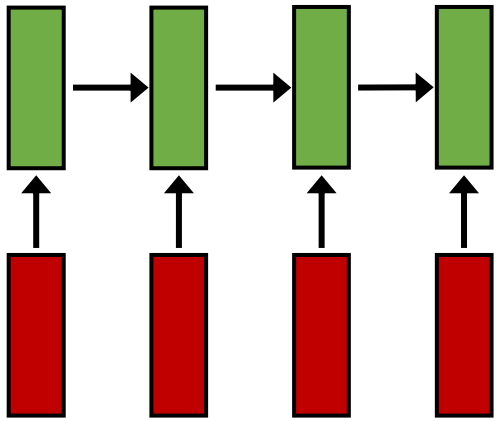
Multi-head Self-Attention



Slide credit: Justin Johnson (EECS-498-007, UMich)

Three Ways of Processing Sequences

Recurrent Neural Network

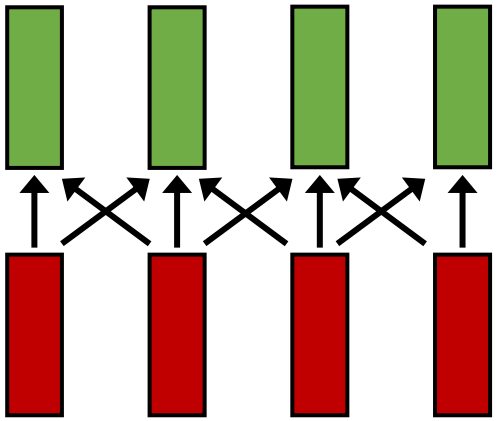


Works on **Ordered Sequences**

(+) large and adaptive receptive field via hidden state

(-) Not parallelizable: need to process states sequentially

1D Convolution

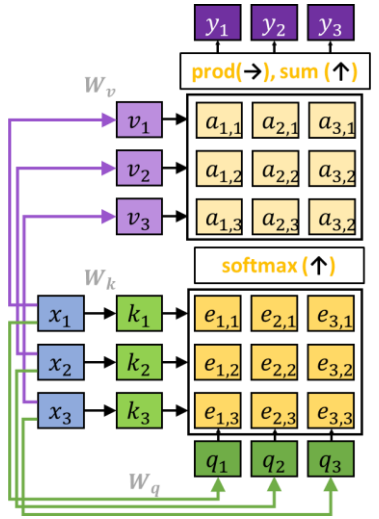


Works on **Multidimensional Grids**

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

(+) Highly parallelizable

Self-Attention



Works on **Sets**

(+) receptive filed = entire sequence

(+) parallelizable

(-) Very memory intensive

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

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 lukaszkaiser@google.com

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

NeurIPS 2017

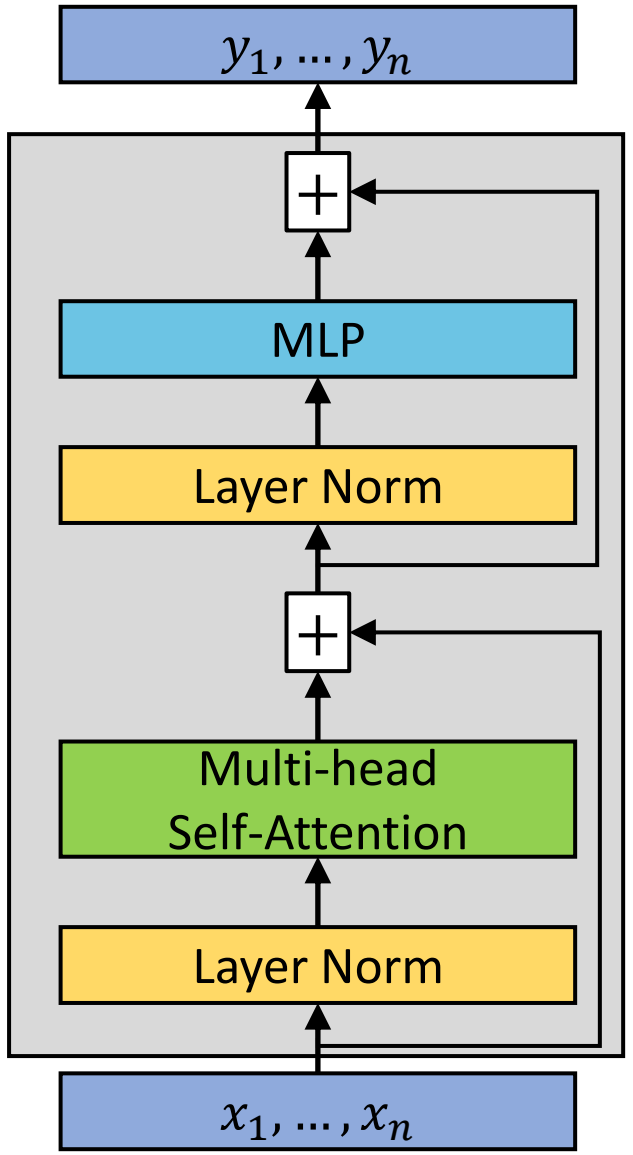
Slide credit: Justin Johnson (EECS-498-007, UMich)

Transformer Layer

Input: $x_1, \ldots, x_n$ (n tokens in D dimensions)
Output: $y_1, \ldots, y_n$ (n tokens in D dimensions)

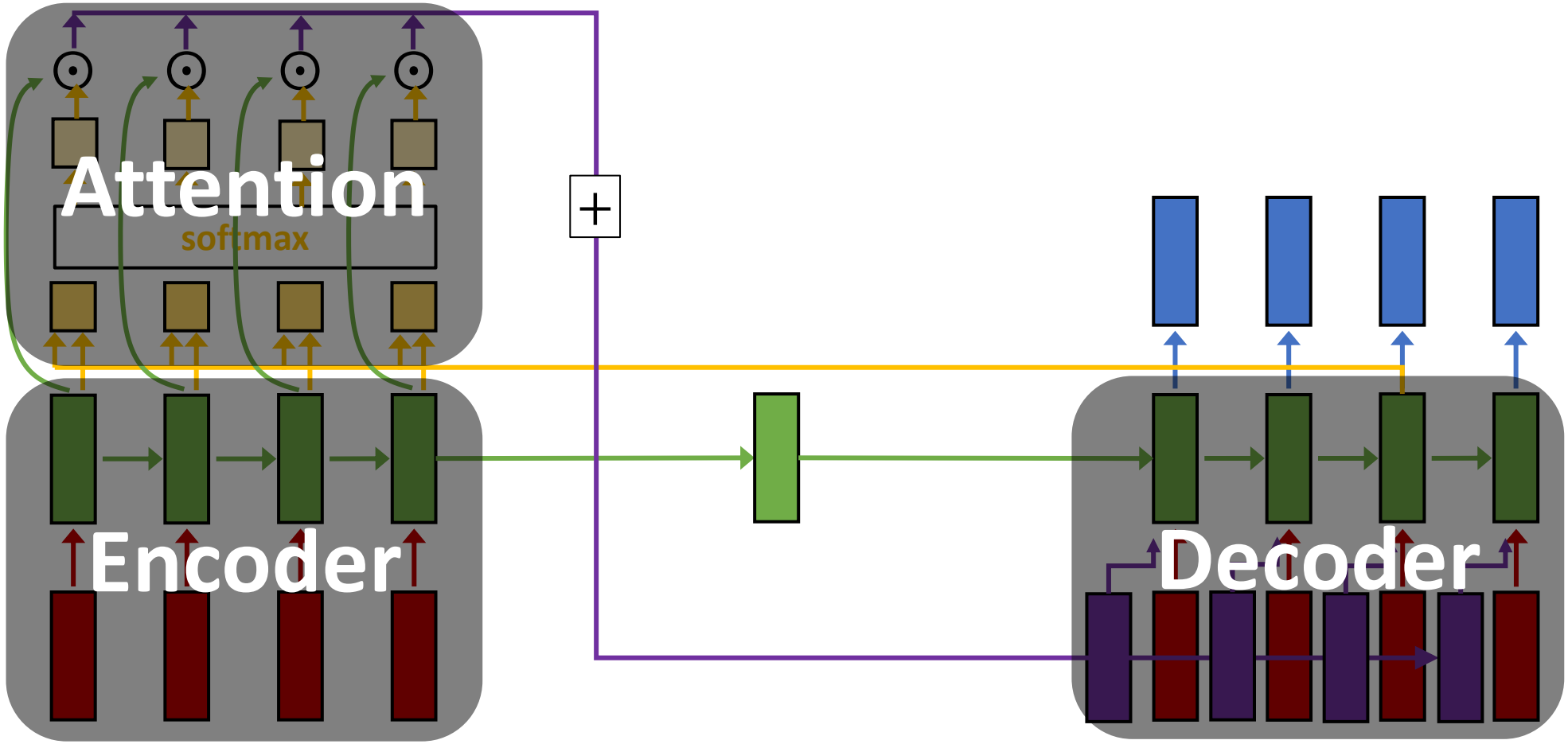
Highly scalable

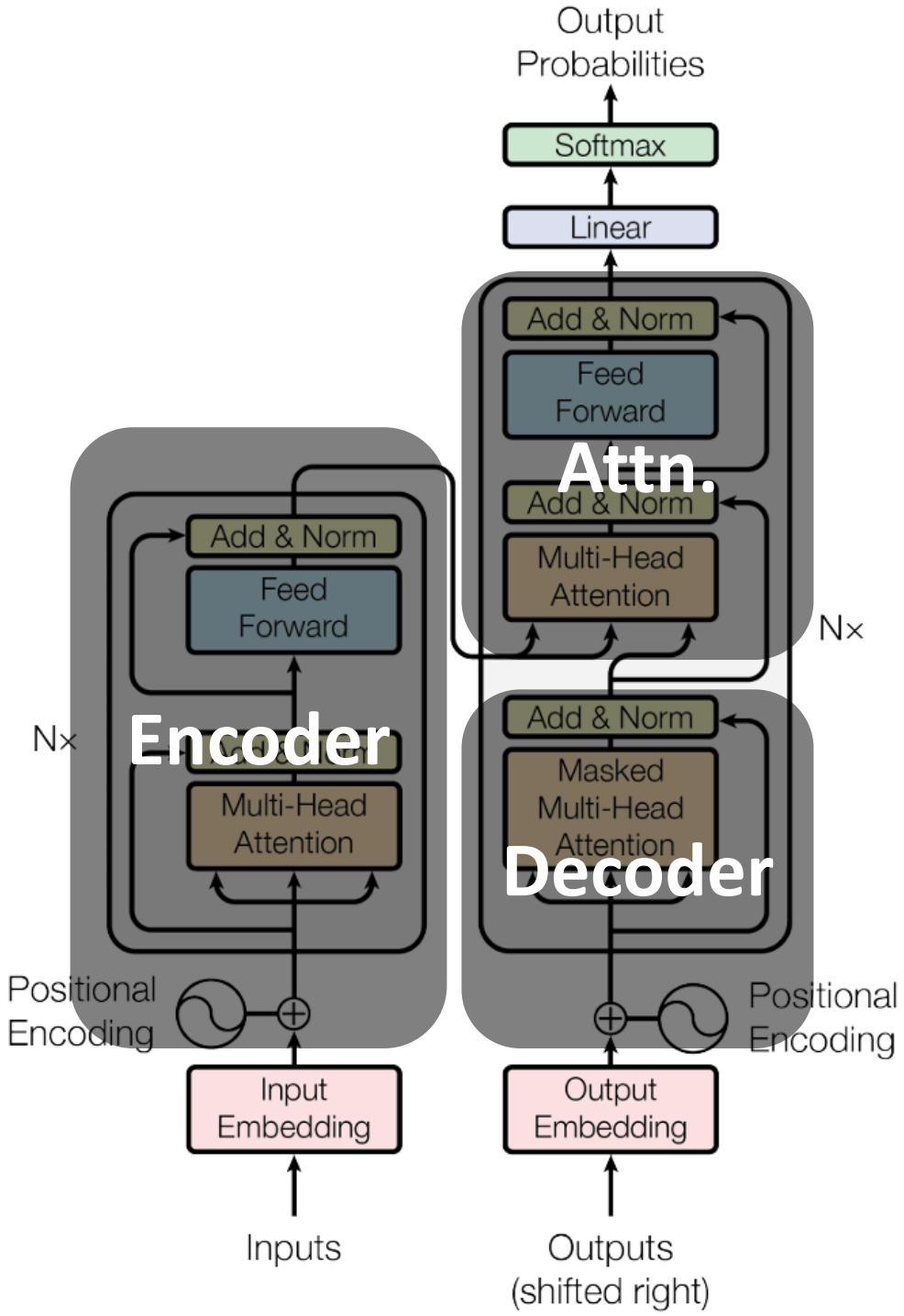
Highly parallelizable



Slide credit: Justin Johnson (EECS-498-007, UMich)

Sequence to Sequence




Output Probabilities
Softmax
Linear
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Attn.
Nx
Add & Norm
Masked Multi-Head Attention
Decoder
Add & Norm
Feed Forward
Add & Norm
Multi-Head Attention
Nx
Encoder
Positional Encoding
Positional Encoding
Input Embedding
Output Embedding
Inputs
Outputs (shifted right)

Transformers Network

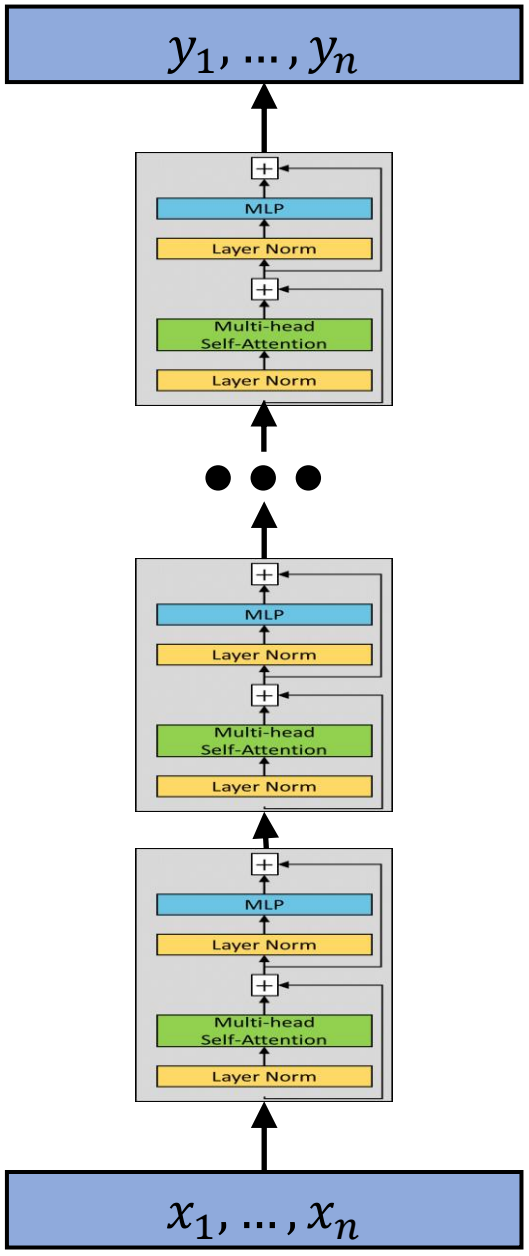
Pretraining:
Download a LOT of text from the internet

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

What’s next?

Next lecture:

Vision Transformers – ViT (Shai)