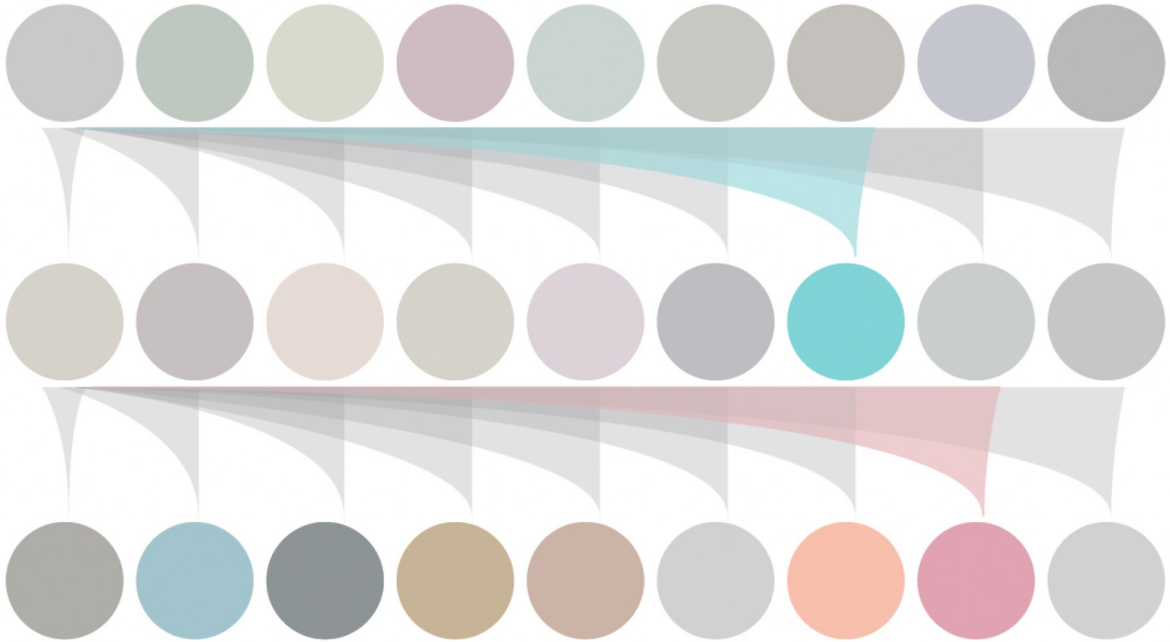
Lecture 11

Sequence Modeling



Sara Beery, Kaiming He, Mina Konaković Luković, Vincent Sitzmann

Overview

- Sequence Problems in Vision
- ConvNets for Sequence Problems
- Recurrent Neural Networks
- Attention and Transformers

Sequence Problems in Vision

Video: Sequence of Images

- Video counterparts of 2D problems
  - classification, detection, segmentation, …
- Video-specific problems
  - tracking, temporal reasoning, structure-from-motion, …

Vision-Language Problems

- pixels in, language out
  - captioning, visual question answering (VQA), reasoning, referring…

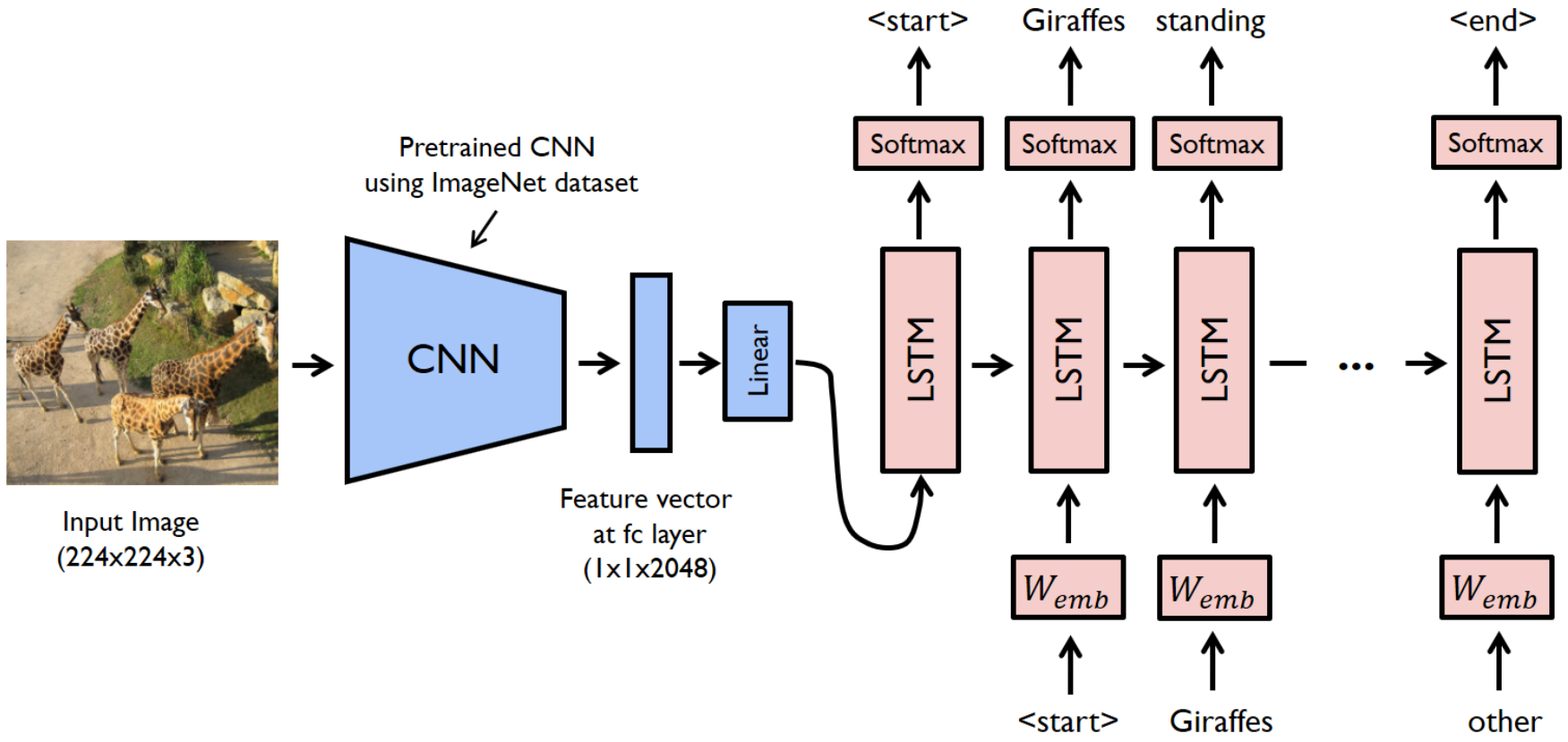


figure credit: https://github.com/yunjey/pytorch-tutorial/tree/master/tutorials/03-advanced/image_captioning

Vision-Language Problems

- language in, pixels out
  - text-to-image, text-to-video, text-to-3D generation

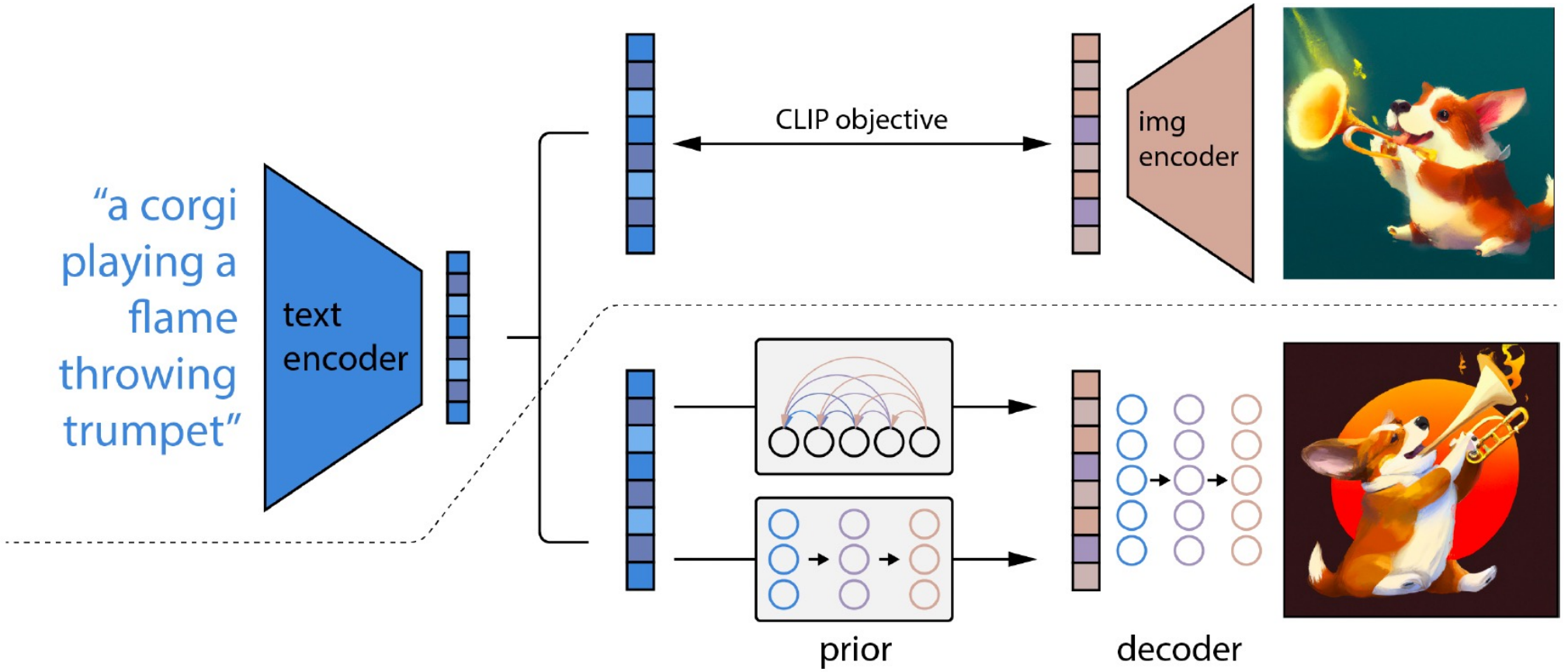
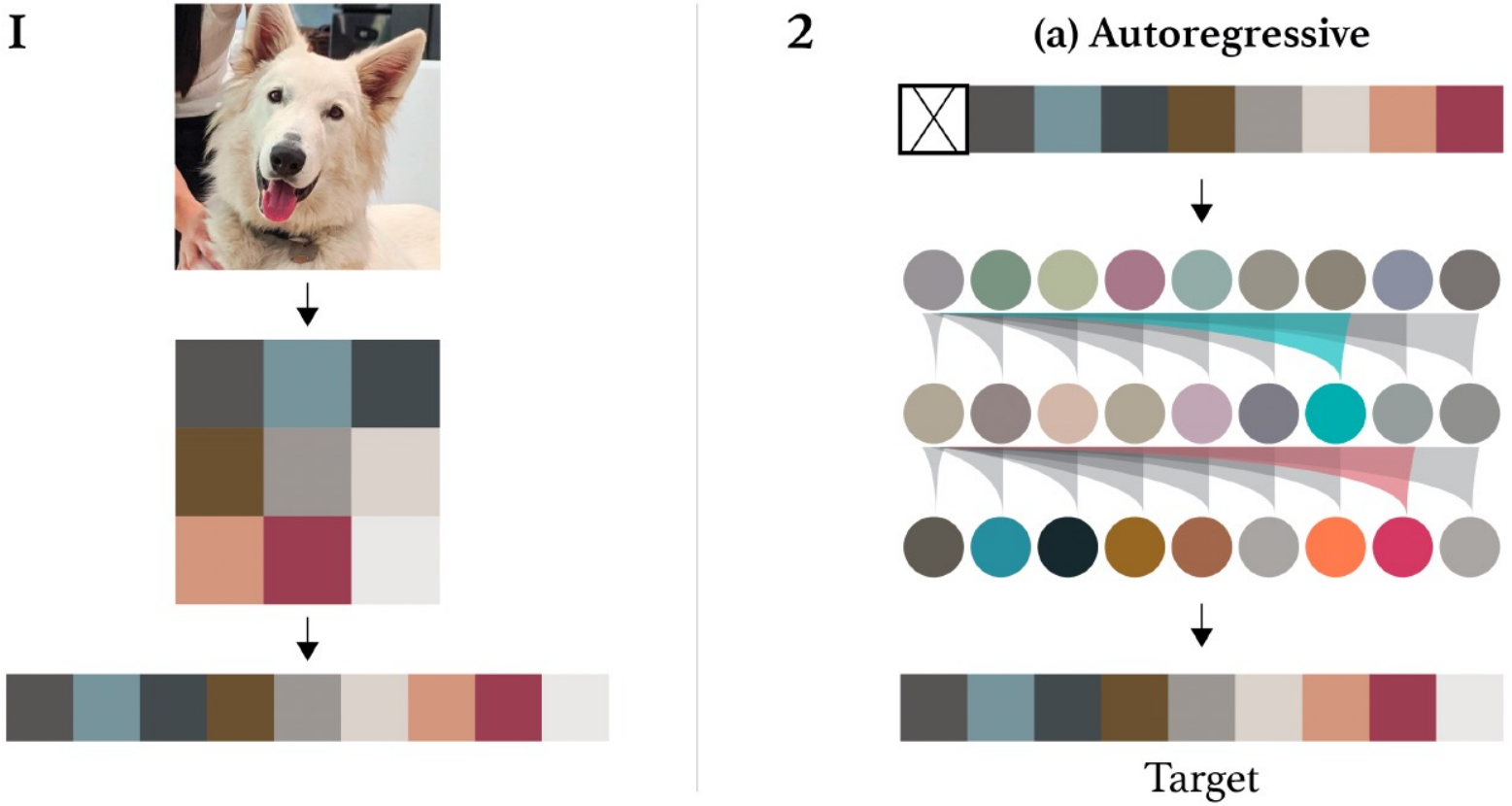


figure credit: DALL-E

Image as a Sequence

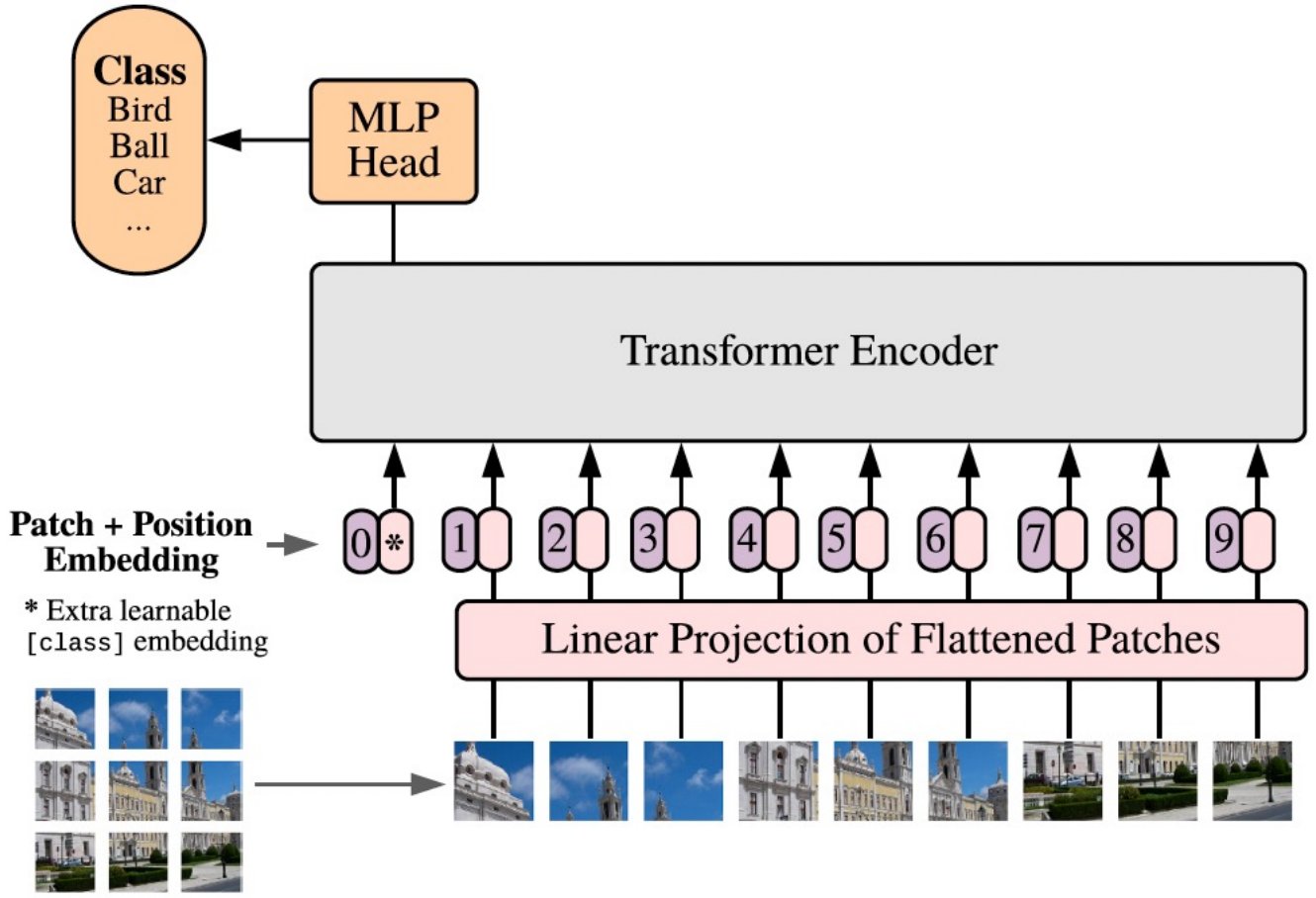
- Sequence of pixels (iGPT)



Chen et al., Generative Pretraining from Pixels, 2020

Image as a Sequence

- Sequence of patches (ViT)



Dosovitskiy, et al., An Image is Worth 16x16 Words- Transformers for Image Recognition at Scale, arXiv 2020, ICLR 2021

Seq2Seq Everywhere

one to one

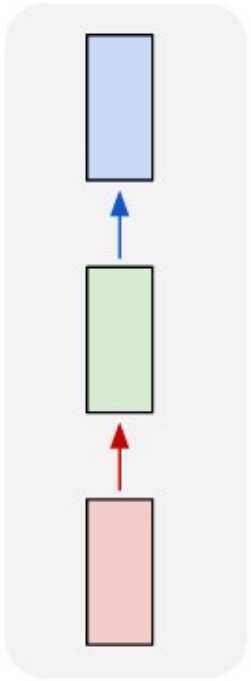


image classification

one to many

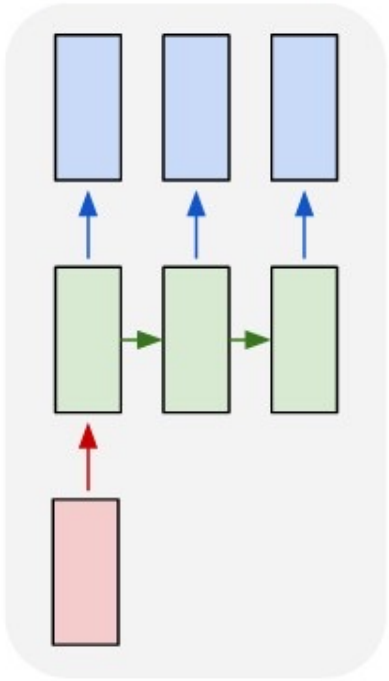
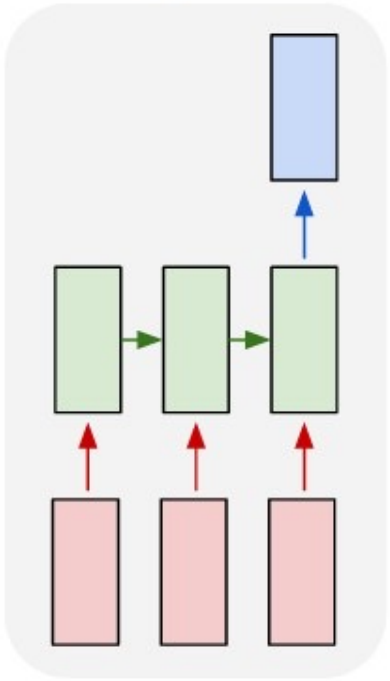


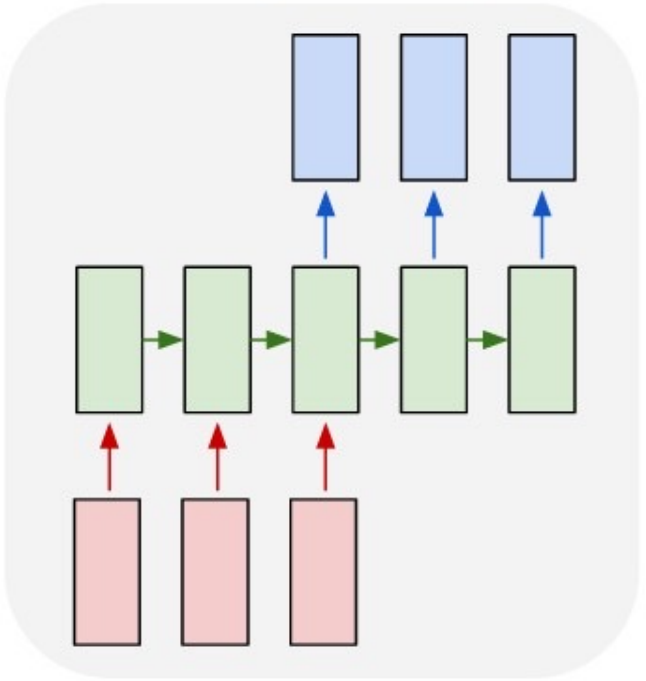
image captioning
object detection

many to one



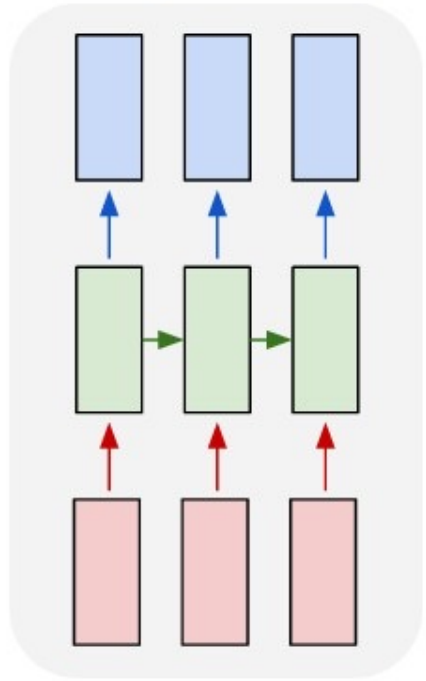
video classification

many to many



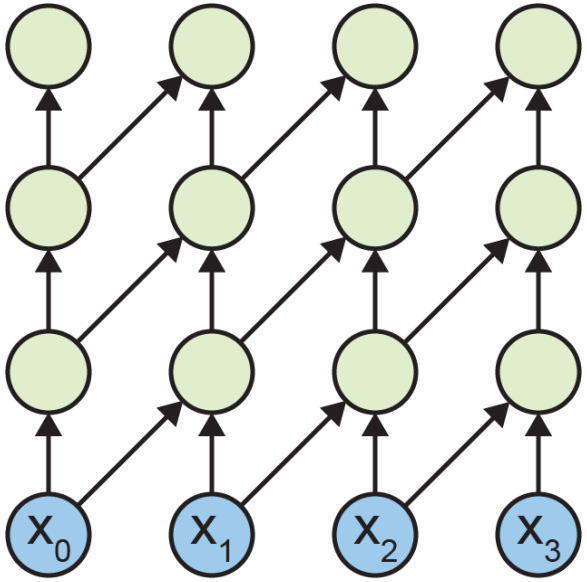
visual question answering, visual dialog
text-to-image, text-to-video generation, …

many to many

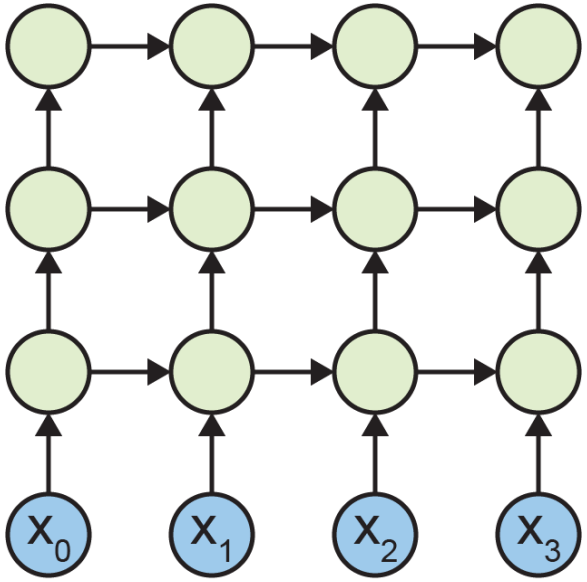


http://karpathy.github.io/2015/05/21/rnn-effectiveness/

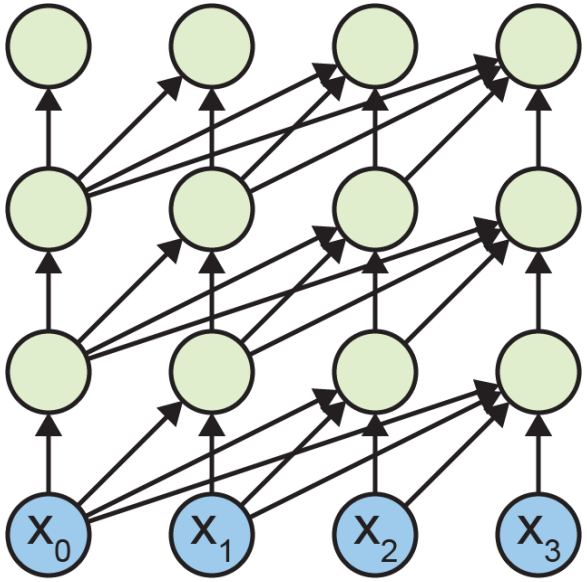
Paradigms of Sequence Modeling



CNN



RNN

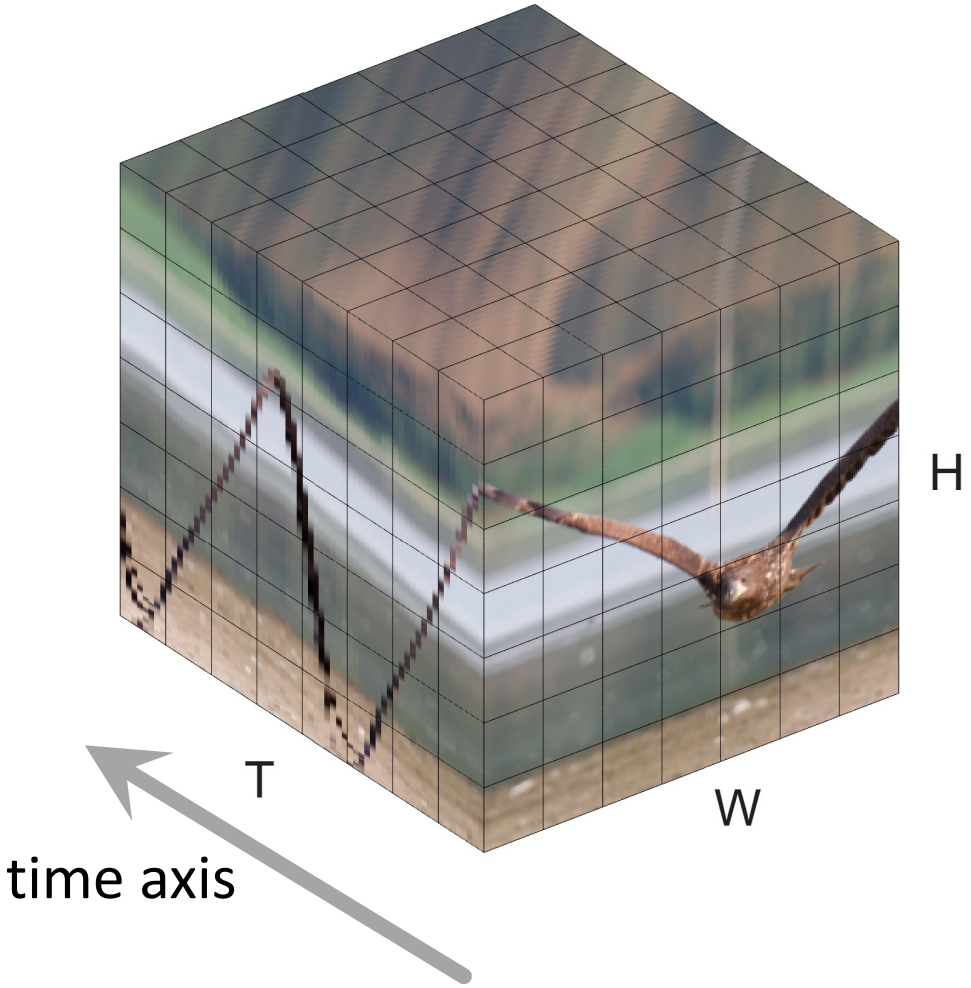


Attention

ConvNets for Sequence Problems

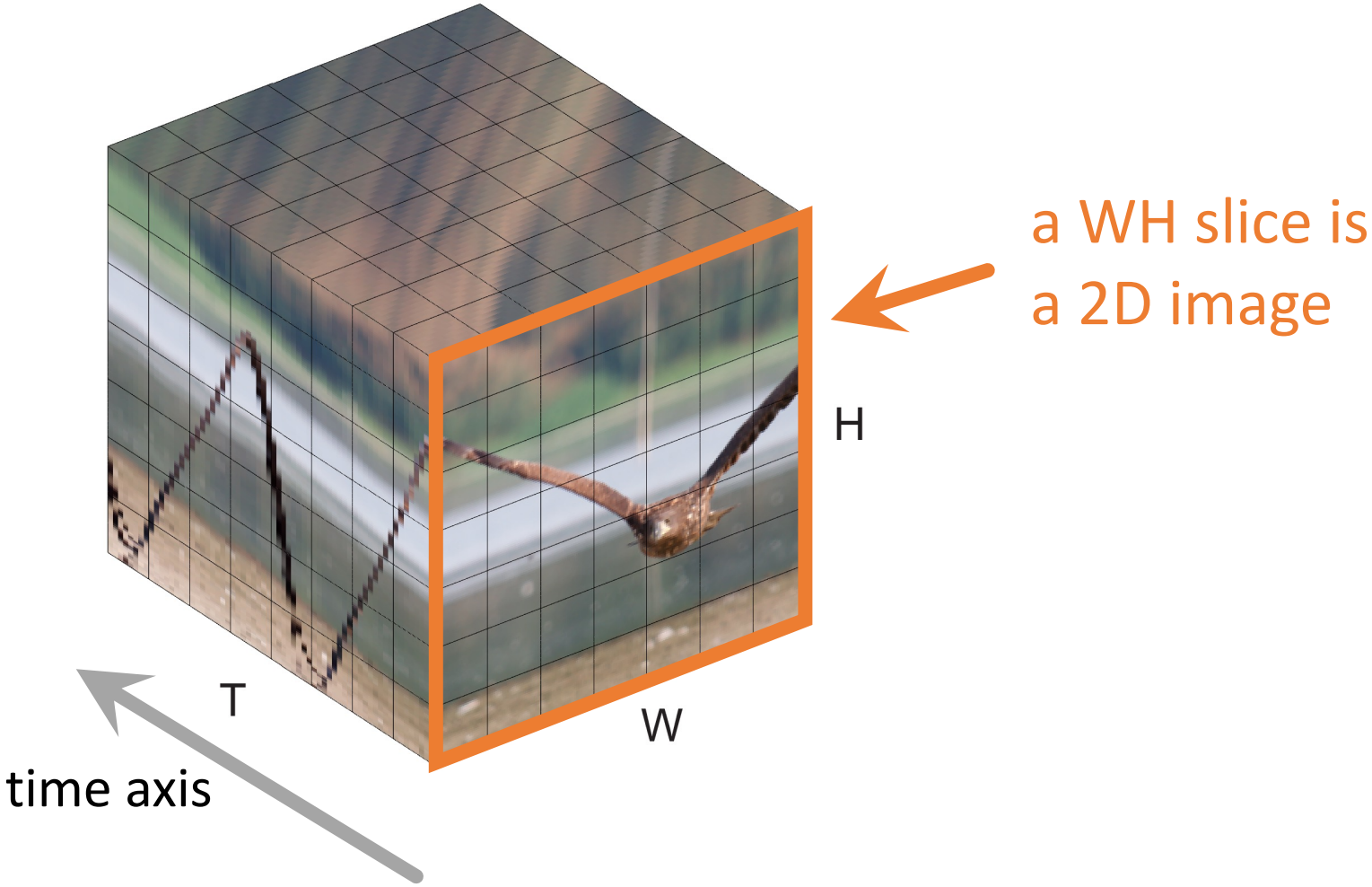
3D ConvNet for Videos

- Videos: 3D counterparts of images



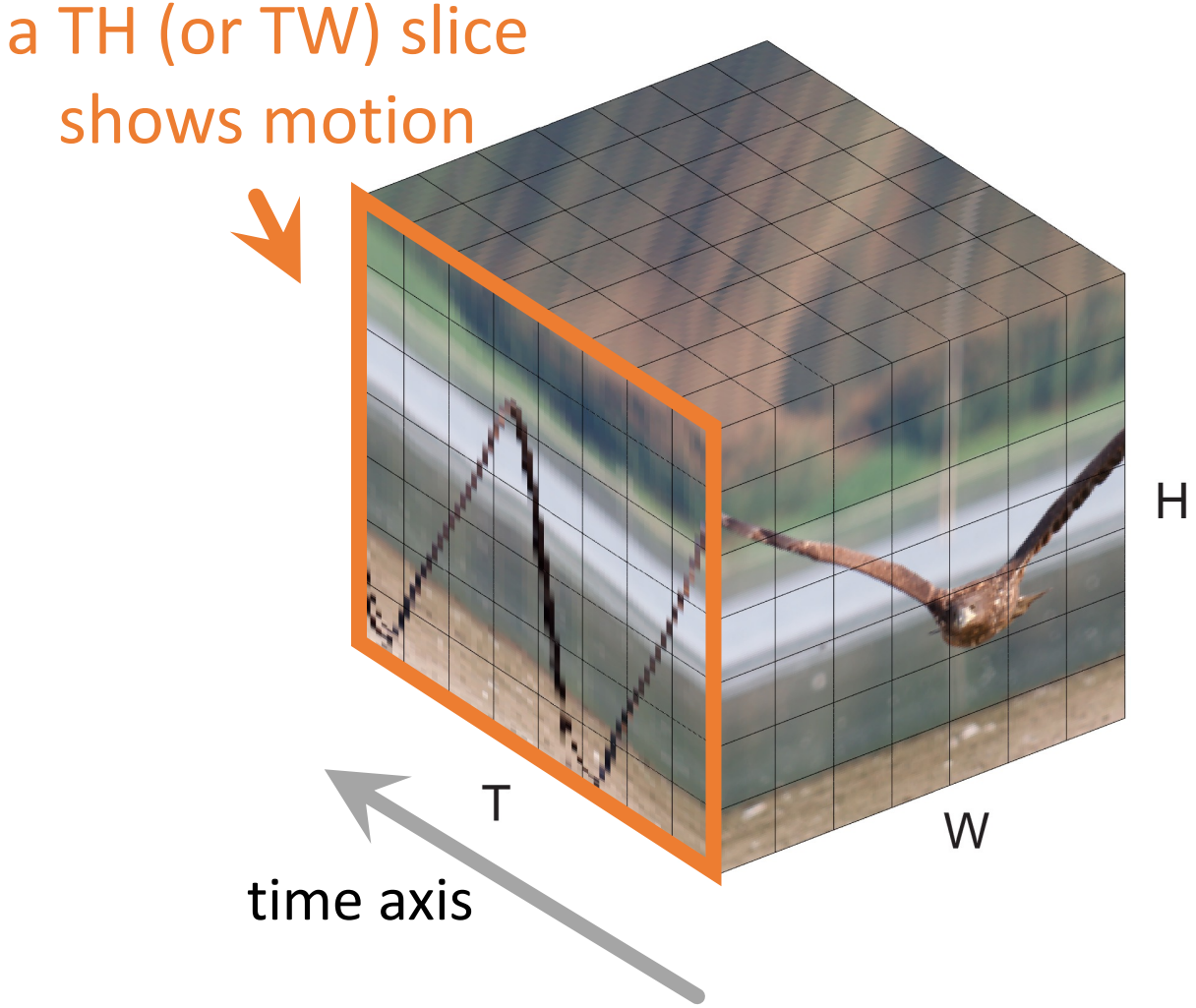
3D ConvNet for Videos

- Videos: 3D counterparts of images



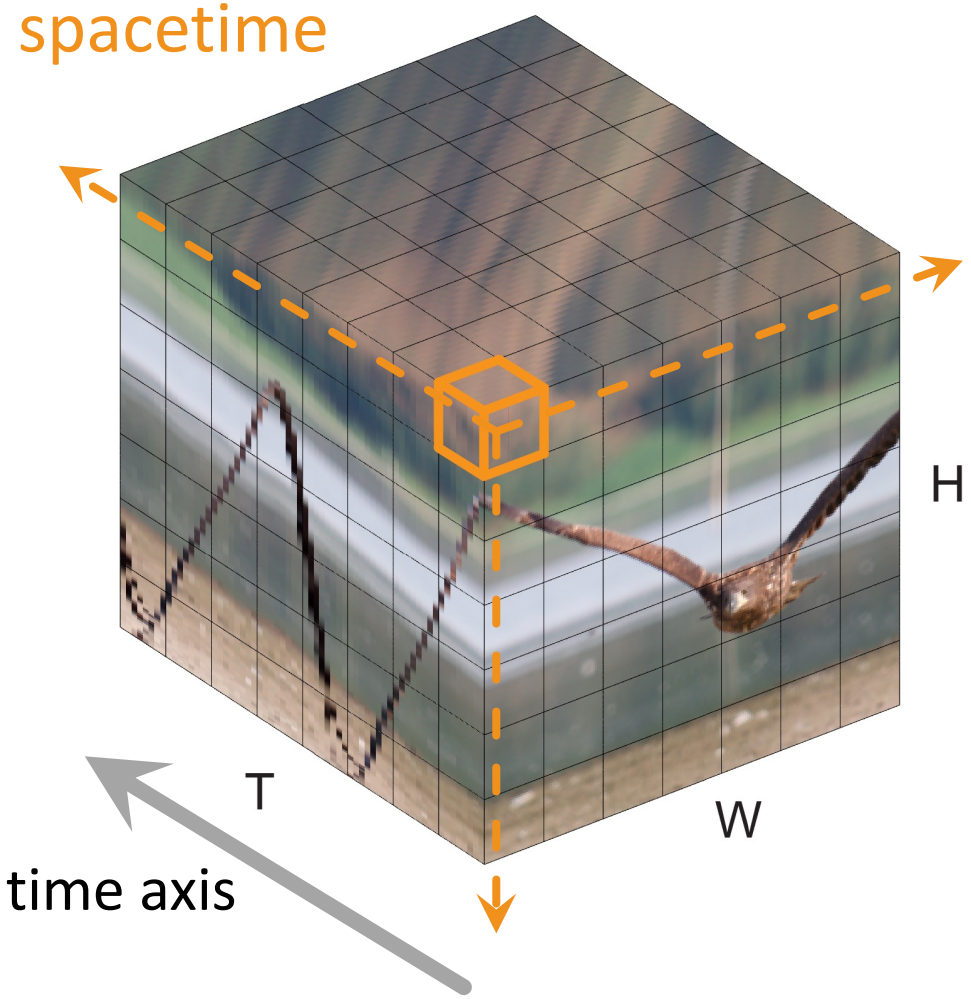
3D ConvNet for Videos

- Videos: 3D counterparts of images



3D ConvNet for Videos

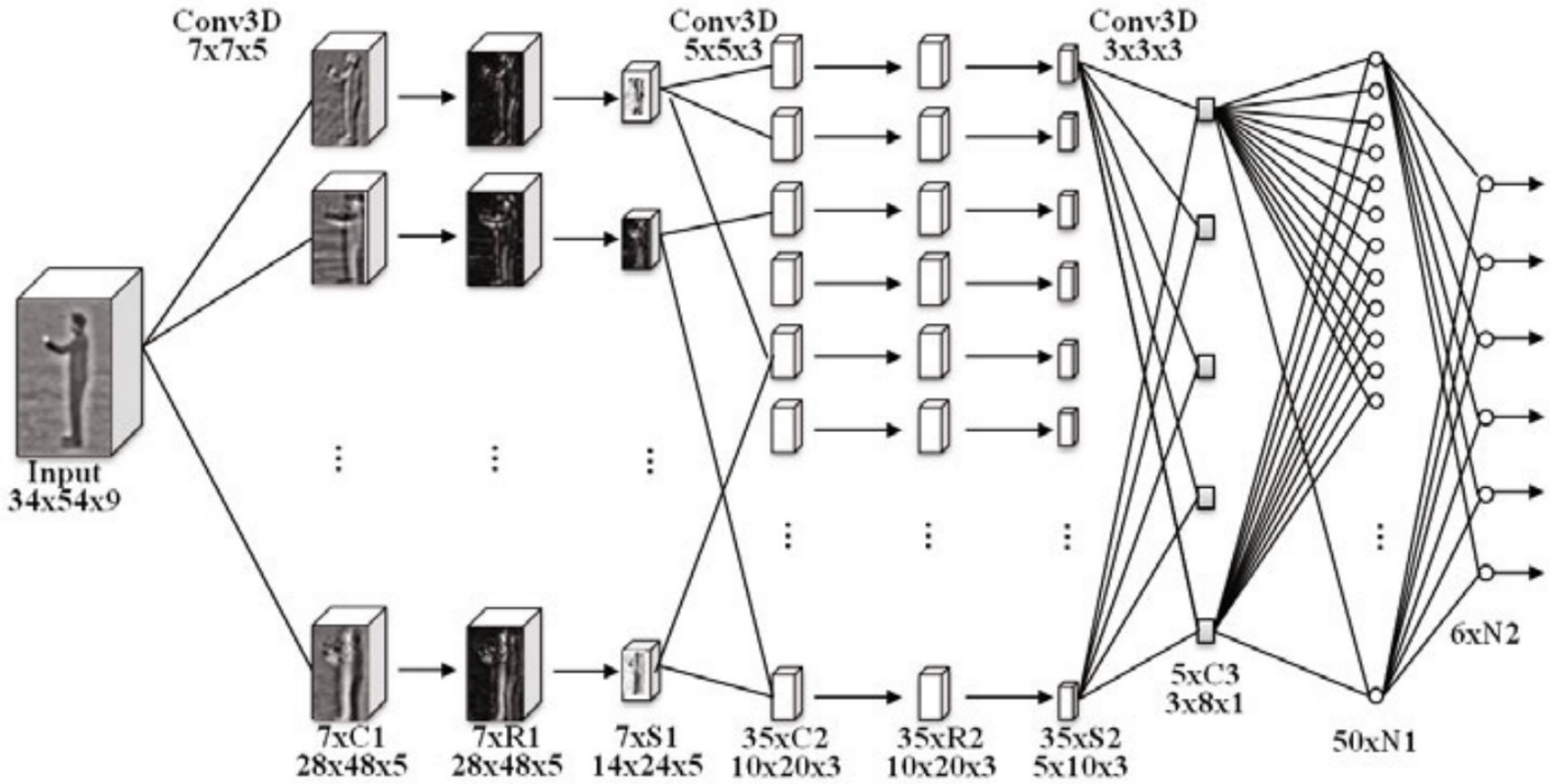
sliding a 3D window
in 3D spacetime



3D convolution

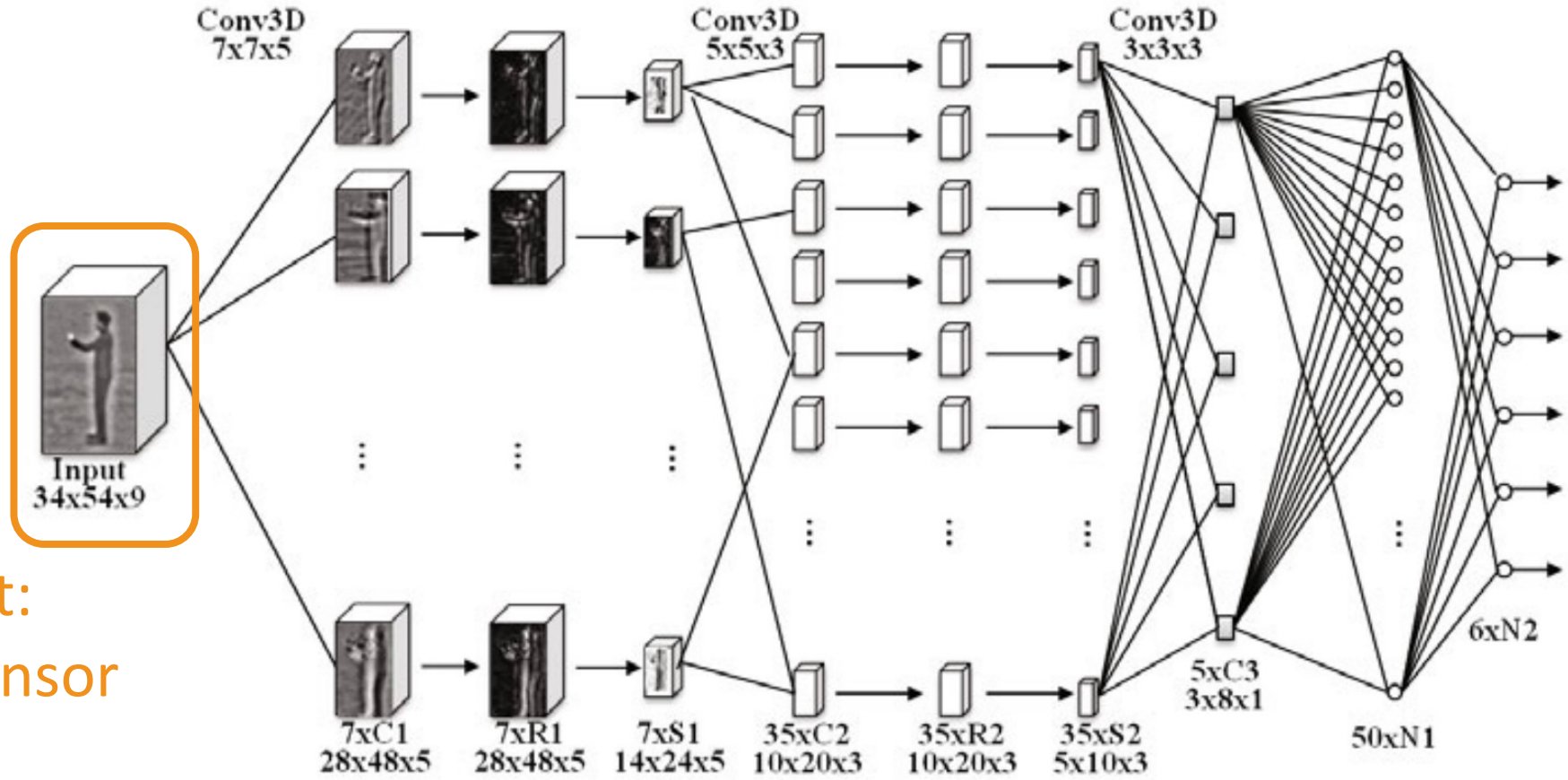
- features
  - **4-D** tensor: C × H × W × **T**
  - C: channels
  - H: height
  - W: width
  - **T**: time
- filters
  - **5-D** tensor: $C_o \times C_i \times K_h \times K_w \times \mathbf{K_T}$
  - C_o: output channels
  - C_i: input channels
  - K_h, K_w, $\mathbf{K_T}$: filter size

3D ConvNet for Videos



Baccouche, et al., “Sequential Deep Learning for Human Action Recognition”, 2011

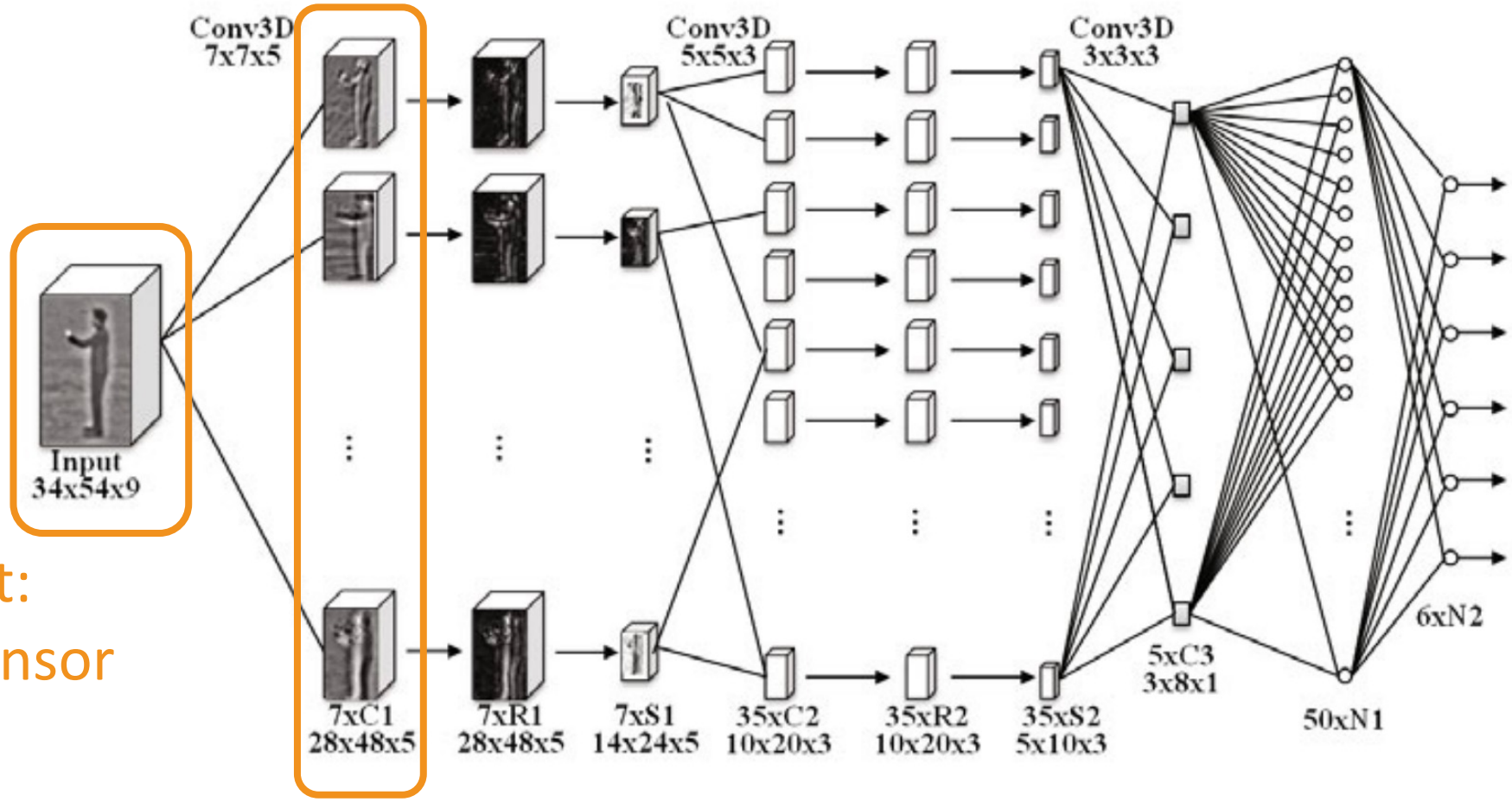
3D ConvNet for Videos



input:
a 3-D tensor

Baccouche, et al., "Sequential Deep Learning for Human Action Recognition", 2011

3D ConvNet for Videos



input:
a 3-D tensor

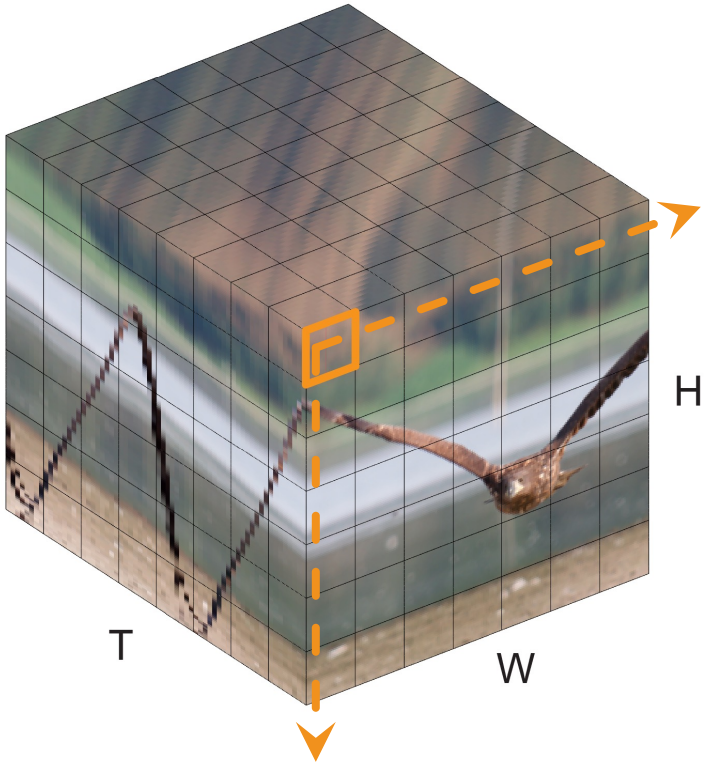
features:
a 4-D tensor

Baccouche, et al., "Sequential Deep Learning for Human Action Recognition", 2011

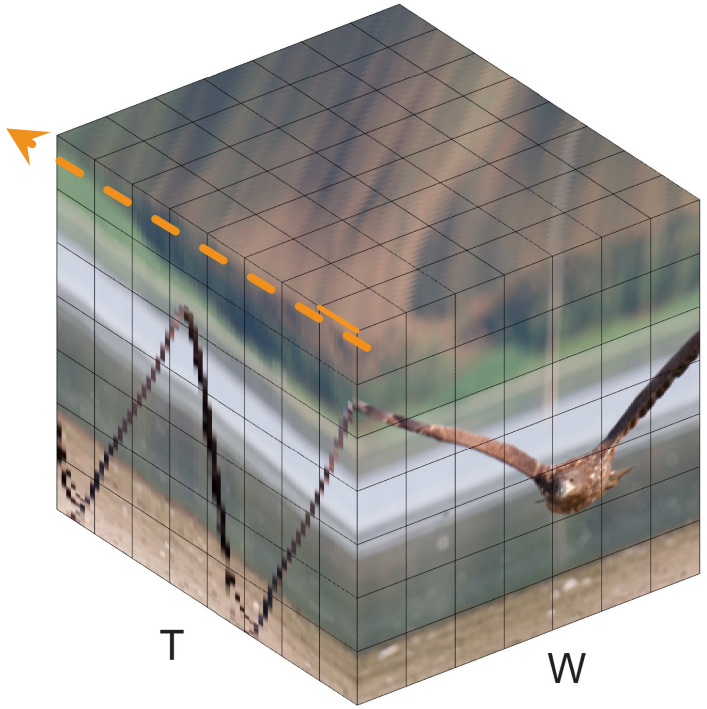
Separating Space and Time

- Separable convolution: 2D+1D

2D conv in space, for each frame



1D conv in time, for each pixel



parameters

- **3D conv:**

  $C_o \times C_i \times K_h \times K_w \times K_T$

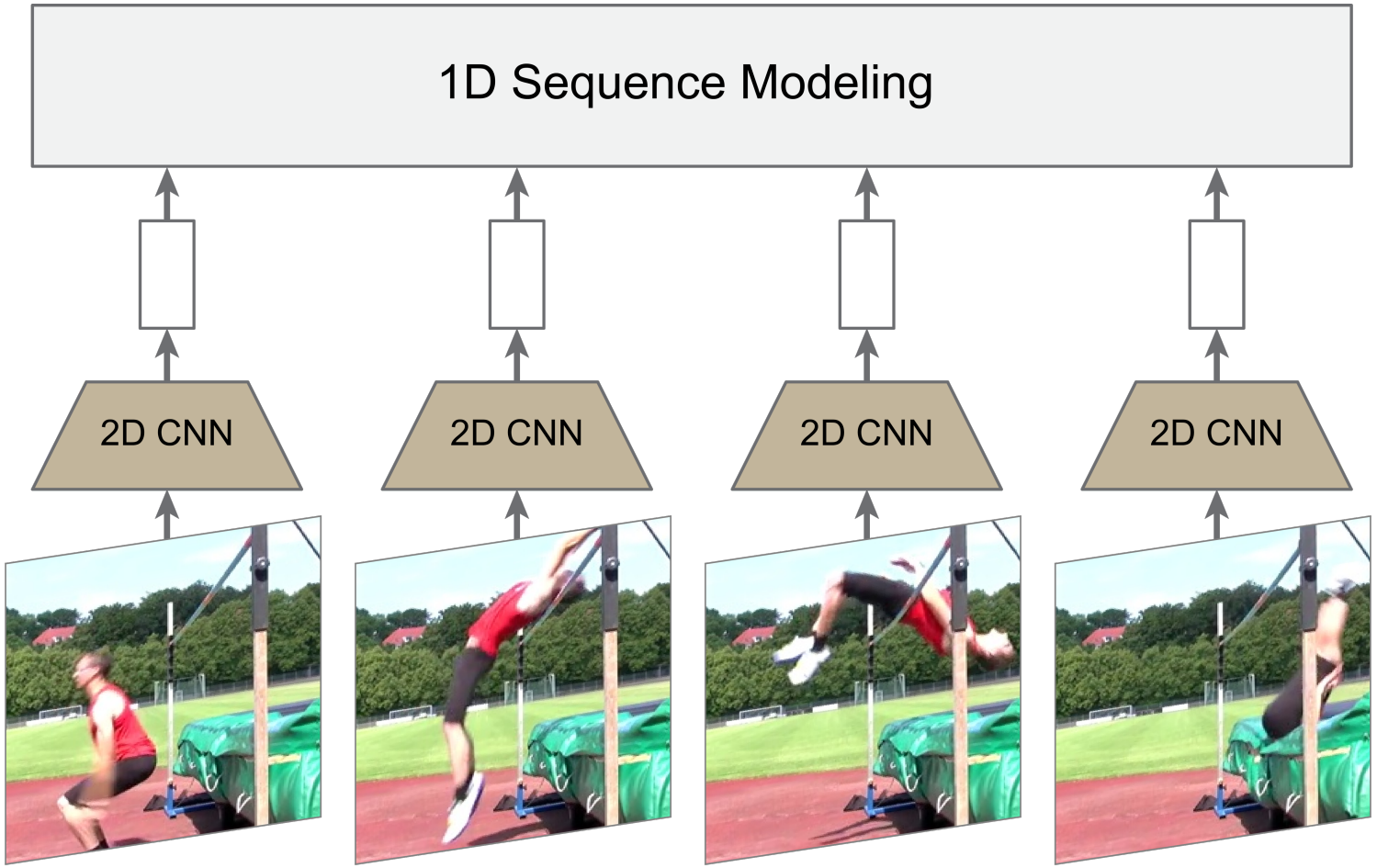
- **seperable conv:**

  $C_o' \times C_i \times K_h \times K_w \times 1 +$
  $C_o \times C_o' \times 1 \times 1 \times K_T$

operations = # params × H × W × T

Separating Space and Time

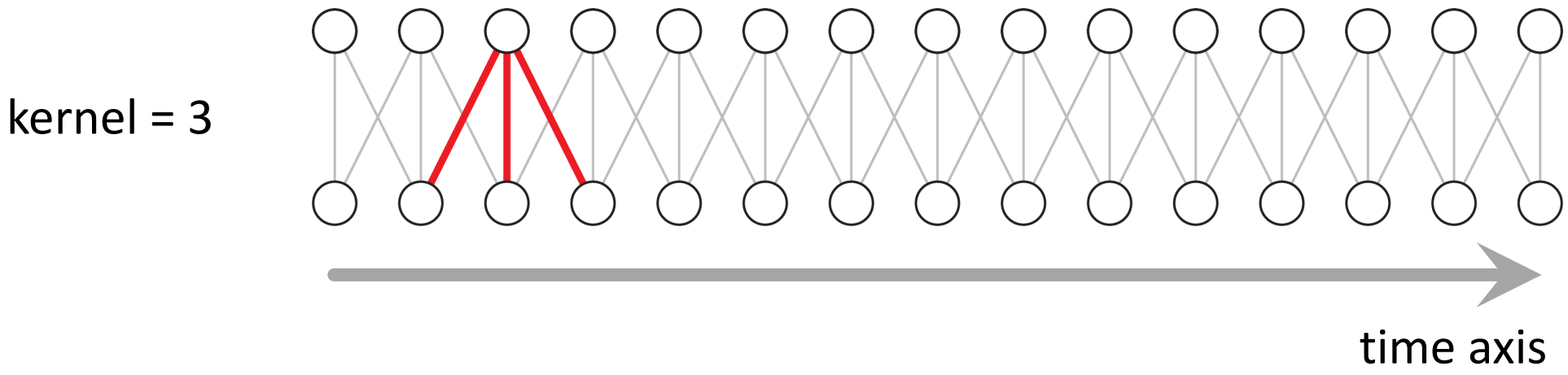
- 2D ConvNet per frame + 1D Sequence Modeling



1D ConvNet for Sequence Modeling

Convolution (in time)

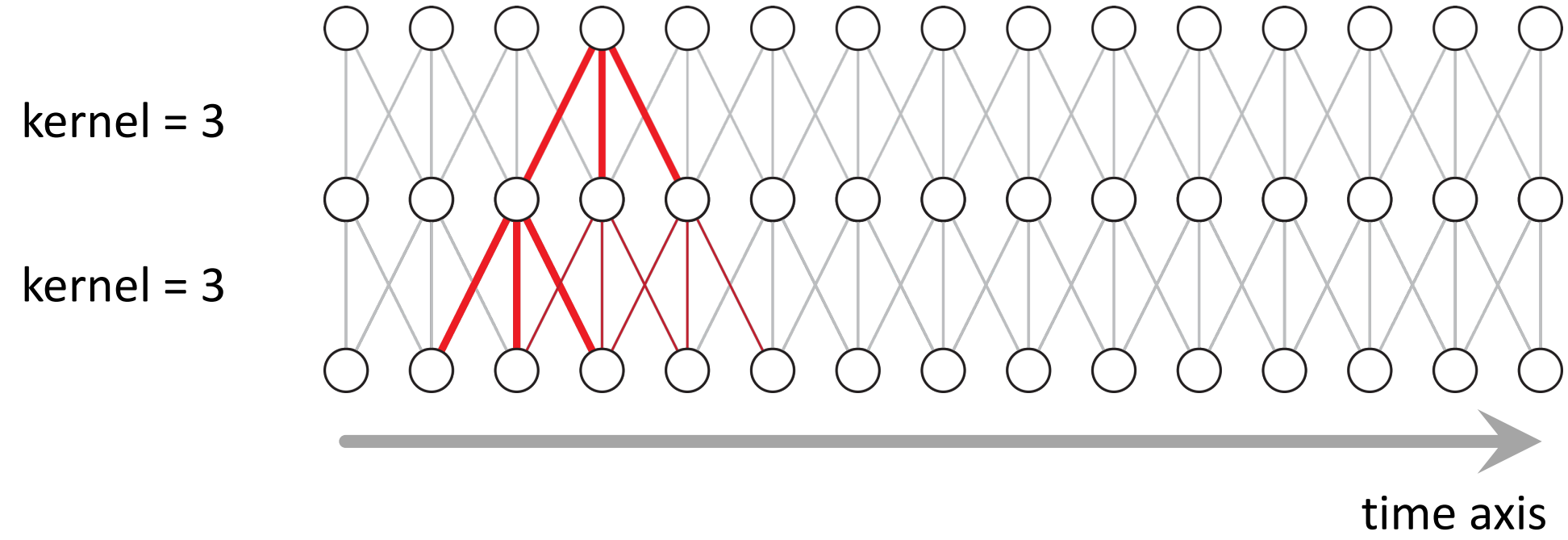
- sliding window (in time)
- local connection (in time)
- weight-sharing (in time)



1D ConvNet for Sequence Modeling

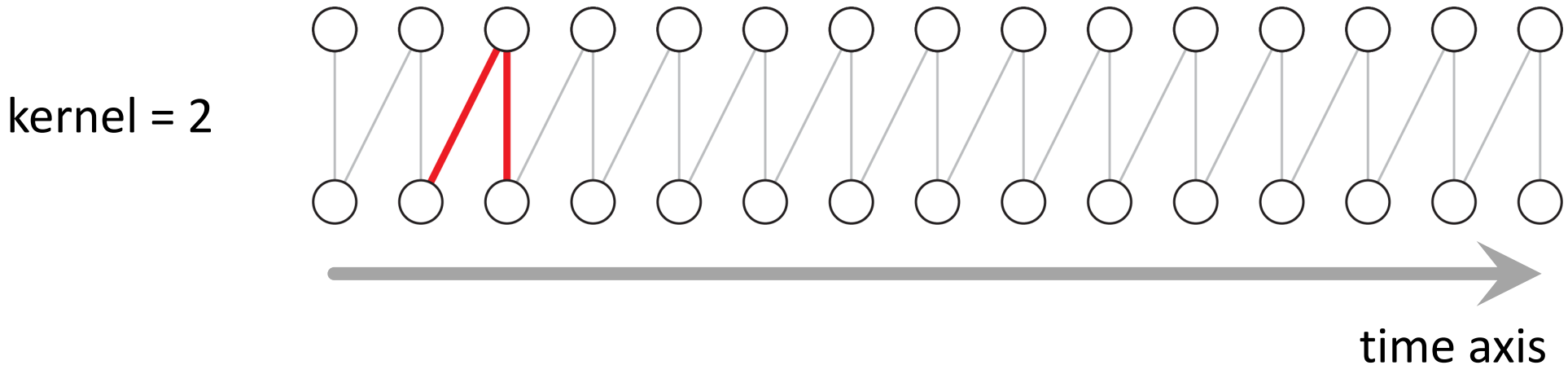
Convolution (in time)

- sliding window (in time)
- local connection (in time)
- weight-sharing (in time)



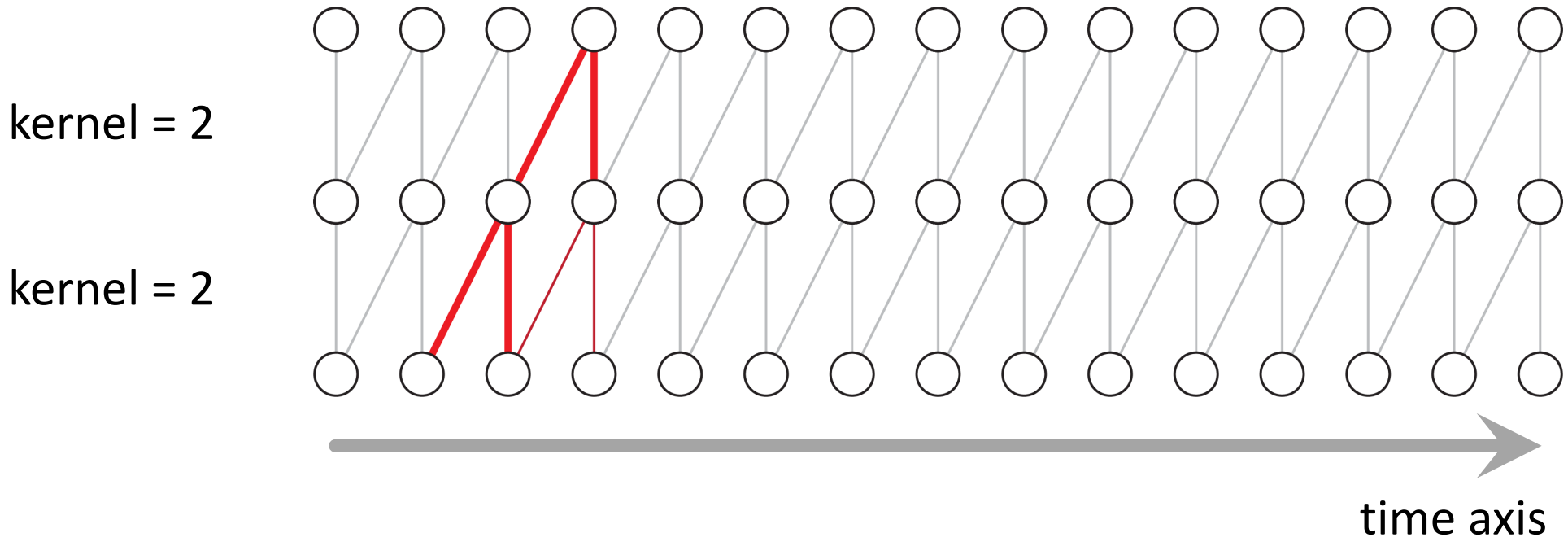
1D ConvNet: Causal convolution

- **Causal**: output at a time step does not depend on future time steps



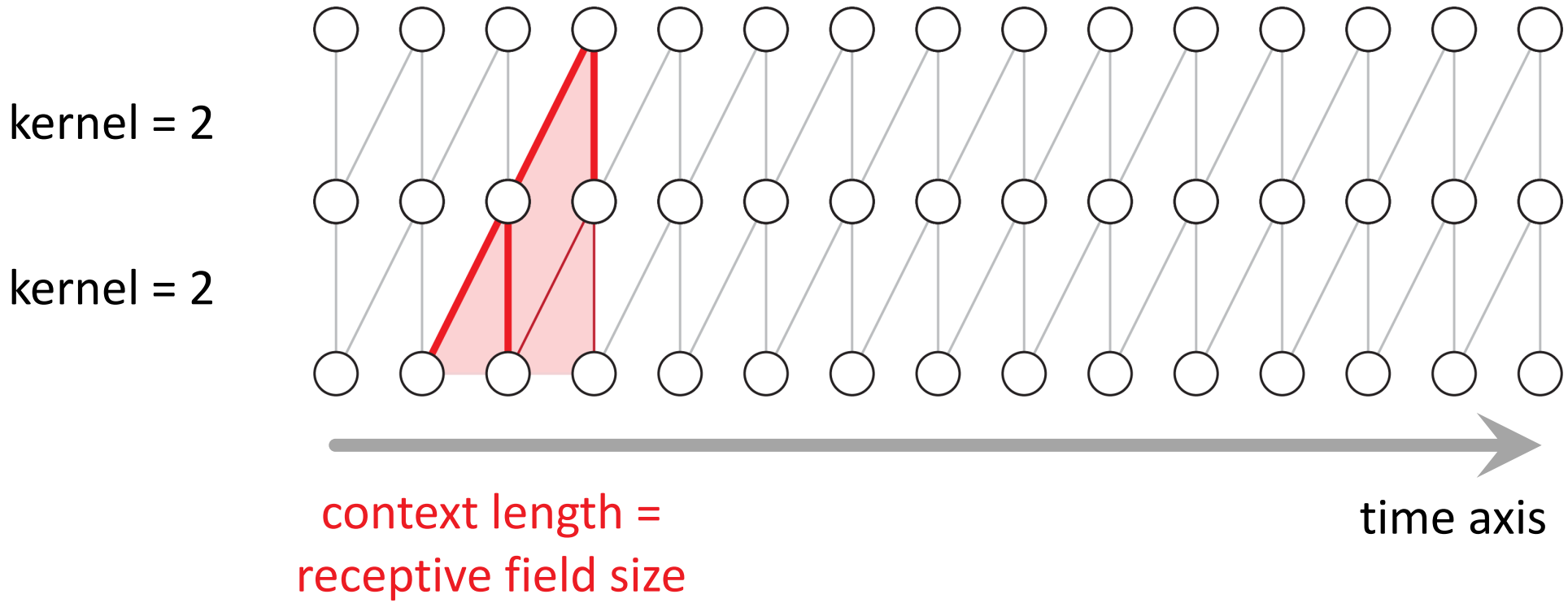
1D ConvNet: Causal convolution

- **Causal**: output at a time step does not depend on future time steps



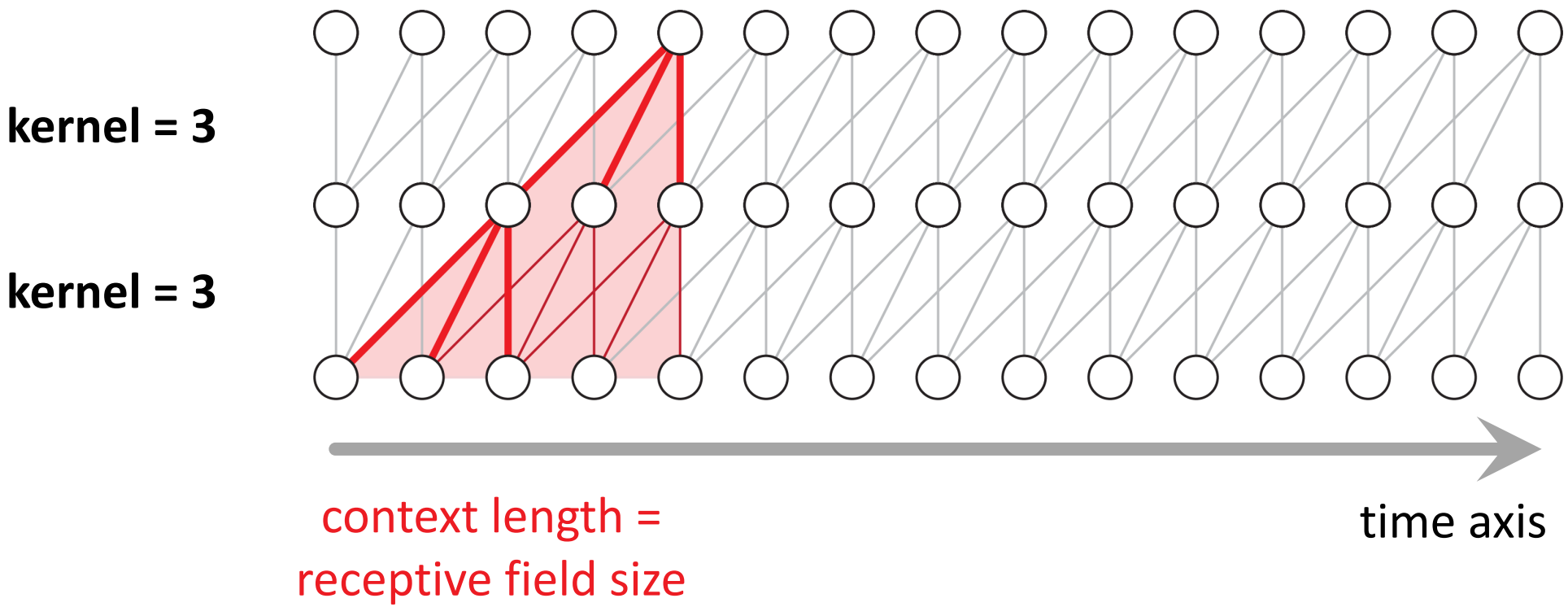
1D ConvNet: Causal convolution

- **Causal**: output at a time step does not depend on future time steps



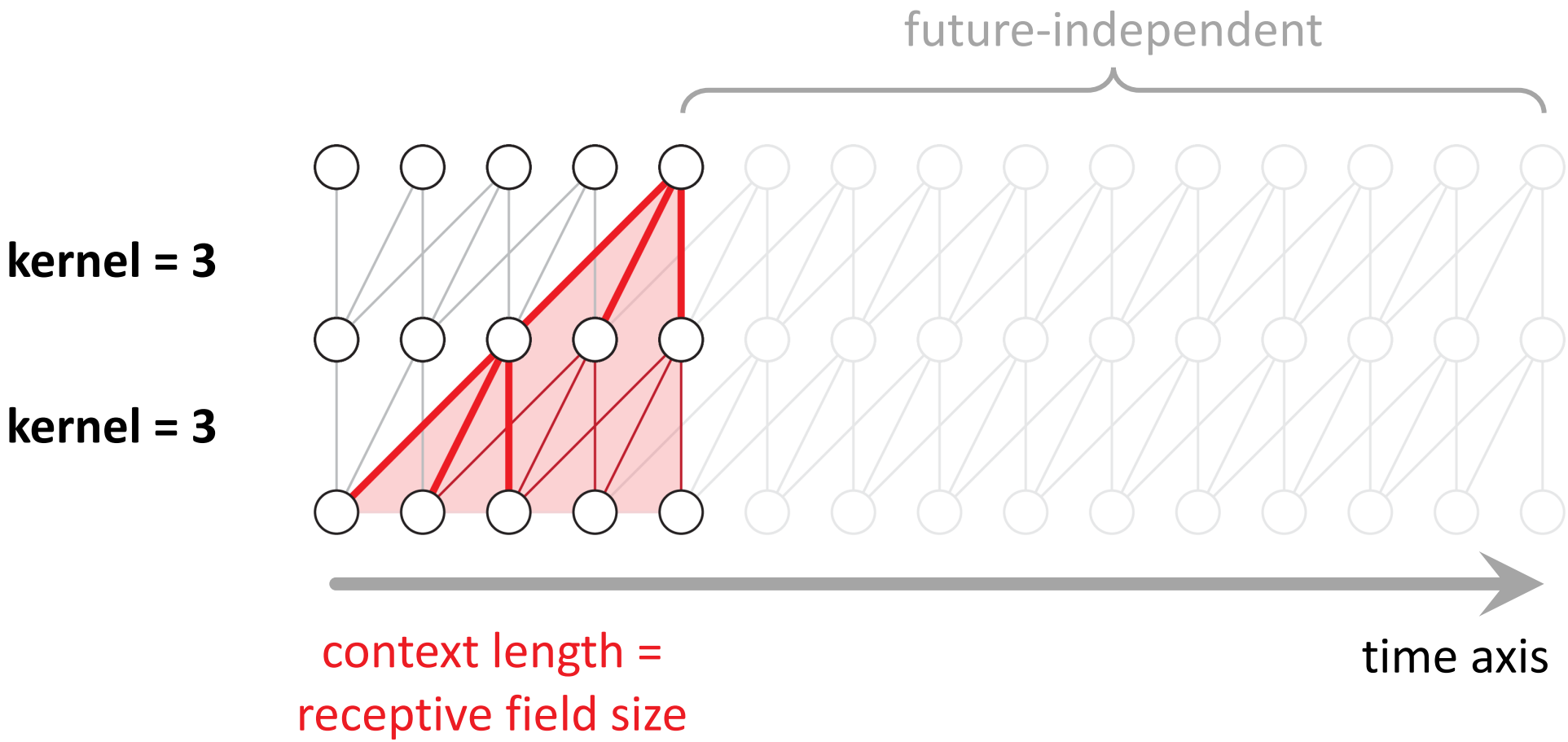
1D ConvNet: Causal convolution

- **Causal**: output at a time step does not depend on future time steps



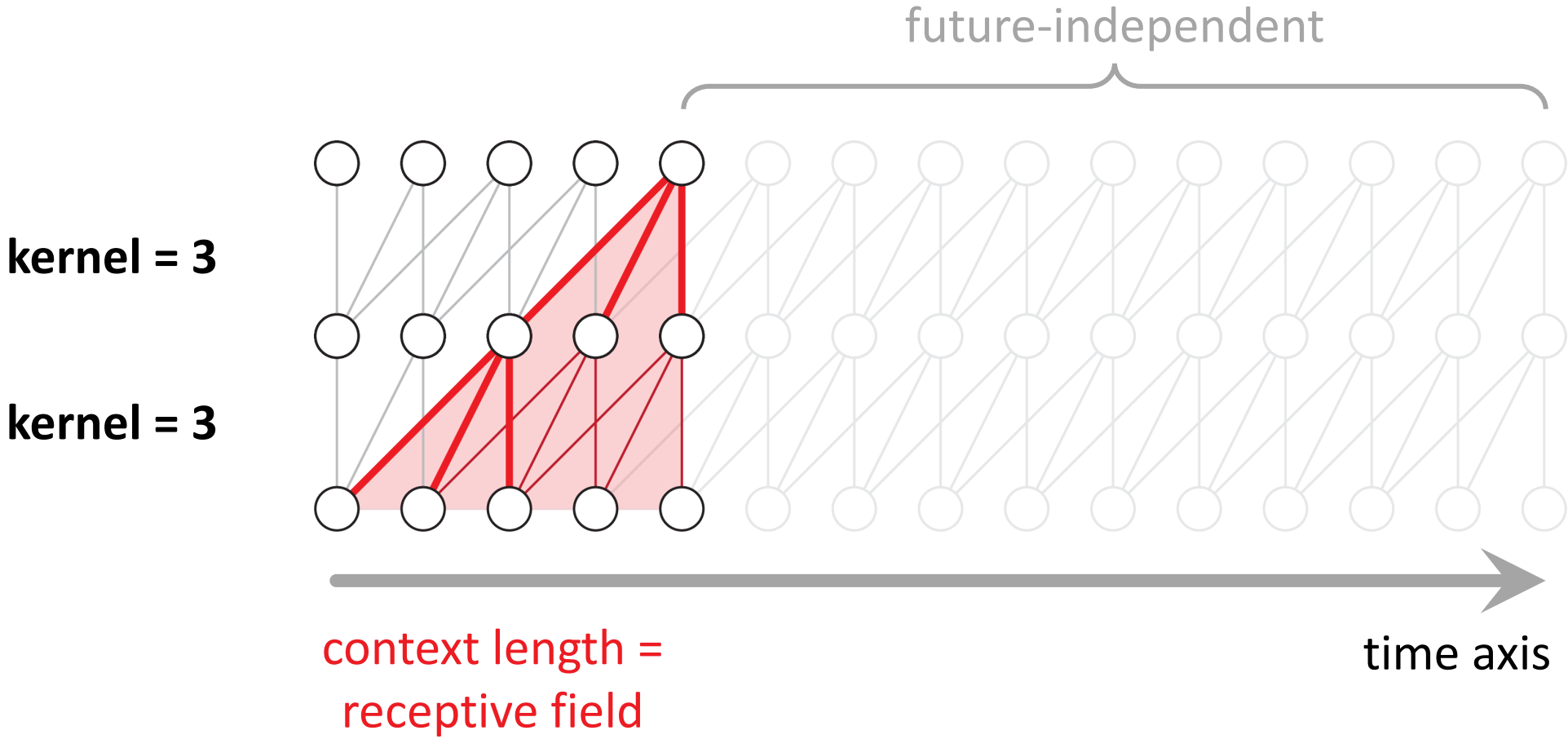
1D ConvNet: Causal convolution

- **Causal**: output at a time step does not depend on future time steps



1D ConvNet: Causal convolution

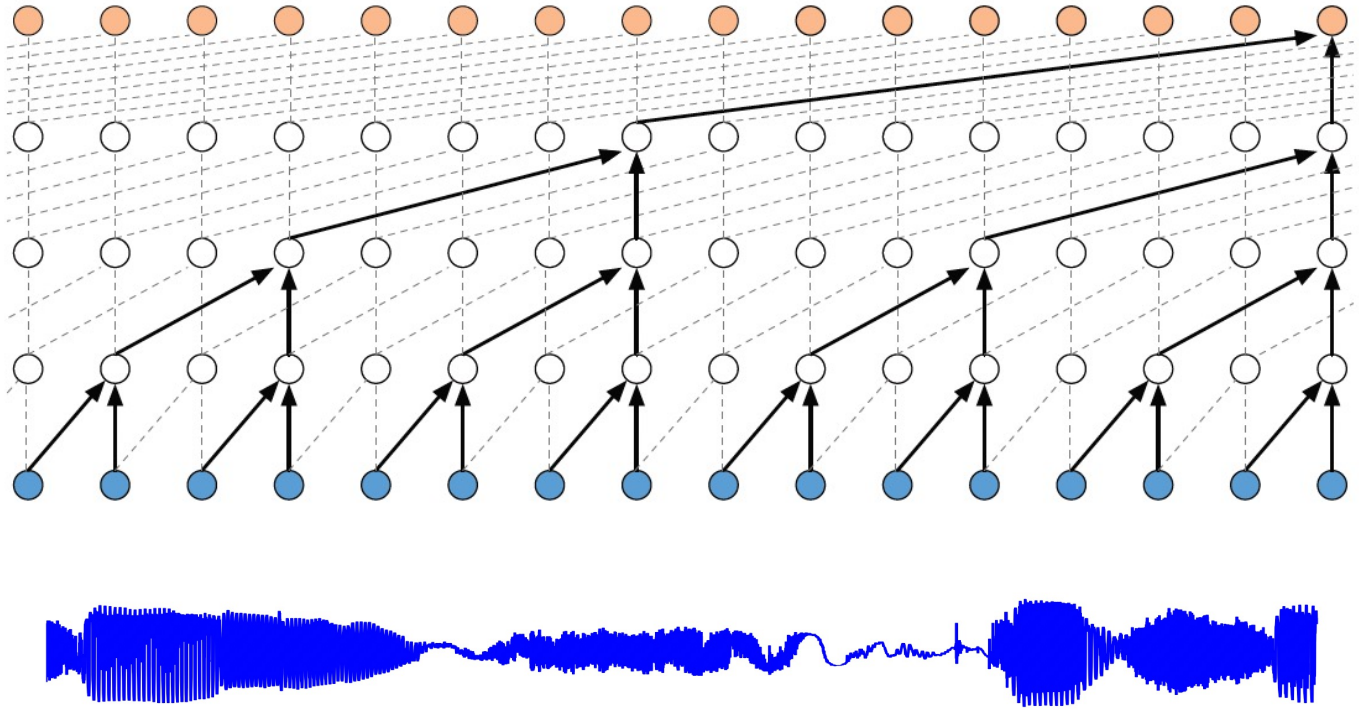
- **Causal**: output at a time step does not depend on future time steps
- conv w/ **pad = kernel – 1**, only on left (no pad on right)

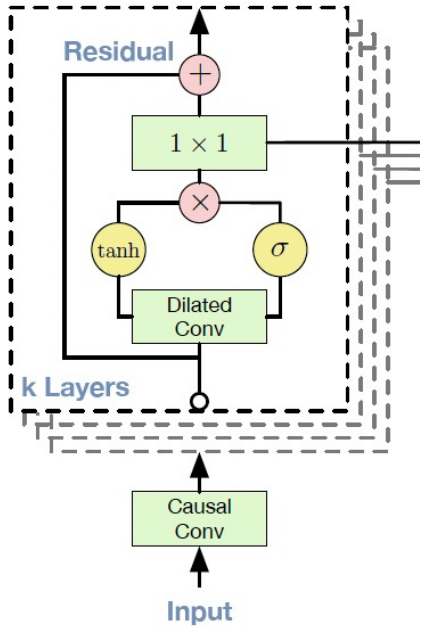


Example: DeepMind's WaveNet for Audio Generation

Causal ConvNet with long context

- going deep with residual connections
- "dilated" convolution





"WaveNet: A Generative Model for Raw Audio", van den Oord, et al., 2016

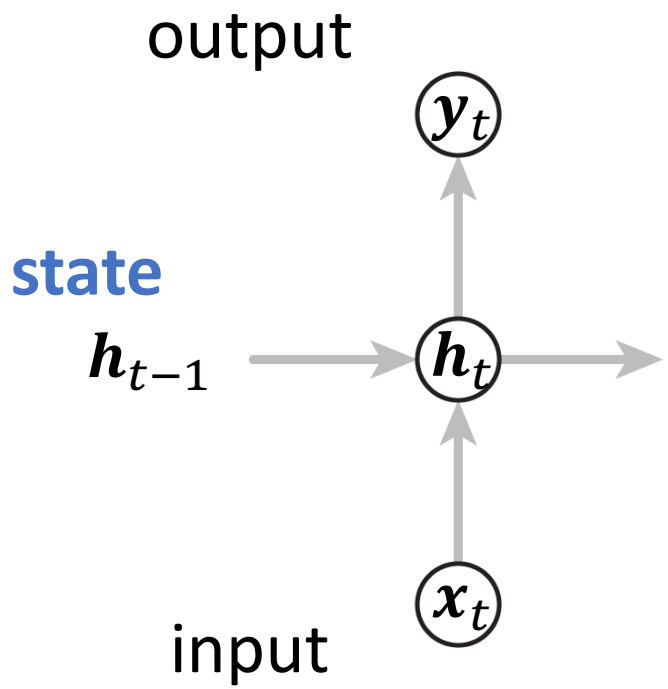
Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs)

- Hidden state: summary of the past
- Weight-sharing in time
- Ordered sequences
- BackProp Through Time (BPTT)

Recurrent NN: Hidden State

- Hidden state: summary/memory of the sequence seen so far

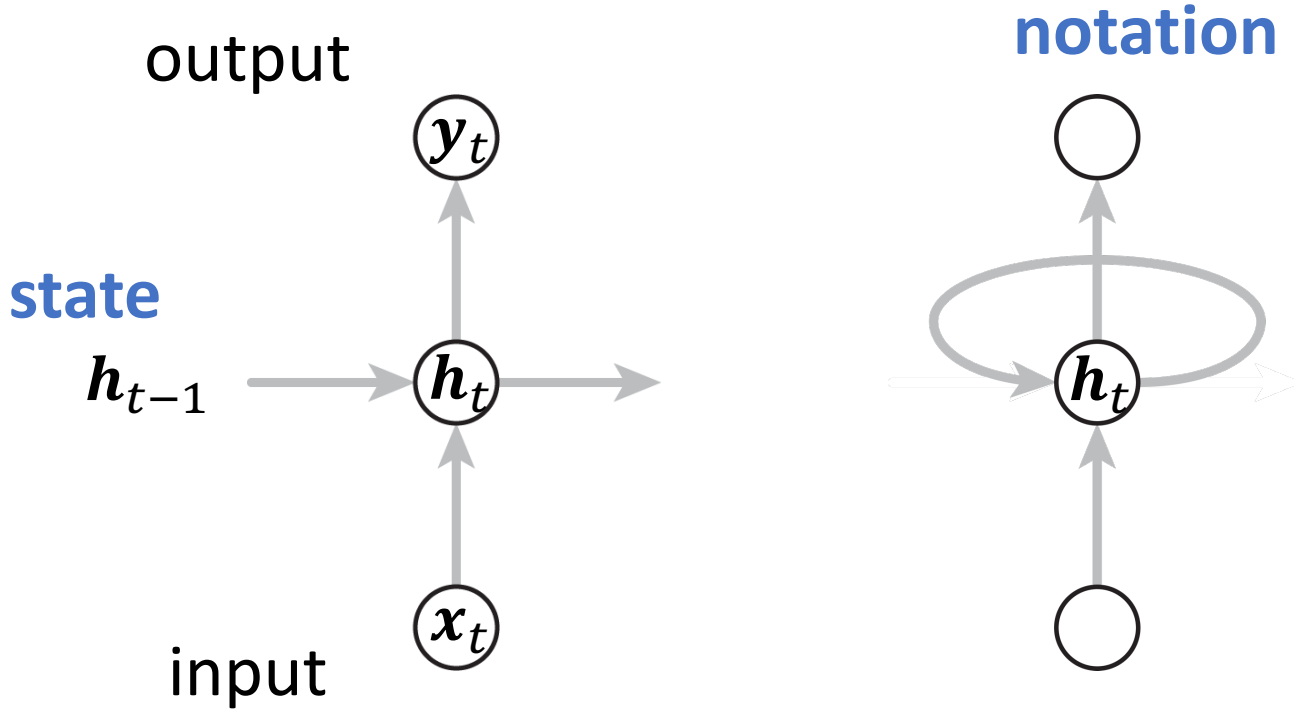


$$h_t = f(h_{t-1}, x_t)$$
$$y_t = g(h_t)$$

- recurrent
- feedforward

Recurrent NN: Hidden State

- Hidden state: summary/memory of the sequence seen so far

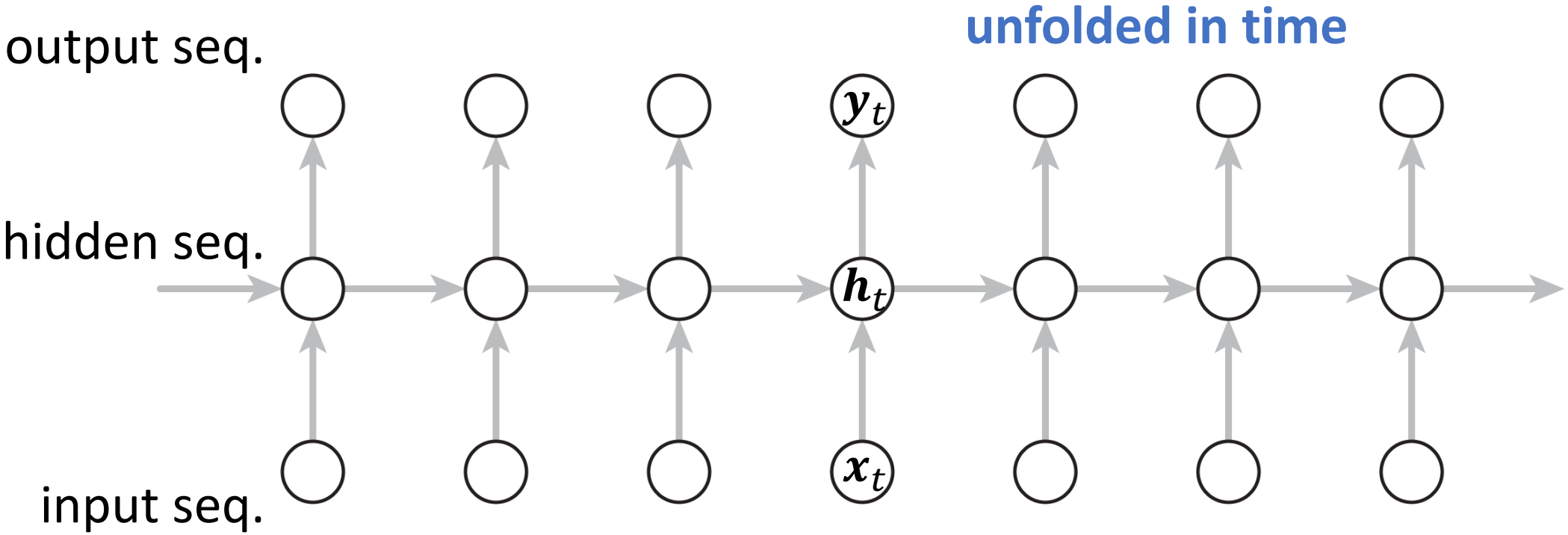


$$h_t = f(h_{t-1}, x_t)$$
$$y_t = g(h_t)$$

- recurrent
- feedforward

Recurrent NN: Hidden State

- Hidden state: summary/memory of the sequence seen so far

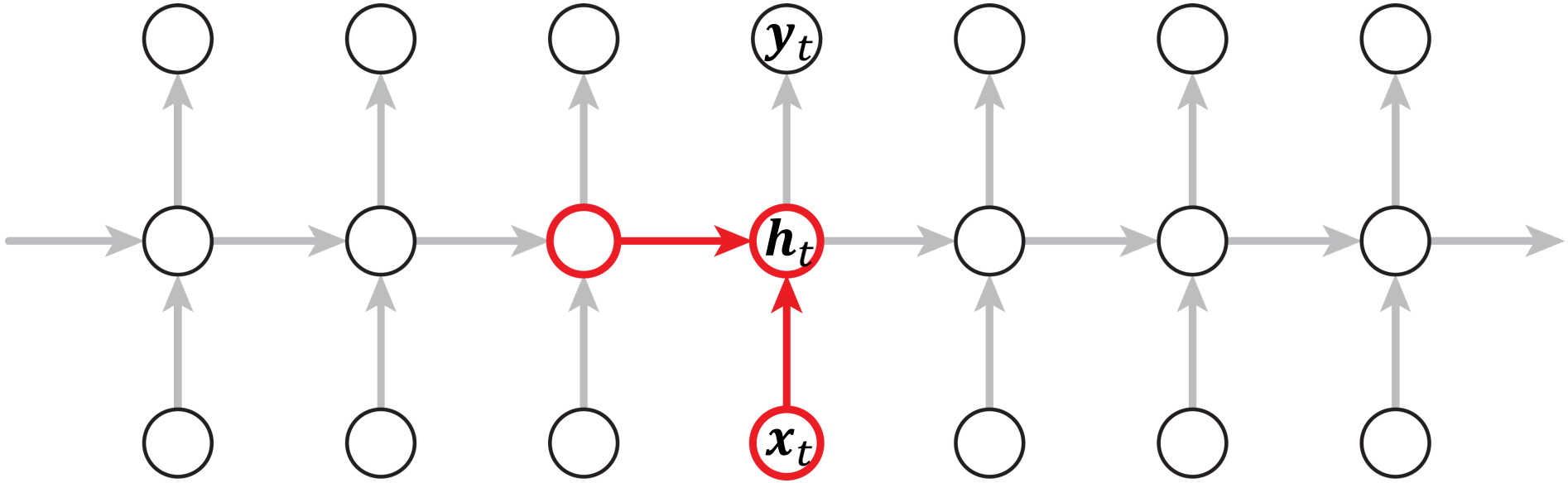


$$\boldsymbol{h}_t = f(\boldsymbol{h}_{t-1}, \boldsymbol{x}_t)$$
$$\boldsymbol{y}_t = g(\boldsymbol{h}_t)$$

- recurrent
- feedforward

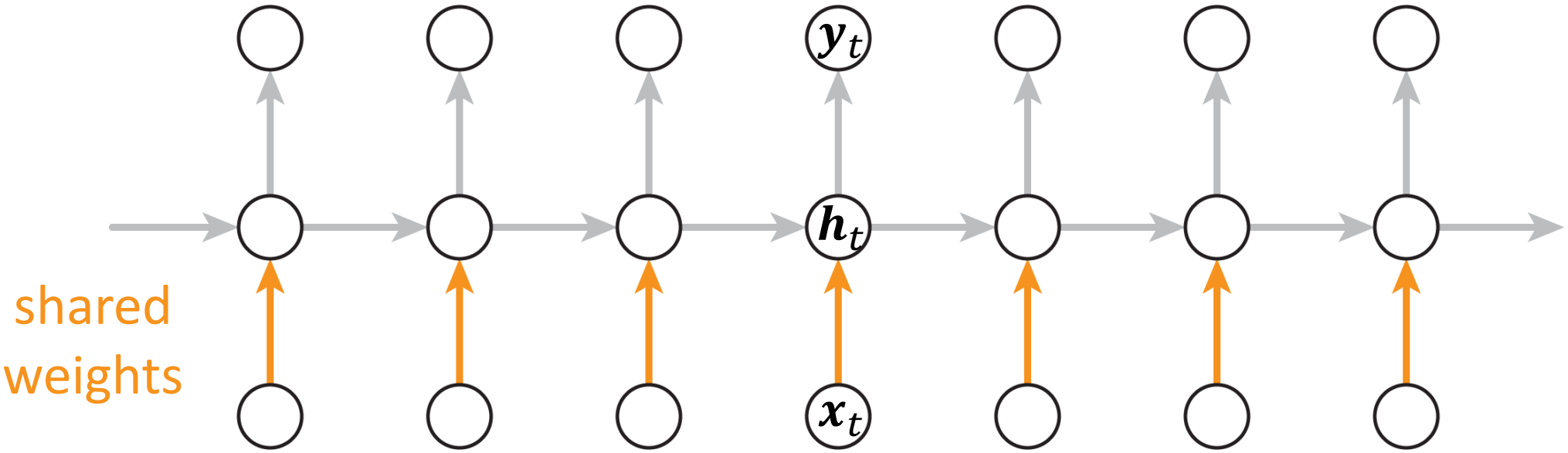
RNN: local connection + weight-sharing

- local connection in time



RNN: local connection + weight-sharing

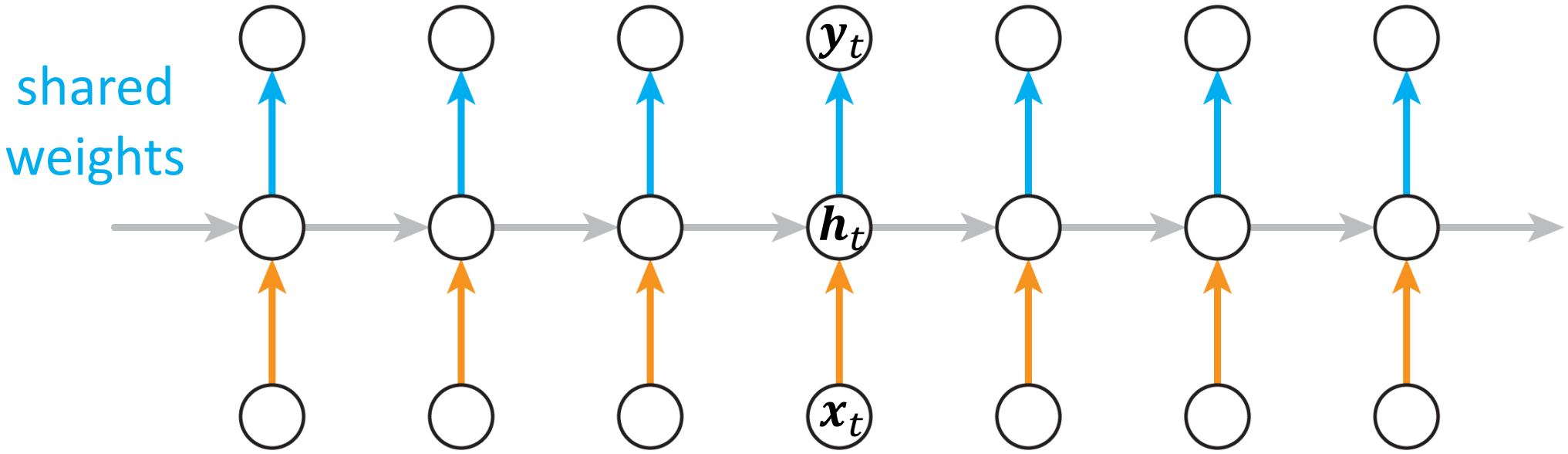
- weight-sharing in time



similar to conv
with kernel size 1

RNN: local connection + weight-sharing

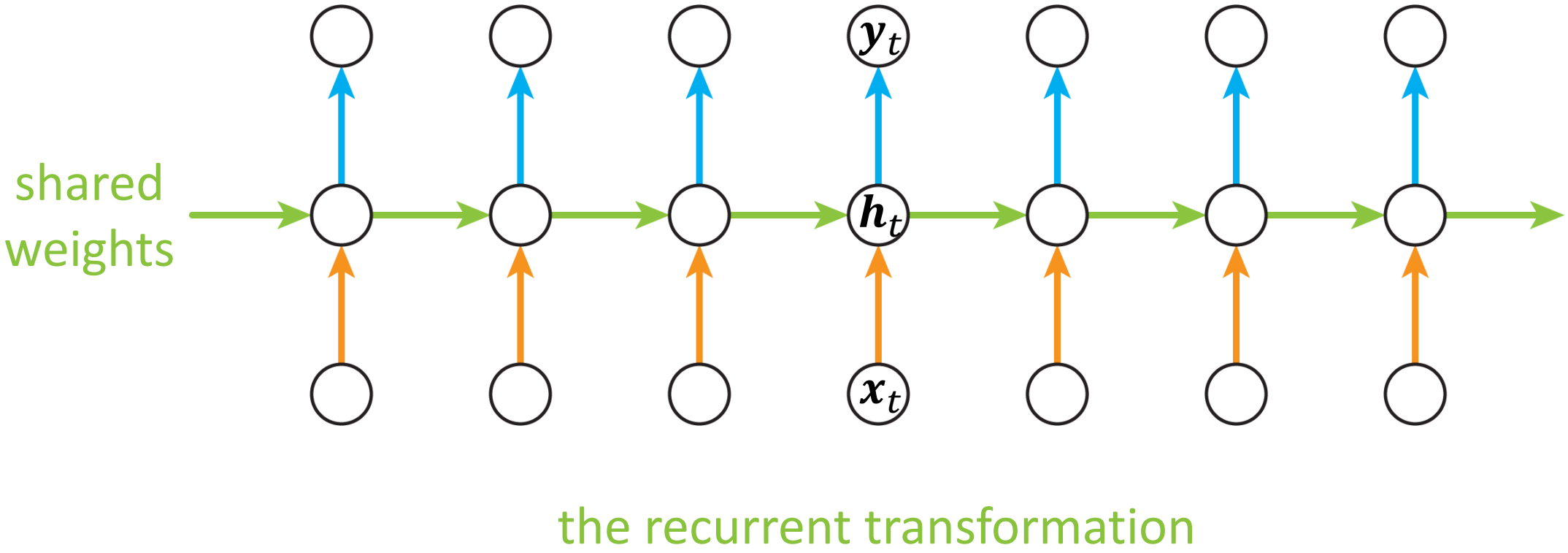
- weight-sharing in time



similar to conv
with kernel size 1

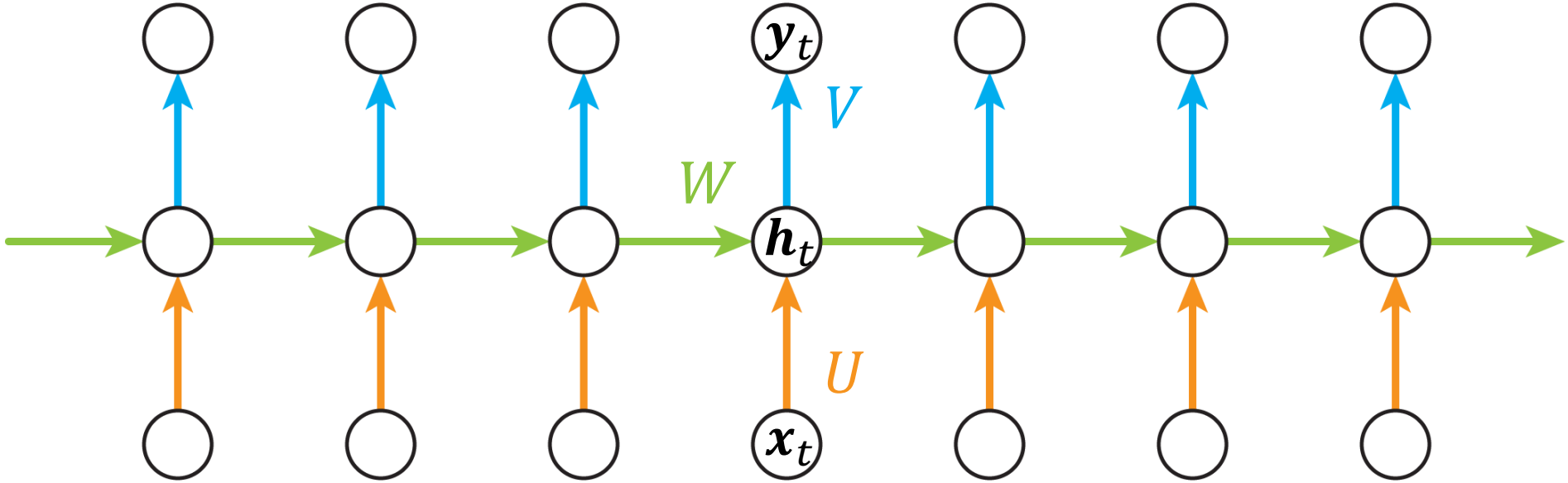
RNN: local connection + weight-sharing

- weight-sharing in time



Example: Vanilla RNN

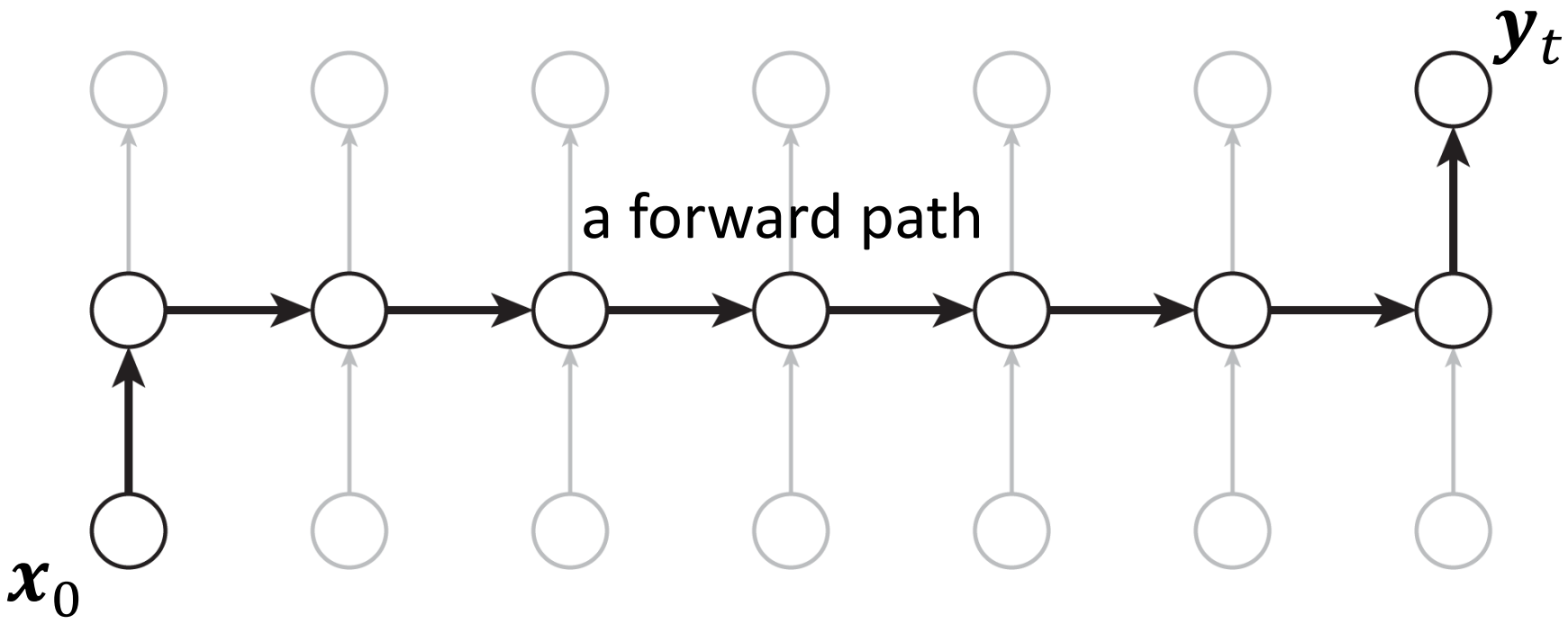
$$\boldsymbol{h}_t = \tanh(W\boldsymbol{h}_{t-1} + U\boldsymbol{x}_t)$$
$$\boldsymbol{y}_t = V\boldsymbol{h}_t$$



*bias omitted for simplicity

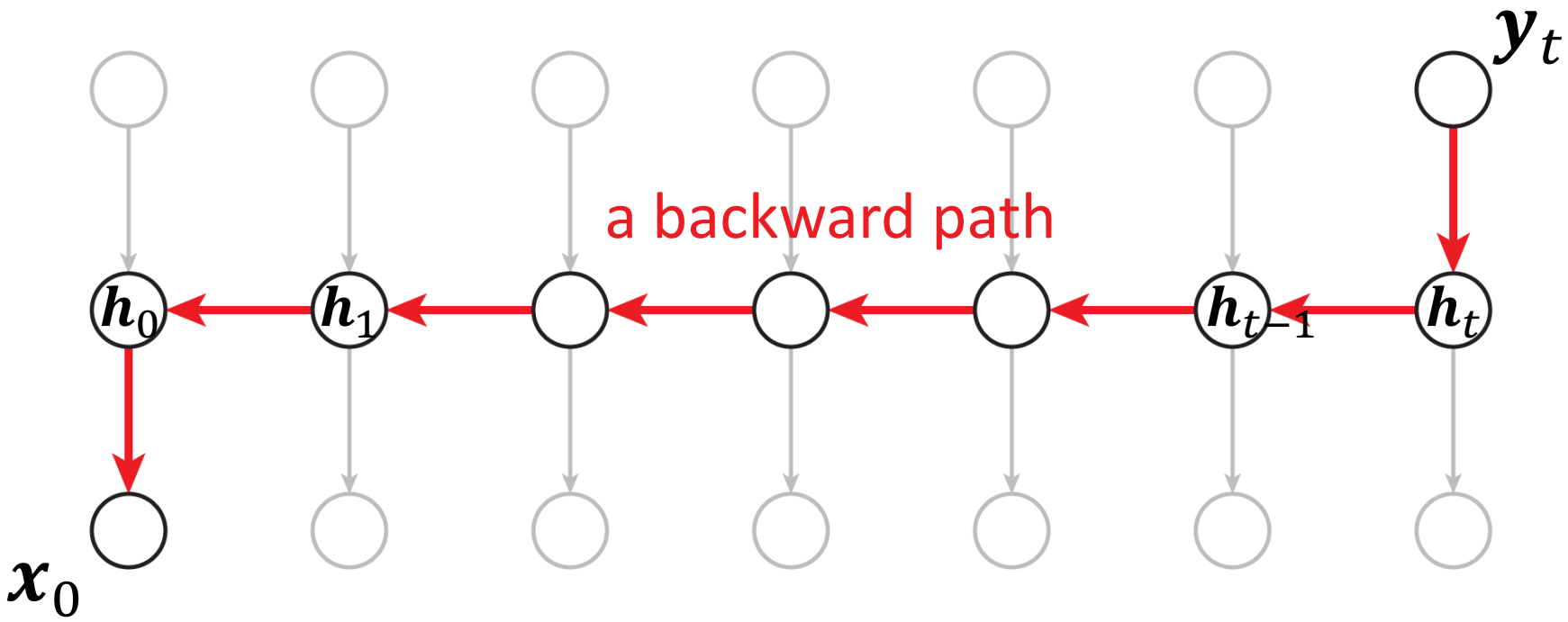
RNN: BackProp Through Time (BPTT)

- output at time t is like a t-layer deep net



RNN: BackProp Through Time (BPTT)

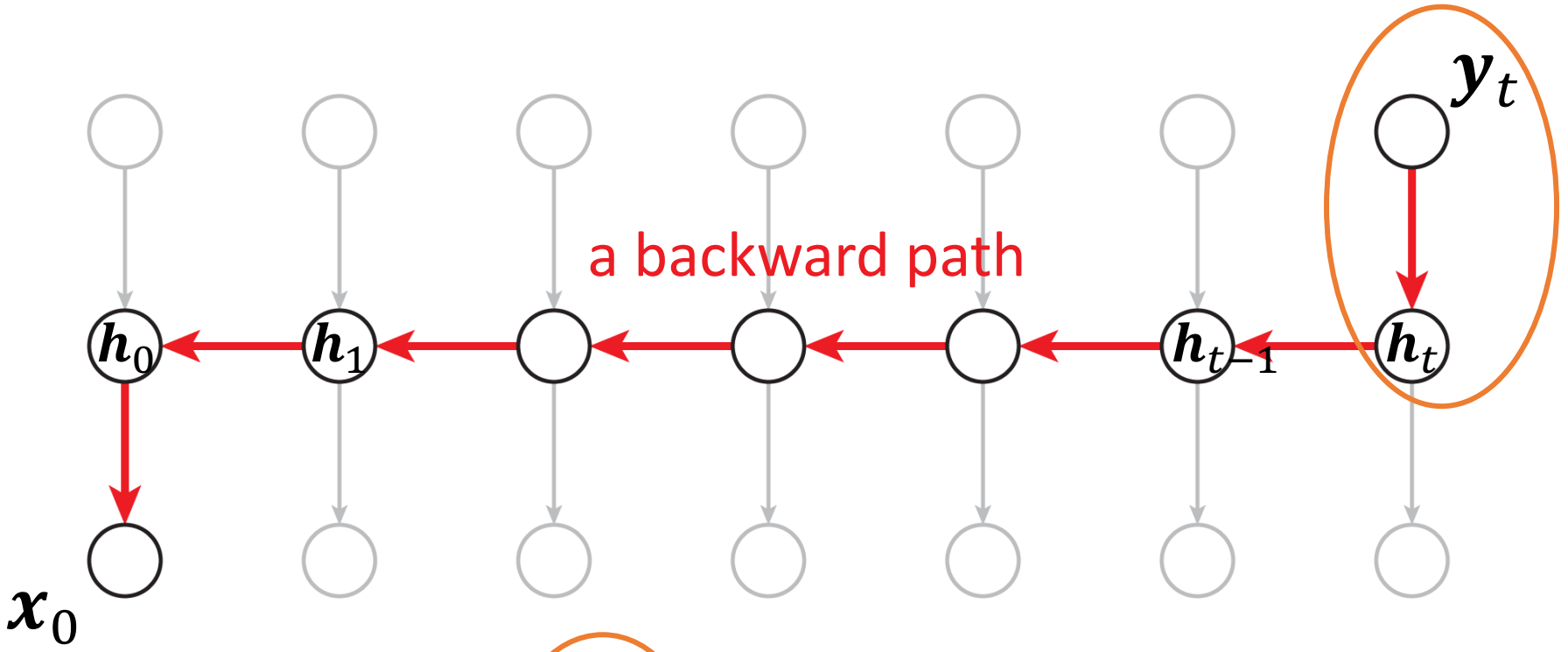
- output at time t is like a t-layer deep net



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{h}_0} \frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

RNN: BackProp Through Time (BPTT)

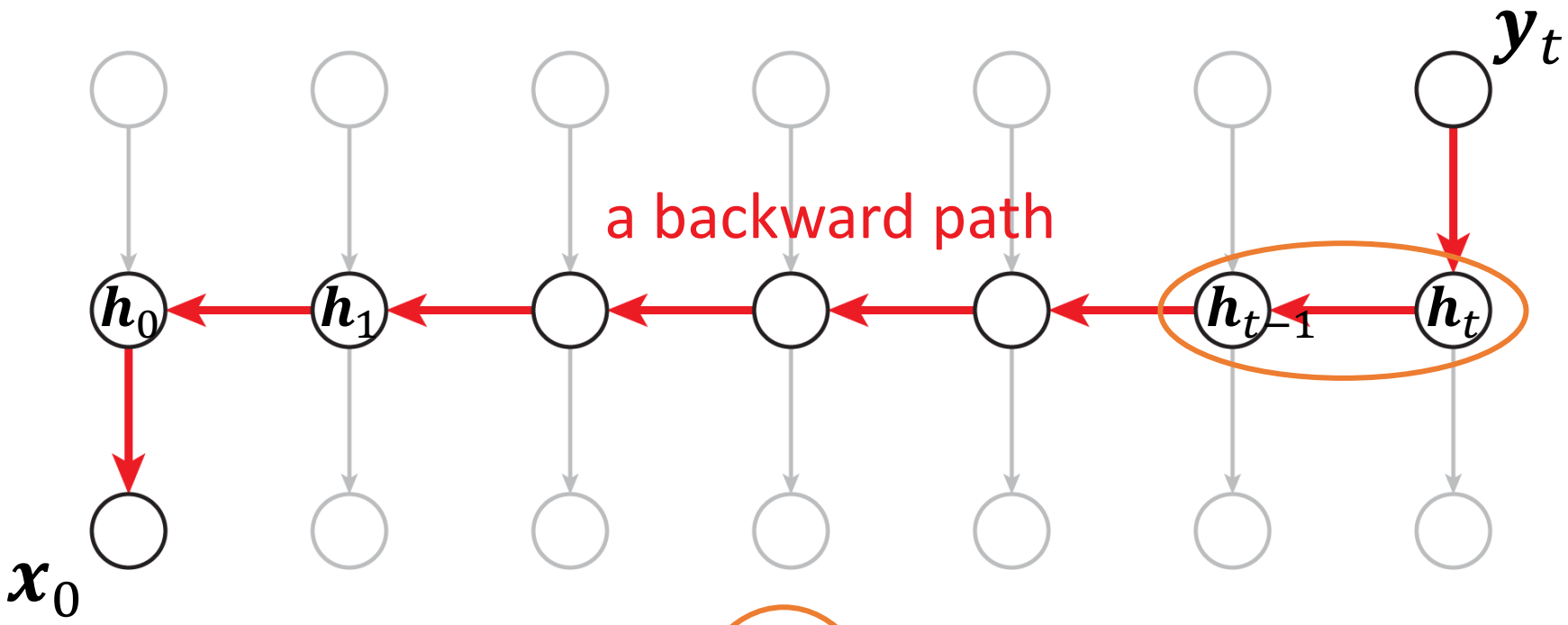
- output at time t is like a t-layer deep net



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{h}_0} \frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

RNN: BackProp Through Time (BPTT)

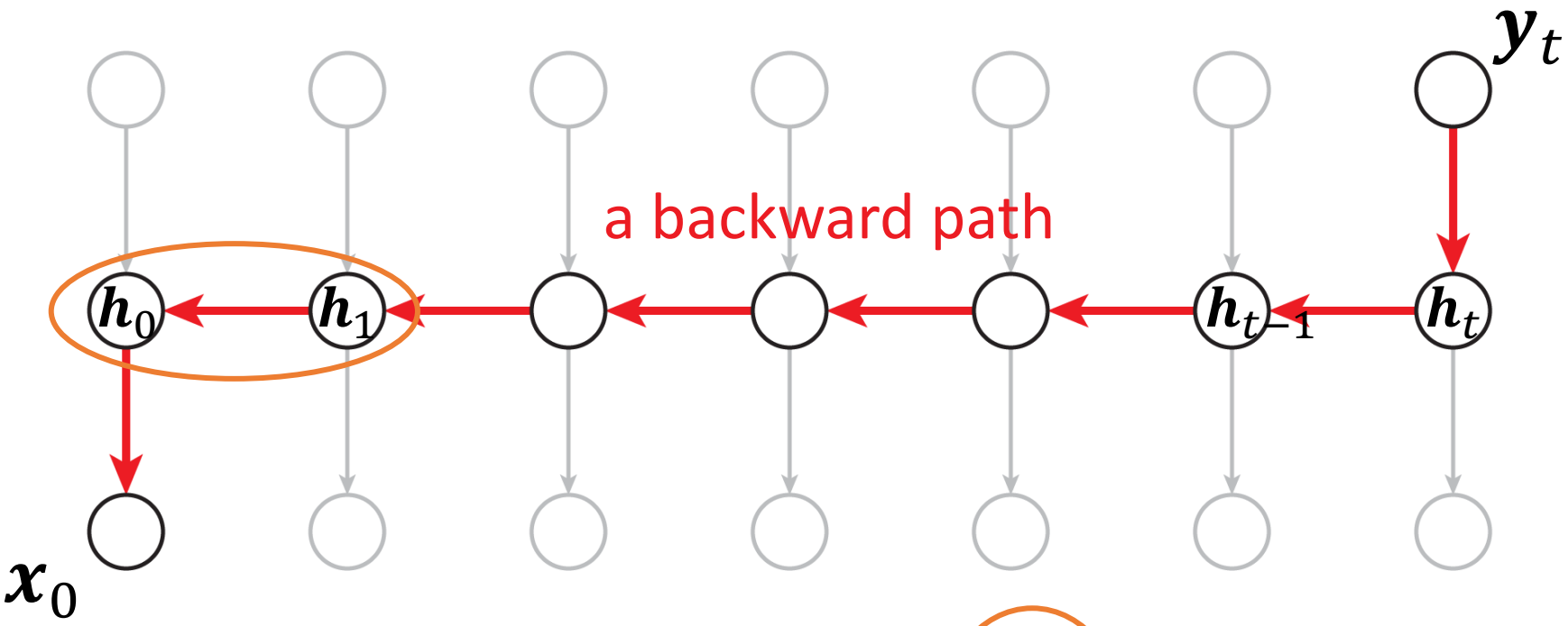
- output at time t is like a t-layer deep net



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{h}_0} \frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

RNN: BackProp Through Time (BPTT)

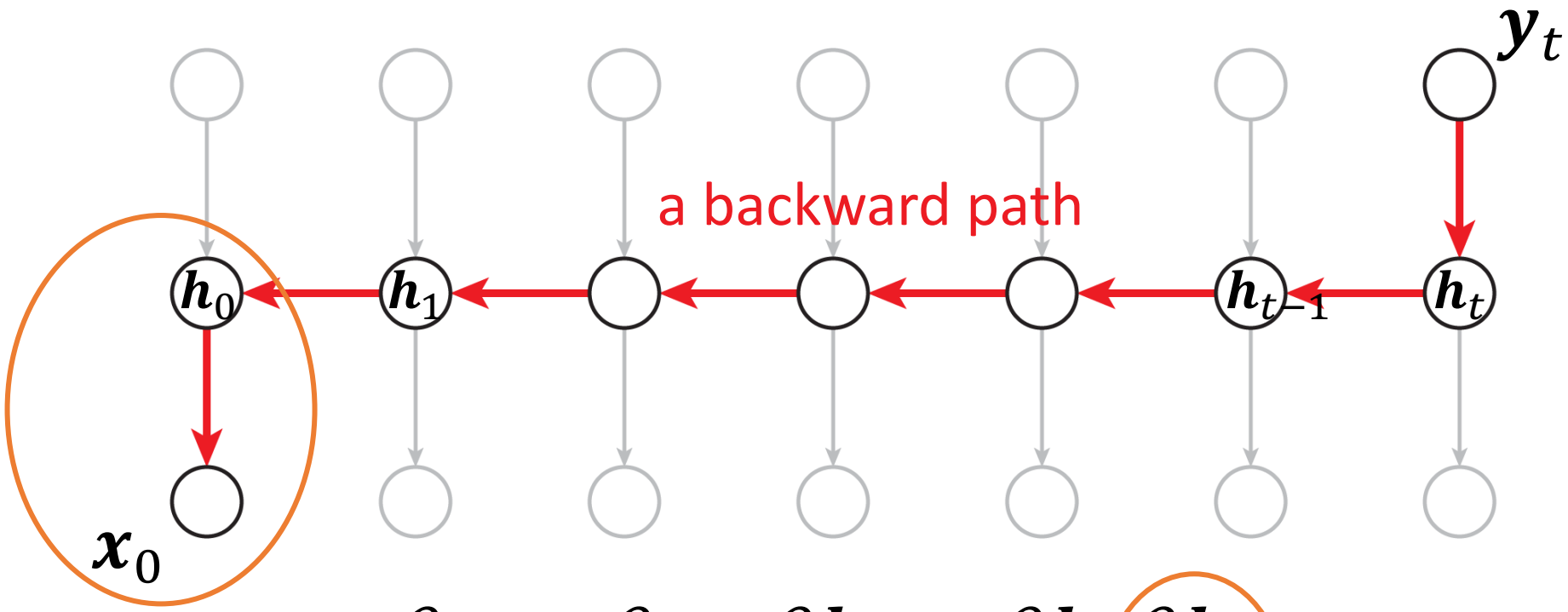
- output at time t is like a t-layer deep net



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{h}_0} \frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

RNN: BackProp Through Time (BPTT)

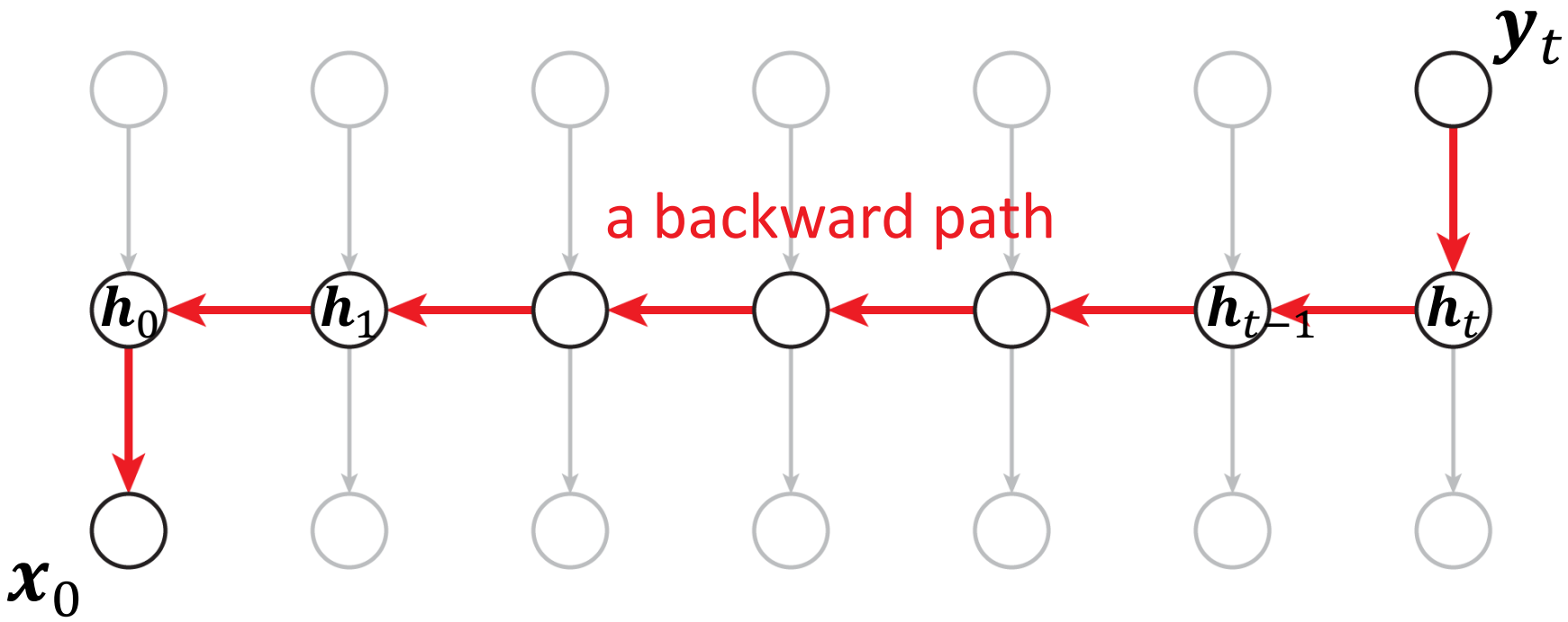
- output at time t is like a t-layer deep net



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{h}_0} \frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

RNN: BackProp Through Time (BPTT)

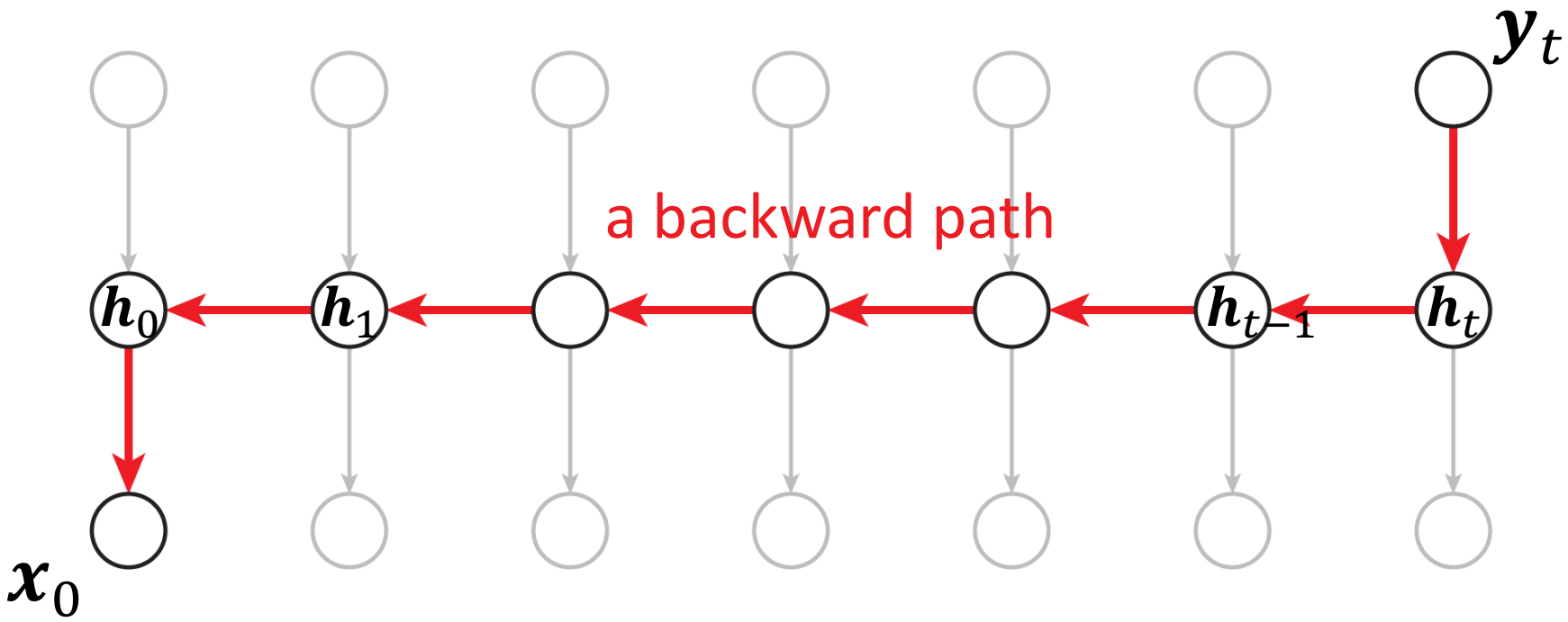
Example (linear vanilla RNN): $\boldsymbol{h}_t = W\boldsymbol{h}_{t-1} + U\boldsymbol{x}_t$



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t}\frac{\partial \boldsymbol{h}_t}{\partial \boldsymbol{h}_{t-1}} \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{h}_0}\frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

RNN: BackProp Through Time (BPTT)

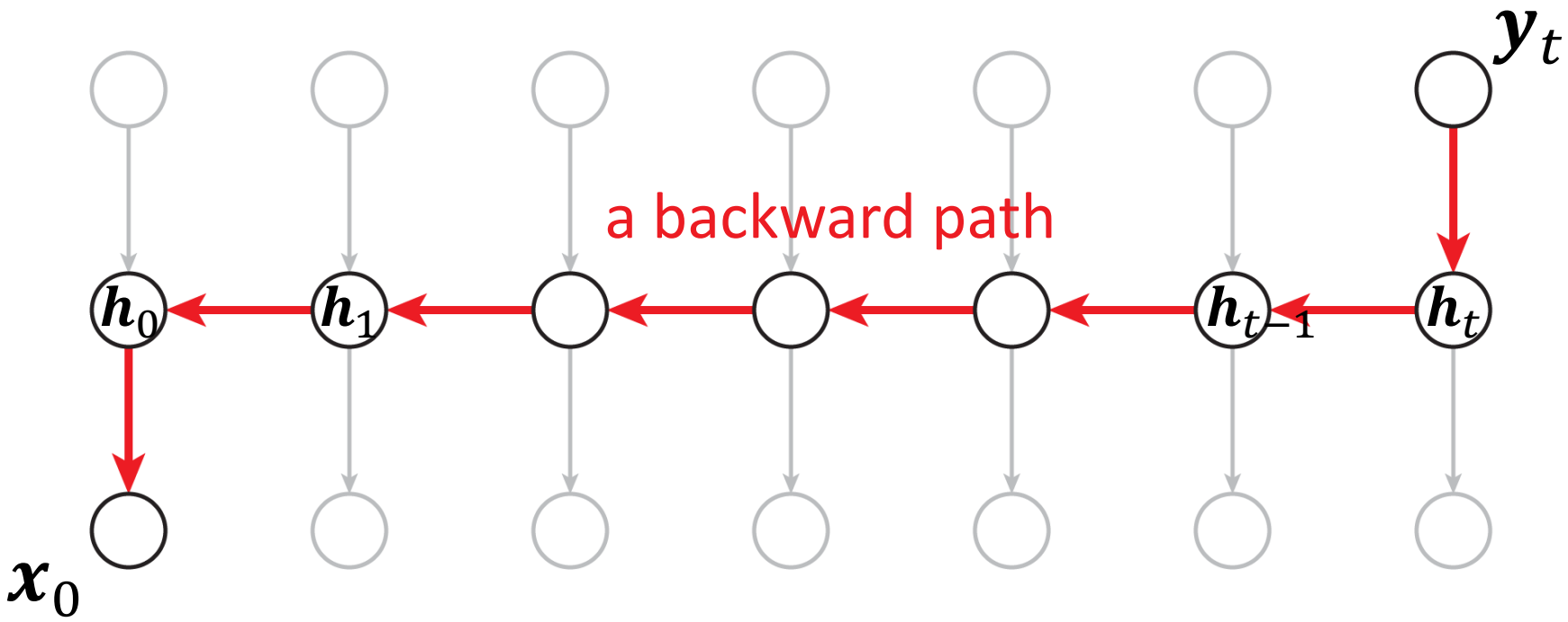
Example (linear vanilla RNN): $\boldsymbol{h}_t = W\boldsymbol{h}_{t-1} + U\boldsymbol{x}_t$



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} W \cdots \frac{\partial \boldsymbol{h}_1}{\partial \boldsymbol{h}_0}\frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

RNN: BackProp Through Time (BPTT)

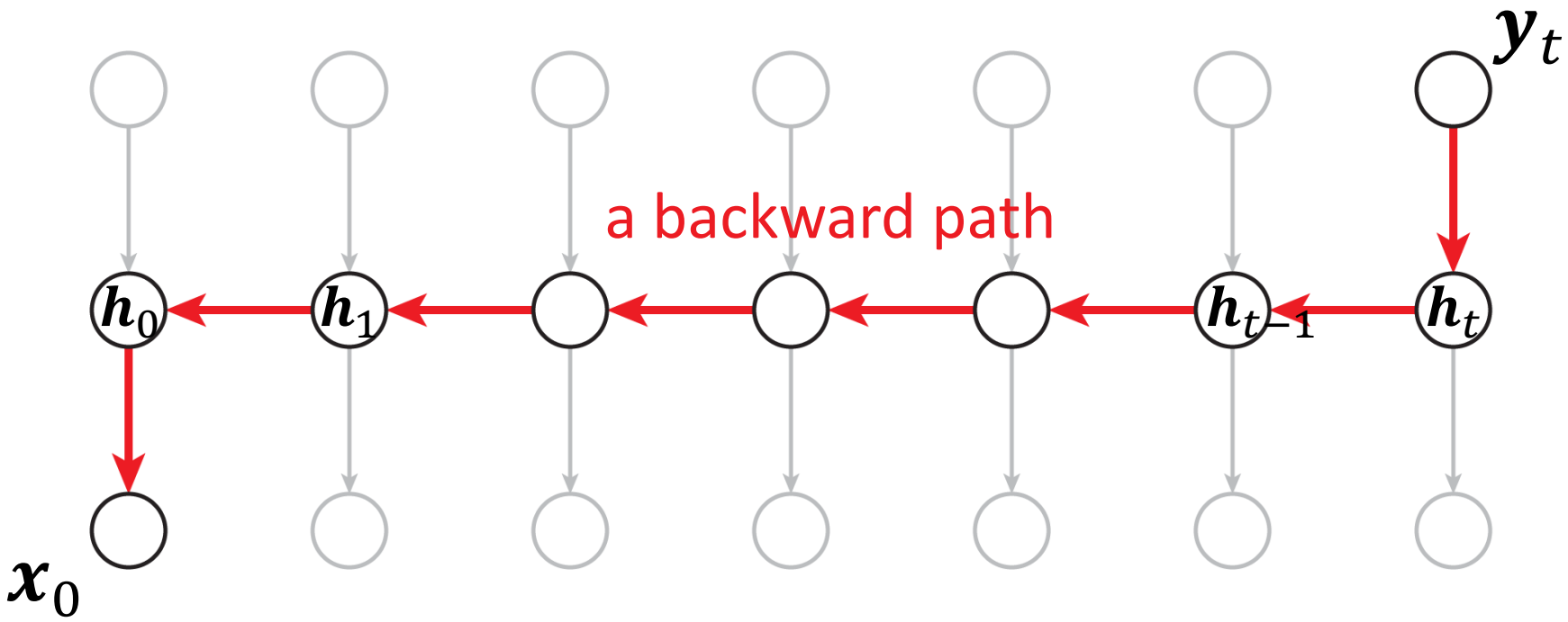
Example (linear vanilla RNN): $\boldsymbol{h}_t = W\boldsymbol{h}_{t-1} + U\boldsymbol{x}_t$



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} \underbrace{W \quad \dots W}_{t\times} \frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

RNN: BackProp Through Time (BPTT)

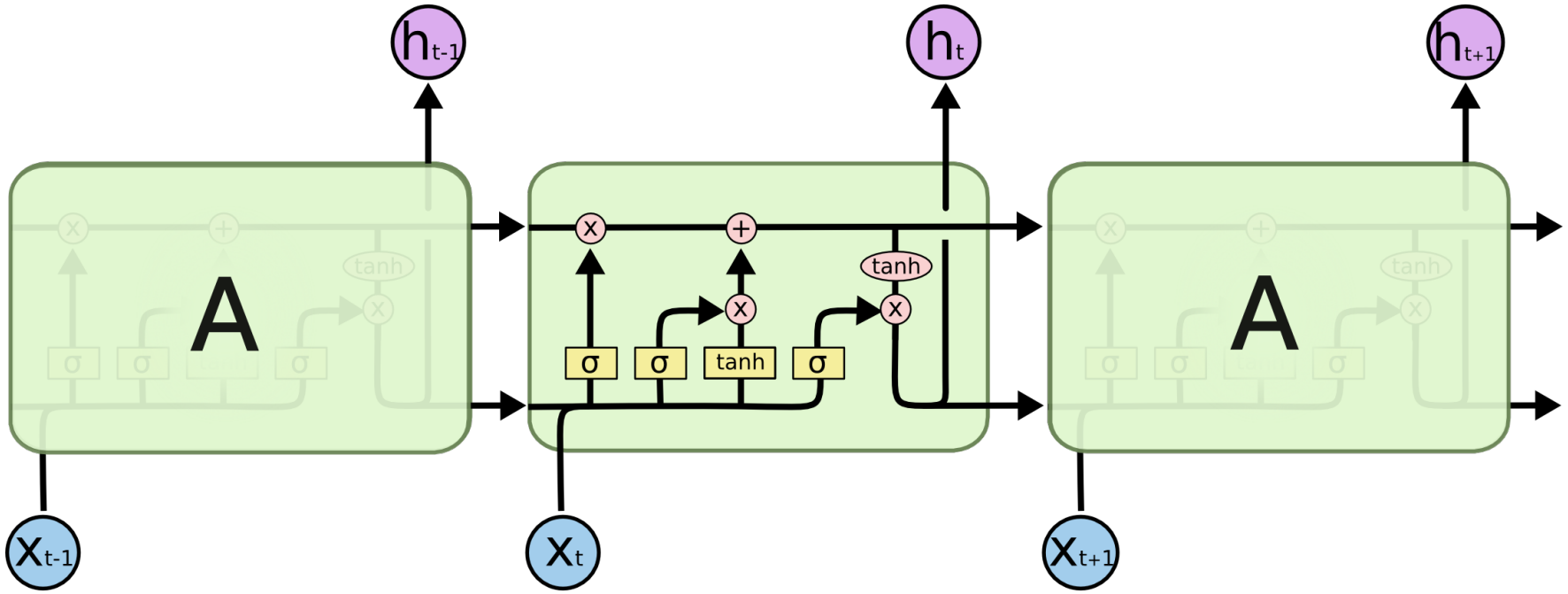
Example (linear vanilla RNN): $\boldsymbol{h}_t = W\boldsymbol{h}_{t-1} + U\boldsymbol{x}_t$



$$\frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{x}_0} = \frac{\partial \boldsymbol{y}_t}{\partial \boldsymbol{h}_t} W^t \frac{\partial \boldsymbol{h}_0}{\partial \boldsymbol{x}_0}$$

Long Short-Term Memory (LSTM) unit

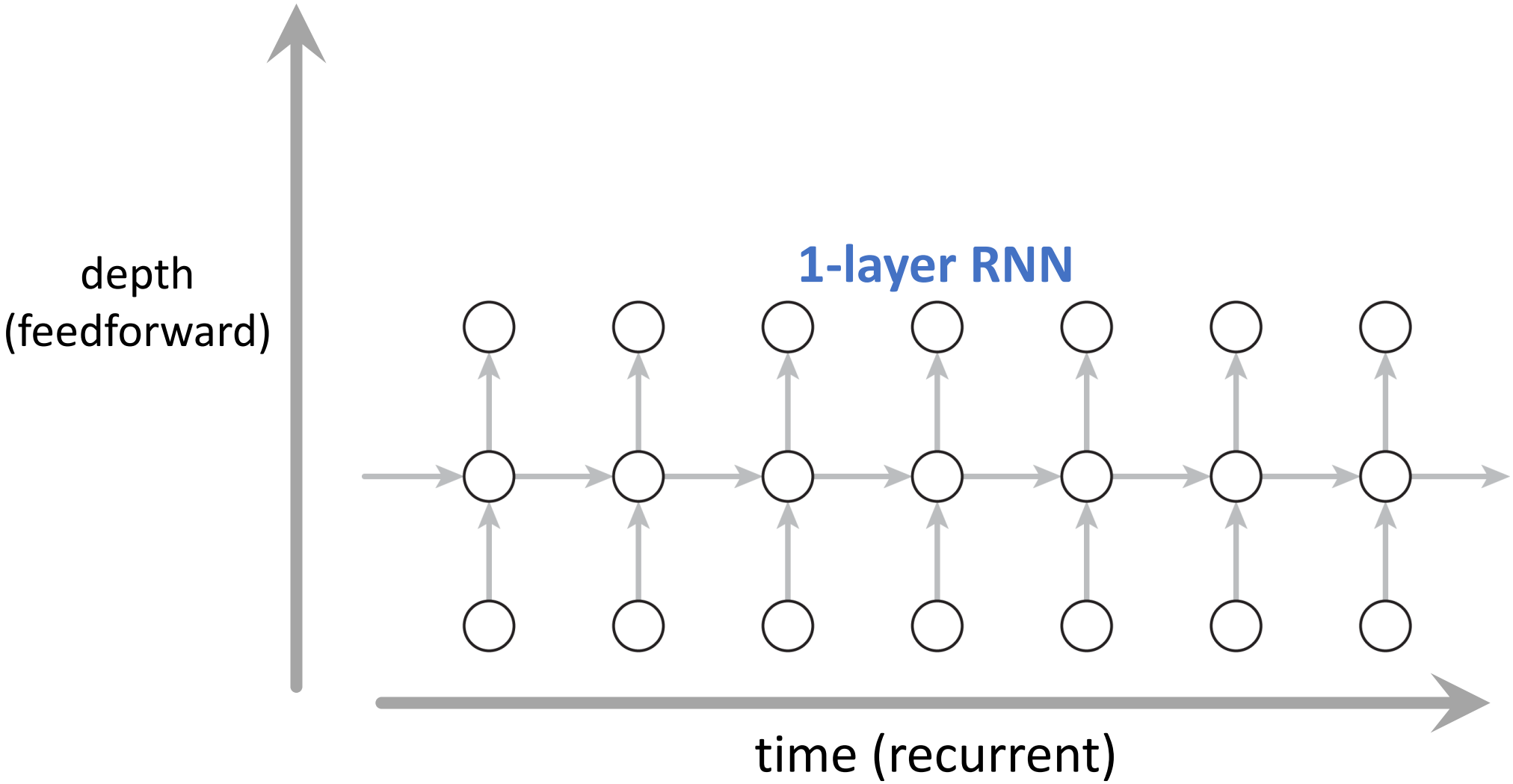
- Recurrent NN w/ gating



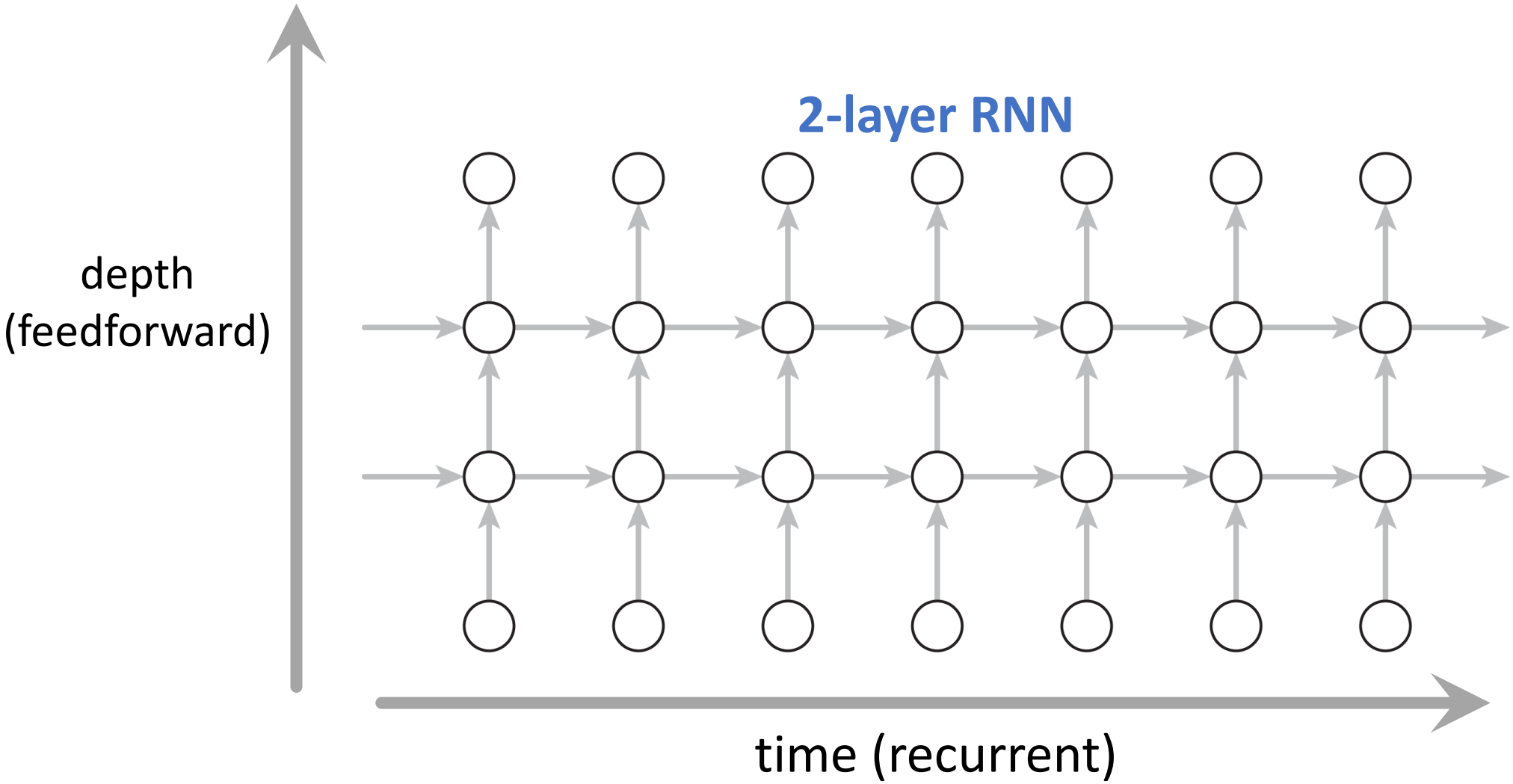
See Christopher Olah's blog (highly recommended!)
https://colah.github.io/posts/2015-08-Understanding-LSTMs/

Hochreiter & Schmidhuber, "Long Short-Term Memory", 1997

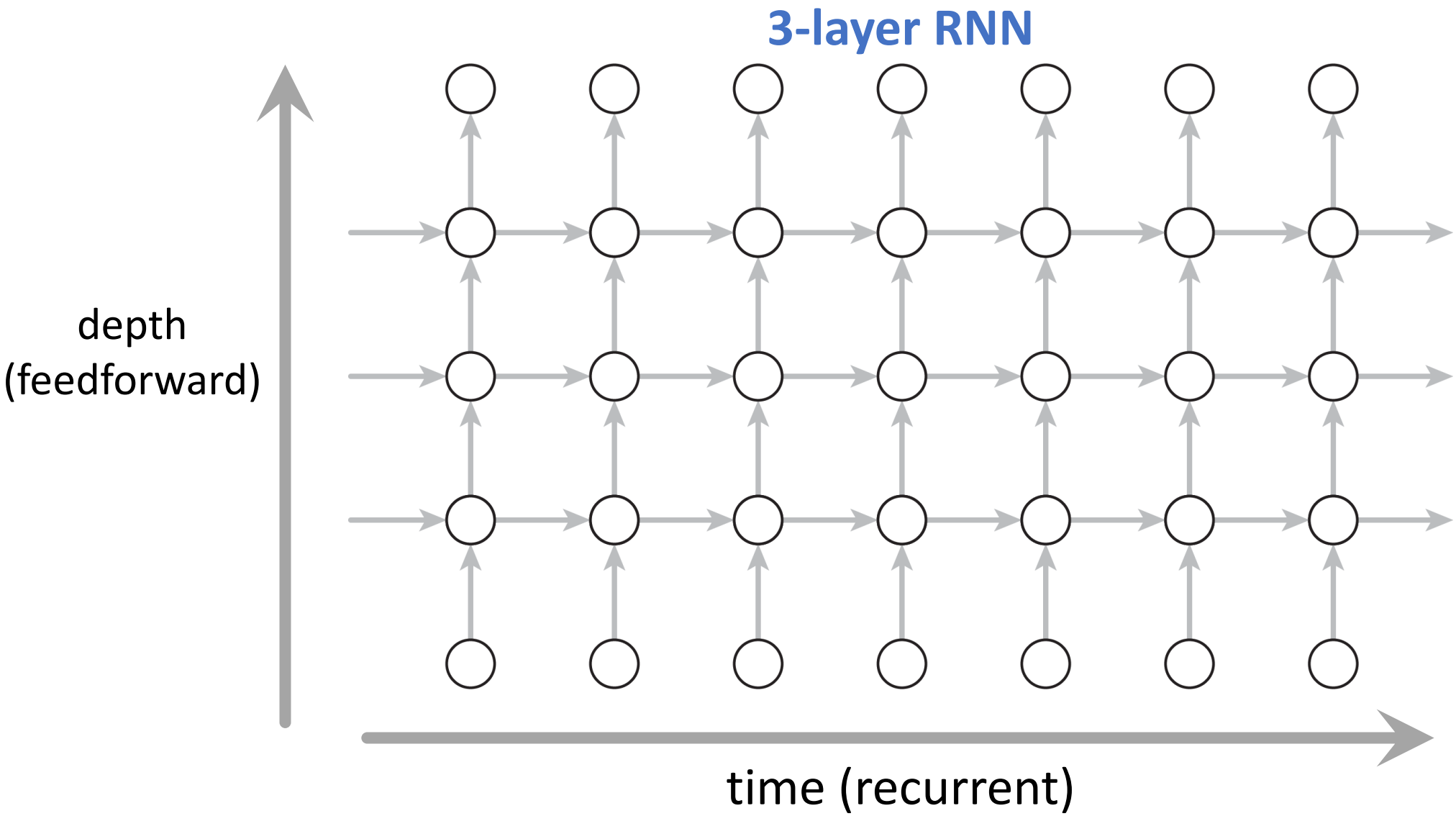
Deep RNN



Deep RNN

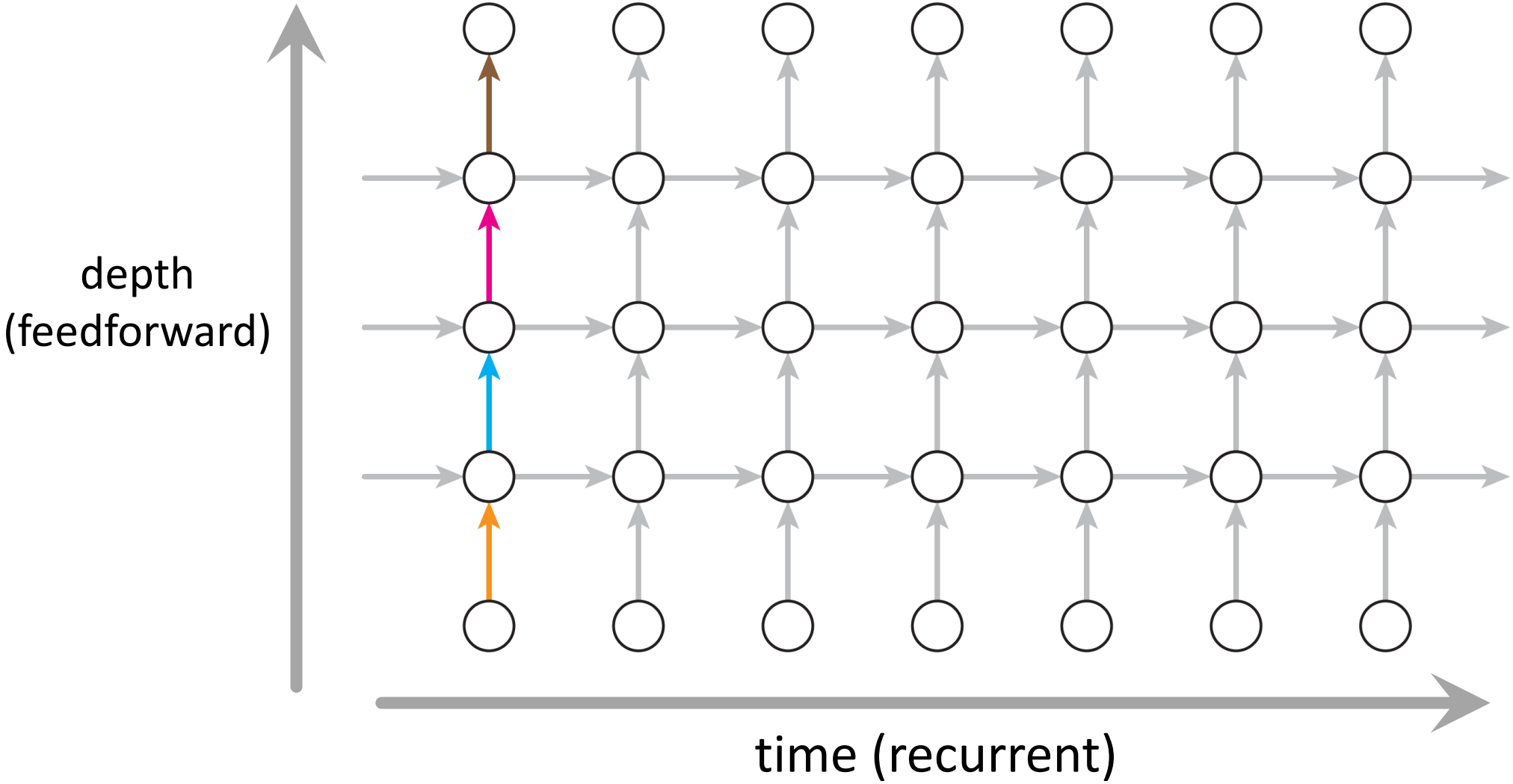


Deep RNN



Deep RNN

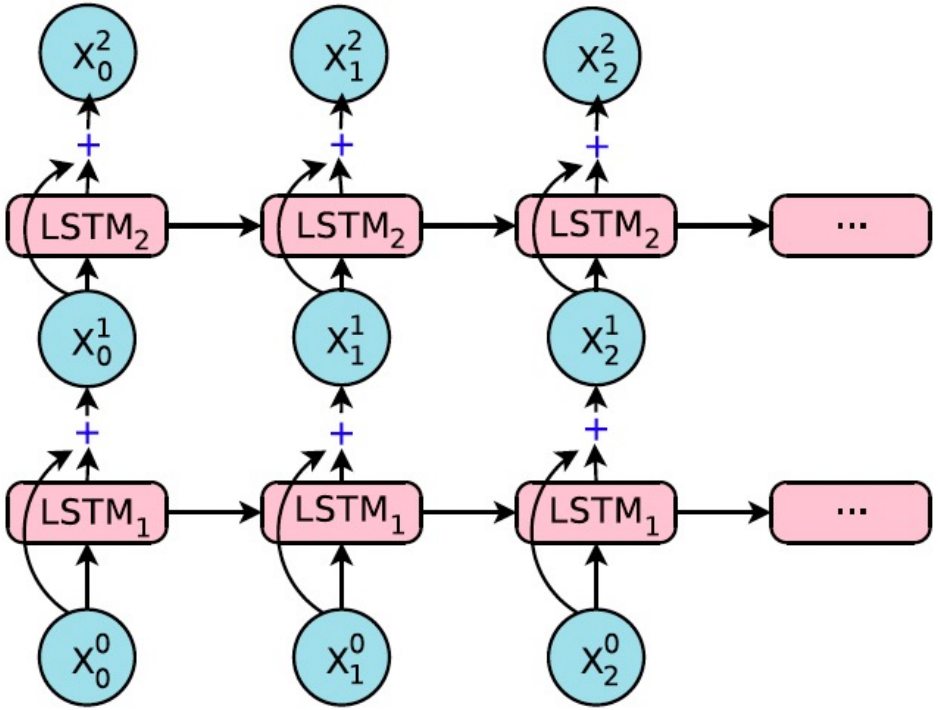
going **deep** in this direction



Example: Google's Neural Machine Translation System 2016

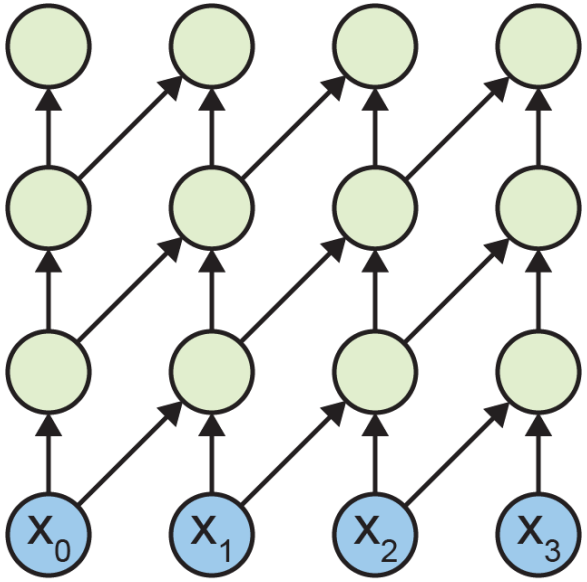
Deep RNN with LSTM units

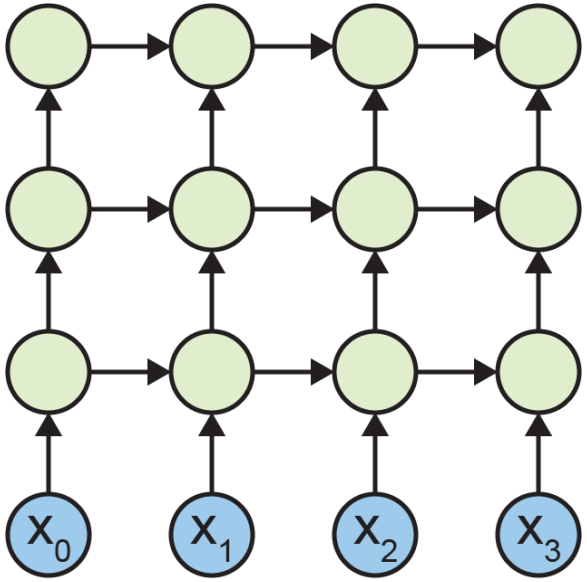
- 16 layers enabled w/ residual connections
- degrade >4 layers w/o residual connections



"Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation", Wu, et al., 2016

Sequence Modeling: CNN vs. RNN





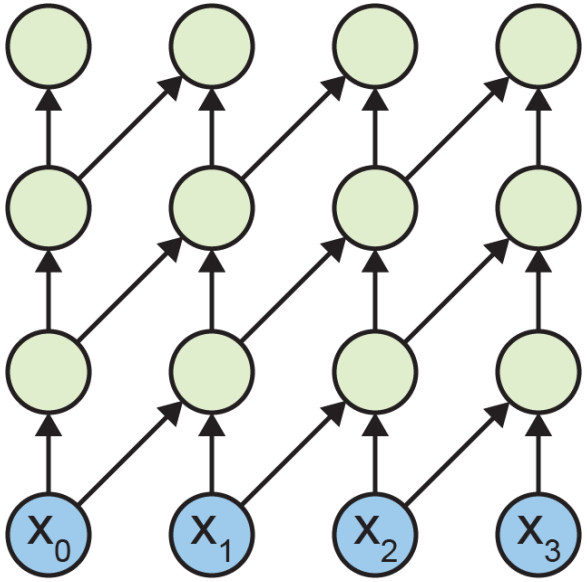
CNN

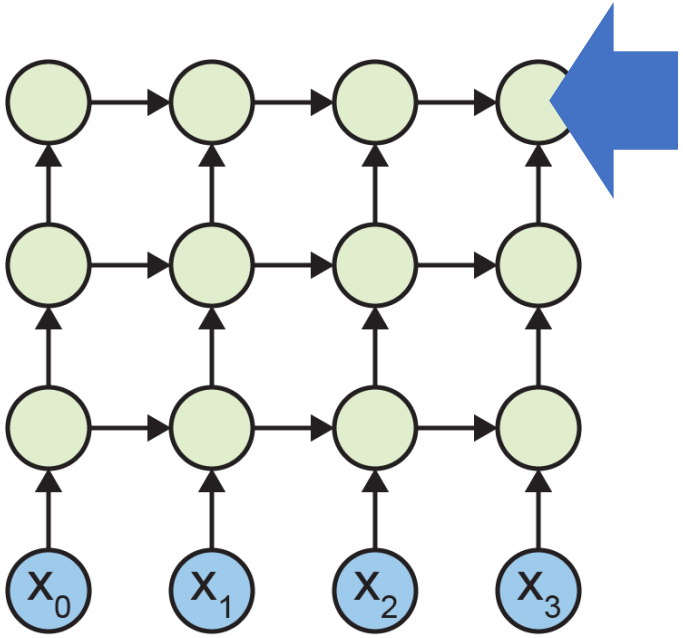
- limited context
- feedforward

RNN

- long context
- not feedforward

Sequence Modeling: CNN vs. RNN





Not feedforward:
step at time t must wait for all steps at time < t to complete.

(not parallelism-friendly)

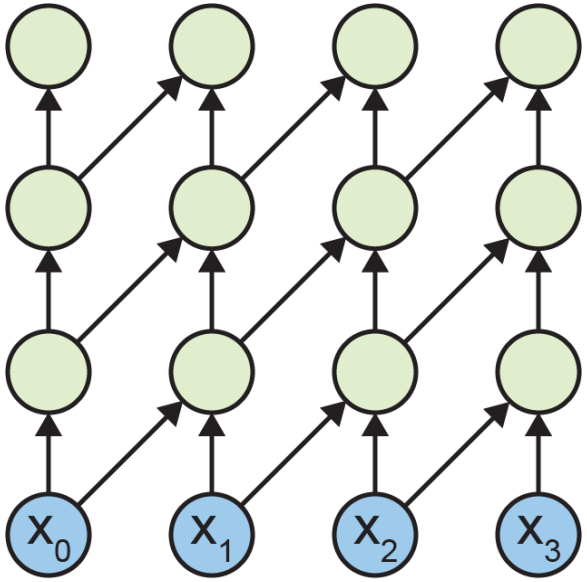
CNN

- limited context
- feedforward

RNN

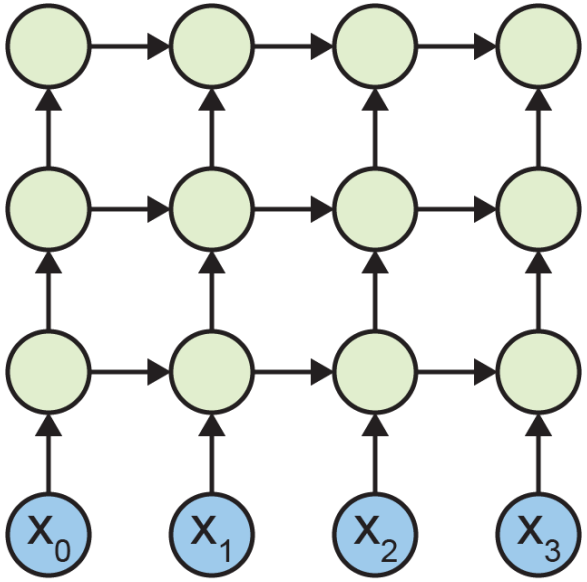
- long context
- not feedforward

Sequence Modeling: CNN vs. RNN vs. ?



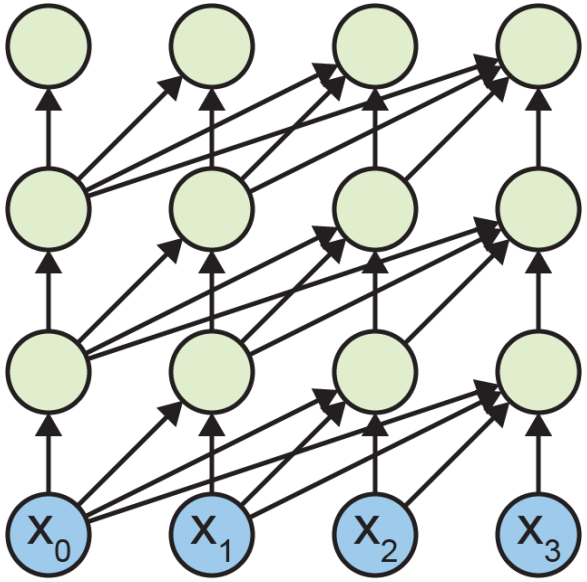
CNN

- limited context
- feedforward



RNN

- long context
- not feedforward



Attention

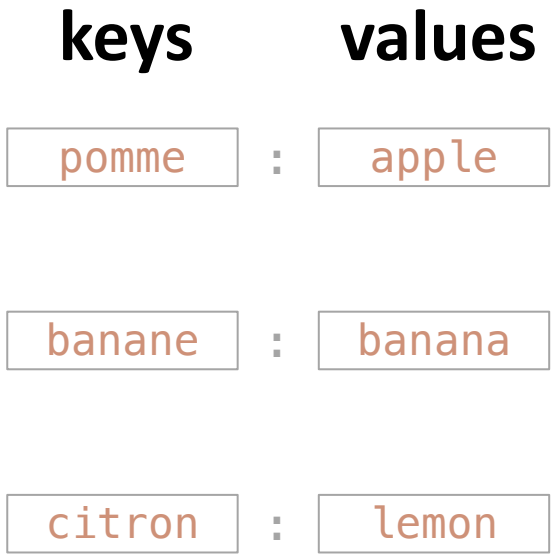
best of both world?

Attention and Transformers

Dictionary look-up

Think of a Python dict:

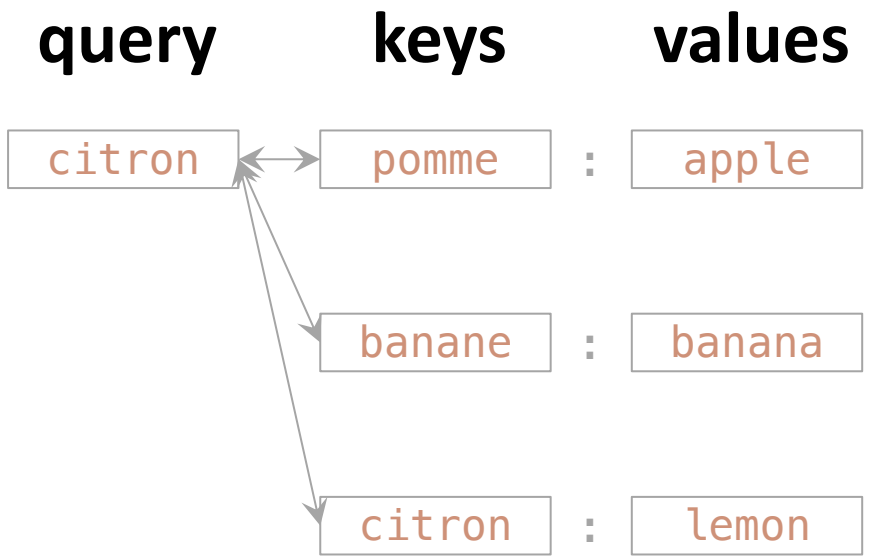
```
dict_fr2en = {
    "pomme": "apple",
    "banane": "banana",
    "citron": "lemon"
}
```



Dictionary look-up

Think of a Python dict:

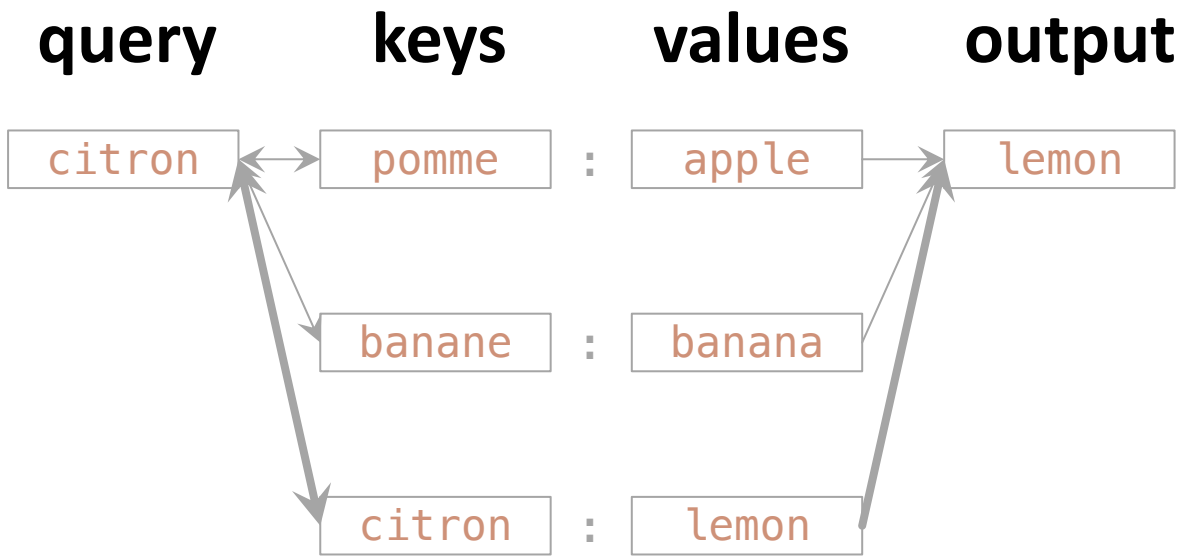
```
dict_fr2en = {
    "pomme": "apple",
    "banane": "banana",
    "citron": "lemon"
}

query = "citron"
output = dict_fr2en[query]
```



Dictionary look-up

Think of a Python dict:

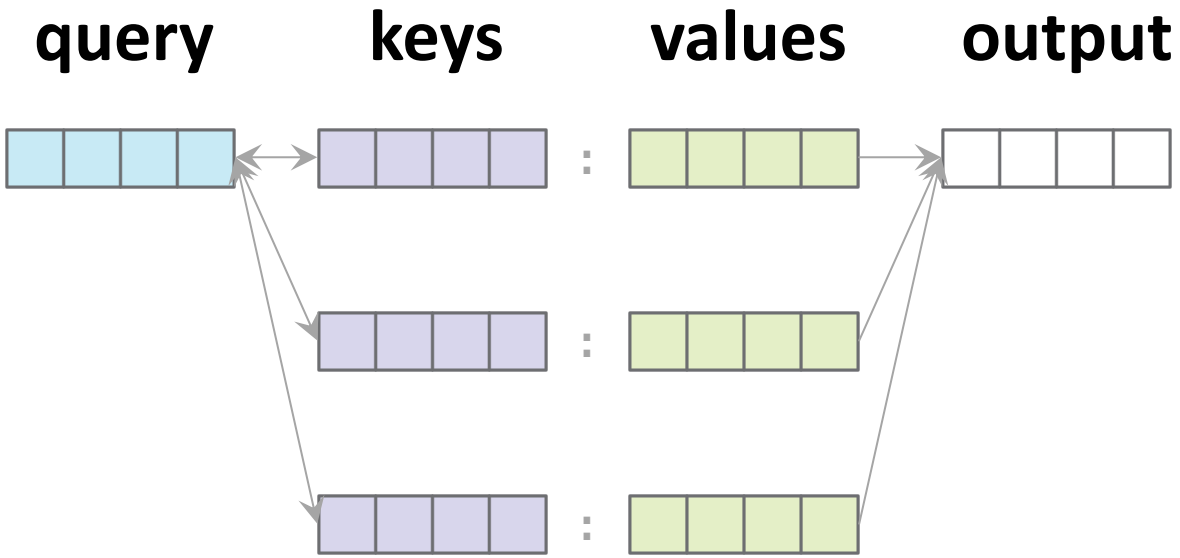
```
dict_fr2en = {
    "pomme": "apple",
    "banane": "banana",
    "citron": "lemon"
}

query = "citron"
output = dict_fr2en[query]
```



Dictionary look-up

Think of a Python dict:

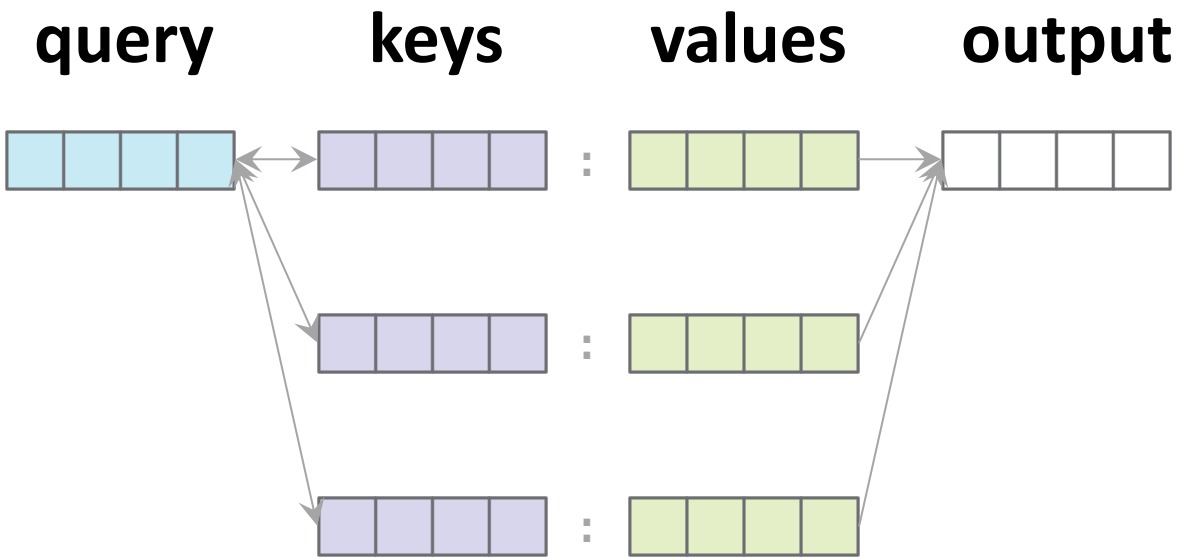
```
dict_fr2en = {
    "pomme": "apple",
    "banane": "banana",
    "citron": "lemon"
}

query = "citron"
output = dict_fr2en[query]
```



What if everything is an abstract representation (“embedding”)?

Attention

Attention is **soft** dict look-up in the representation space.

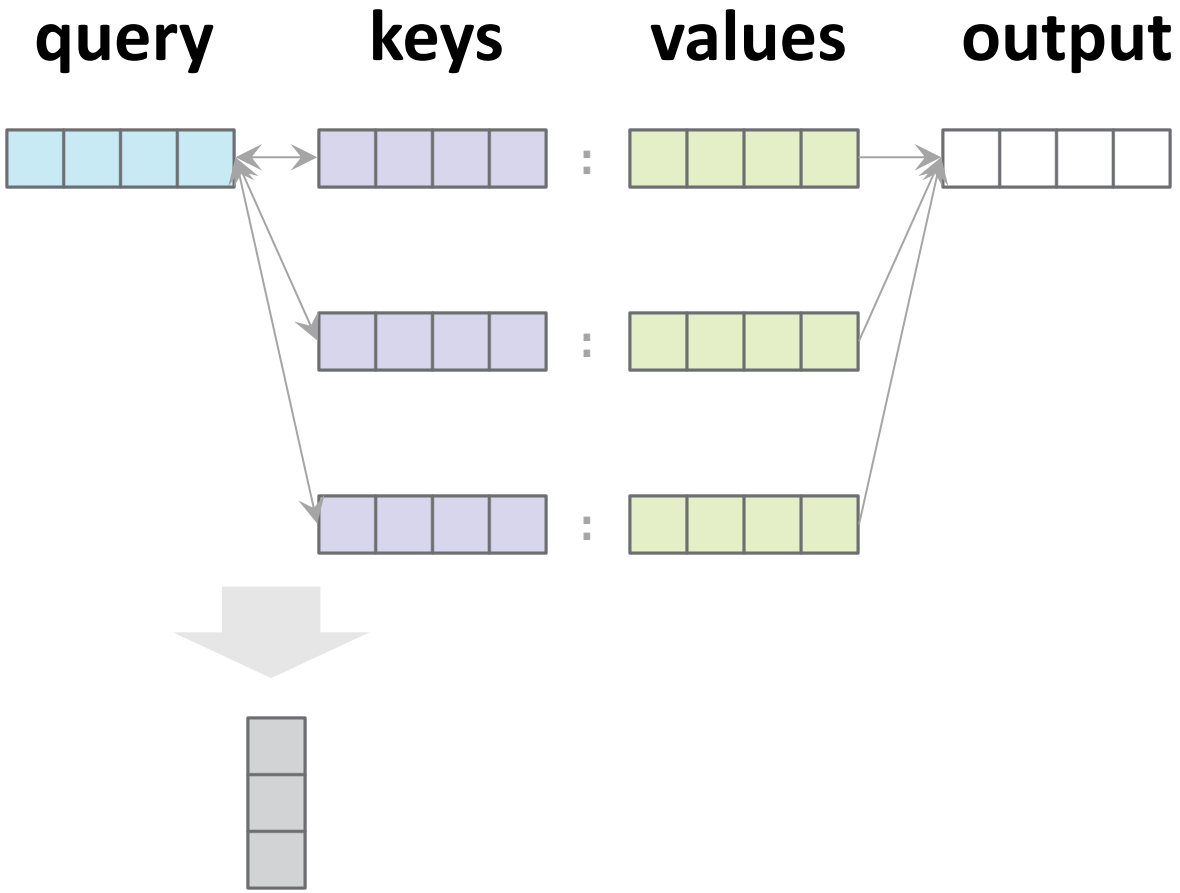


Attention

Attention is **soft** dict look-up in the representation space.

1. query-key similarity:

$$s_i = \boldsymbol{q} \cdot \boldsymbol{k}_i$$



Attention

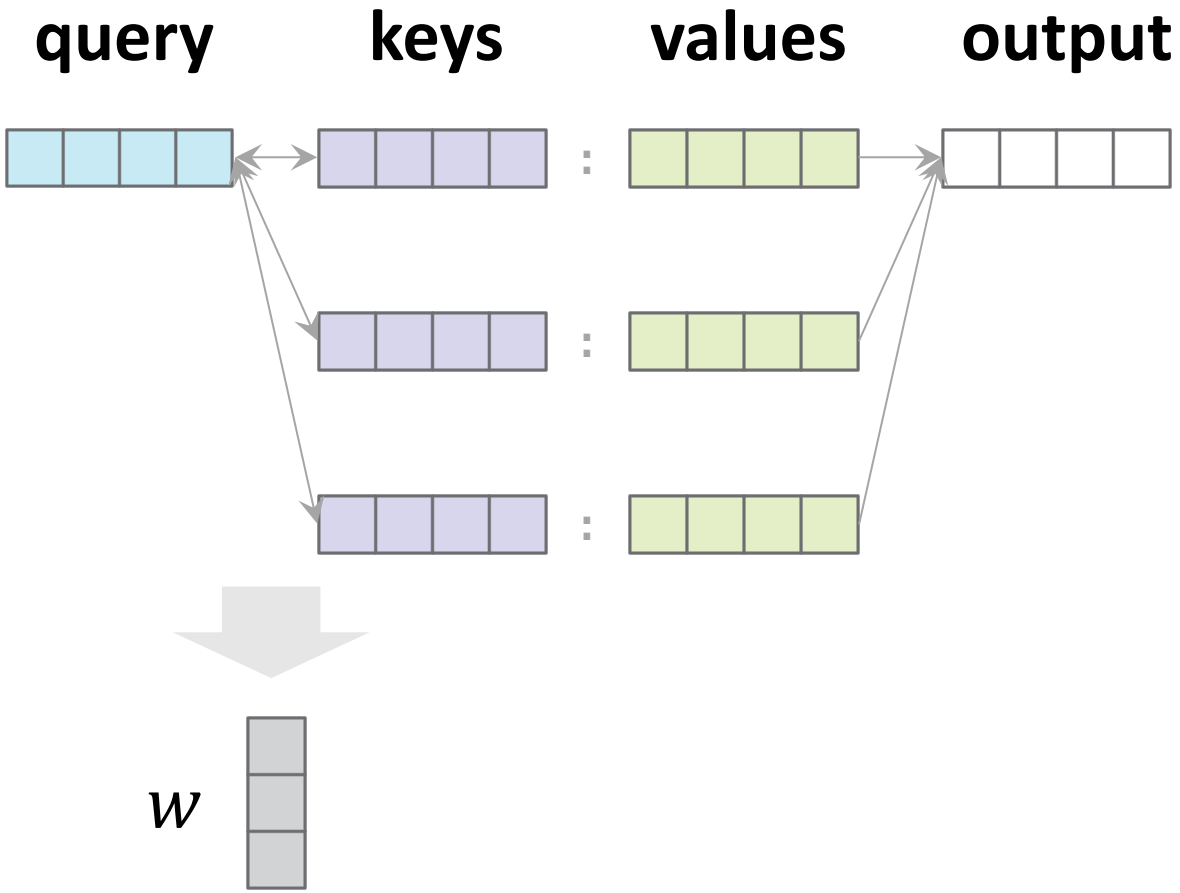
Attention is **soft** dict look-up in the representation space.

1. query-key similarity:

$$s_i = \boldsymbol{q} \cdot \boldsymbol{k}_i$$

2. attention weight (softmax):

$$w_i = e^{s_i} / \Sigma_j e^{s_j}$$



*In practice, the logits s_i are scaled by 1/sqrt(dim) before softmax.

Attention

Attention is **soft** dict look-up in the representation space.

1. query-key similarity:

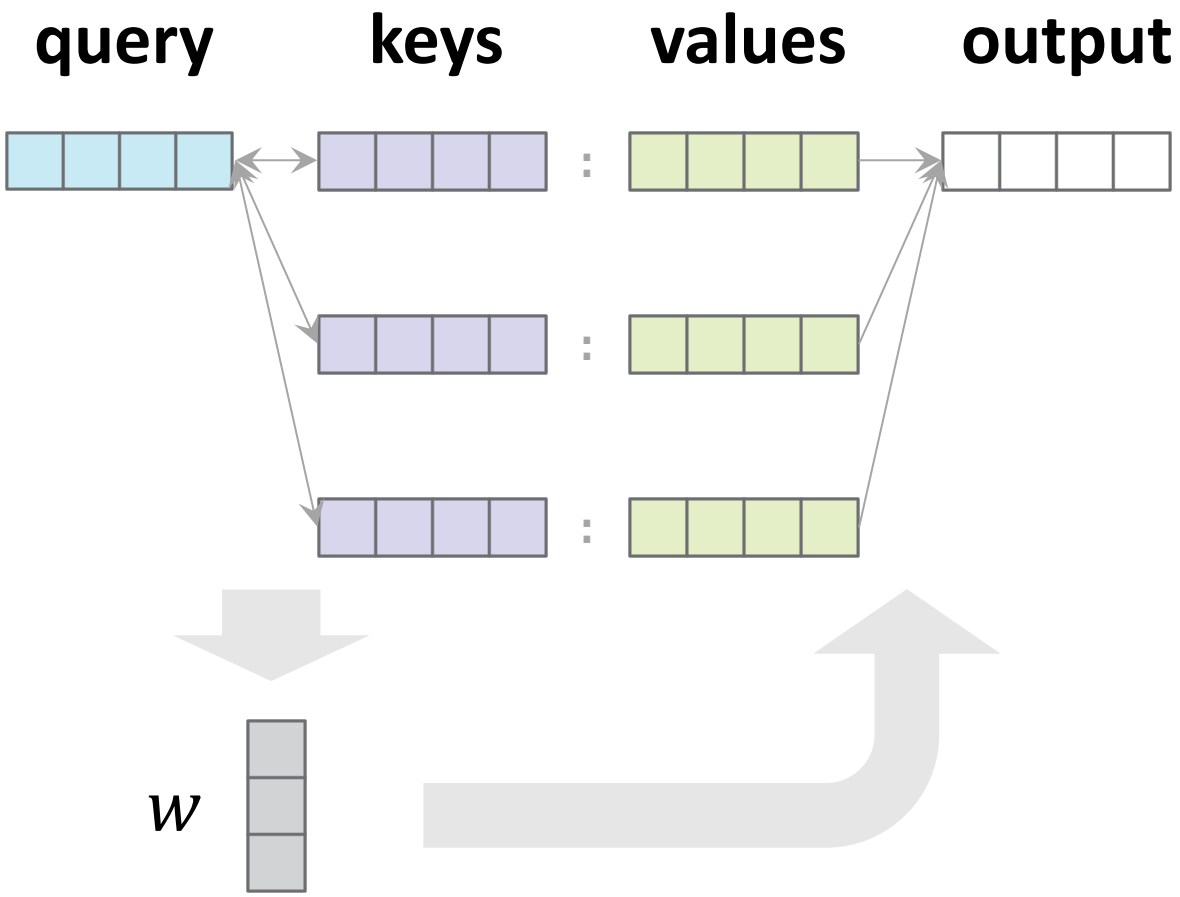
$$s_i = \boldsymbol{q} \cdot \boldsymbol{k}_i$$

2. attention weight (softmax):

$$w_i = e^{s_i} / \Sigma_j e^{s_j}$$

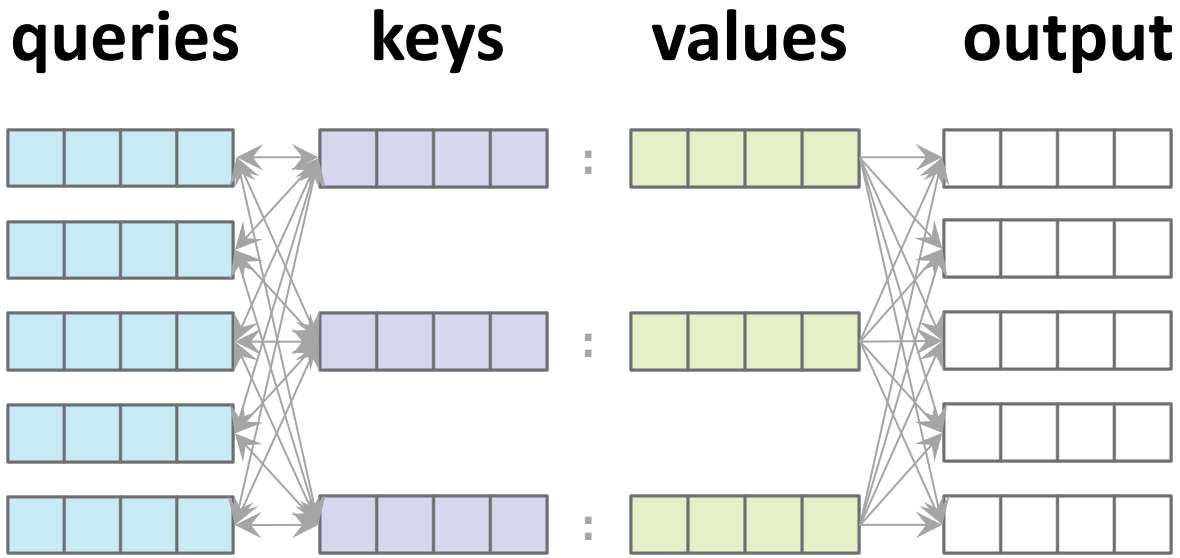
3. weighted average:

$$\boldsymbol{z} = \Sigma_i w_i \boldsymbol{v}_i$$



Attention

Multiple queries

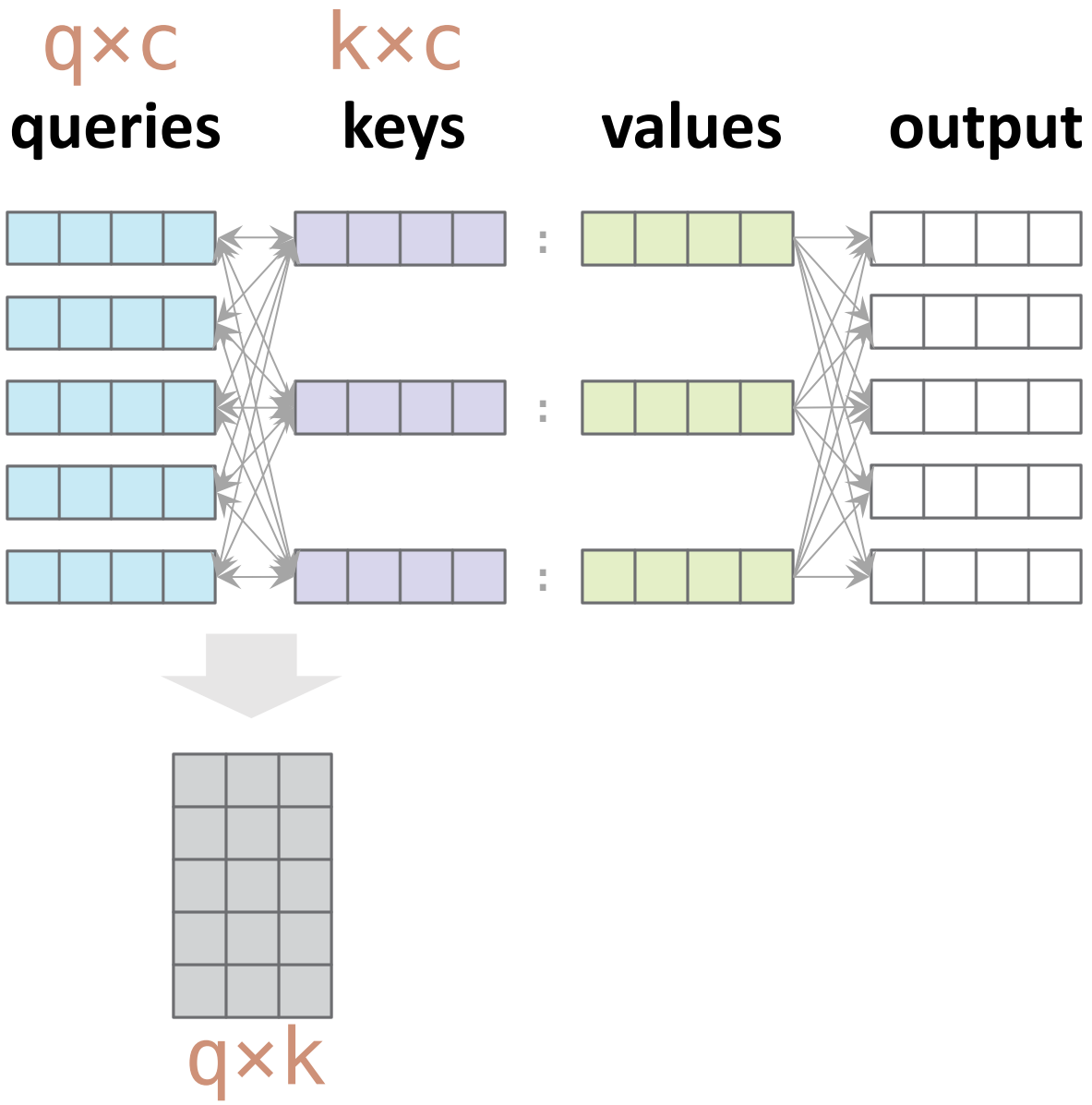


Attention

Multiple queries

1. query-key similarity:

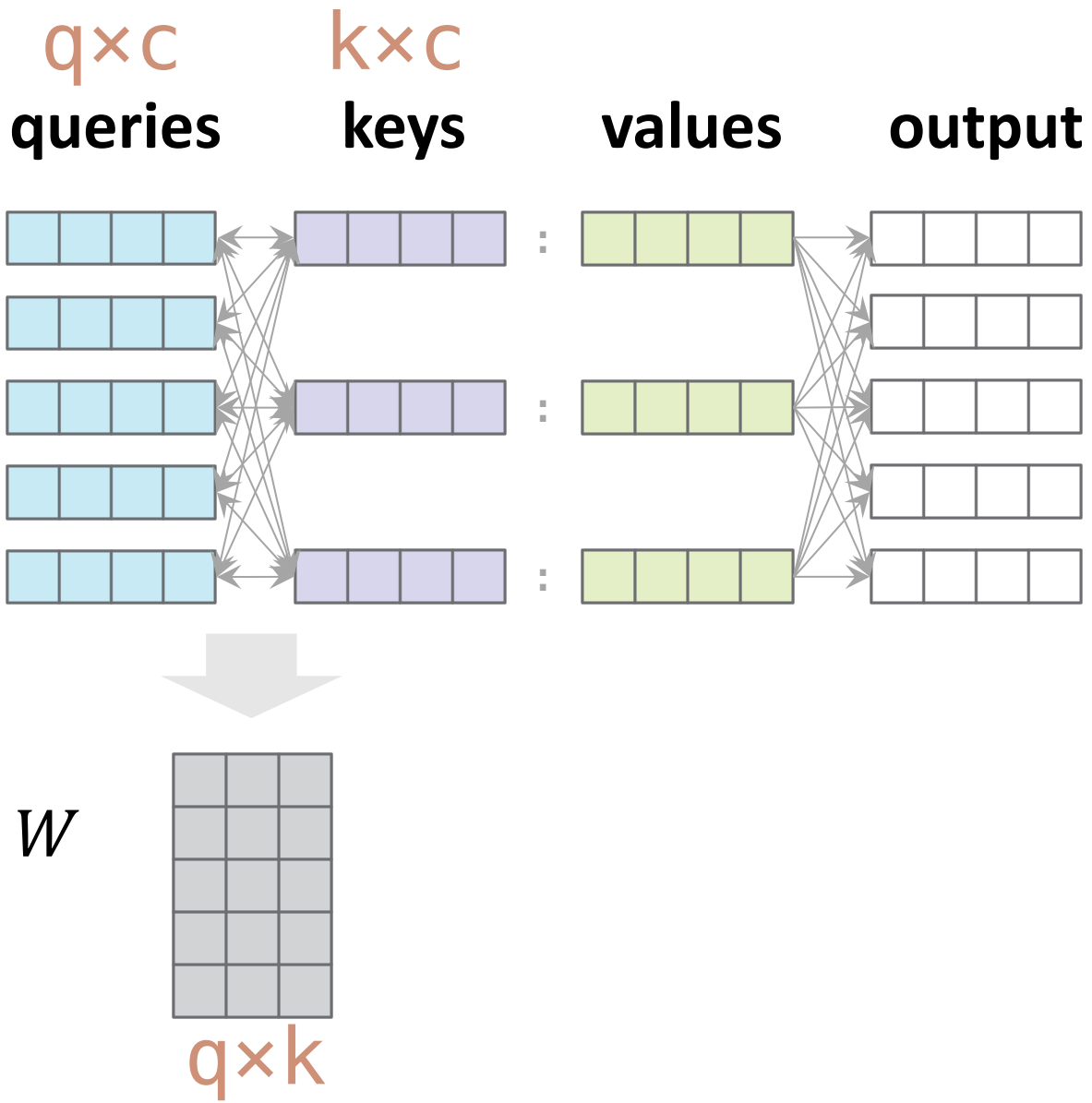
$$S = QK^{\mathrm{T}}$$



Attention

Multiple queries

1. query-key similarity:
$$S = QK^{\mathrm{T}}$$
2. attention weight (softmax):
$$W = \mathrm{softmax}(S)$$

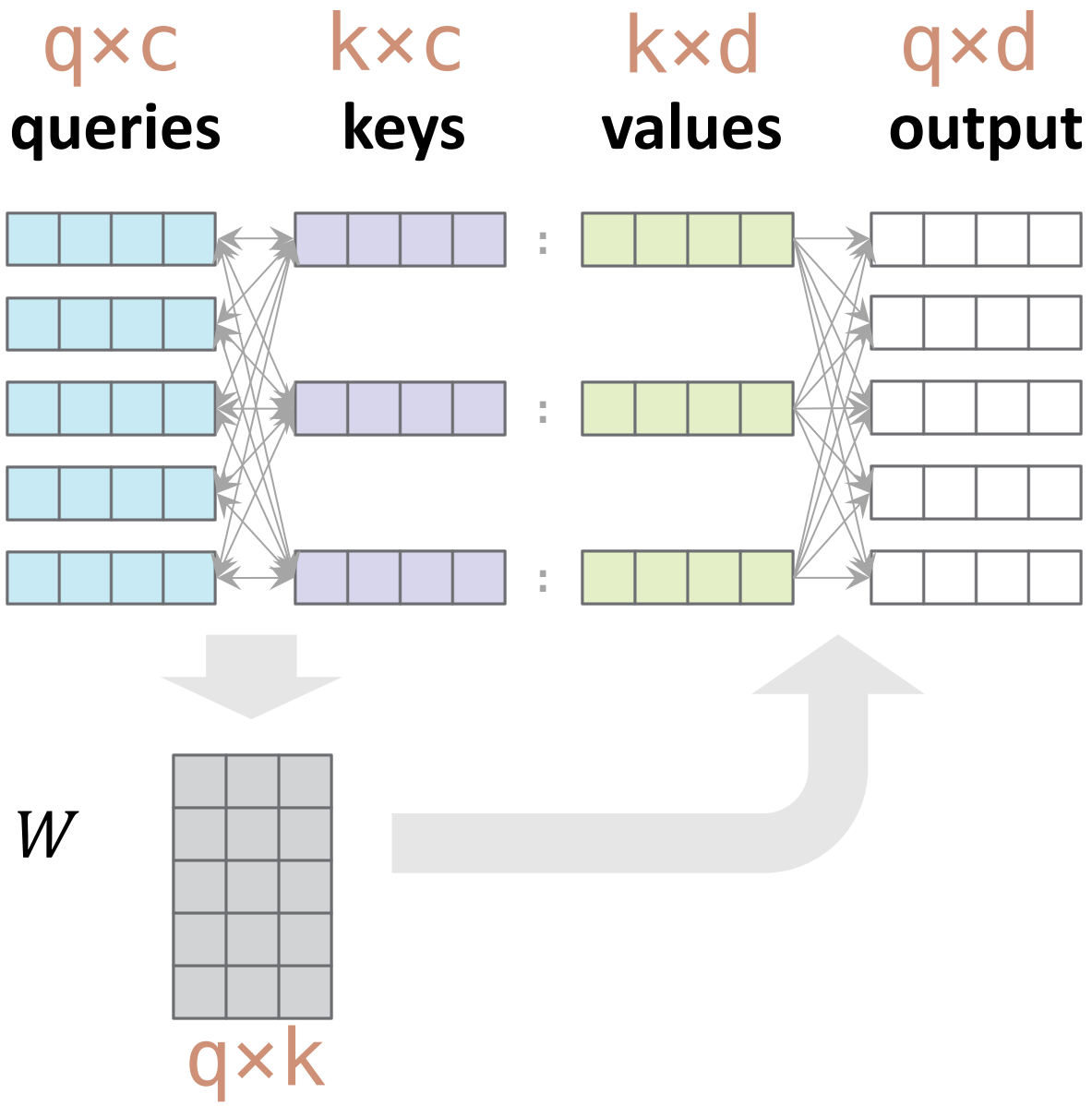


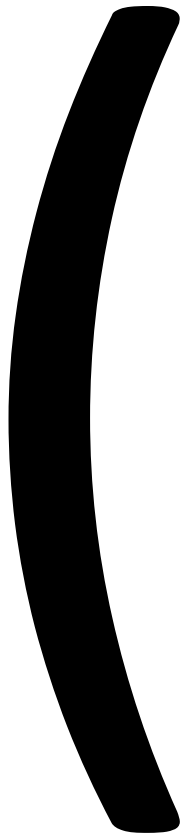
*In practice, the logits s_i are scaled by 1/sqrt(dim) before softmax.

Attention

Multiple queries

1. query-key similarity:
$$S = QK^{\mathrm{T}}$$
2. attention weight (softmax):
$$W = \mathrm{softmax}(S)$$
3. weighted average:
$$Z = WV$$





Einstein Sum: notations of brevity

Tim Rocktäschel, “Einsum is all you need”, https://rockt.github.io/2018/04/30/einsum

Einstein Sum: notations of brevity

Traditional notation (e.g., matrix-matrix product):

let $A \in \mathbb{R}^{\mathrm{I}\times\mathrm{K}}$, $B \in \mathbb{R}^{\mathrm{K}\times\mathrm{J}}$,

$$C = AB$$

where $C \in \mathbb{R}^{\mathrm{I}\times\mathrm{J}}$ with:

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

Tim Rocktäschel, “Einsum is all you need”, https://rockt.github.io/2018/04/30/einsum

Einstein Sum: notations of brevity

Traditional notation (e.g., matrix-matrix product):

let $A \in \mathbb{R}^{\mathrm{I}\times\mathrm{K}}$, $B \in \mathbb{R}^{\mathrm{K}\times\mathrm{J}}$,

$$C = AB$$

where $C \in \mathbb{R}^{\mathrm{I}\times\mathrm{J}}$ with:

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

most important messages are in subscripts

```
C = einsum('ik,kj->ij', A, B)
```

(An index that is summed over won't show in the output)

Tim Rocktäschel, "Einsum is all you need", https://rockt.github.io/2018/04/30/einsum

Examples

```
# column sum
einsum('ij->j', X) # shape: (10, 5) -> (5,)

# row sum
einsum('ij->i', X) # shape: (10, 5) -> (10,)

# sum
einsum('ij->', X) # shape: (10, 5) -> scalar
```

(An index that is summed over won't show in the output)

Tim Rocktäschel, "Einsum is all you need", https://rockt.github.io/2018/04/30/einsum

Examples

```
# transpose
einsum('ij->ji', X) # (10, 5) -> (5, 10)

# permute
einsum('nchw->nhwc', X) # (n, 3, 256, 256) -> (n, 256, 256, 3)
```

more intuitive than calling “permute” directly:

```
permute(X, [0, 2, 3, 1]) # (n, 3, 256, 256) -> (n, 256, 256, 3)
```

Tim Rocktäschel, “Einsum is all you need”, https://rockt.github.io/2018/04/30/einsum

Examples

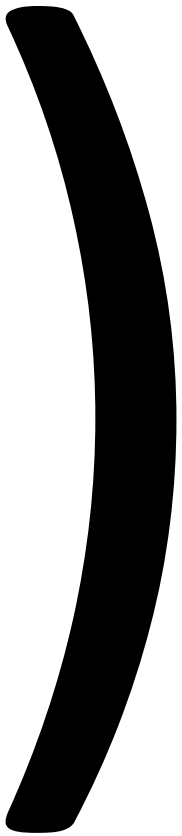
```
# inner prod
einsum('i,i->', A, B) # (10,) x (10,) -> scalar

# outer prod
einsum('i,j->ij', A, B) # (10,) x (5,) -> (10, 5)

# element-wise prod
einsum('i,i->i', A, B) # (10,) x (10,) -> (10,)
```

Tim Rocktäschel, “Einsum is all you need”, https://rockt.github.io/2018/04/30/einsum

Examples

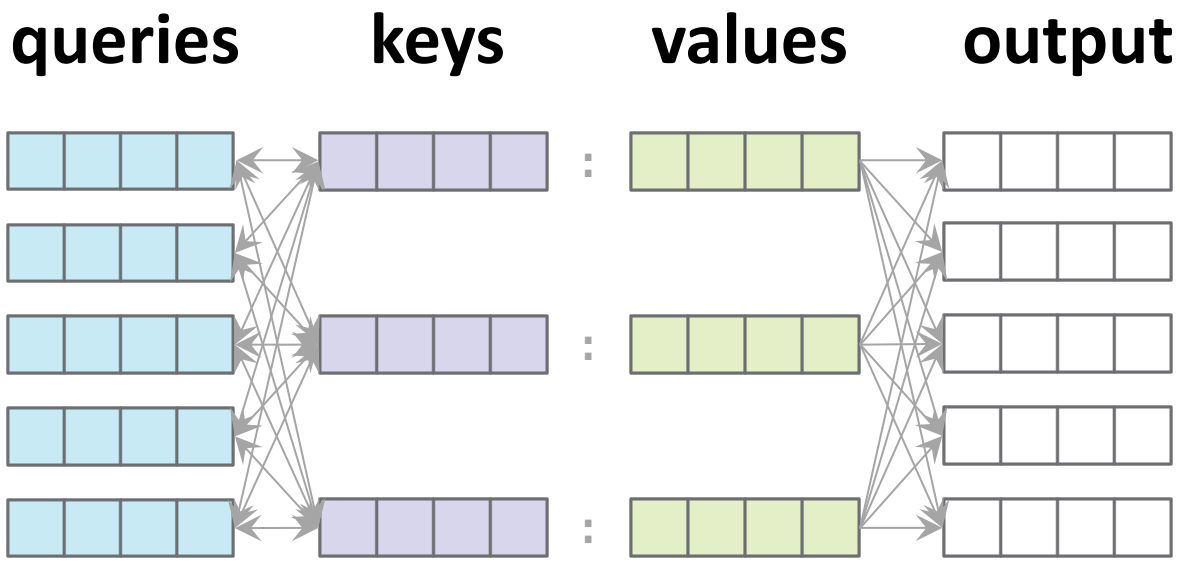
```
# matrix-vector prod
einsum('ik,k->i', A, B) # (10, 4) x (4,) -> (10,)

# matrix-matrix prod
einsum('ik,kj->ij', A, B) # (10, 4) x (4, 5) -> (10, 5)

# batch matrix-matrix prod
einsum('nik,nkj->nij', A, B) # (n, 10, 4) x (n, 4, 5) -> (n, 10, 5)
```

Tim Rocktäschel, “Einsum is all you need”, https://rockt.github.io/2018/04/30/einsum



Attention

w/ Einsum notation

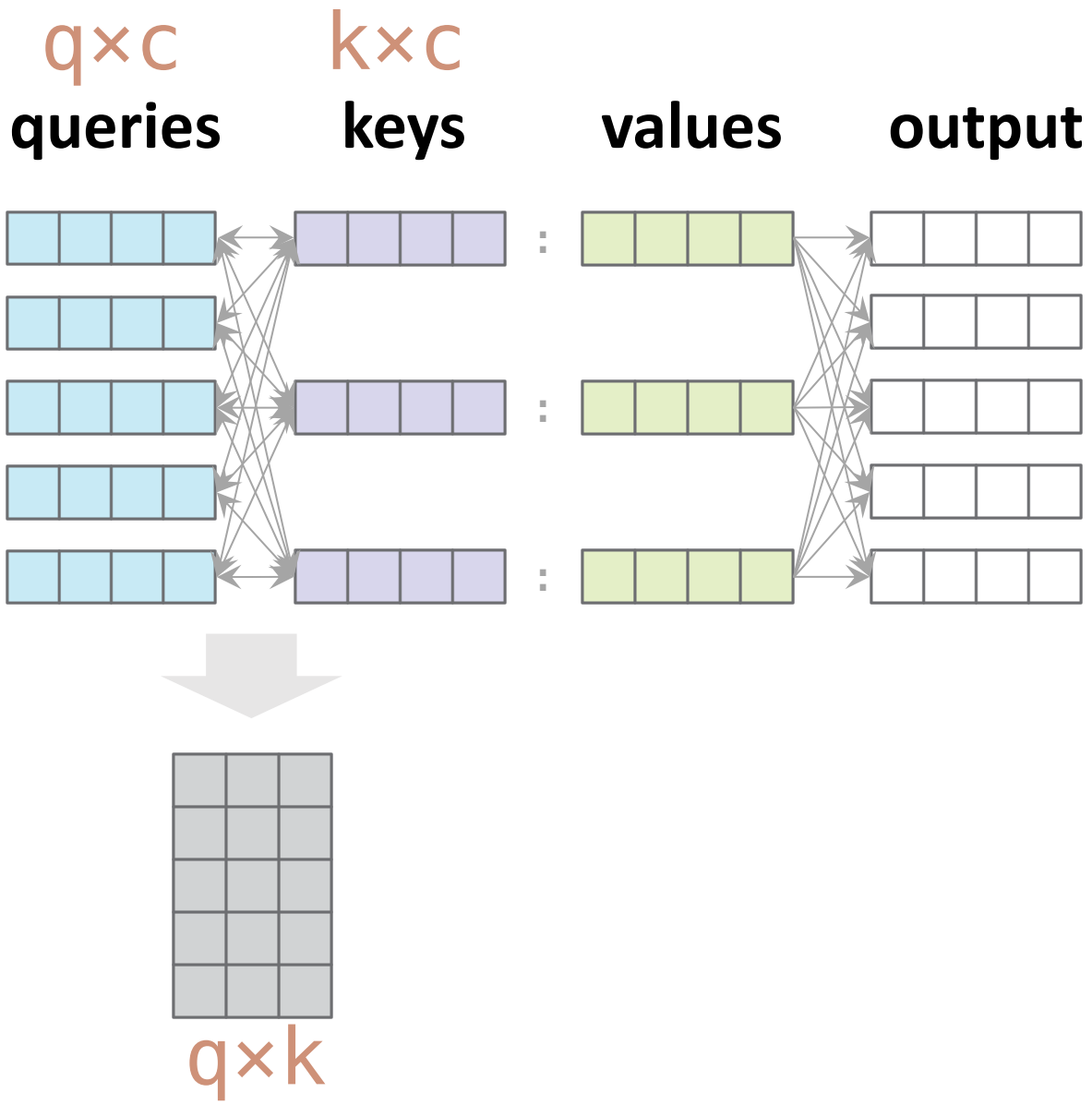


Attention

w/ Einsum notation

1. query-key similarity:

```
S = einsum('qc,kc->qk', Q, K)
```



Attention

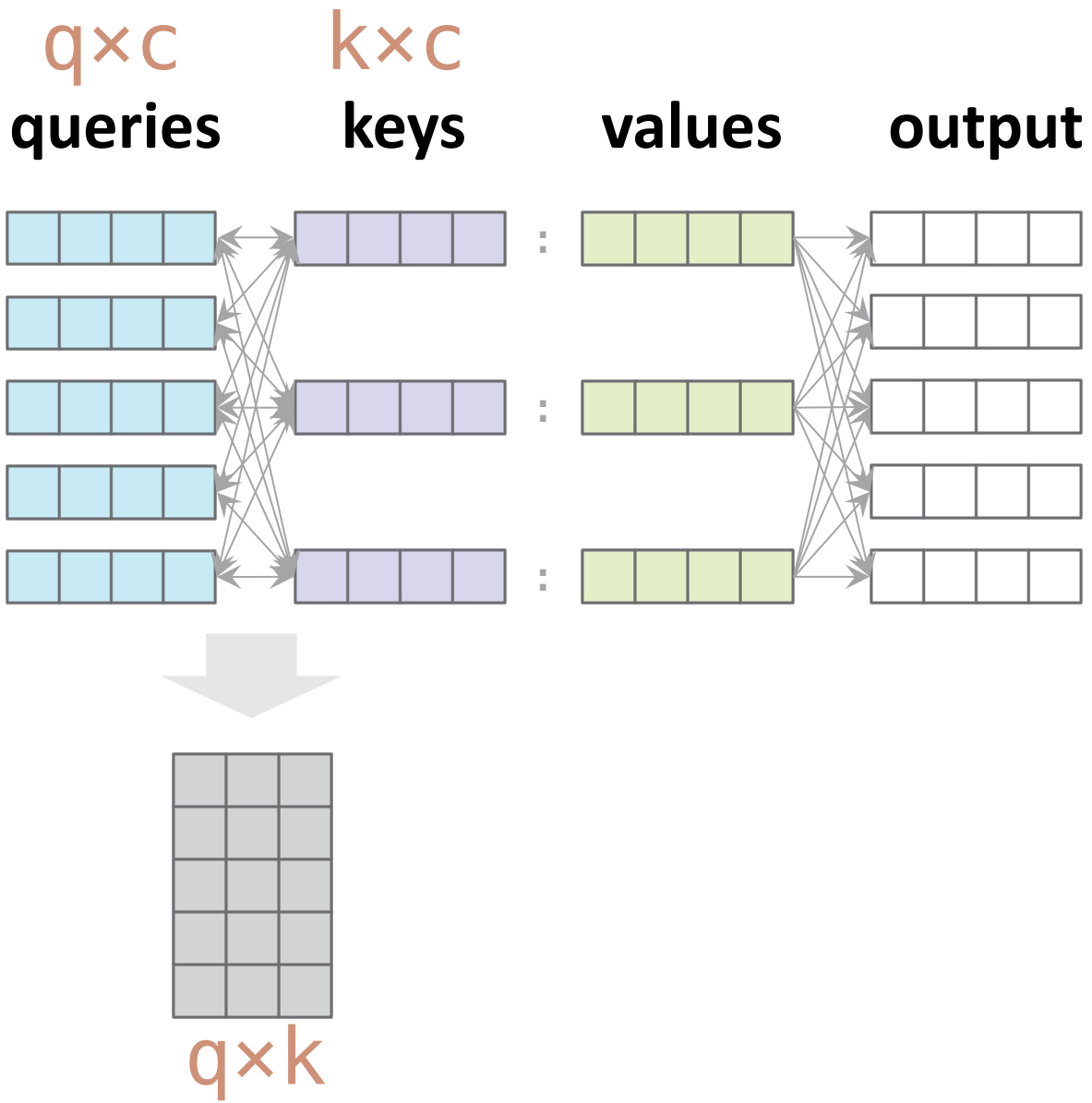
w/ Einsum notation

1. query-key similarity:

```
S = einsum('qc,kc->qk', Q, K)
```

2. attention weight (softmax):

```
W = softmax(S, dim=-1)
```



*In practice, the logits s_i are scaled by 1/sqrt(dim) before softmax.

Attention

w/ Einsum notation

1. query-key similarity:

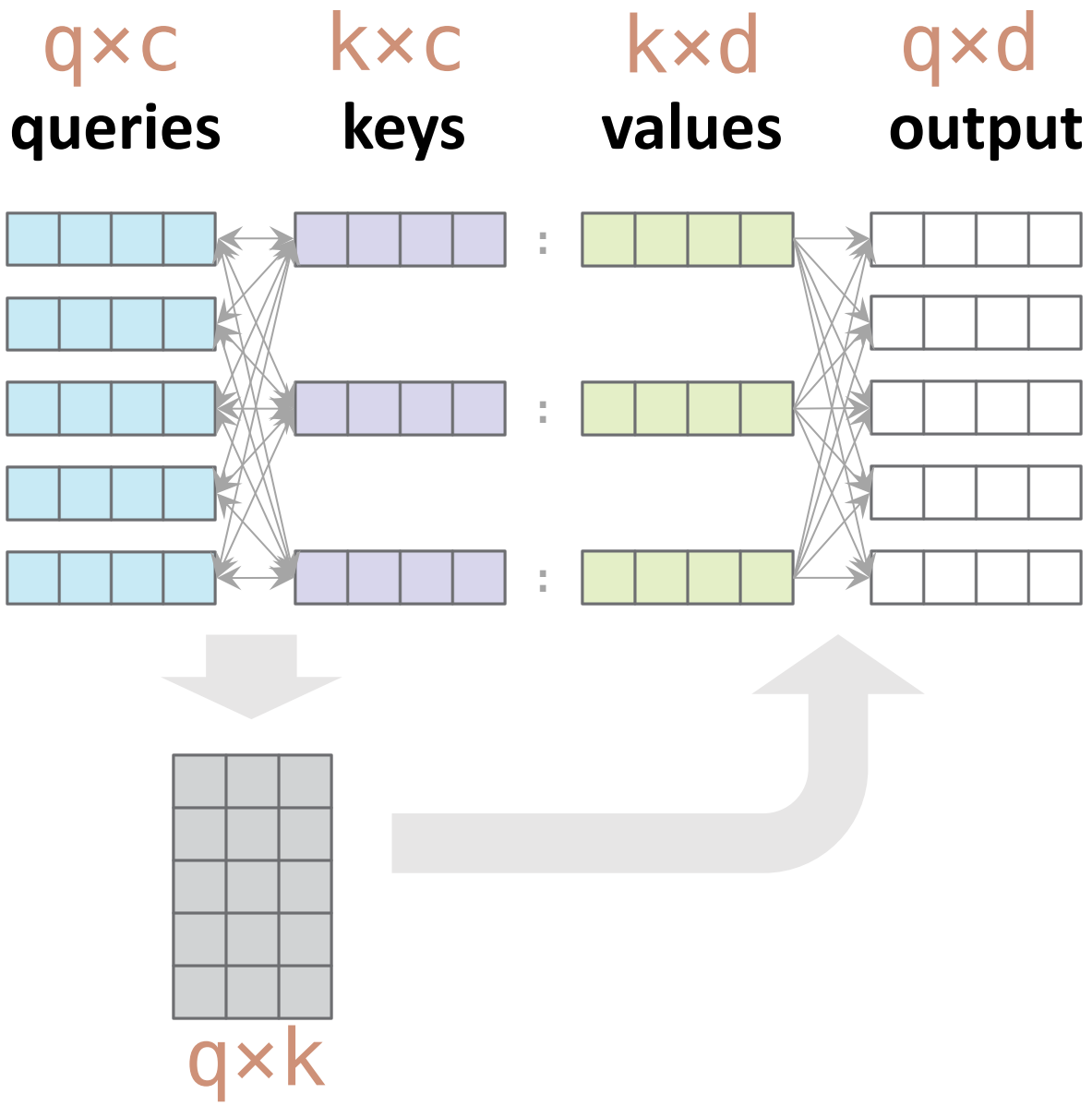
```
S = einsum('qc,kc->qk', Q, K)
```

2. attention weight (softmax):

```
W = softmax(S, dim=-1)
```

3. weighted average:

```
Z = einsum('qk,kd->qd', W, V)
```



Attention

Multi-Head: repeat multiple times

1. query-key similarity:

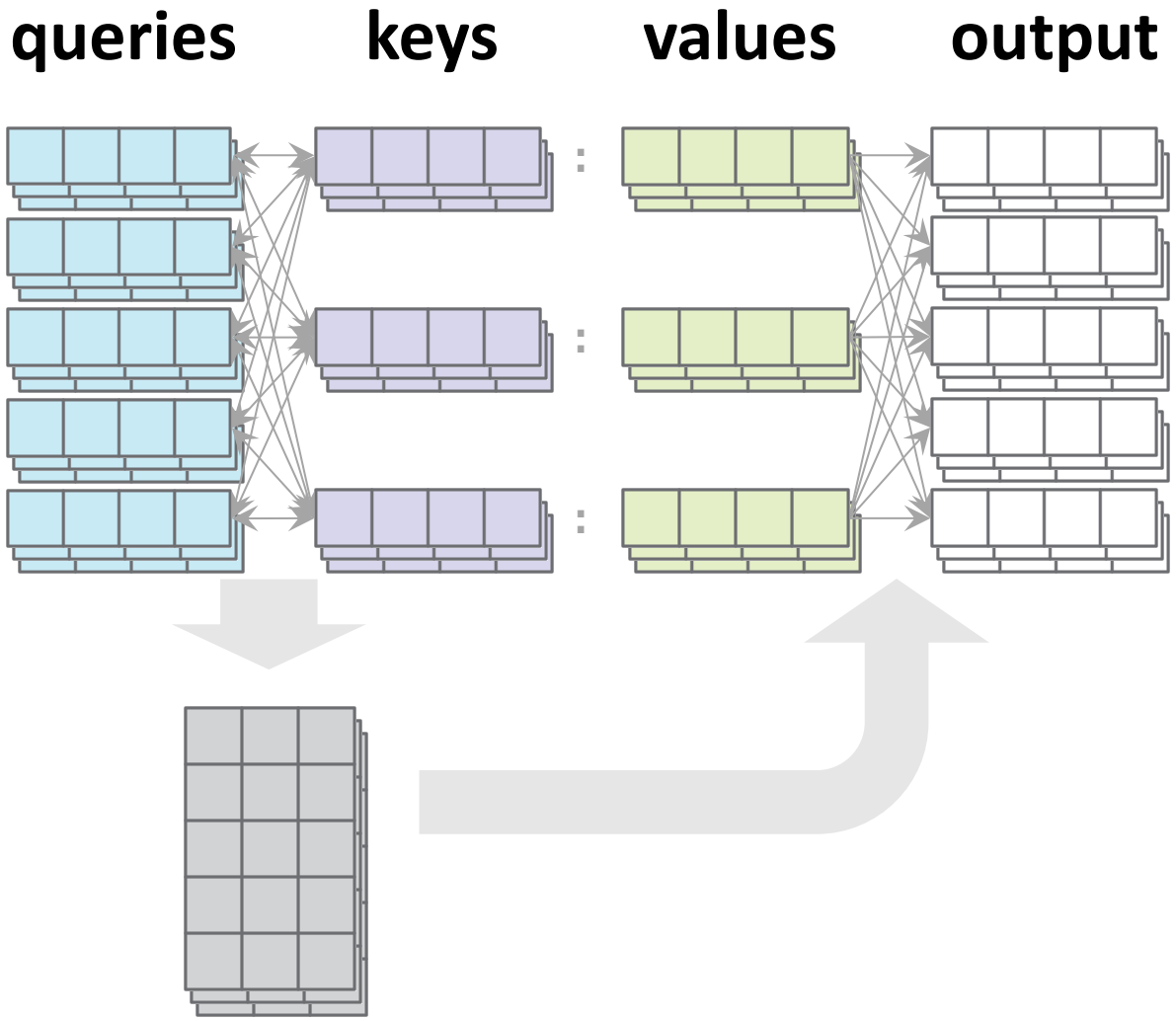
```
S = einsum('hqc,hkc->hqk', Q, K)
```

2. attention weight (softmax):

```
W = softmax(S, dim=-1)
```

3. weighted average:

```
Z = einsum('hqk,hkd->hqd', W, V)
```



Attention

Multi-Head: repeat multiple times

h: # heads

1. query-key similarity:

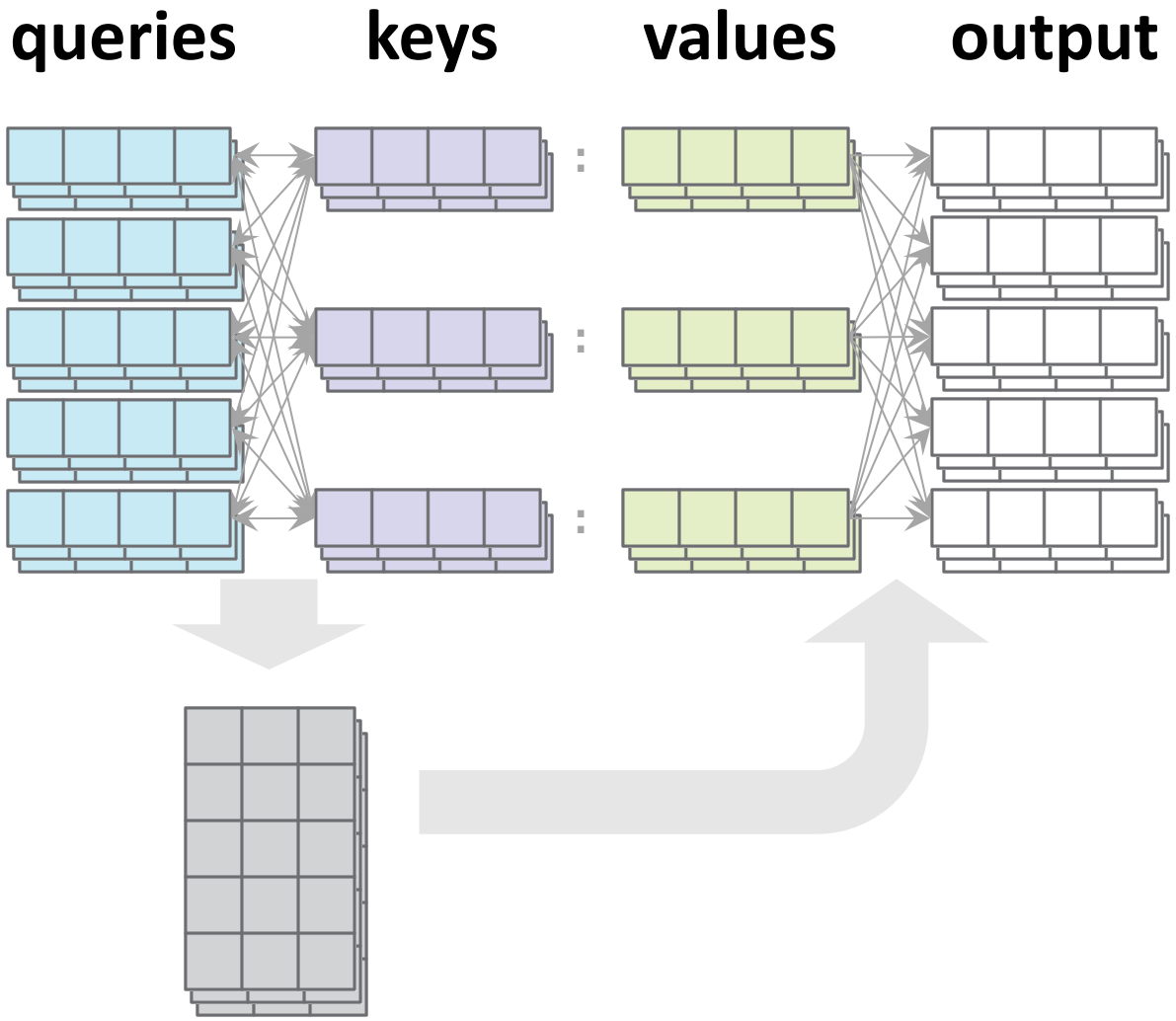
```
S = einsum('hqc,hkc->hqk', Q, K)
```

2. attention weight (softmax):

```
W = softmax(S, dim=-1)
```

3. weighted average:

```
Z = einsum('hqk,hkd->hqd', W, V)
```



Attention

Multi-Head: repeat multiple times (w/ batching)

n: batch size

1. query-key similarity:

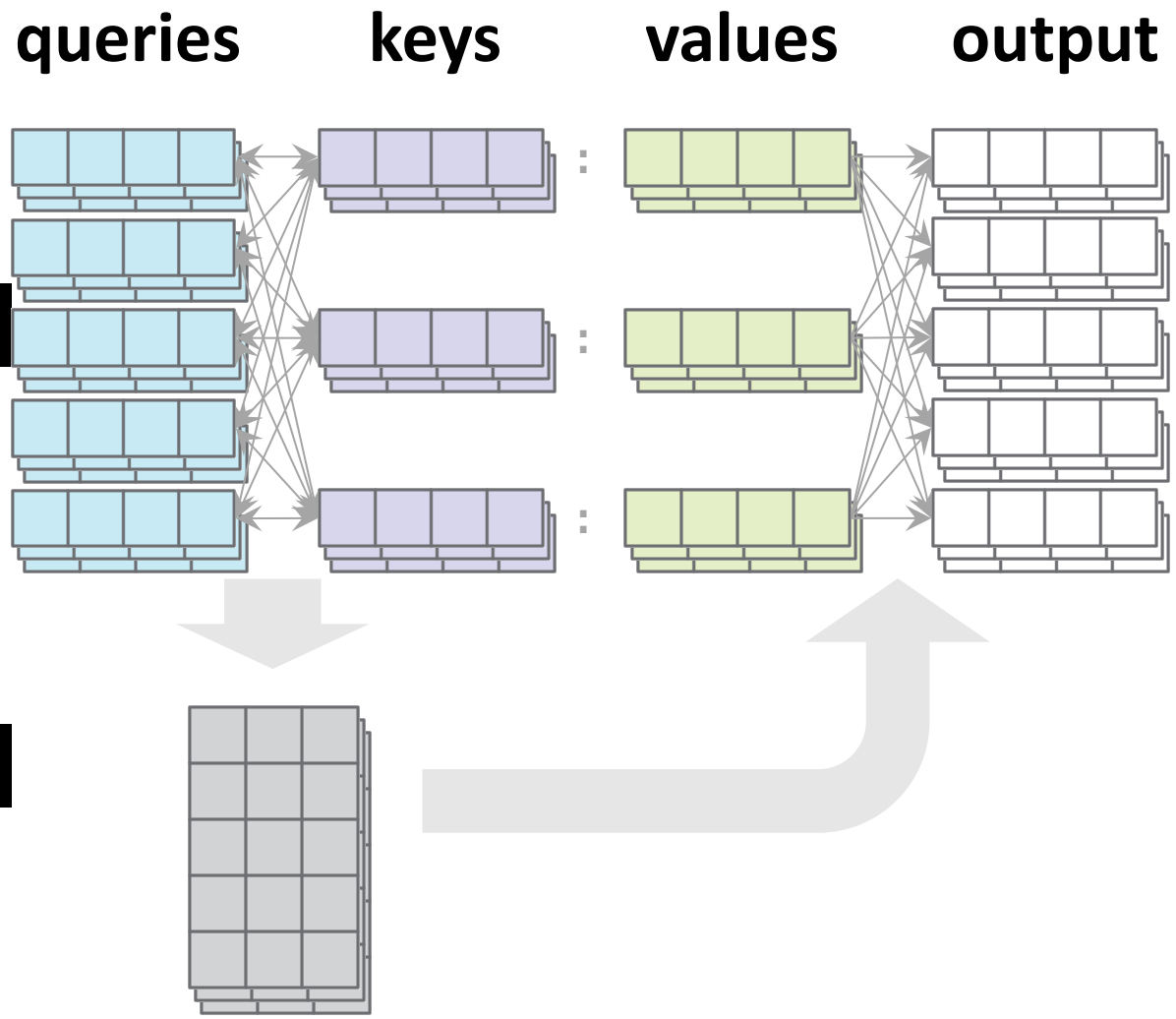
```
S = einsum('nhqc,nhkc->nhqk', Q, K)
```

2. attention weight (softmax):

```
W = softmax(S, dim=-1)
```

3. weighted average:

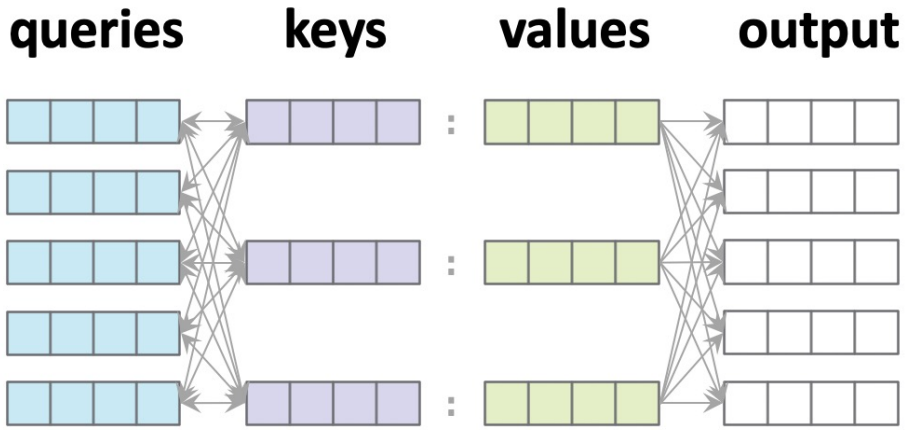
```
Z = einsum('nhqk,nhkd->nhqd', W, V)
```



Attention: Notes

In its general form …

- **Attention says nothing about how queries, keys, values are provided.**
  - # queries may not equal # keys
  - # queries = # outputs
  - # key channels may not equal # value channels
  - # key channels = # query channels
  - # value channels = # output channels

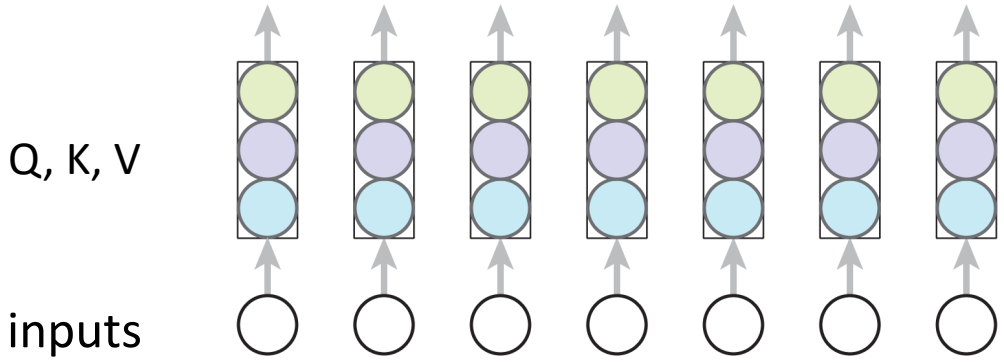


- **Attention is parameter-free!**

- **Every output is connected to every value and every key.**

Self-attention

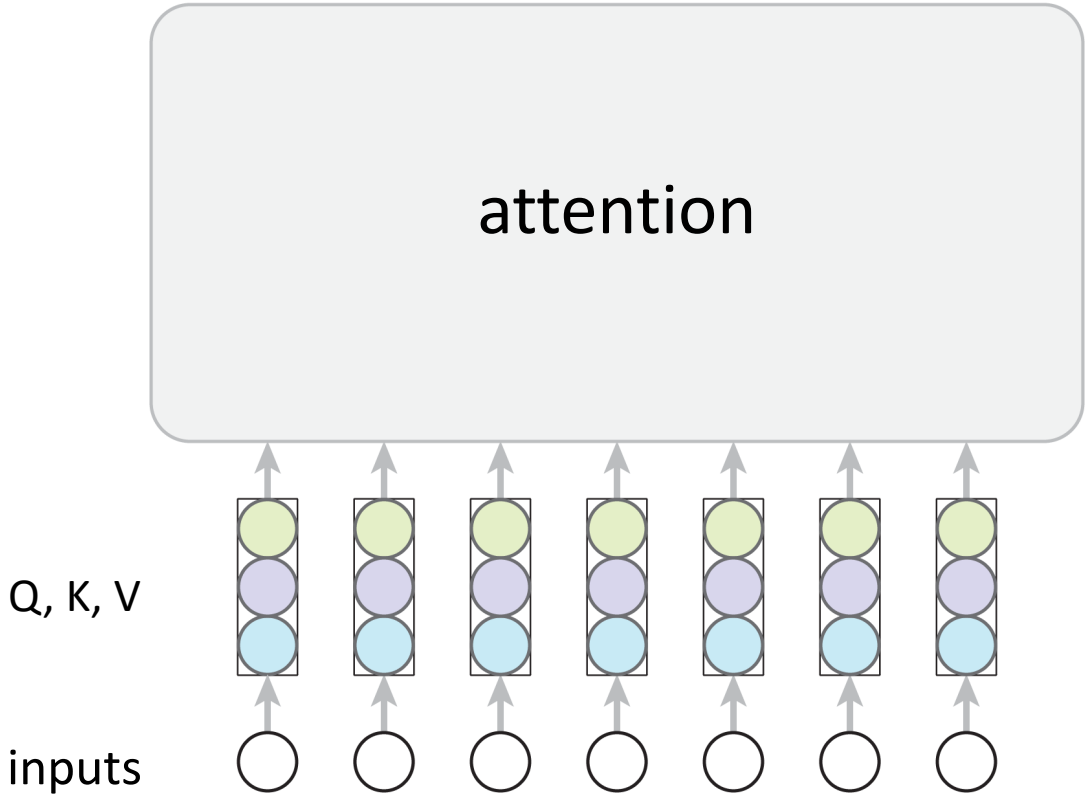
- Q, K, V sequences: produced by the input sequence **itself**.



projection
(conv w/ kernel = 1)

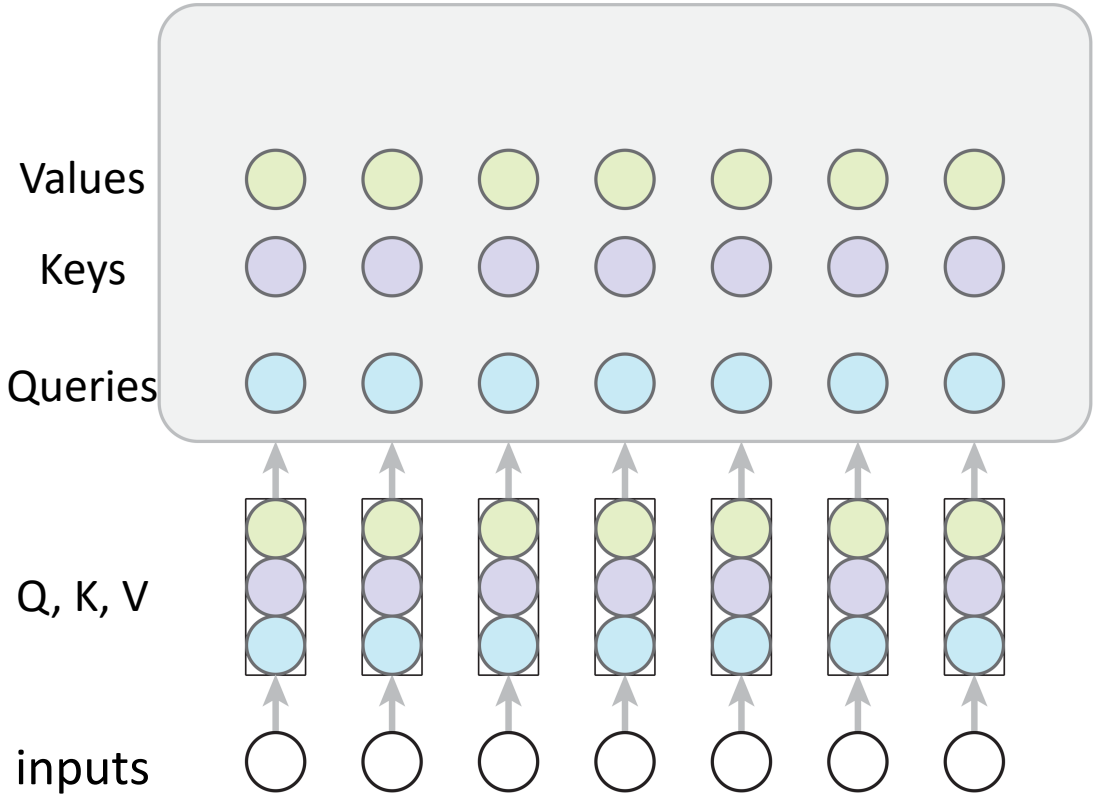
Self-attention

- Q, K, V sequences: produced by the input sequence **itself**.



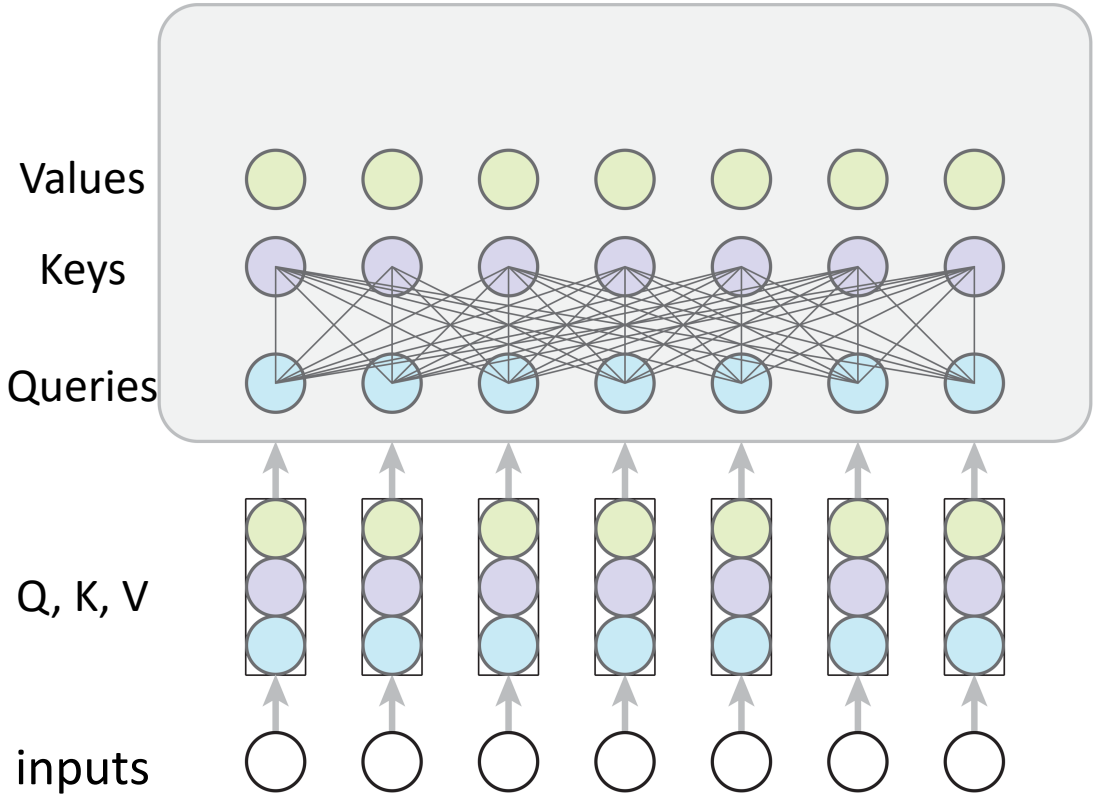
Self-attention

- Q, K, V sequences: produced by the input sequence **itself**.



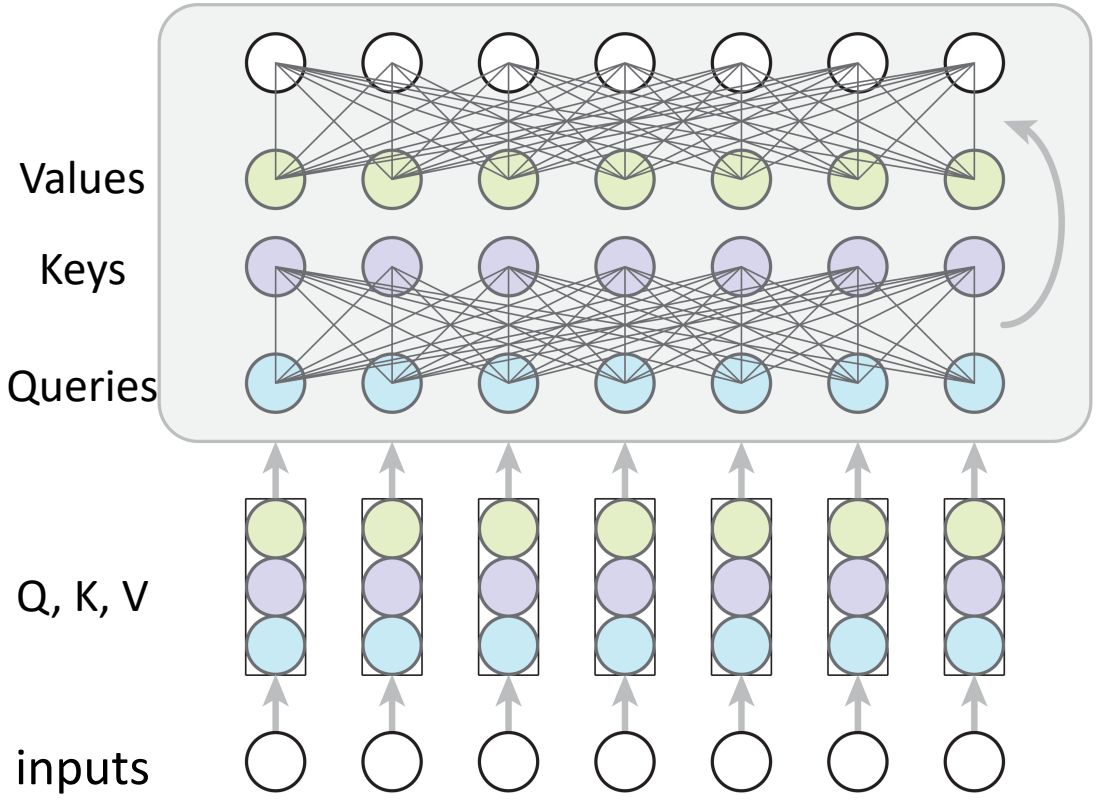
Self-attention

- Q, K, V sequences: produced by the input sequence **itself**.



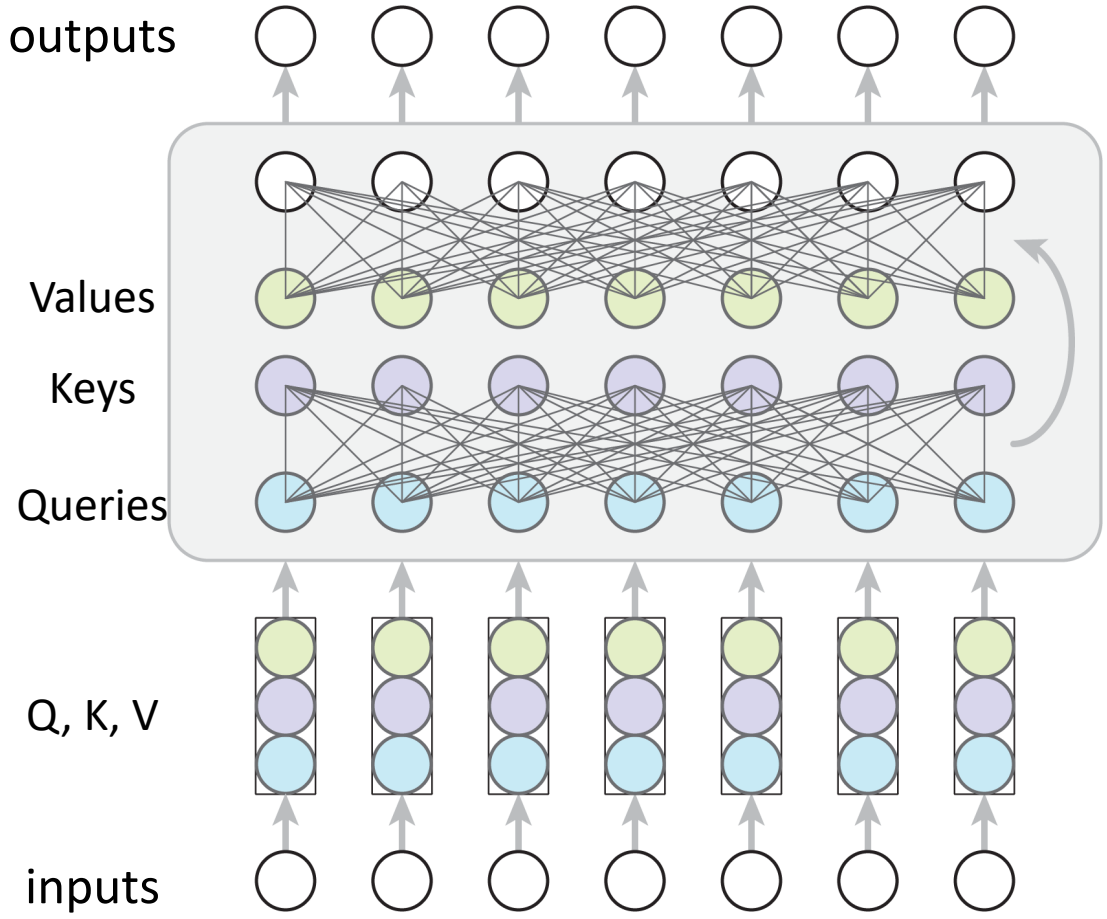
Self-attention

- Q, K, V sequences: produced by the input sequence **itself**.



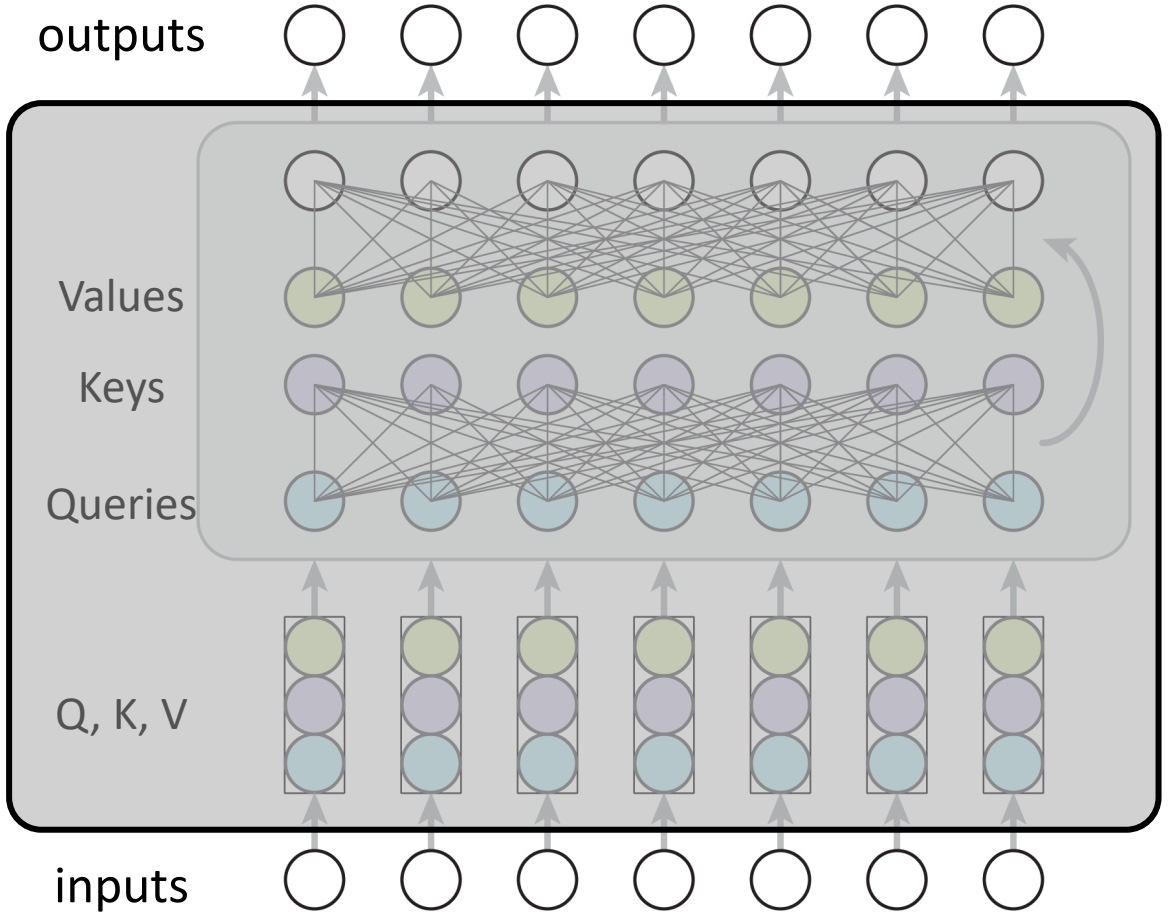
Self-attention

- Q, K, V sequences: produced by the input sequence **itself**.

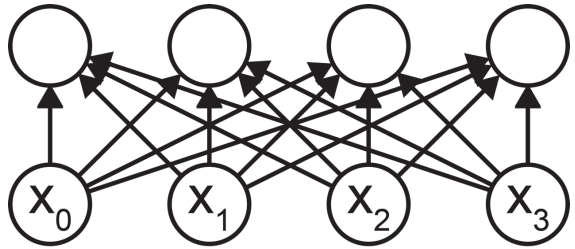


Self-attention

- Q, K, V sequences: produced by the input sequence **itself**.



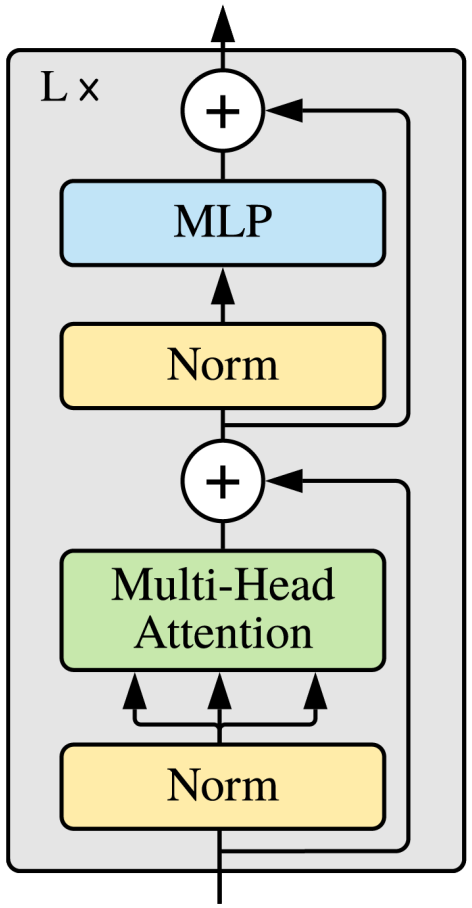
Every output is connected to every input!



(Each arrow indicates computational dependence)

Transformer

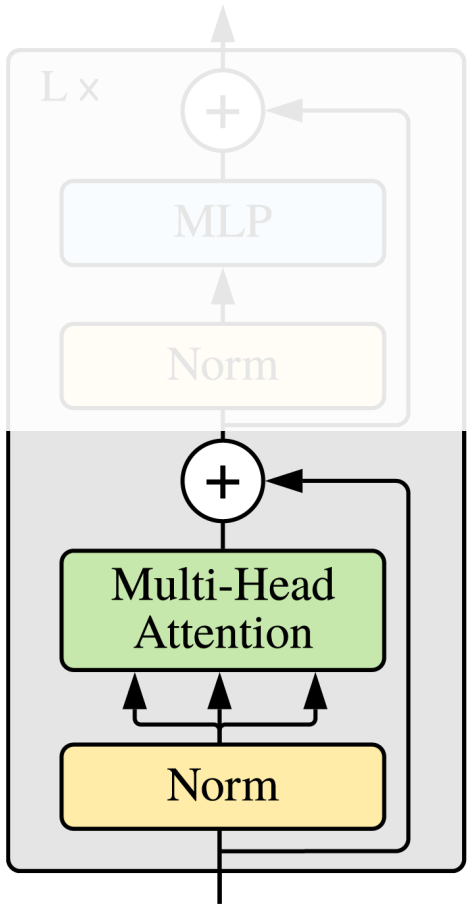
- Transformer: a deep neural net based on the attention mechanism.



A Transformer block
(pre-norm variant)

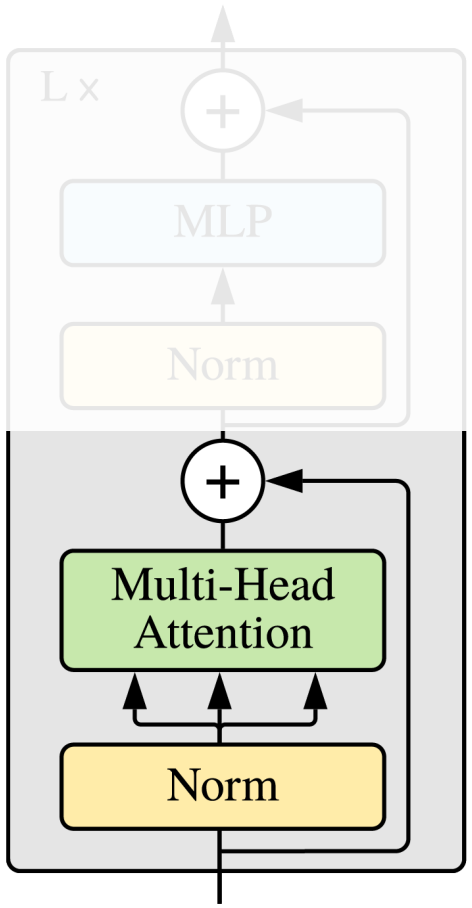
Vaswani, et al., “Attention is all you need”, 2017

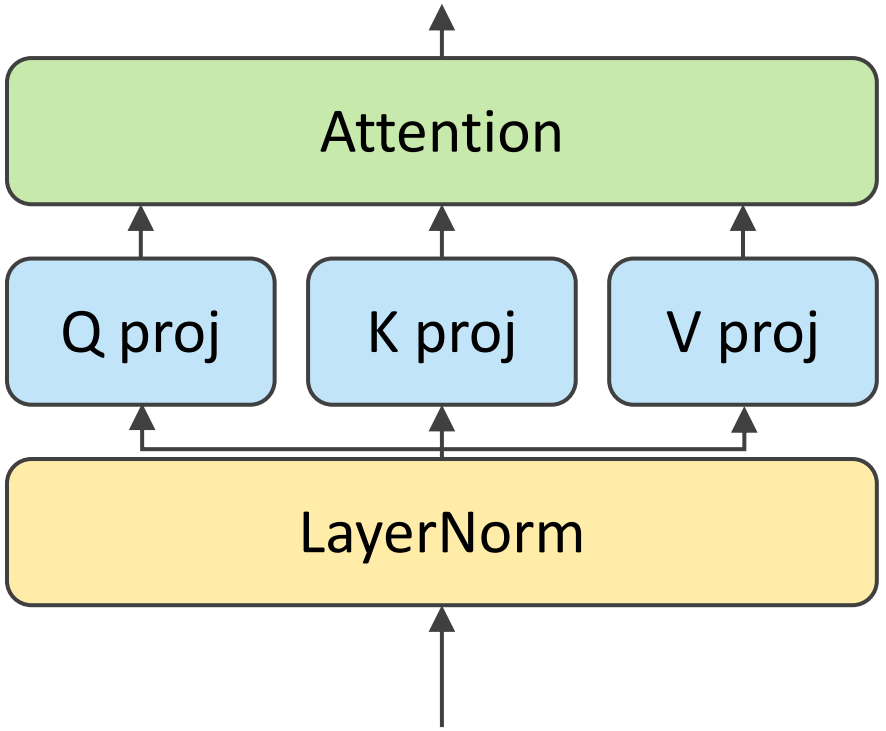
Transformer: Multi-Head Attention (MHA) Block



Vaswani, et al., “Attention is all you need”, 2017

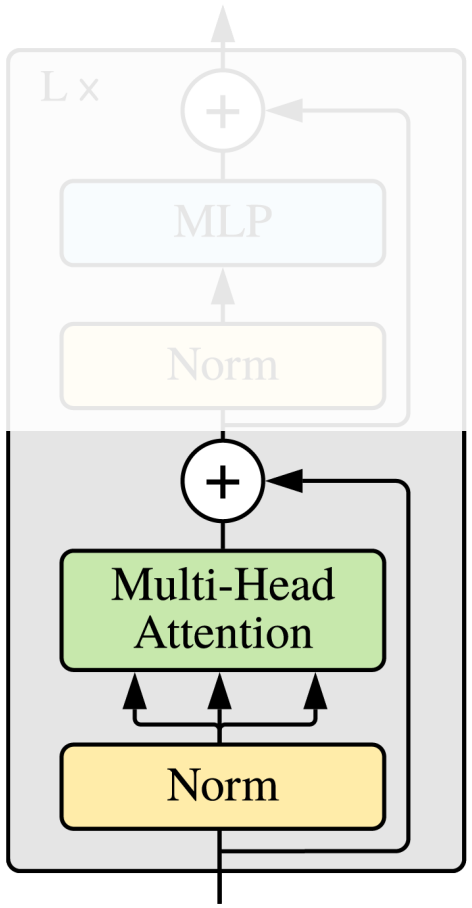
Transformer: Multi-Head Attention (MHA) Block

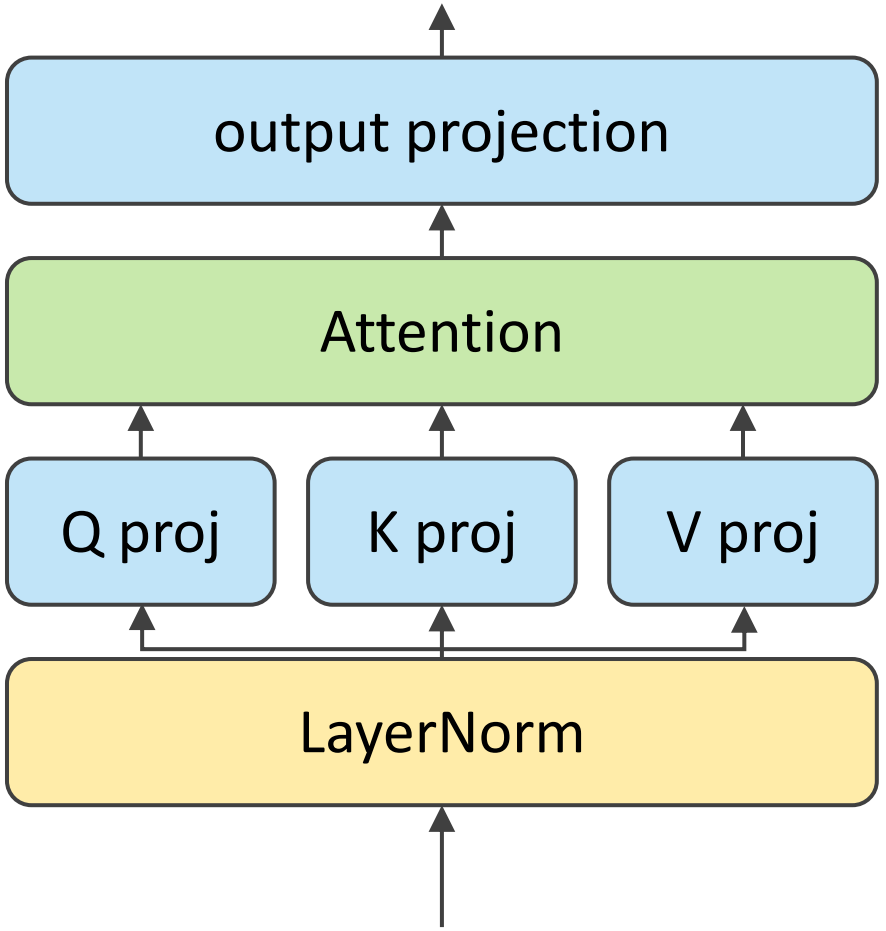




Vaswani, et al., “Attention is all you need”, 2017

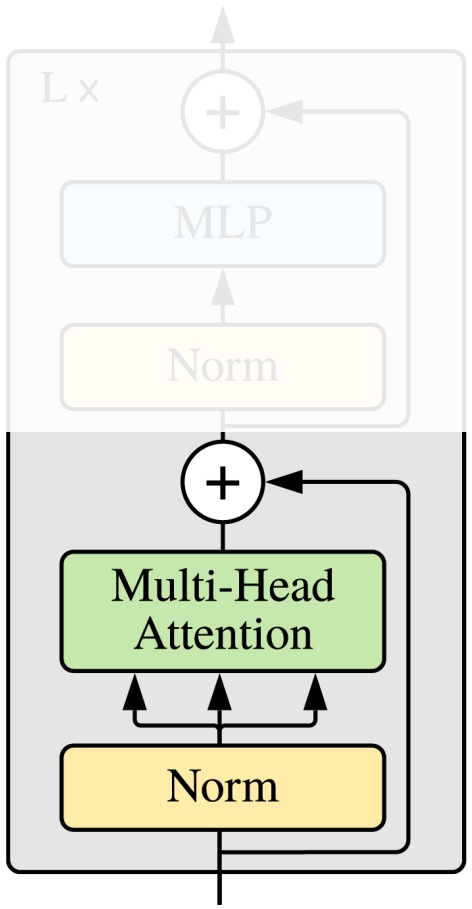
Transformer: Multi-Head Attention (MHA) Block

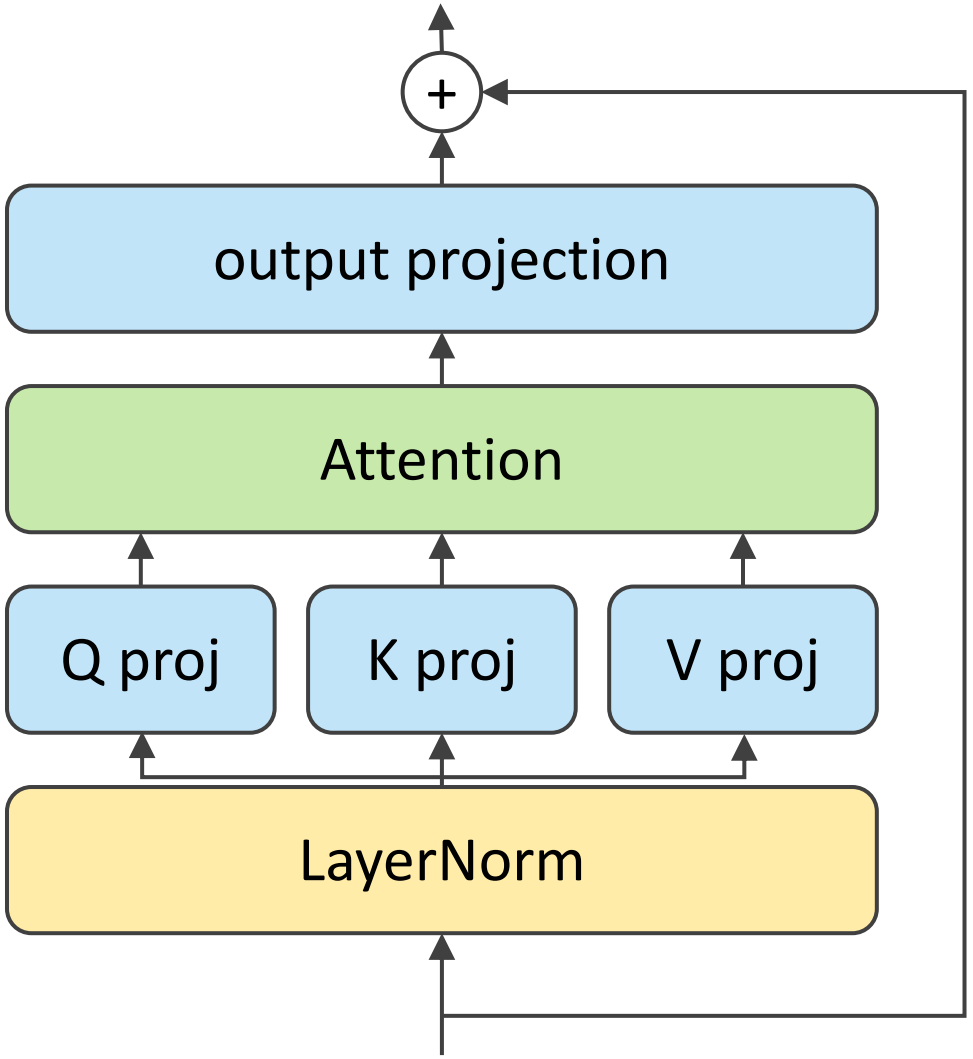




Vaswani, et al., “Attention is all you need”, 2017

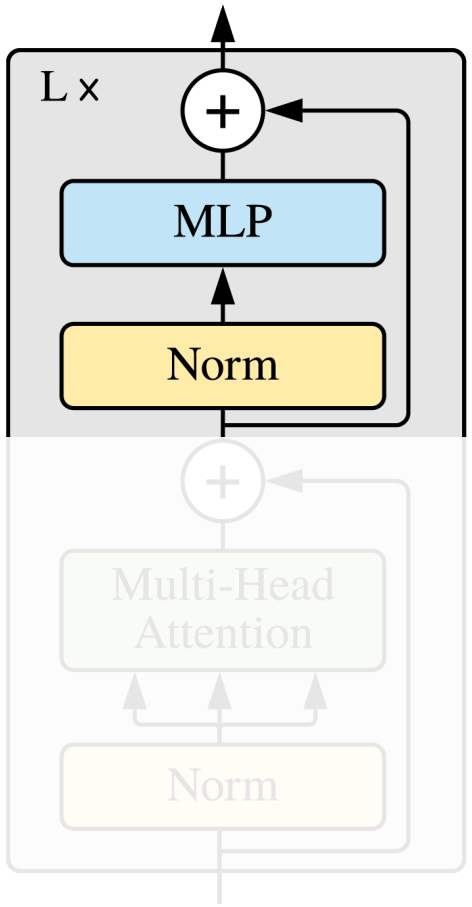
Transformer: Multi-Head Attention (MHA) Block





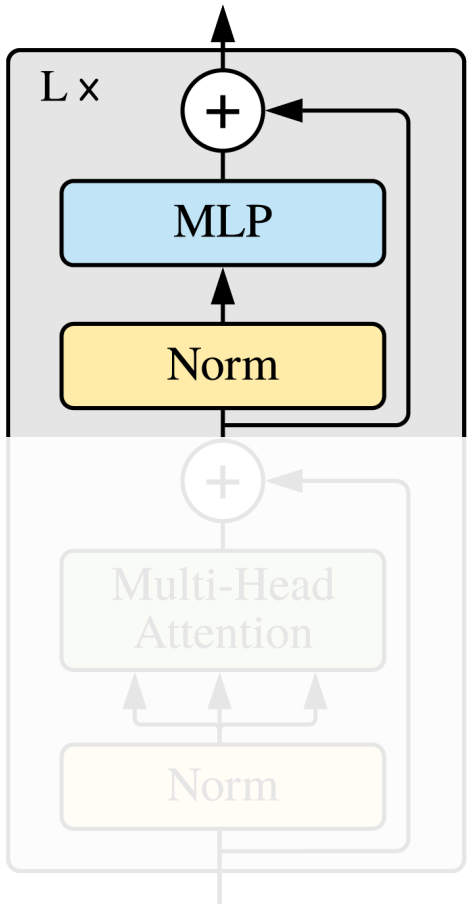
Vaswani, et al., “Attention is all you need”, 2017

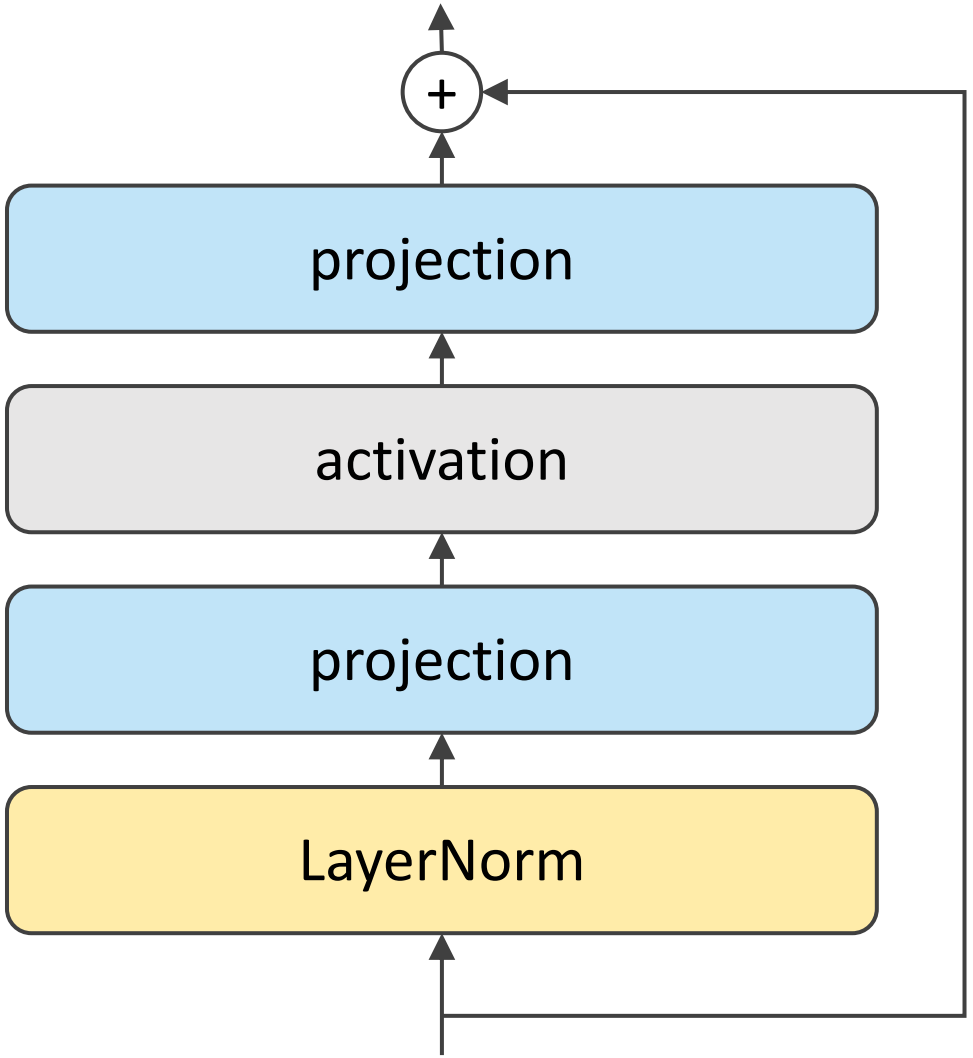
Transformer: Multi-Layer Perception (MLP) Block



Vaswani, et al., “Attention is all you need”, 2017

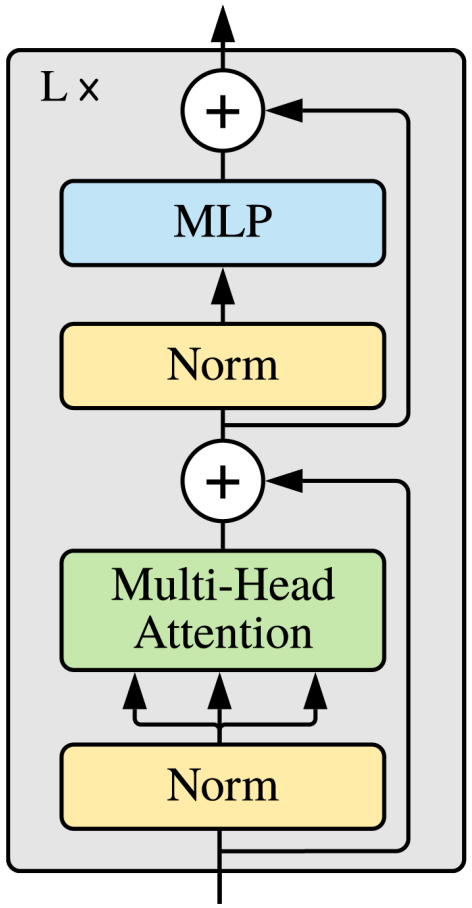
Transformer: Multi-Layer Perception (MLP) Block





Vaswani, et al., “Attention is all you need”, 2017

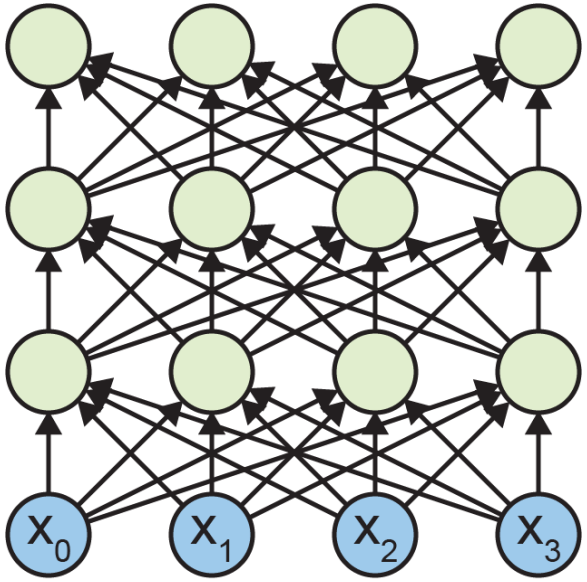
Transformer



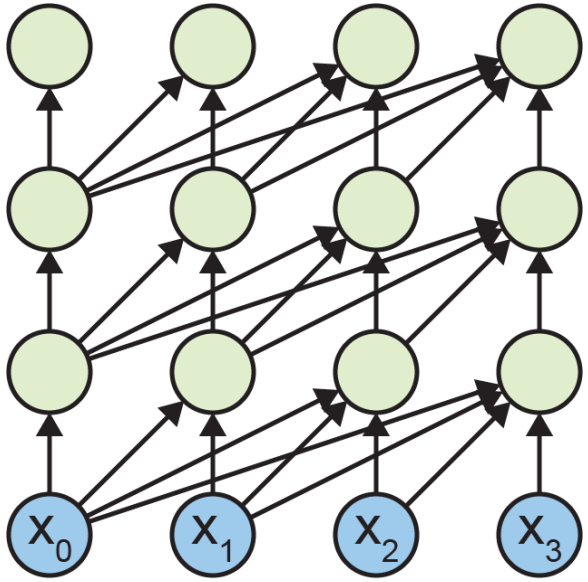
- MHA block
  - QKV **projection**
  - attention
  - output **projection**
- MLP block
  - two **projections**
- All parameters are in **projections**
  - conceptually, all conv with kernel size 1

Vaswani, et al., “Attention is all you need”, 2017

Transformer: Ordered Sequences



bidirectional
(Transformer “Encoder”)



causal
(Transformer “Decoder”)

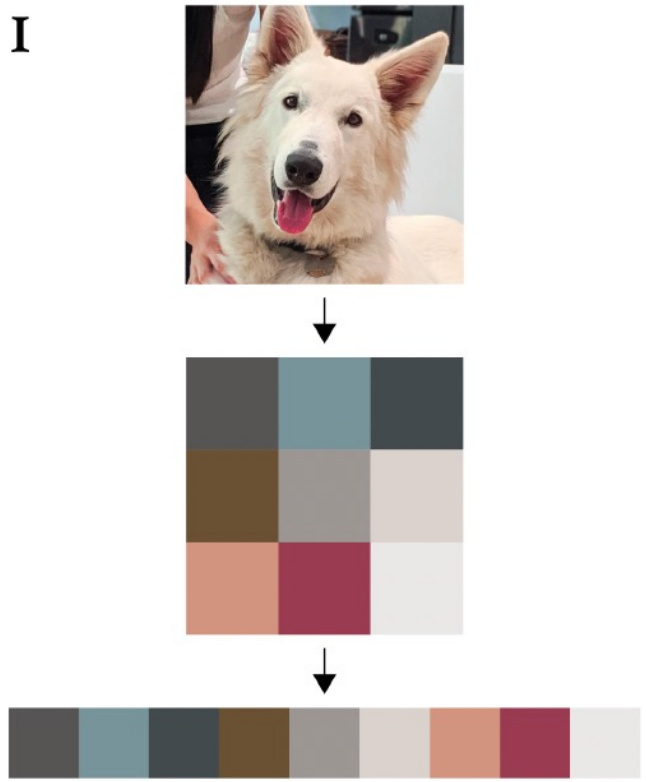
Notes:

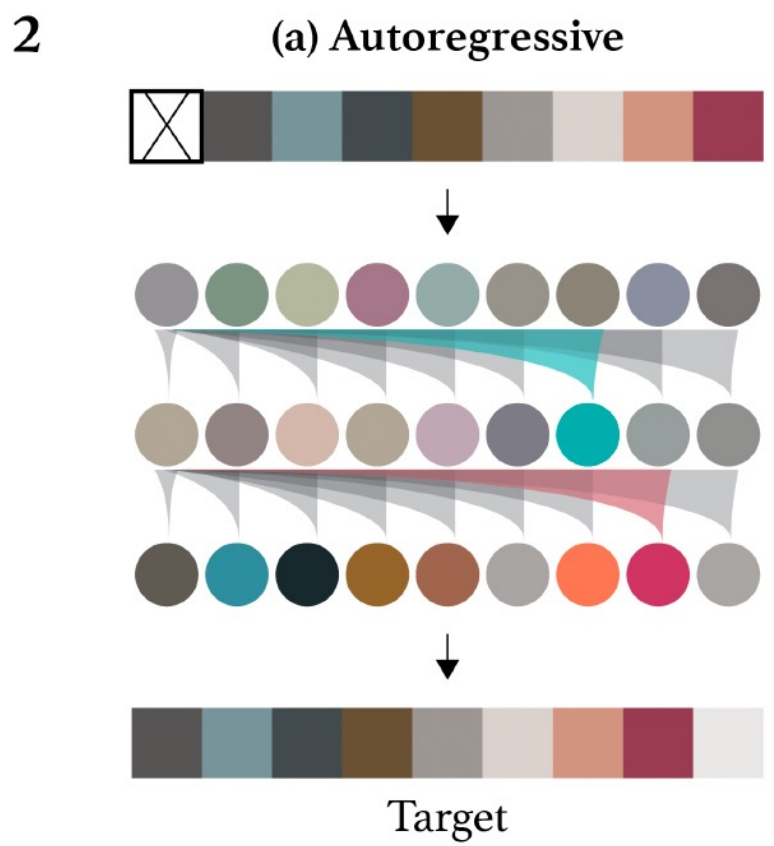
- Every “layer” denotes a Transformer block
- Every arrow indicates computational dependence
- Causality is done by masking (before softmax)

Example: image GPT (iGPT)

iGPT: Transformer on sequences of pixels

- Autoregressive models (GPT) on pixels for image generation



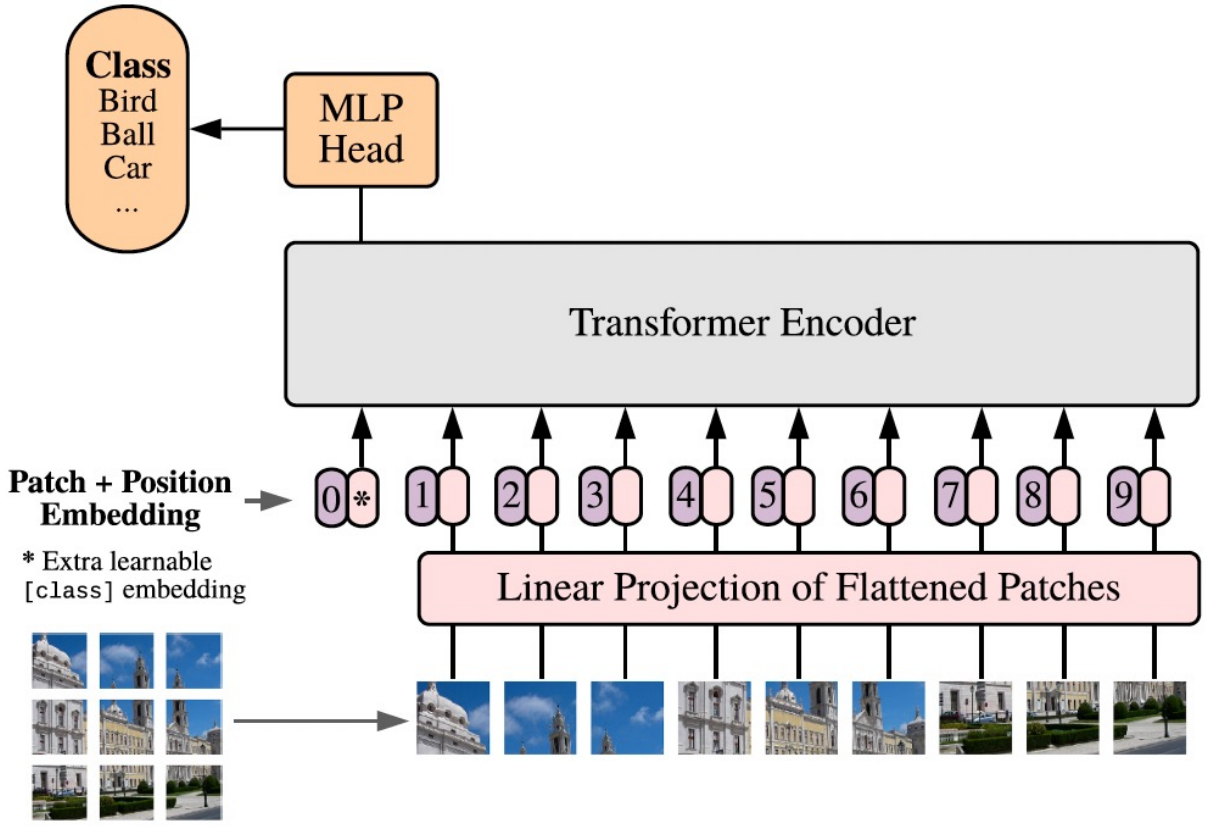


Chen, et al., “Generative Pretraining from Pixels”, ICML 2020

Example: Vision Transformer (ViT)

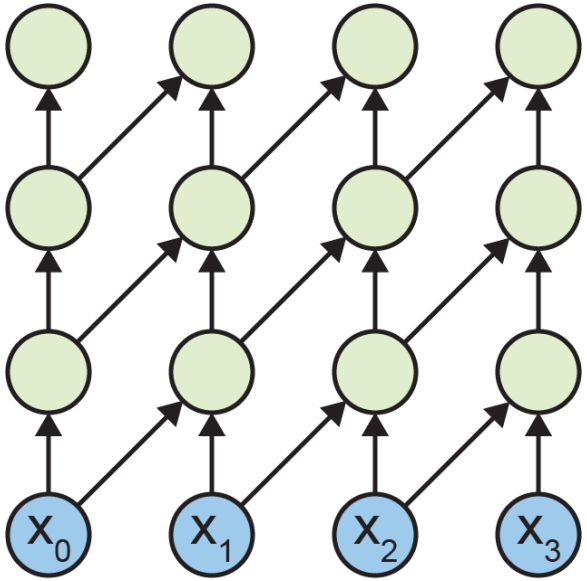
ViT: Transformer on sequences of patches

- Image: sequence of patches
- Patch embedding
  - 16×16 **conv** w/ stride 16
- MLP and QKV projection
  - 1×1 **conv**
- “Transforming” computer vision!



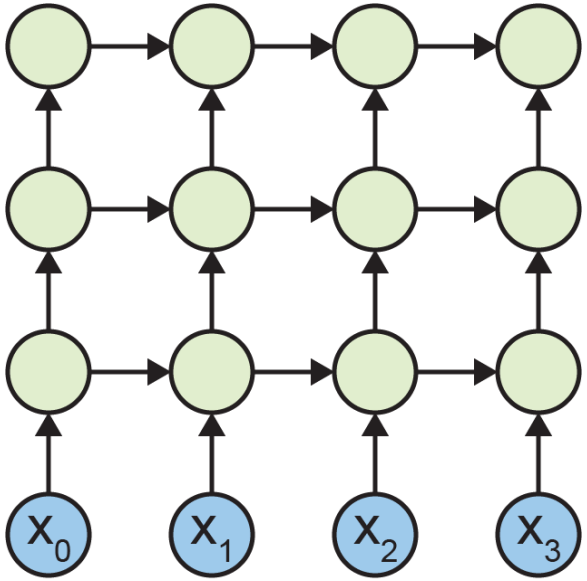
Dosovitskiy, et al., An Image is Worth 16x16 Words- Transformers for Image Recognition at Scale, arXiv 2020, ICLR 2021

Sequence Modeling



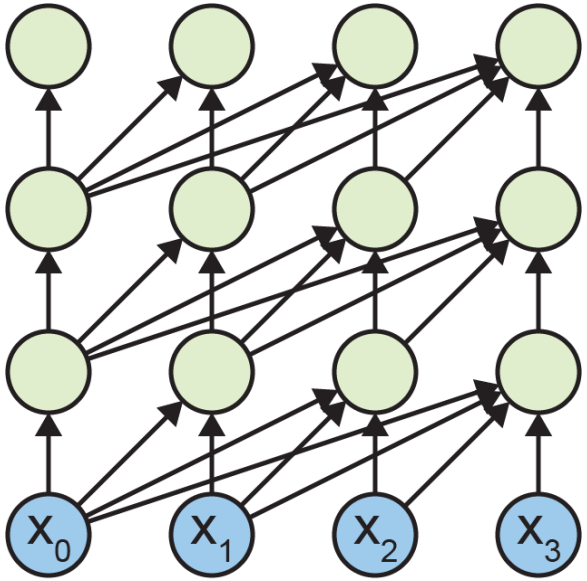
CNN

- limited context
- feedforward



RNN

- long context
- not feedforward



Attention

- long context
- feedforward

Lecture 11: Sequence Modeling

- Sequence Problems in Vision
- ConvNets for Sequence Problems
- Recurrent Neural Networks
- Attention and Transformers