



FACULTY OF MATHEMATICS,
PHYSICS AND INFORMATICS
Comenius University
Bratislava

Neural Networks for Computer Vision

Lecture 9: RNNs and Transformers

Ing. Viktor Kocur, PhD., RNDr. Zuzana Černeková, PhD.

24.11.2021

Contents



■ RNNs

- ▶ General Idea
- ▶ LSTM
- ▶ GRU

■ Attention

■ Transformers

- ▶ Original NLP transformers
- ▶ Visual Transformers
- ▶ Future of CV

Acknowledgment

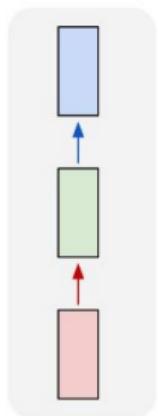


Most of the slides are directly adopted from slides for CS231n¹ course at Stanford University!

¹Fei-Fei Li, Ranjay Krishna, and Danfei Xu. *Stanford CS231n lecture slides*. <http://cs231n.stanford.edu/slides/>

“Vanilla” Neural Network

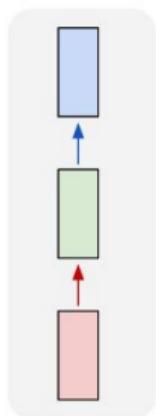
one to one



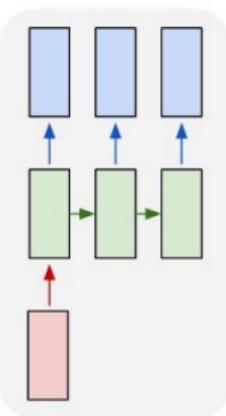
Vanilla Neural Networks

Recurrent Neural Networks: Process Sequences

one to one



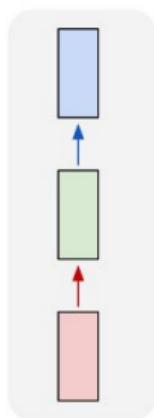
one to many



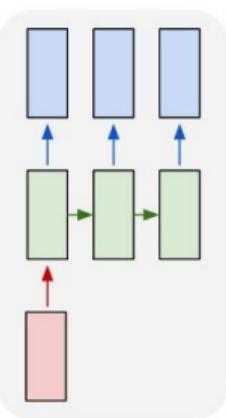
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences

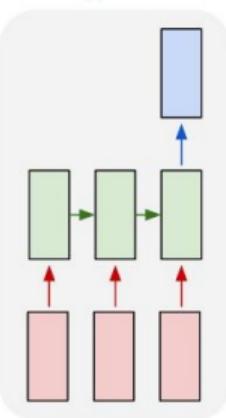
one to one



one to many



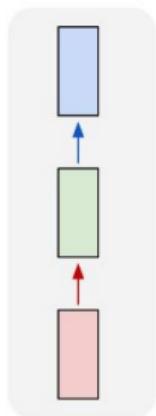
many to one



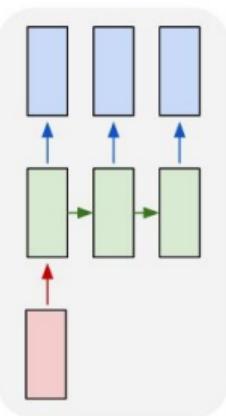
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences

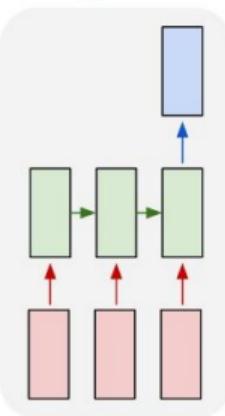
one to one



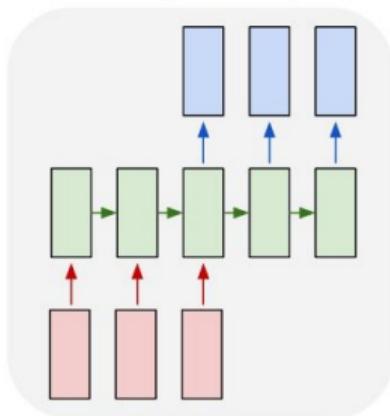
one to many



many to one



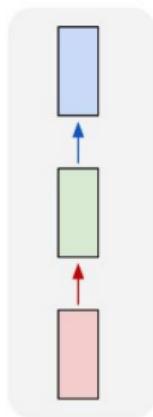
many to many



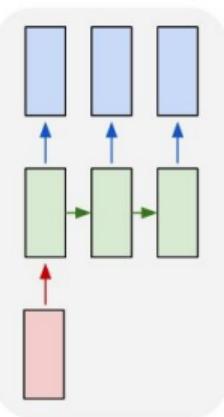
E.g. **Video Captioning**
Sequence of video frames -> caption

Recurrent Neural Networks: Process Sequences

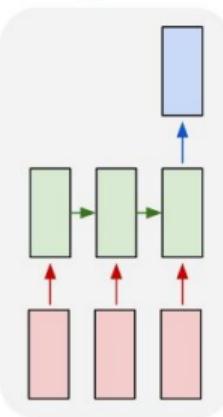
one to one



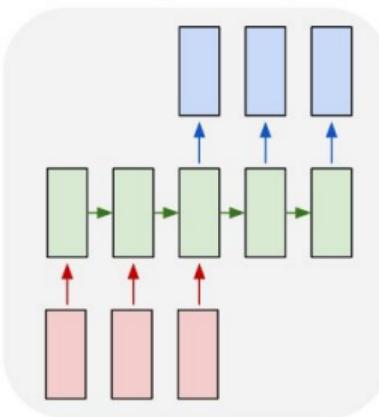
one to many



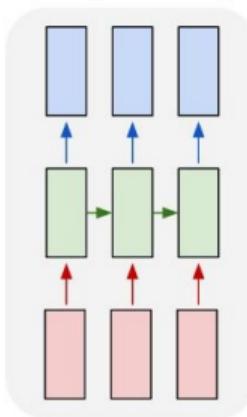
many to one



many to many



many to many



e.g. **Video classification on frame level**



Why do we need RNNs?



Example task: Video Captioning

- Input length is variable

Why do we need RNNs?



Example task: Video Captioning

- Input length is variable
- Output length is also variable

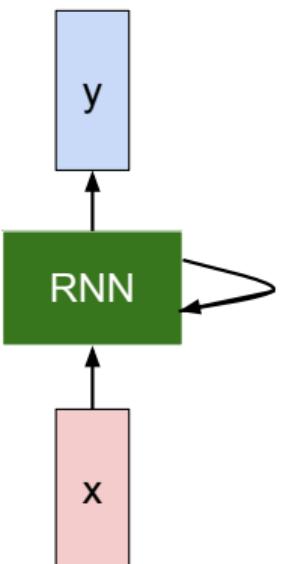
Why do we need RNNs?

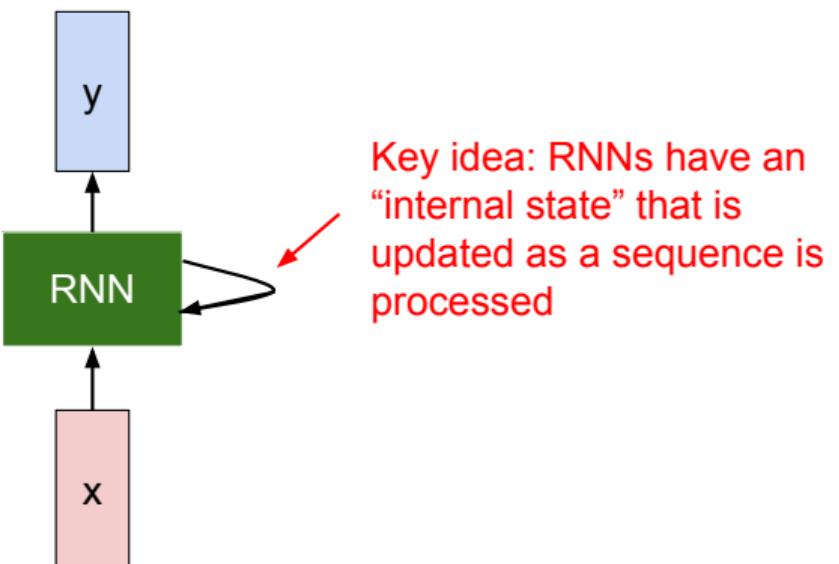


Example task: Video Captioning

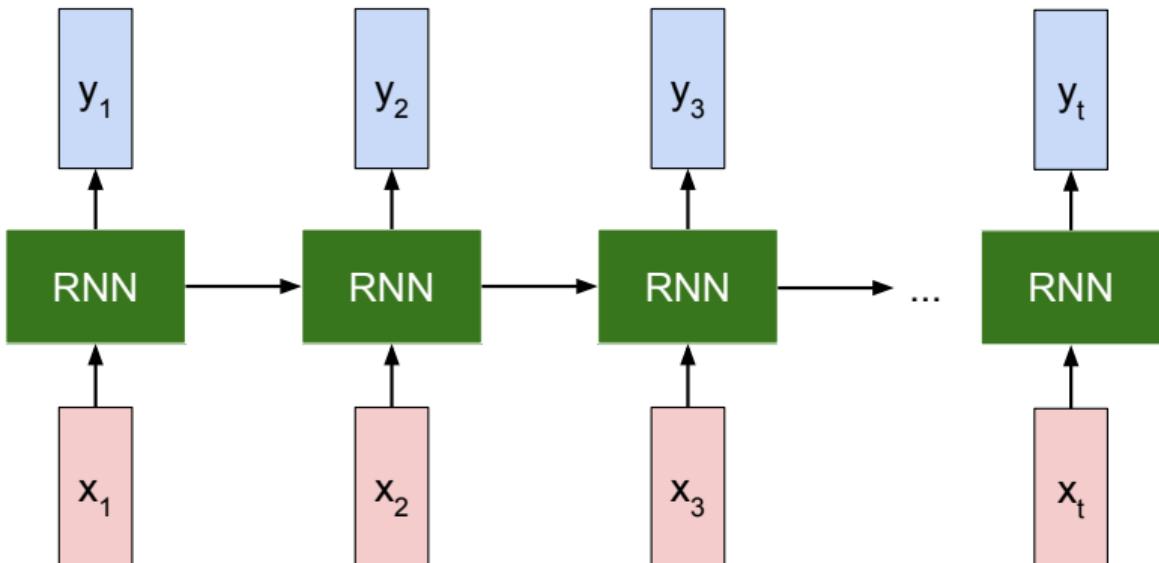
- Input length is variable
- Output length is also variable
- Forcing input and output lengths might introduce issues

RNN





RNN

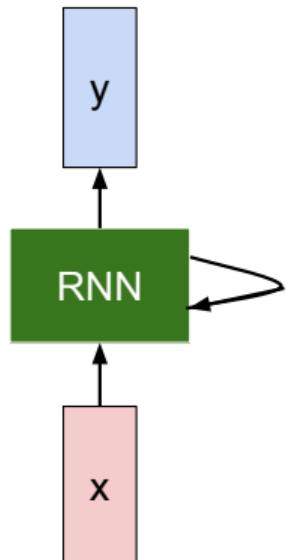




We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

new state / old state input vector at
some function | some time step
with parameters W



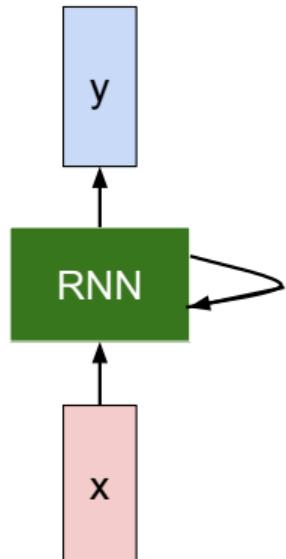


We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

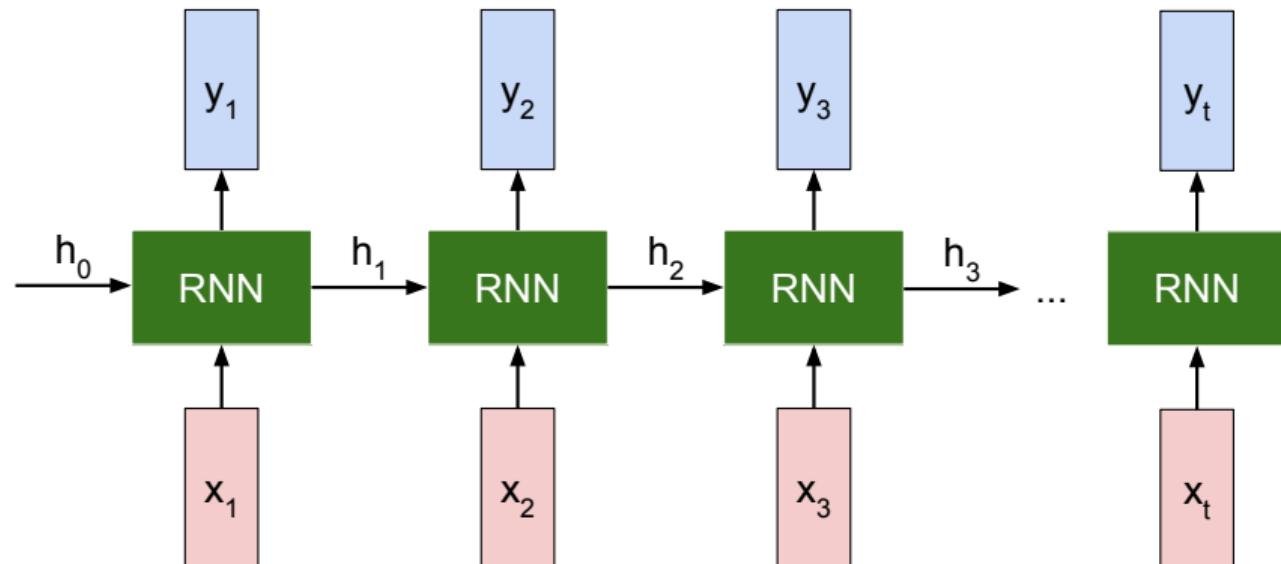
$$y_t = f_{W_{hy}}(h_t)$$

output new state

another function
with parameters W_o



RNN

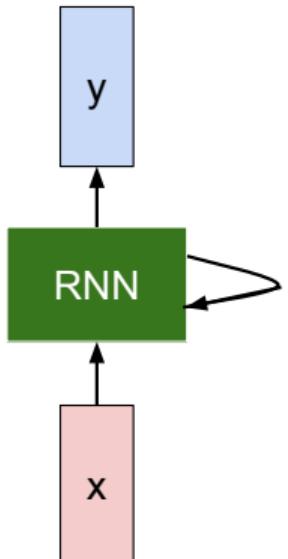


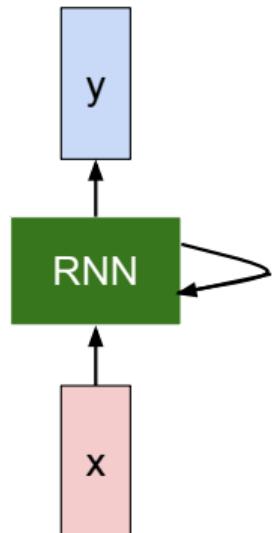


We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

$$h_t = f_W(h_{t-1}, x_t)$$

Notice: the same function and the same set of parameters are used at every time step.





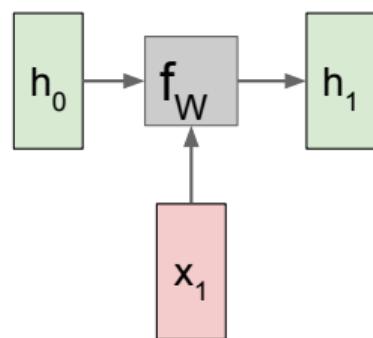
$$h_t = f_W(h_{t-1}, x_t)$$

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

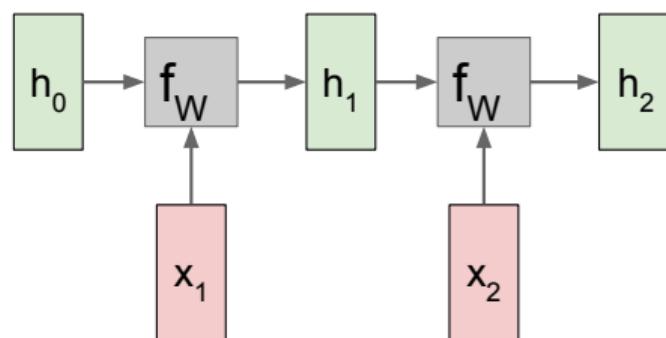
$$y_t = W_{hy}h_t$$

Sometimes called a “Vanilla RNN” or an
“Elman RNN” after Prof. Jeffrey Elman

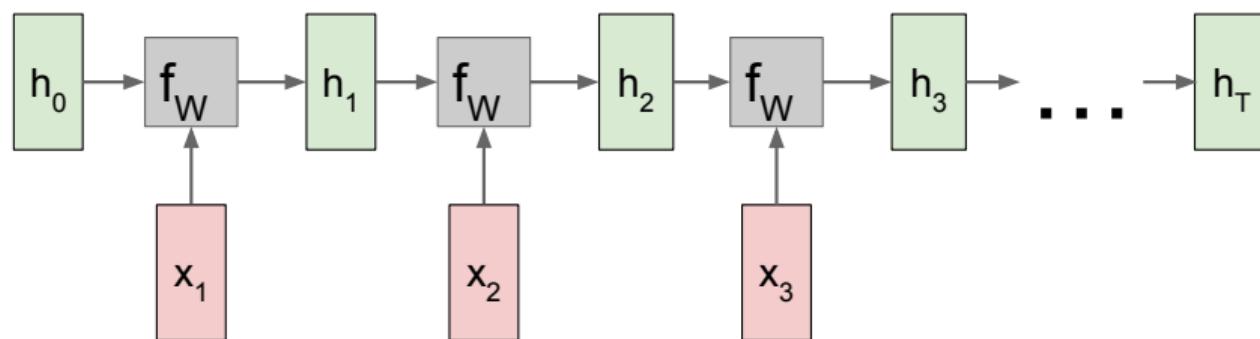
RNN: Computational Graph



RNN: Computational Graph

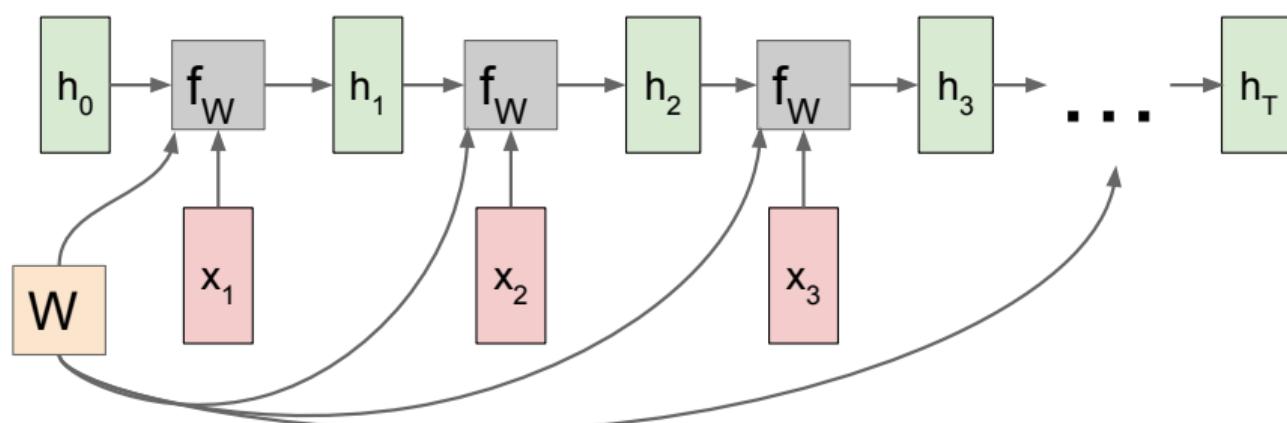


RNN: Computational Graph

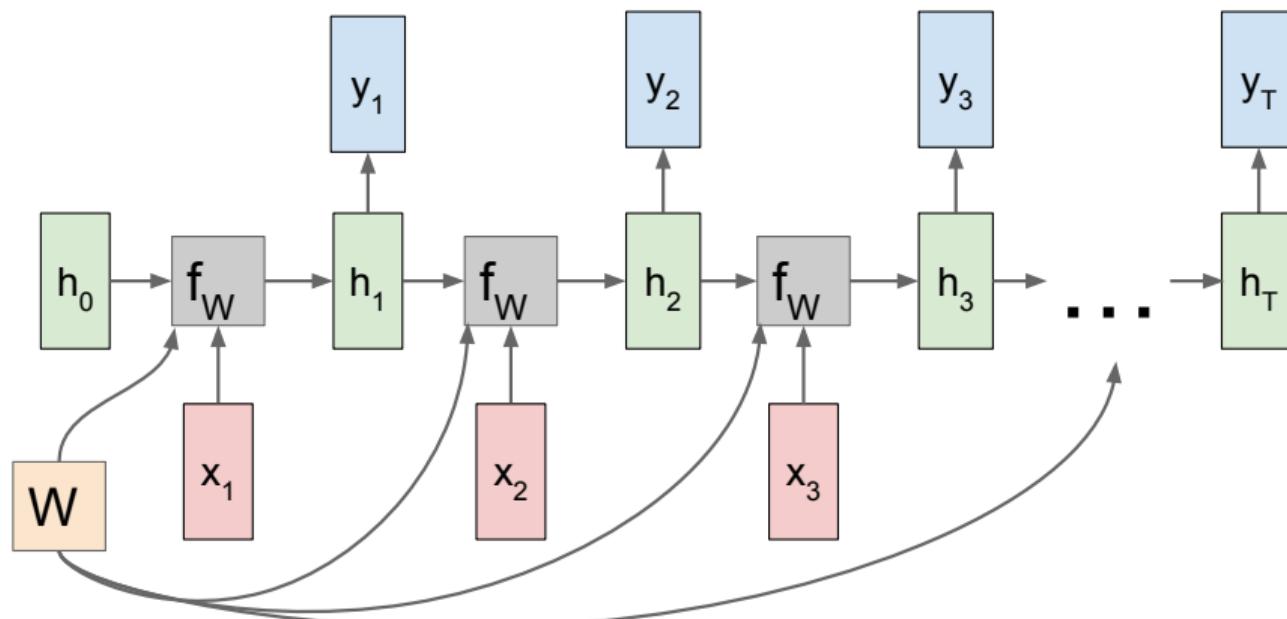


RNN: Computational Graph

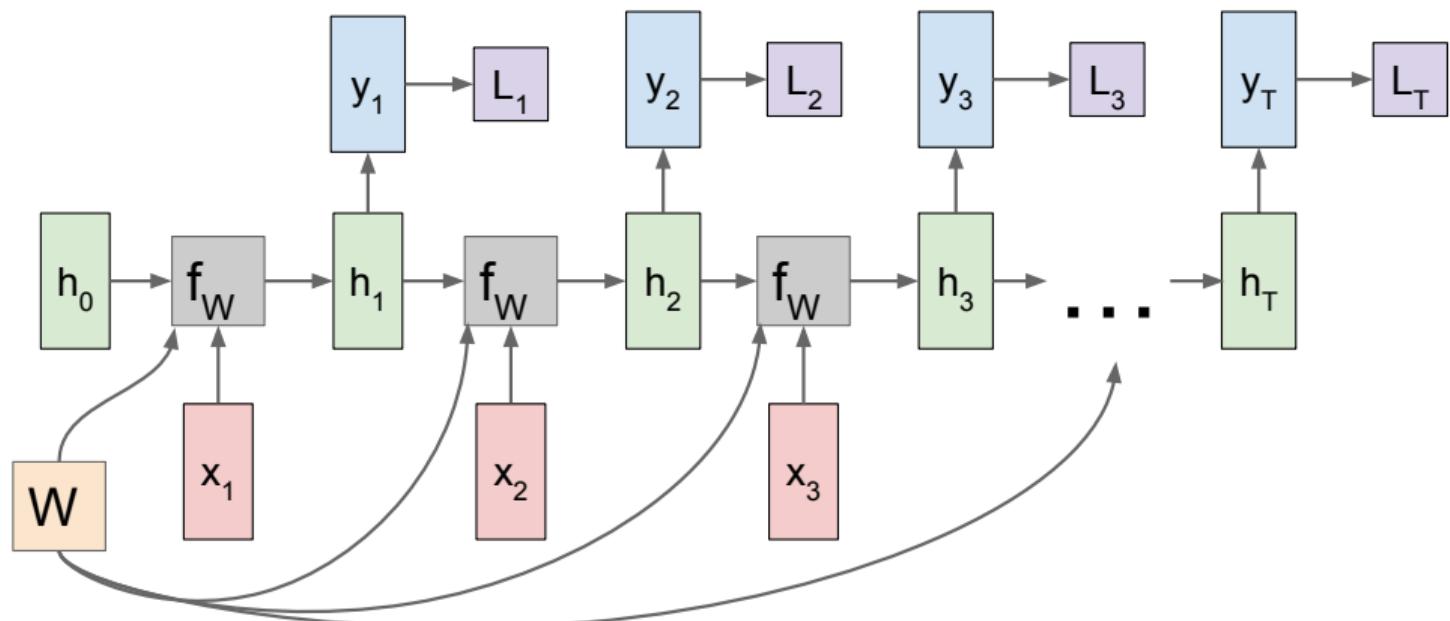
Re-use the same weight matrix at every time-step



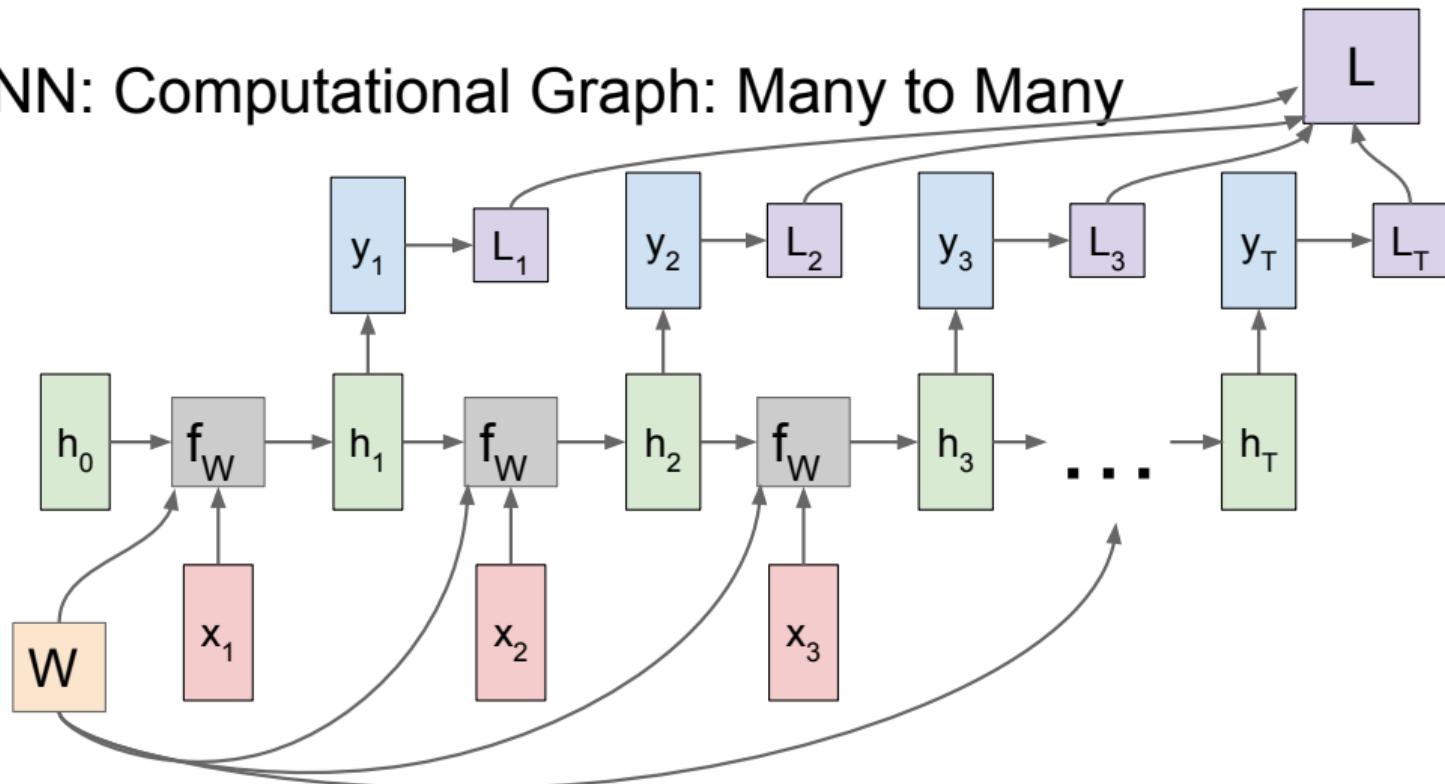
RNN: Computational Graph: Many to Many



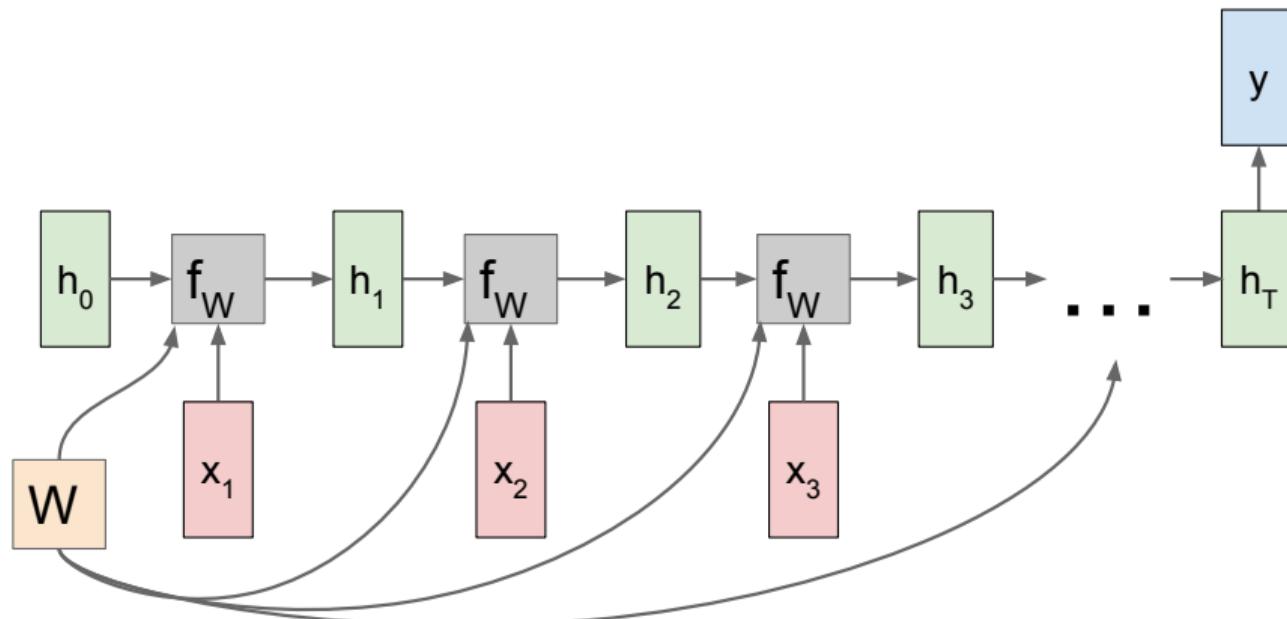
RNN: Computational Graph: Many to Many



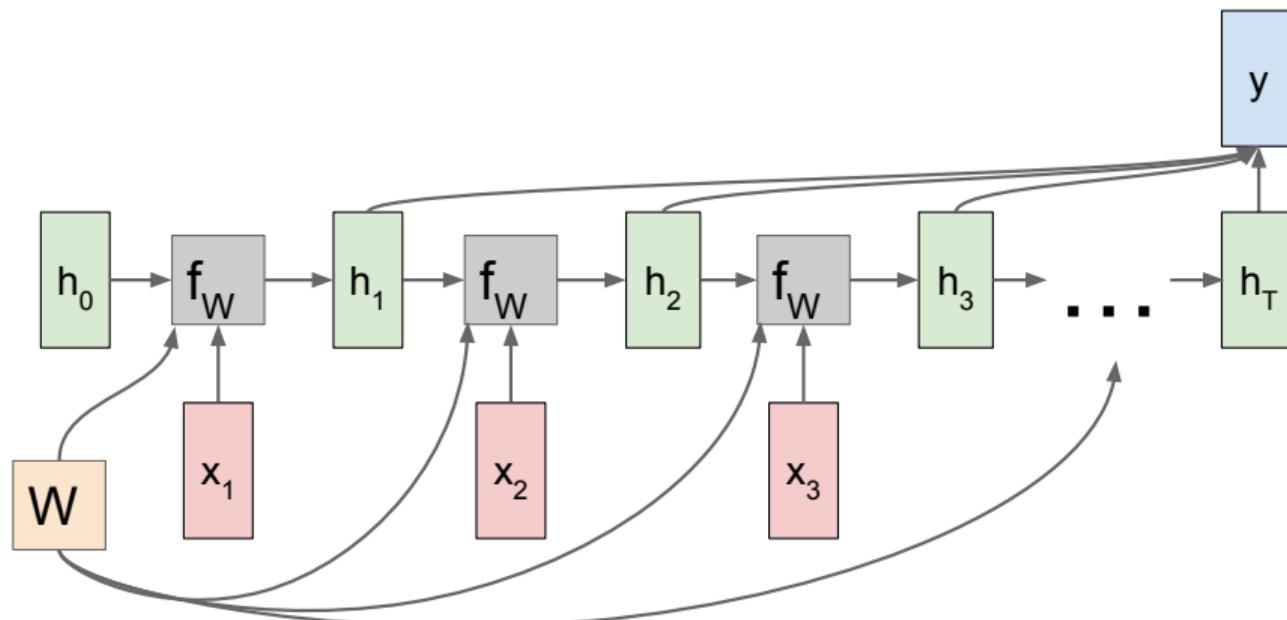
RNN: Computational Graph: Many to Many



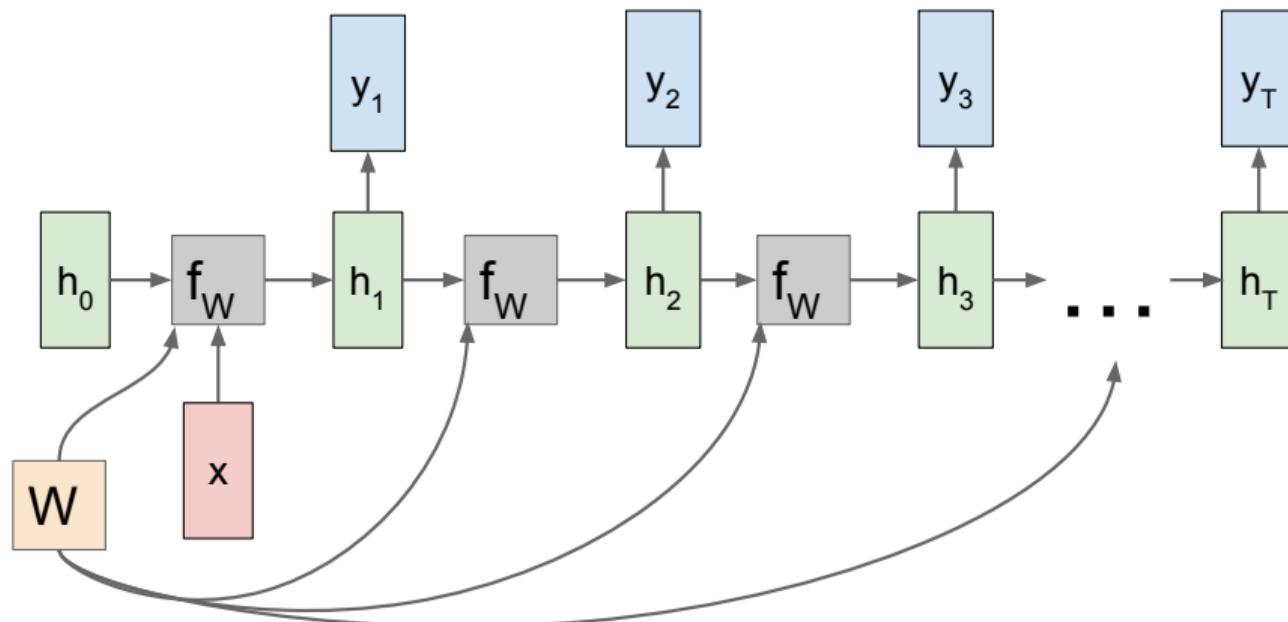
RNN: Computational Graph: Many to One



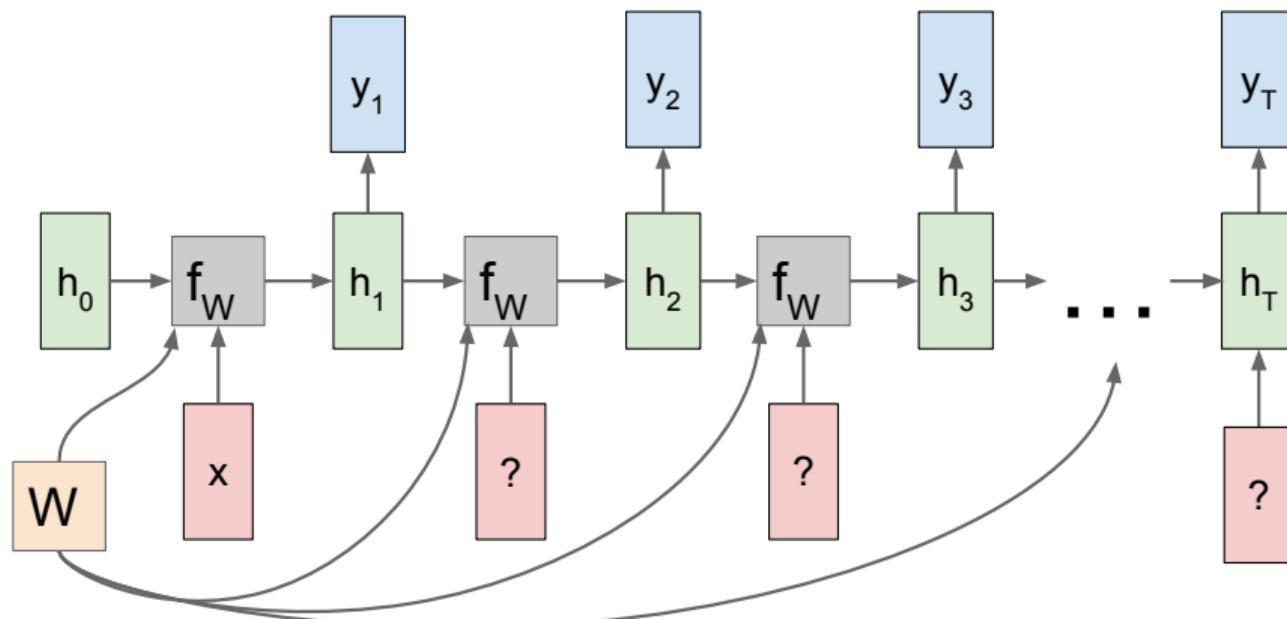
RNN: Computational Graph: Many to One



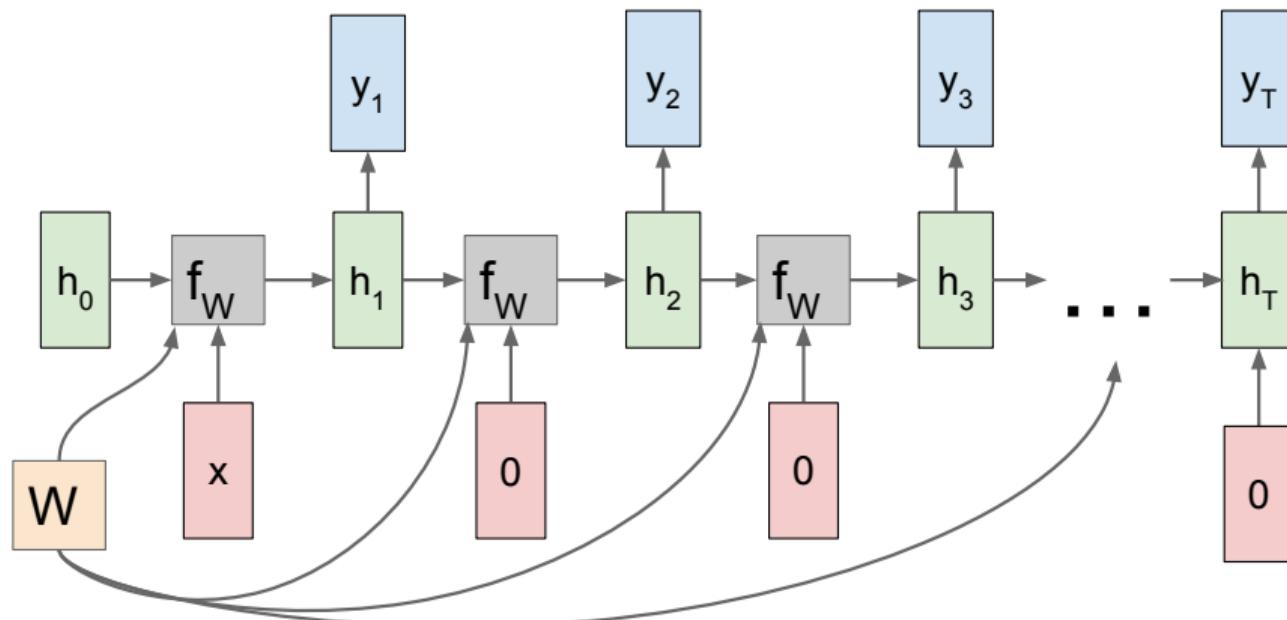
RNN: Computational Graph: One to Many



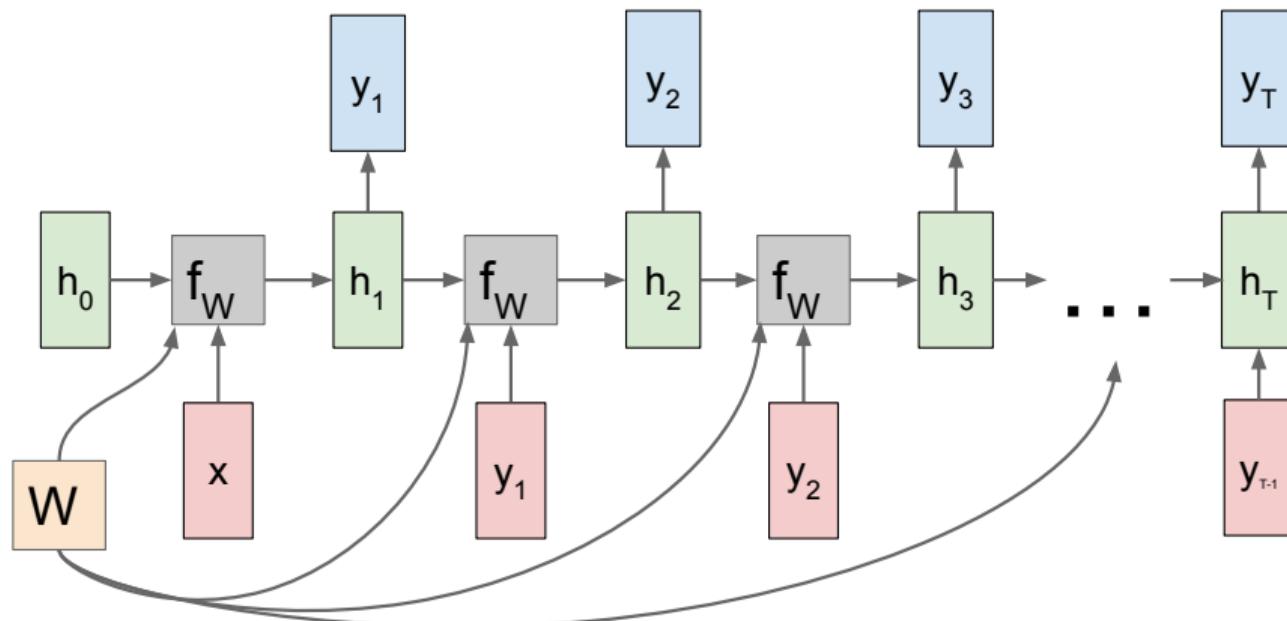
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many

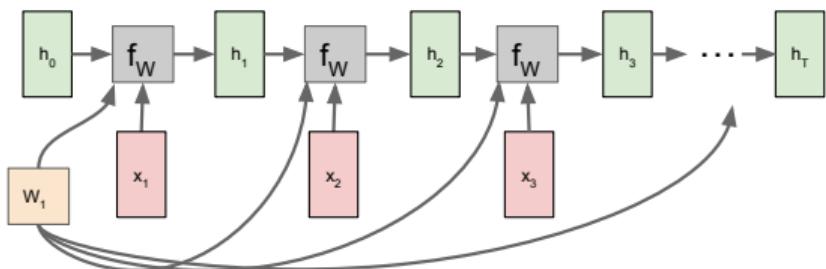


RNN: Computational Graph: One to Many



Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

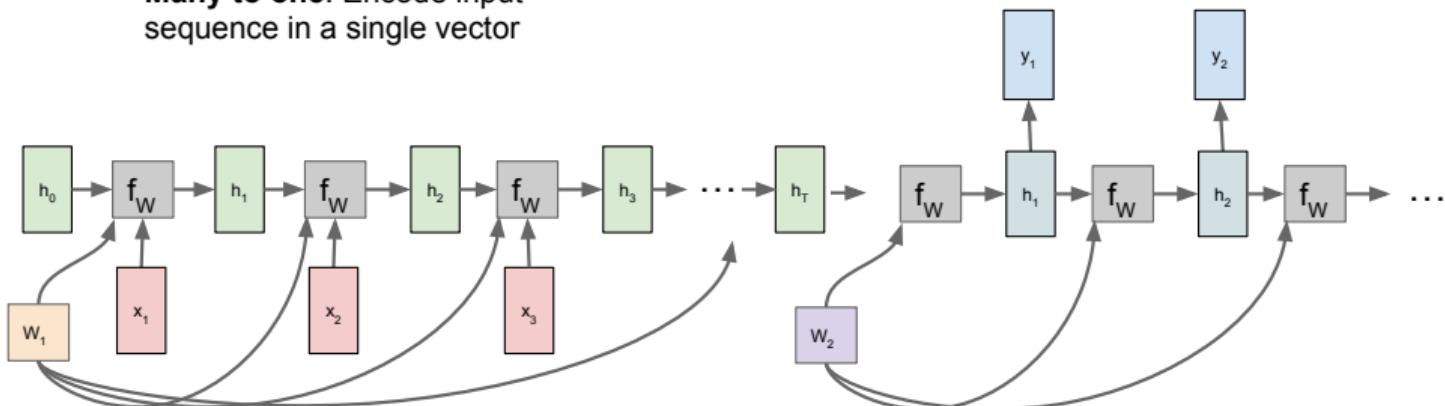


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Sequence to Sequence: Many-to-one + one-to-many

Many to one: Encode input sequence in a single vector

One to many: Produce output sequence from single input vector



Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

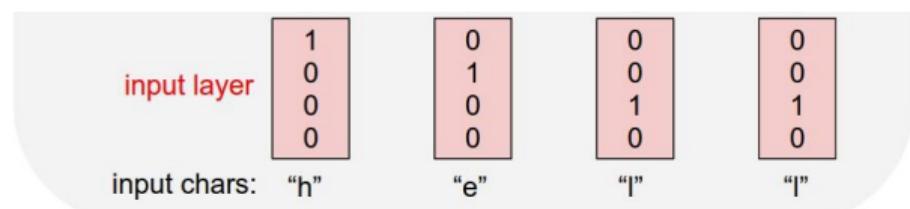
Simple Example



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”



Simple Example

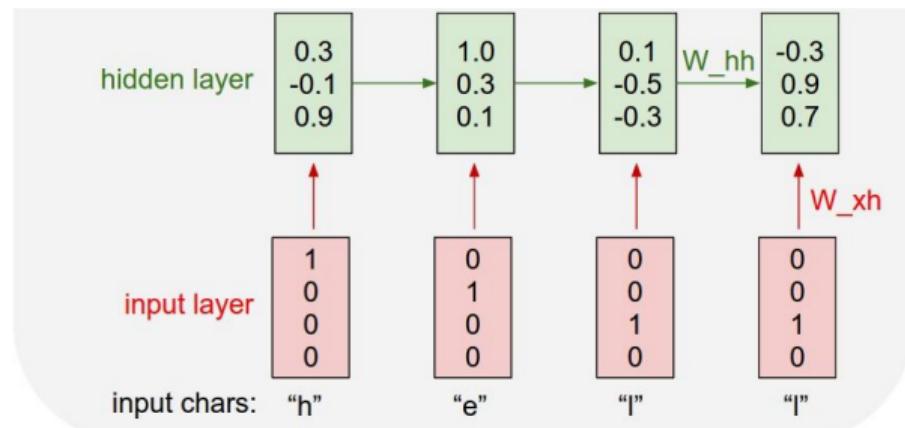


Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

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



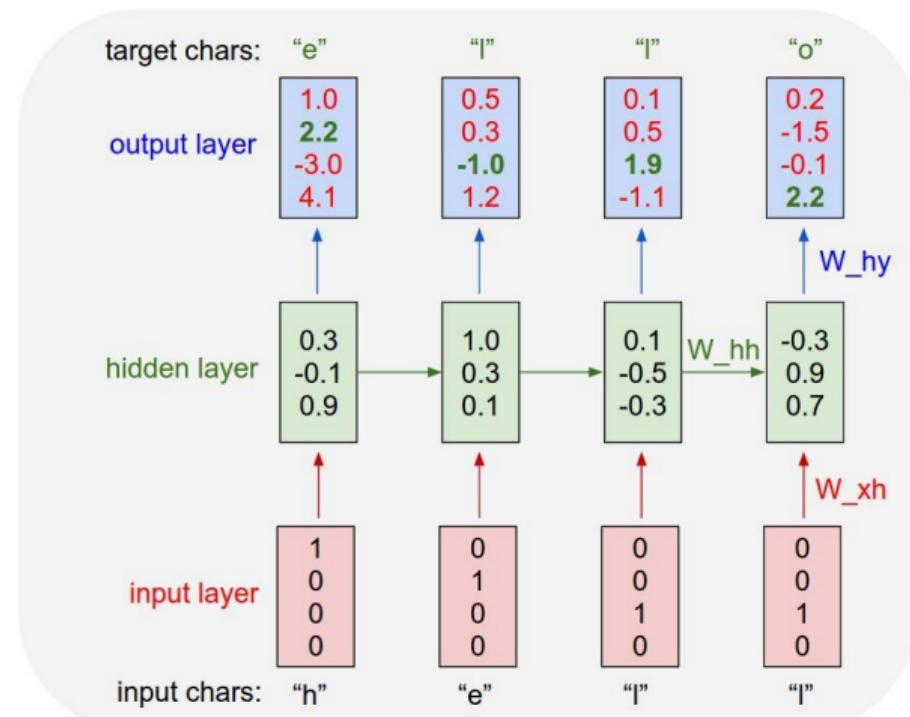
Simple Example



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”



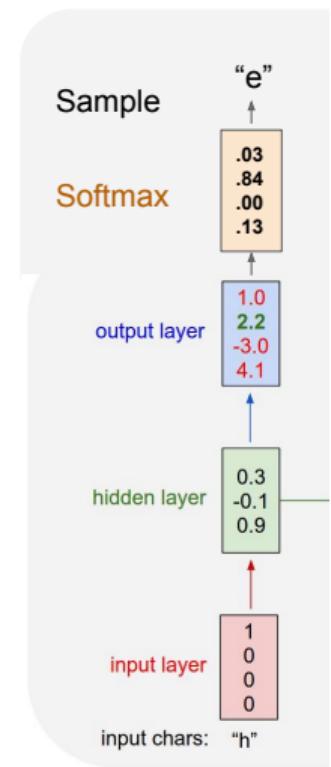
Simple Example



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a time,
feed back to model



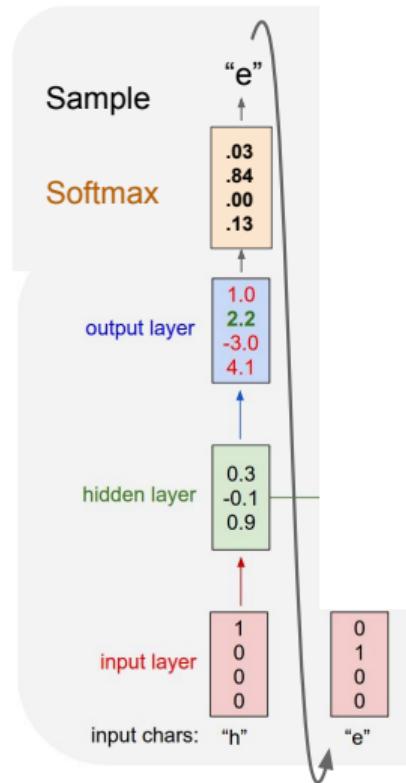
Simple Example



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a time,
feed back to model



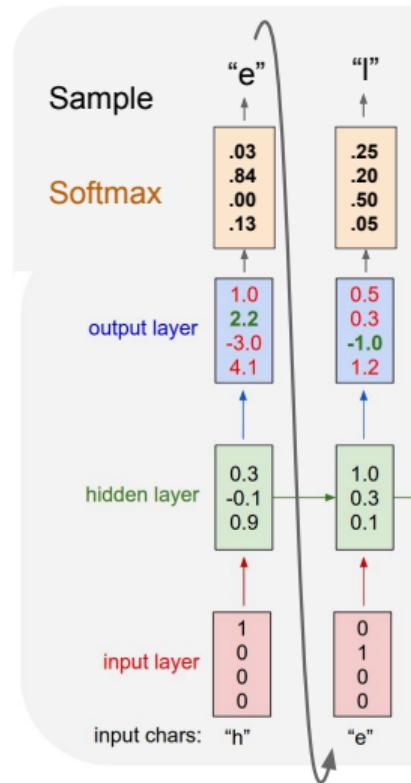
Simple Example



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

At test-time sample
characters one at a time,
feed back to model



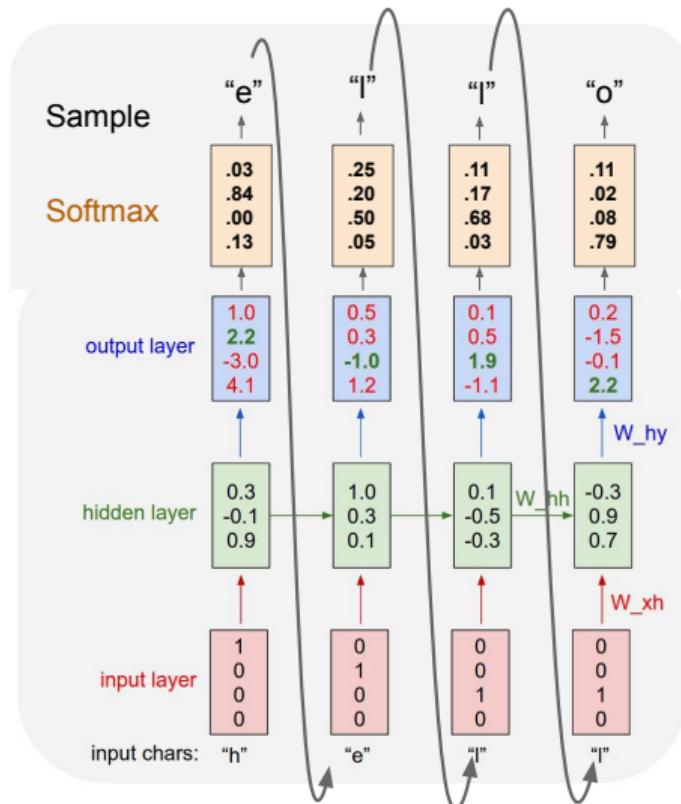
Simple Example



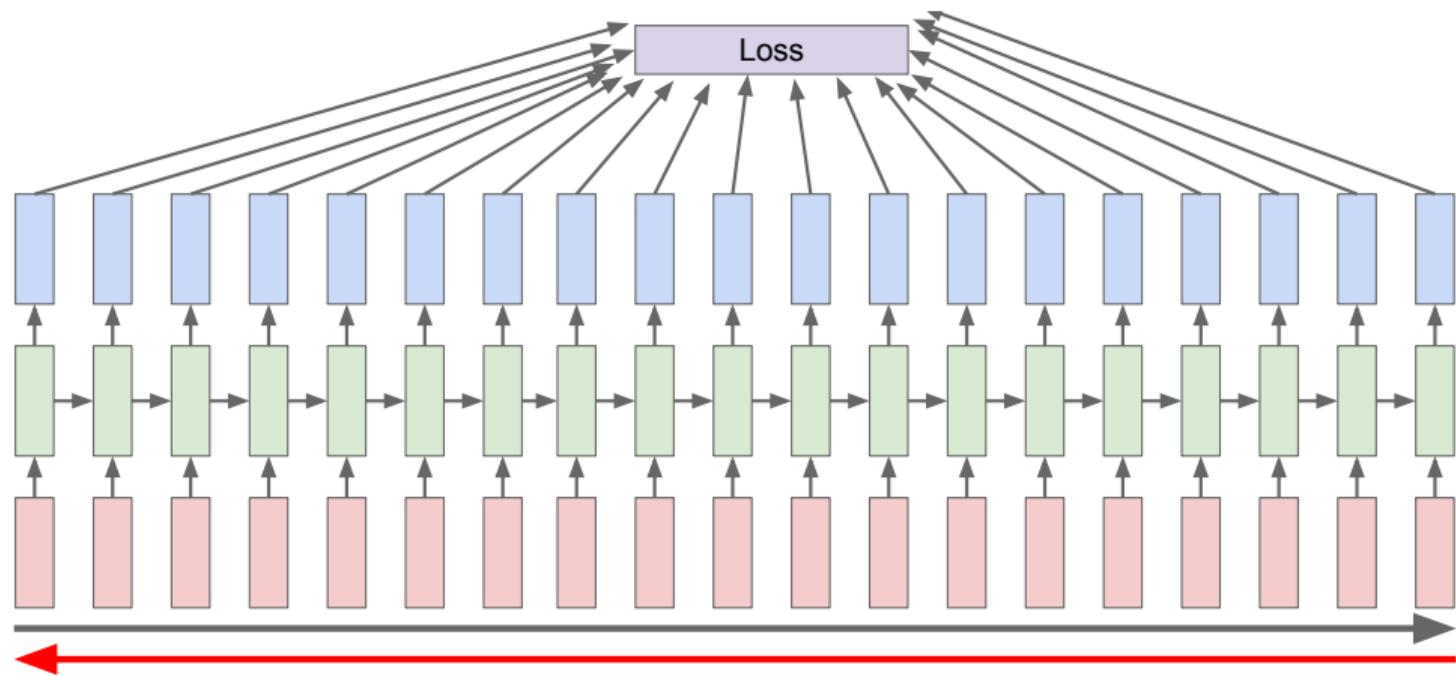
Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

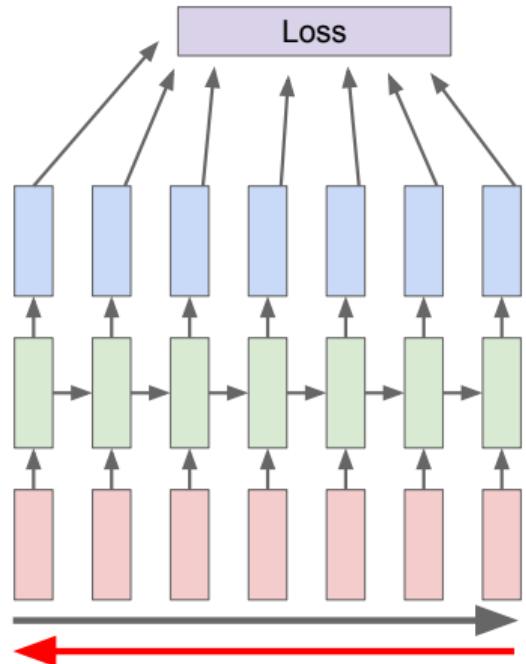
At test-time sample
characters one at a time,
feed back to model



How to backpropagate

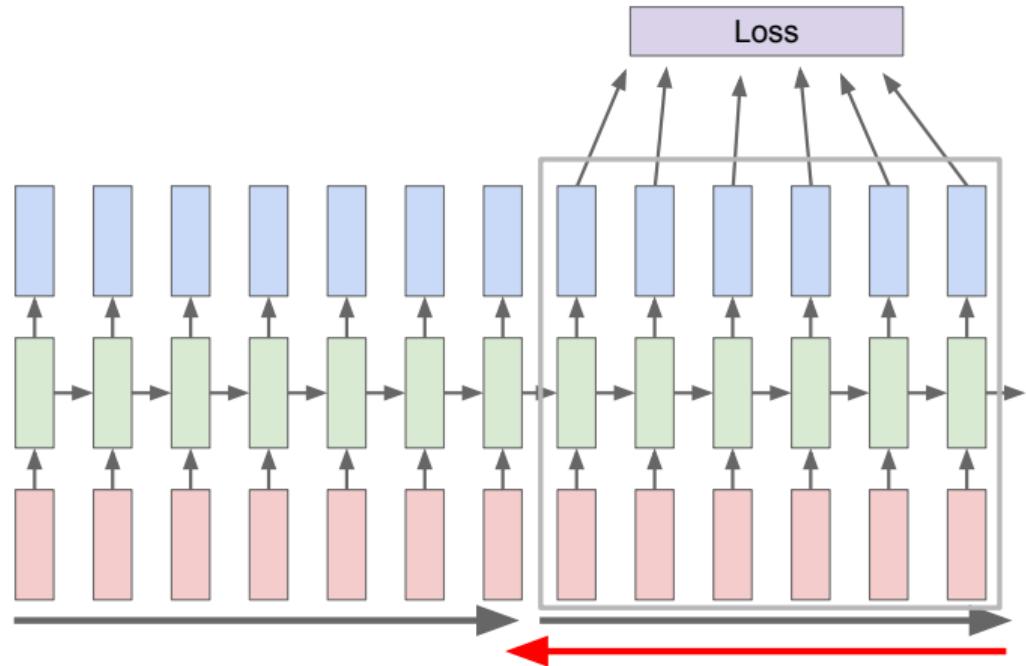


How to backpropagate



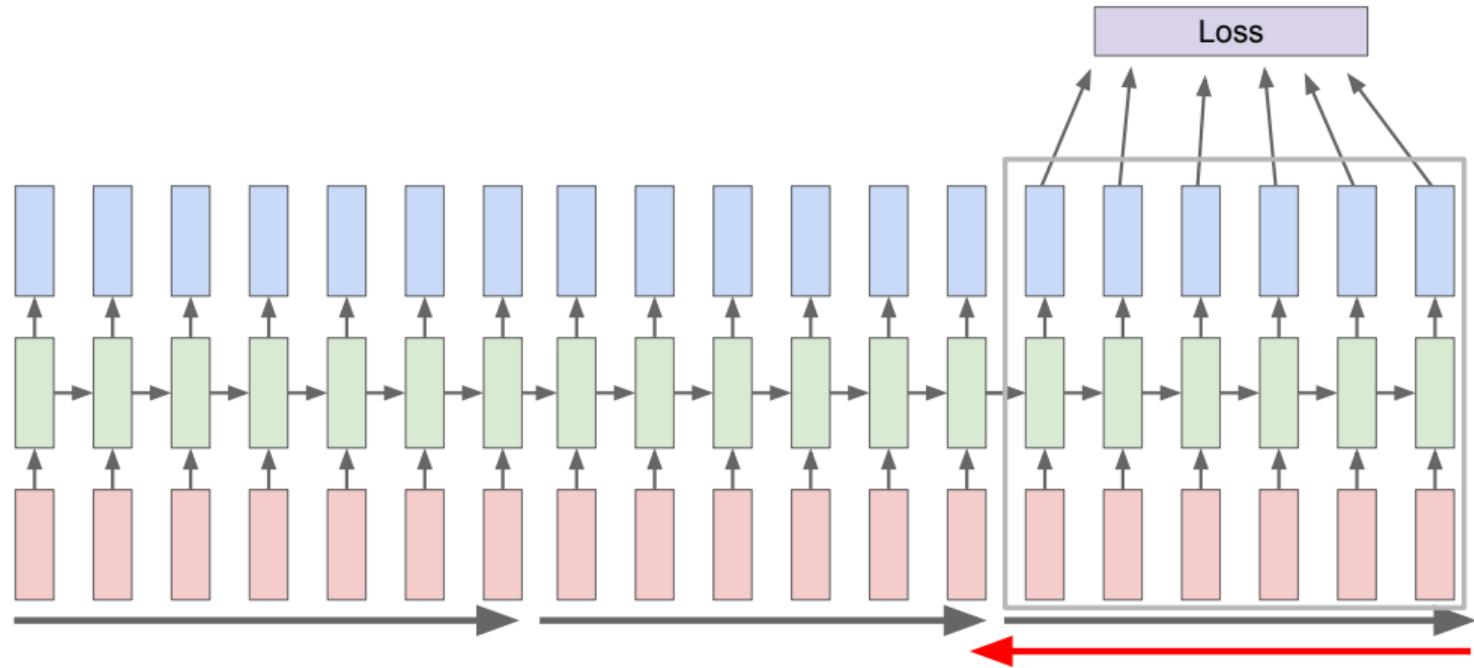
Run forward and backward
through chunks of the
sequence instead of whole
sequence

How to backpropagate



Carry hidden states forward in time forever, but only backpropagate for some smaller number of steps

How to backpropagate





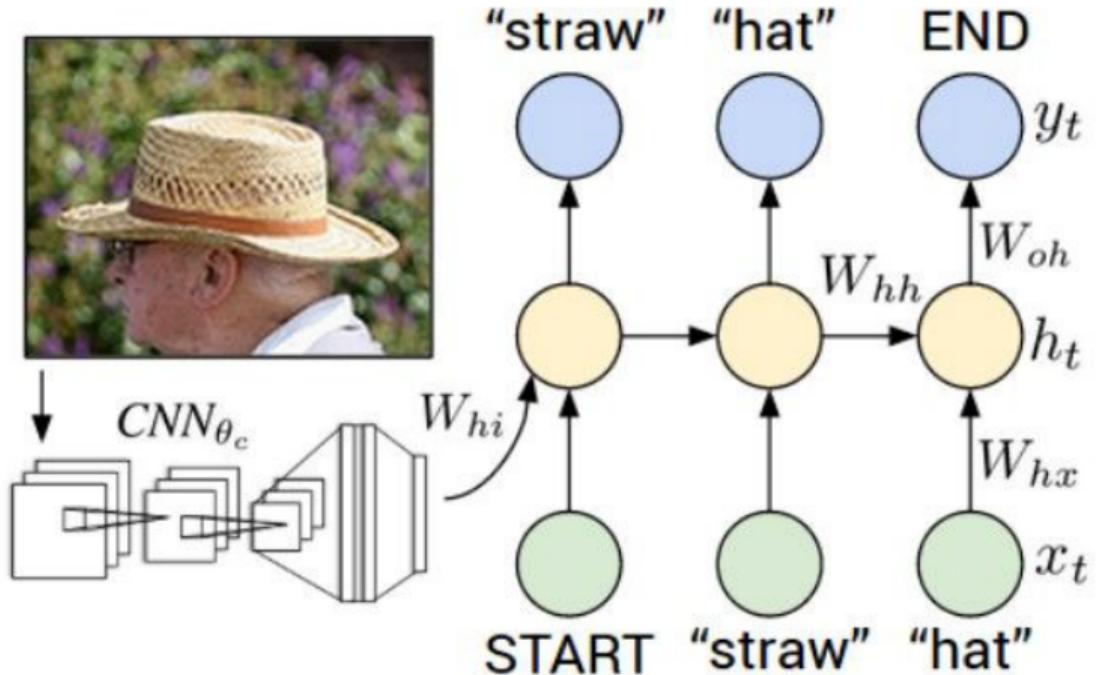
Pros:

- Variable input/output
- Shared weights for steps
- Theoretically can access information in any previous step
- Constant model size for various inputs

Cons:

- Backprop may require too much memory
- Potentially unstable training
- Information from previous steps might be difficult to access
- Slow computation

Image Captioning



Andrej Karpathy and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3128–3137



Vanilla RNN

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

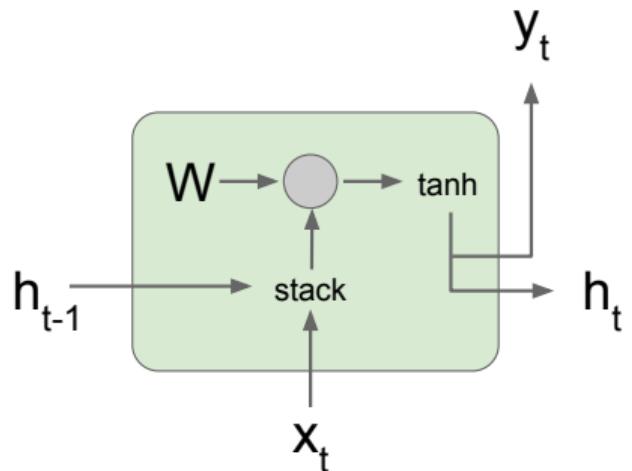
LSTM

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

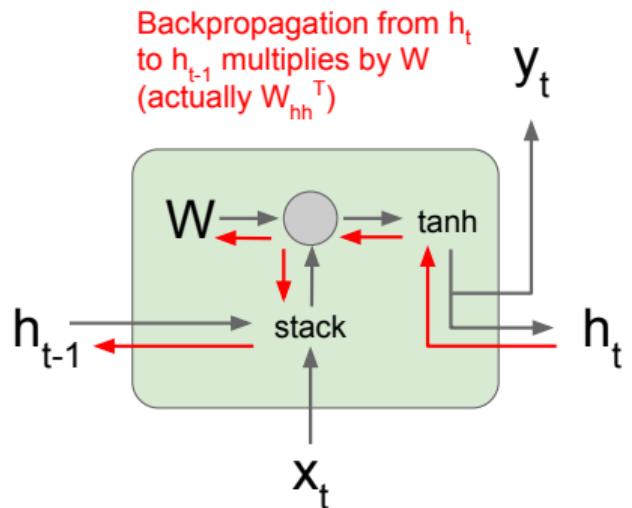
$$h_t = o \odot \tanh(c_t)$$

RNN Backprop



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

RNN Backprop

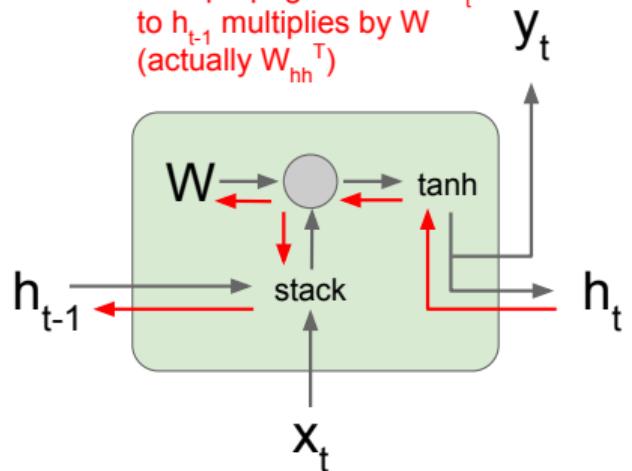


$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

RNN Backprop



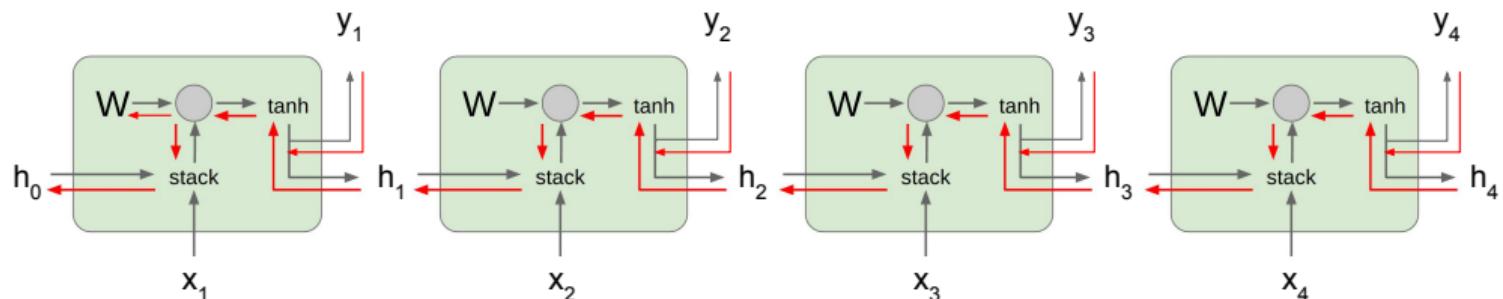
Backpropagation from h_t to h_{t-1} multiplies by W (actually W_{hh}^T)



$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

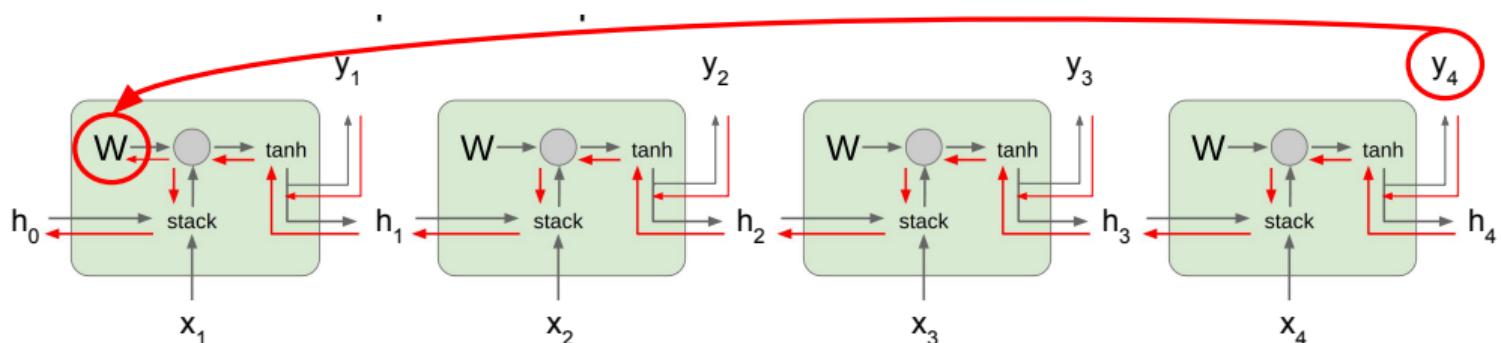
$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

RNN Backprop



$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

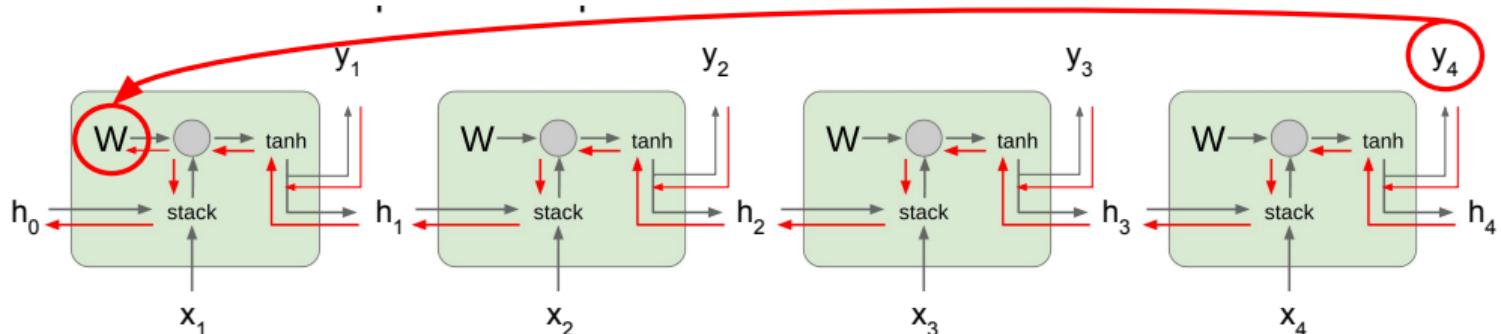
RNN Backprop



$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W}$$

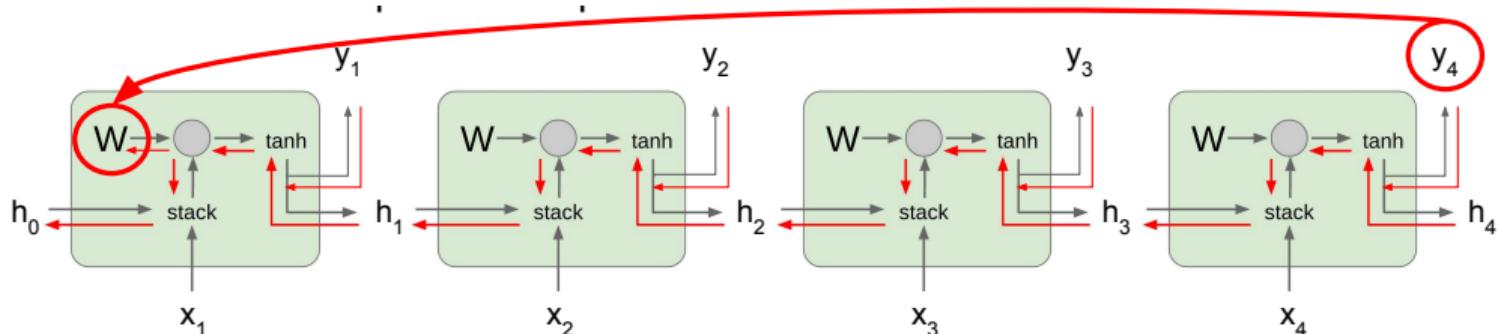
RNN Backprop



$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^T \frac{\partial h_t}{\partial h_{t-1}} \right) \frac{\partial h_1}{\partial W}$$

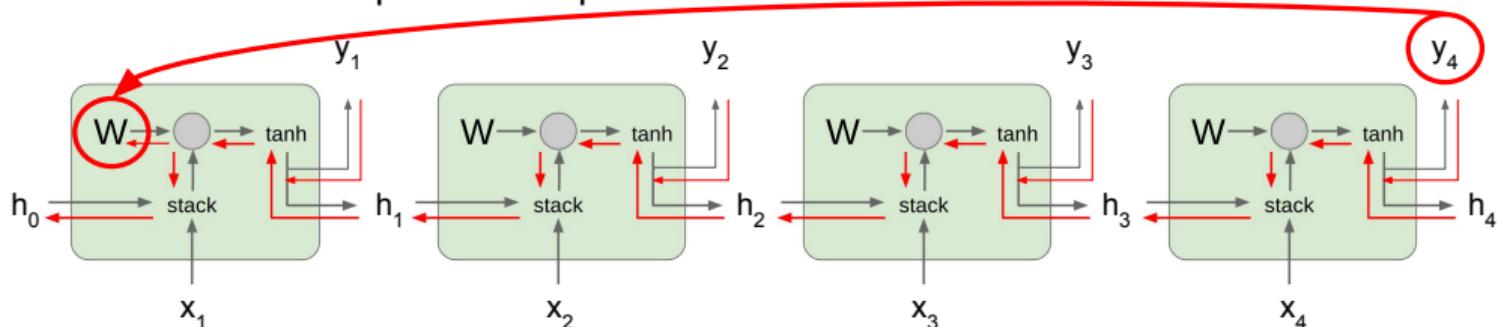
RNN Backprop



$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W} \quad \boxed{\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}}$$

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \frac{\partial h_t}{\partial h_{t-1}} \cdots \frac{\partial h_1}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^T \boxed{\frac{\partial h_t}{\partial h_{t-1}}} \right) \frac{\partial h_1}{\partial W}$$

RNN Backprop

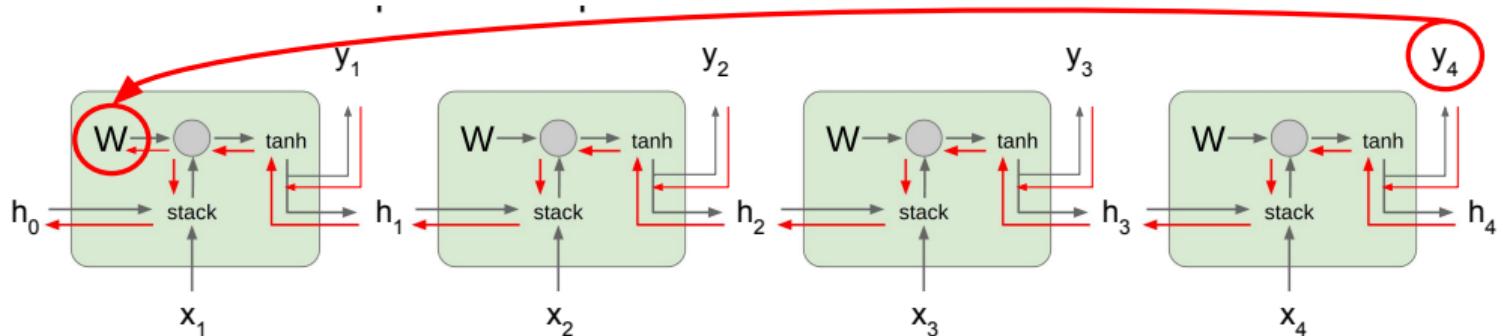


$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

Almost always < 1
Vanishing gradients

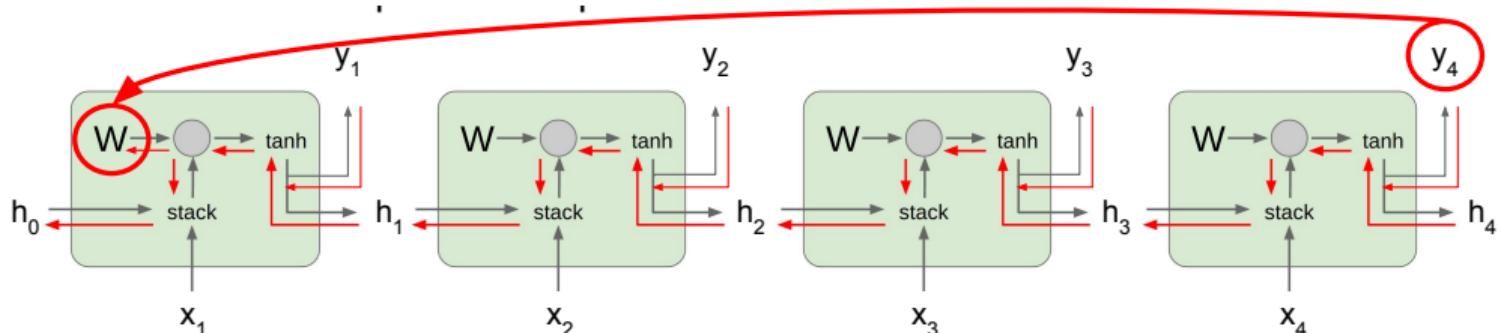
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \left(\prod_{t=2}^T \boxed{\tanh'(W_{hh}h_{t-1} + W_{xh}x_t)} \right) W_{hh}^{T-1} \frac{\partial h_1}{\partial W}$$

RNN Backprop



$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W} \quad \text{What if we assumed no non-linearity?}$$

RNN Backprop



What if we assumed no non-linearity?

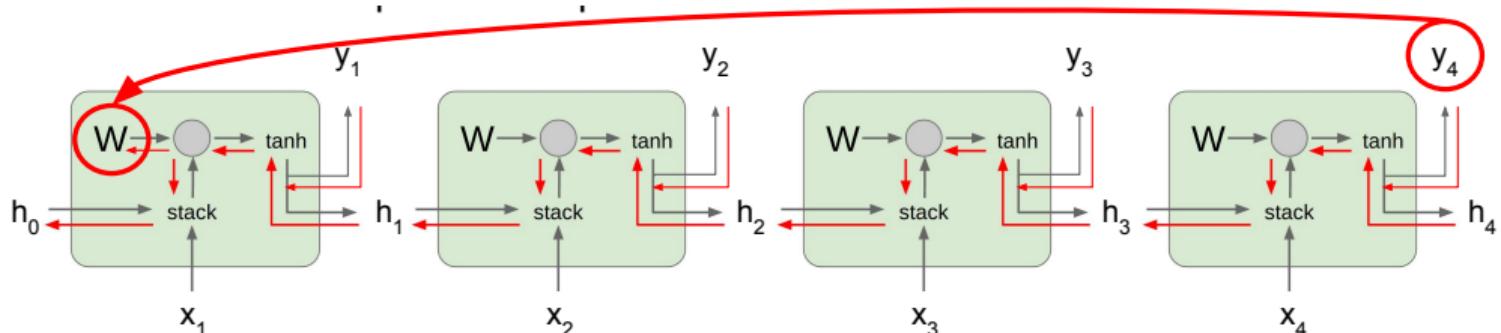
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

Largest singular value > 1 :
Exploding gradients

$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Largest singular value < 1 :
Vanishing gradients

RNN Backprop



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

Largest singular value > 1:
Exploding gradients

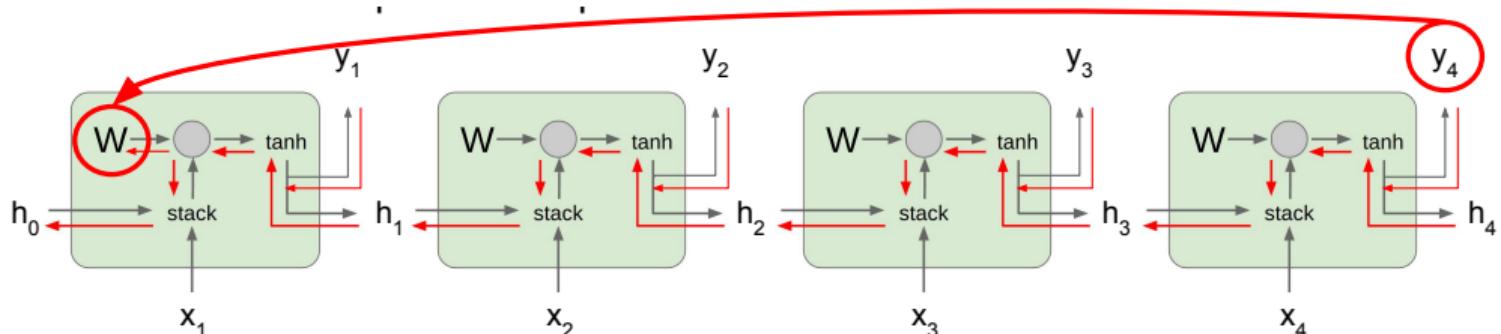
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Largest singular value < 1:
Vanishing gradients

→ **Gradient clipping:**
Scale gradient if its
norm is too big

```
grad_norm = np.sum(grad * grad)
if grad_norm > threshold:
    grad *= (threshold / grad_norm)
```

RNN Backprop



What if we assumed no non-linearity?

$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

Largest singular value > 1 :
Exploding gradients

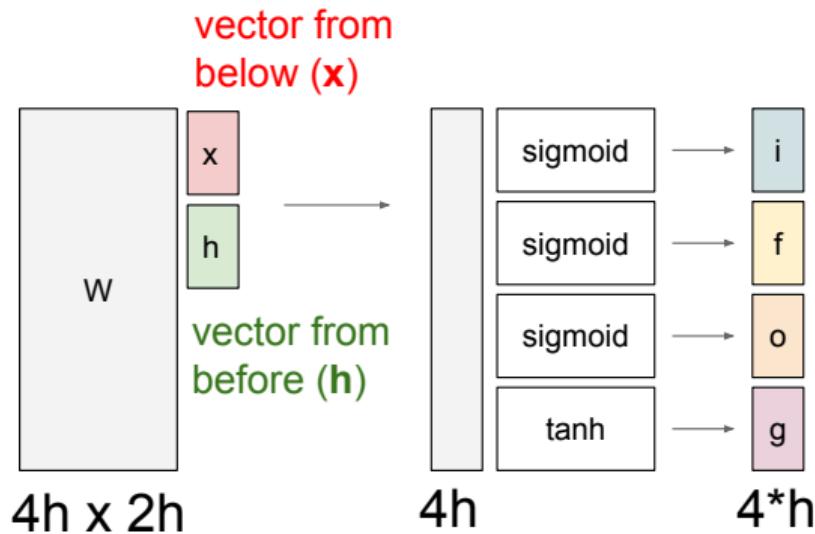
$$\frac{\partial L_T}{\partial W} = \frac{\partial L_T}{\partial h_T} \boxed{W_{hh}^{T-1}} \frac{\partial h_1}{\partial W}$$

Largest singular value < 1 :
Vanishing gradients

→ Change RNN architecture

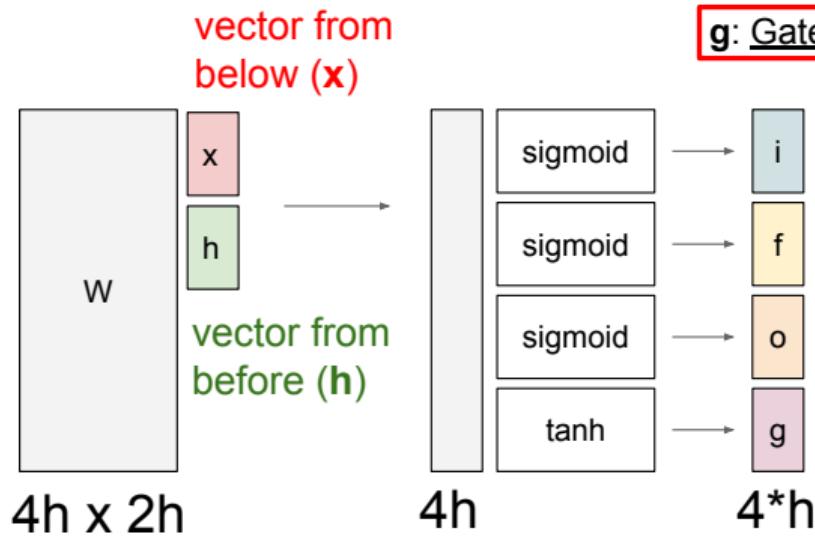
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



g: Gate gate (?), How much to write to cell

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

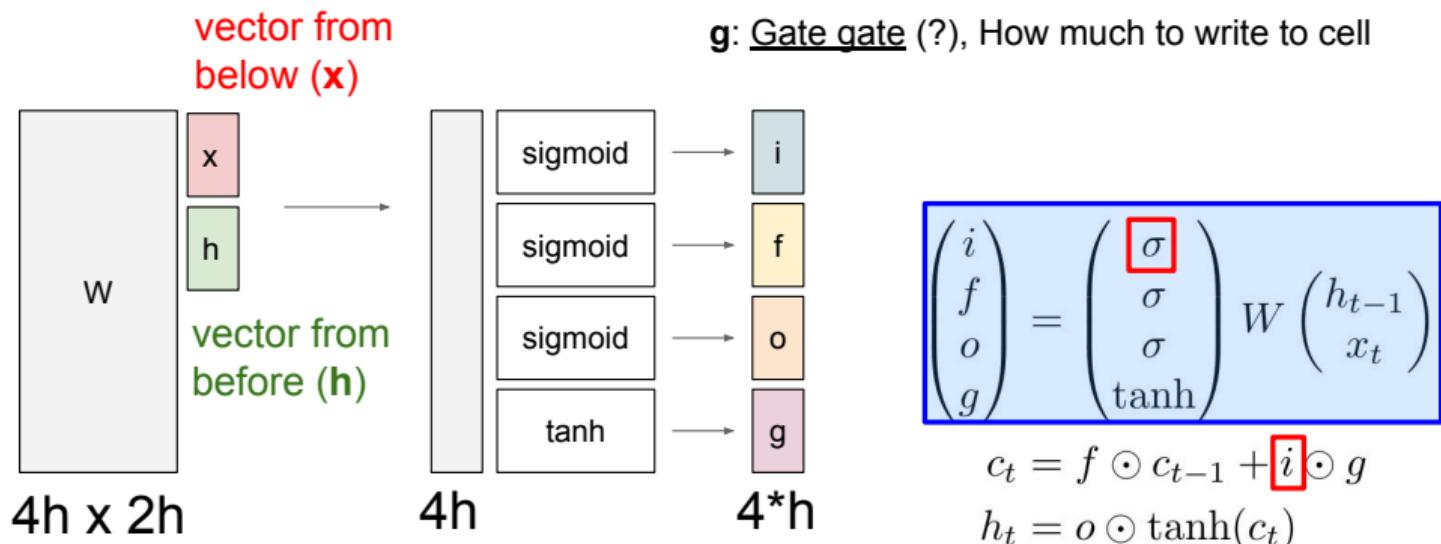
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

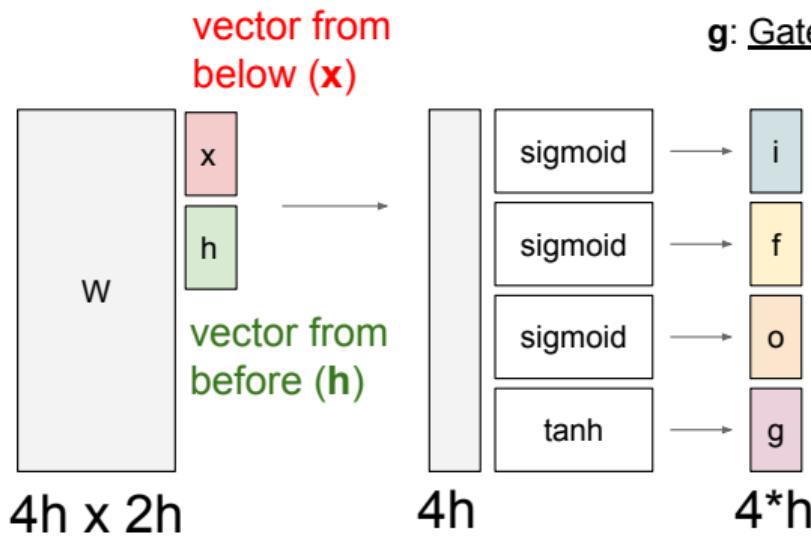
i: Input gate, whether to write to cell



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell



g: Gate gate (?), How much to write to cell

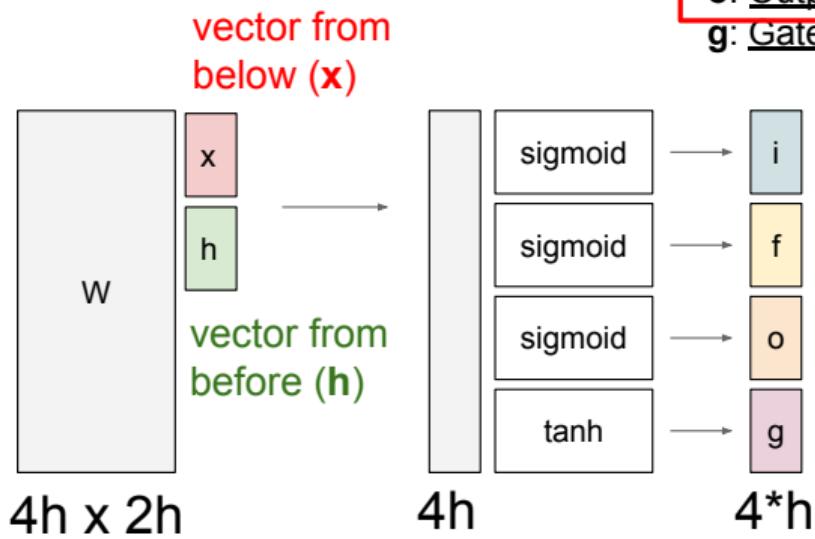
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



- i: Input gate, whether to write to cell
- f: Forget gate. Whether to erase cell
- o: Output gate, How much to reveal cell**
- g: Gate gate (?), How much to write to cell

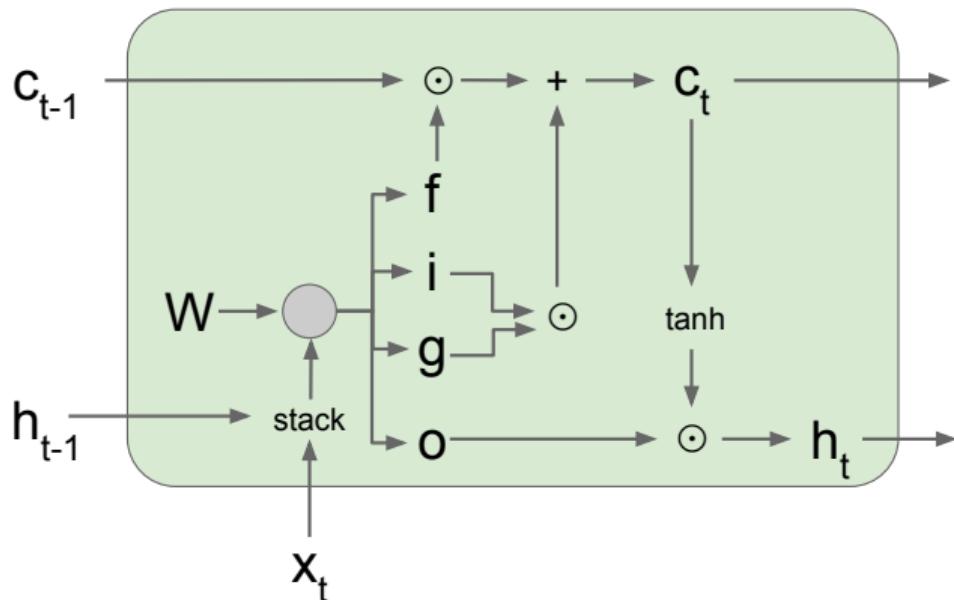
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



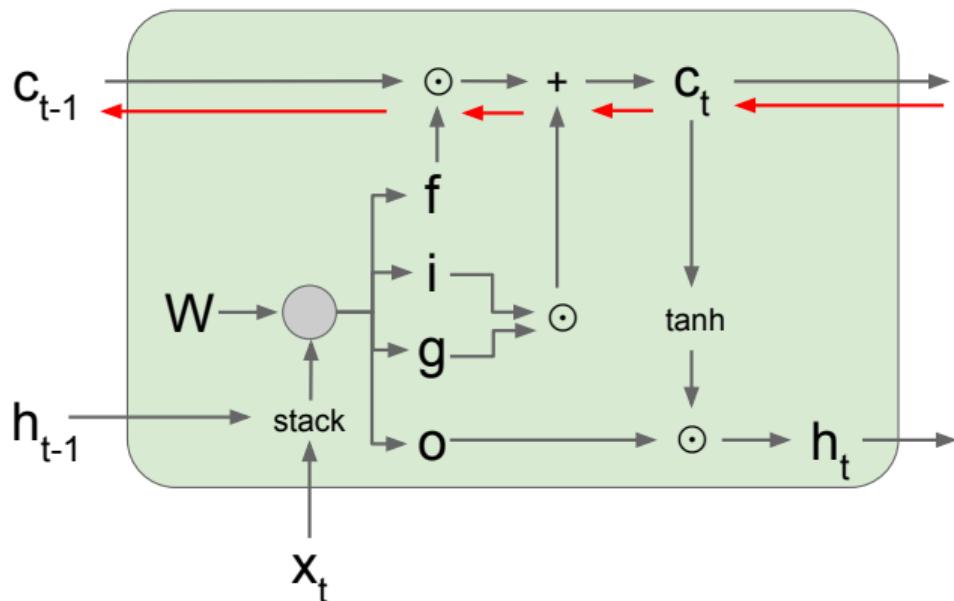
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

$$c_t = f \odot c_{t-1} + i \odot g$$

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

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]



Backpropagation from c_t to c_{t-1} only elementwise multiplication by f , no matrix multiply by W

$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

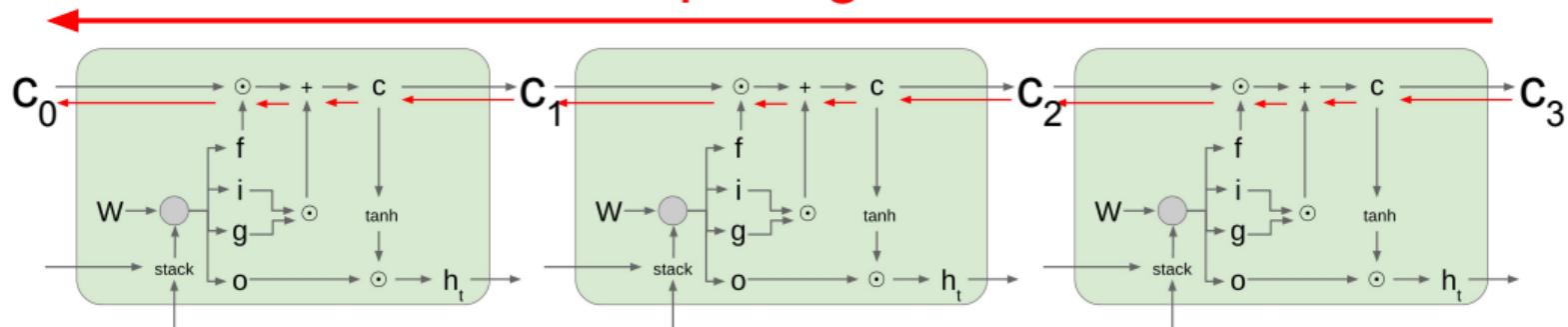
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Notice that the gradient contains the **f** gate's vector of activations

- allows better control of gradients values, using suitable parameter updates of the forget gate.

Also notice that are added through the **f**, **i**, **g**, and **o** gates

- better balancing of gradient values

Do LSTMs solve the vanishing gradient problem?

The LSTM architecture makes it easier for the RNN to preserve information over many timesteps

- e.g. **if the $f = 1$ and the $i = 0$** , then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix W_h that preserves info in hidden state

LSTM **doesn't guarantee** that there is no vanishing/exploding gradient, but it does provide an easier way for the model to learn long-distance dependencies



Gated Recurrent Unit⁴:

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

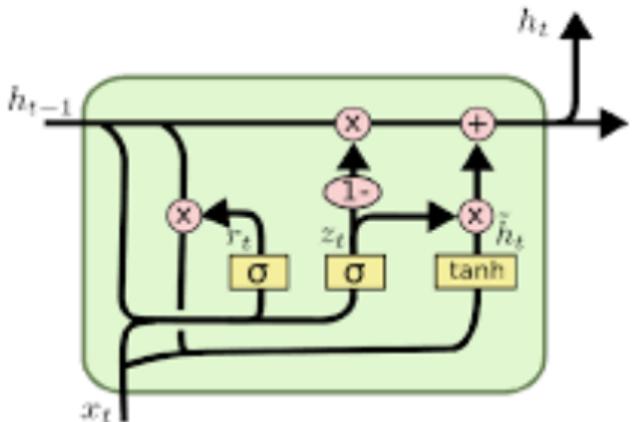
$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$\hat{h}_t = \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$

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

⁴Kyunghyun Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." In: *arXiv preprint arXiv:1406.1078* (2014)

GRU



GRU is like LSTM with a forget gate, but no output gate. It has fewer parameters and is usually easier to train!

Using RNNs



It is probably a better idea to use Transformers nowadays, but if you choose RNNs there are some tips based on my own limited experience:

- GRU is easier to train than LSTM.
- Selecting the proper training regime is mostly based on the amount of memory available for training.
- Keeping hidden states during training might introduce problems.
- ELU or PReLU may be better activations than ReLU.

Transformers



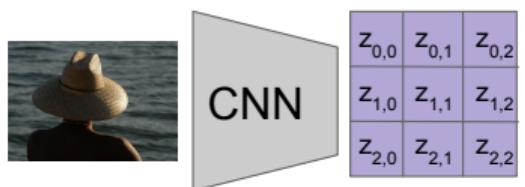
- 2014-2016 - RNNs were dominant technique to use on sequential data in both NLP and CV.
- 2015-2016 - ResNets and Attention show that there are better ways to solve some issues than LSTM
- 2017 - Transformers dominate in NLP
- 2018-2020 - Transformers get applied to CV with varying levels of success
- 2021 - Are Transformers the new CNNs?

Attention in CV



Input: Image I

Output: Sequence $y = y_1, y_2, \dots, y_T$



Extract spatial
features from a
pretrained CNN

Features:
 $H \times W \times D$

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV



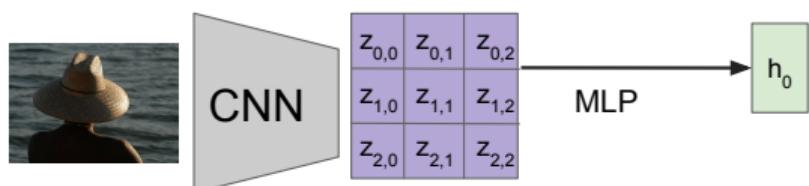
Input: Image I

Output: Sequence $y = y_1, y_2, \dots, y_T$

Encoder: $h_0 = f_w(z)$

where z is spatial CNN features

$f_w(\cdot)$ is an MLP



Extract spatial
features from a
pretrained CNN

Features:
 $H \times W \times D$

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV



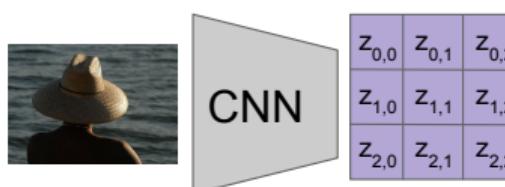
Input: Image I

Output: Sequence $y = y_1, y_2, \dots, y_T$

Encoder: $h_0 = f_w(z)$

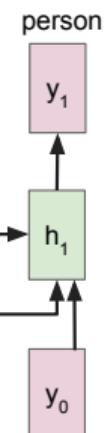
where z is spatial CNN features

$f_w(\cdot)$ is an MLP



Extract spatial
features from a
pretrained CNN

Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

[START]

Attention in CV



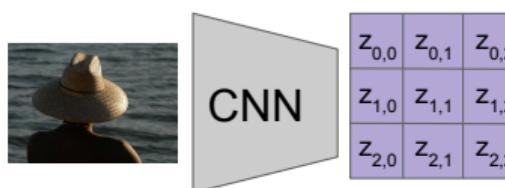
Input: Image I

Output: Sequence $y = y_1, y_2, \dots, y_T$

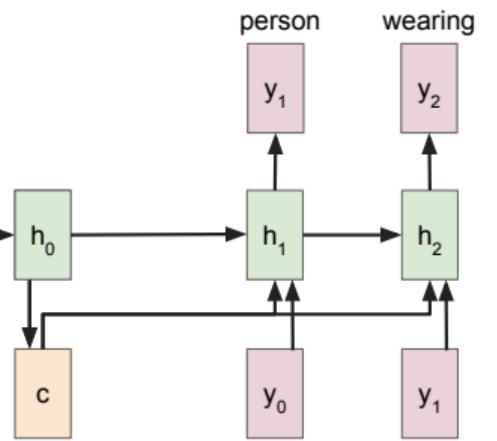
Encoder: $h_0 = f_w(z)$

where z is spatial CNN features

$f_w(\cdot)$ is an MLP



Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



Attention in CV



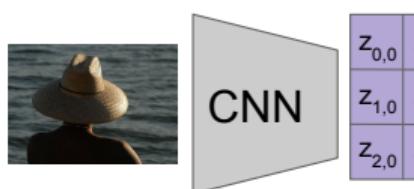
Input: Image I

Output: Sequence $y = y_1, y_2, \dots, y_T$

Encoder: $h_0 = f_w(z)$

where z is spatial CNN features

$f_w(\cdot)$ is an MLP

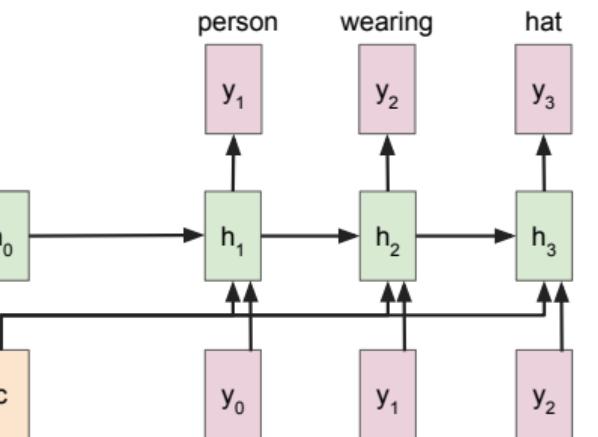


$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

Features:
 $H \times W \times D$

MLP

Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



Attention in CV



Input: Image I

Output: Sequence $y = y_1, y_2, \dots, y_T$

Encoder: $h_0 = f_w(z)$

where z is spatial CNN features

$f_w(\cdot)$ is an MLP

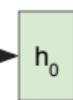


Extract spatial
features from a
pretrained CNN

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

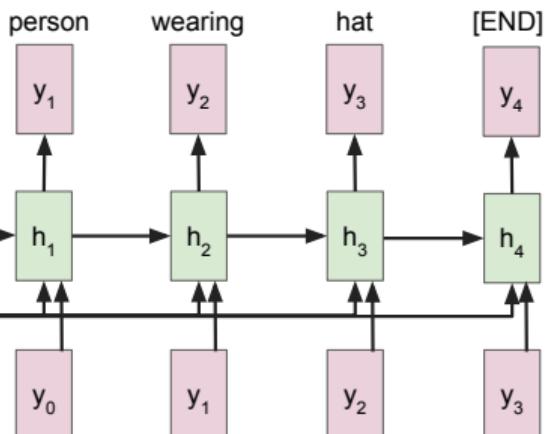
Features:
 $H \times W \times D$

MLP



c

Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$
where context vector c is often $c = h_0$



Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

[START]

person

wearing

hat

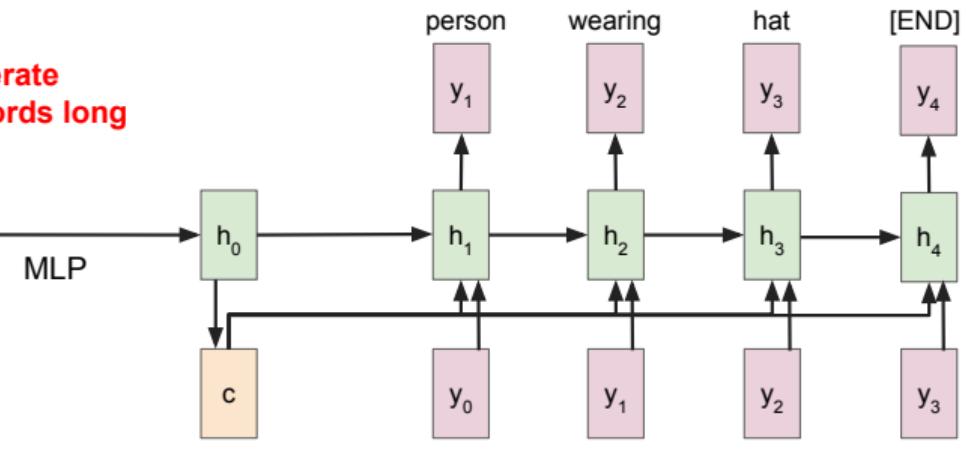
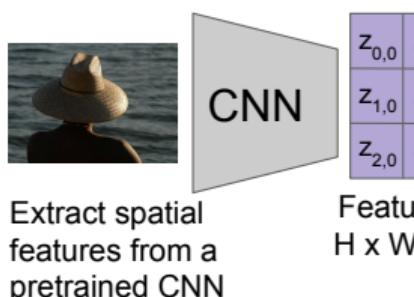
Attention in CV



Problem: Input is "bottlenecked" through c

- Model needs to encode everything it wants to say within c

This is a problem if we want to generate really long descriptions? 100s of words long

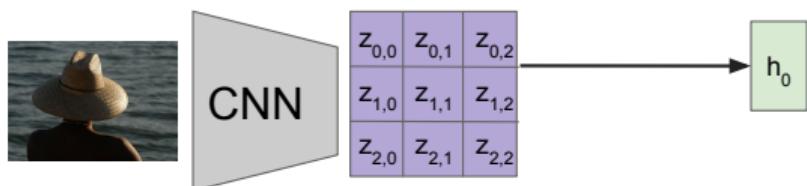


Attention in CV



Attention idea: New context vector at every time step.

Each context vector will attend to different image regions



Extract spatial
features from a
pretrained CNN

Features:
 $H \times W \times D$



Attention Saccades in humans



Attention in CV

Compute alignments scores (scalars):

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$f_{att}(\cdot)$ is an MLP



Extract spatial features from a pretrained CNN

Alignment scores:

$H \times W$

$e_{1,0,0}$	$e_{1,0,1}$	$e_{1,0,2}$
$e_{1,1,0}$	$e_{1,1,1}$	$e_{1,1,2}$
$e_{1,2,0}$	$e_{1,2,1}$	$e_{1,2,2}$

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

Features:
 $H \times W \times D$

h_0

Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015



Attention in CV

Compute alignments scores (scalars):

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$f_{att}(\cdot)$ is an MLP



Extract spatial features from a pretrained CNN

Alignment scores:
H x W

$e_{1,0,0}$	$e_{1,0,1}$	$e_{1,0,2}$
$e_{1,1,0}$	$e_{1,1,1}$	$e_{1,1,2}$
$e_{1,2,0}$	$e_{1,2,1}$	$e_{1,2,2}$

Attention:
H x W

$a_{1,0,0}$	$a_{1,0,1}$	$a_{1,0,2}$
$a_{1,1,0}$	$a_{1,1,1}$	$a_{1,1,2}$
$a_{1,2,0}$	$a_{1,2,1}$	$a_{1,2,2}$

Normalize to get attention weights:

$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$

$0 < a_{t,i,j} < 1$,
attention values sum to 1

Features:
H x W x D

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$



Attention in CV



Compute alignments scores (scalars):

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$f_{att}(\cdot)$ is an MLP



Extract spatial features from a pretrained CNN

Alignment scores:
 $H \times W$

$e_{1,0,0}$	$e_{1,0,1}$	$e_{1,0,2}$
$e_{1,1,0}$	$e_{1,1,1}$	$e_{1,1,2}$
$e_{1,2,0}$	$e_{1,2,1}$	$e_{1,2,2}$

Attention:
 $H \times W$

$a_{1,0,0}$	$a_{1,0,1}$	$a_{1,0,2}$
$a_{1,1,0}$	$a_{1,1,1}$	$a_{1,1,2}$
$a_{1,2,0}$	$a_{1,2,1}$	$a_{1,2,2}$

Features:
 $H \times W \times D$

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

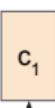
Normalize to get attention weights:

$$a_{t,:,j} = softmax(e_{t,:,j})$$

$0 < a_{t,i,j} < 1$,
attention values sum to 1

Compute context vector:

$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$



Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

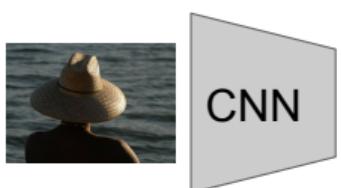


Each timestep of decoder uses a different context vector that looks at different parts of the input image

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$

$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$

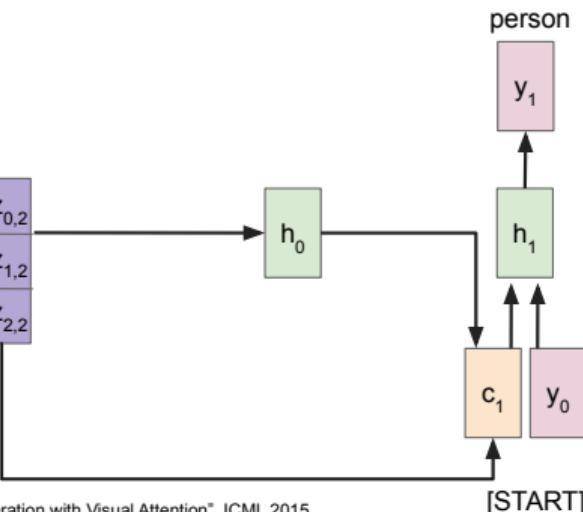


Extract spatial features from a pretrained CNN

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

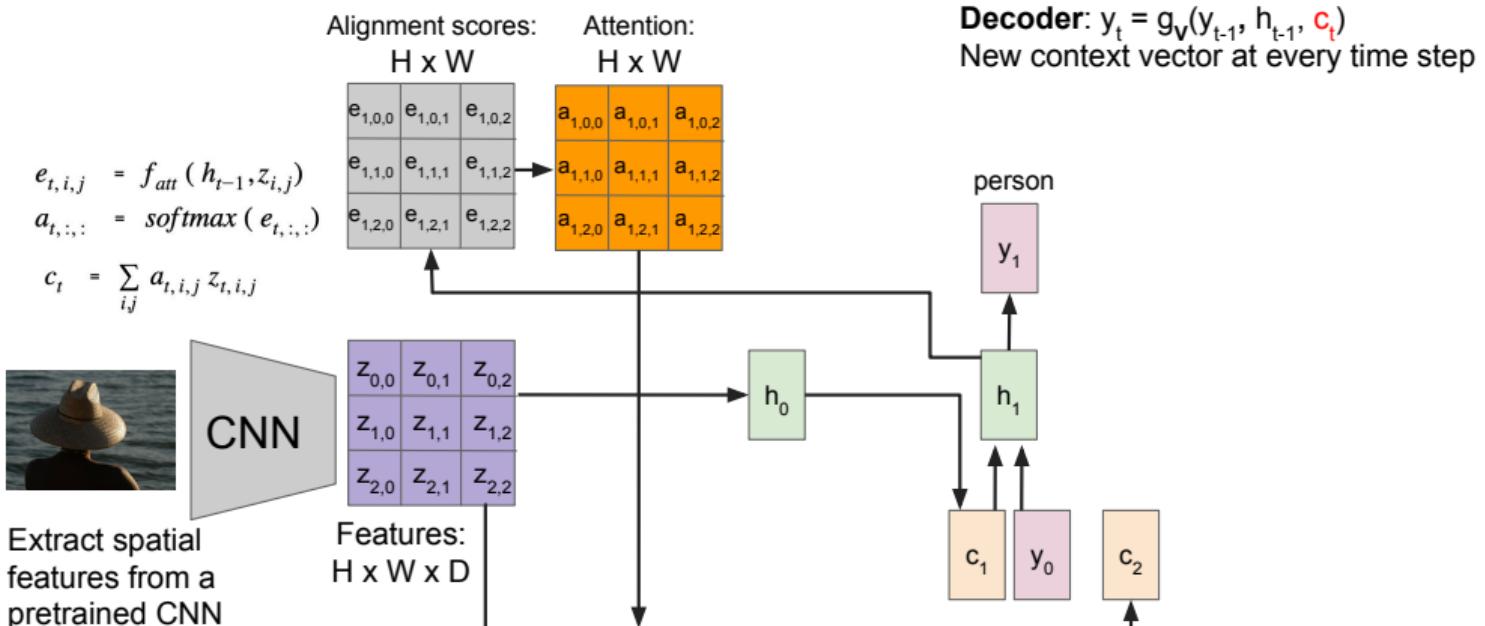
Features:
 $H \times W \times D$

Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step



Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV



Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV

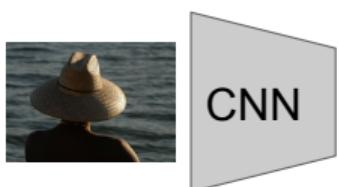


Each timestep of decoder uses a different context vector that looks at different parts of the input image

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$

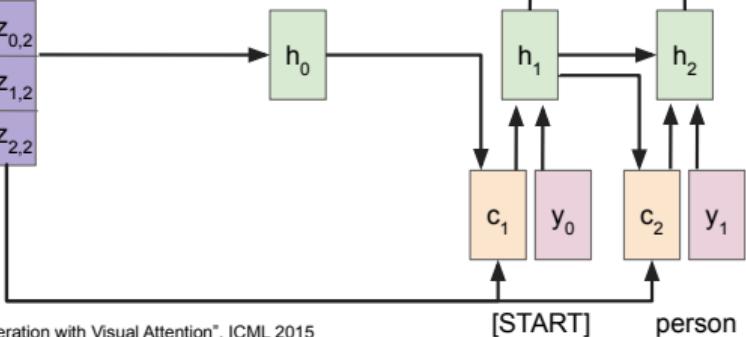
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$



Extract spatial features from a pretrained CNN

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

Features:
 $H \times W \times D$



Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step

Attention in CV

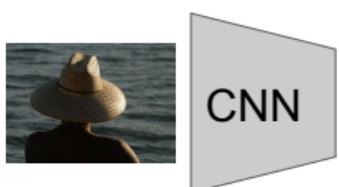


Each timestep of decoder uses a different context vector that looks at different parts of the input image

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$

$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$



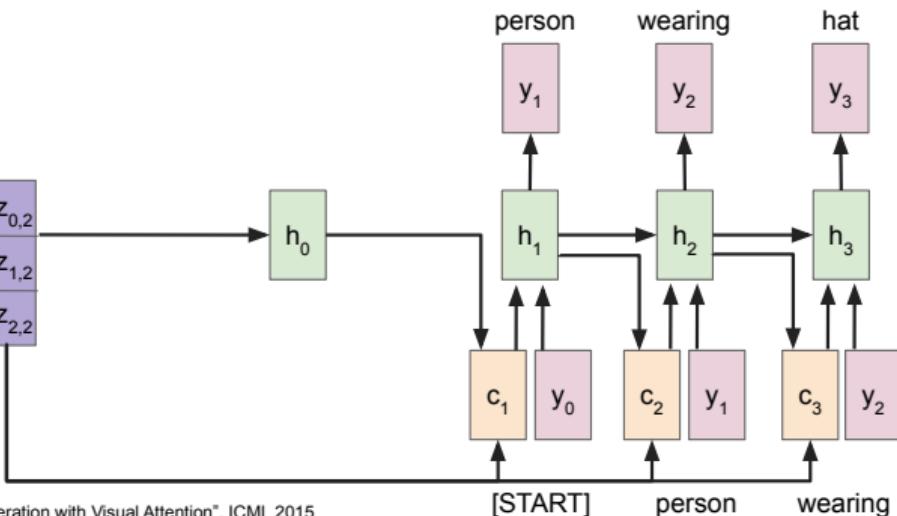
Extract spatial features from a pretrained CNN

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

Features:
 $H \times W \times D$

Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step



Attention in CV

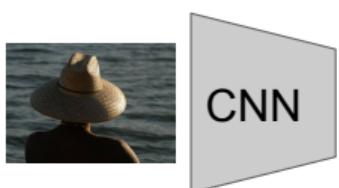


Each timestep of decoder uses a different context vector that looks at different parts of the input image

$$e_{t,i,j} = f_{att}(h_{t-1}, z_{i,j})$$

$$a_{t,:,:} = \text{softmax}(e_{t,:,:})$$

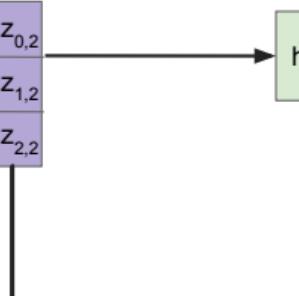
$$c_t = \sum_{i,j} a_{t,i,j} z_{t,i,j}$$



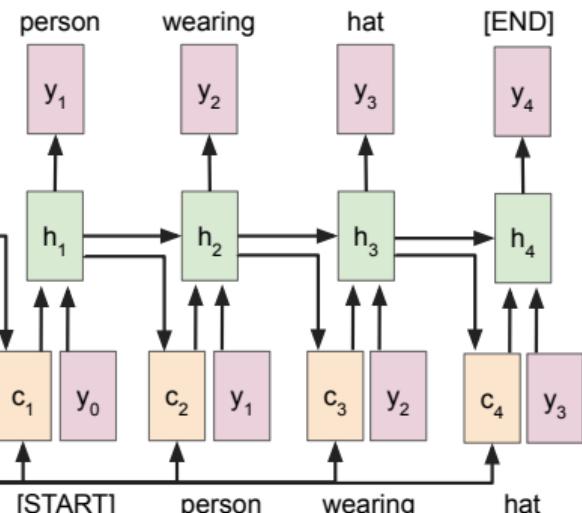
Extract spatial features from a pretrained CNN

$z_{0,0}$	$z_{0,1}$	$z_{0,2}$
$z_{1,0}$	$z_{1,1}$	$z_{1,2}$
$z_{2,0}$	$z_{2,1}$	$z_{2,2}$

Features:
 $H \times W \times D$

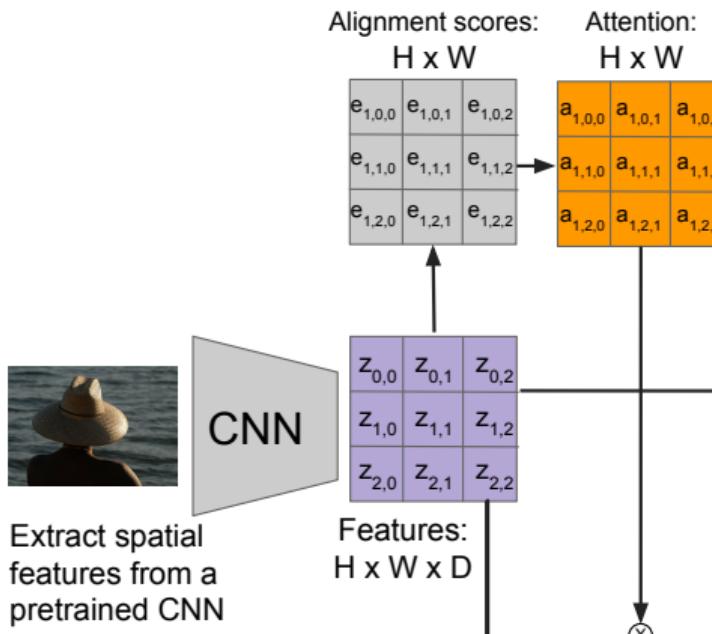


Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c_t)$
New context vector at every time step



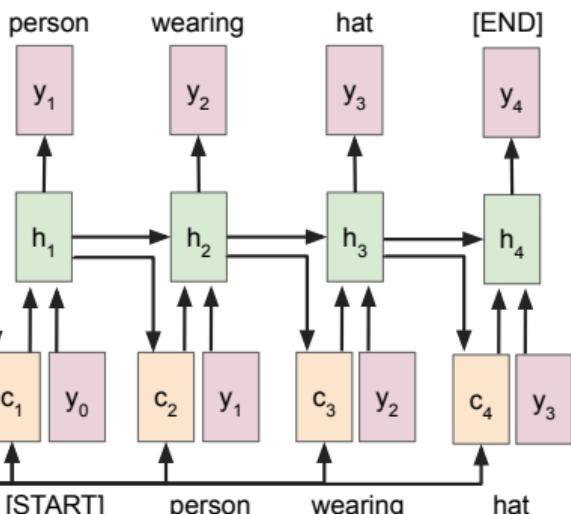
Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Attention in CV



This entire process is differentiable.

- model chooses its own attention weights. No attention supervision is required



Xu et al., "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Transformers



Transformers are based on this attention mechanism. To see how attention works we will check out amazing illustrations by Jay Alammar at
<https://jalammar.github.io/illustrated-transformer/>

Transformers in NLP



Transformers have dominated NLP in recent years:

- Original Transformer Paper - Attention is all you need⁵
- Generative Pre-trained Transformer (GPT)⁶
- BERT⁷

⁵ Ashish Vaswani et al. "Attention is all you need." In: *Advances in neural information processing systems*. 2017, pp. 5998–6008

⁶ Alec Radford et al. "Improving language understanding by generative pre-training." In: (2018)

⁷ Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding." In: *arXiv preprint arXiv:1810.04805* (2018)

Transformers in CV



There have been multiple attempts in recent years to bring transformers to CV.
We will discuss some of the most influential papers.

- DETR - object detection with transformers⁸
- Visual Transformer (ViT)⁹ and DEiT¹⁰
- Pyramid Vision Transformer¹¹
- Swin Transformer¹²

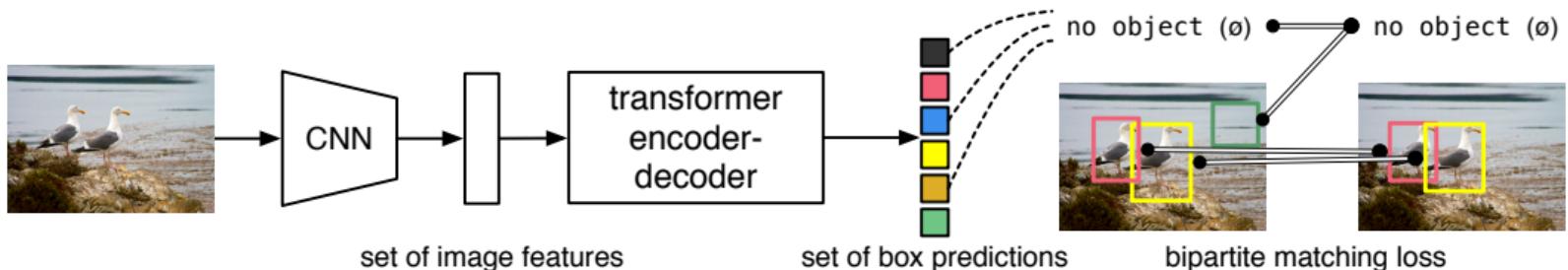
⁸ Nicolas Carion et al. "End-to-end object detection with transformers." In: *European Conference on Computer Vision*. Springer. 2020, pp. 213–229

⁹ Alexey Dosovitskiy et al. "An image is worth 16x16 words: Transformers for image recognition at scale." In: *arXiv preprint arXiv:2010.11929* (2020)

¹⁰ Hugo Touvron et al. "Training data-efficient image transformers & distillation through attention." In: *International Conference on Machine Learning*. PMLR. 2021, pp. 10347–10357

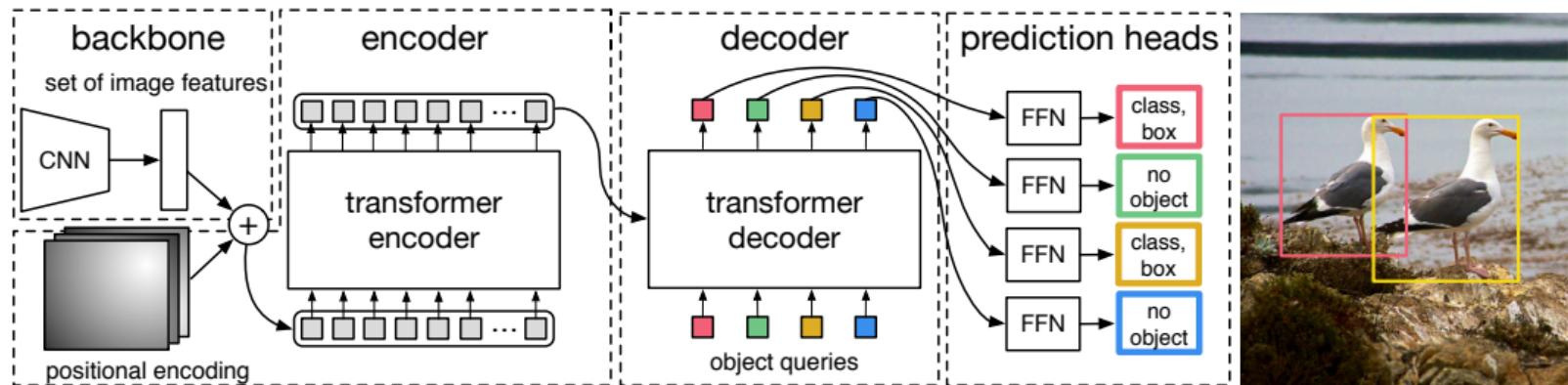
¹¹ Wenhui Wang et al. "Pyramid vision transformer: A versatile backbone for dense prediction without convolutions." In: *arXiv preprint arXiv:2102.12122* (2021)

¹² Ze Liu et al. "Swin transformer: Hierarchical vision transformer using shifted windows." In: *arXiv preprint arXiv:2103.14030* (2021)



DETR directly predicts (in parallel) the final set of detections by combining a common CNN with a transformer architecture. During training, bipartite matching uniquely assigns predictions with ground truth boxes. Prediction with no match should yield a special no-object class prediction.

DETR - Architecture

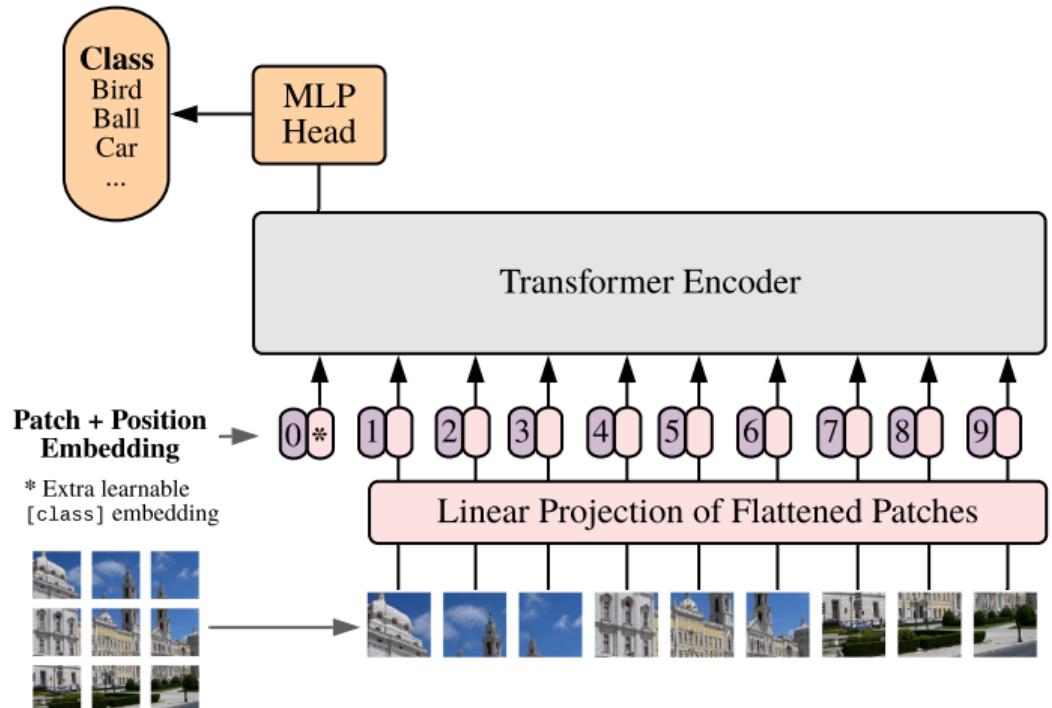


This approach led to competitive results, but authors point out that the training process is tricky. This is an interesting application of transformers, but it still relies on a CNN to extract features.

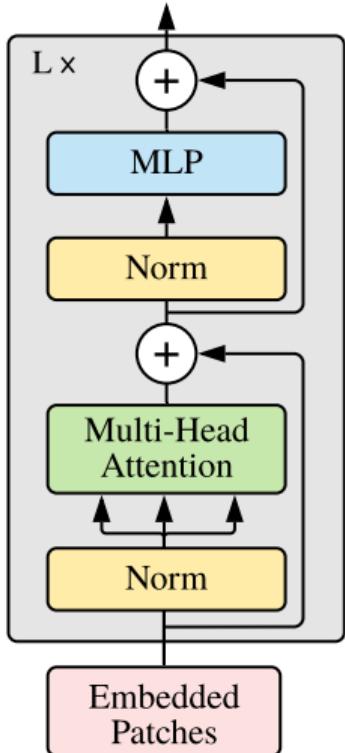
ViT Model



Vision Transformer (ViT)



Transformer Encoder





$$\mathbf{z}_0 = [\mathbf{x}_{\text{class}}; \mathbf{x}_p^1 \mathbf{E}; \mathbf{x}_p^2 \mathbf{E}; \dots; \mathbf{x}_p^N \mathbf{E}] + \mathbf{E}_{pos}, \quad (1)$$

$$\mathbf{z}'_\ell = \text{MSA}(\text{LN}(\mathbf{z}_{\ell-1})) + \mathbf{z}_{\ell-1}, \quad \ell = 1 \dots L \quad (2)$$

$$\mathbf{z}_\ell = \text{MLP}(\text{LN}(\mathbf{z}'_\ell)) + \mathbf{z}'_\ell, \quad \ell = 1 \dots L \quad (3)$$

$$\mathbf{y} = \text{LN}(\mathbf{z}_L^0) \quad (4)$$

$$\mathbf{E} \in \mathbb{R}^{(P^2 \cdot C) \times D}, \mathbf{E}_{pos} \in \mathbb{R}^{(N+1) \times D}$$

ViT - Limitations



There were several issues of ViT which made them very difficult to use:

- Pre-training on huge datasets: ImageNet-21k, JFT-300 (internal Google dataset)
- Training requires a lot of resources
- Results are only reproducible by huge corporations/labs

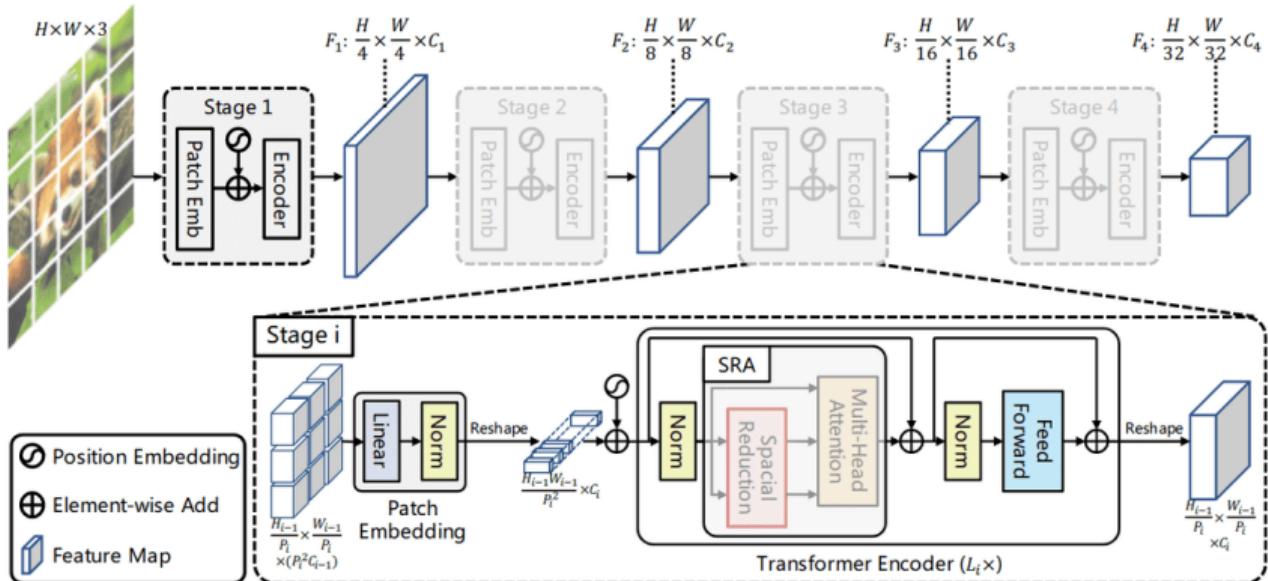


DEiT paper introduced several tricks to train ViT on a reasonable scale. However the results were not that much better than CNNs.

- Self-distillation using a CNN teacher
- CNN features also input into transformer part
- Heavy augmentation + regularization

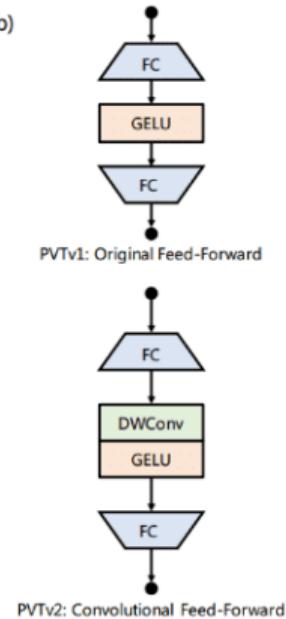
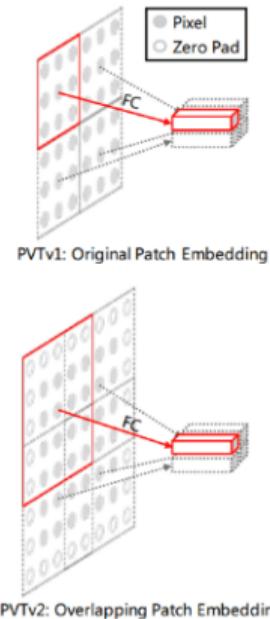
This still requires a pre-trained CNN!

Pyramid Vision Transformers

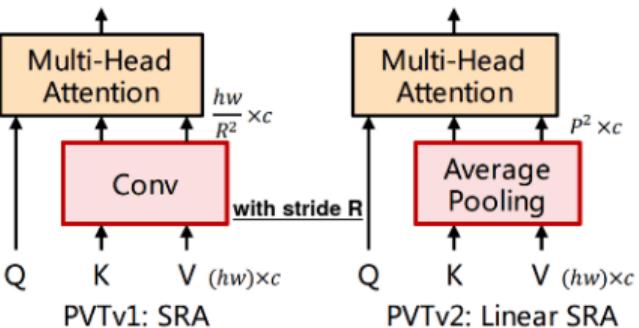


PVTs utilize Spatial-Reduction Attention (SRA) which reduces the dimensionality of both keys and values. This enables the use of transformers in hi-res images for object detection and semantic segmentation.

Pyramid Vision Transformers

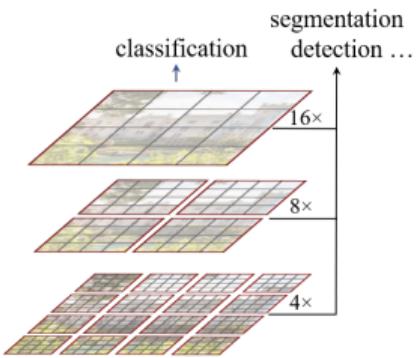


$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{dk}}\right)\mathbf{V}$$

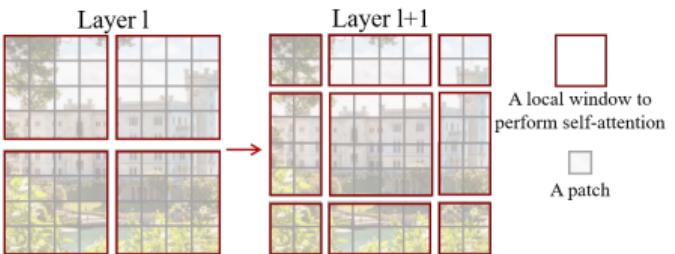


PVT v2 is an improvement with overlapping patch embedding, convolutional feed-forward networks and linear-complexity self-attention layers.

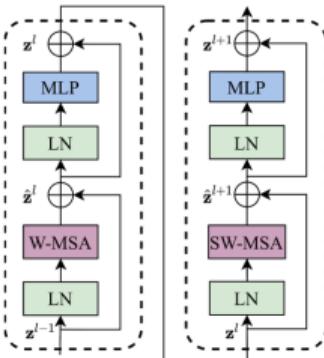
Swin Transformers



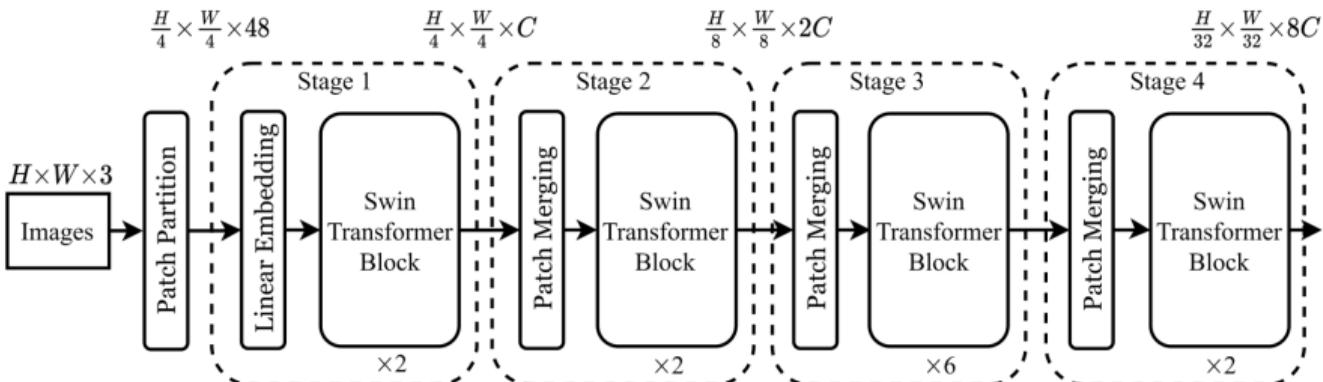
(a) Swin Transformer



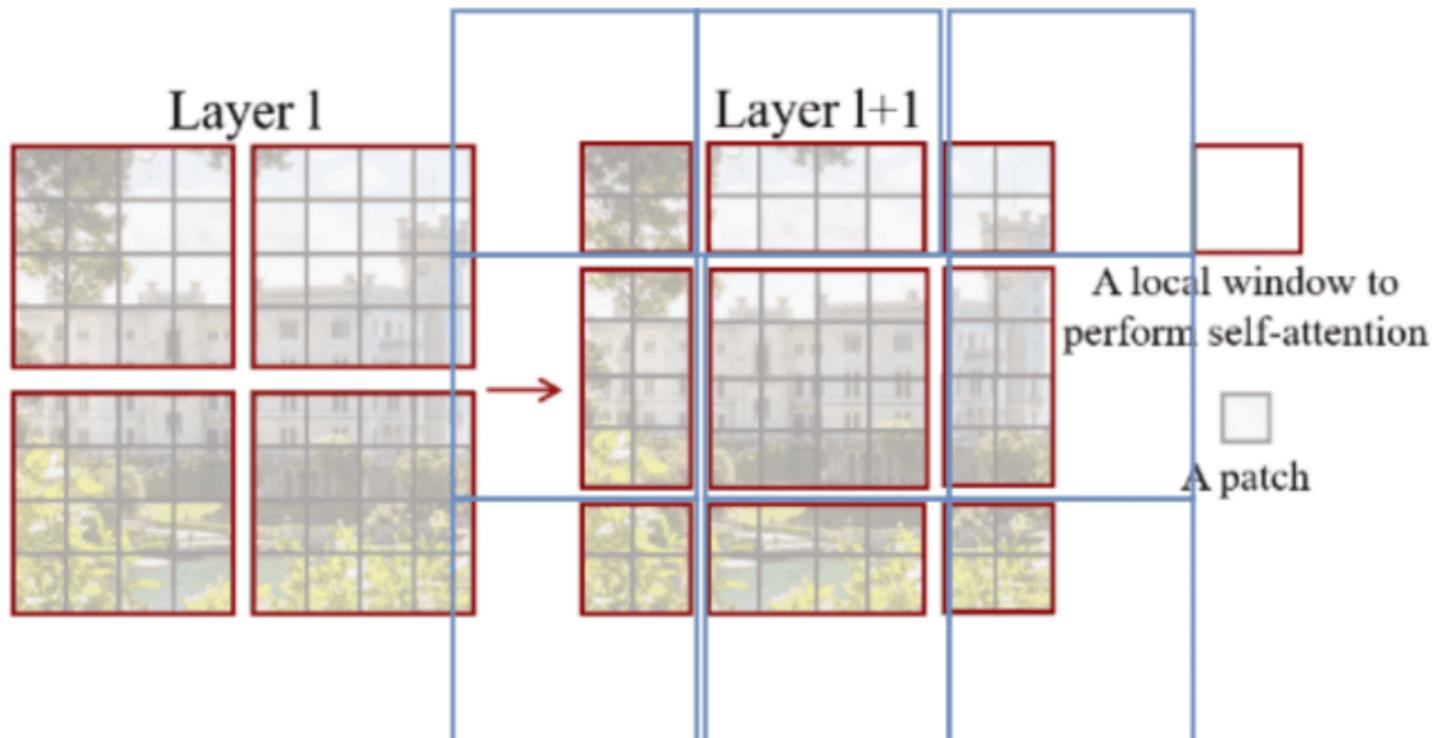
(b) Shifted Window



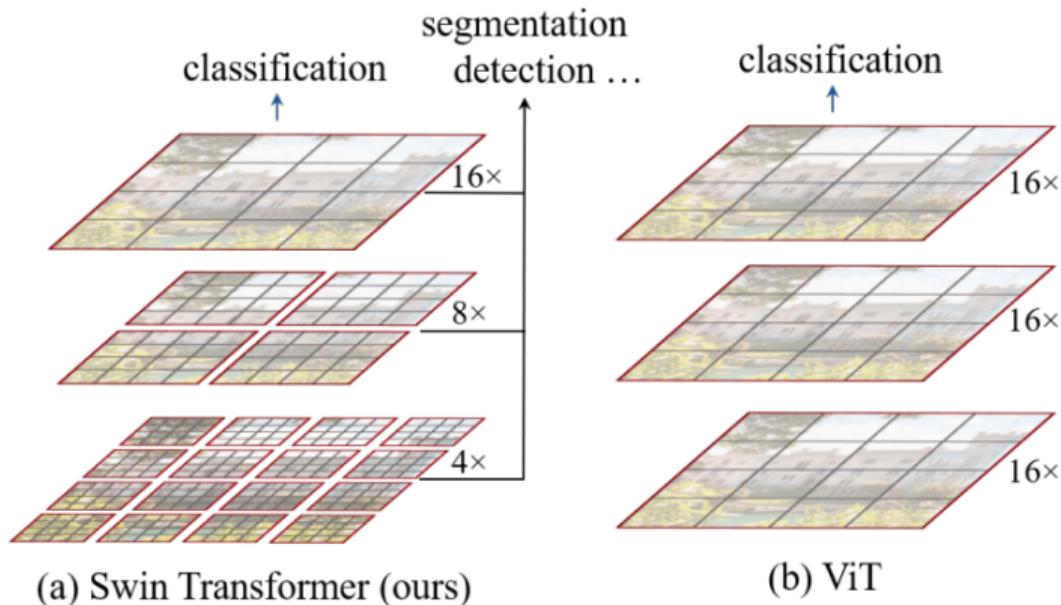
(c) Two Successive Swin Transformer Blocks



Shifted Window Based Self-Attention



Swin vs ViT Hierarchy



Swin introduces hierarchy to computation. The authors also propose to mirror the spatial resolution hierarchies of popular CNNs (e.g. ResNets) so the swin transformers can easily replace other meta-architectures with CNN backbones.

What does the future hold?



The authors of the swin transformer paper claim SOTA on MS COCO object detection and ADE20k object segmentation. They also claim results close to EfficientNets while being faster.

What does the future hold?



The authors of the swin transformer paper claim SOTA on MS COCO object detection and ADE20k object segmentation. They also claim results close to EfficientNets while being faster.

The paper won best paper at ICCV 2021. Since its publishing in summer 2021 on arxiv it has over 300 citations on Google scholar. Is this a serious CV milestone? We will probably see whether transformers took over on CVPR 2022.