

Lecture 9: Object Detection and Image Segmentation

Image Classification: A core task in Computer Vision



(assume given a set of possible labels)
{dog, cat, truck, plane, ...}



cat

This image by [Nikita](#) is
licensed under [CC-BY 2.0](#)

Computer Vision Tasks

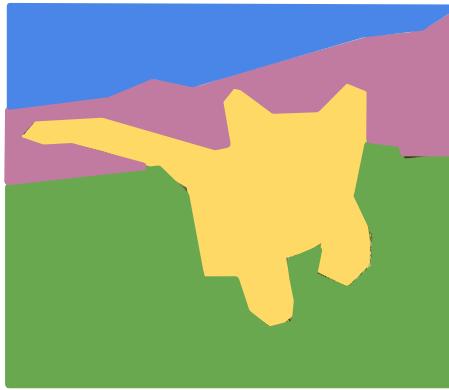
Classification



CAT

No spatial extent

Semantic Segmentation



GRASS, CAT,
TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Multiple Object

Instance Segmentation



DOG, DOG, CAT

This image is CC0 public domain

Semantic Segmentation

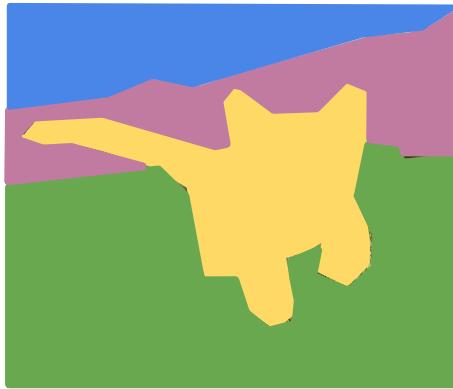
Classification



CAT

No spatial extent

Semantic Segmentation



GRASS, CAT,
TREE, SKY

No objects, just pixels

Object
Detection



DOG, DOG, CAT

Instance
Segmentation



DOG, DOG, CAT

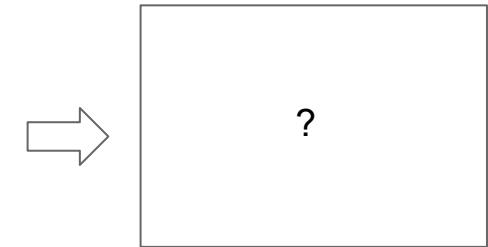
Multiple Object

Semantic Segmentation: The Problem



**GRASS, CAT,
TREE, SKY, ...**

Paired training data: for each training image,
each pixel is labeled with a semantic category.



At test time, classify each pixel of a new image.

Semantic Segmentation Idea: Sliding Window

Full image



Semantic Segmentation Idea: Sliding Window

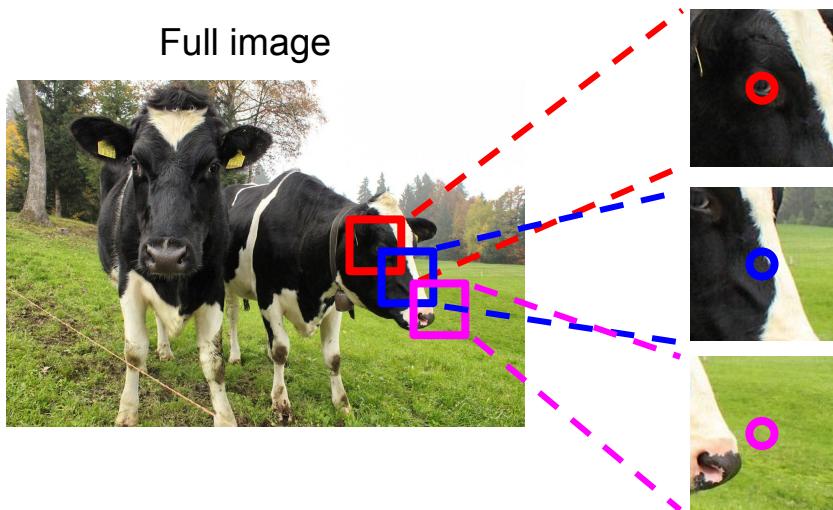


?

Impossible to classify without context

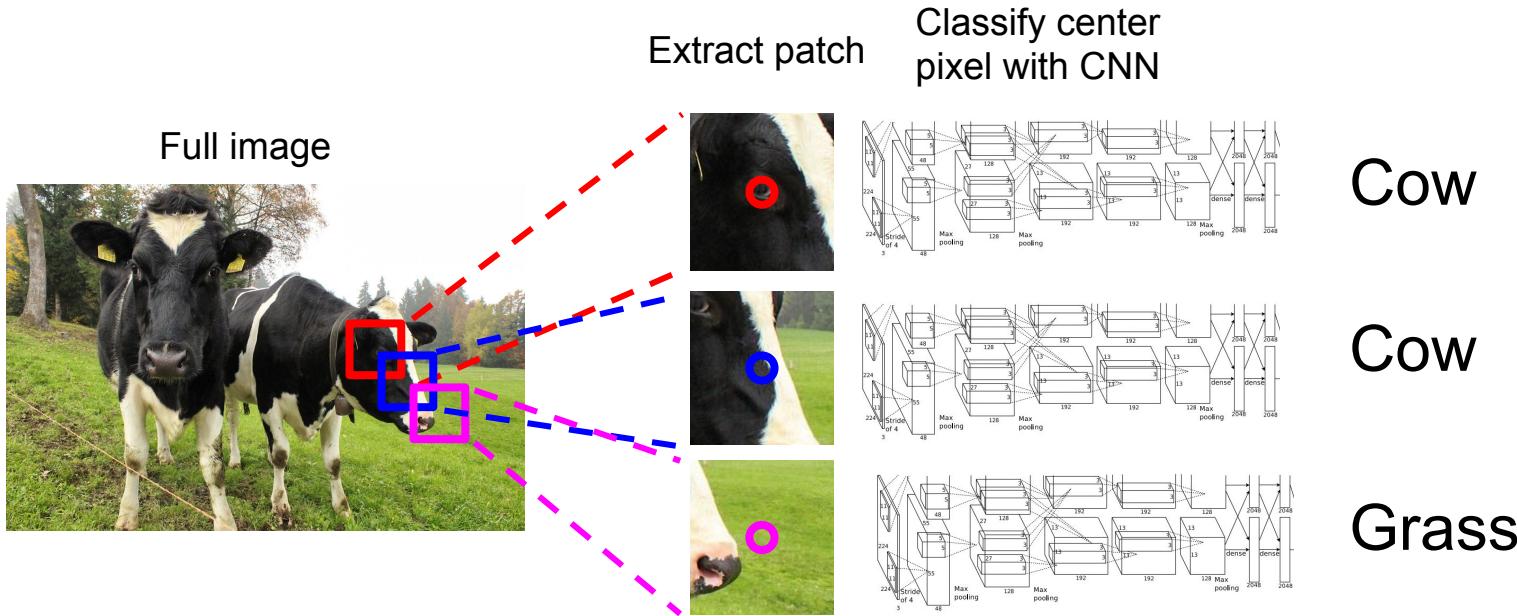
Q: how do we include context?

Semantic Segmentation Idea: Sliding Window



Q: how do we model this?

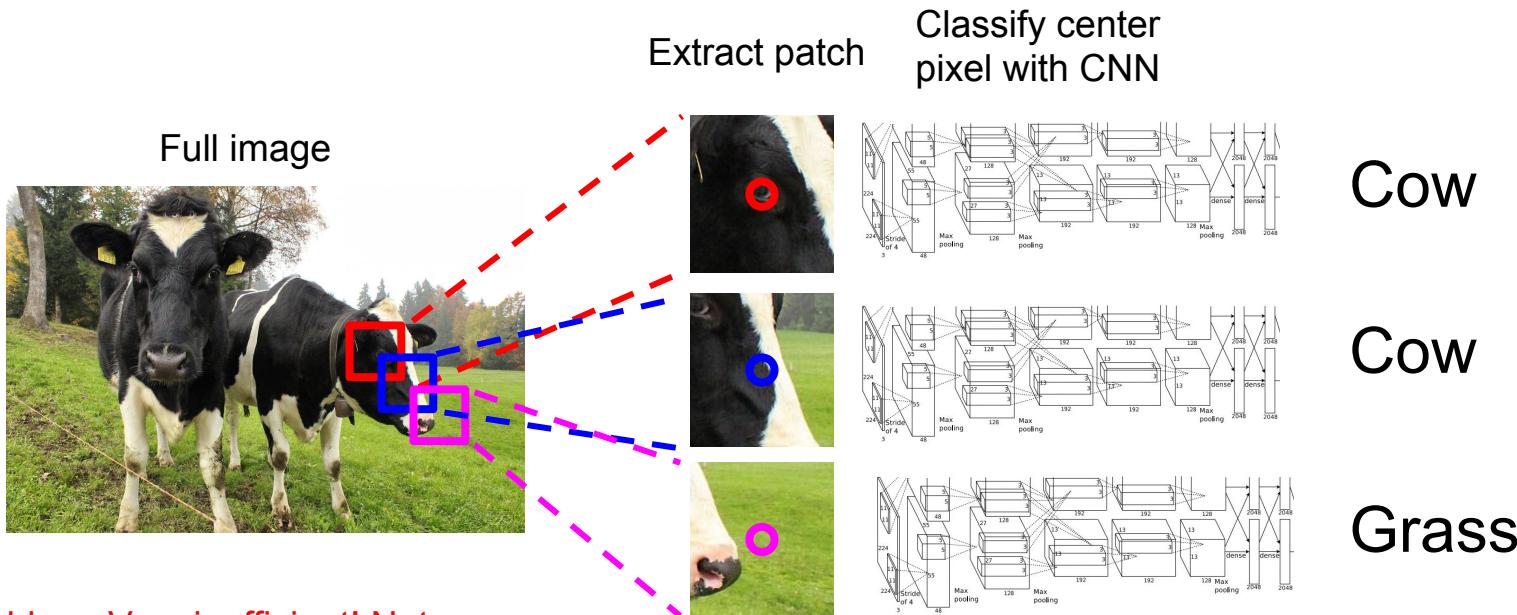
Semantic Segmentation Idea: Sliding Window



Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013

Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation Idea: Sliding Window

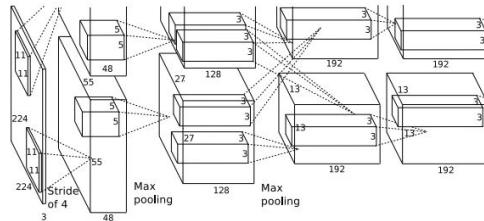


Problem: Very inefficient! Not reusing shared features between overlapping patches

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014

Semantic Segmentation Idea: Convolution

Full image

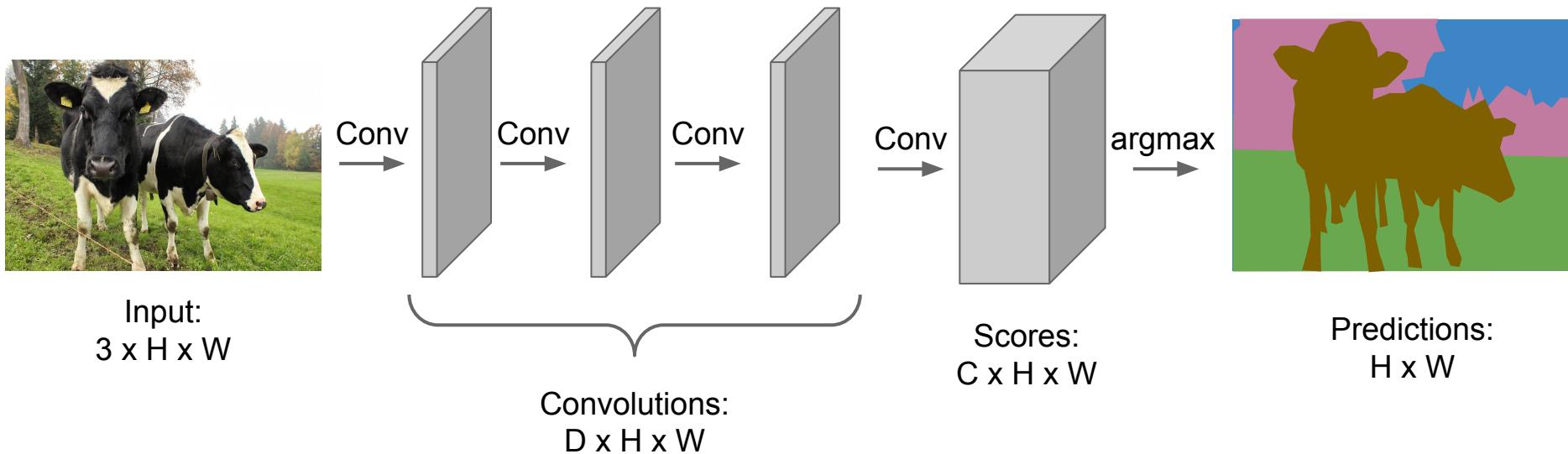


An intuitive idea: encode the entire image with conv net, and do semantic segmentation on top.

Problem: classification architectures often reduce feature spatial sizes to go deeper, but semantic segmentation requires the output size to be the same as input size.

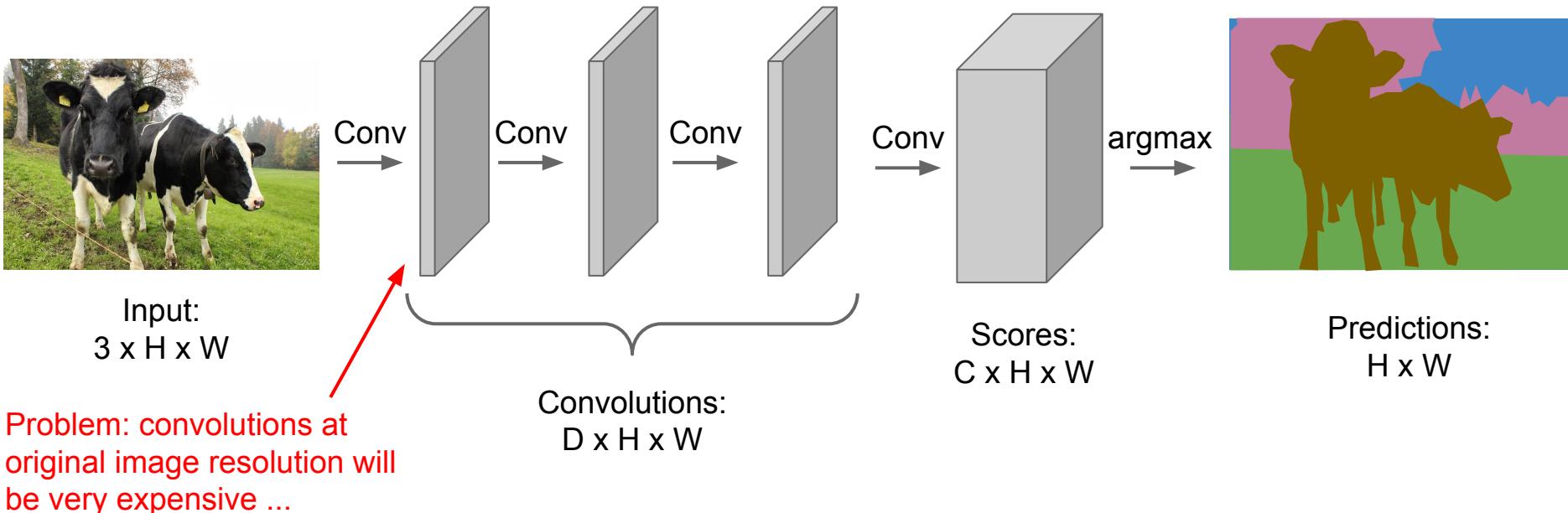
Semantic Segmentation Idea: Fully Convolutional

Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!



Semantic Segmentation Idea: Fully Convolutional

Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!



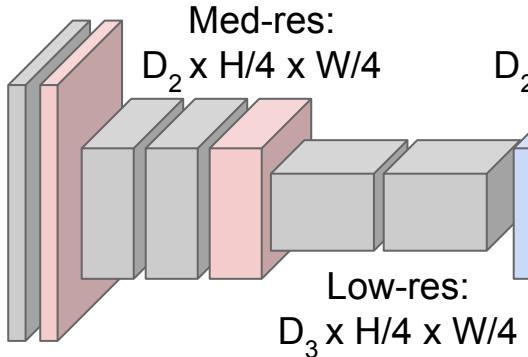
Semantic Segmentation Idea: Fully Convolutional

Design network as a bunch of convolutional layers, with
downsampling and **upsampling** inside the network!

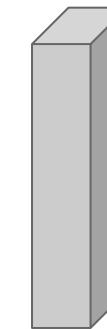


Input:
 $3 \times H \times W$

High-res:
 $D_1 \times H/2 \times W/2$



High-res:
 $D_1 \times H/2 \times W/2$



Predictions:
 $H \times W$



Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

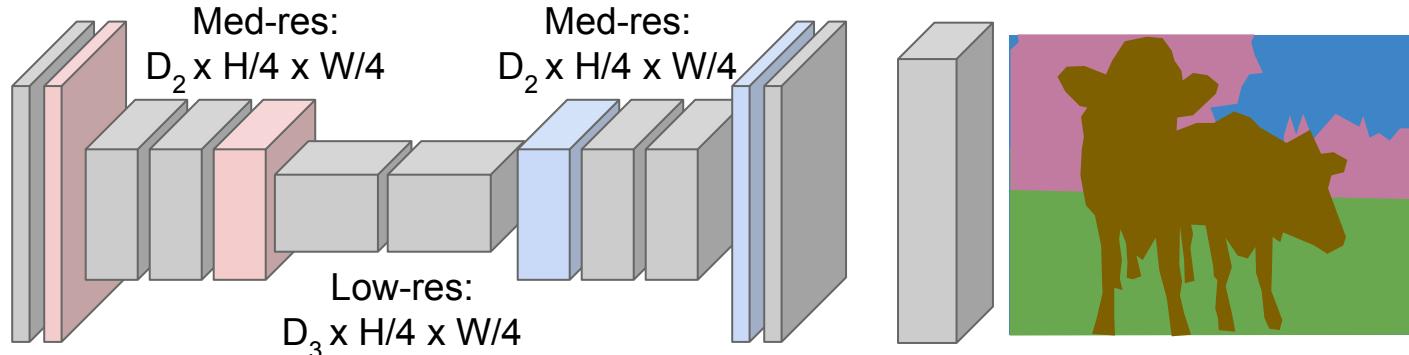
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided
convolution



Input:
 $3 \times H \times W$

Design network as a bunch of convolutional layers, with
downsampling and **upsampling** inside the network!



Upsampling:
???

In-Network upsampling: “Unpooling”

Nearest Neighbor

1	2
3	4



1	1	2	2
1	1	2	2
3	3	4	4
3	3	4	4

Input: 2 x 2

Output: 4 x 4

“Bed of Nails”

1	2
3	4



1	0	2	0
0	0	0	0
3	0	4	0
0	0	0	0

Output: 4 x 4

In-Network upsampling: “Max Unpooling”

Max Pooling

Remember which element was max!

1	2	6	3
3	5	2	1
1	2	2	1
7	3	4	8

Input: 4 x 4

5	6
7	8

Output: 2 x 2

Max Unpooling

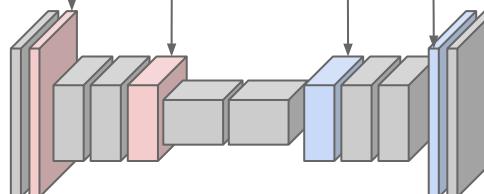
Use positions from pooling layer

1	2
3	4

0	0	2	0
0	1	0	0
0	0	0	0
3	0	0	4

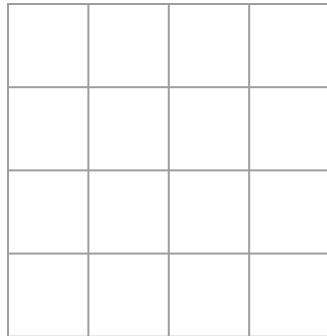
Output: 4 x 4

Corresponding pairs of
downsampling and
upsampling layers

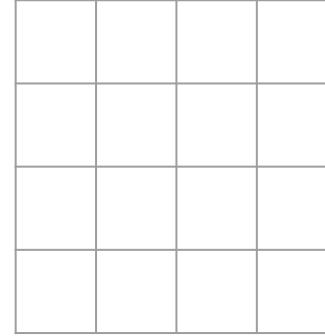


Learnable Upsampling

Recall: Normal 3×3 convolution, stride 1 pad 1



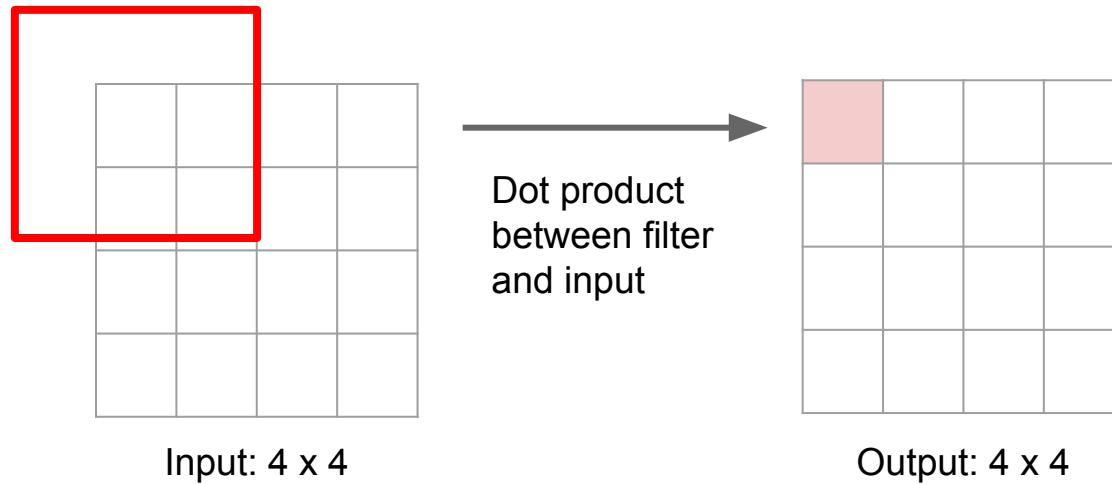
Input: 4×4



Output: 4×4

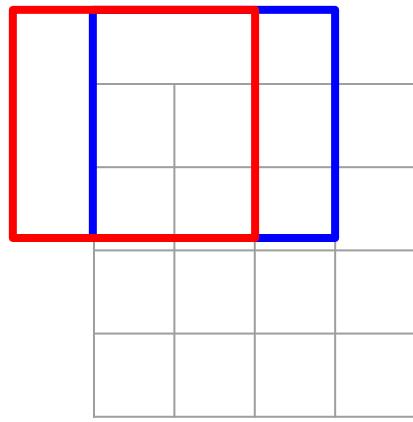
Learnable Upsampling

Recall: Normal 3×3 convolution, stride 1 pad 1



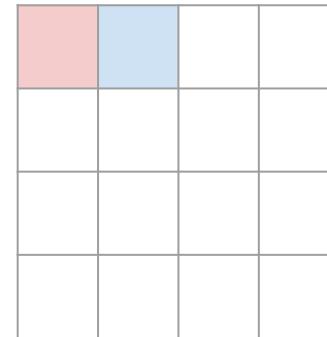
Learnable Upsampling

Recall: Normal 3×3 convolution, stride 1 pad 1



Input: 4×4

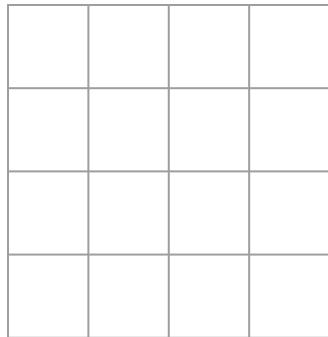
Dot product
between filter
and input



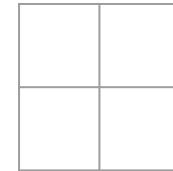
Output: 4×4

Learnable Upsampling

Recall: Normal 3×3 convolution, stride 2 pad 1



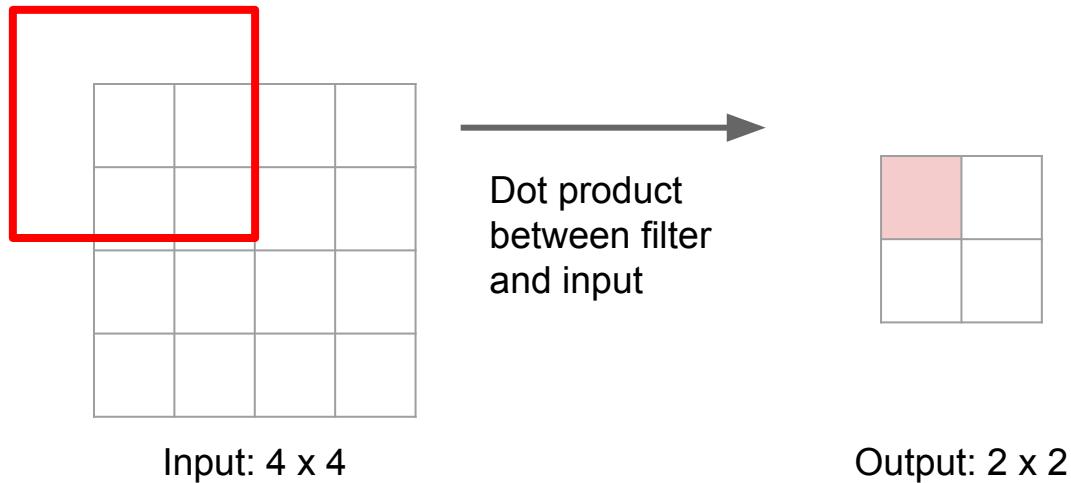
Input: 4×4



Output: 2×2

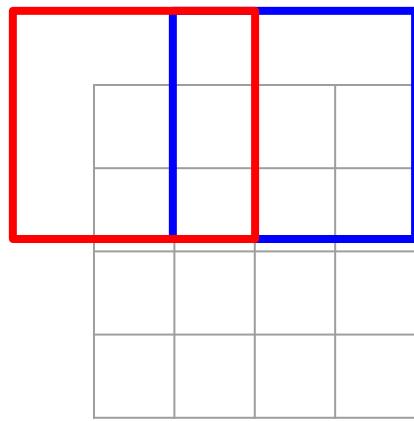
Learnable Upsampling

Recall: Normal 3×3 convolution, stride 2 pad 1



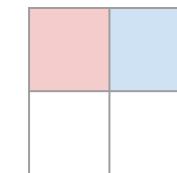
Learnable Upsampling

Recall: Normal 3×3 convolution, stride 2 pad 1



Input: 4×4

Dot product
between filter
and input



Output: 2×2

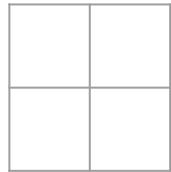
Filter moves 2 pixels in
the input for every one
pixel in the output

Stride gives ratio between
movement in input and
output

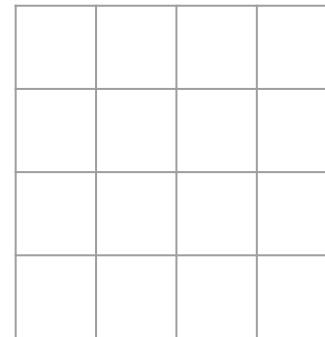
We can interpret strided
convolution as “learnable
downsampling”.

Learnable Upsampling: Transposed Convolution

3×3 **transposed** convolution, stride 2 pad 1



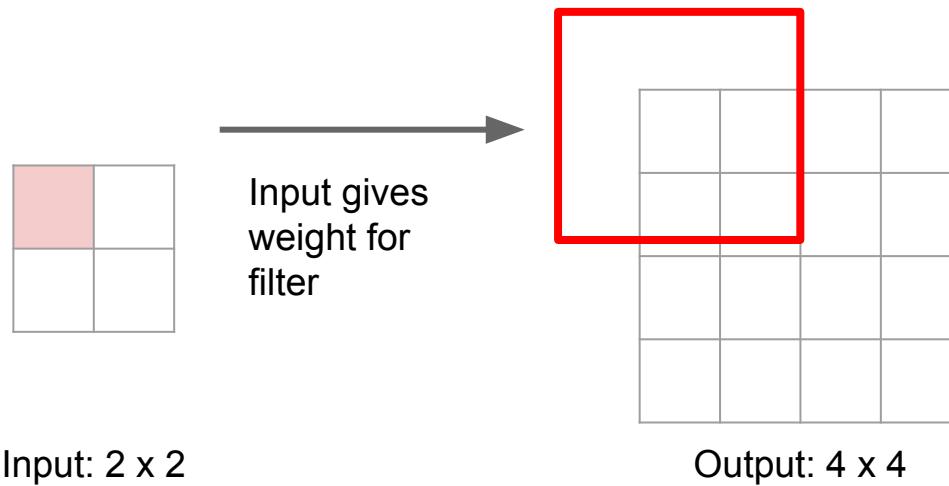
Input: 2×2



Output: 4×4

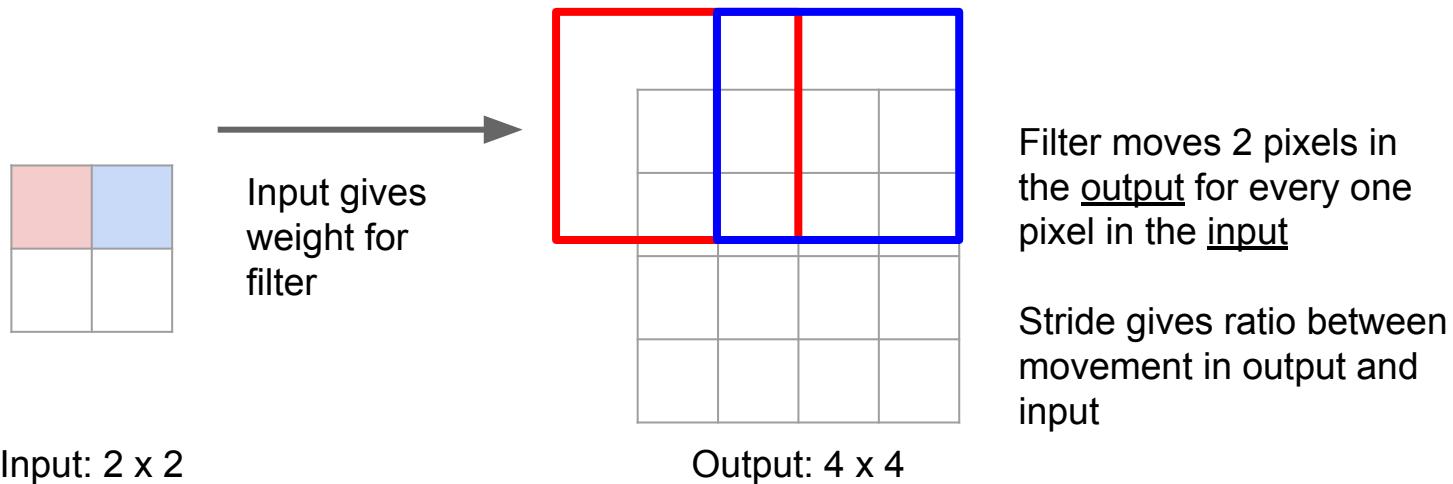
Learnable Upsampling: Transposed Convolution

3 x 3 **transposed** convolution, stride 2 pad 1

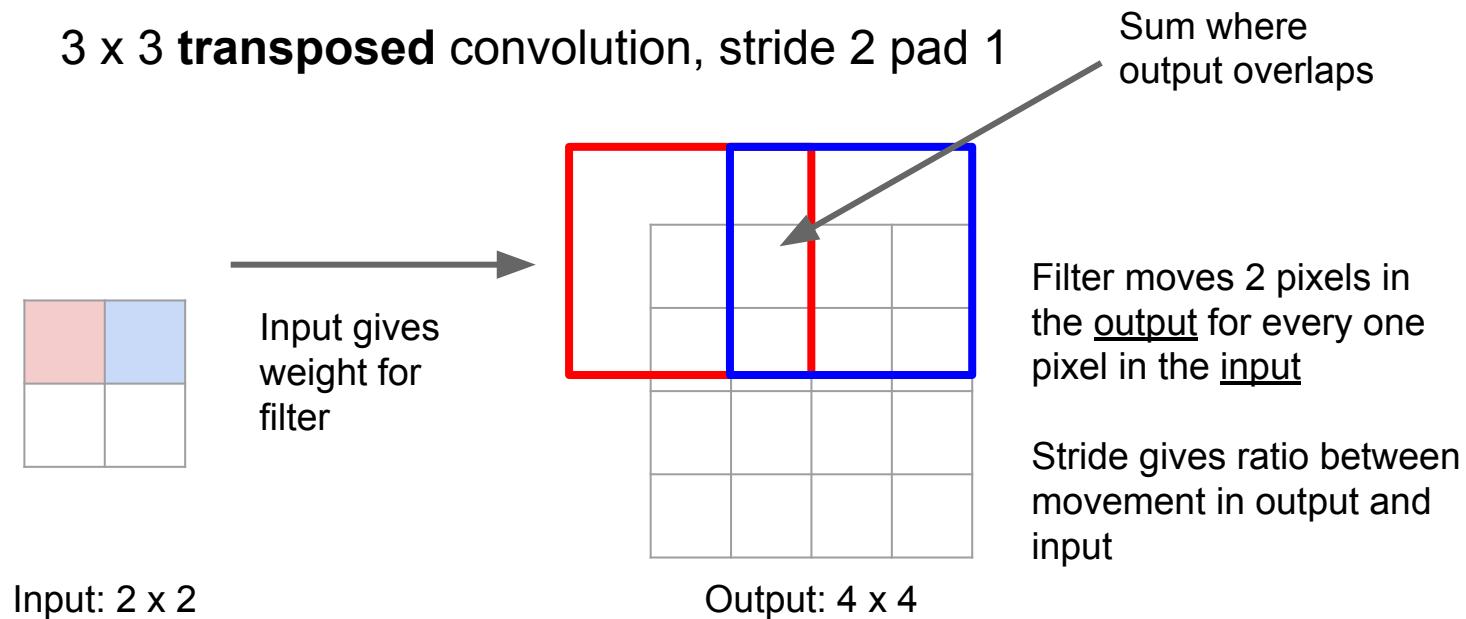


Learnable Upsampling: Transposed Convolution

3 x 3 **transposed** convolution, stride 2 pad 1



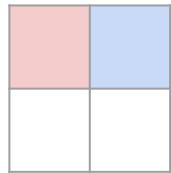
Learnable Upsampling: Transposed Convolution



Learnable Upsampling: Transposed Convolution

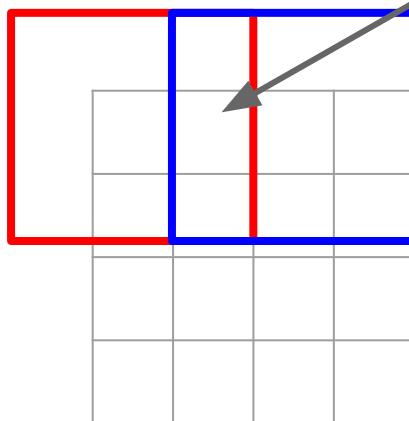
Q: Why is it called transposed convolution?

3 x 3 **transposed** convolution, stride 2 pad 1



Input: 2 x 2

Input gives weight for filter



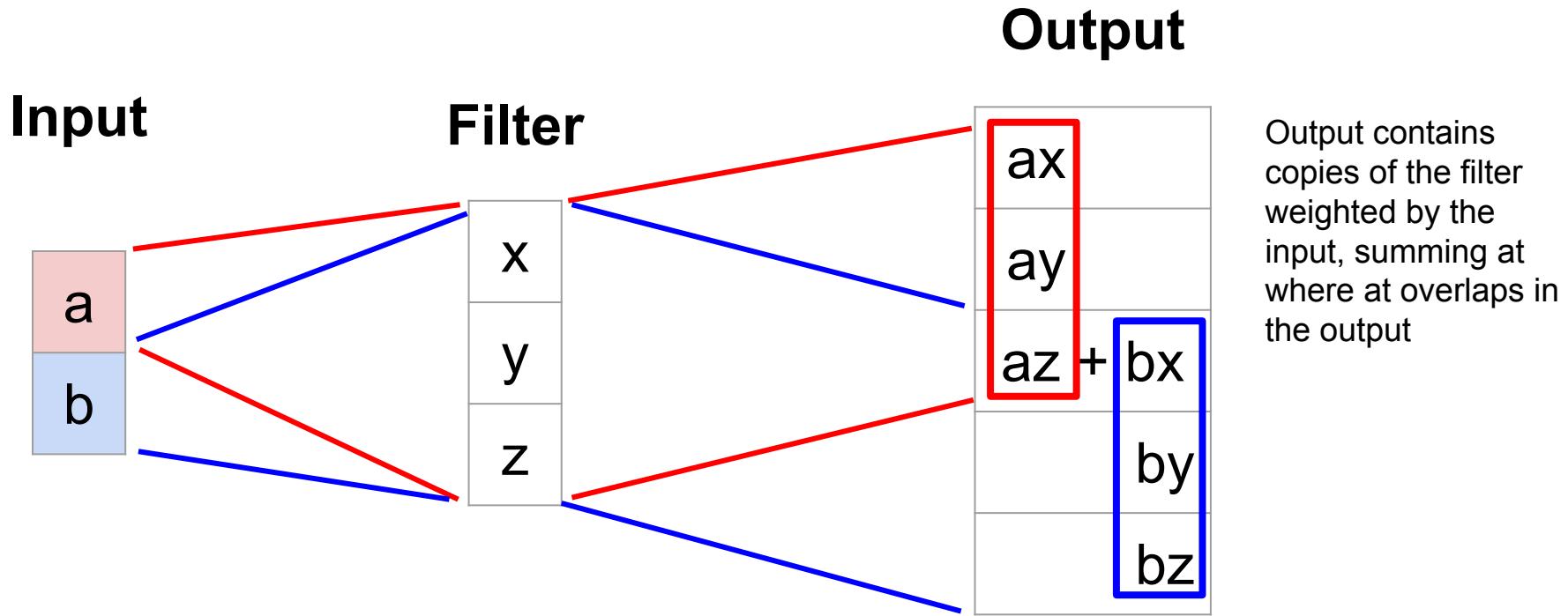
Output: 4 x 4

Sum where output overlaps

Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio between movement in output and input

Learnable Upsampling: 1D Example



Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

$$\vec{x} * \vec{a} = X\vec{a}$$

$$\begin{bmatrix} x & y & z & 0 & 0 & 0 \\ 0 & 0 & x & y & z & 0 \end{bmatrix} \begin{bmatrix} 0 \\ a \\ b \\ c \\ d \\ 0 \end{bmatrix} = \begin{bmatrix} ay + bz \\ bx + cy + dz \end{bmatrix}$$

Example: 1D conv, kernel size=3, stride=2, padding=1

Transposed convolution multiplies by the transpose of the same matrix:

$$\vec{x} *^T \vec{a} = X^T \vec{a}$$

$$\begin{bmatrix} x & 0 \\ y & 0 \\ z & x \\ 0 & y \\ 0 & z \\ 0 & 0 \end{bmatrix} \begin{bmatrix} a \\ b \end{bmatrix} = \begin{bmatrix} ax \\ ay \\ az + bx \\ by \\ bz \\ 0 \end{bmatrix}$$

Example: 1D transposed conv, kernel size=3, stride=2, padding=0

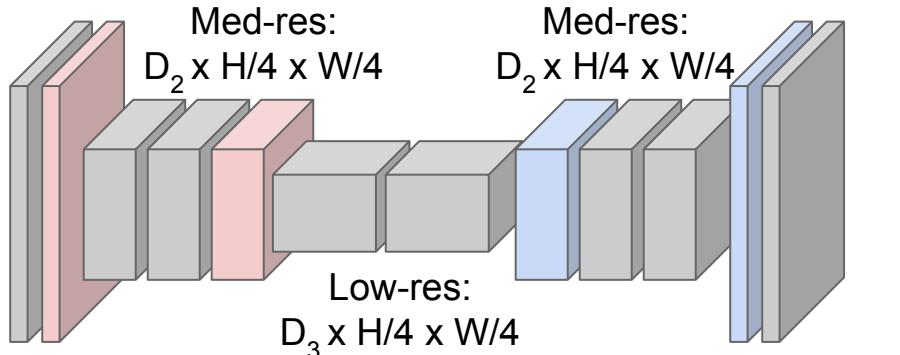
Semantic Segmentation Idea: Fully Convolutional

Downsampling:
Pooling, strided convolution



Input:
 $3 \times H \times W$

Design network as a bunch of convolutional layers, with **downsampling** and **upsampling** inside the network!



High-res:
 $D_1 \times H/2 \times W/2$

High-res:
 $D_1 \times H/2 \times W/2$

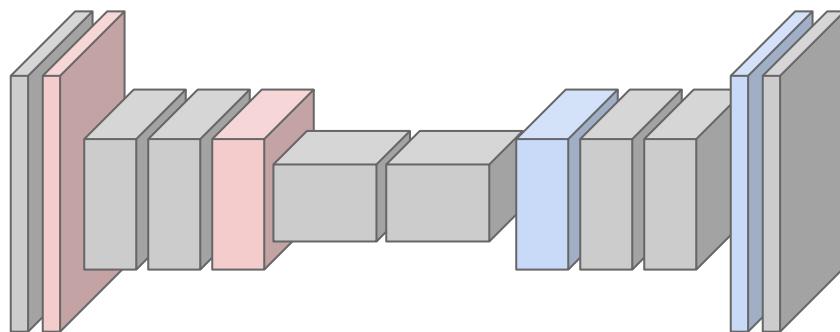
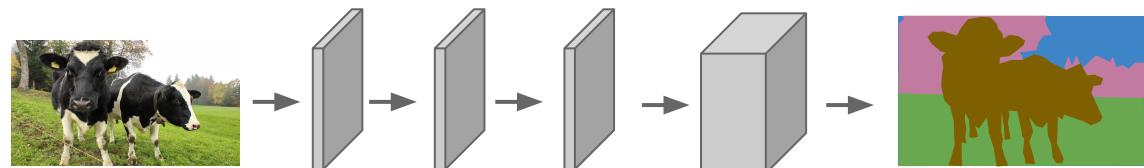
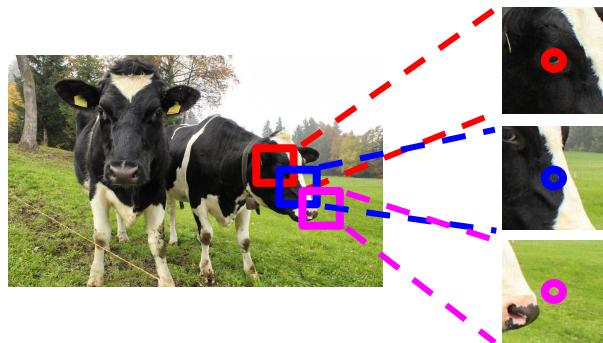
Upsampling:
Unpooling or strided transposed convolution



Predictions:
 $H \times W$

Long, Shelhamer, and Darrell, "Fully Convolutional Networks for Semantic Segmentation", CVPR 2015
Noh et al, "Learning Deconvolution Network for Semantic Segmentation", ICCV 2015

Semantic Segmentation: Summary



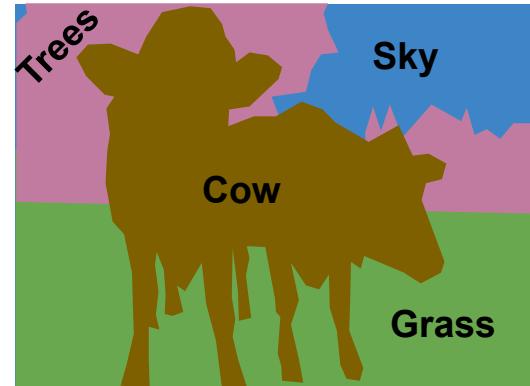
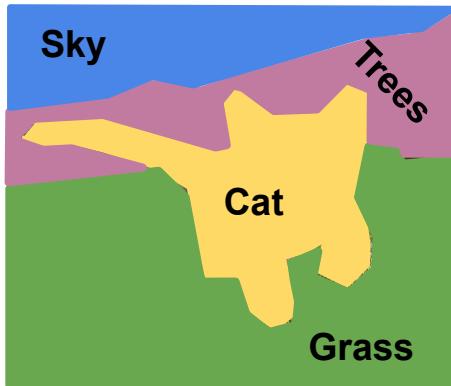
Semantic Segmentation

Label each pixel in the image with a category label

Don't differentiate instances, only care about pixels



[This image is CC0 public domain](#)



Object Detection

Classification



CAT

No spatial extent

Semantic Segmentation



GRASS, CAT,
TREE, SKY

No objects, just pixels

Object
Detection



DOG, DOG, CAT

Multiple Object

Instance
Segmentation



DOG, DOG, CAT

Object Detection

Classification



CAT

No spatial extent

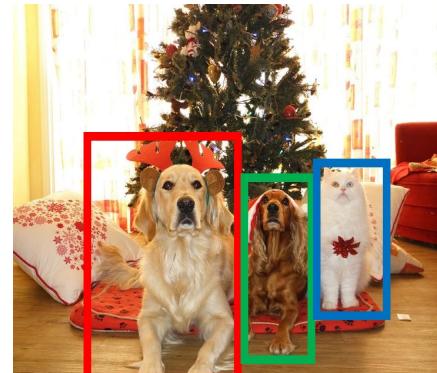
Semantic Segmentation



GRASS, CAT,
TREE, SKY

No objects, just pixels

Object
Detection



DOG, DOG, CAT

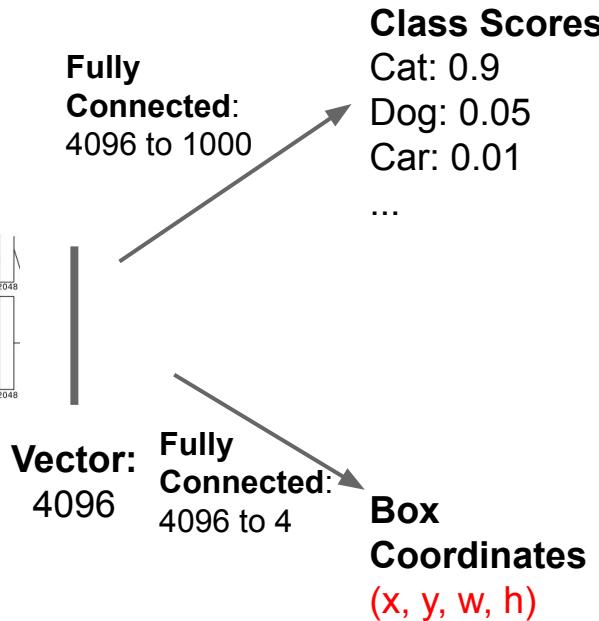
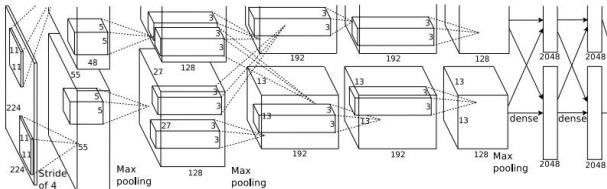
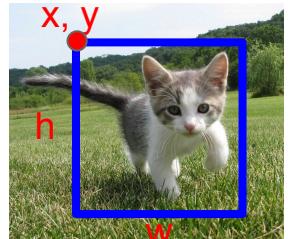
Multiple Object

Instance
Segmentation

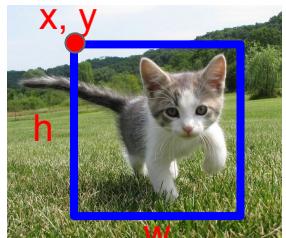


DOG, DOG, CAT

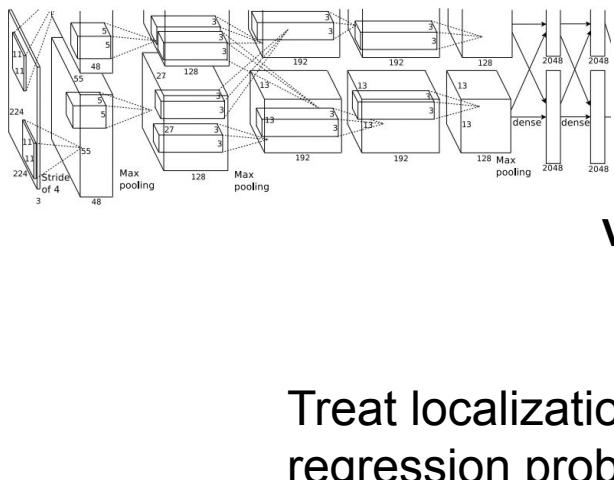
Object Detection: Single Object (Classification + Localization)



Object Detection: Single Object (Classification + Localization)



This image is CC0 public domain



Treat localization as a
regression problem!

Vector:
4096 **Fully Connected:**
4096 to 4

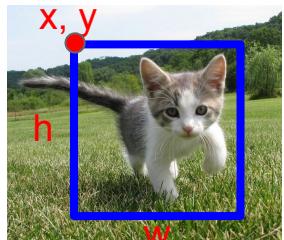
Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Box Coordinates → L2 Loss
(x, y, w, h)
Correct box:
(x', y', w', h')

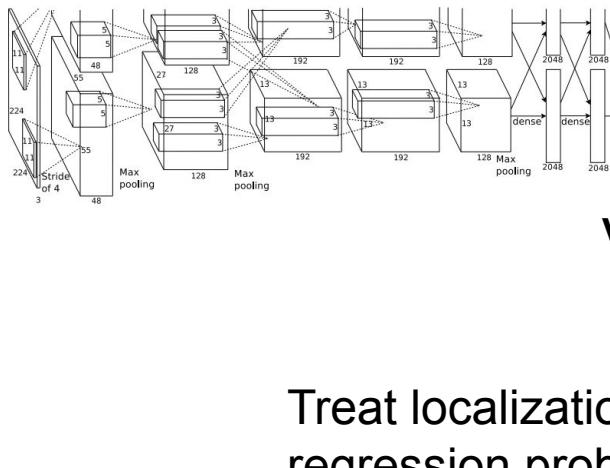
Correct label:
Cat

Softmax
Loss

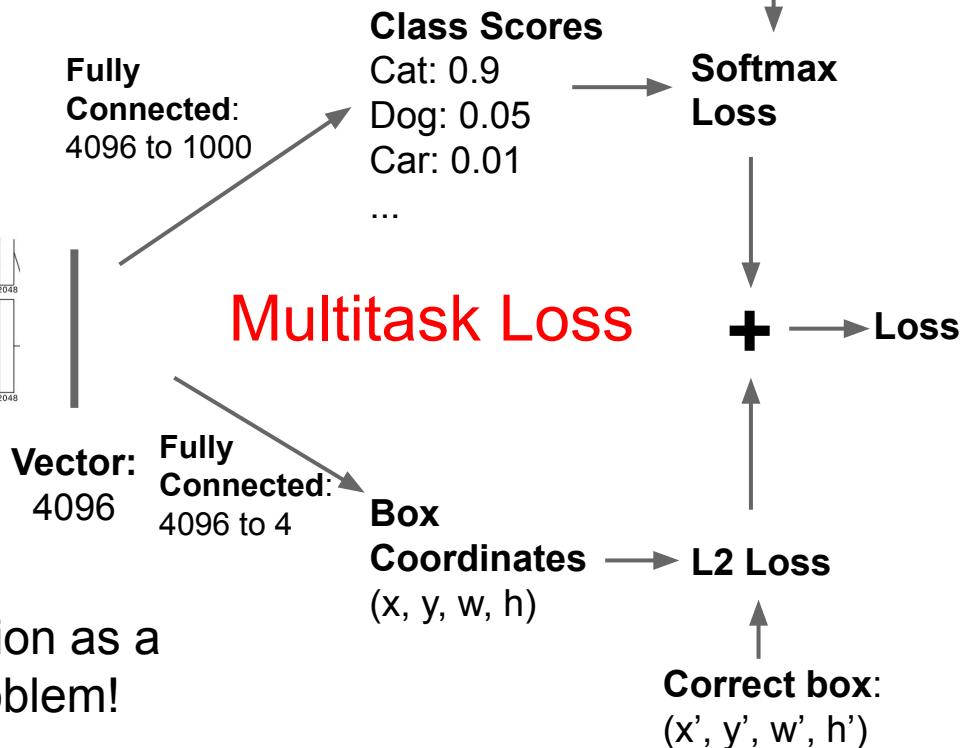
Object Detection: Single Object (Classification + Localization)



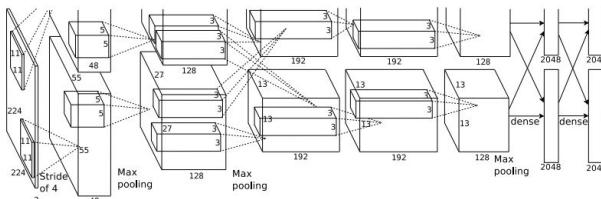
This image is CC0 public domain



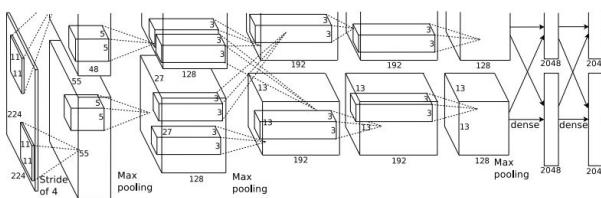
Treat localization as a
regression problem!



Object Detection: Multiple Objects



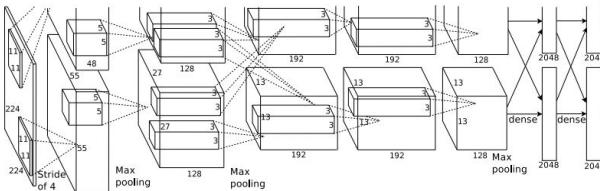
CAT: (x, y, w, h)



DOG: (x, y, w, h)

DOG: (x, y, w, h)

CAT: (x, y, w, h)



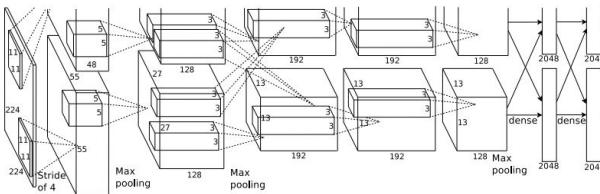
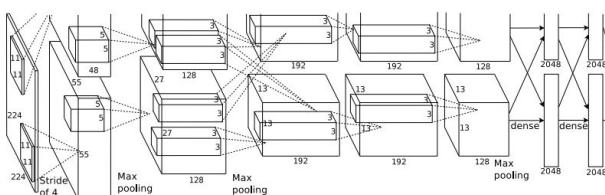
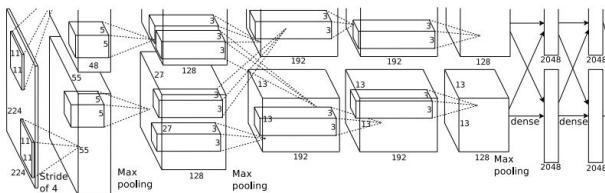
DUCK: (x, y, w, h)

DUCK: (x, y, w, h)

...

Object Detection: Multiple Objects

Each image needs a different number of outputs!



CAT: (x, y, w, h)

4 numbers

DOG: (x, y, w, h)

12 numbers

CAT: (x, y, w, h)

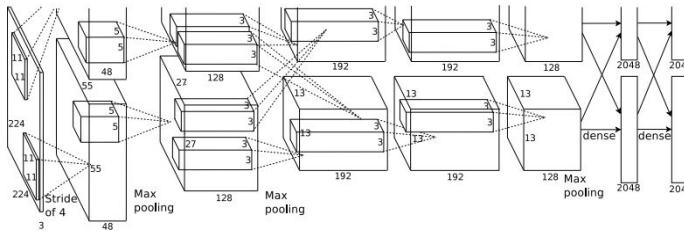
DUCK: (x, y, w, h)

Many numbers!

DUCK: (x, y, w, h)

Object Detection: Multiple Objects

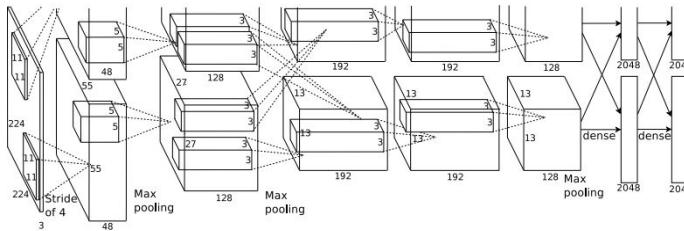
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? NO
Cat? NO
Background? YES

Object Detection: Multiple Objects

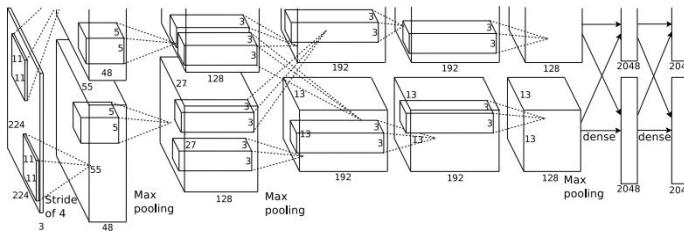
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

Object Detection: Multiple Objects

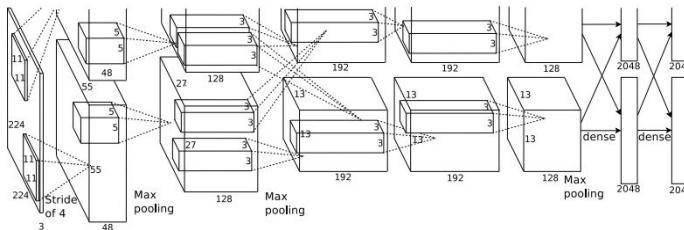
Apply a CNN to many different crops of the image, CNN classifies each crop as object or background



Dog? YES
Cat? NO
Background? NO

Object Detection: Multiple Objects

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

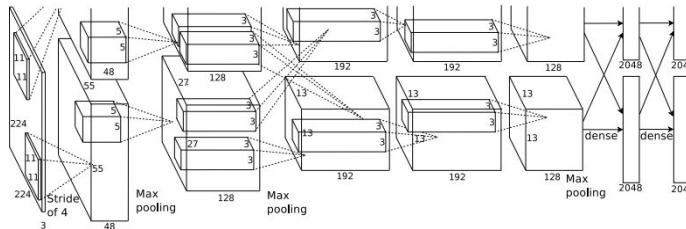
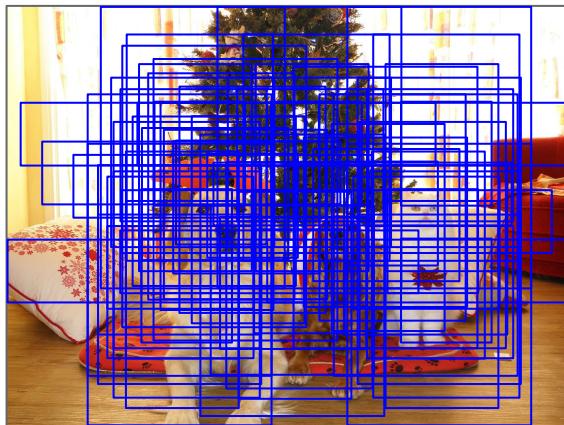


Dog? NO
Cat? YES
Background? NO

Q: What's the problem with this approach?

Object Detection: Multiple Objects

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

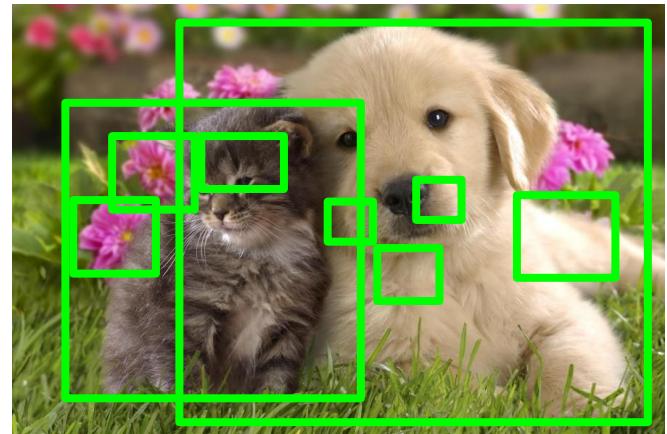


Dog? NO
Cat? YES
Background? NO

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!

Region Proposals: Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU



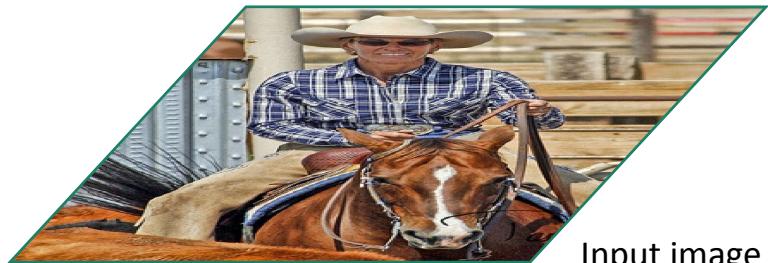
Alexe et al, "Measuring the objectness of image windows", TPAMI 2012

Uijlings et al, "Selective Search for Object Recognition", IJCV 2013

Cheng et al, "BING: Binarized normed gradients for objectness estimation at 300fps", CVPR 2014

Zitnick and Dollar, "Edge boxes: Locating object proposals from edges", ECCV 2014

R-CNN



Input image

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

R-CNN

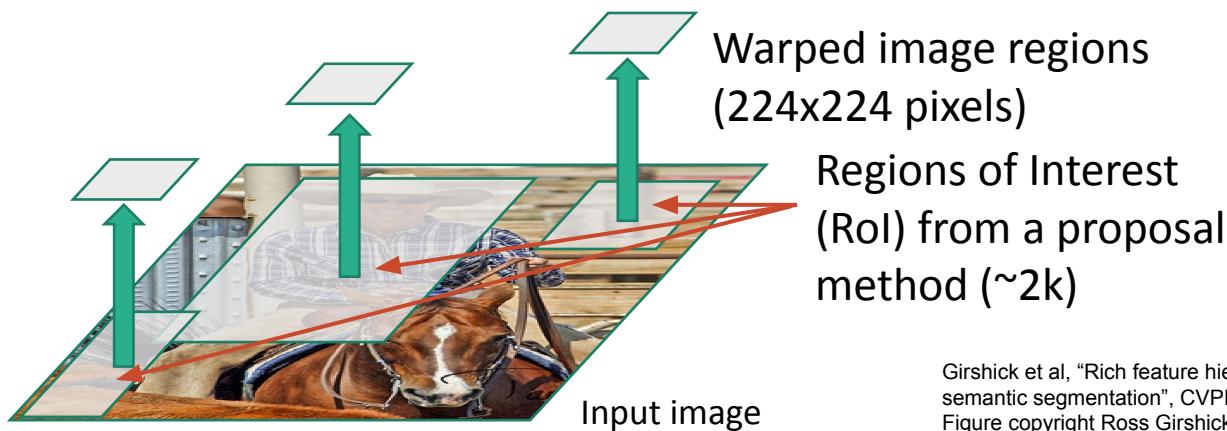


Input image

Regions of Interest
(RoI) from a proposal
method (~2k)

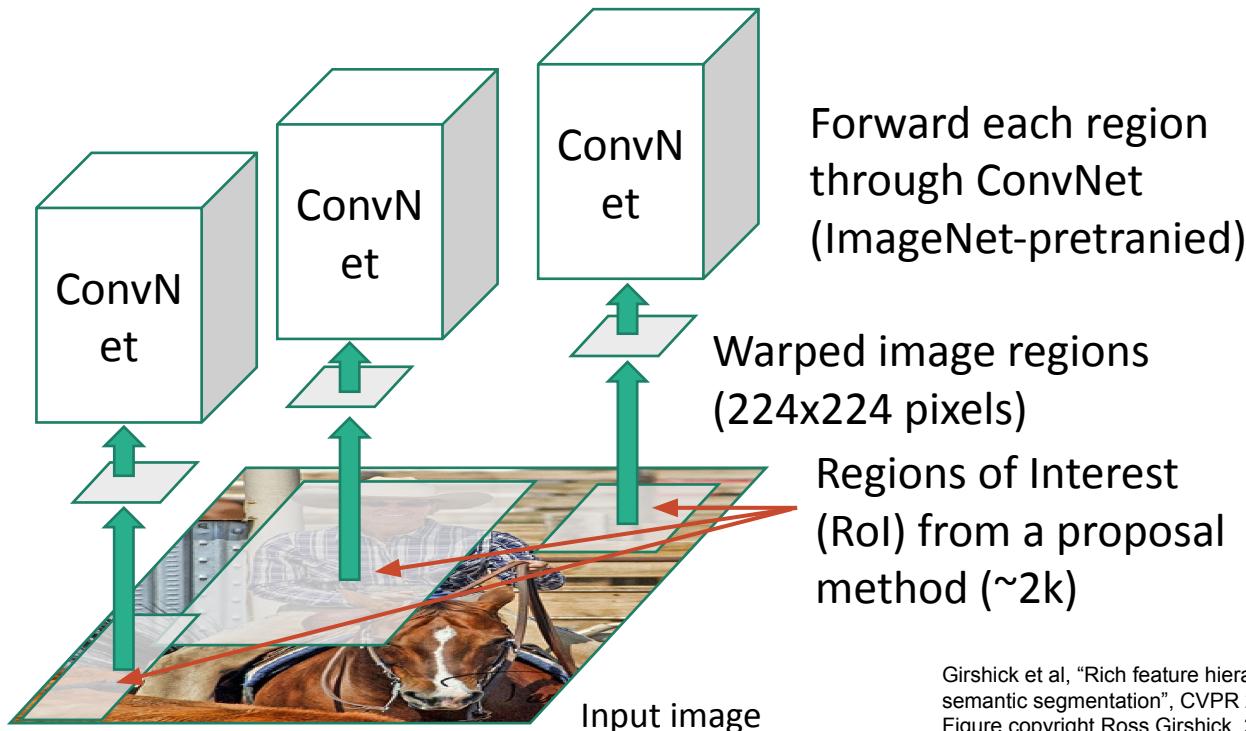
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

R-CNN



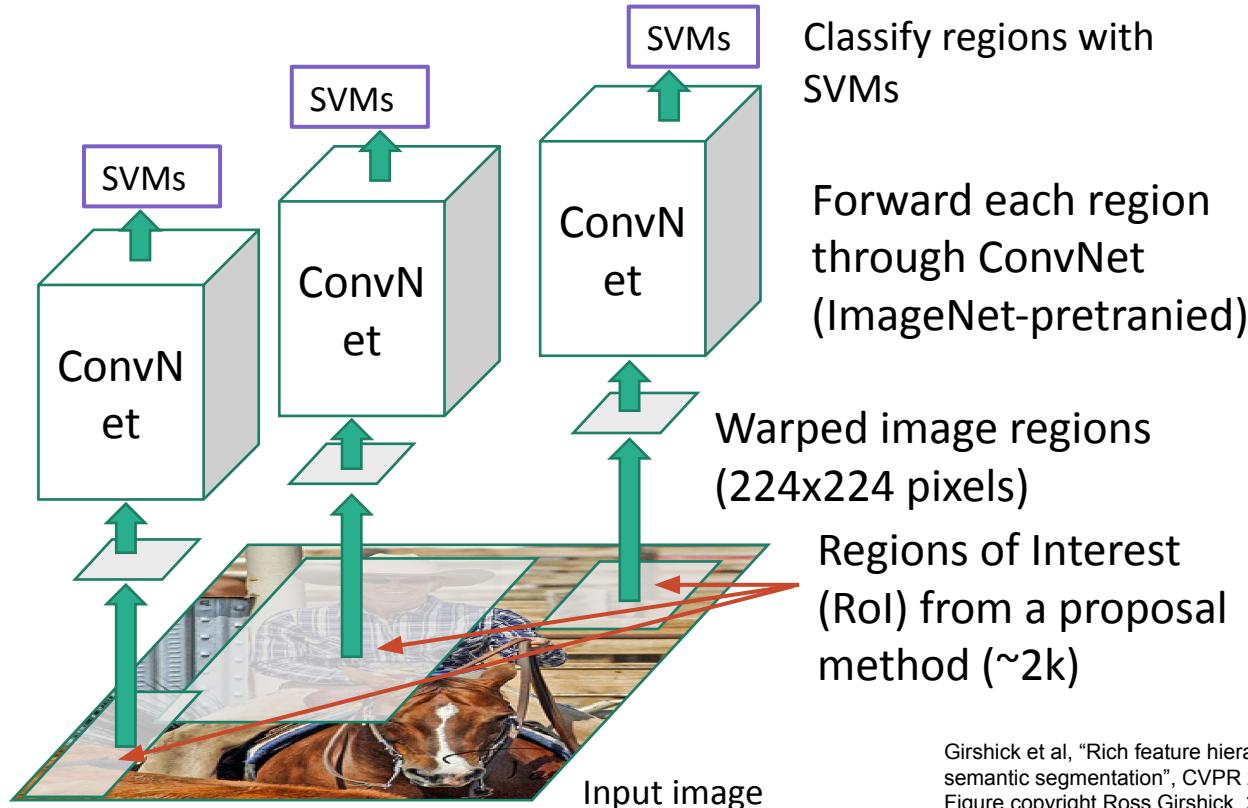
Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

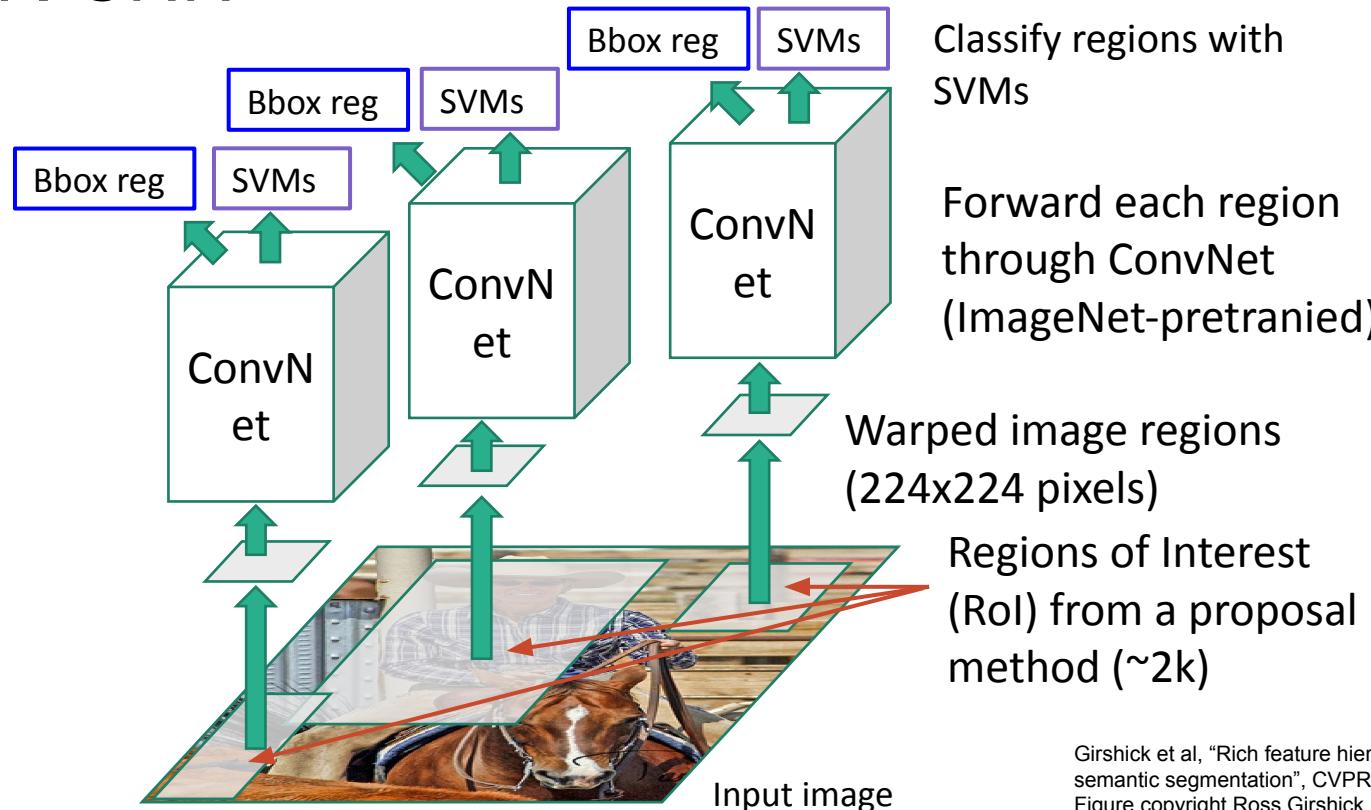
R-CNN



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

R-CNN

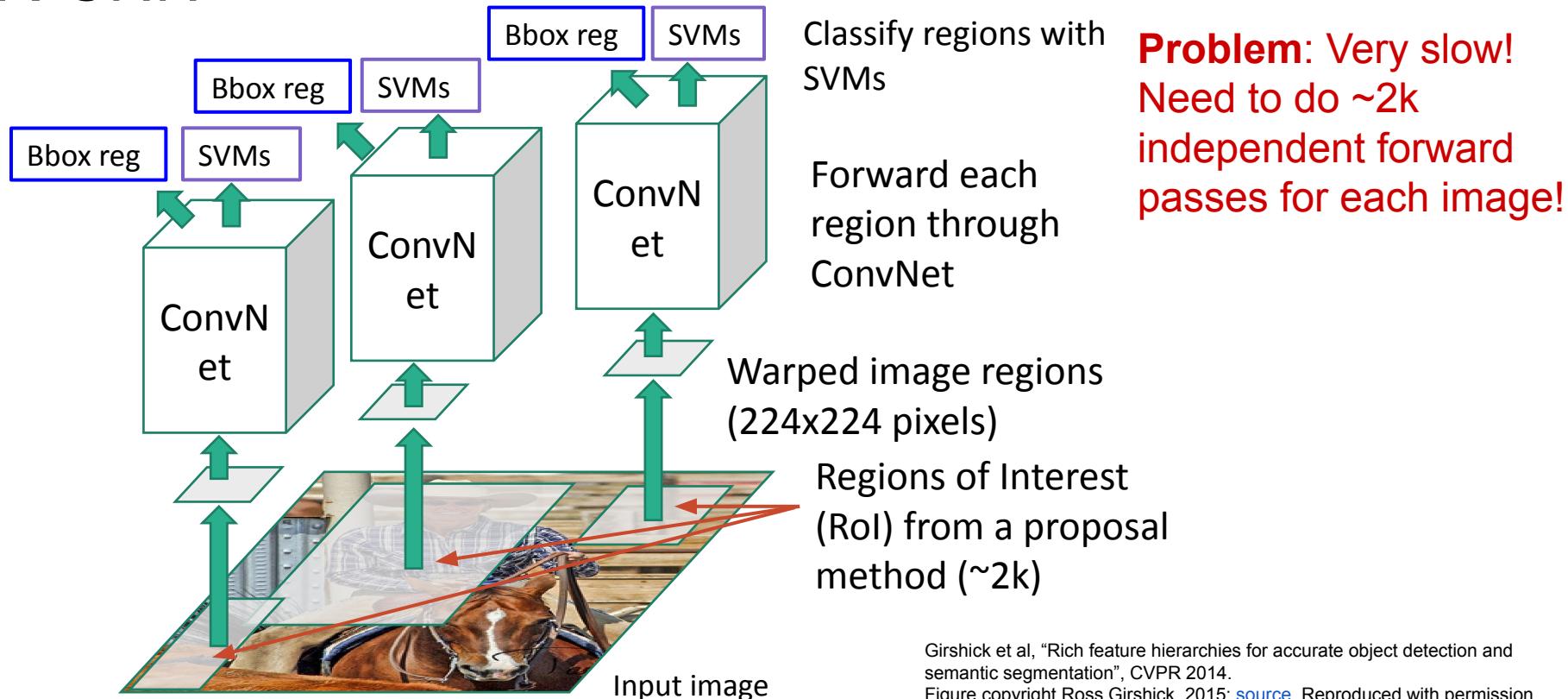
Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)



Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

R-CNN

Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)

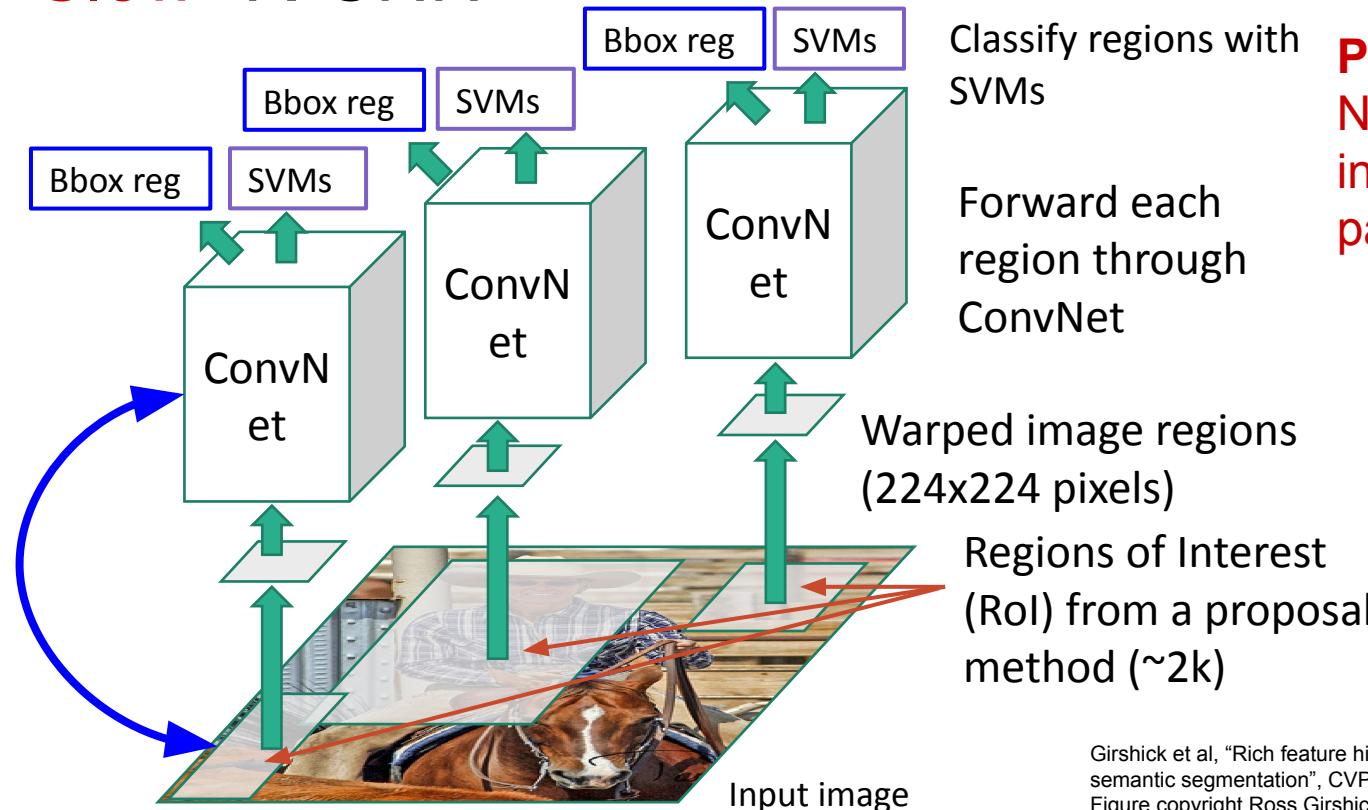


Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

“Slow” R-CNN

Predict “corrections” to the RoI: 4 numbers: (dx, dy, dw, dh)



Problem: Very slow!
Need to do ~2k independent forward passes for each image!

Idea: Pass the image through convnet before cropping! Crop the conv feature instead!

Girshick et al, “Rich feature hierarchies for accurate object detection and semantic segmentation”, CVPR 2014.

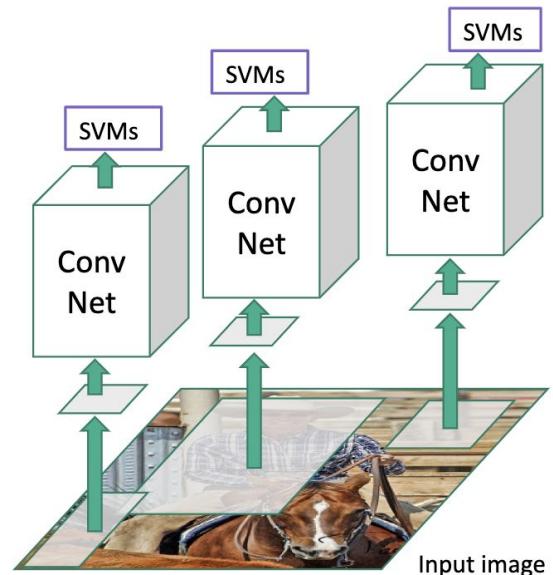
Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

Fast R-CNN



Input image

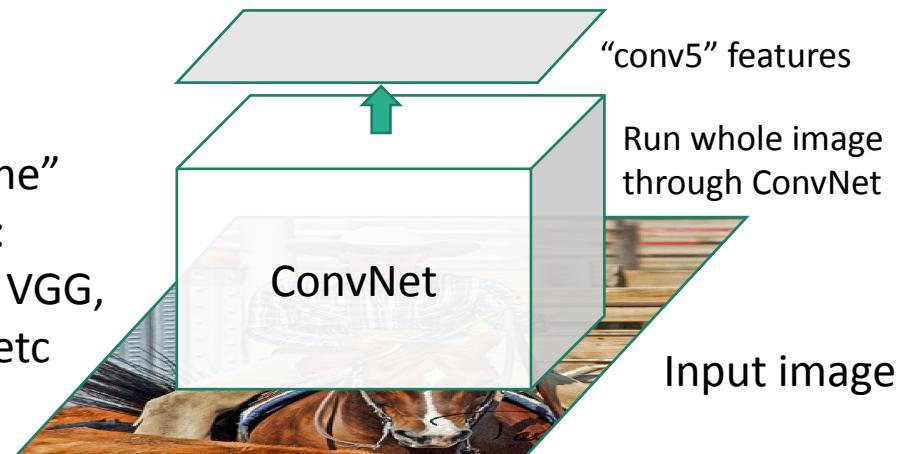
“Slow” R-CNN



Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

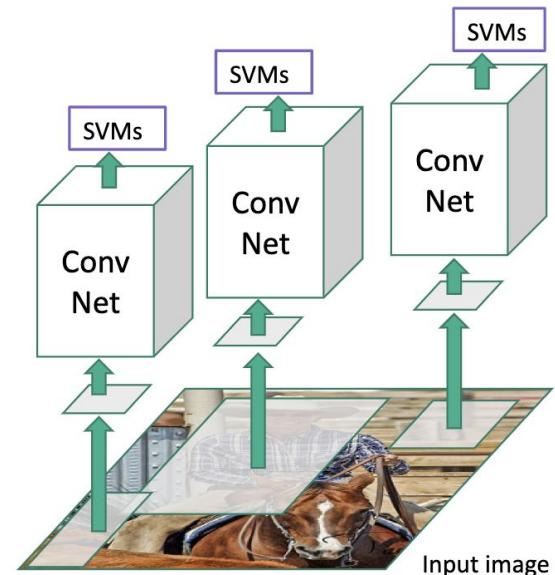
Fast R-CNN

“Backbone” network:
AlexNet, VGG,
ResNet, etc



Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

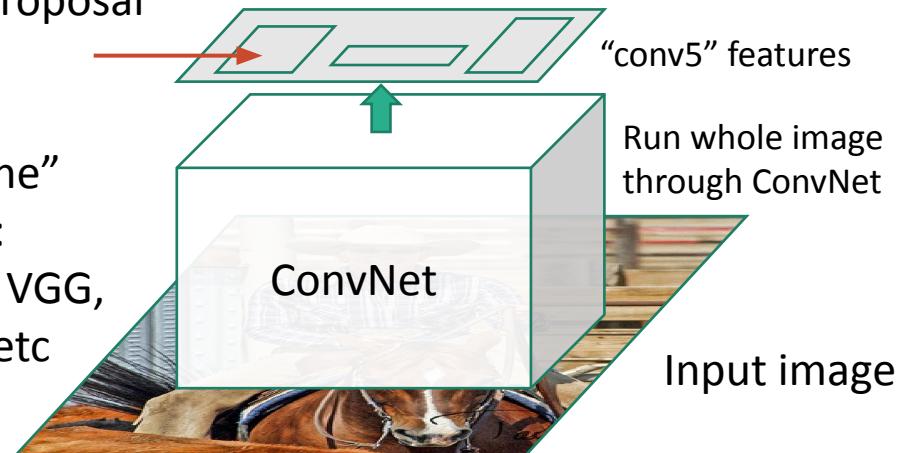
“Slow” R-CNN



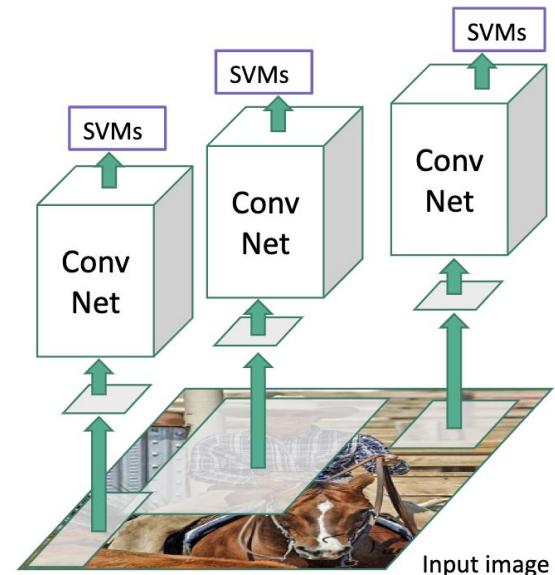
Fast R-CNN

Regions of Interest (RoIs)
from a proposal
method

“Backbone”
network:
AlexNet, VGG,
ResNet, etc



“Slow” R-CNN

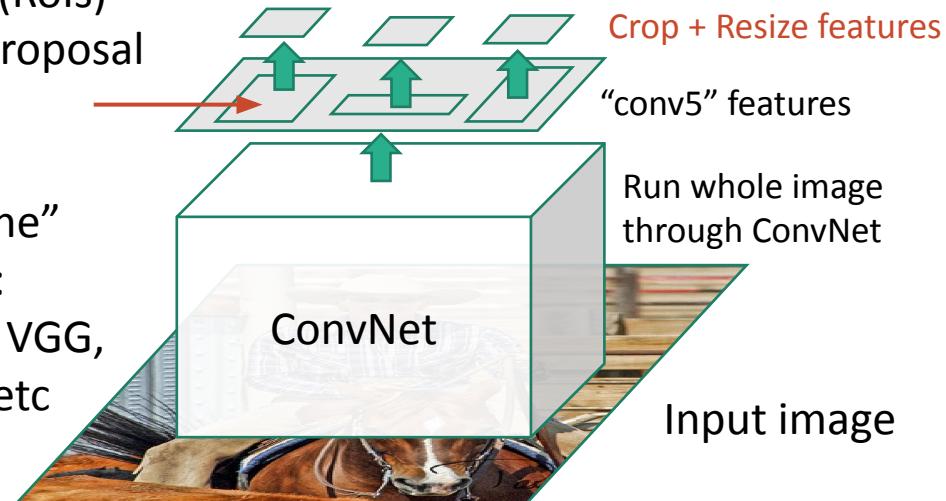


Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

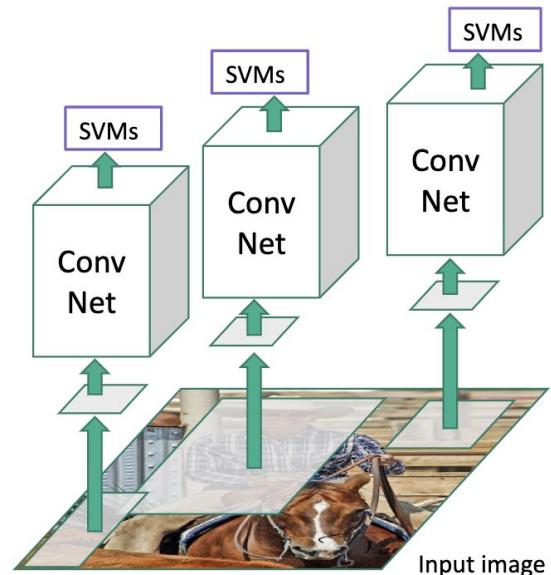
Fast R-CNN

Regions of Interest (RoIs)
from a proposal
method

“Backbone”
network:
AlexNet, VGG,
ResNet, etc

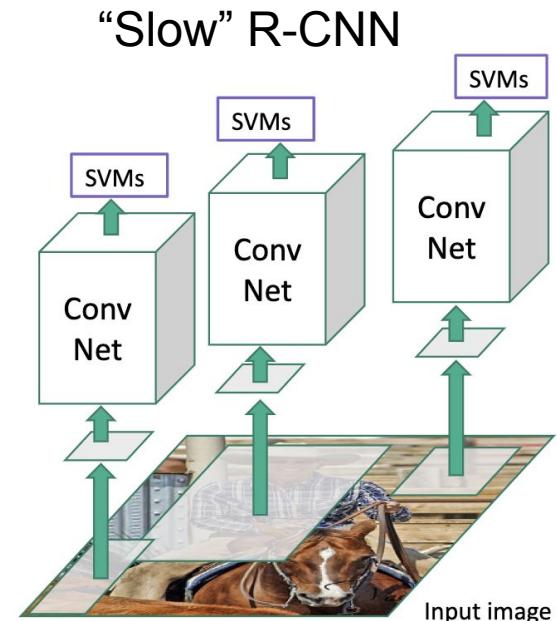
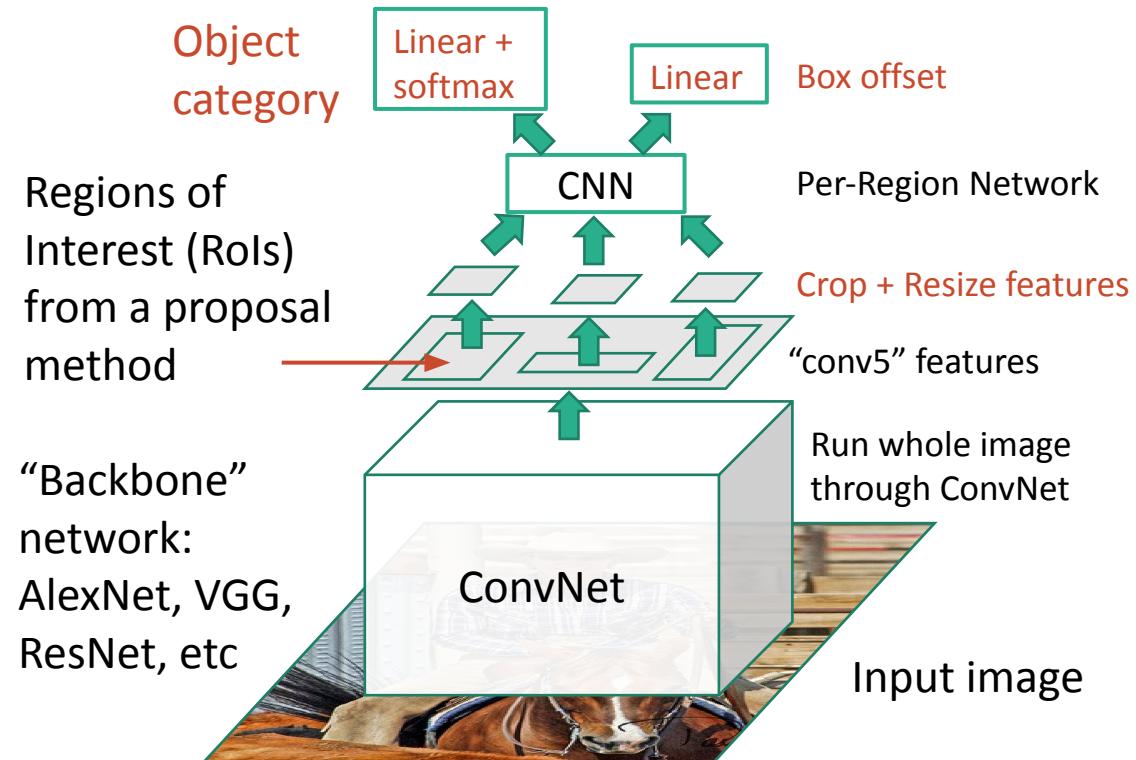


“Slow” R-CNN



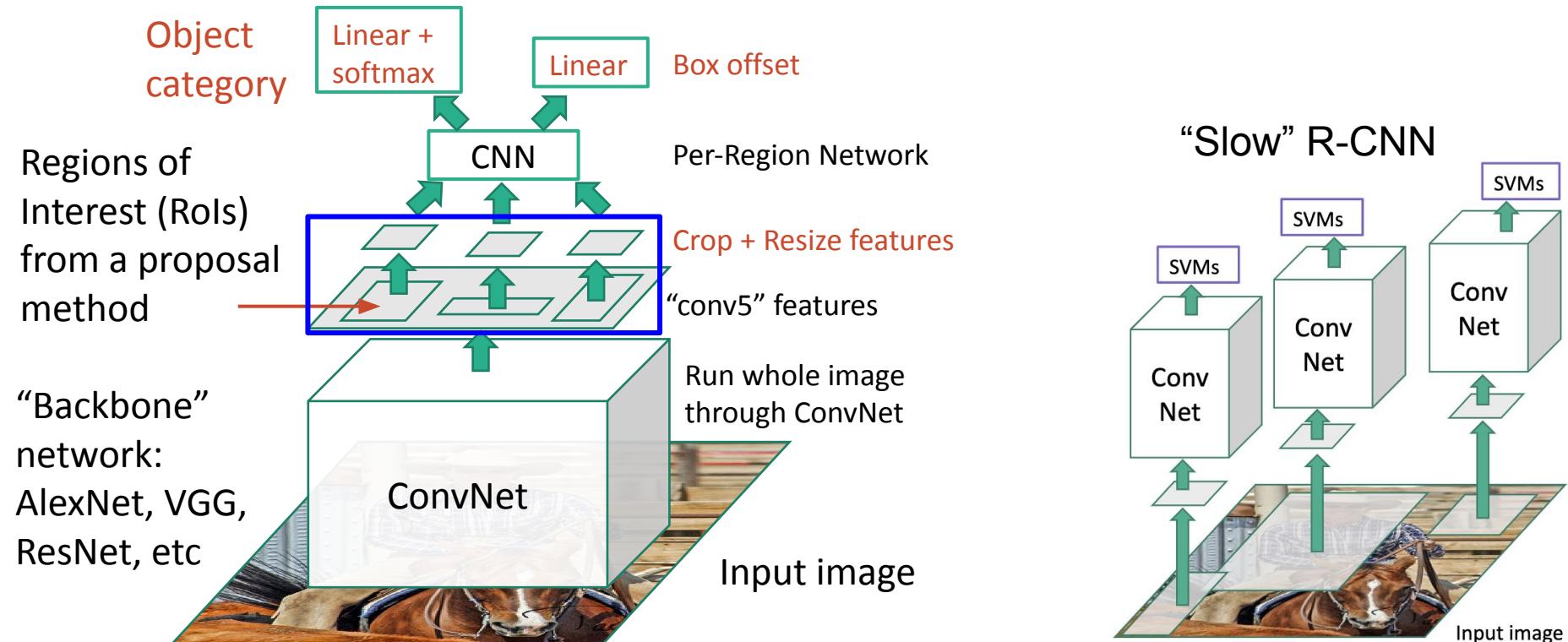
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

Fast R-CNN



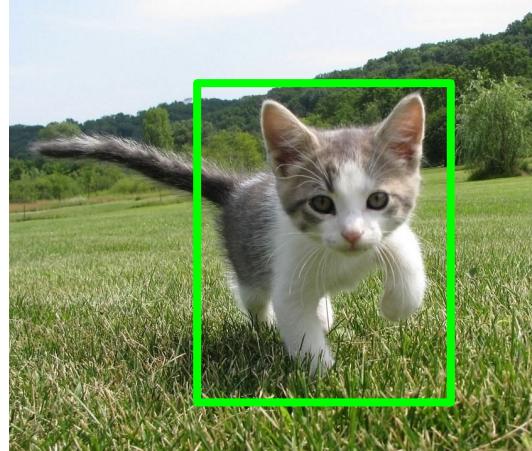
Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

Fast R-CNN



Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.

Cropping Features: RoI Pool



Input Image
(e.g. $3 \times 640 \times 480$)

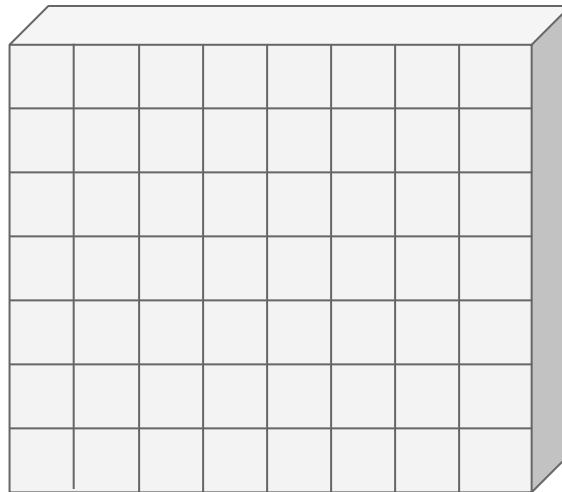
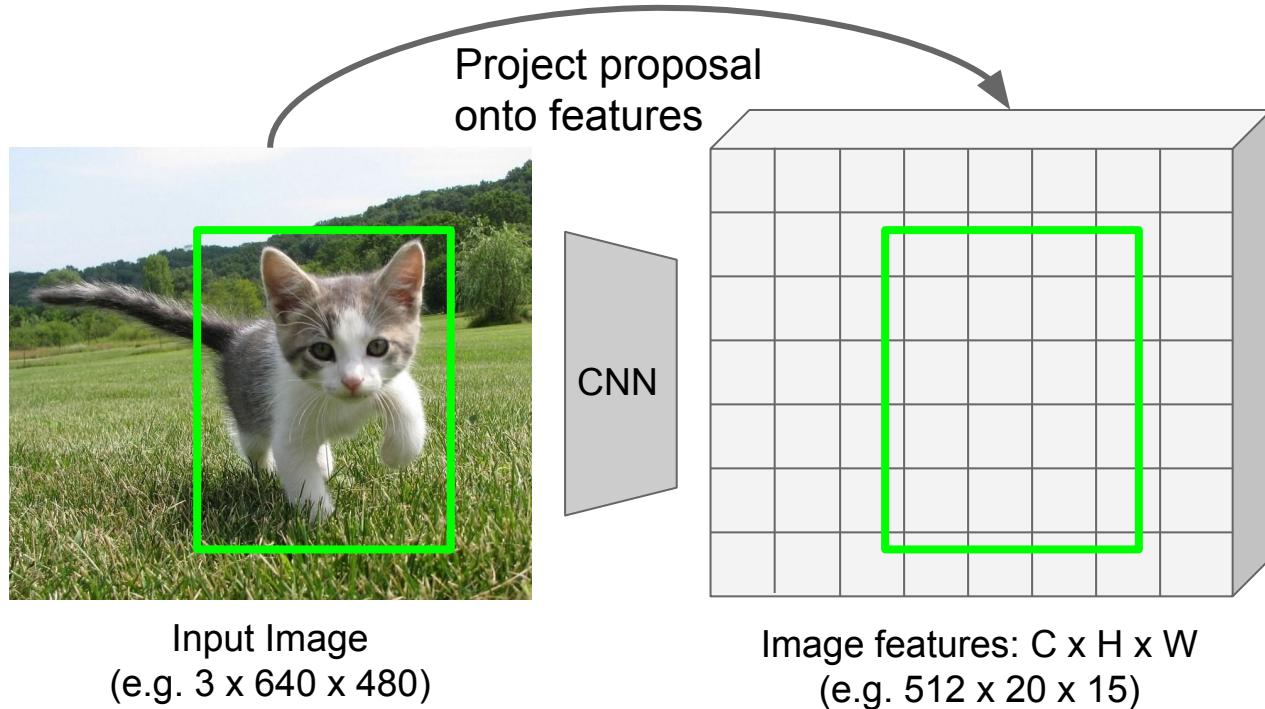


Image features: $C \times H \times W$
(e.g. $512 \times 20 \times 15$)

Girshick, "Fast R-CNN", ICCV 2015.

Girshick, "Fast R-CNN", ICCV 2015.

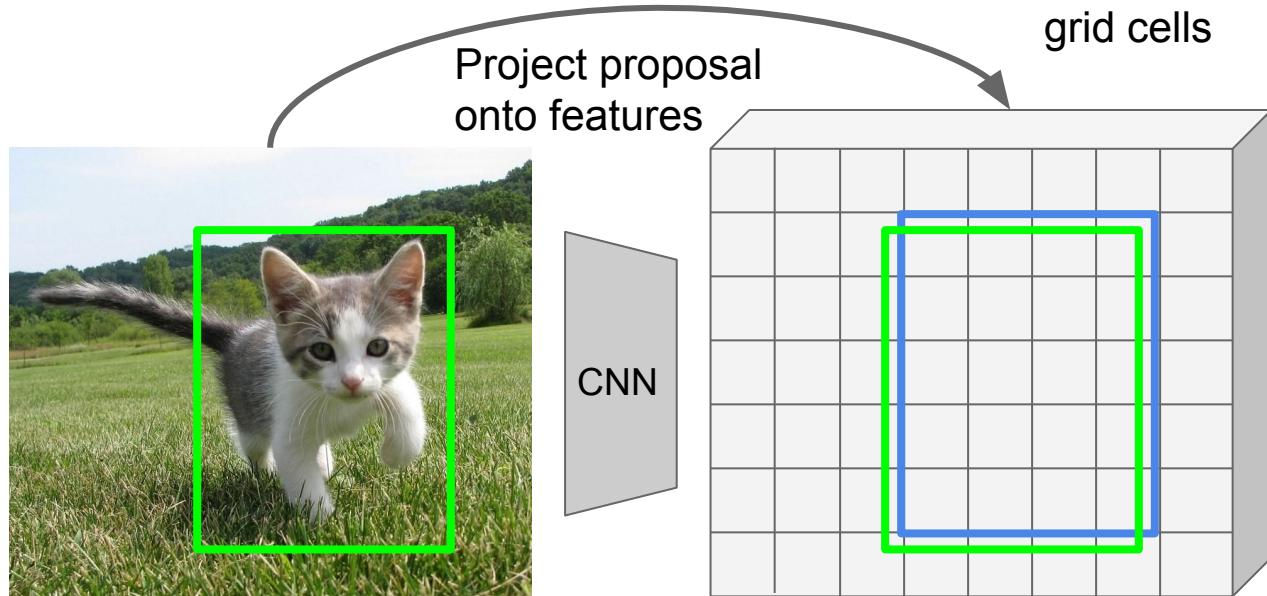
Cropping Features: RoI Pool



Girshick, "Fast R-CNN", ICCV 2015.

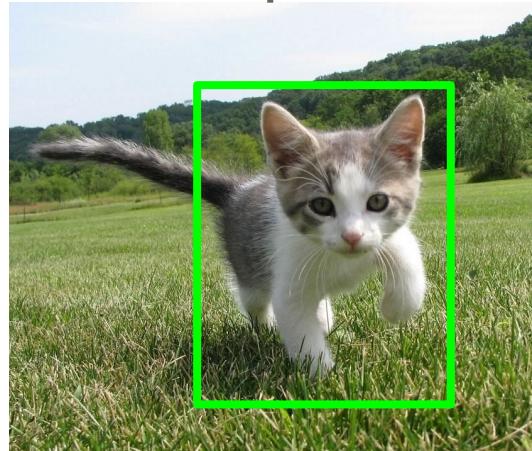
Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool



Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool



Input Image
(e.g. $3 \times 640 \times 480$)

Project proposal
onto features

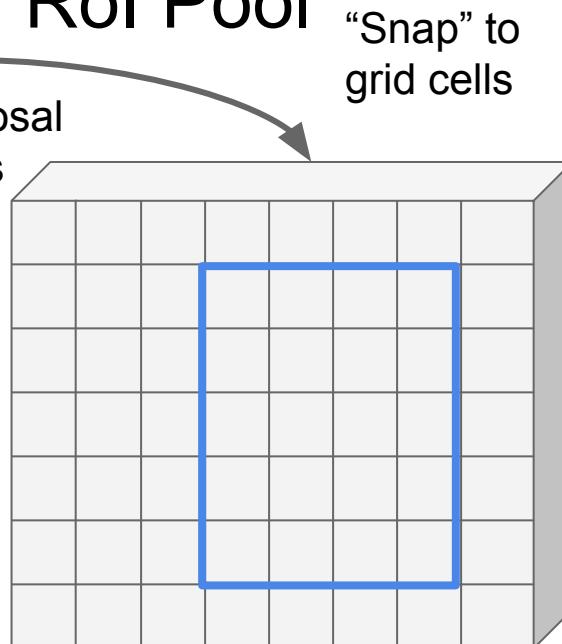


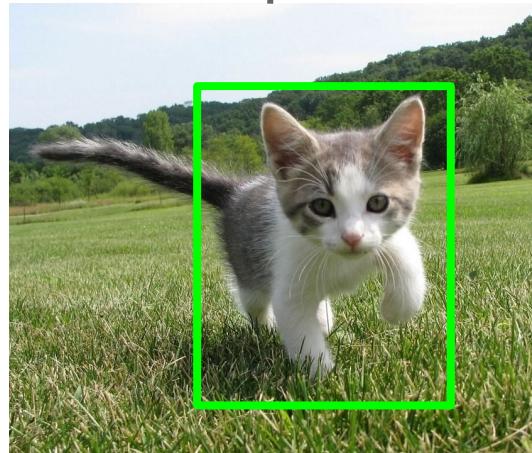
Image features: $C \times H \times W$
(e.g. $512 \times 20 \times 15$)

“Snap” to
grid cells

Q: how do we resize the $512 \times 5 \times 4$ region to, e.g., a $512 \times 2 \times 2$ tensor?.

Girshick, “Fast R-CNN”, ICCV 2015.

Cropping Features: RoI Pool



Input Image
(e.g. $3 \times 640 \times 480$)

Project proposal
onto features

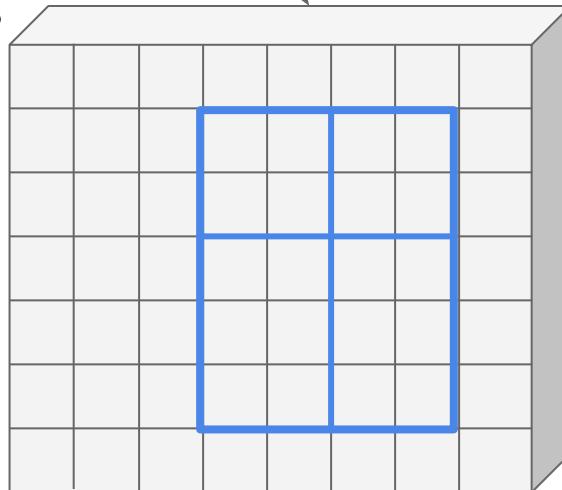


Image features: $C \times H \times W$
(e.g. $512 \times 20 \times 15$)

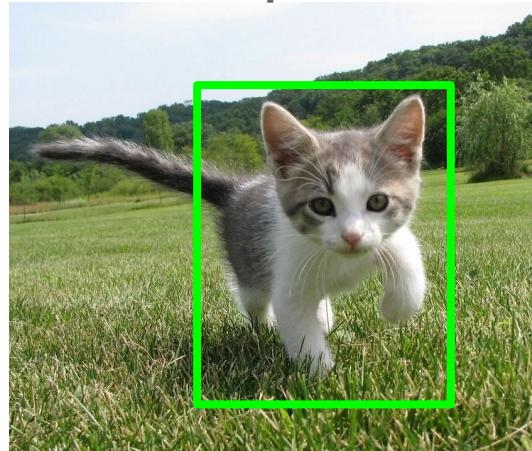
“Snap” to
grid cells

Divide into 2×2
grid of (roughly)
equal subregions

Q: how do we resize the $512 \times 5 \times 4$ region to, e.g., a $512 \times 2 \times 2$ tensor?.

Girshick, “Fast R-CNN”, ICCV 2015.

Cropping Features: RoI Pool



Input Image
(e.g. $3 \times 640 \times 480$)

Project proposal
onto features

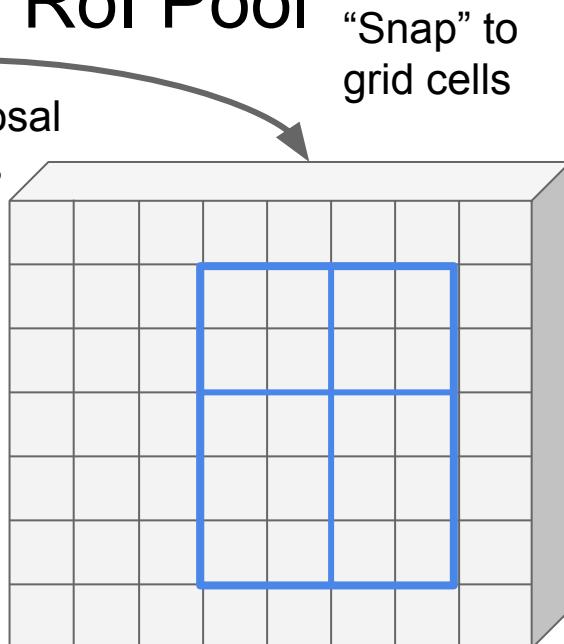
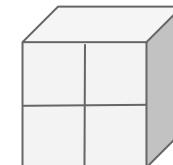


Image features: $C \times H \times W$
(e.g. $512 \times 20 \times 15$)

Divide into 2×2
grid of (roughly)
equal subregions

Max-pool within
each subregion

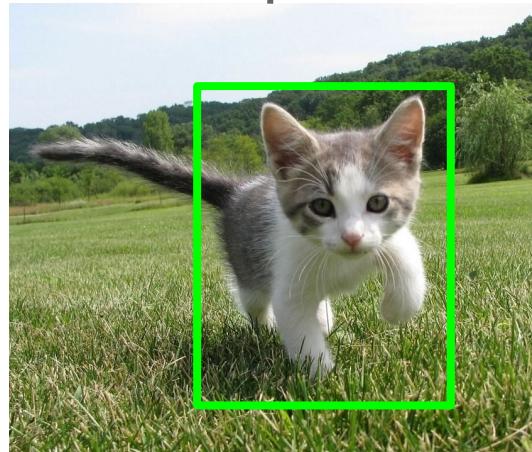


Region features
(here $512 \times 2 \times 2$;
In practice e.g. $512 \times 7 \times 7$)

Region features always the
same size even if input
regions have different sizes!

Girshick, "Fast R-CNN", ICCV 2015.

Cropping Features: RoI Pool



Input Image
(e.g. $3 \times 640 \times 480$)

Project proposal
onto features

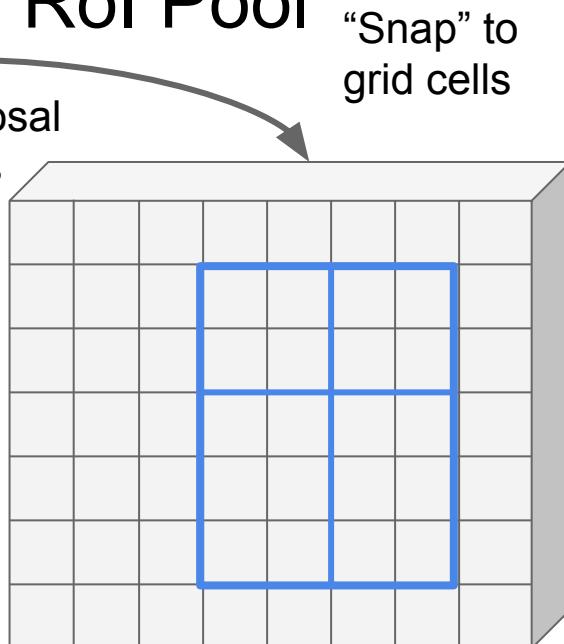
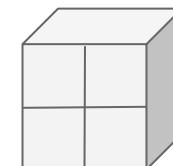


Image features: $C \times H \times W$
(e.g. $512 \times 20 \times 15$)

“Snap” to
grid cells

Divide into 2×2
grid of (roughly)
equal subregions

Max-pool within
each subregion



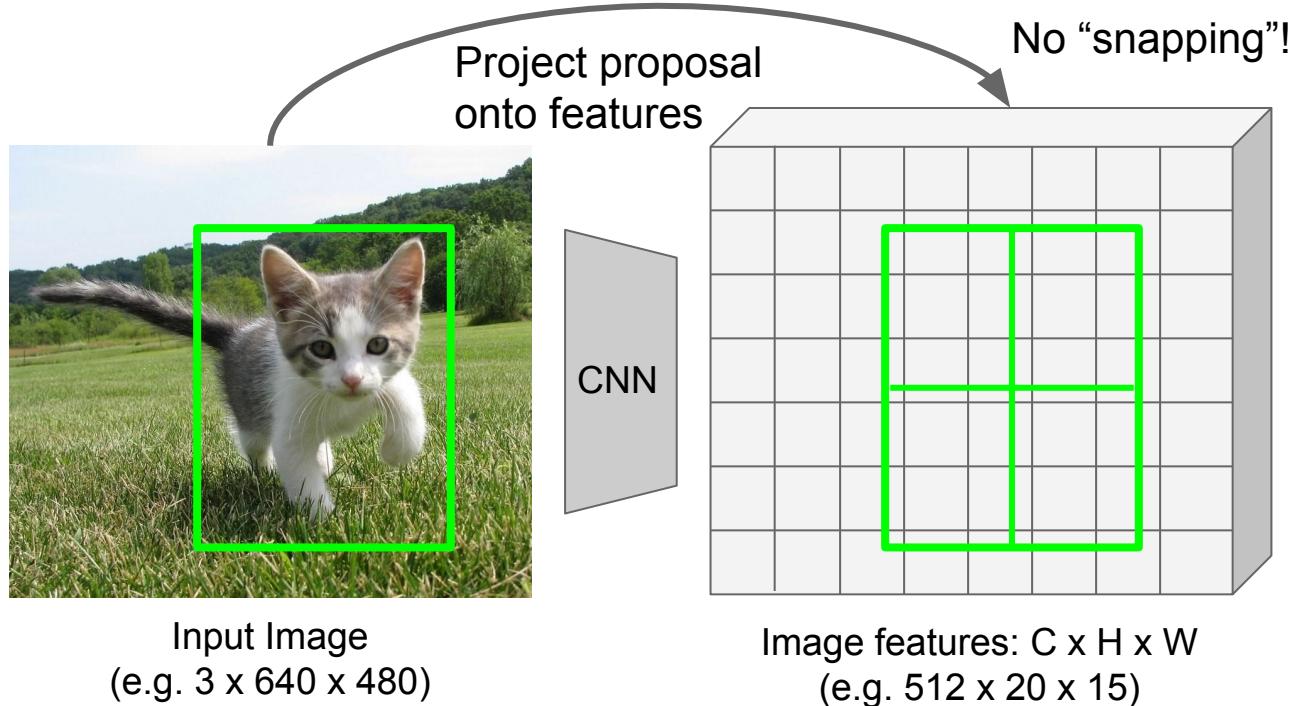
Region features
(here $512 \times 2 \times 2$;
In practice e.g. $512 \times 7 \times 7$)

Region features always the
same size even if input
regions have different sizes!

Problem: Region features slightly misaligned

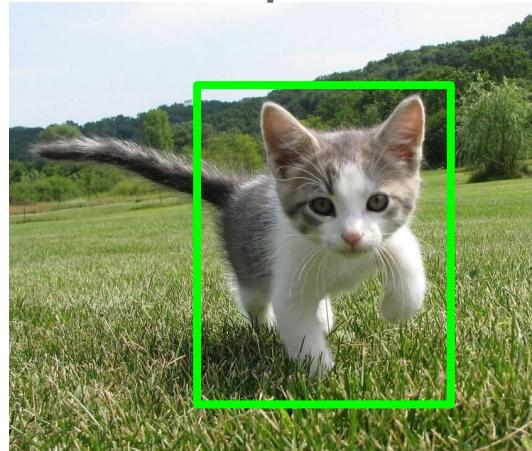
Girshick, “Fast R-CNN”, ICCV 2015.

Cropping Features: RoI Align



He et al, "Mask R-CNN", ICCV 2017

Cropping Features: RoI Align



Input Image
(e.g. $3 \times 640 \times 480$)

Project proposal
onto features



No “snapping”!

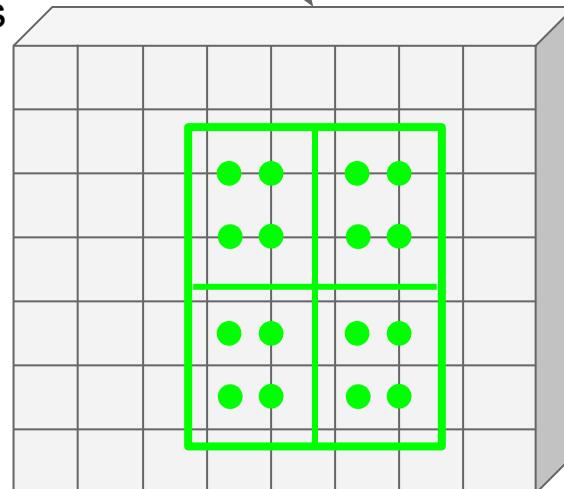
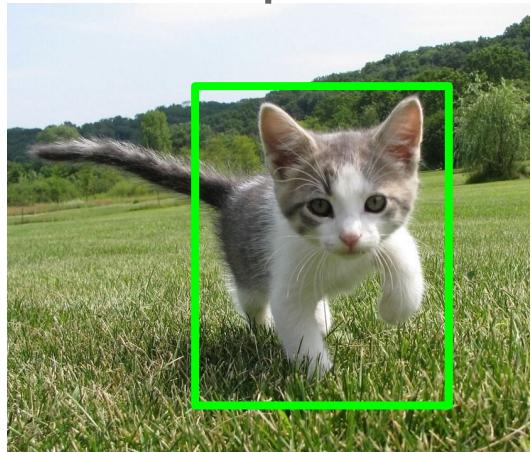


Image features: $C \times H \times W$
(e.g. $512 \times 20 \times 15$)

Sample at regular points
in each subregion using
bilinear interpolation

Cropping Features: RoI Align



Input Image
(e.g. $3 \times 640 \times 480$)

Project proposal
onto features



No “snapping”!

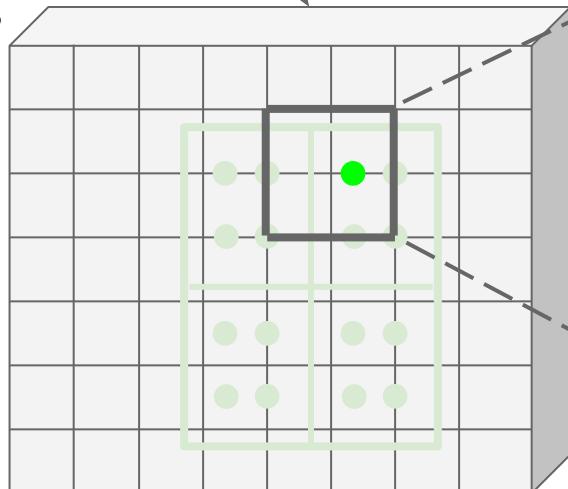
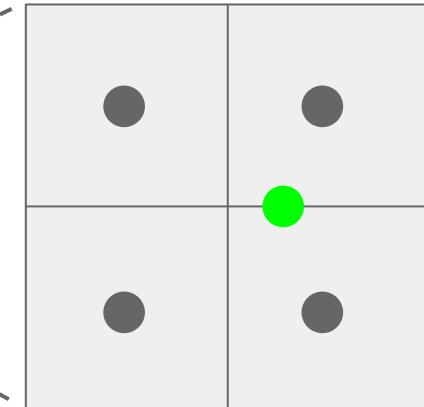


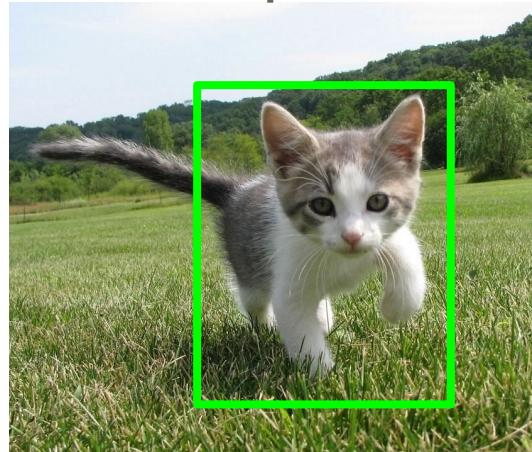
Image features: $C \times H \times W$
(e.g. $512 \times 20 \times 15$)

Sample at regular points
in each subregion using
bilinear interpolation



Feature f_{xy} for point (x, y)
is a linear combination of
features at its four
neighboring grid cells:

Cropping Features: RoI Align



Input Image
(e.g. 3 x 640 x 480)

Project proposal
onto features



No “snapping”!

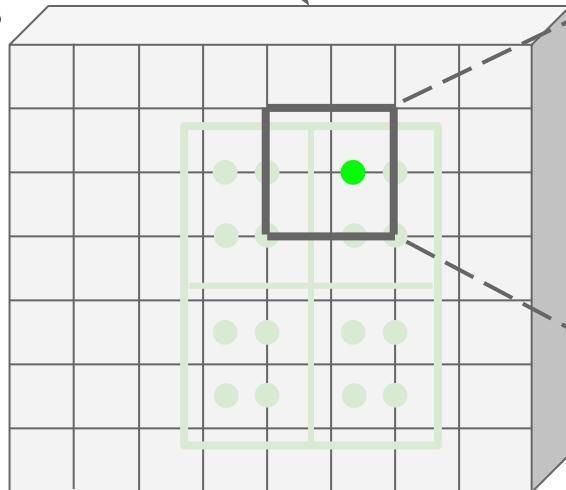
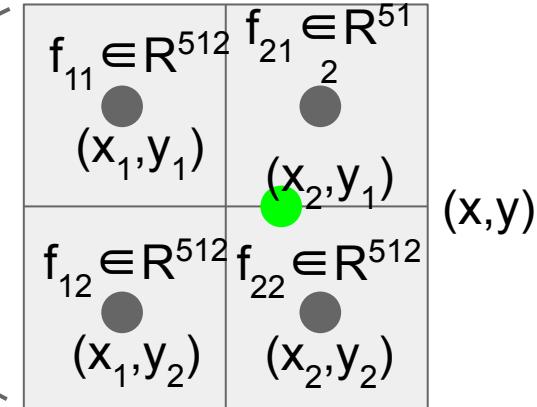


Image features: C x H x W
(e.g. 512 x 20 x 15)

Sample at regular points
in each subregion using
bilinear interpolation

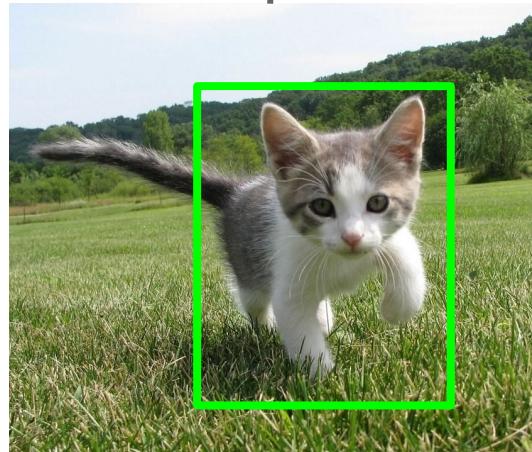


Feature f_{xy} for point (x, y)
is a linear combination of
features at its four
neighboring grid cells:

$$f_{xy} = \sum_{i,j=1}^2 f_{i,j} \max(0, 1 - |x - x_i|) \max(0, 1 - |y - y_j|)$$

He et al, “Mask R-CNN”, ICCV 2017

Cropping Features: RoI Align



Input Image
(e.g. $3 \times 640 \times 480$)

Project proposal
onto features



No “snapping”!

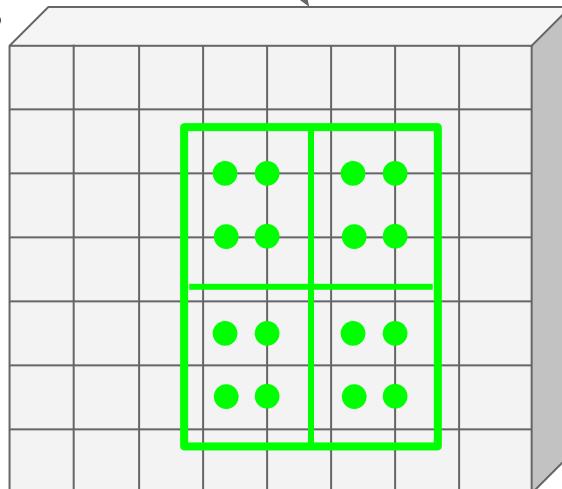
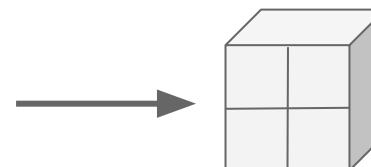


Image features: $C \times H \times W$
(e.g. $512 \times 20 \times 15$)

Sample at regular points
in each subregion using
bilinear interpolation

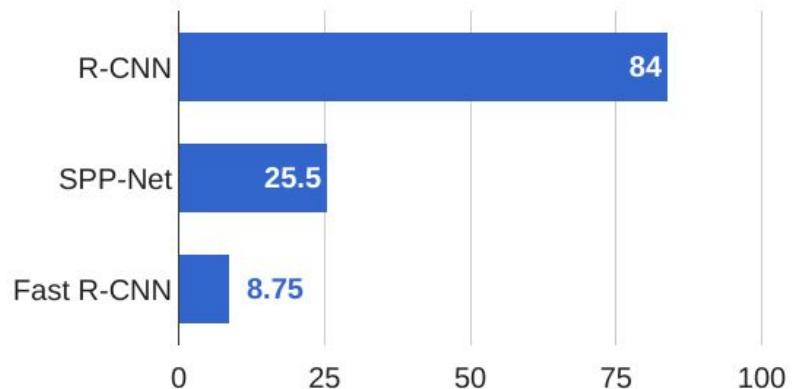
Max-pool within
each subregion



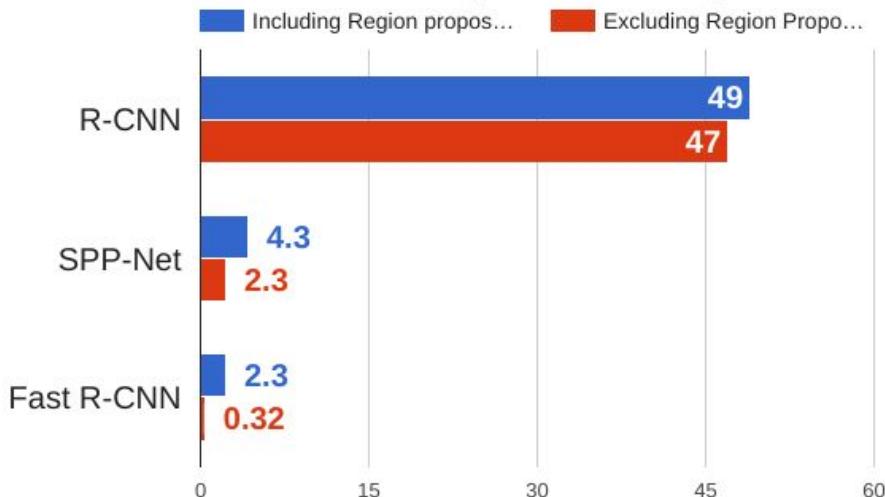
Region features
(here $512 \times 2 \times 2$;
In practice e.g $512 \times 7 \times 7$)

R-CNN vs Fast R-CNN

Training time (Hours)



Test time (seconds)



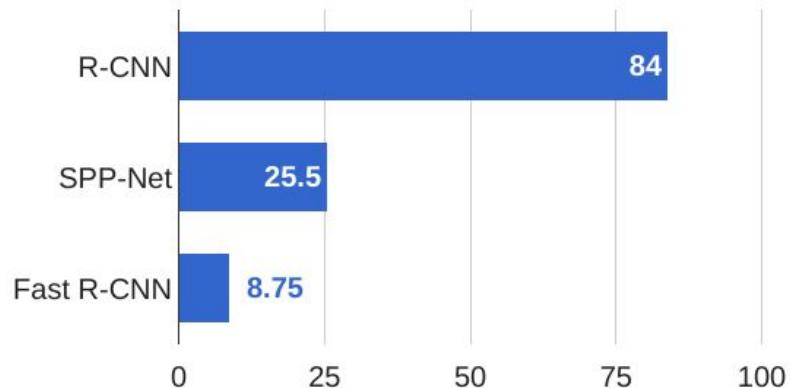
Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014

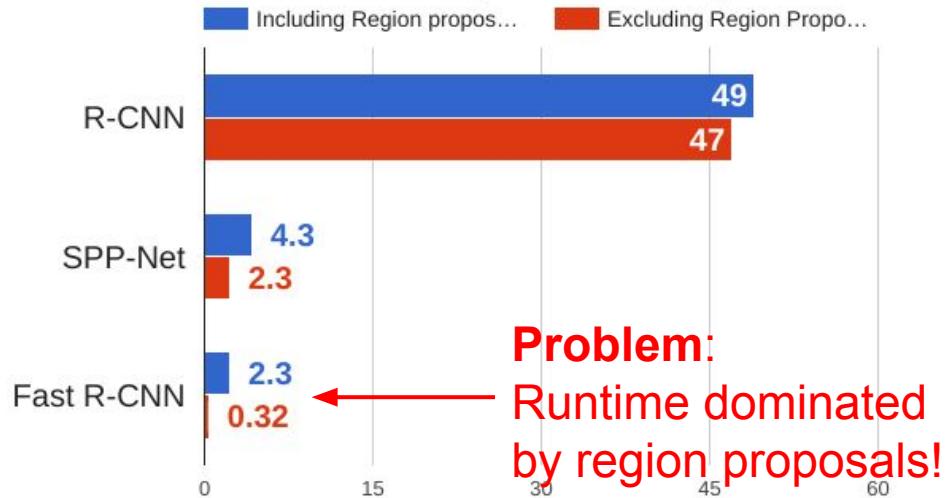
Girshick, "Fast R-CNN", ICCV 2015

R-CNN vs Fast R-CNN

Training time (Hours)



Test time (seconds)



Girshick et al, "Rich feature hierarchies for accurate object detection and semantic segmentation", CVPR 2014.

He et al, "Spatial pyramid pooling in deep convolutional networks for visual recognition", ECCV 2014

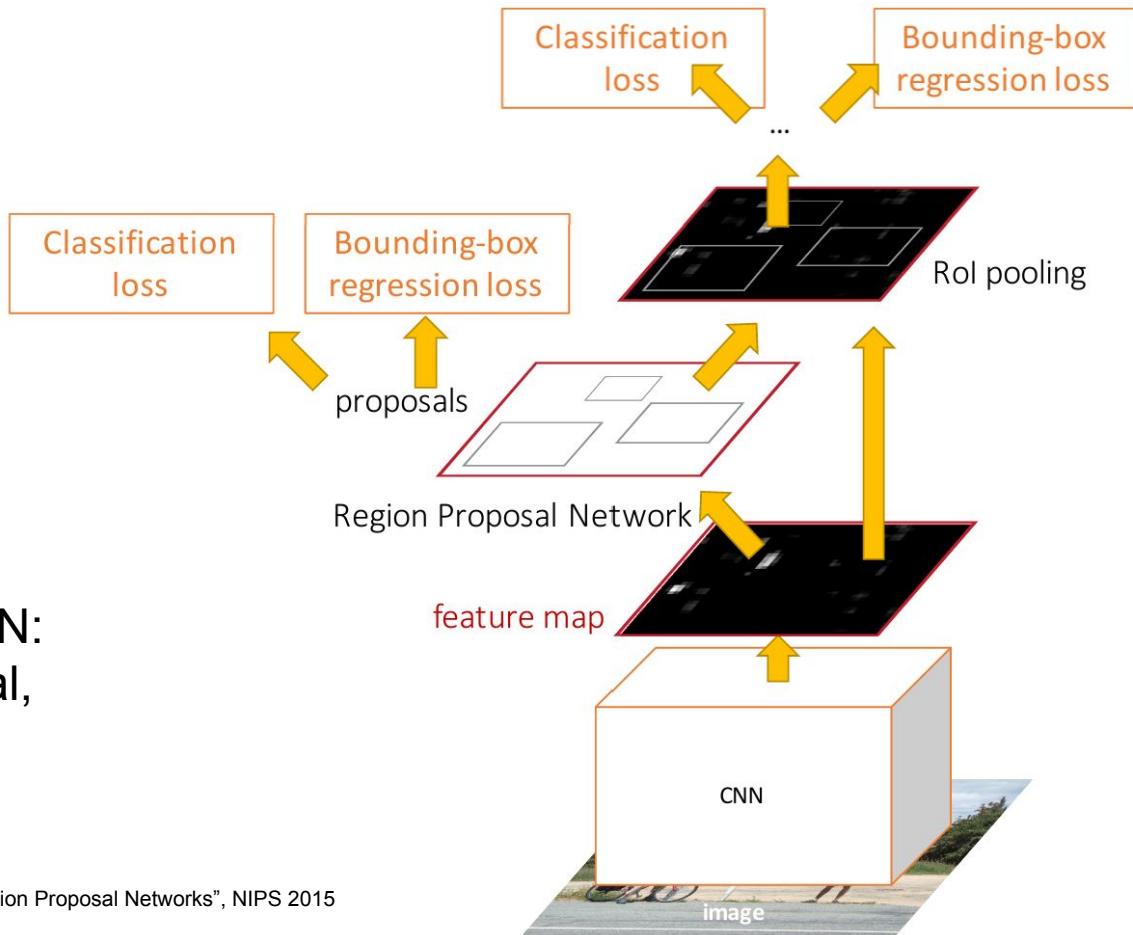
Girshick, "Fast R-CNN", ICCV 2015

Faster R-CNN:

Make CNN do proposals!

Insert **Region Proposal Network (RPN)** to predict proposals from features

Otherwise same as Fast R-CNN:
Crop features for each proposal,
classify each one



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Region Proposal Network



Input Image
(e.g. $3 \times 640 \times 480$)

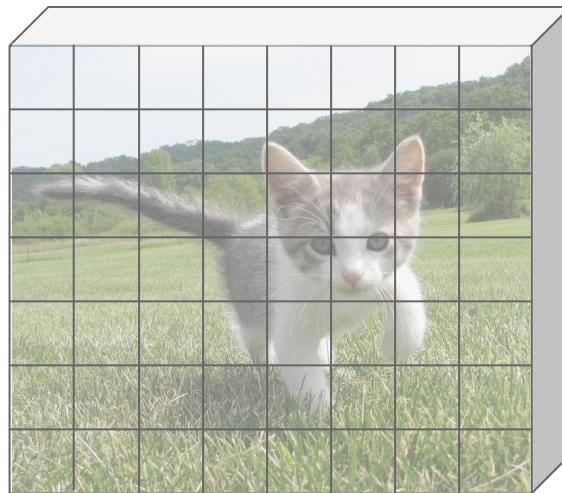
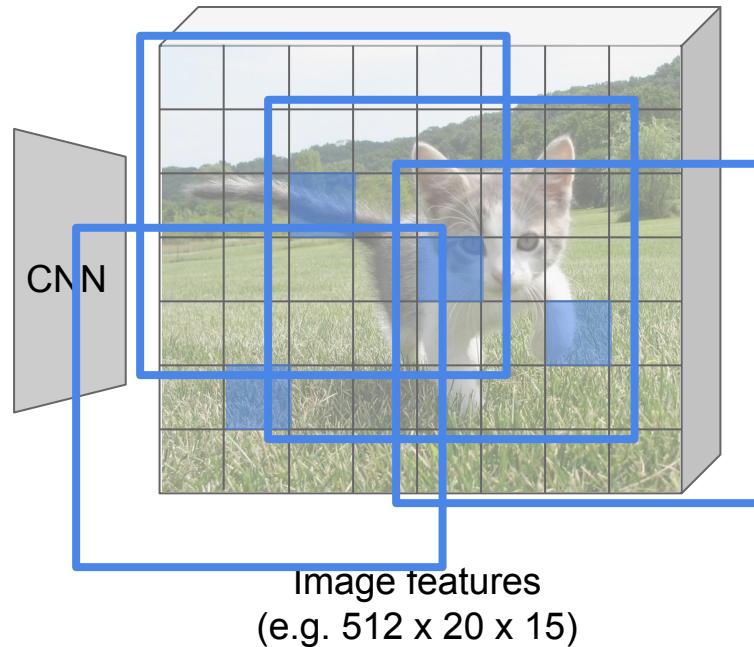


Image features
(e.g. $512 \times 20 \times 15$)

Region Proposal Network



Input Image
(e.g. $3 \times 640 \times 480$)

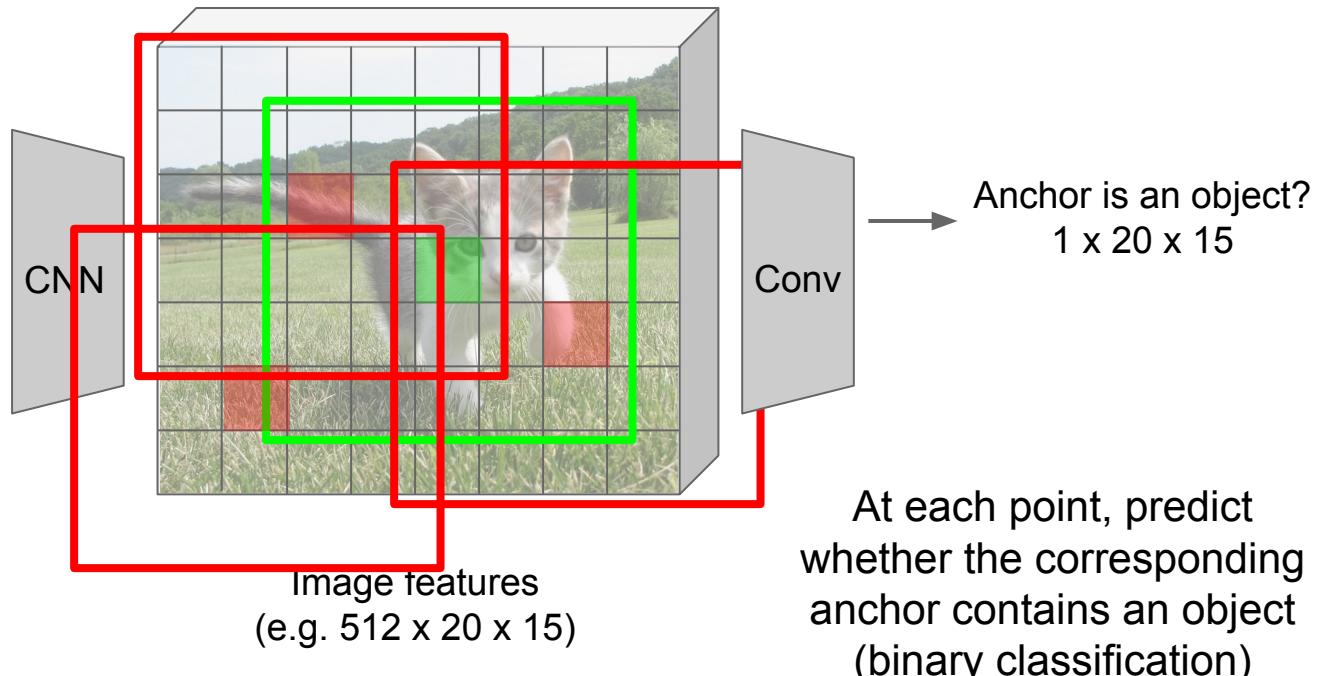


Imagine an **anchor box** of fixed size at each point in the feature map

Region Proposal Network



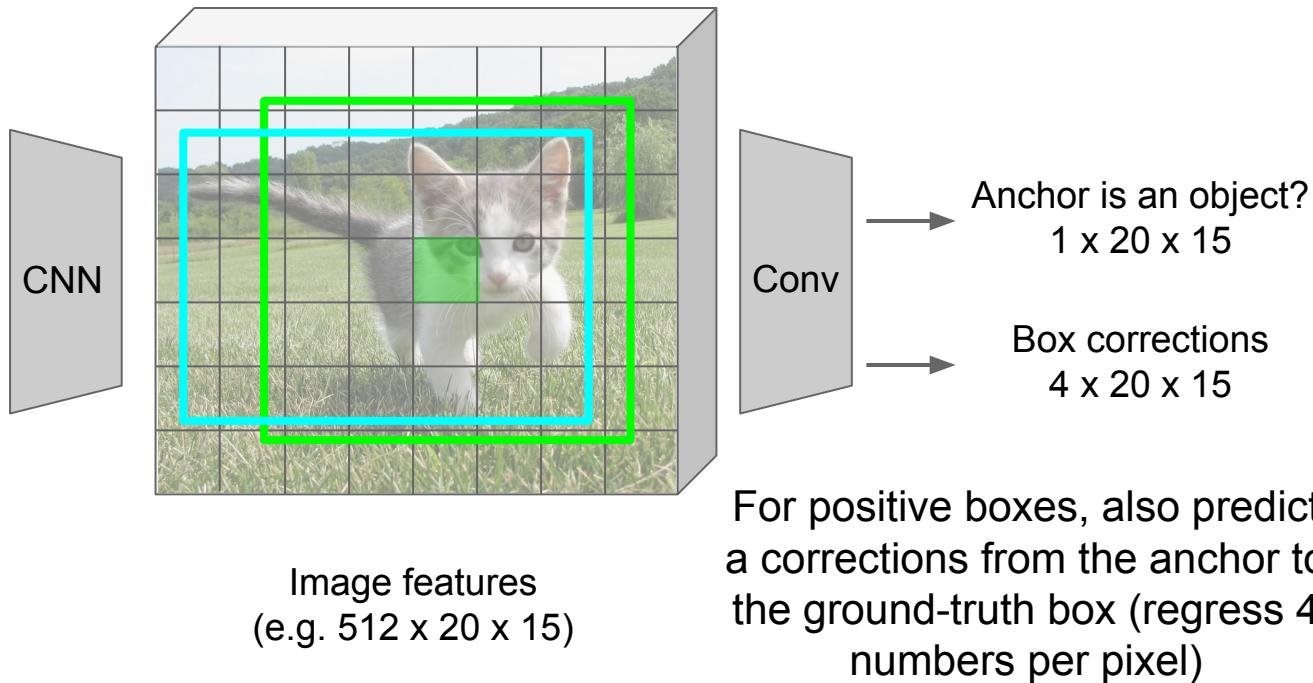
Input Image
(e.g. $3 \times 640 \times 480$)



Region Proposal Network



Input Image
(e.g. $3 \times 640 \times 480$)



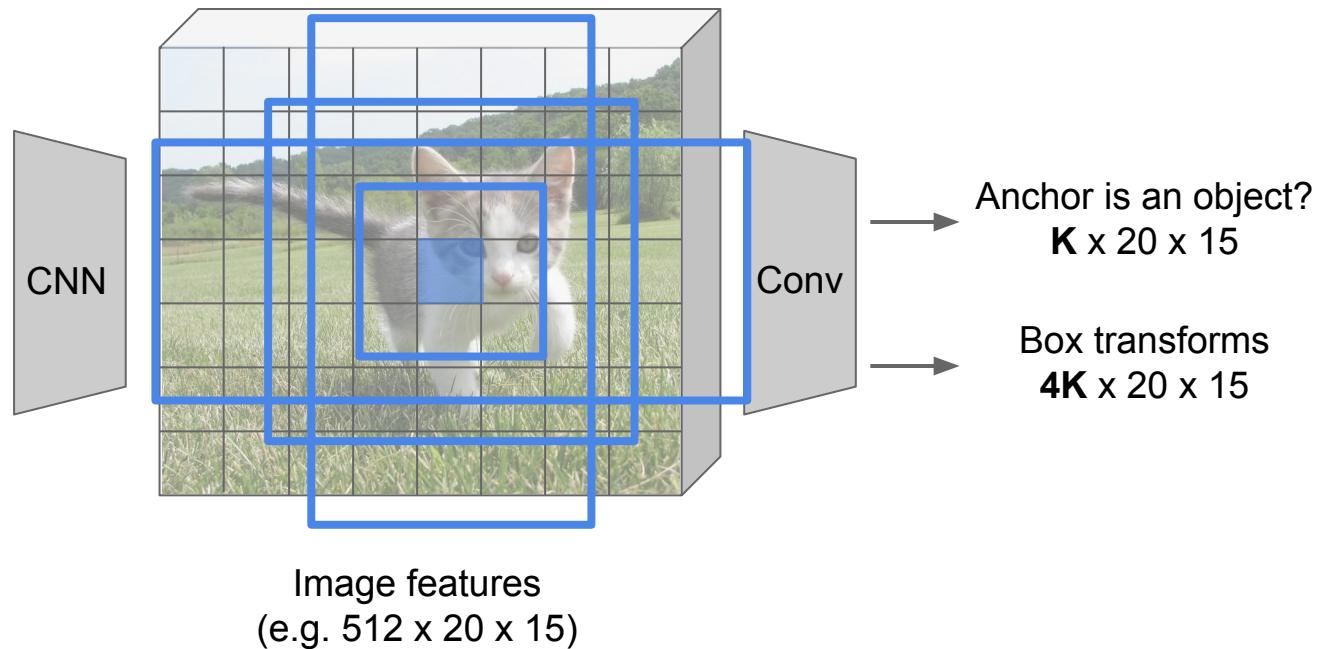
Imagine an **anchor box** of fixed size at each point in the feature map

Region Proposal Network

In practice use K different anchor boxes of different size / scale at each point



Input Image
(e.g. $3 \times 640 \times 480$)

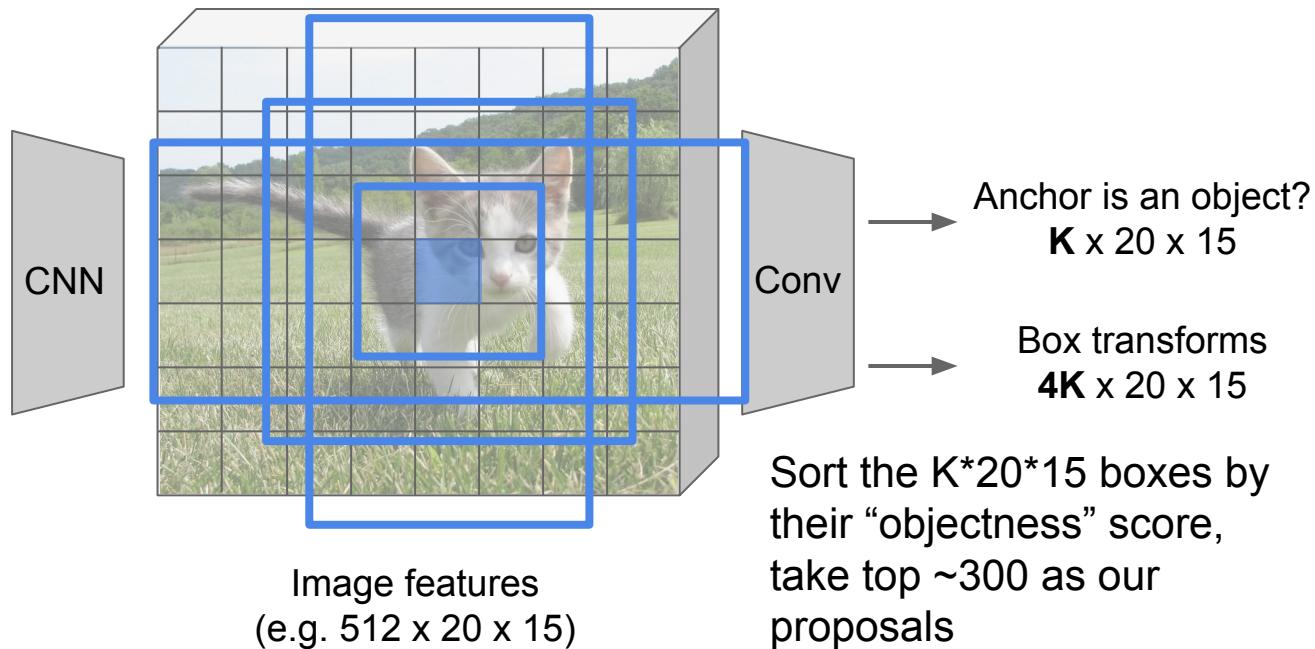


Region Proposal Network

In practice use K different anchor boxes of different size / scale at each point



Input Image
(e.g. $3 \times 640 \times 480$)

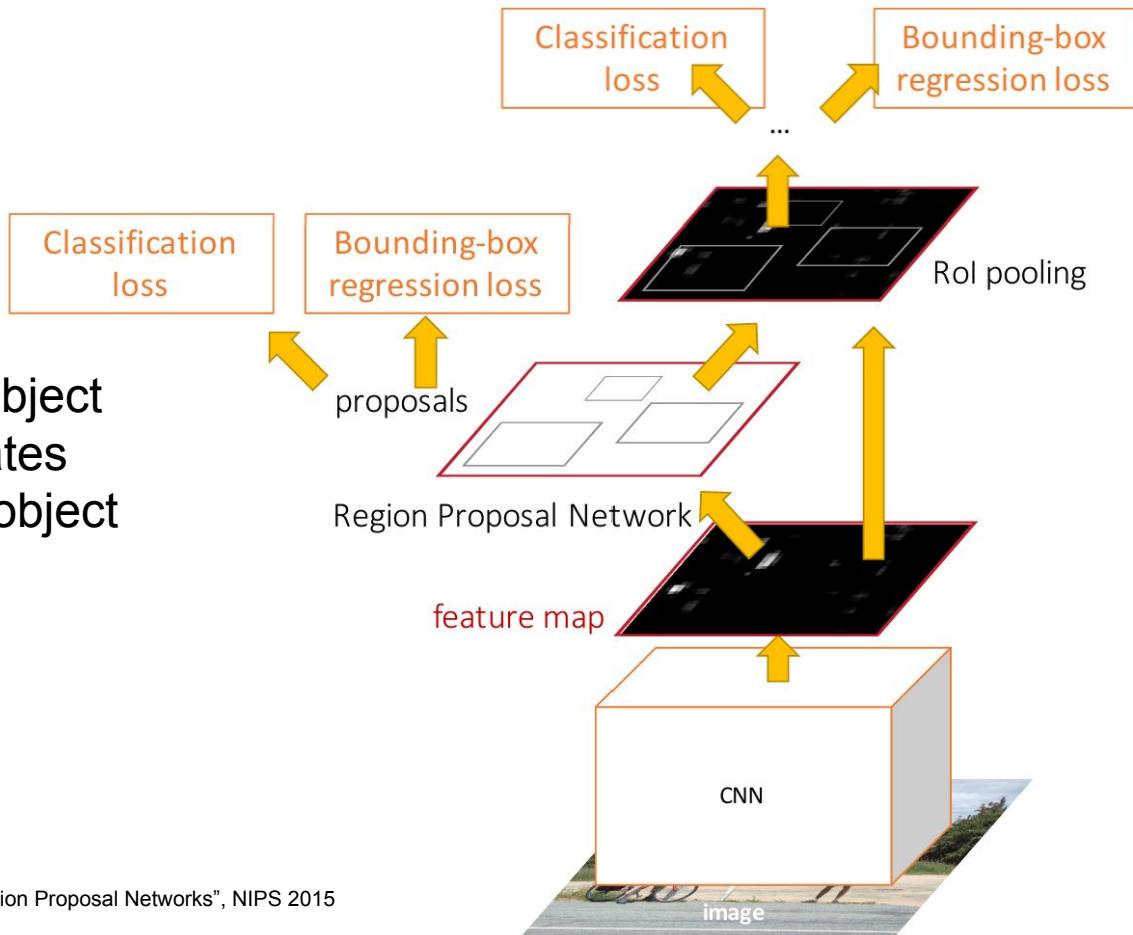


Faster R-CNN:

Make CNN do proposals!

Jointly train with 4 losses:

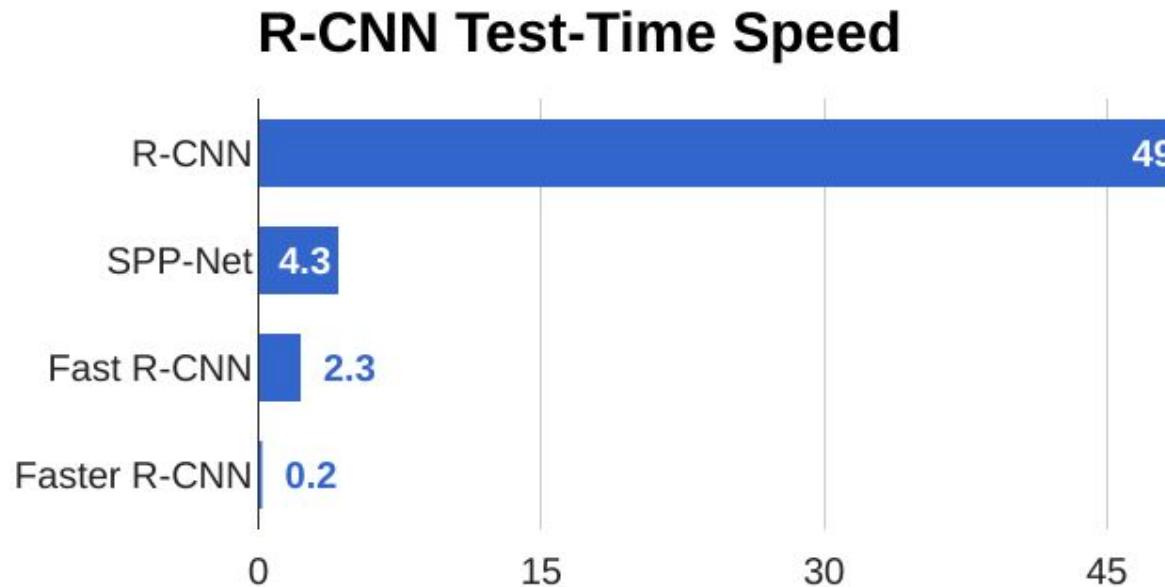
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Faster R-CNN:

Make CNN do proposals!

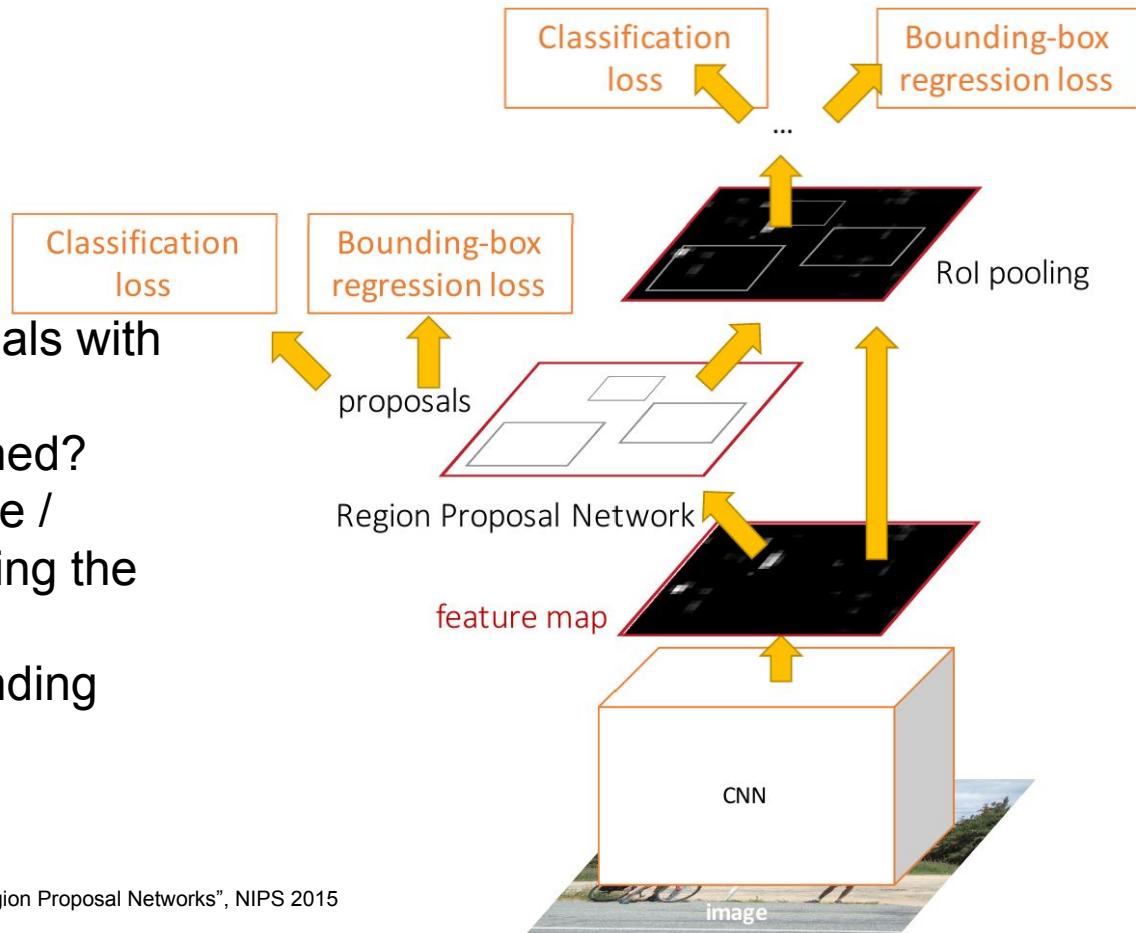


Faster R-CNN:

Make CNN do proposals!

Glossing over many details:

- Ignore overlapping proposals with **non-max suppression**
- How are anchors determined?
- How do we sample positive / negative samples for training the RPN?
- How to parameterize bounding box regression?



Ren et al, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks", NIPS 2015
Figure copyright 2015, Ross Girshick; reproduced with permission

Faster R-CNN:

Make CNN do proposals!

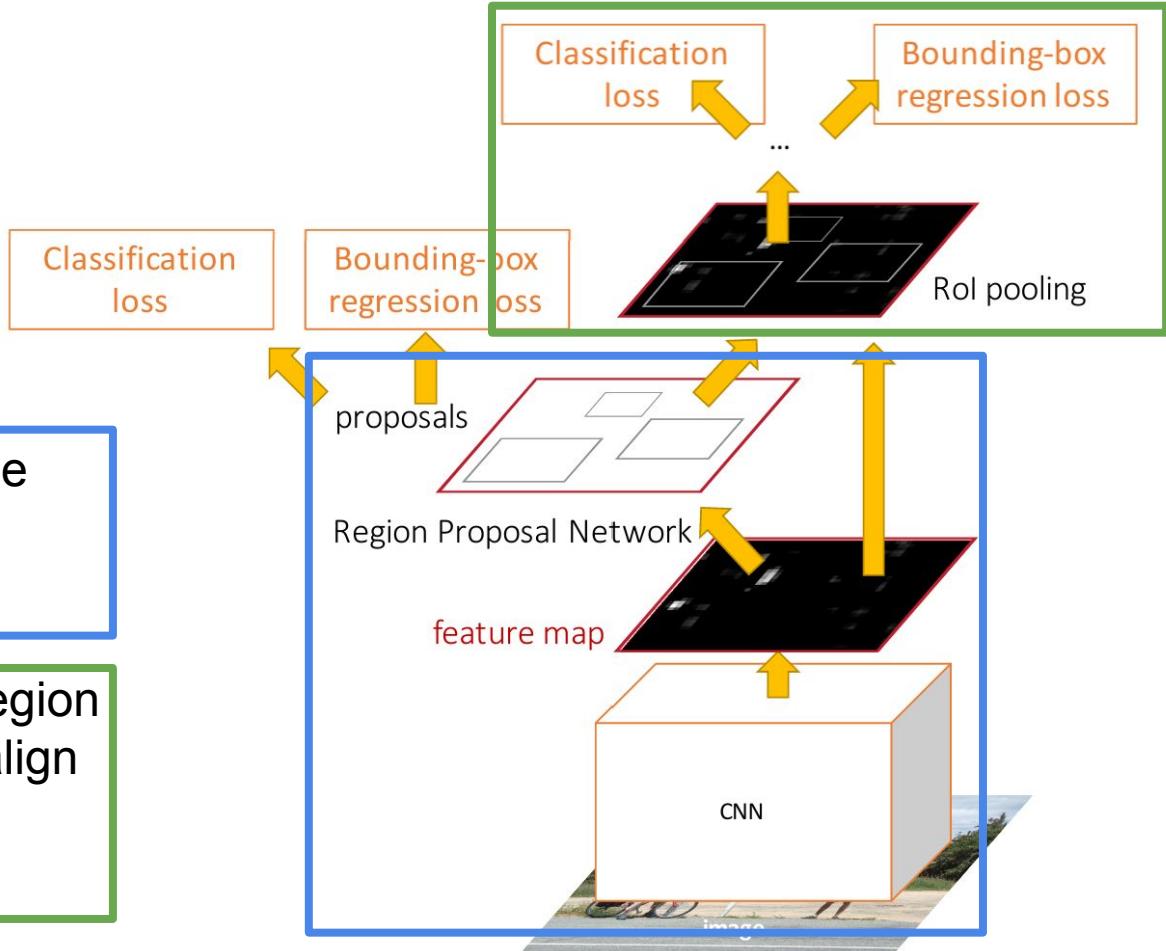
Faster R-CNN is a
Two-stage object detector

First stage: Run once per image

- Backbone network
- Region proposal network

Second stage: Run once per region

- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset



Faster R-CNN: Make CNN do proposals!

Faster R-CNN is a
Two-stage object detector

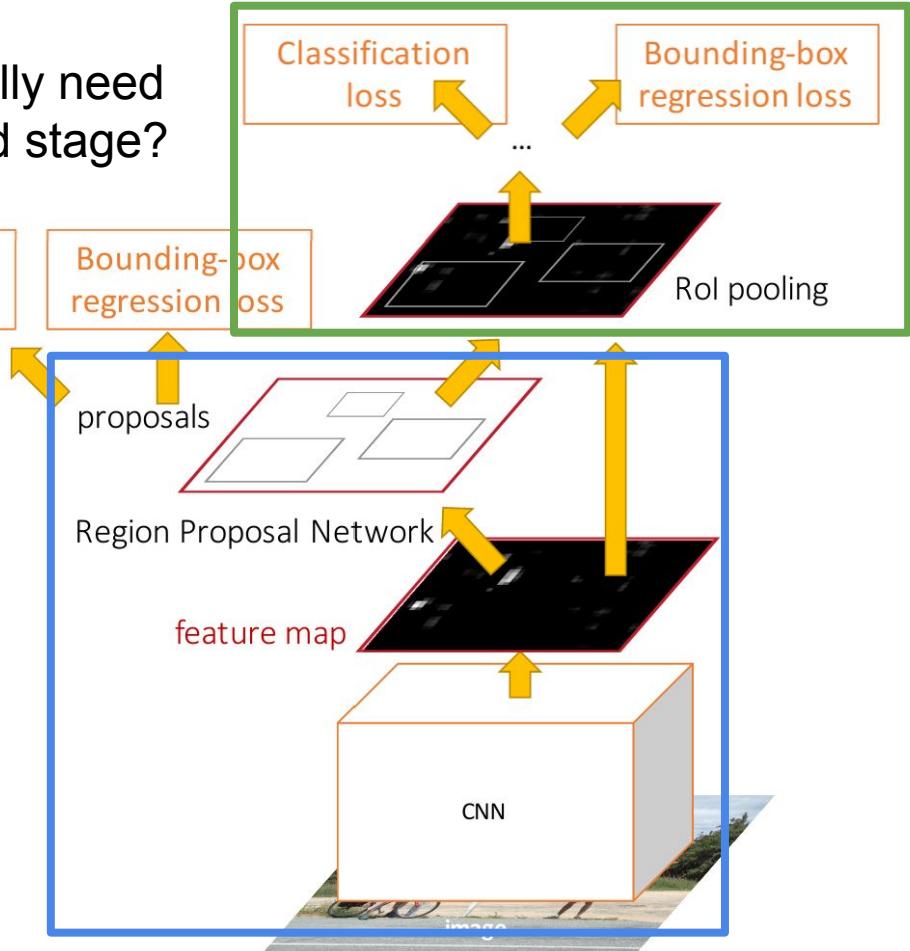
First stage: Run once per image

- Backbone network
- Region proposal network

Second stage: Run once per region

- Crop features: RoI pool / align
- Predict object class
- Prediction bbox offset

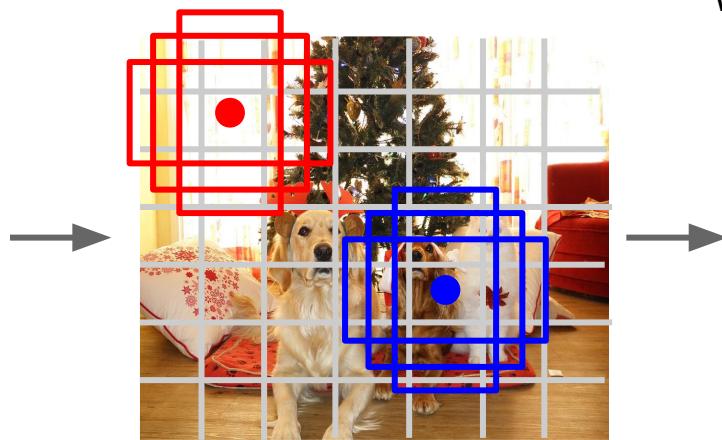
Do we really need
the second stage?



Single-Stage Object Detectors: YOLO / SSD / RetinaNet



Input image
 $3 \times H \times W$



Divide image into grid
 7×7

Image a set of **base boxes**
centered at each grid cell
Here $B = 3$

Within each grid cell:

- Regress from each of the B base boxes to a final box with 5 numbers:
(dx , dy , dh , dw , confidence)
- Predict scores for each of C classes (including background as a class)
- Looks a lot like RPN, but category-specific!

Output:

$$7 \times 7 \times (5 * B + C)$$

Redmon et al, "You Only Look Once: Unified, Real-Time Object Detection", CVPR 2016
Liu et al, "SSD: Single-Shot MultiBox Detector", ECCV 2016
Lin et al, "Focal Loss for Dense Object Detection", ICCV 2017

Object Detection: Lots of variables ...

Backbone Network
VGG16
ResNet-101
Inception V2
Inception V3
Inception
ResNet
MobileNet

“Meta-Architecture”
Two-stage: Faster R-CNN
Single-stage: YOLO / SSD
Hybrid: R-FCN

Image Size
Region Proposals
...

Takeaways
Faster R-CNN is slower but more accurate
SSD is much faster but not as accurate
Bigger / Deeper backbones work better

Huang et al, “Speed/accuracy trade-offs for modern convolutional object detectors”, CVPR 2017

R-FCN: Dai et al, “R-FCN: Object Detection via Region-based Fully Convolutional Networks”, NIPS 2016

Inception-V2: Ioffe and Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, ICML 2015

Inception V3: Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”, arXiv 2016

Inception ResNet: Szegedy et al, “Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning”, arXiv 2016

MobileNet: Howard et al, “Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv 2017

Object Detection: Lots of variables ...

Backbone Network
VGG16
ResNet-101
Inception V2
Inception V3
Inception
ResNet
MobileNet

“Meta-Architecture”
Two-stage: Faster R-CNN
Single-stage: YOLO / SSD
Hybrid: R-FCN

Image Size
Region Proposals
...

Takeaways
Faster R-CNN is slower but more accurate
SSD is much faster but not as accurate
Bigger / Deeper backbones work better

Huang et al, “Speed/accuracy trade-offs for modern convolutional object detectors”, CVPR 2017

Zou et al, “Object Detection in 20 Years: A Survey”, arXiv 2019

R-FCN: Dai et al, “R-FCN: Object Detection via Region-based Fully Convolutional Networks”, NIPS 2016

Inception-V2: Ioffe and Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift”, ICML 2015

Inception V3: Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”, arXiv 2016

Inception ResNet: Szegedy et al, “Inception-V4, Inception-ResNet and the Impact of Residual Connections on Learning”, arXiv 2016

MobileNet: Howard et al, “Efficient Convolutional Neural Networks for Mobile Vision Applications”, arXiv 2017

Instance Segmentation

Classification



CAT

No spatial extent

Semantic
Segmentation



GRASS, CAT,
TREE, SKY

No objects, just pixels

Object
Detection



DOG, DOG, CAT

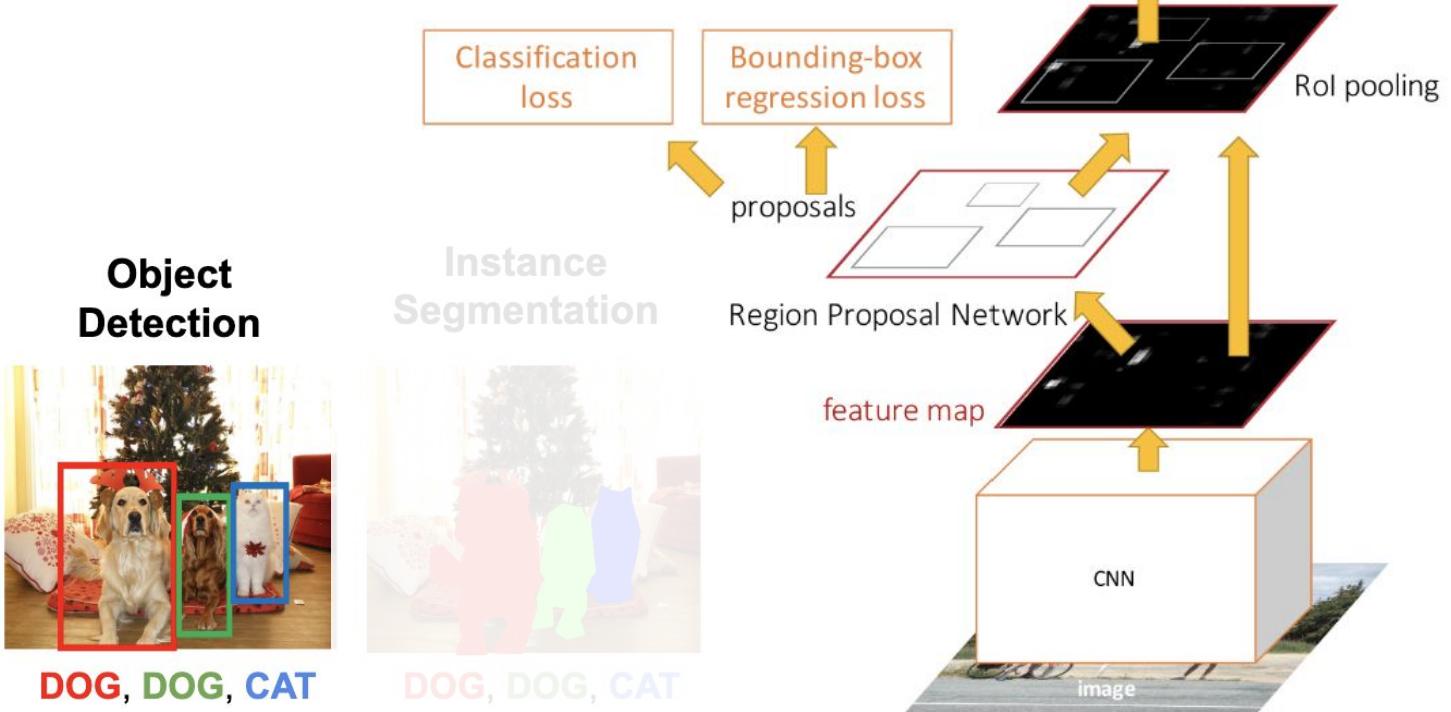
Multiple Object

Instance
Segmentation



DOG, DOG, CAT

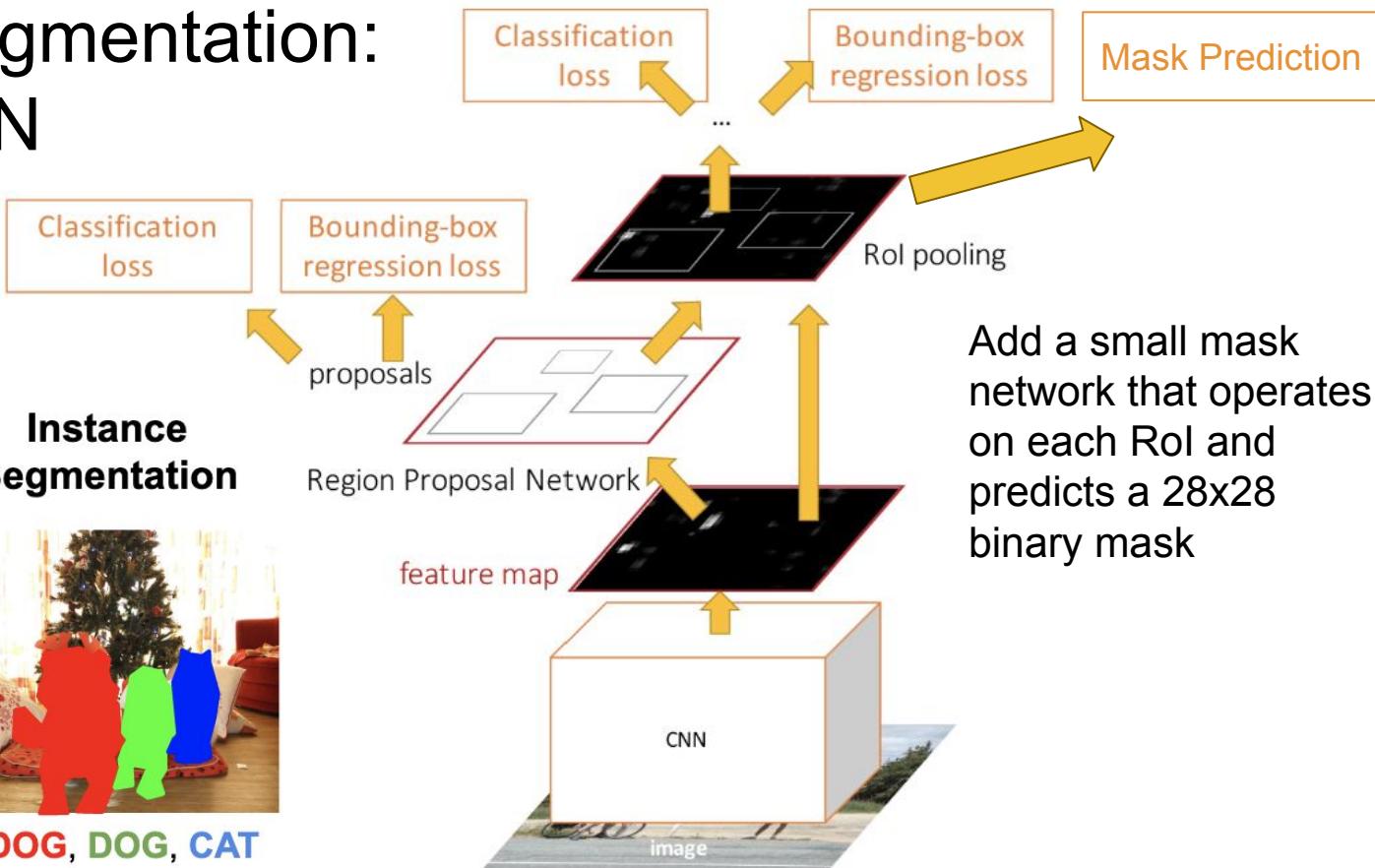
Object Detection: Faster R-CNN



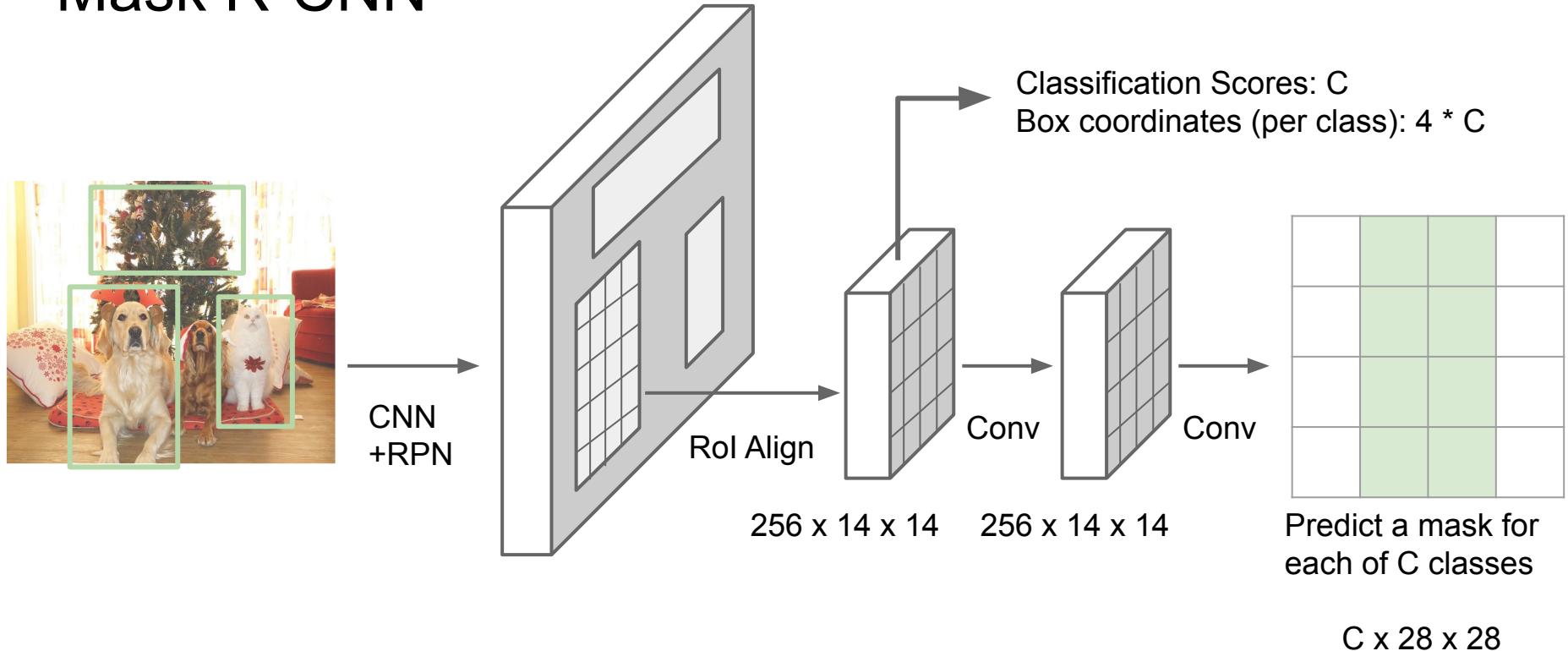
Instance Segmentation: Mask R-CNN



He et al, "Mask R-CNN", ICCV 2017

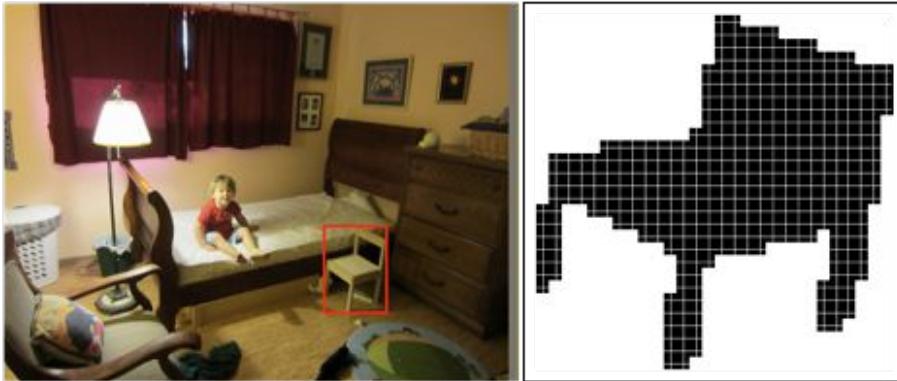


Mask R-CNN

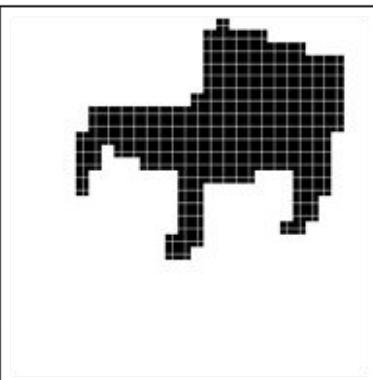


He et al, "Mask R-CNN", arXiv 2017

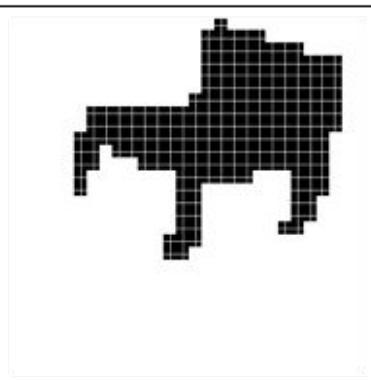
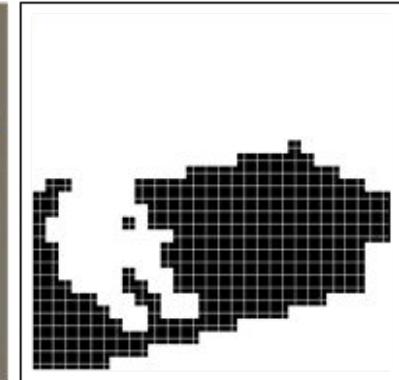
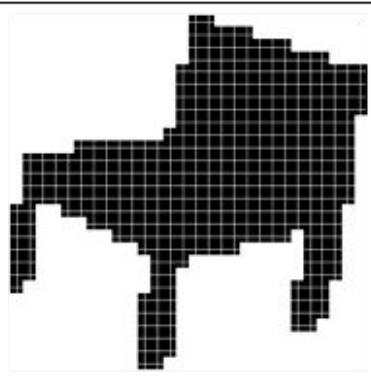
Mask R-CNN: Example Mask Training Targets



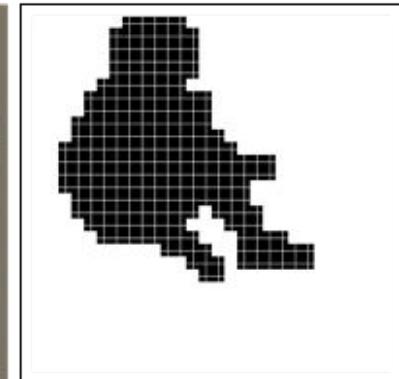
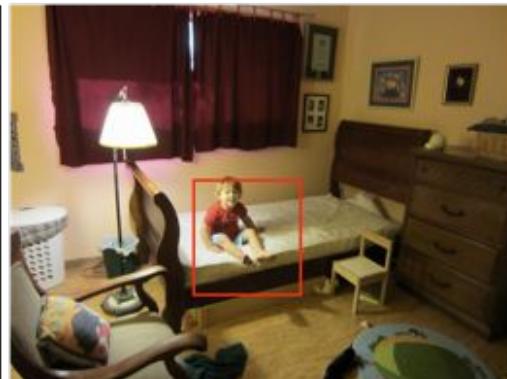
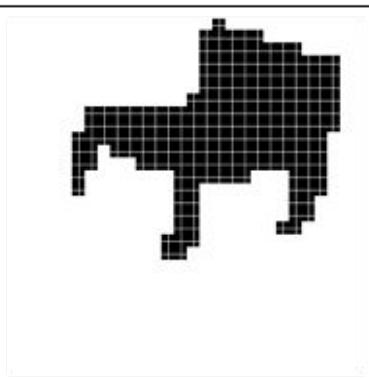
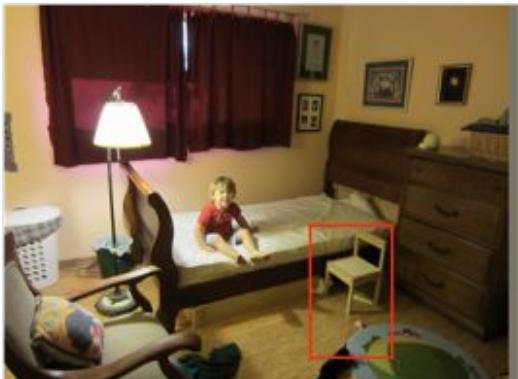
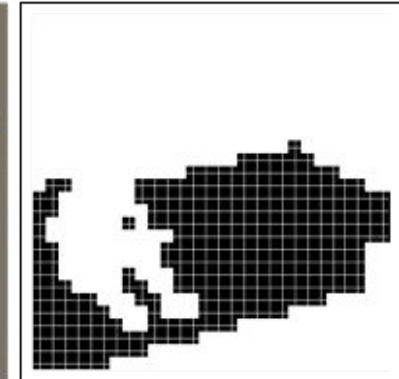
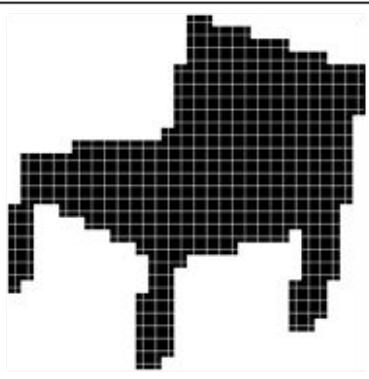
Mask R-CNN: Example Mask Training Targets



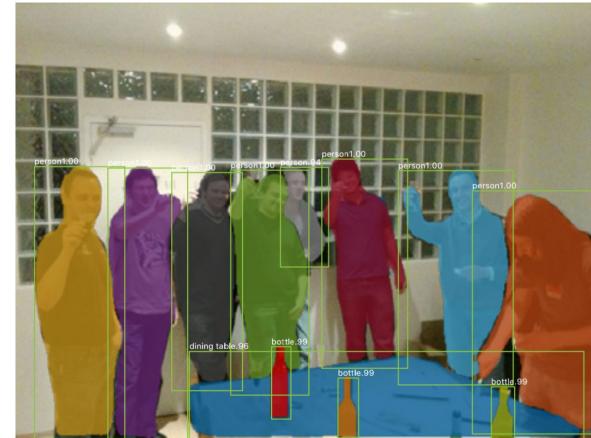
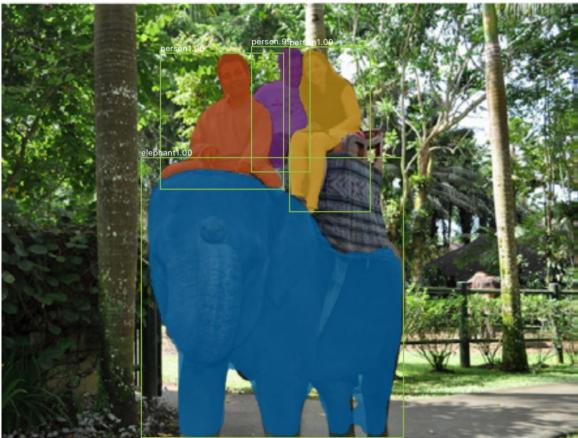
Mask R-CNN: Example Mask Training Targets



Mask R-CNN: Example Mask Training Targets



Mask R-CNN: Very Good Results!



He et al, "Mask R-CNN", ICCV 2017

Mask R-CNN

Also does pose



He et al, "Mask R-CNN", ICCV 2017

Open Source Frameworks

Lots of good implementations on GitHub!

TensorFlow Detection API:

https://github.com/tensorflow/models/tree/master/research/object_detection

Faster RCNN, SSD, RFCN, Mask R-CNN, ...

Detectron2 (PyTorch)

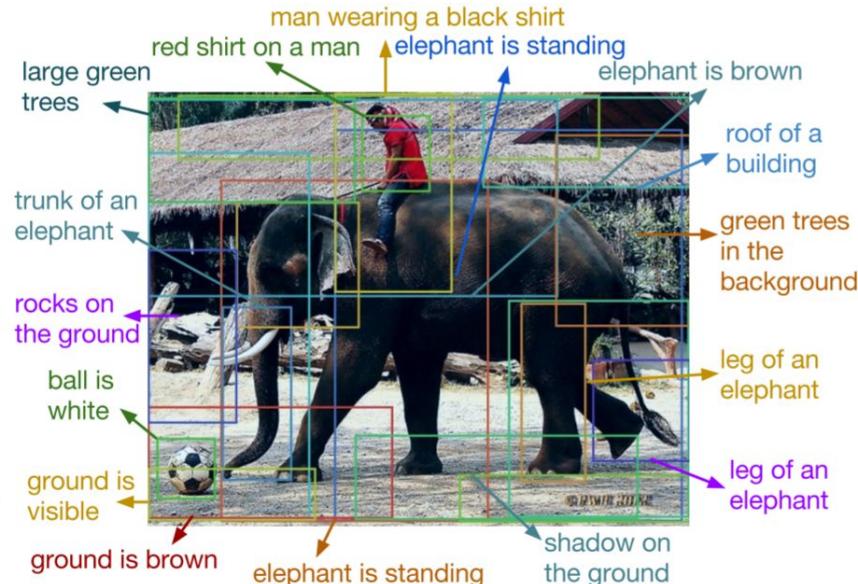
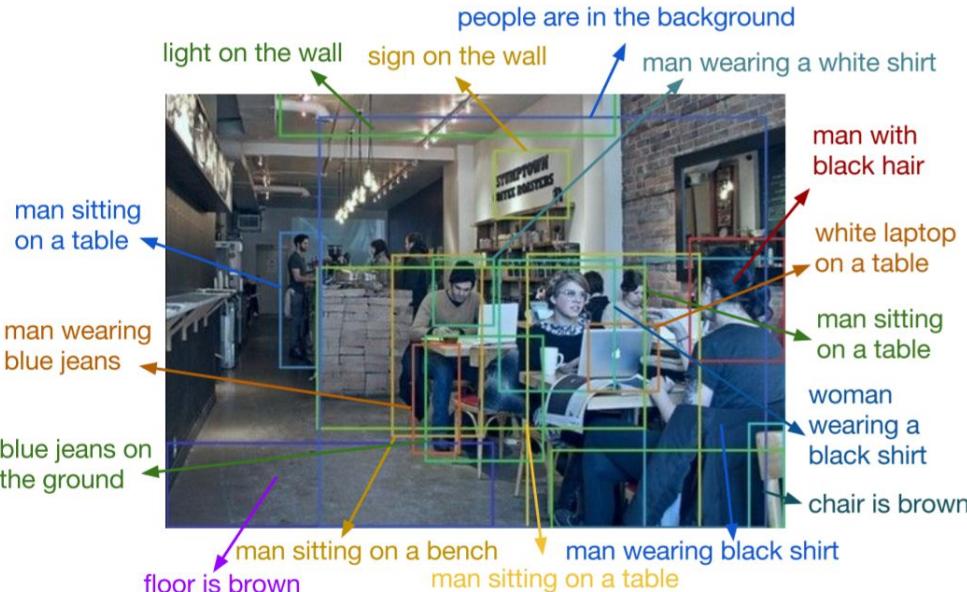
<https://github.com/facebookresearch/detectron2>

Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, R-FCN, ...

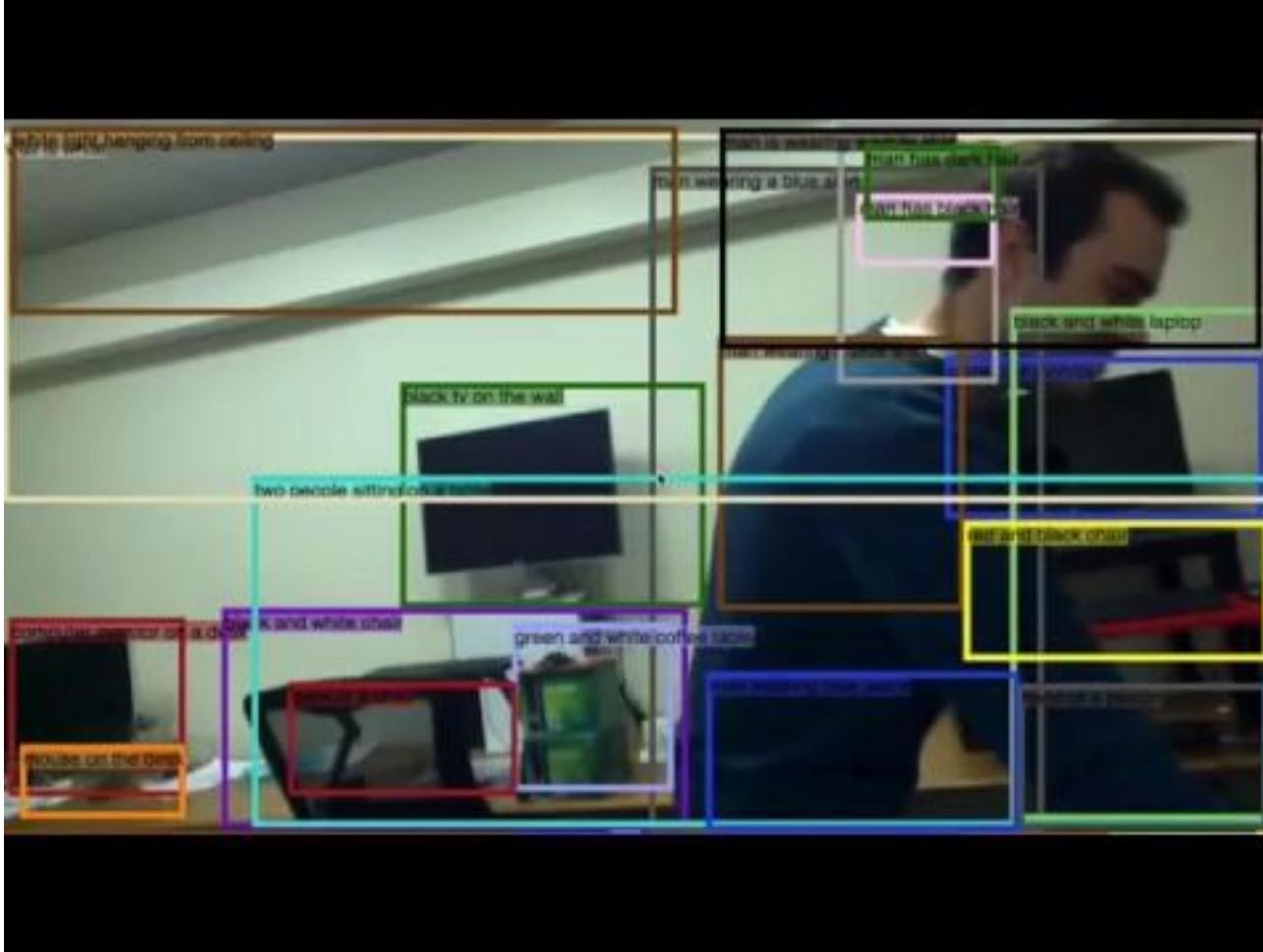
Finetune on your own dataset with pre-trained models

Beyond 2D Object Detection...

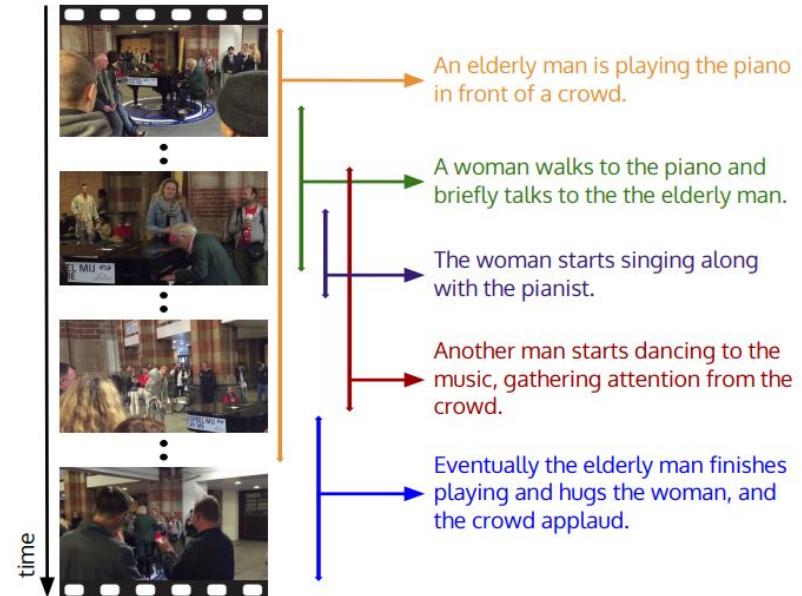
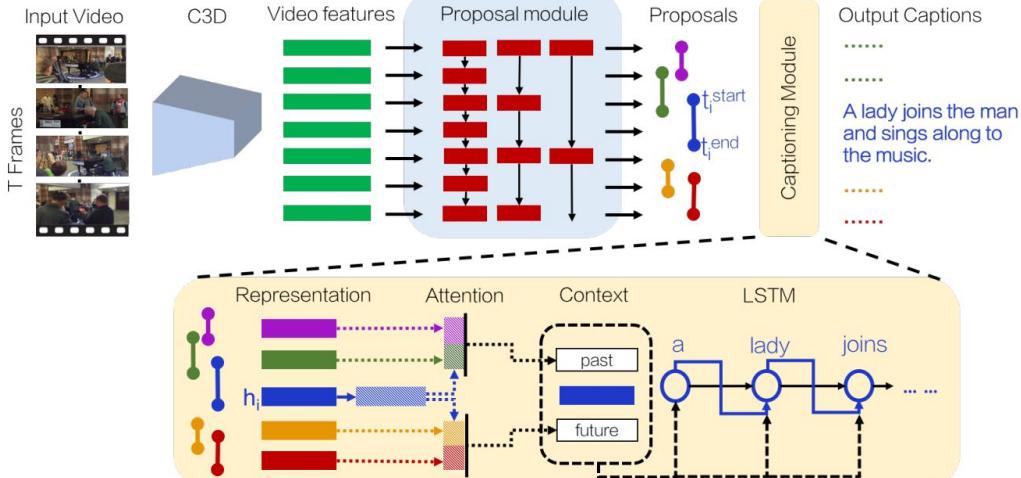
Object Detection + Captioning = Dense Captioning



Johnson, Karpathy, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016
Figure copyright IEEE, 2016. Reproduced for educational purposes.

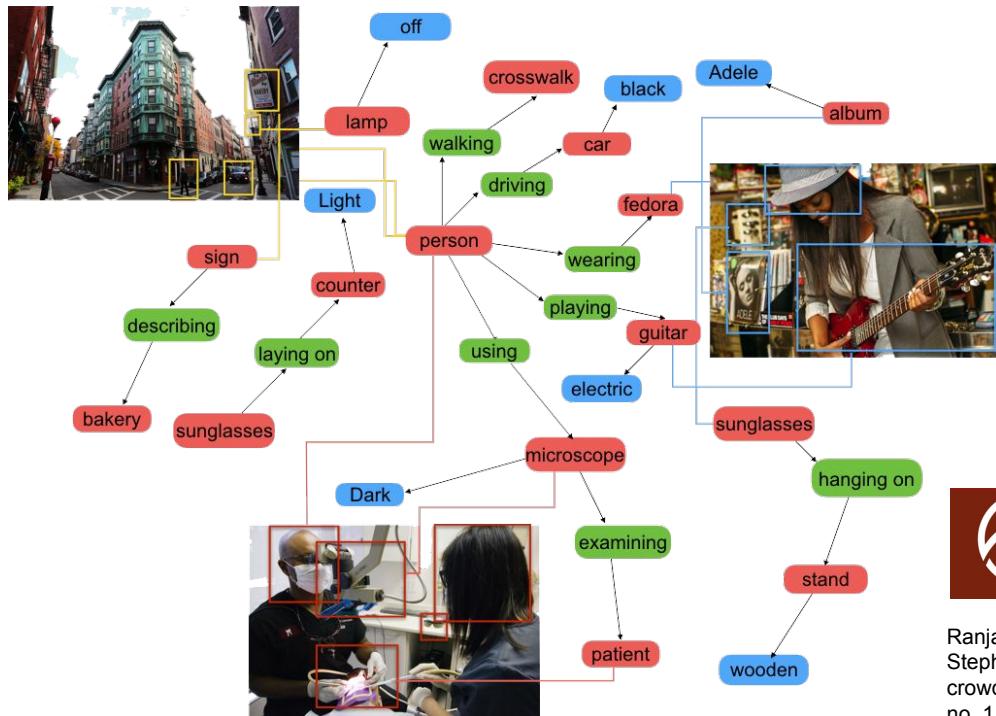


Dense Video Captioning



Ranjay Krishna et al., "Dense-Captioning Events in Videos", ICCV 2017
Figure copyright IEEE, 2017. Reproduced with permission.

Objects + Relationships = Scene Graphs



108,077 Images

next to

5.4 Million Region Descriptions

1.7 Million Visual Question Answers

3.8 Million Object Instances

2.8 Million Attributes

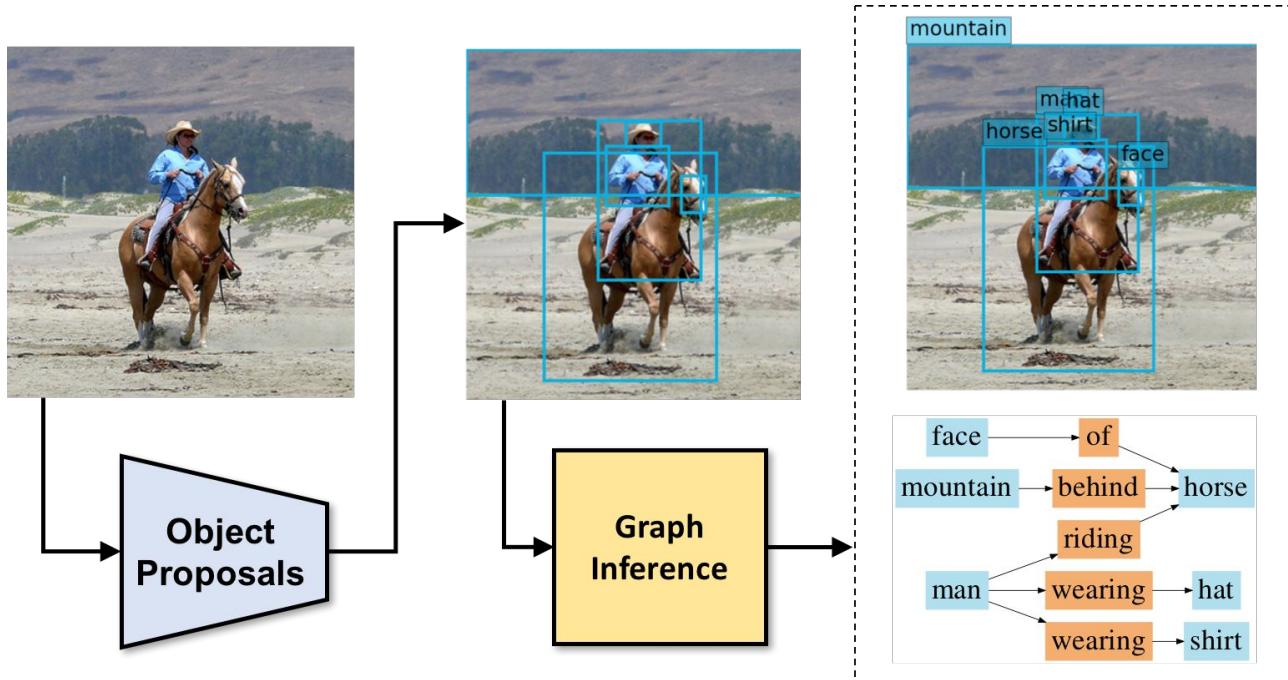
2.3 Million Relationships

Everything Mapped to Wordnet Synsets

 VISUALGENOME

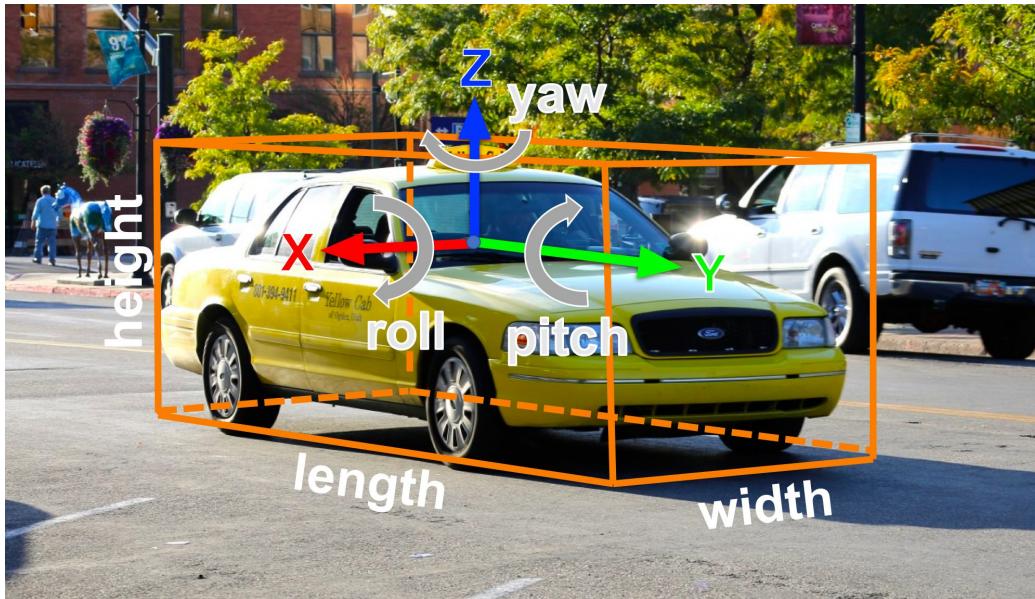
Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." International Journal of Computer Vision 123, no. 1 (2017): 32-73.

Scene Graph Prediction



Xu, Zhu, Choy, and Fei-Fei, "Scene Graph Generation by Iterative Message Passing", CVPR 2017
Figure copyright IEEE, 2018. Reproduced for educational purposes.

3D Object Detection



2D Object Detection:

2D bounding box

(x, y, w, h)

3D Object Detection:

3D oriented bounding box

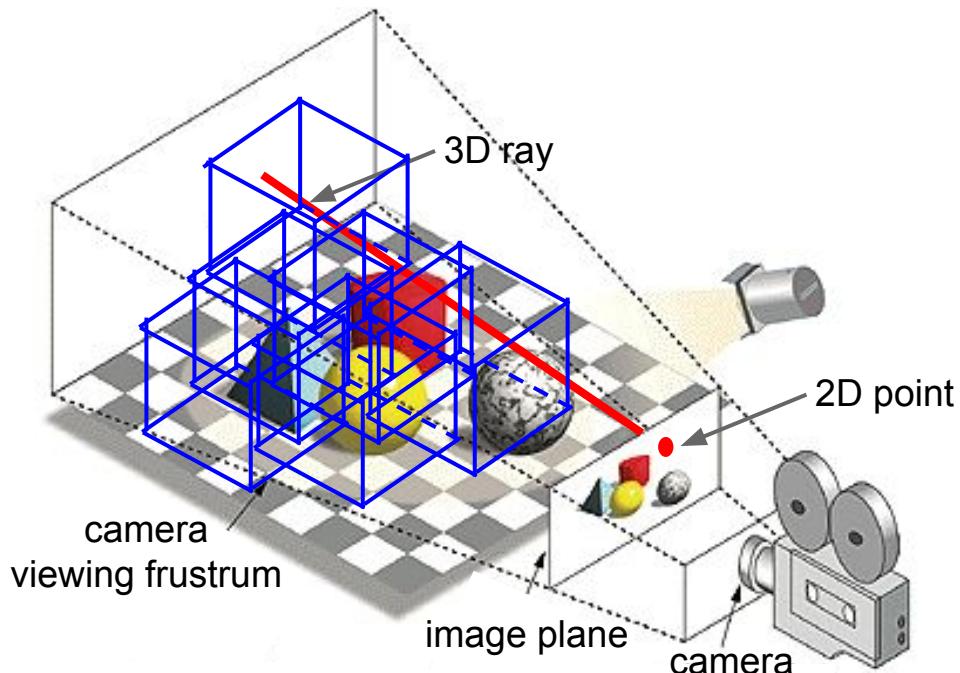
$(x, y, z, w, h, l, r, p, y)$

Simplified bbox: no roll & pitch

Much harder problem than 2D object detection!

[This image](#) is CC0 public domain

3D Object Detection: Simple Camera Model



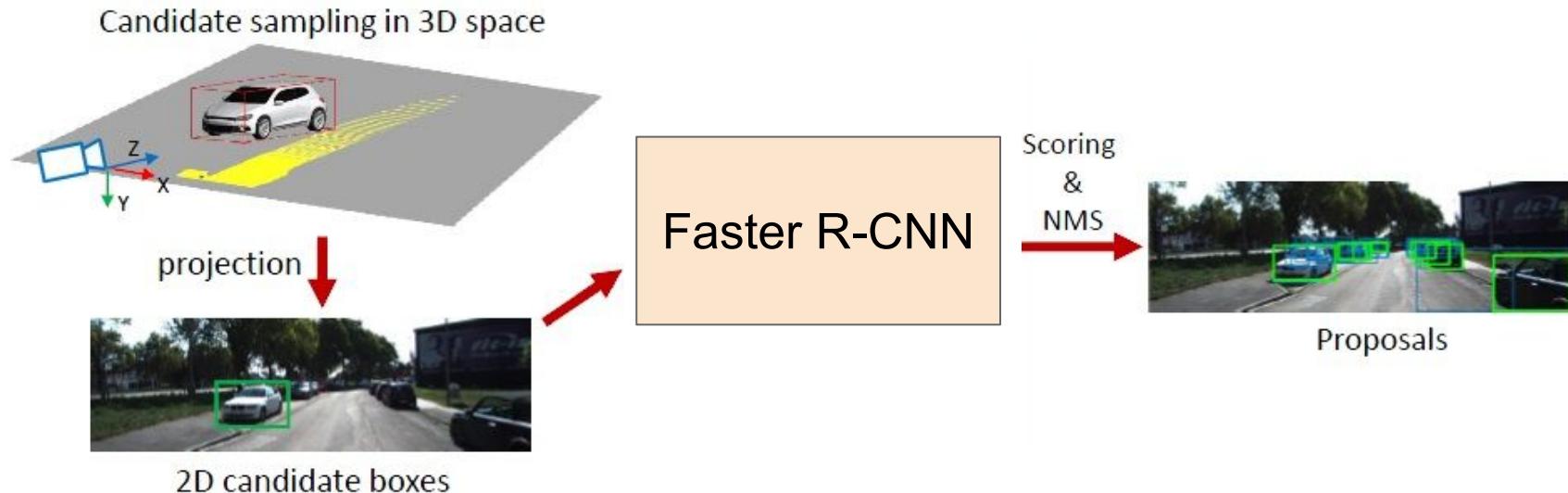
A point on the image plane corresponds to a **ray** in the 3D space

A 2D bounding box on an image is a **frustrum** in the 3D space

Localize an object in 3D:
The object can be anywhere in the **camera viewing frustum!**

Image source: https://www.pcmag.com/encyclopedia_images/_FRUSTUM.GIF

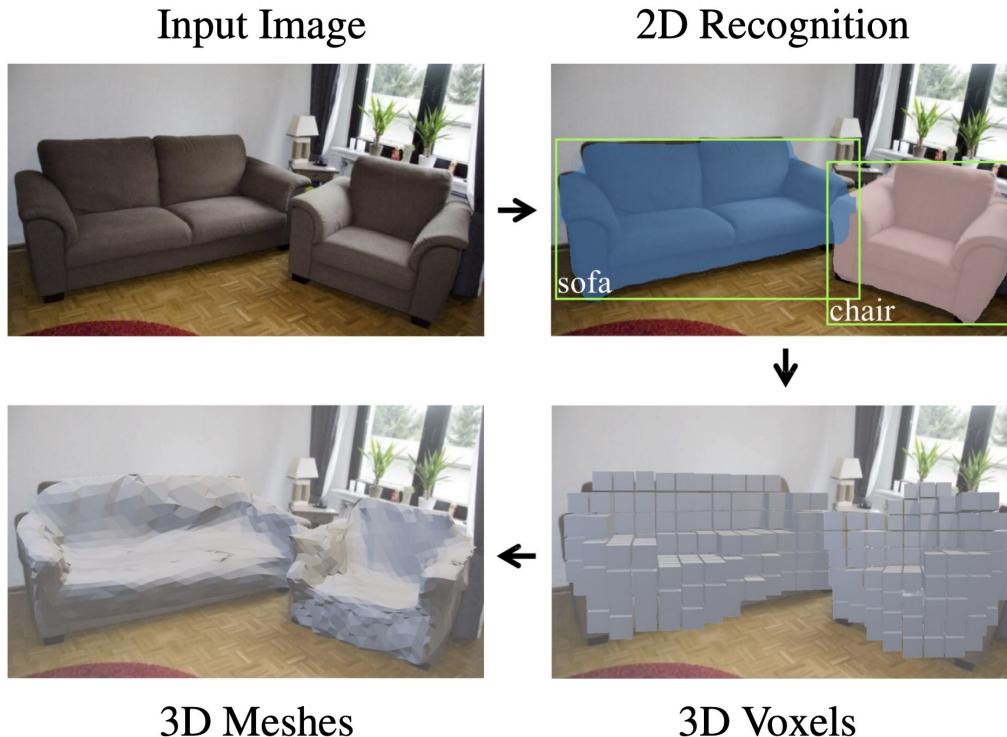
3D Object Detection: Monocular Camera



- Same idea as Faster RCNN, but proposals are in 3D
- 3D bounding box proposal, regress 3D box parameters + class score

Chen, Xiaozhi, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. "Monocular 3d object detection for autonomous driving." CVPR 2016.

3D Shape Prediction: Mesh R-CNN



Gkioxari et al., Mesh RCNN, ICCV 2019

Recap: Lots of computer vision tasks!

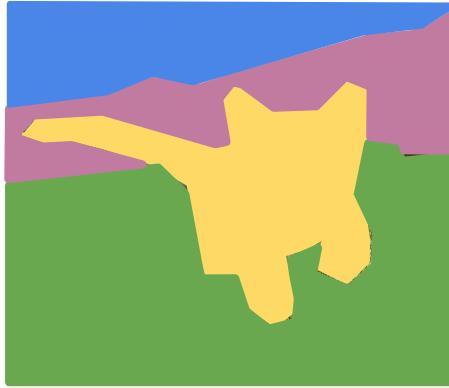
Classification



CAT

No spatial extent

Semantic Segmentation



GRASS, CAT,
TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Multiple Object

Instance Segmentation



DOG, DOG, CAT

This image is CC0 public domain

Next time: Recurrent Neural Networks

Object Detection & RNNs

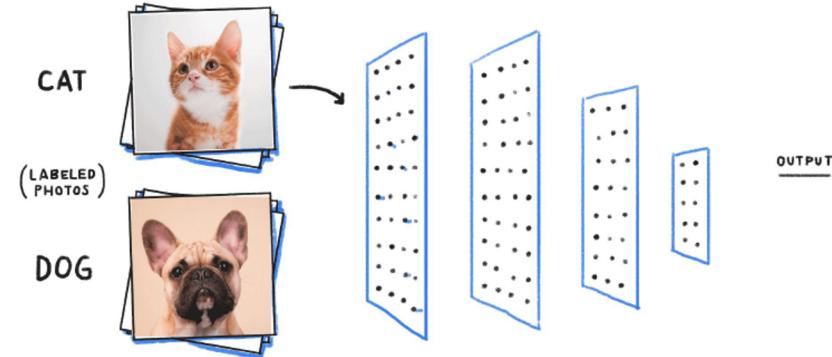
Zhuoyi Huang

Partial slides credit to JunYoung Gwak

1. Object Detection

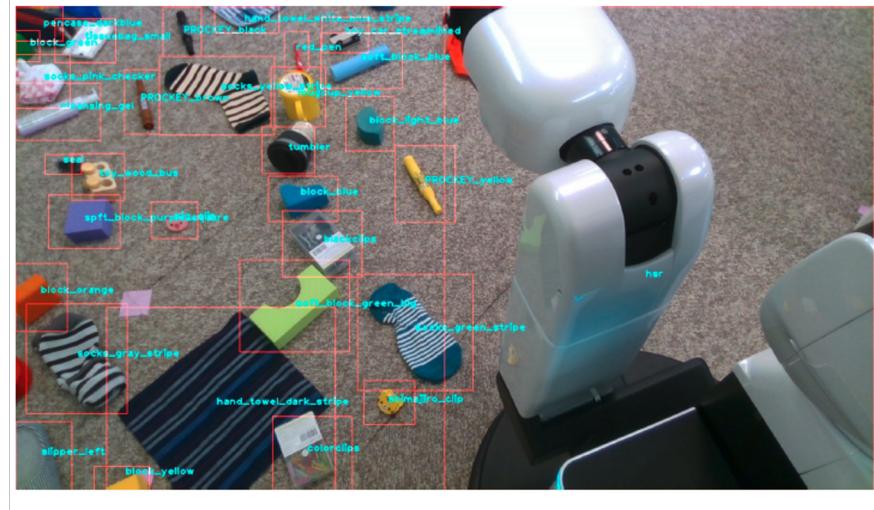
Motivation

- **Image classification:** often assume there is a single object in the image



Motivation

- **Image classification:** often assume there is a single object in the image
 - Real-world images can include multiple instances of objects with the same/different classes



Motivation

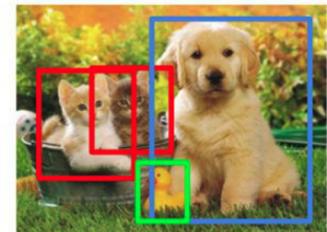
- **Image classification:** often assume there is a single object in the image
- Real-world images can include multiple instances of objects with the same/different classes
- **Object Detection:** produce bounding boxes that surround each instance

Classification



CAT

Object Detection



CAT, DOG, DUCK

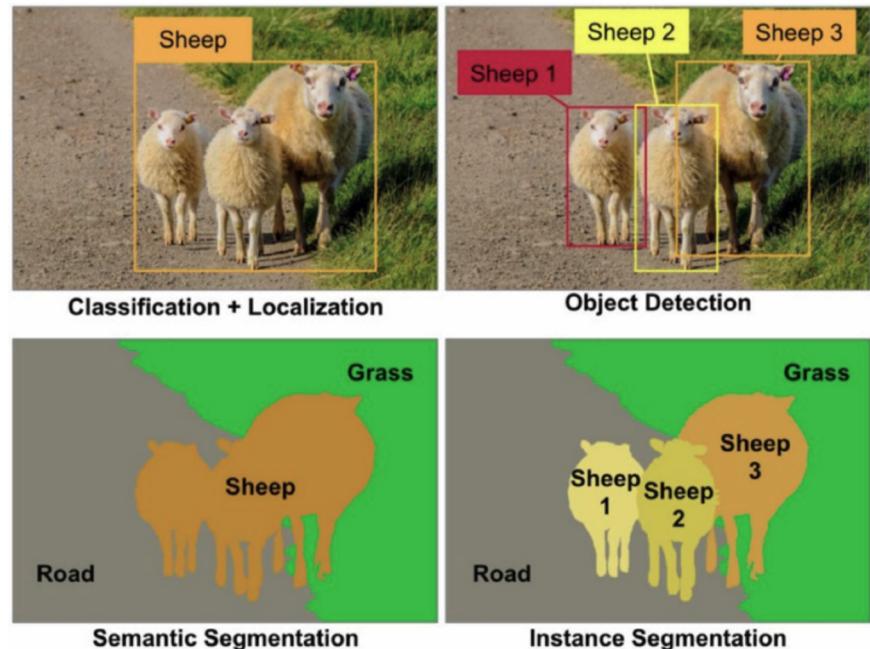
Problem Definition: Object Detection

Object Detection

- Input: Image
- Output: multiple **instances** of
 - object location (bounding box)
 - object class

Instance

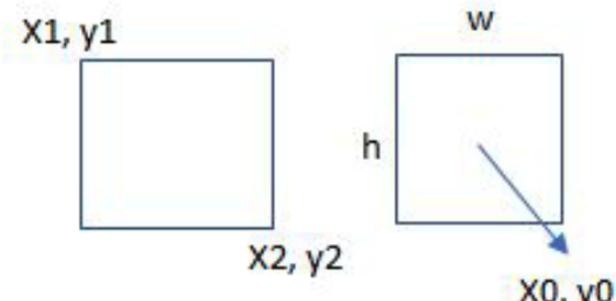
- Distinguishes individual objects, in contrast to considering them as a single semantic class



Problem Definition: Object Detection

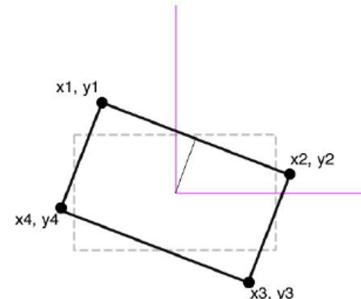
Object Detection

- Input: Image
- Output: multiple instances of
 - object location (bounding box)
 - object class



Bounding box

- Rigid box that confines the instance
- Multiple possible parametrizations
 - (width, height, center x, center y)
 - (x_1, y_1, x_2, y_2)
 - $(x_1, y_1, x_2, y_2, \text{rotation})$



Problem Definition: Object Detection

Object Detection

- Input: Image
- Output: multiple instances of
 - object location (bounding box)
 - object class

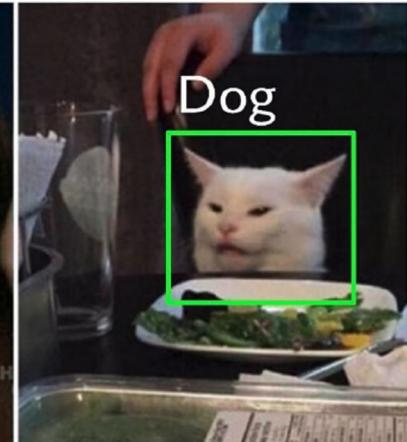
Object class

- Semantic class of the instance
 - Similar to image classification
 - Predict a vector of scores

People that say
that AI will take
over the world:



My own AI:

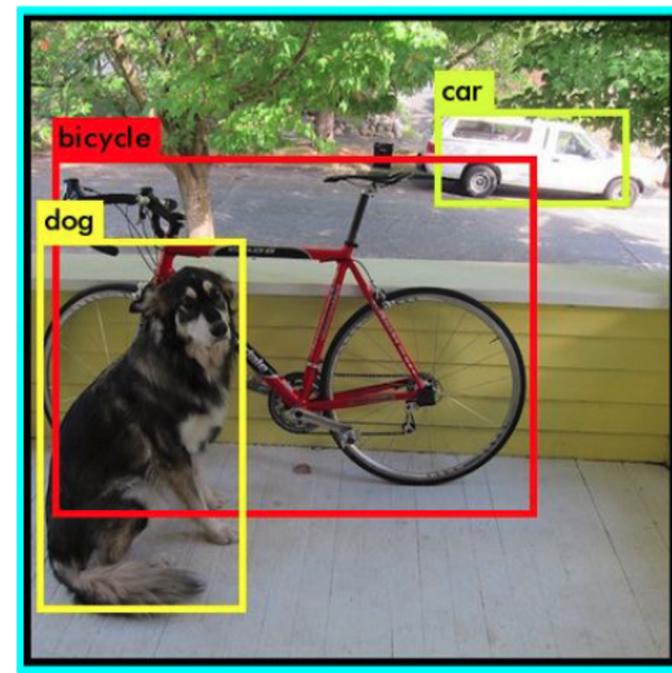


Modern Object Detection Architecture

- R-CNN
- Fast R-CNN
- Faster R-CNN
- Mask R-CNN
- SSD
- YOLO (v1, v2, v3, v4)
- FPN
- DETR

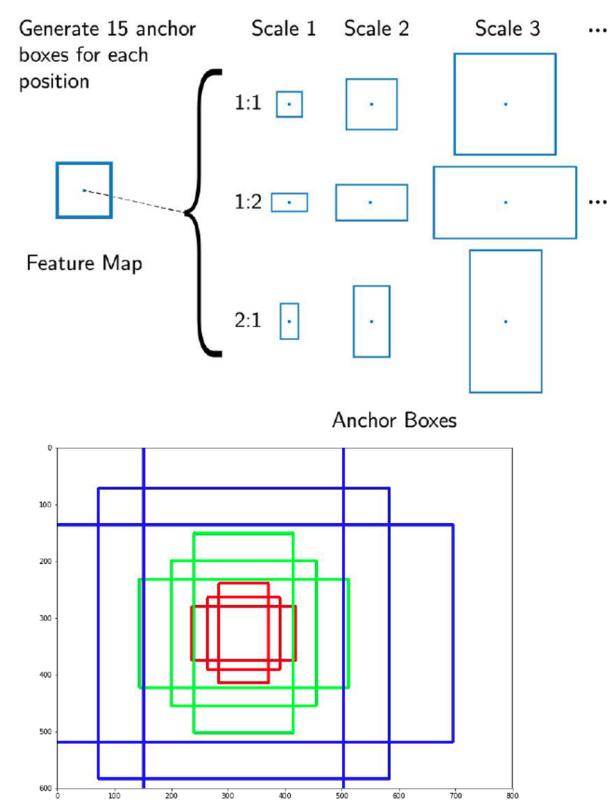
Object Detection: how can we detect multiple instances?

- Boxes can be centered at any location in the image
- Varying width/height
- Sliding window: infeasible



Object Detection: Anchor Boxes!

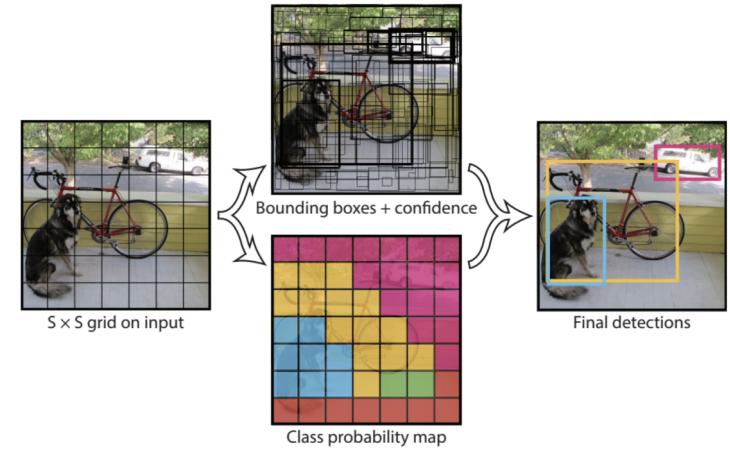
- Neural network prefers discrete prediction over continuous regression
- Preselect templates of bounding boxes to alleviate the regression problem
- For each anchor box, NN decides
 - Does it contain an object? (objectness classification)
 - Small refinement to the box (object localization)



Object Detection: Single-Stage and Two-Stage Architectures

Stage 1

- For every output pixels
 - For every anchor boxes
 - Predict bounding box offsets
 - Predict anchor confidence (objectness/class)
- Output
 - Bounding boxes if single-stage
 - Region proposals (region-of-interest, RoI) if two-stage



Stage 2

- For RoI
 - Perform pooling using the RoI (RoI pooling)
 - Predict bounding box offsets
 - Predict object class

Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

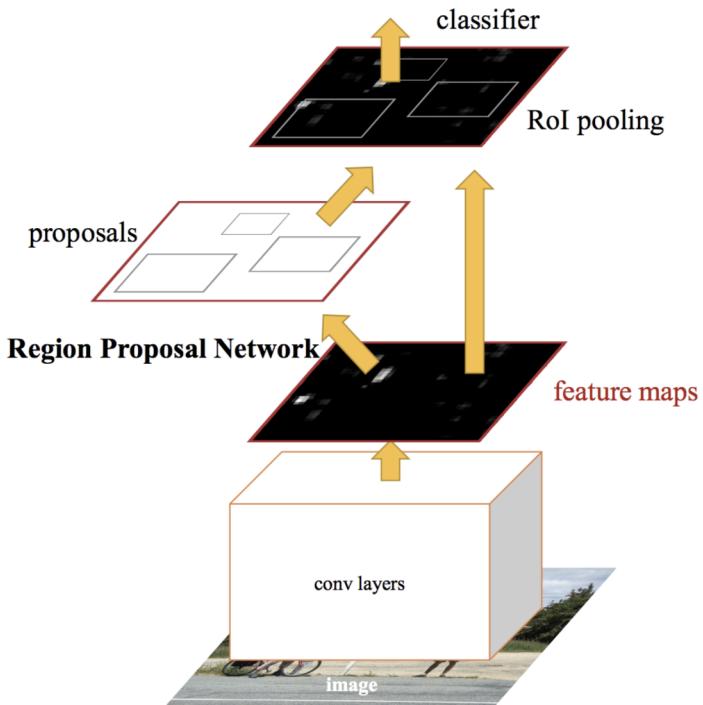
Object Detection: Single-Stage and Two-Stage Architectures

Stage 1

- For every output pixels
 - For every anchor boxes
 - Predict bounding box offsets
 - Predict anchor confidence (objectness/class)
- Output
 - Bounding boxes if single-stage
 - Region proposals (region-of-interest, RoI) if two-stage

Stage 2

- For RoI
 - Perform pooling using the RoI (RoI pooling)
 - Predict bounding box offsets



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *arXiv preprint arXiv:1506.01497* (2015).

Object Detection: Single-Stage vs Two-Stage Architectures

- Single-Stage
 - + Faster
 - - Can perform worse on small objects due to the low resolution of feature maps
- Two-Stage
 - + Performance is often higher
 - + Easily extendable to various instance-based tasks
 - - Slow

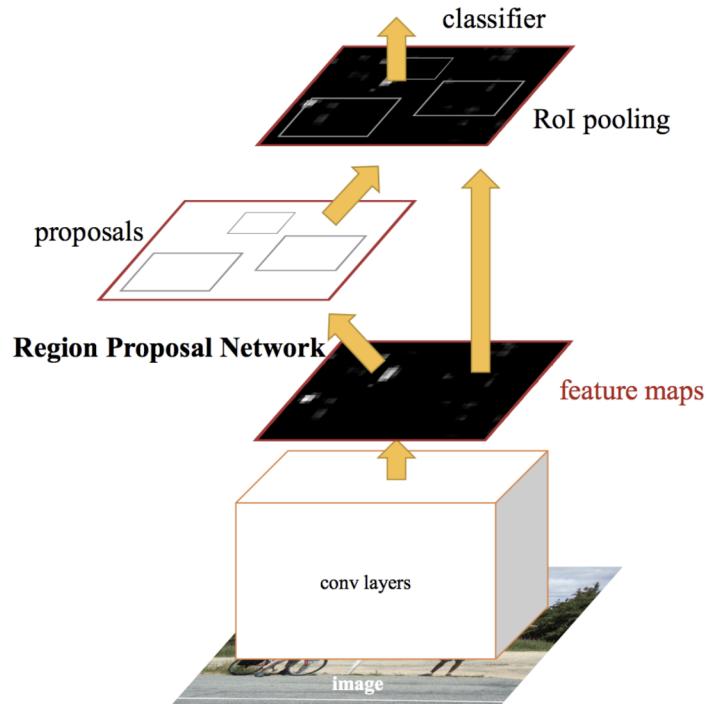
Details for Two-Stage Object Detectors

Stage 1

- For every output pixels
 - For every anchor boxes
 - Predict bounding box offsets
 - Predict anchor confidence (objectness/class)
- Output
 - Region proposals (region-of-interest, RoI)

Stage 2

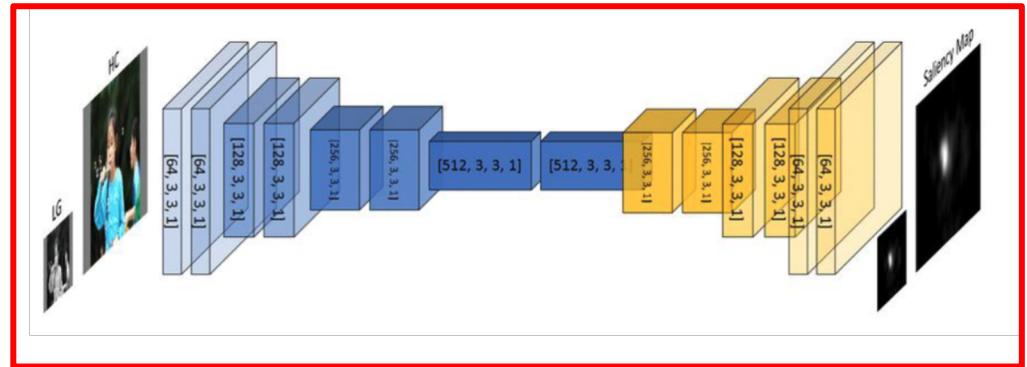
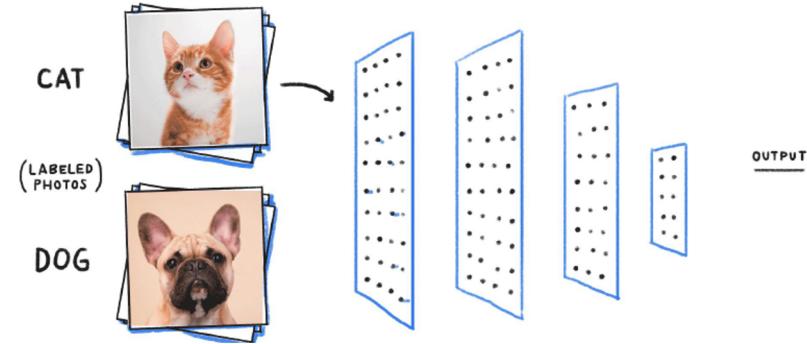
- For RoI
 - Perform pooling using the RoI (RoI pooling)
 - Predict bounding box offsets
 - Predict object class



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *arXiv preprint arXiv:1506.01497* (2015).

Feature extractor

- Every pixel makes prediction
- Image classification: single output



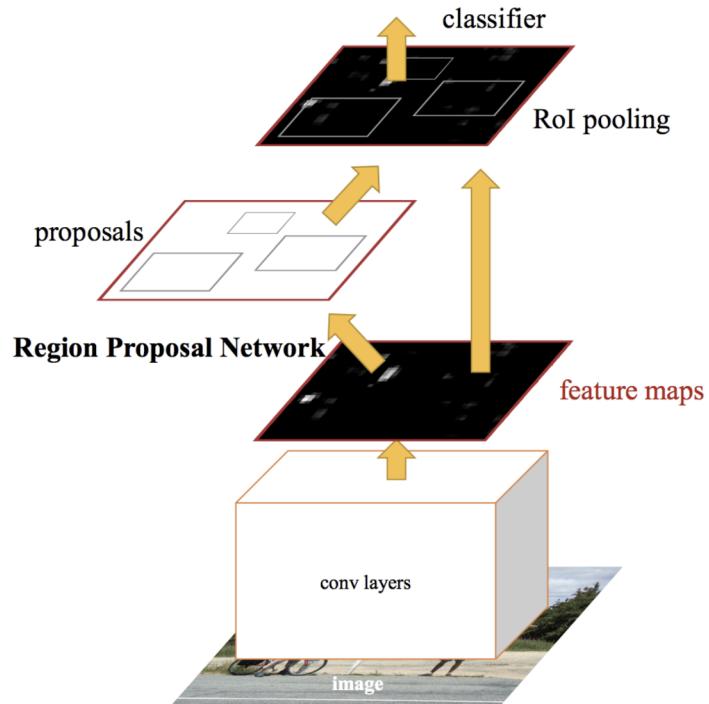
Details for Two-Stage Object Detectors

Stage 1

- For every output pixels
 - For every anchor boxes
 - Predict bounding box offsets
 - Predict anchor confidence (objectness/class)
- Output
 - Region proposals (region-of-interest, RoI)

Stage 2

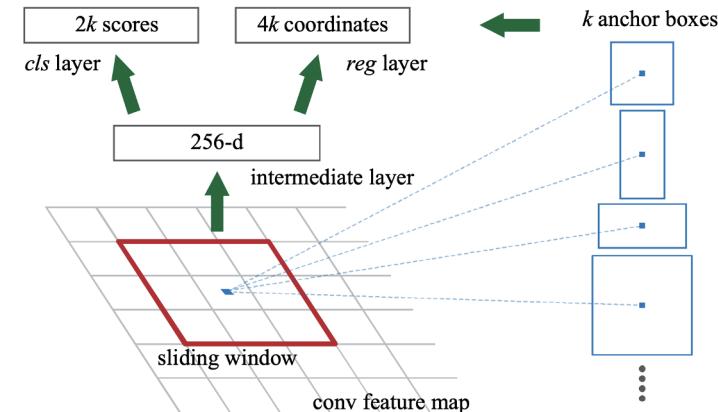
- For RoI
 - Perform pooling using the RoI (RoI pooling)
 - Predict bounding box offsets
 - Predict object class



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *arXiv preprint arXiv:1506.01497* (2015).

Extract Anchor Boxes

- For each output pixel
 - "Objectness" classification
 - Regression
- Often thousands of anchors for an image
- Pass anchors that correspond to ground-truth locations to the next stage, plus negative anchors



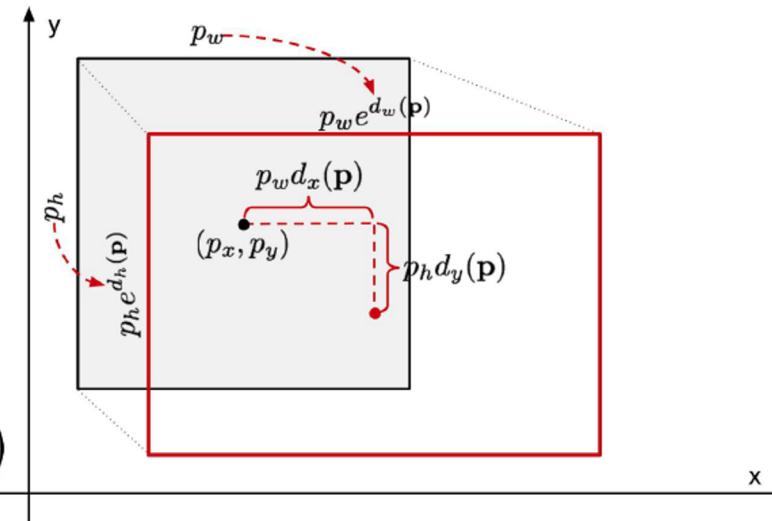
Bounding Box Regression

Given

- Anchor box size (p_w, p_h)
- Output pixel center location (p_x, p_y)

Predict bounding box refinement toward b

- Log-scaled scale relative ratio
 $d_w = \log(b_w/p_w), d_h = \log(b_h/p_h)$
- Relative center offset
 $d_x = (b_x - p_x)/p_w, d_y = (b_y - p_y)/p_h$



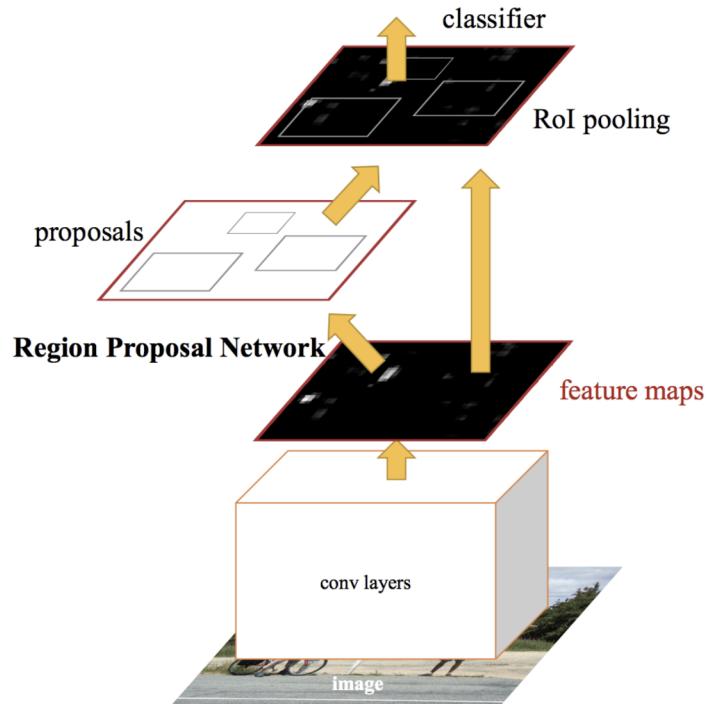
Details for Two-Stage Object Detectors

Stage 1

- For every output pixels
 - For every anchor boxes
 - Predict bounding box offsets
 - Predict anchor confidence (objectness/class)
- Output
 - Region proposals (region-of-interest, RoI)

Stage 2

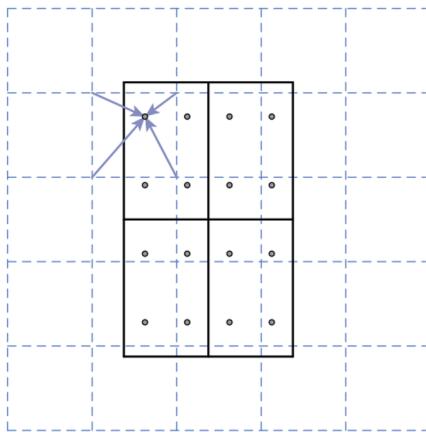
- For each RoI
 - **Perform pooling using the RoI (RoI pooling)**
 - Predict bounding box offsets
 - Predict object class



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *arXiv preprint arXiv:1506.01497* (2015).

RoI Pooling

- Given region-of-interests (RoIs), we want to pool from the backbone features



0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

0.8	0.6
0.9	0.6

0.1	0.3	0.2	0.3	0.2	0.6	0.8	0.9
0.4	0.5	0.1	0.4	0.7	0.1	0.4	0.3
0.2	0.1	0.3	0.8	0.6	0.2	0.1	0.1
0.4	0.6	0.2	0.1	0.3	0.6	0.1	0.2
0.1	0.8	0.3	0.3	0.5	0.3	0.3	0.3
0.2	0.9	0.4	0.5	0.1	0.1	0.1	0.2
0.3	0.1	0.8	0.6	0.3	0.3	0.6	0.5
0.5	0.5	0.2	0.1	0.1	0.2	0.1	0.2

0.88	0.6
0.9	0.6

He, Kaiming, et al. "Mask r-cnn." *Proceedings of the IEEE international conference on computer vision*. 2017.

<https://jonathan-hui.medium.com/image-segmentation-with-mask-r-cnn-ebe6d793272>

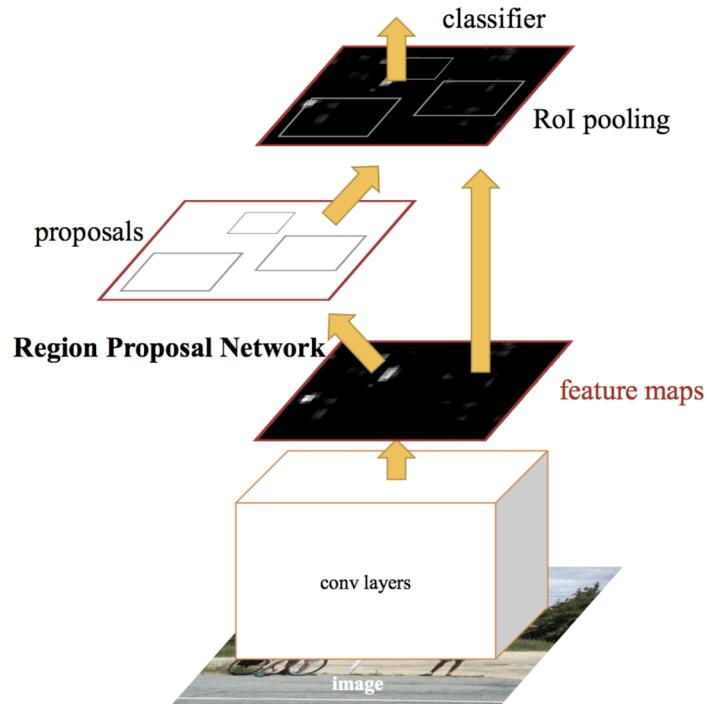
Details for Two-Stage Object Detectors

Stage 1

- For every output pixels
 - For every anchor boxes
 - Predict bounding box offsets
 - Predict anchor confidence (objectness/class)
- Output
 - Region proposals (region-of-interest, RoI)

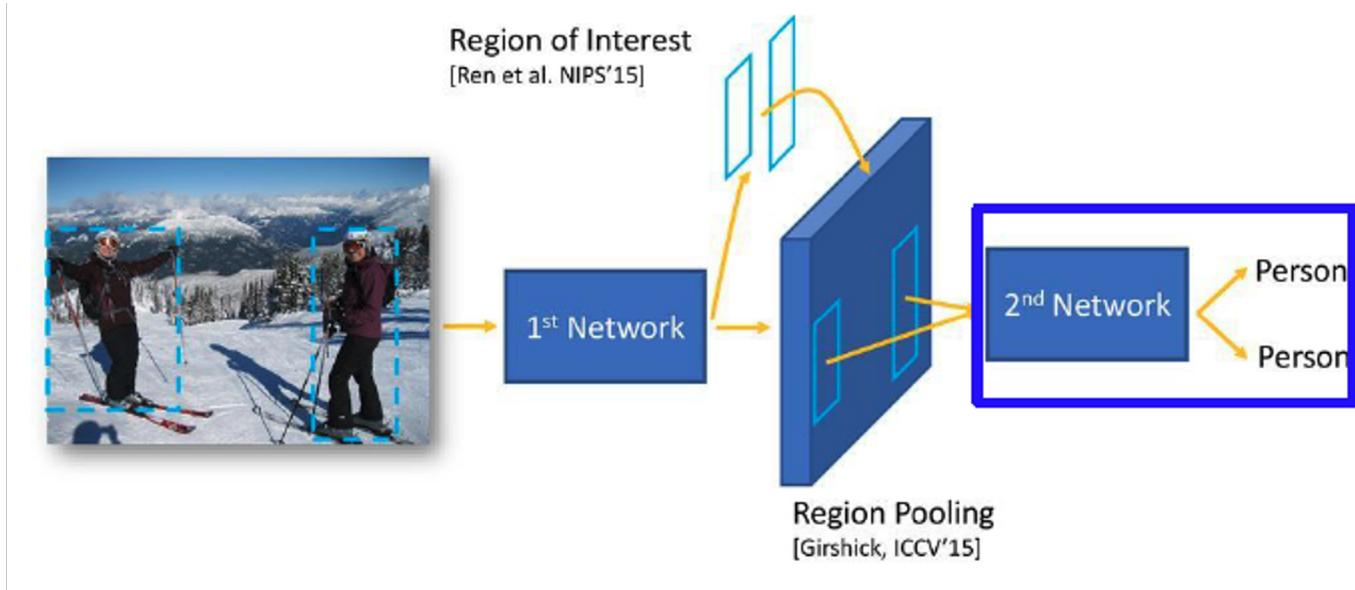
Stage 2

- For each RoI
 - Perform pooling using the RoI (RoI pooling)
 - **Predict bounding box offsets**
 - **Predict object class (semantic class / background)**



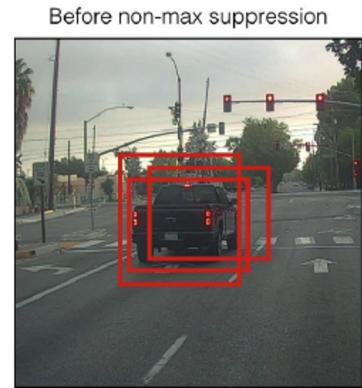
Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *arXiv preprint arXiv:1506.01497* (2015).

Details for Two-Stage Object Detectors



Are We Done?

- Prediction might contain multiple boxes of the same instance

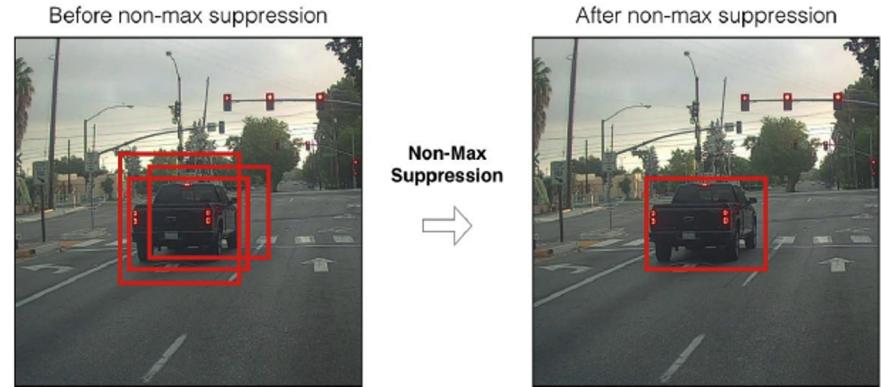


Non-Max
Suppression
→



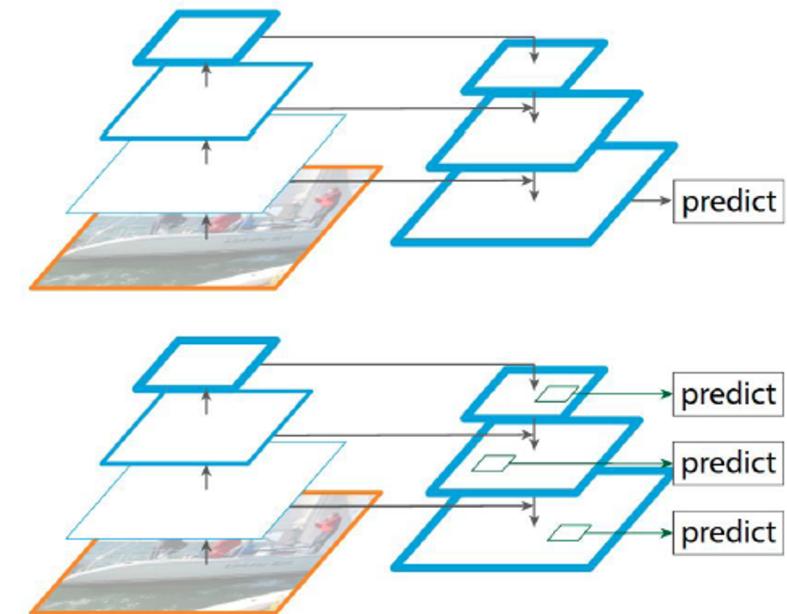
Post-Processing: Non-Maximum Suppression

- For boxes overlapping with each other above a threshold: keep the one with the maximum confidence score
- Implementation
 - Sort by confidence
 - For each box (conf high to low)
 - If overlaps with confirmed predictions above a threshold
 - Discard
 - Else
 - Add to predictions



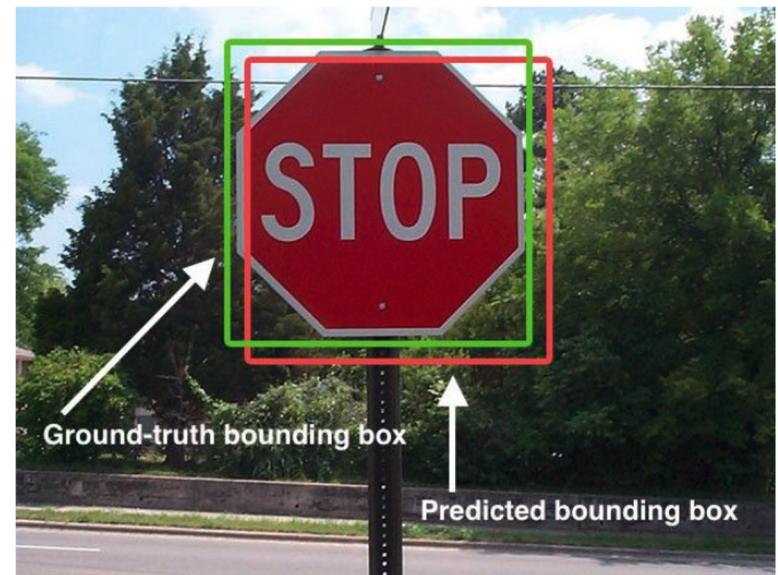
Feature Pyramid Network as the feature extractor

- Traditional backbone
 - Small feature maps have larger receptive field and contain better-extracted overall semantic information
 - Want this semantic information in larger feature maps for prediction
- Feature Pyramid Network
 - Richer representation
 - Enables multi-scale predictions



How should we evaluate our results?

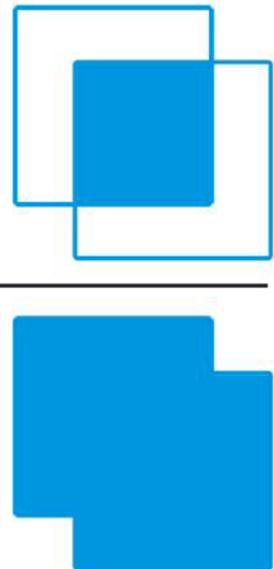
- Start with the most simple case
- Given
 - a single ground-truth box
 - a single predicted box



How should we evaluate our results?

- Start with the most simple case
- Given
 - a single ground-truth box
 - a single predicted box
- Use Intersection-over-Union (IoU)

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

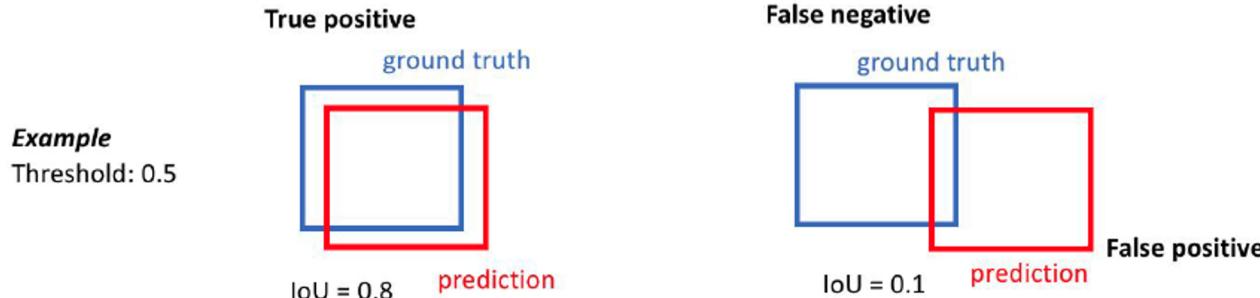


What if there are multiple boxes?

- Multiple ground-truth boxes
- Multiple predictions
- Might include
 - True positive (prediction matched with GT)
 - False positive (prediction not matched with any GT)
 - False negative (GT not matched with any prediction)

Bounding Box Matching

- Use IoU threshold
- Matched if
 - IoU above certain threshold
 - Class prediction is correct
 - GT not matched with other boxes (1-to-1)

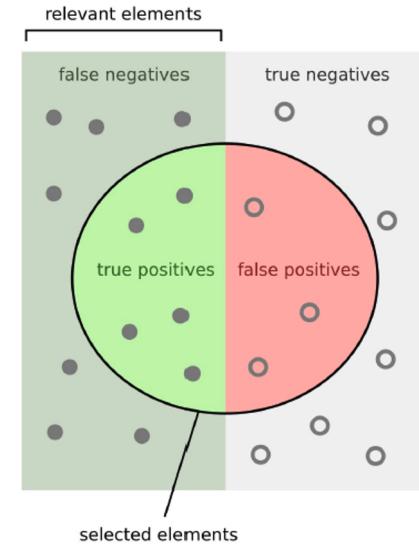


Evaluation Metrics: Precision and Recall

- True Positive (TP)
- False Negative (FN)
- False Positive (FP)

Precision = $TP / (TP + FP)$

Recall = $TP / (TP + FN)$



How many selected items are relevant?

$$\text{Precision} = \frac{\text{green}}{\text{red} + \text{green}}$$

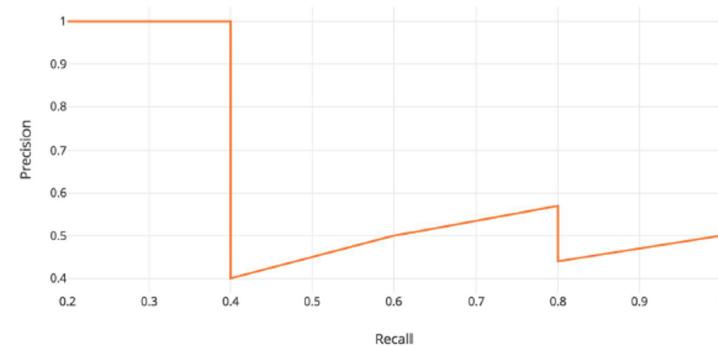
How many relevant items are selected?

$$\text{Recall} = \frac{\text{green}}{\text{green} + \text{light blue}}$$

Evaluation Metrics: Average Precision

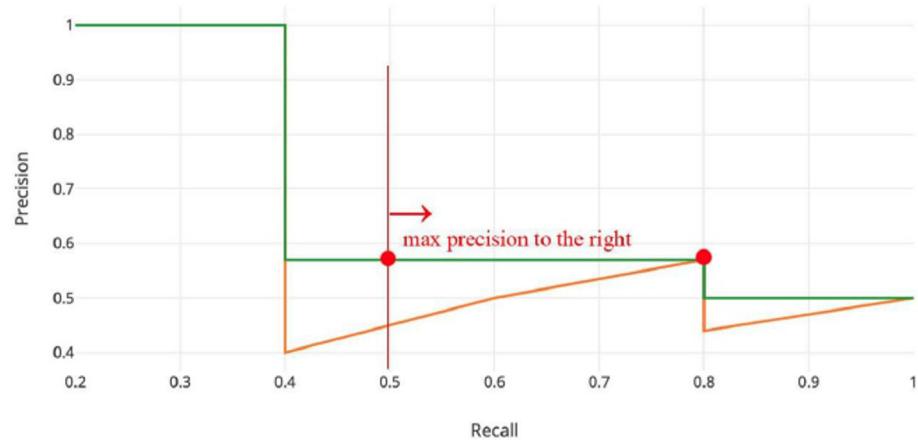
- Go through every prediction in descending order of the prediction confidence
- Plot Precision-Recall Curve
- Area below the curve is **Average Precision (AP)**

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0



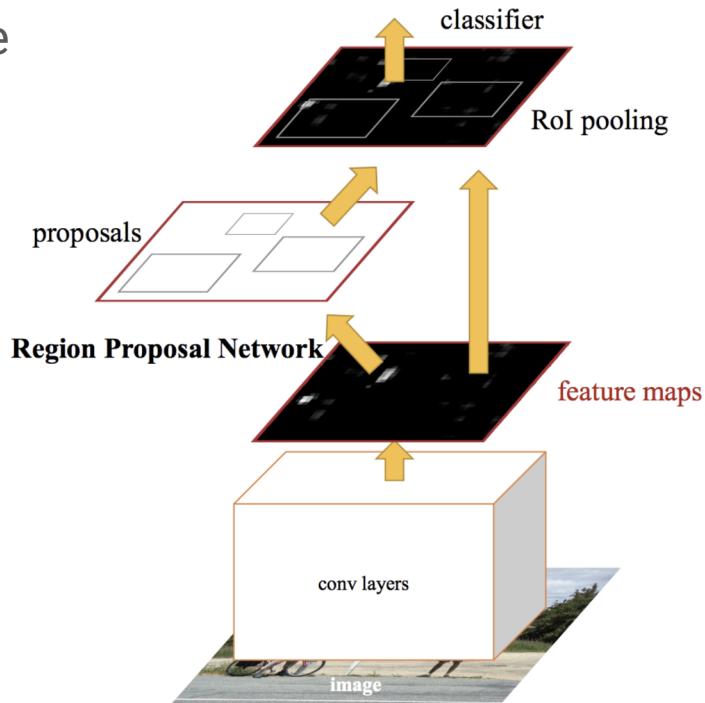
Evaluation Metrics: Average Precision

- To make AP more stable to score ordering, we sometimes take max precision to the right of the PR curve
- Use different IoU threshold for matching
 - AP50, AP75, AP95: match IoU threshold of 0.5, 0.75, 0.95
 - AP: average of AP with match IoU threshold of [0.5, 0.55, 0.6, ..., 0.95]



Two-Stage Detectors can do more!

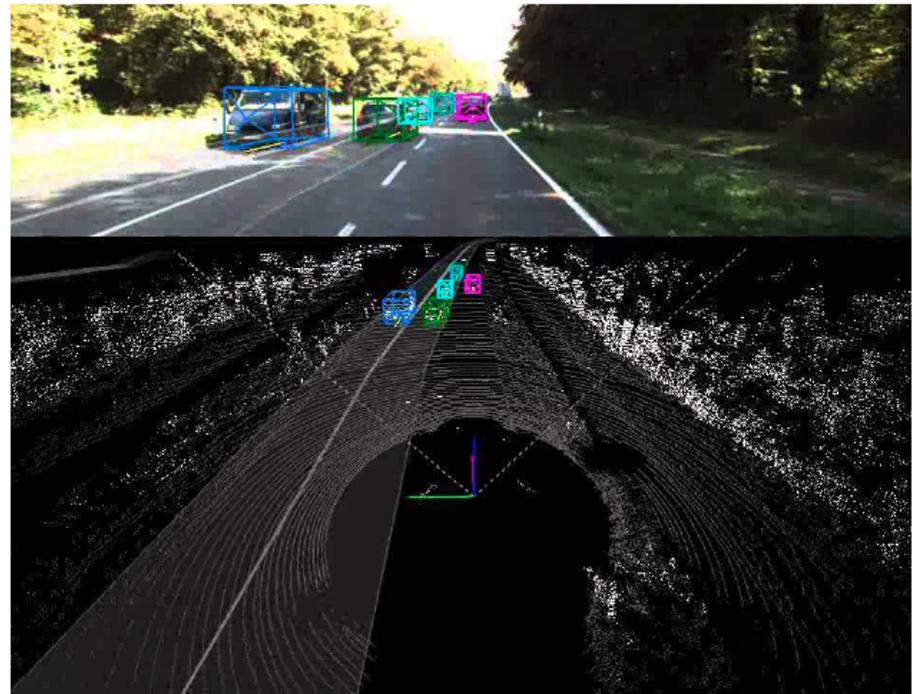
- In addition to detecting boxes, at the final stage using RoI features, we can predict
 - 3D bounding boxes
 - Instance segmentation
 - Keypoints (human pose)
 - Meshes
 - Scene graphs
 - ...
- A family of R-CNNs!



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *arXiv preprint arXiv:1506.01497* (2015).

3D Object Detection

- Input
 - 2D image and/or 3D point cloud
- Output
 - 3D bounding box
 - Center location: x, y, z
 - Bounding box size: w, h, l
 - Rotation



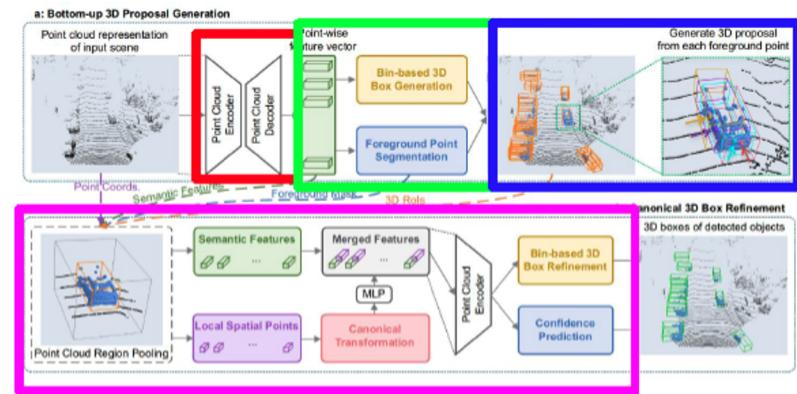
3D Object Detection

Stage 1

- For every output pixel (from backbone)
 - For every anchor boxes
 - Predict bounding box offsets
 - Predict anchor confidence

(Optional, if two-stage networks) Stage 2

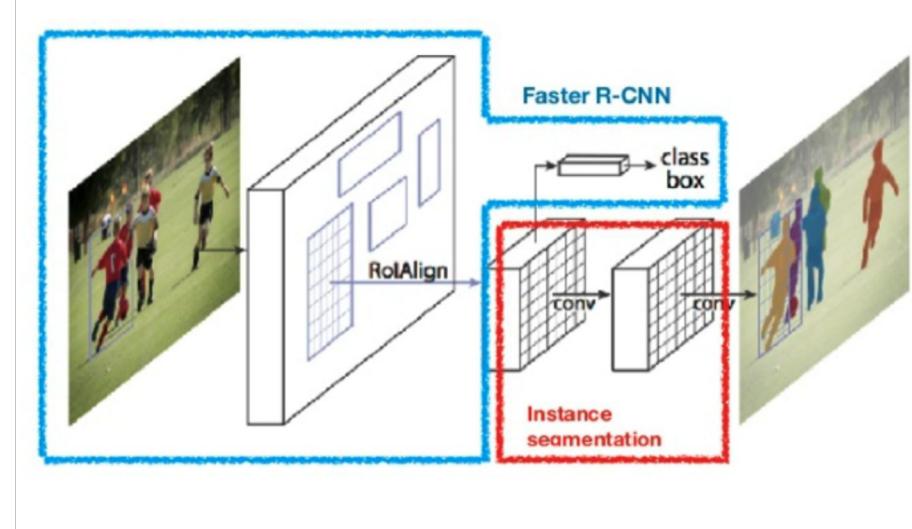
- For every region proposals
 - Predict bounding box offsets
 - Predict its semantic class



For example,
Point R-CNN

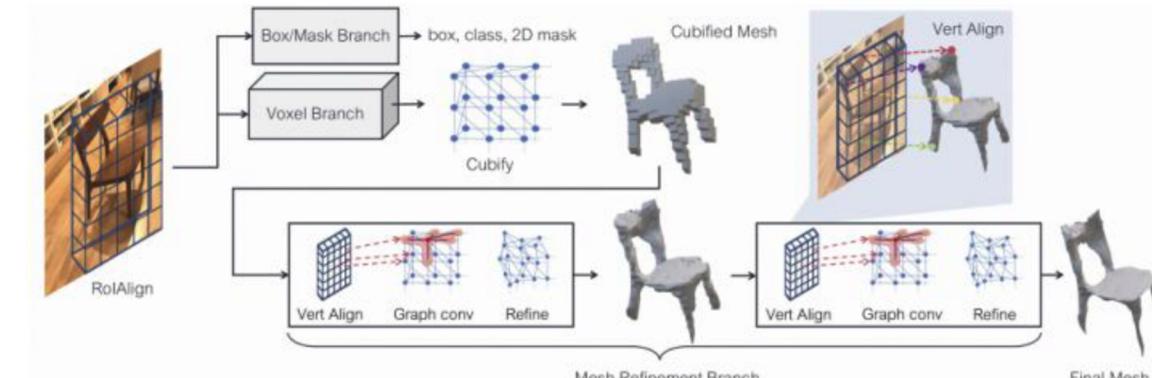
Mask R-CNN

- Final stage parallel to box prediction
 - Predict instance mask using a convolution head
- ROI Align especially helpful for segmentation by aggregating fine-grained features



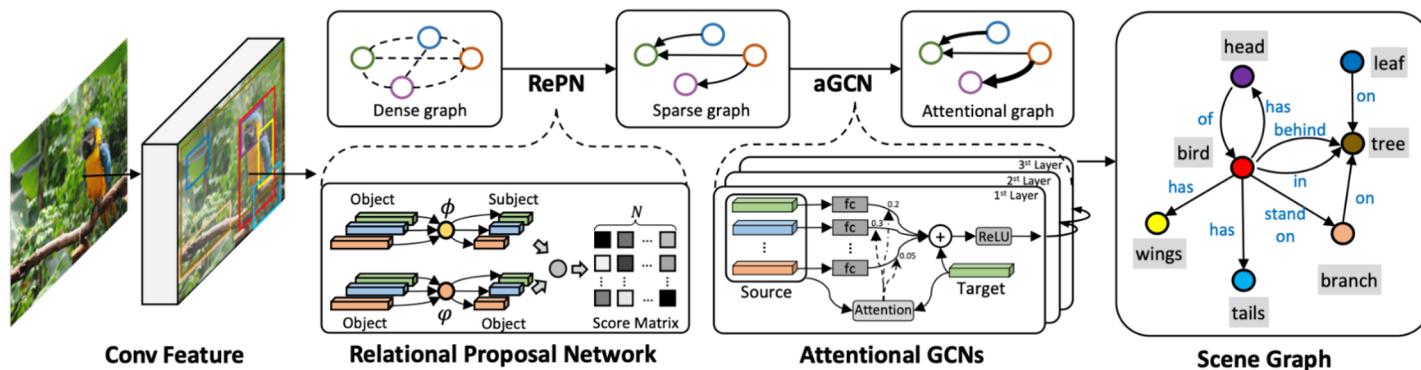
Mesh R-CNN

- Final stage parallel to box prediction
 - Predict voxels
 - Align and refine meshes with graph convolution



Graph R-CNN

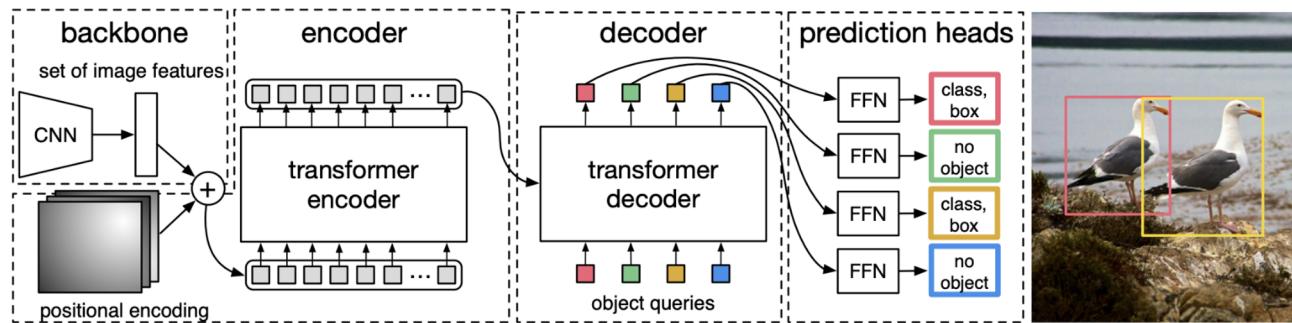
- Object detection + relationship detection
- Additional Relation Proposal Network
- Use Graph Convolution Network (GCN) for scene graph refinement



Yang, Jianwei, et al. "Graph r-cnn for scene graph generation." *Proceedings of the European conference on computer vision (ECCV)*. 2018.

DETR: End-to-End Object Detection with Transformers

- Using Transformer to directly produce boxes
- Predict objects (much larger than number of boxes) using learned fixed number of object queries



Carion, Nicolas, et al. "End-to-end object detection with transformers." *European Conference on Computer Vision*. Springer, Cham, 2020.

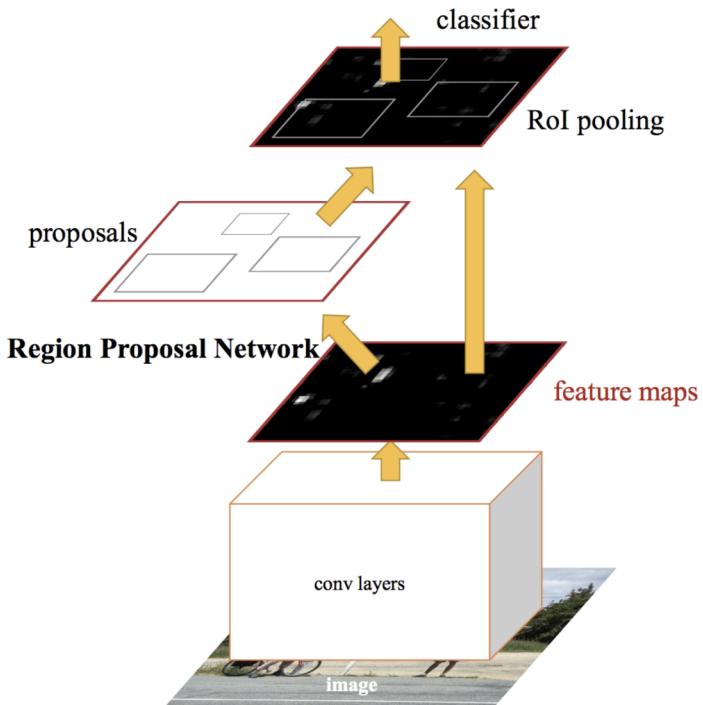
Conclusion

Stage 1

- For every output pixels
 - For every anchor boxes
 - Predict bounding box offsets
 - Predict anchor confidence (objectness/class)
- Output
 - Region proposals (region-of-interest, RoI)

Stage 2

- For each RoI
 - Perform pooling using the RoI (RoI pooling)
 - Predict bounding box offsets
 - Predict object class (semantic class / background)
 - Predict other stuff! (segmentation, pose, mesh, etc.)
- Non-maximum Suppression



Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." *arXiv preprint arXiv:1506.01497* (2015).

Implementing a Detector: Detectron2

- Open-source software for object detection and more
- Developed by Facebook with PyTorch
- Easily extendable with extensive documentations

2. RNNs

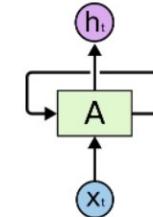
Recurrent Neural Networks

Traditional Neural Networks

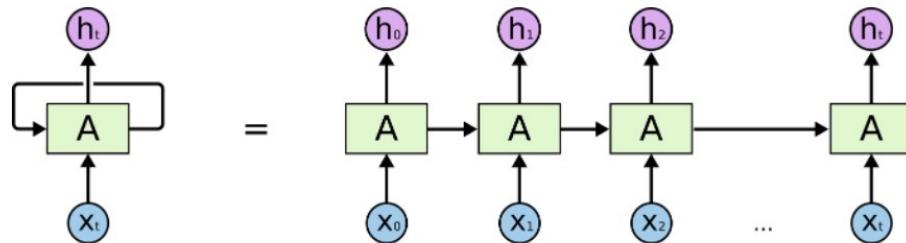
- Can't use its reasoning about previous events to inform later ones.

Recurrent Neural Networks

- Networks with loops allow information to persist.
- Chain-like, multiple copies of the same network



Recurrent Neural Networks have loops.



An unrolled recurrent neural network.

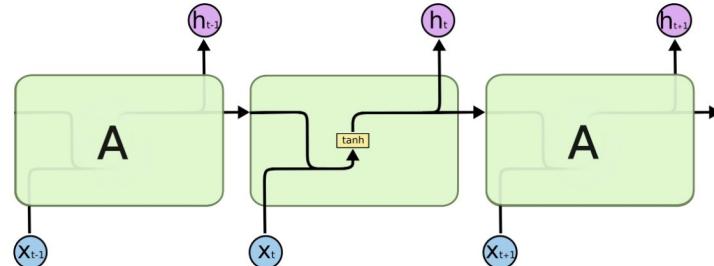
LSTM Networks

Recurrent Neural Networks

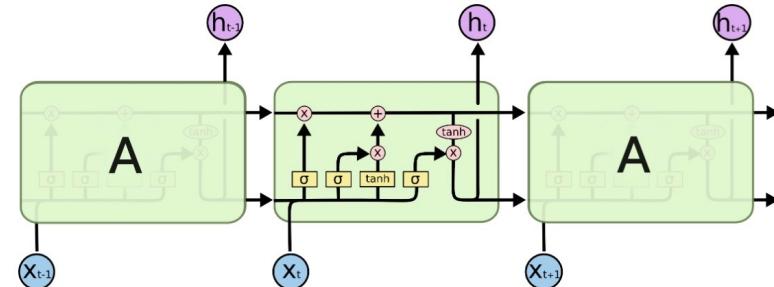
- Difficult to handle long-term dependencies.
- Explained in [Hochreiter \(1991\)](#)
[\[German\]](#) and [Bengio, et al. \(1994\)](#)

Long Short Term Memory Networks (LSTMs)

- A special kind of RNN.
- The repeating module has a different structure.



The repeating module in a standard RNN contains a single layer.

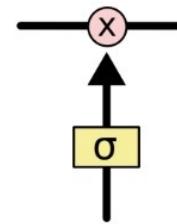
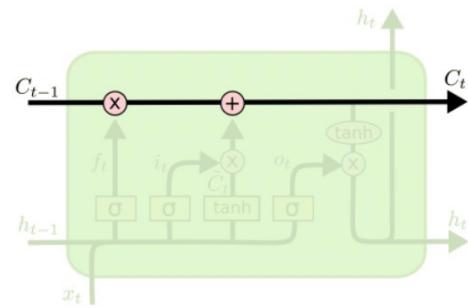
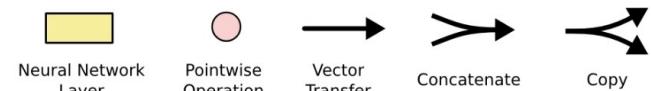


The repeating module in an LSTM contains four interacting layers.

Core Idea Behind LSTM Networks

Key to LSTMs

- Cell state:
 - Only some minor linear interactions, information flow unchanged.
- “Gates”
 - Remove or add information to the cell state:
 - Composed of:
 - Sigmoid neural net layer: output 0 - 1
 - Pointwise multiplication operation



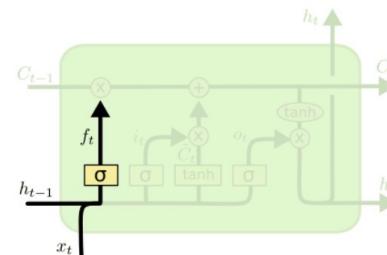
Step-by-Step LSTM walk through

1. Throw away information from the cell state

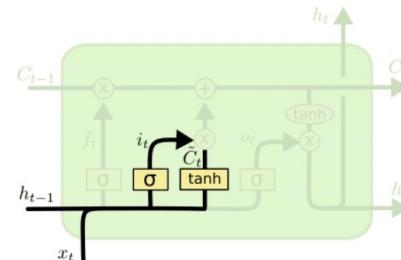
- Forget gate layer
 - For each number in the Cell State C_{t-1} : Input h_{t-1}, x_t , output a number between 0 and 1.

2. Store new information in the cell state

- Input gate layer
 - Sigmoid layer, output between 0 and 1, decides which values we'll update.
- A tanh layer
 - Creates a vector of new candidate values \tilde{C}_t

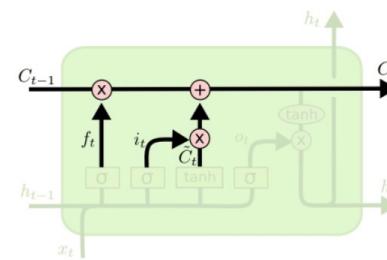


$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$



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

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

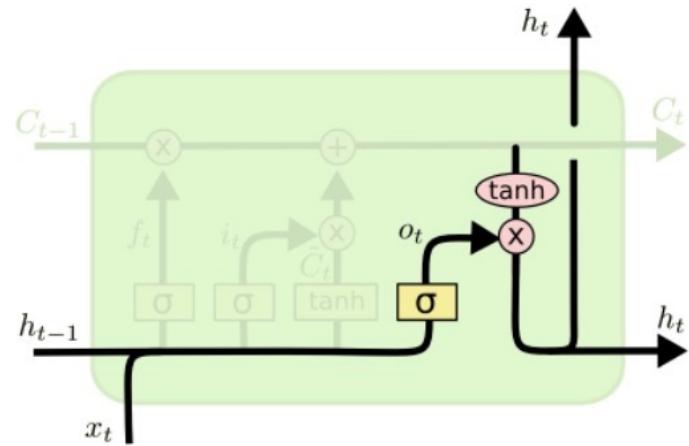


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Step-by-Step LSTM walk through

3. Output – A filtered cell state version

- Sigmoid gate layer
 - Decides what parts of the cell state we're going to output.
- Cell State tanh layer
 - Input : Cell state
 - Output: Push the values to be between -1 and 1
- Multiply the above two, and get the final output h_t .



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

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

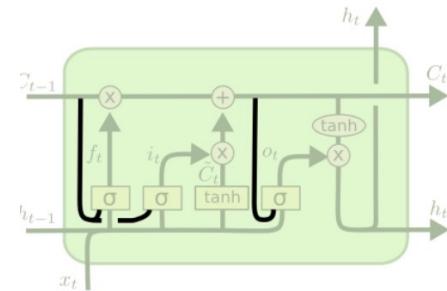
Variants on LSTM

1. Adding “peephole connections”

- Let the gate layers look at the cell state

2. Use coupled forget and input gates

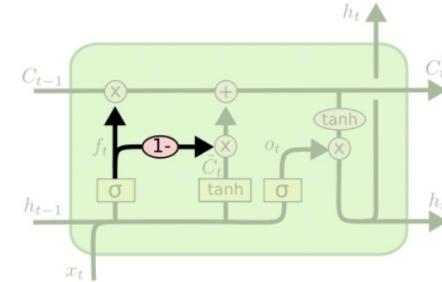
- Instead of separately deciding what to forget and what we should add new information to, we make those decisions together.



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

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

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

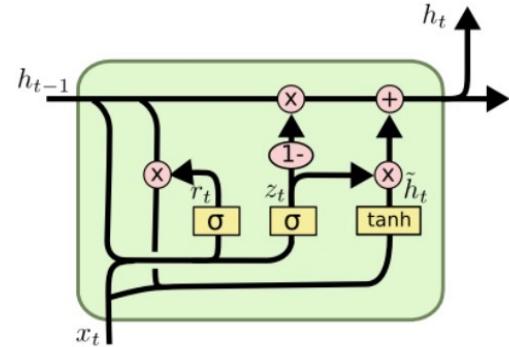


$$C_t = f_t * C_{t-1} + (1 - f_t) * \tilde{C}_t$$

Variants on LSTM

1. Gated Recurrent Unit (GRU)

- Combines the forget and input gates into a single “update gate”.
- Merges the cell state and hidden state.
- Simpler than standard LSTM models, growing increasingly popular.



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

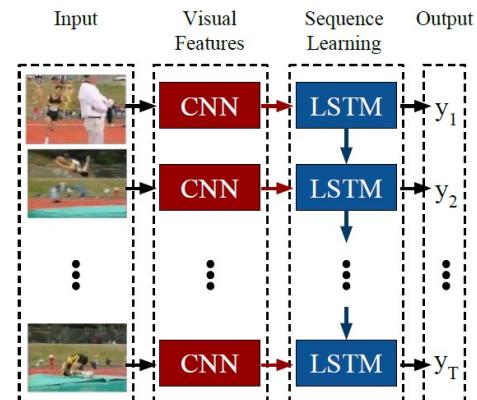
$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t])$$

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

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

RNNs take advantage of sequences

- Sequences are not only text or music, they can also be videos (sets of images).
- E.g. Understand actions in videos: Using RNNs to focus on tracking the convolutional features.
- Using RNNs and CNNs together is possible, and in fact, it could be the most advanced use of Computer Vision we have.
- Action classification, movie generation ...



Suggested Readings

- Rich feature hierarchies for accurate object detection and semantic segmentation
- Fast R-CNN
- Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks
- Mask R-CNN
- Fast Point R-CNN
- Mesh R-CNN
- Graph R-CNN for Scene Graph Generation
- You Only Look Once: Unified, Real-Time Object Detection
- SSD: Single Shot MultiBox Detector
- End-to-End Object Detection with Transformers
- Detectron2
- Recurrent Models of Visual Attention

Lecture 10: Recurrent Neural Networks

Administrative

- Project TA matchups out, see Ed for the link

Administrative

- A2 is due next Monday May 2nd, 11:59pm

Administrative

- Discussion section tomorrow 2:30-3:30PT

Object detection & RNNs Review

Last time: Detection and Segmentation

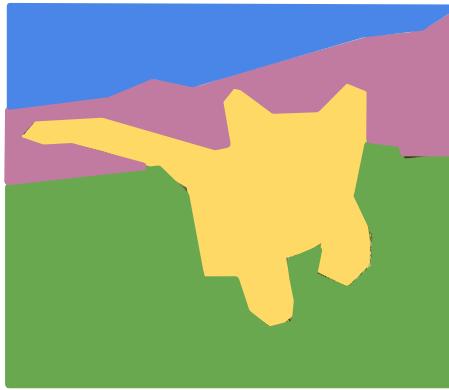
Classification



CAT

No spatial extent

Semantic Segmentation



**GRASS, CAT,
TREE, SKY**

No objects, just pixels

Object Detection



DOG, DOG, CAT

Multiple Objects

Instance Segmentation



DOG, DOG, CAT

This image is CC0 public domain

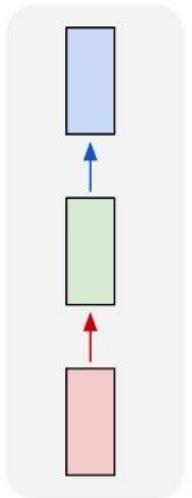
Training “**Feedforward**” Neural Networks

1. **One time set up:** activation functions, preprocessing, weight initialization, regularization, gradient checking
2. **Training dynamics:** babysitting the learning process, parameter updates, hyperparameter optimization
3. **Evaluation:** model ensembles, test-time augmentation, transfer learning

Today: Recurrent Neural Networks

“Vanilla” Neural Network

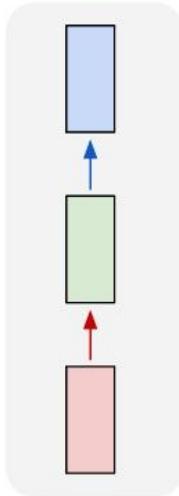
one to one



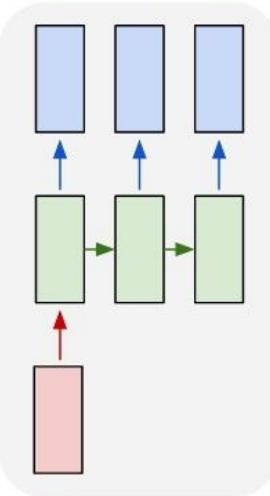
← **Vanilla Neural Networks**

Recurrent Neural Networks: Process Sequences

one to one



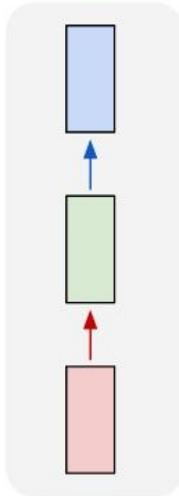
one to many



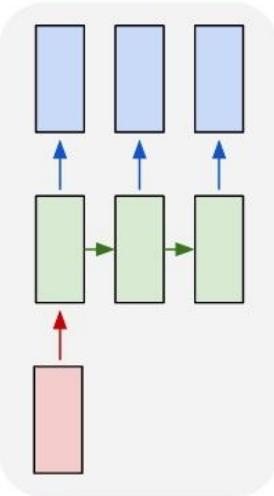
e.g. **Image Captioning**
image -> sequence of words

Recurrent Neural Networks: Process Sequences

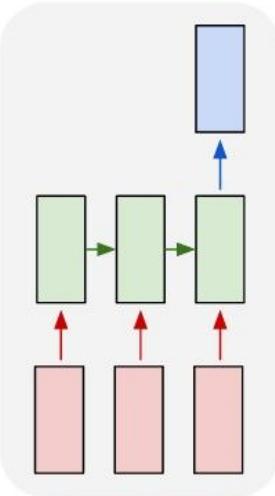
one to one



one to many



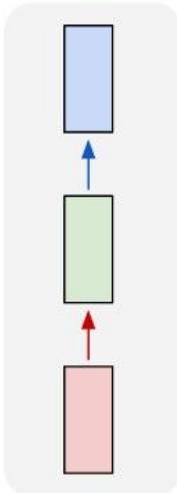
many to one



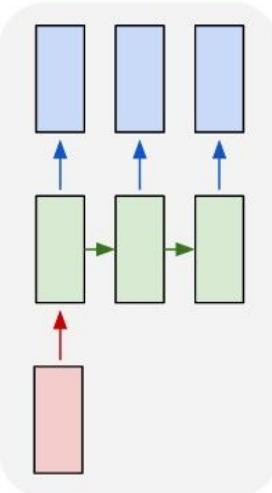
e.g. **action prediction**
sequence of video frames -> action class

Recurrent Neural Networks: Process Sequences

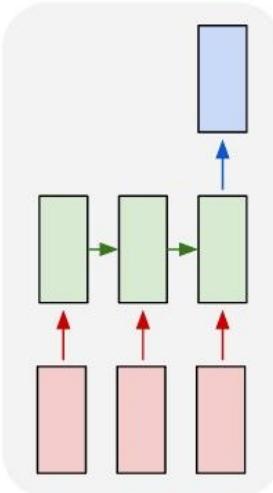
one to one



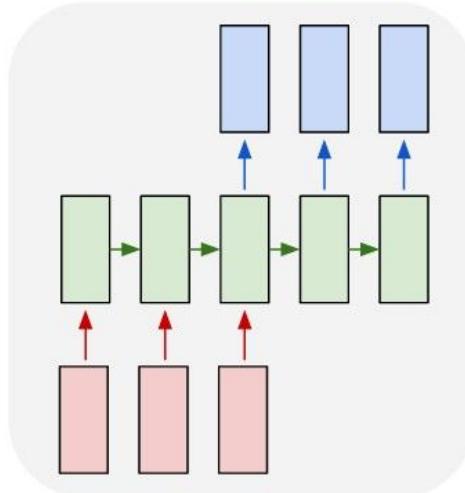
one to many



many to one



many to many

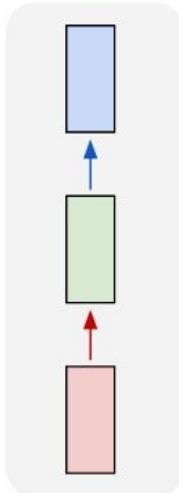


E.g. **Video Captioning**

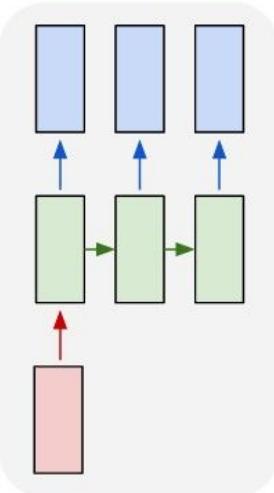
Sequence of video frames -> caption

Recurrent Neural Networks: Process Sequences

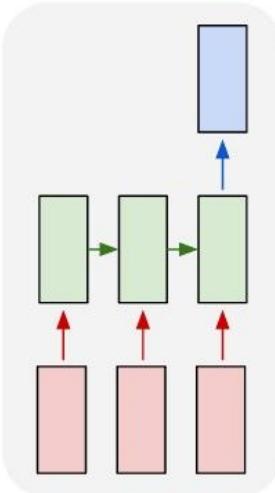
one to one



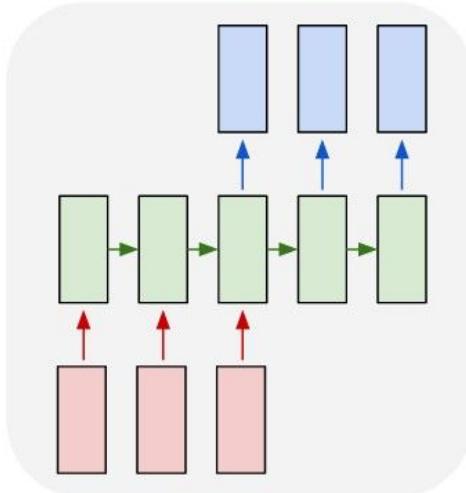
one to many



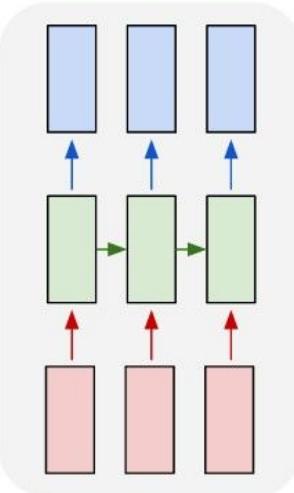
many to one



many to many



many to many



e.g. Video classification on frame level

Sequential Processing of Non-Sequence Data

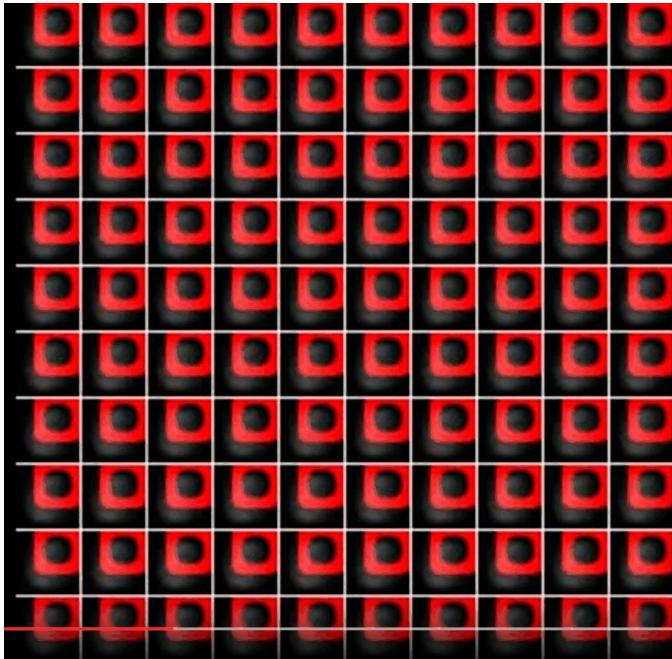
Classify images by taking a series of “glimpses”



Ba, Mnih, and Kavukcuoglu, “Multiple Object Recognition with Visual Attention”, ICLR 2015.
Gregor et al, “DRAW: A Recurrent Neural Network For Image Generation”, ICML 2015
Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.

Sequential Processing of Non-Sequence Data

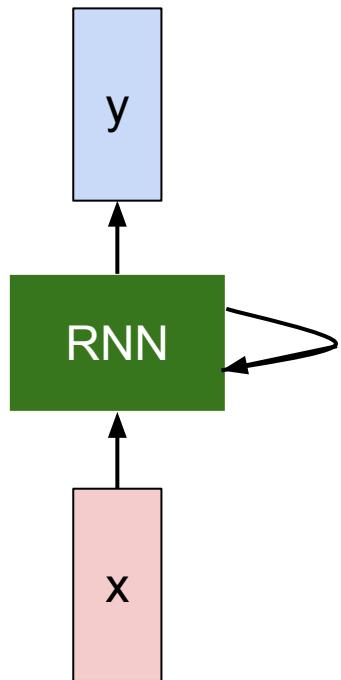
Generate images one piece at a time!



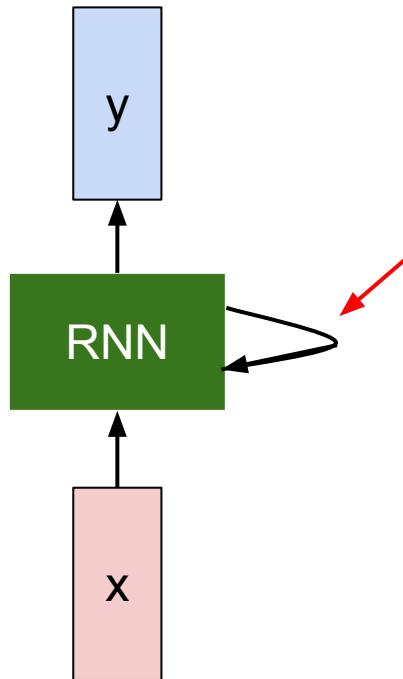
Gregor et al, "DRAW: A Recurrent Neural Network For Image Generation , ICML 2015

Figure copyright Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra, 2015. Reproduced with permission.

Recurrent Neural Network

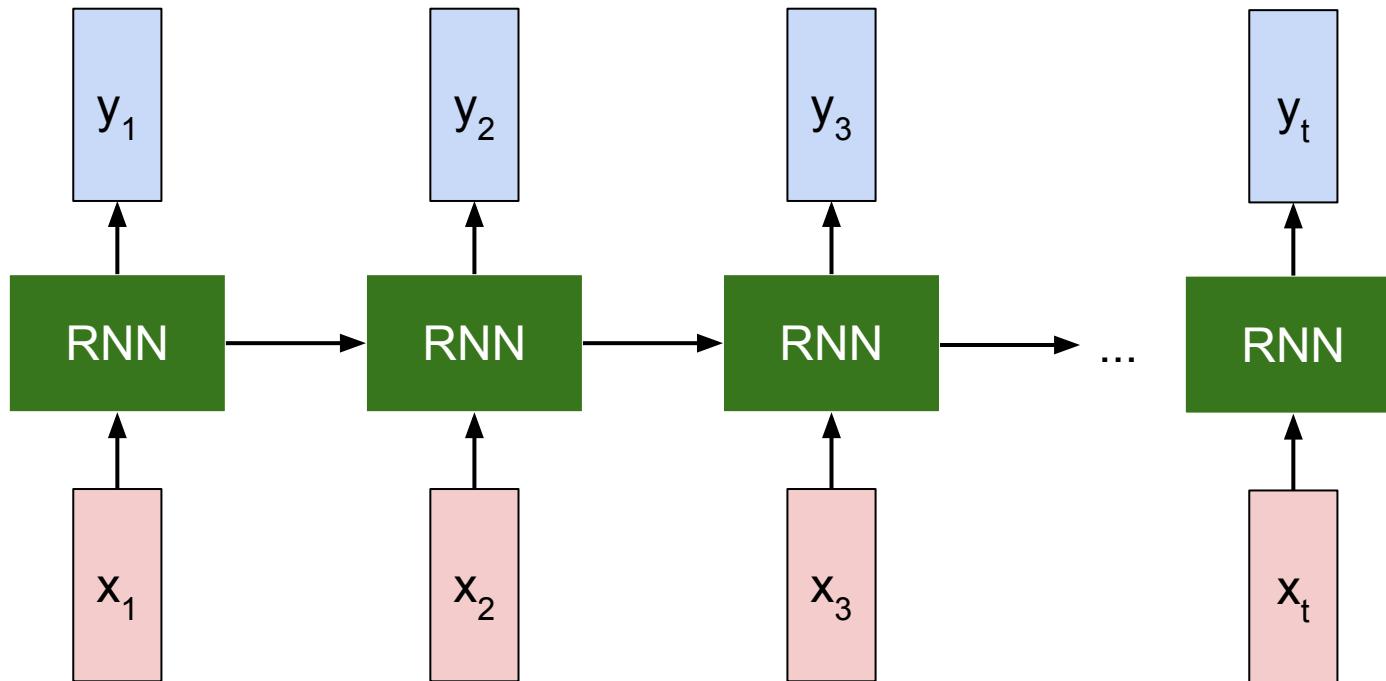


Recurrent Neural Network



Key idea: RNNs have an “internal state” that is updated as a sequence is processed

Unrolled RNN

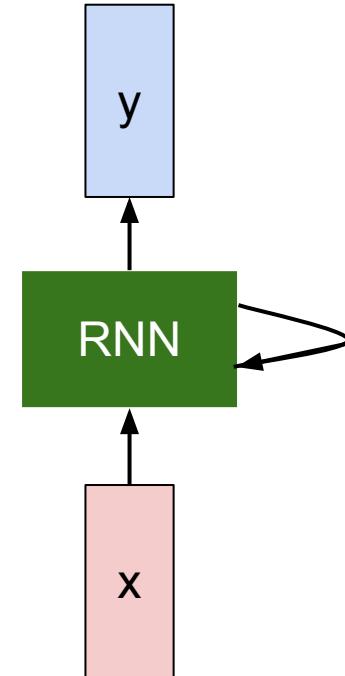


RNN hidden state update

We can process a sequence of vectors 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

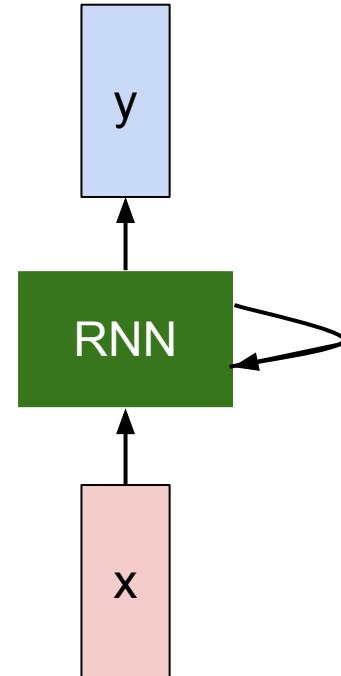


RNN output generation

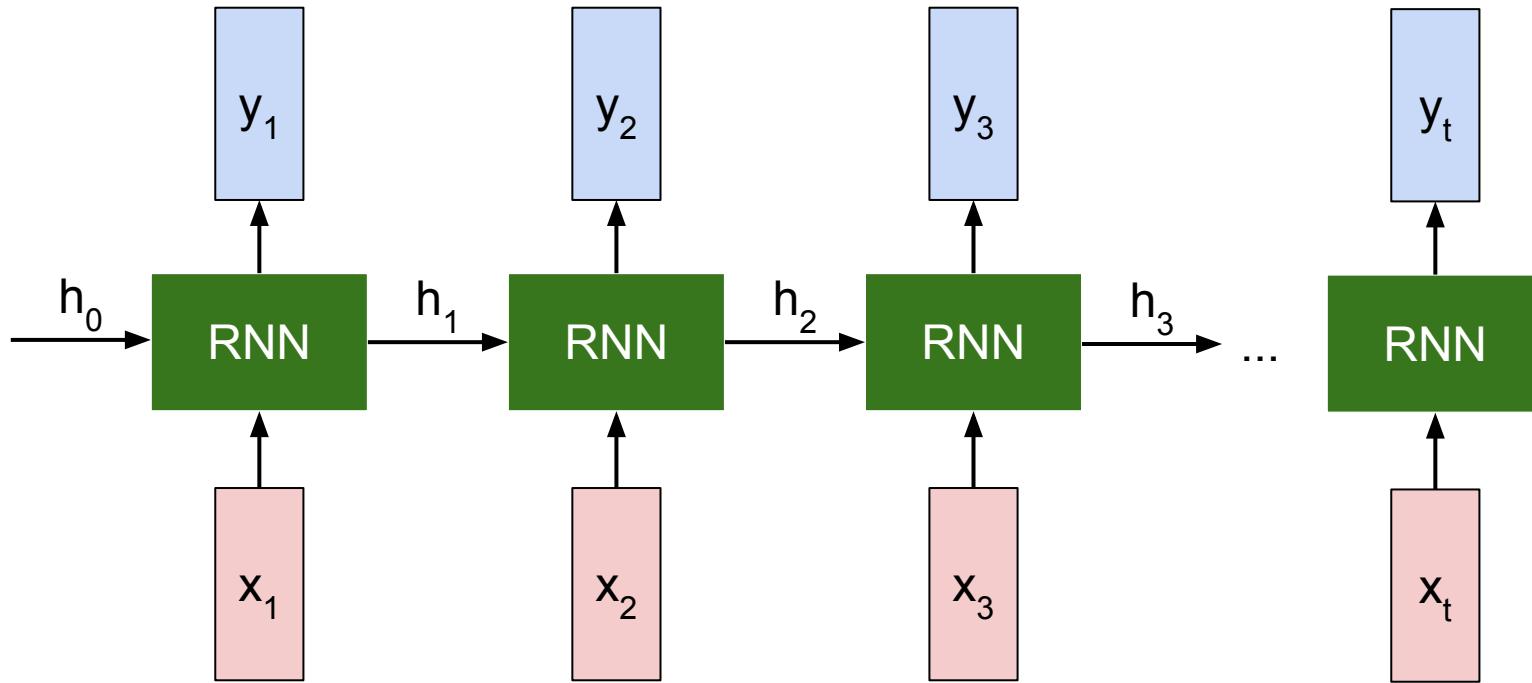
We can process a sequence of vectors \mathbf{x} by applying a **recurrence formula** at every time step:

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

output new state
another function
with parameters W_o



Recurrent Neural Network

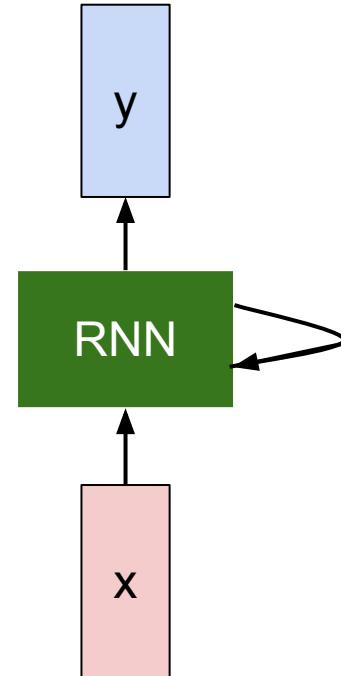


Recurrent Neural Network

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

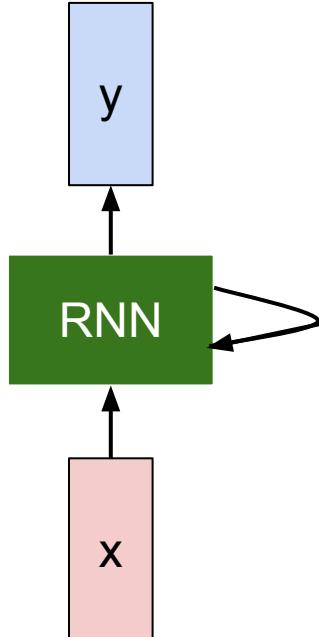
$$h_t = f_W(h_{t-1}, x_t)$$

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



(Vanilla) Recurrent Neural Network

The state consists of a single “*hidden*” vector \mathbf{h} :



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

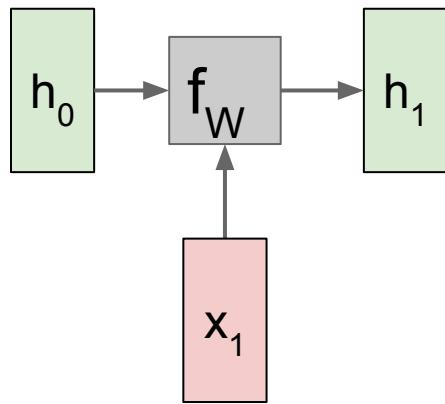


$$\mathbf{h}_t = \tanh(W_{hh}\mathbf{h}_{t-1} + W_{xh}\mathbf{x}_t)$$

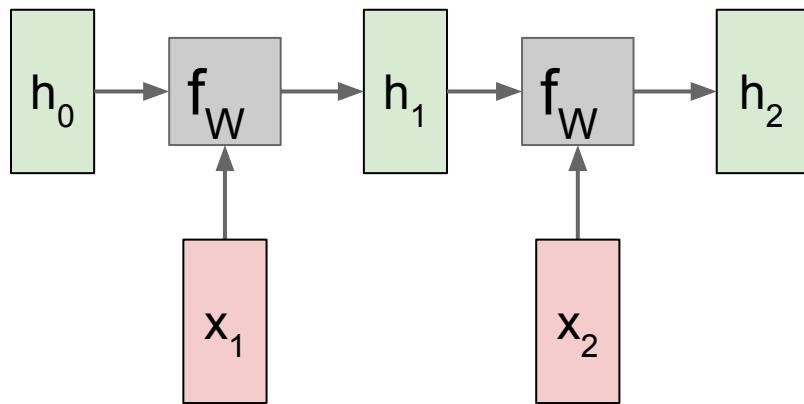
$$y_t = W_{hy}\mathbf{h}_t$$

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

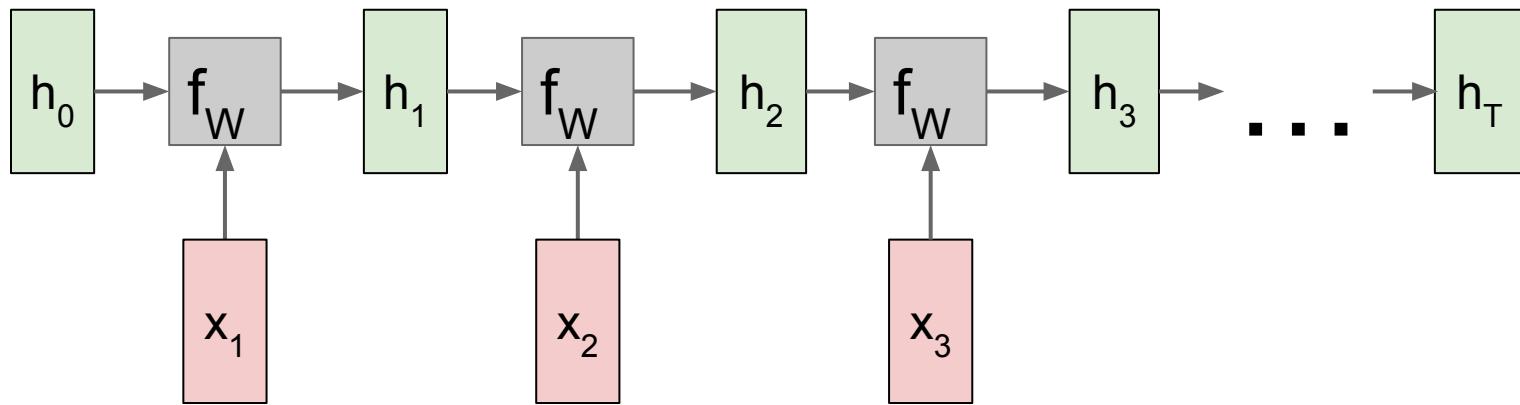
RNN: Computational Graph



RNN: Computational Graph

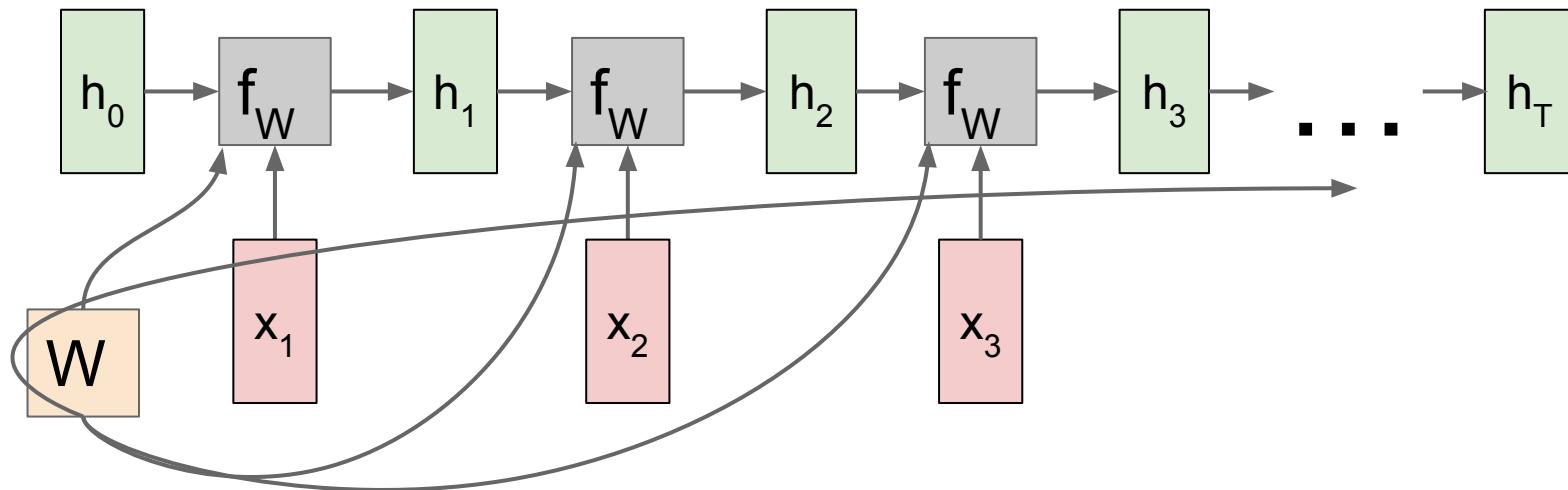


RNN: Computational Graph

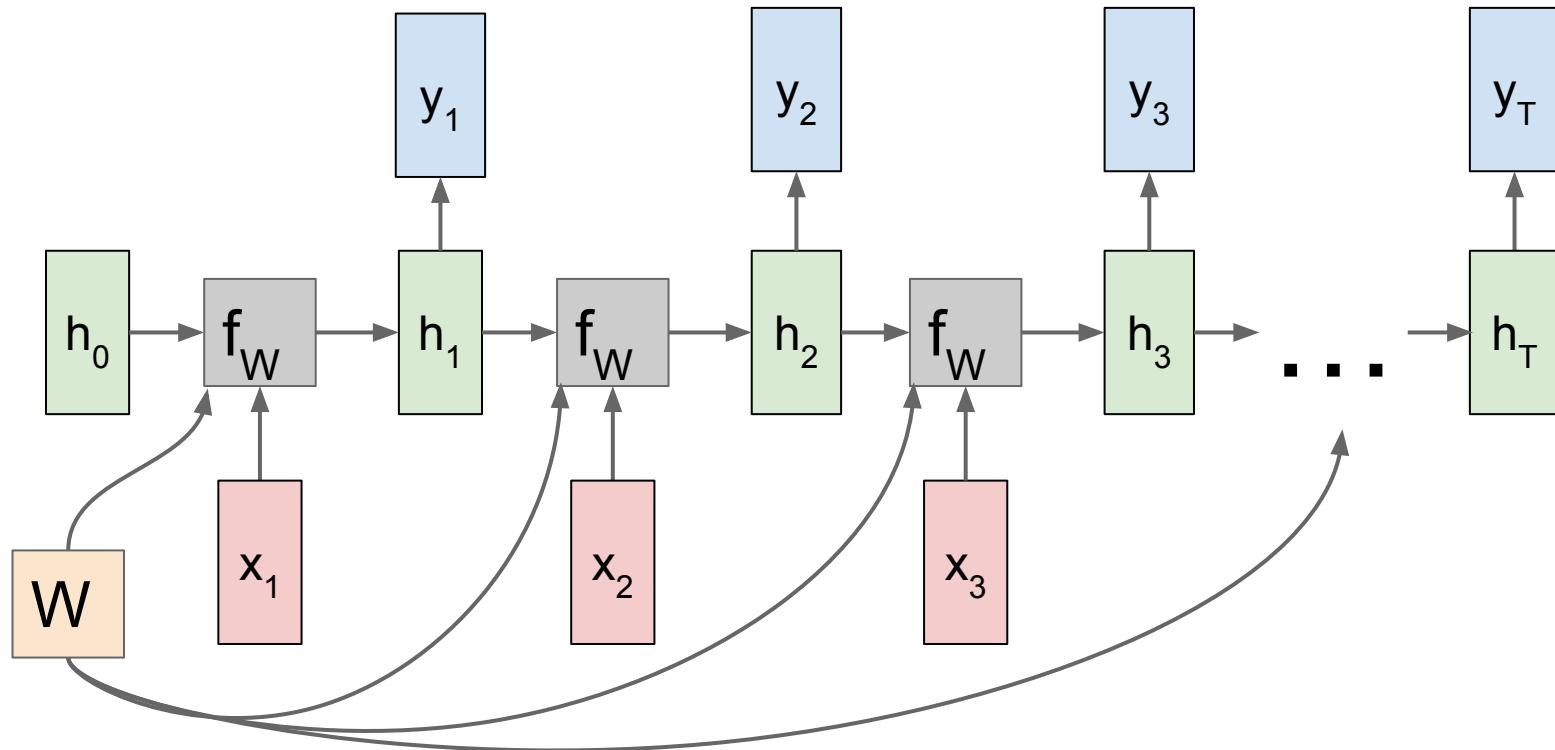


RNN: Computational Graph

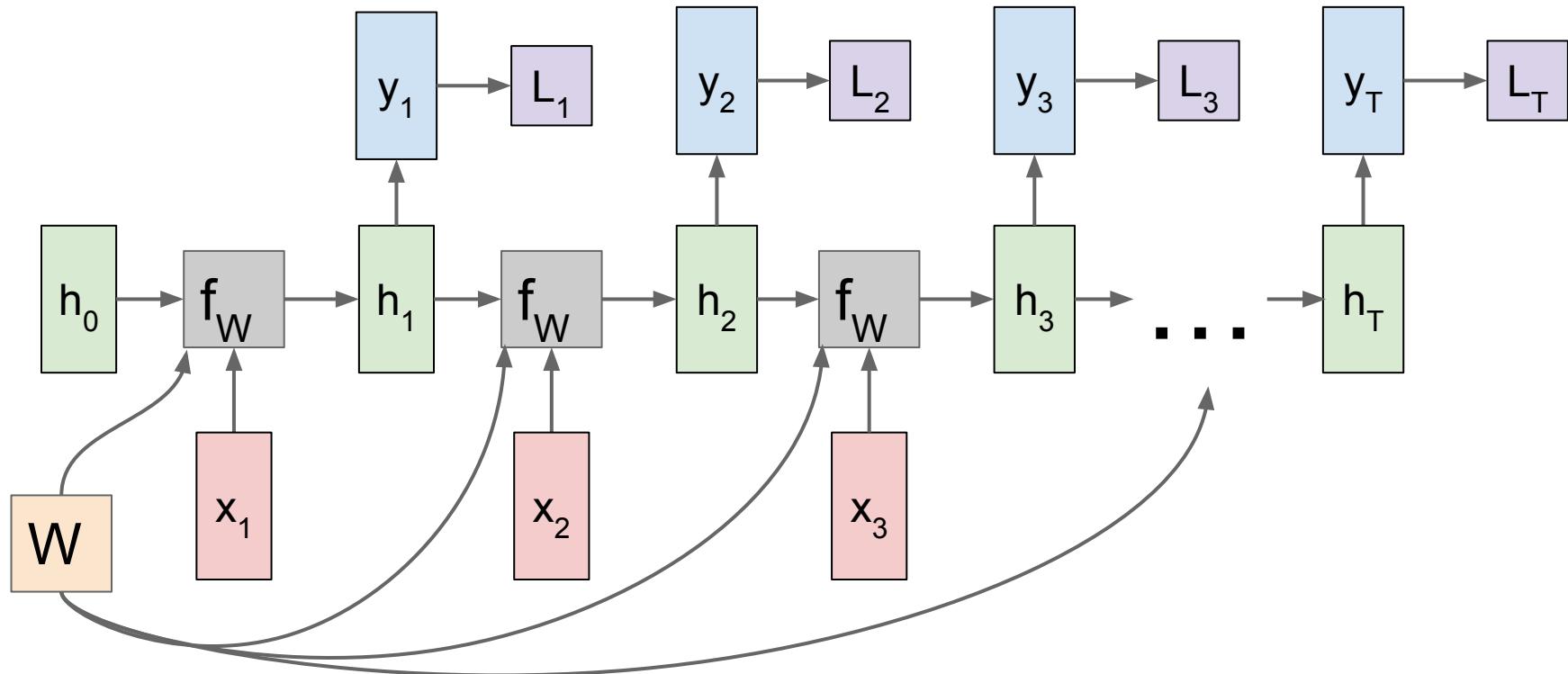
Re-use the same weight matrix at every time-step



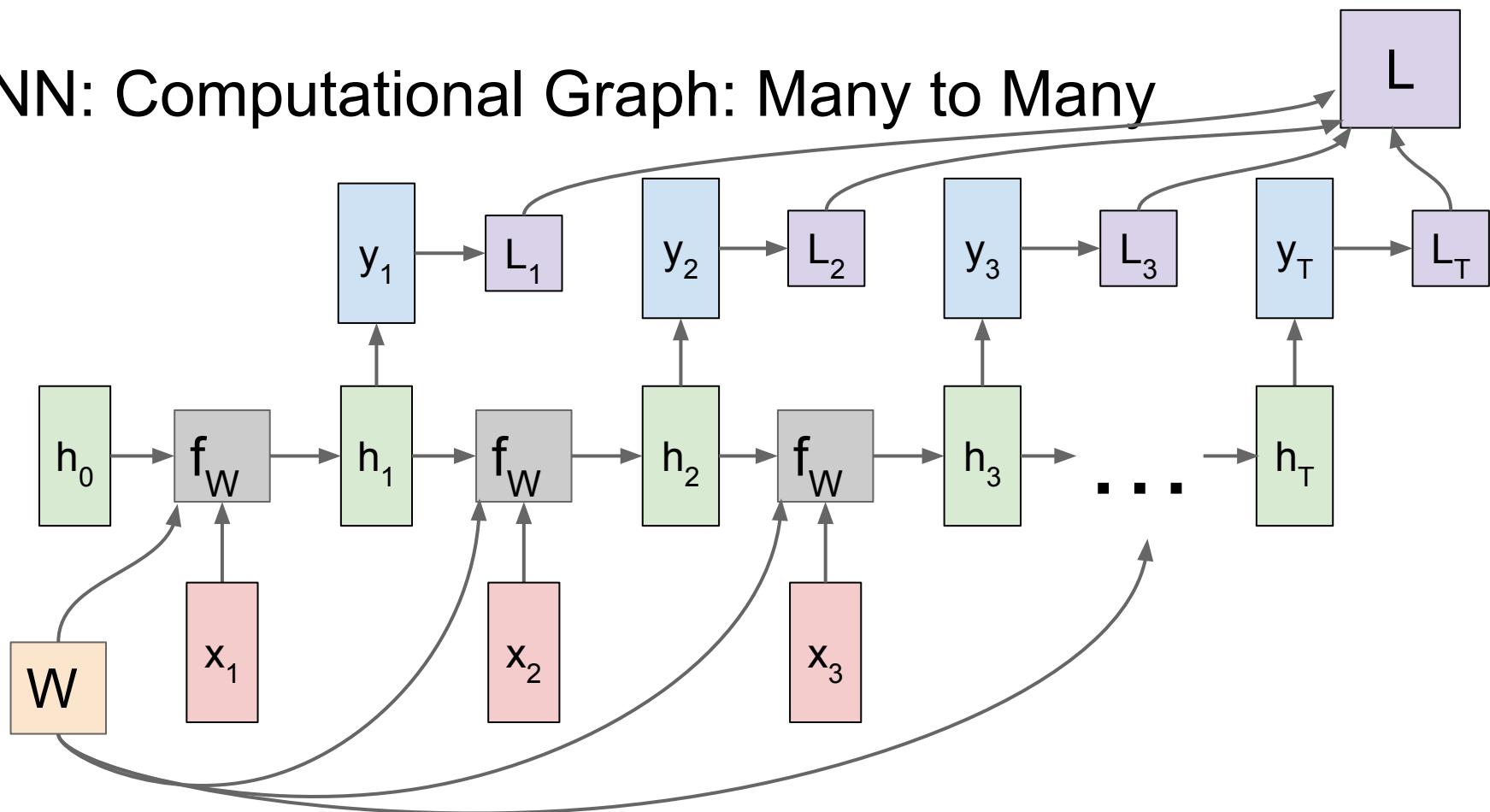
RNN: Computational Graph: Many to Many



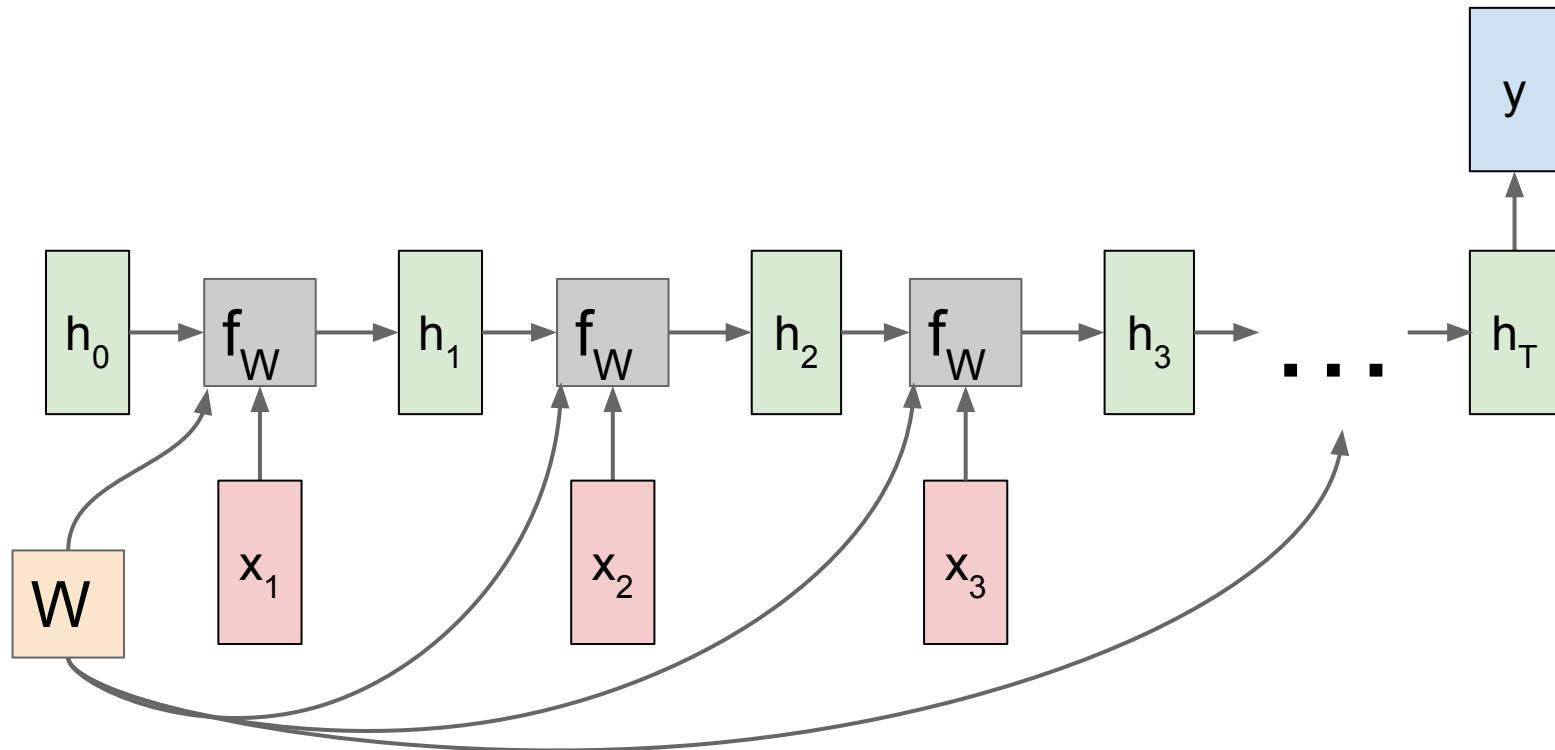
RNN: Computational Graph: Many to Many



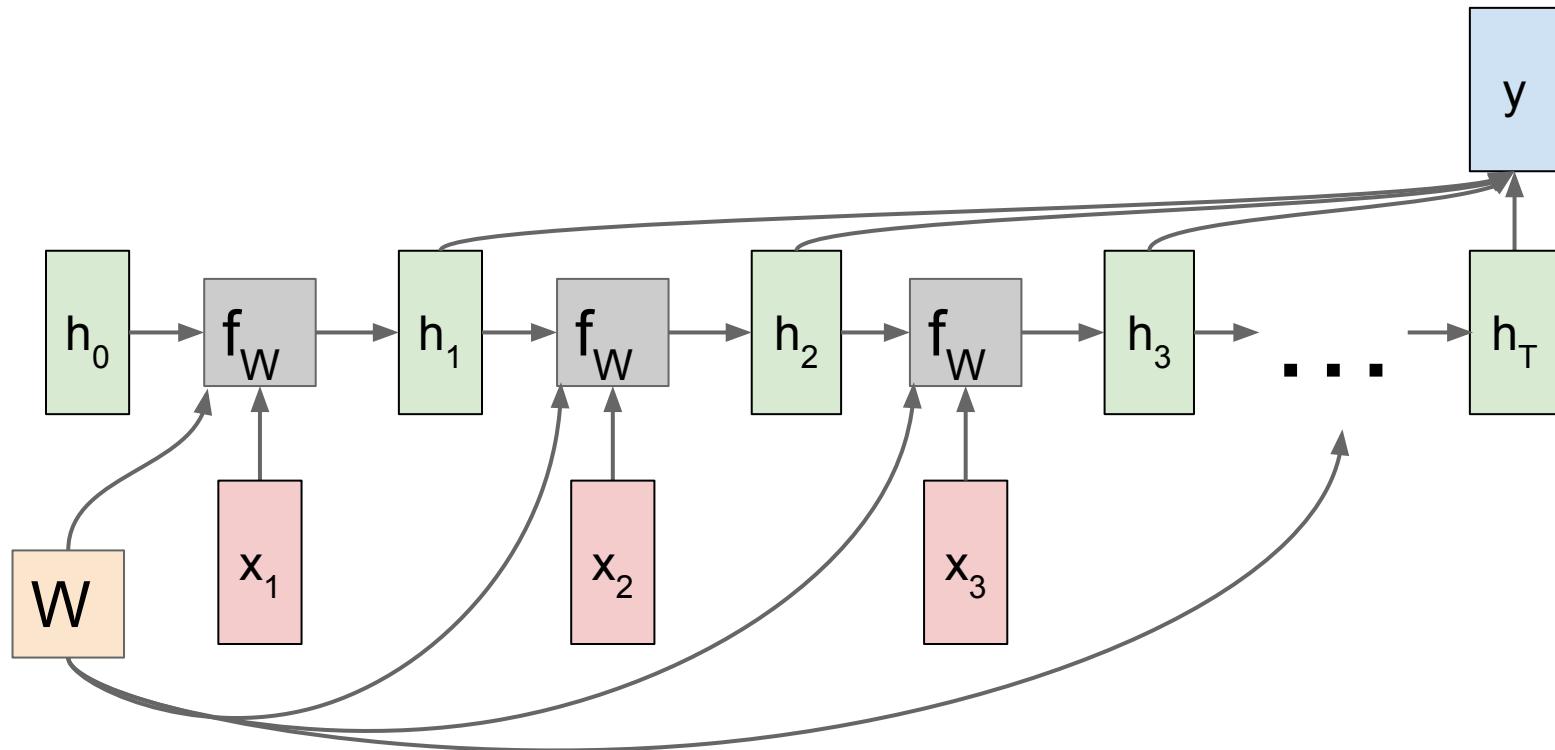
RNN: Computational Graph: Many to Many



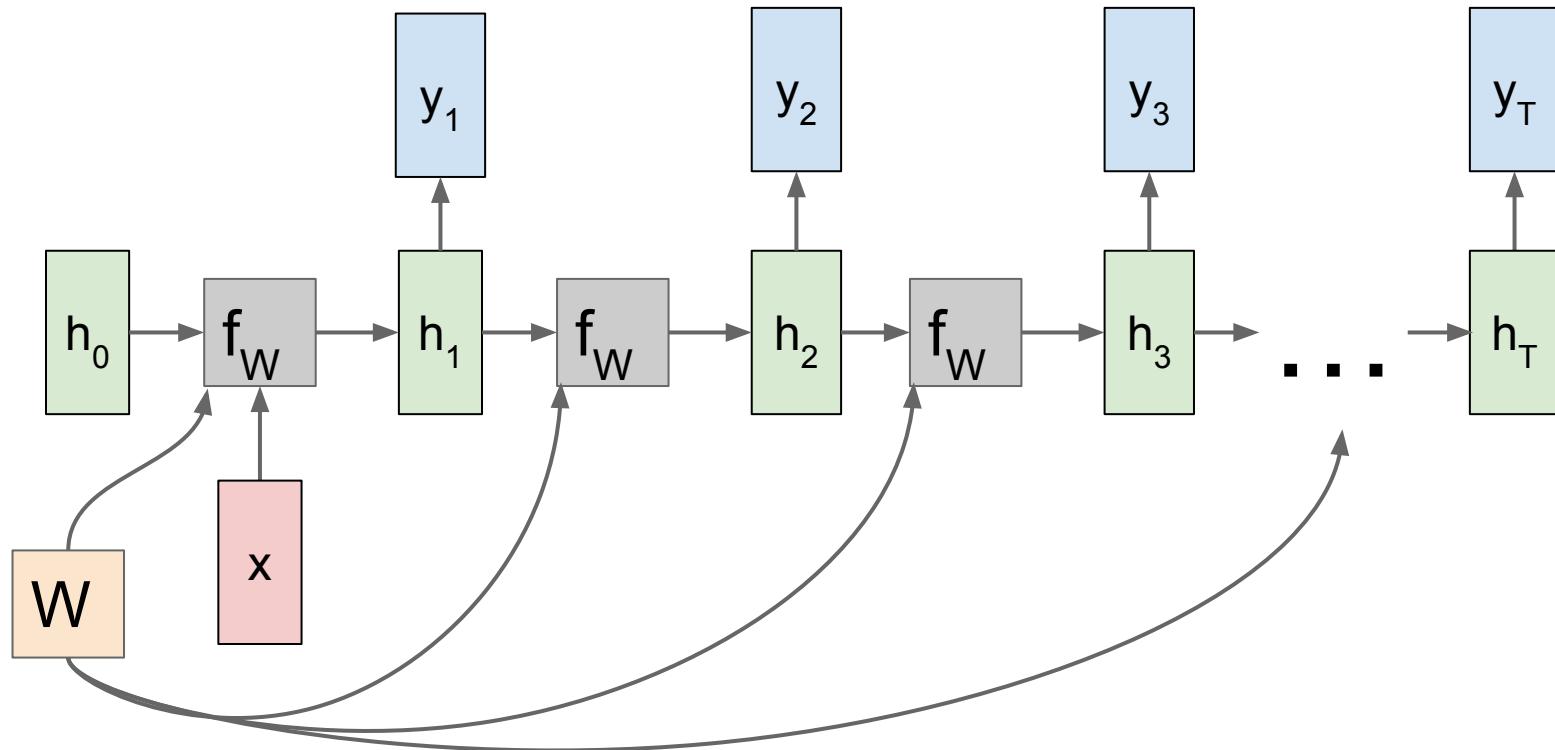
RNN: Computational Graph: Many to One



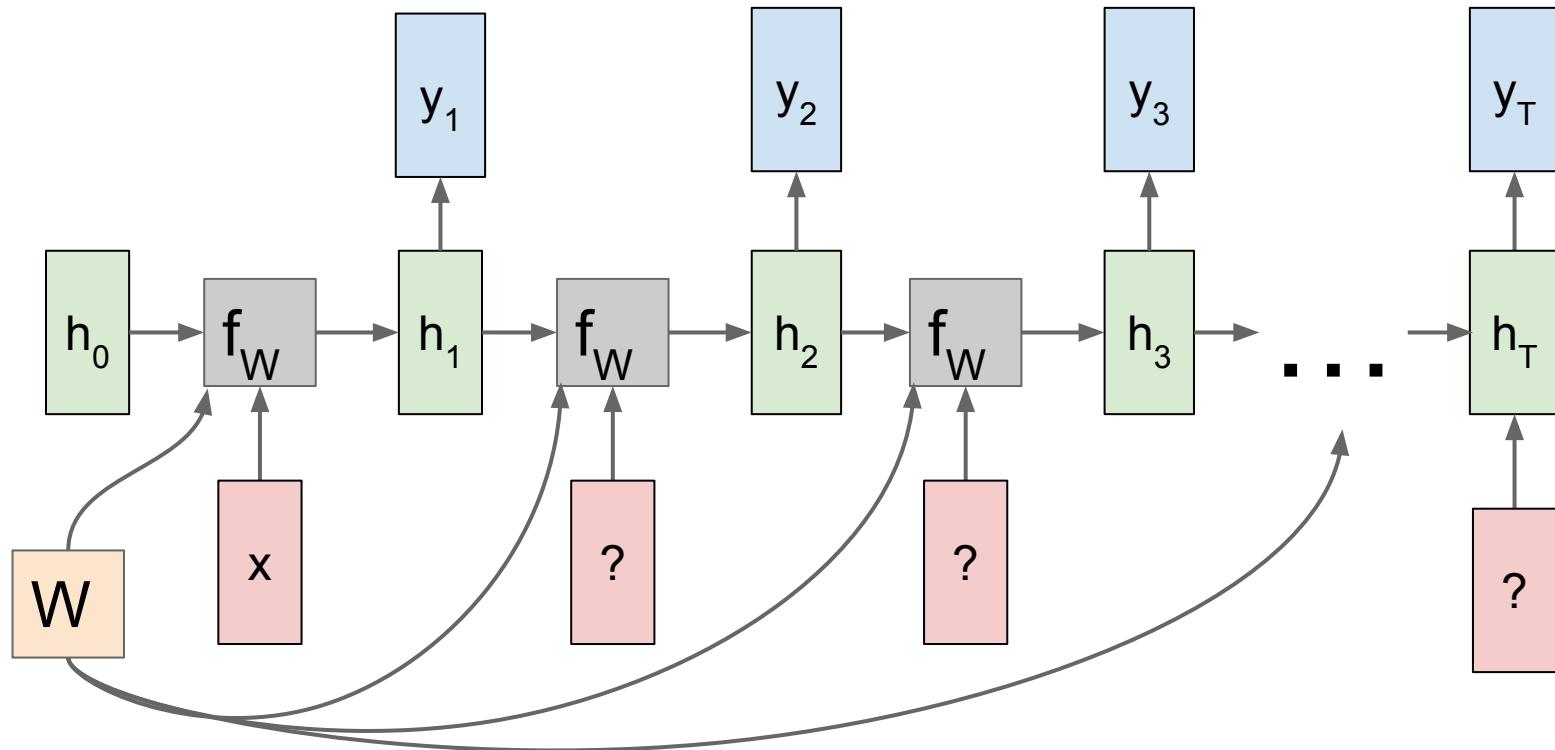
RNN: Computational Graph: Many to One



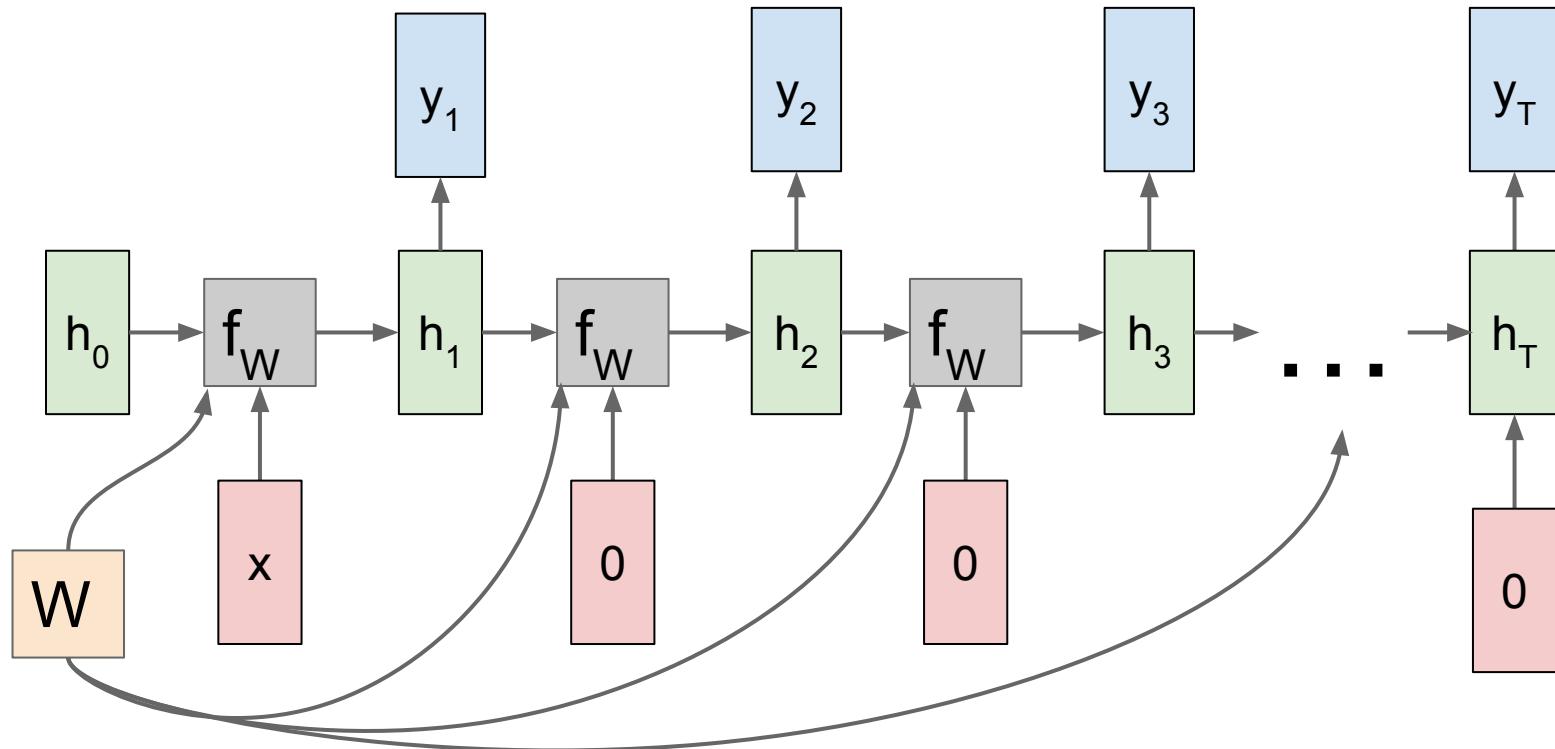
RNN: Computational Graph: One to Many



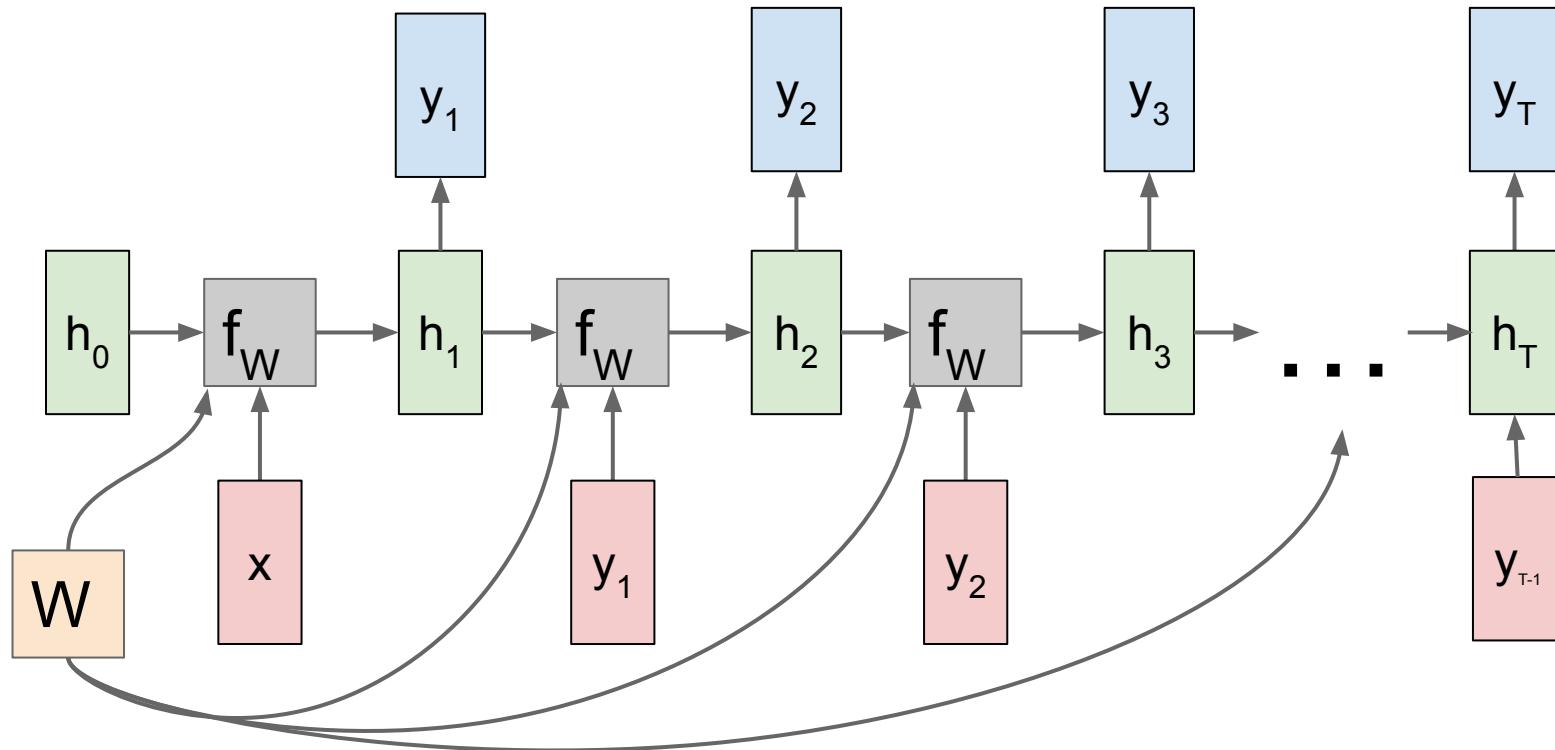
RNN: Computational Graph: One to Many



RNN: Computational Graph: One to Many

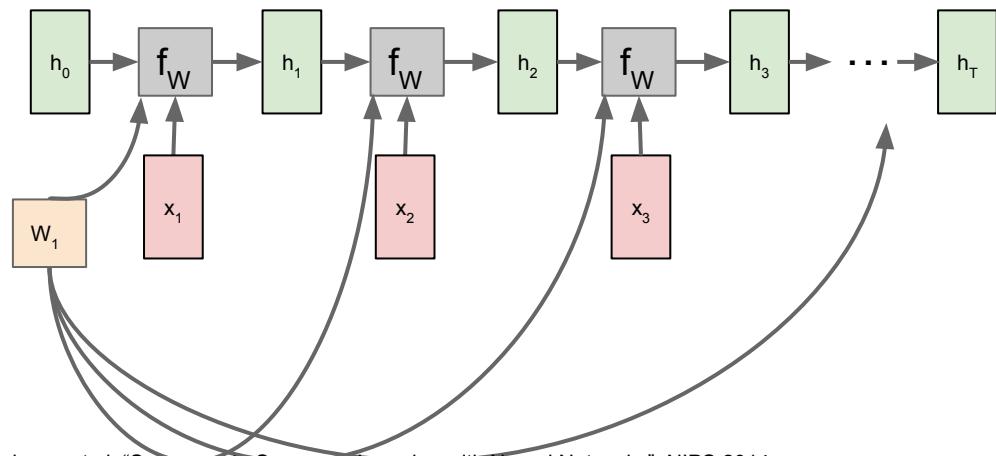


RNN: Computational Graph: One to Many



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

Many to one: Encode input sequence in a single vector

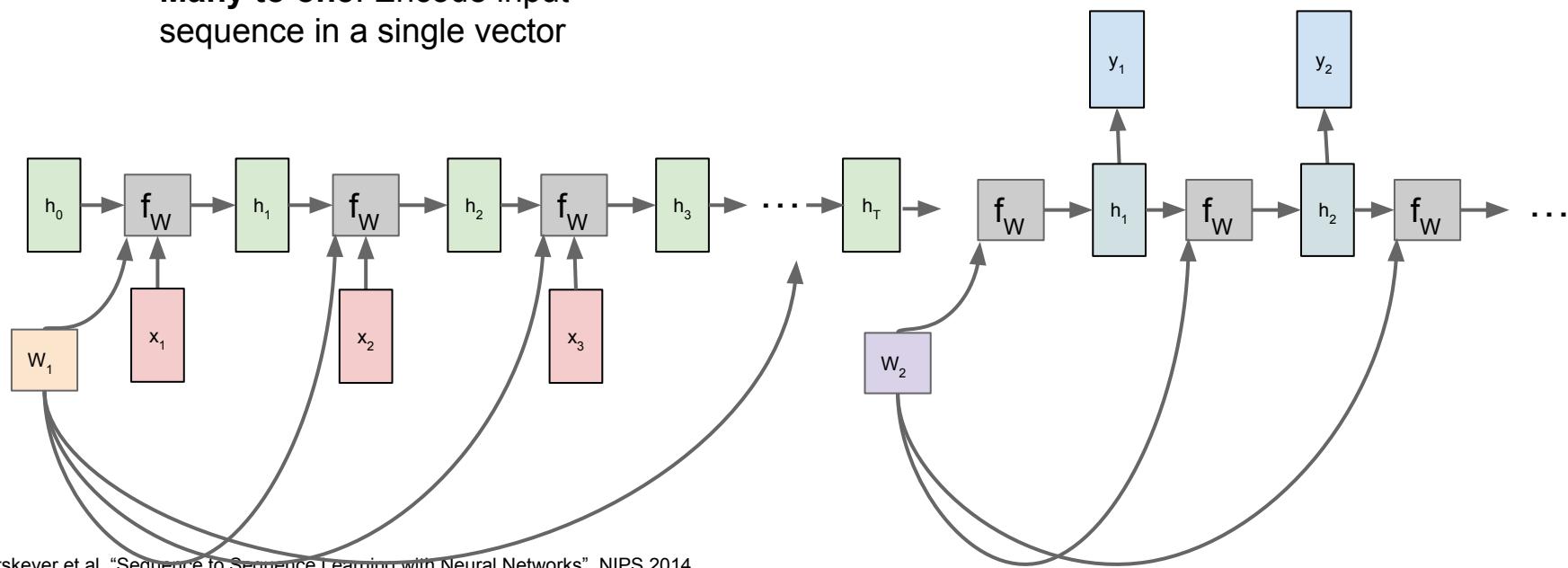


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

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

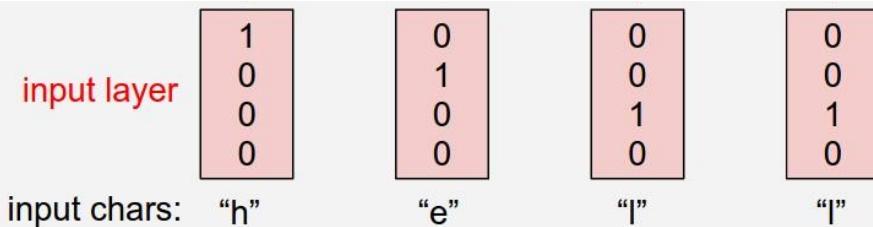


Sutskever et al, "Sequence to Sequence Learning with Neural Networks", NIPS 2014

Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

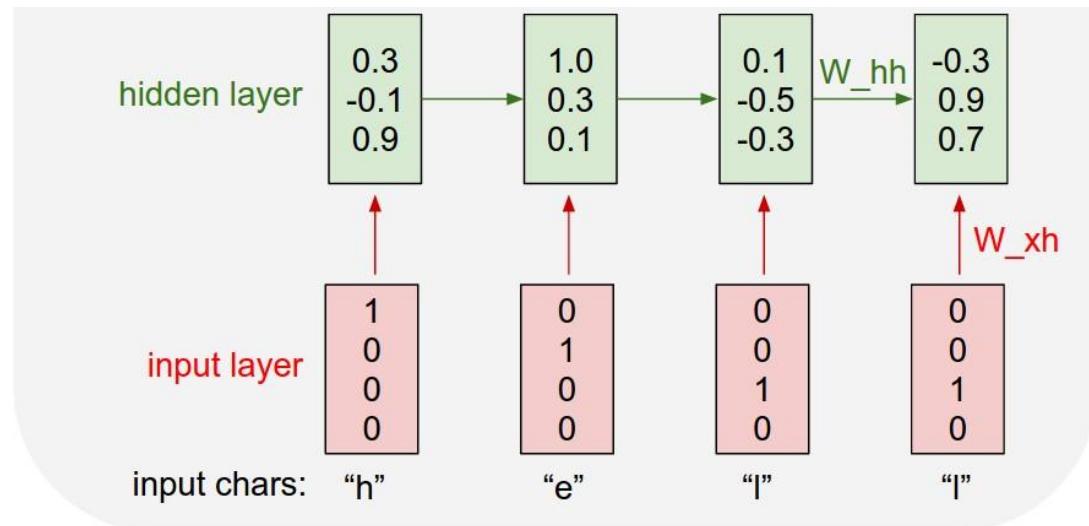


Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

Example training
sequence:
“hello”

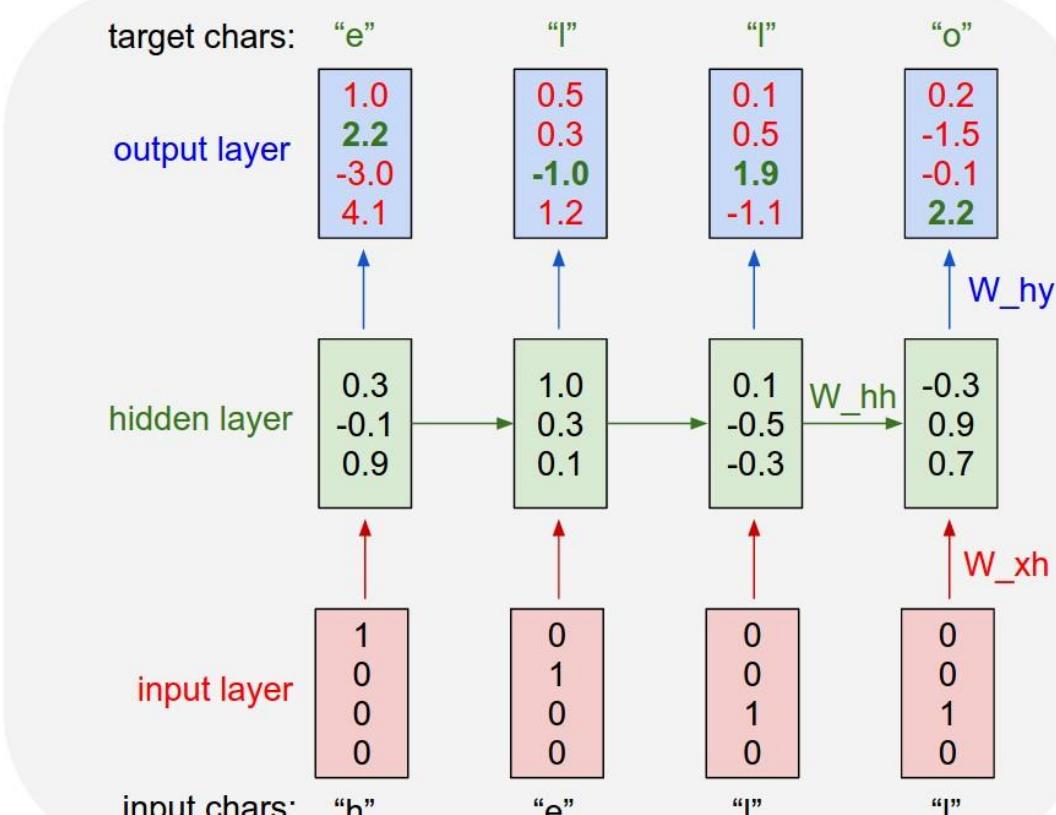
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$



Example: Character-level Language Model

Vocabulary:
[h,e,l,o]

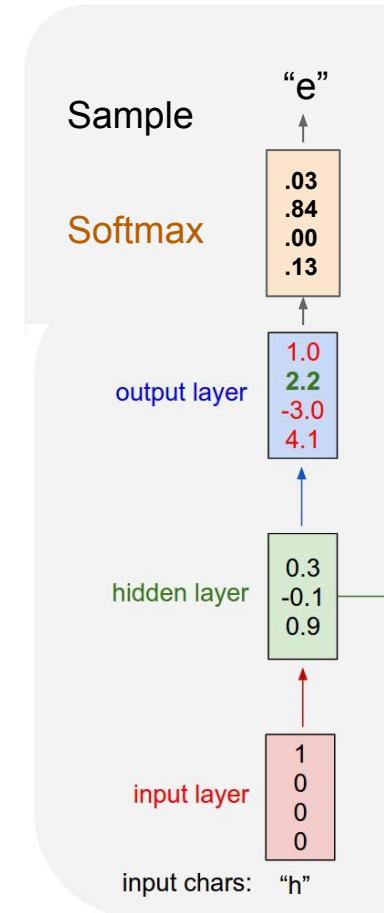
Example training
sequence:
“hello”



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

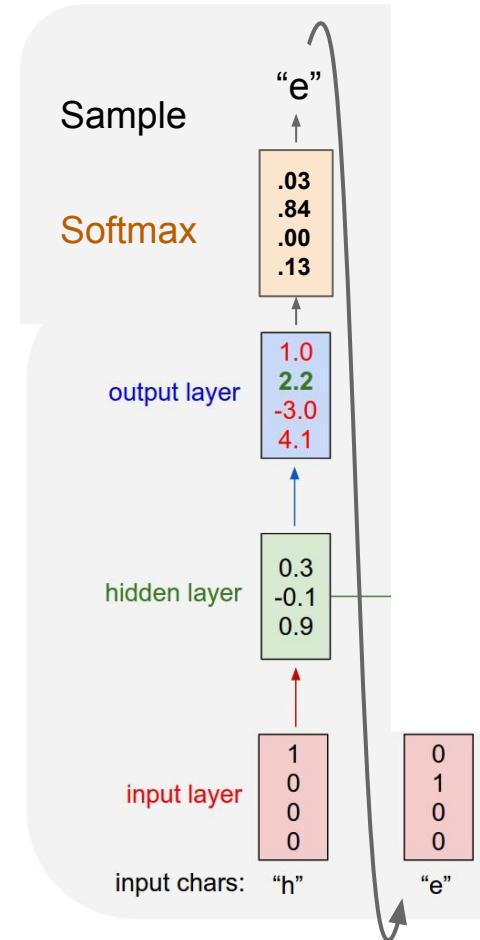
At test-time sample characters one at a time, feed back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

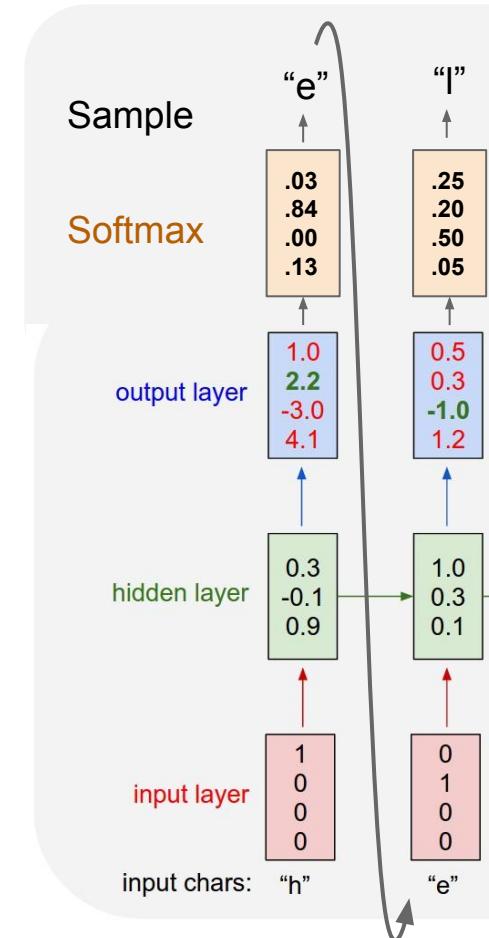
At test-time sample
characters one at a time, feed
back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

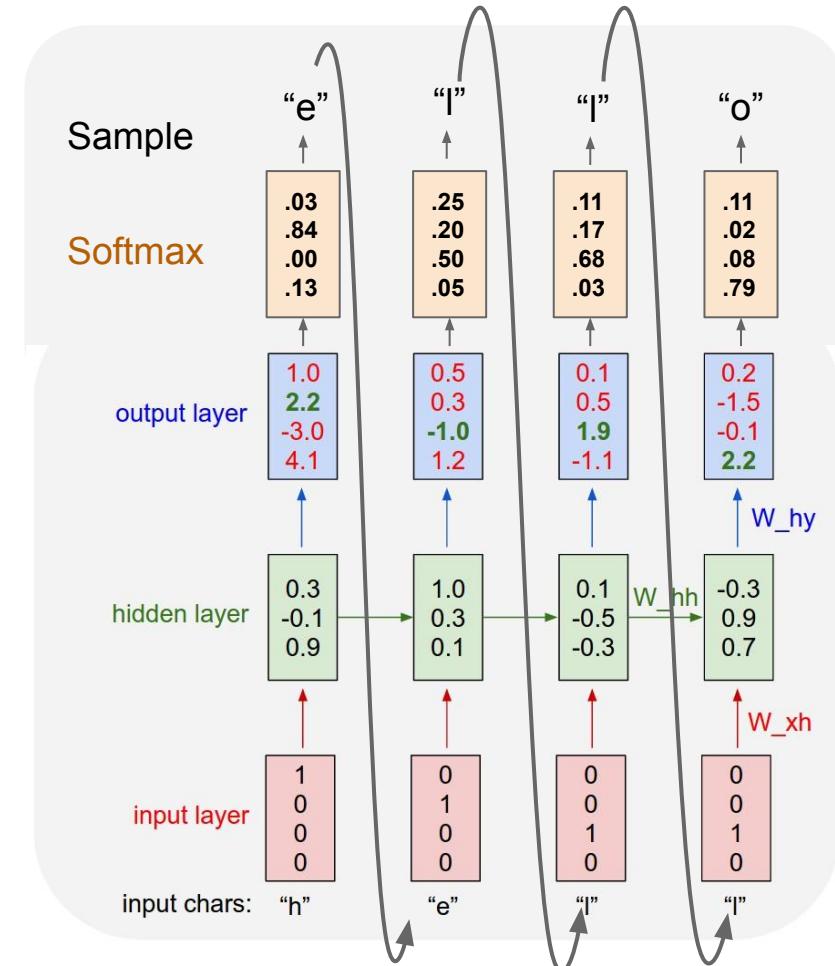
At test-time sample
characters one at a time, feed
back to model



Example: Character-level Language Model Sampling

Vocabulary:
[h,e,l,o]

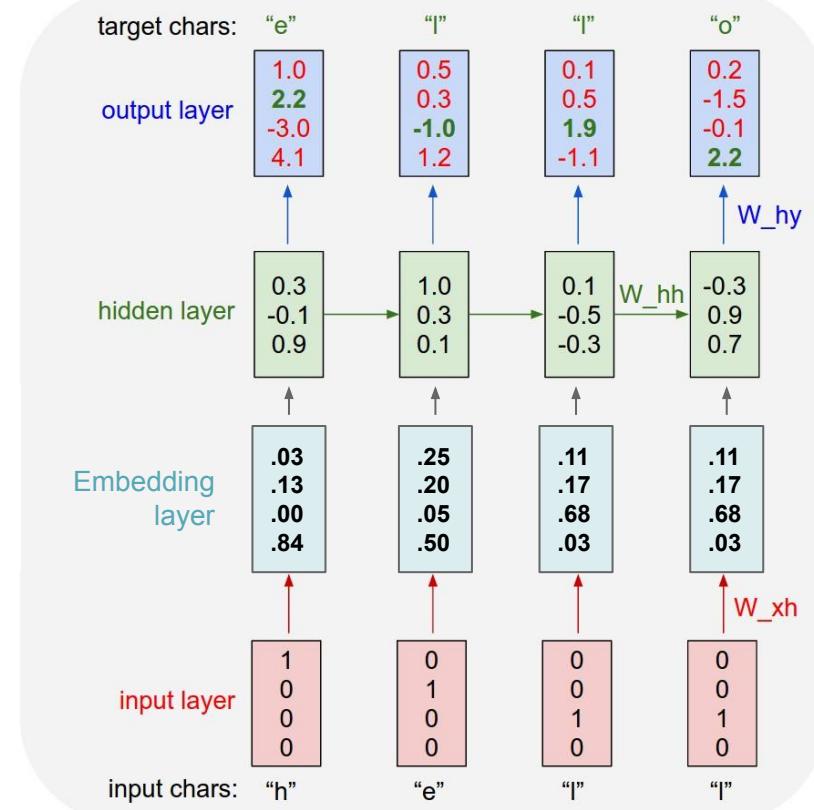
At test-time sample
characters one at a time, feed
back to model



Example: Character-level Language Model Sampling

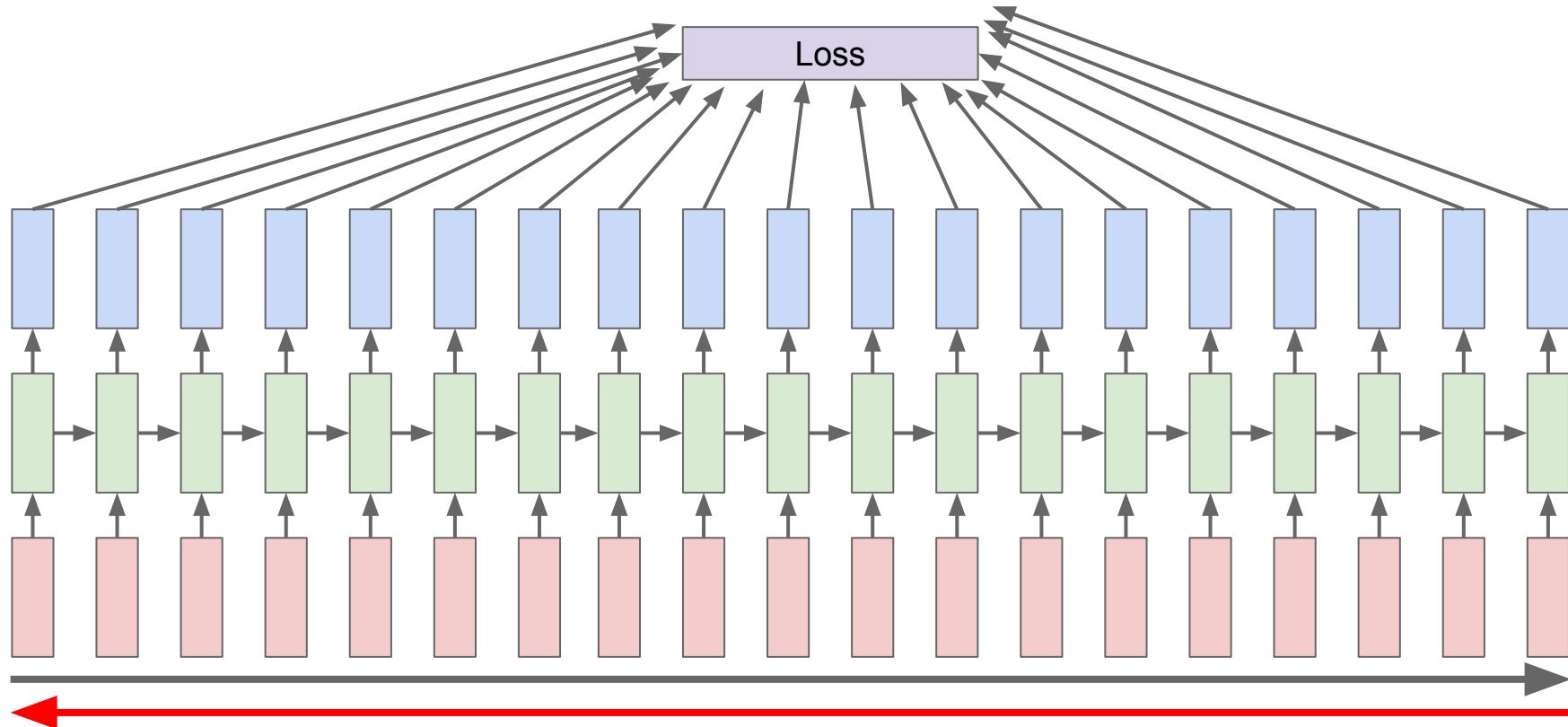
$$\begin{bmatrix} w_{11} & w_{12} & w_{13} & w_{14} \end{bmatrix} [1] \quad [w_{11}] \\ \begin{bmatrix} w_{21} & w_{22} & w_{23} & w_{14} \end{bmatrix} [0] = [w_{21}] \\ \begin{bmatrix} w_{31} & w_{32} & w_{33} & w_{14} \end{bmatrix} [0] \quad [w_{31}] \\ [0] \end{math>$$

Matrix multiply with a one-hot vector just extracts a column from the weight matrix.
We often put a separate **embedding** layer between input and hidden layers.

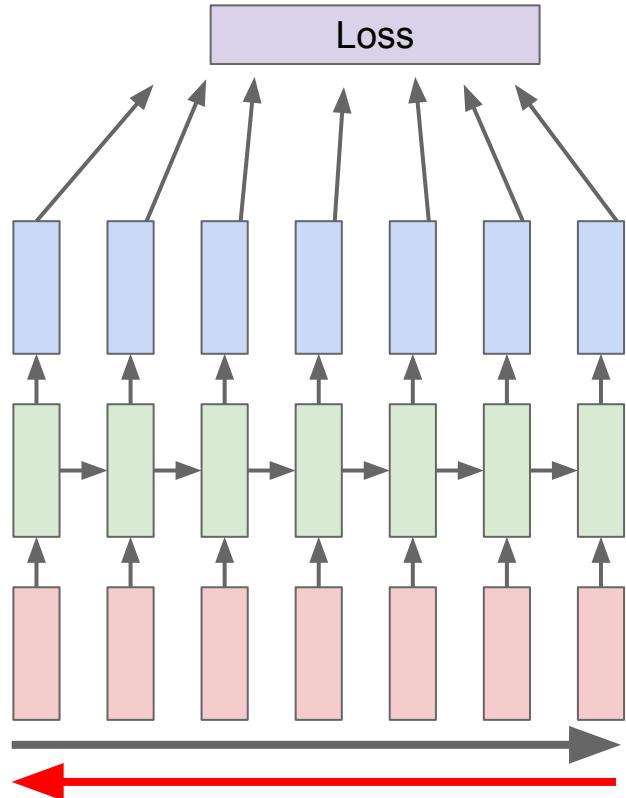


Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient

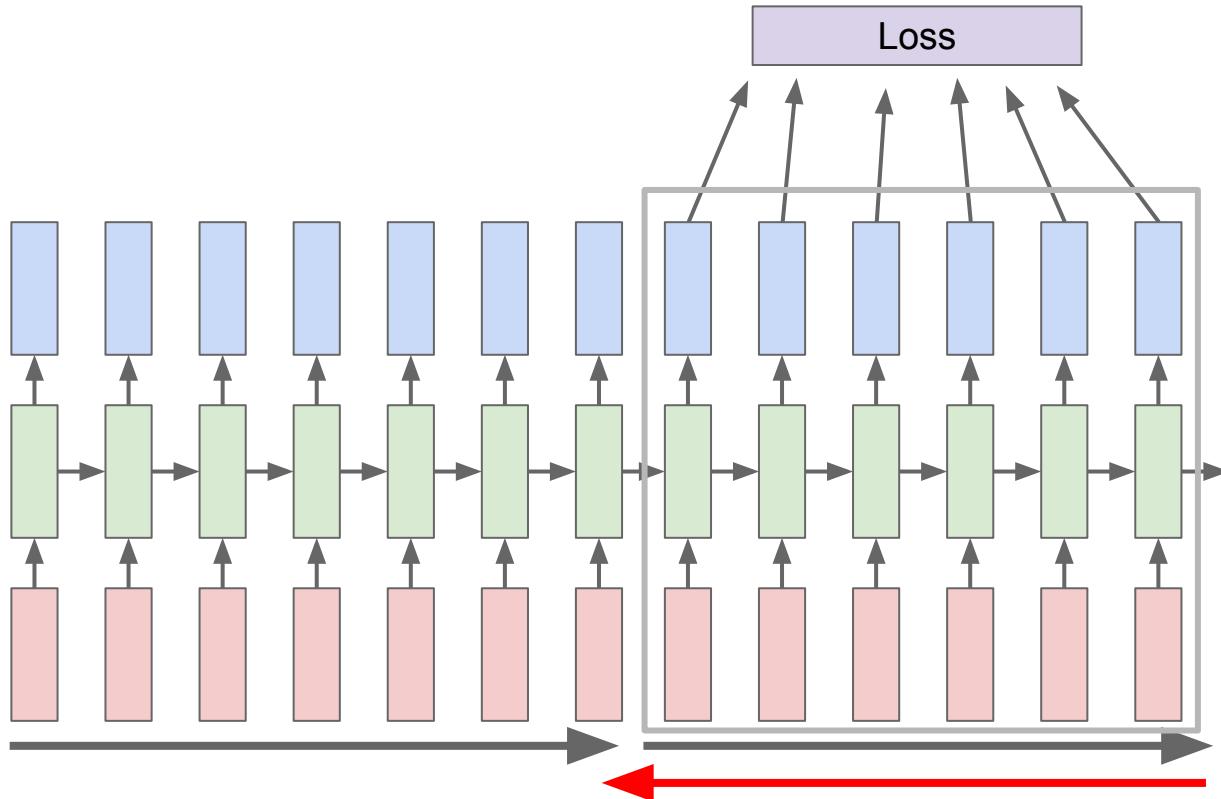


Truncated Backpropagation through time



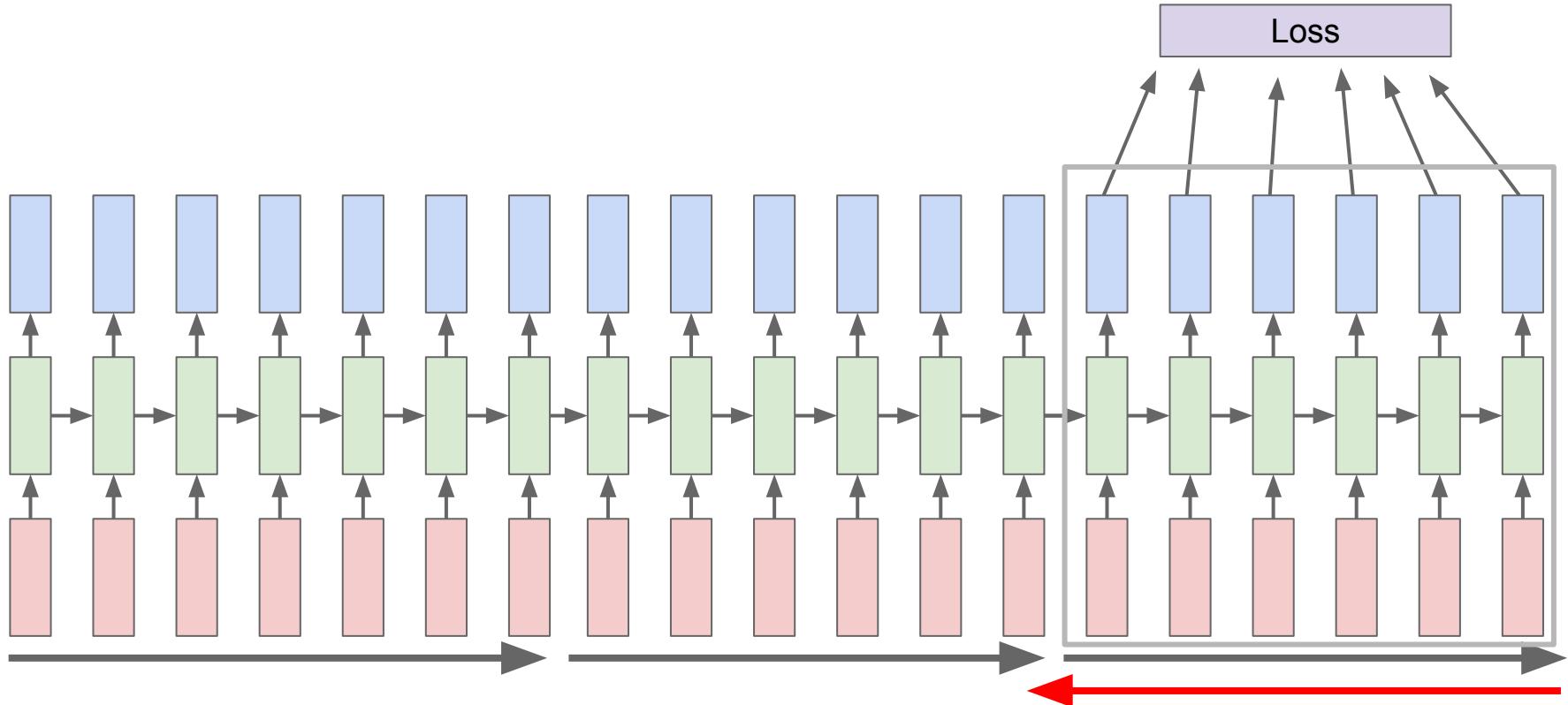
Run forward and backward
through chunks of the
sequence instead of whole
sequence

Truncated Backpropagation through time



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

Truncated Backpropagation through time



min-char-rnn.py gist: 112 lines of Python

```
1  """
2  Minimal character-level Vanilla RNN model. Written by Andrej Karpathy (@karpathy)
3  BSD License
4  """
5  import numpy as np
6
7  # data I/O
8  data = open('input.txt', 'r').read() # should be simple plain text file
9  chars = list(set(data))
10 data_size, vocab_size = len(data), len(chars)
11 print('data has %d characters, %d unique.' % (data_size, vocab_size))
12 char_to_ix = {ch:i for i,ch in enumerate(chars)}
13 ix_to_char = {i:ch for i,ch in enumerate(chars)}
14
15 # hyperparameters
16 hidden_size = 100 # size of hidden layer of neurons
17 seq_length = 25 # number of steps to unroll the RNN for
18 learning_rate = 1e-1
19
20 # model parameters
21 wkh = np.random.rand(hidden_size, vocab_size)*0.01 # input to hidden
22 whh = np.random.rand(hidden_size, hidden_size)*0.01 # hidden to hidden
23 why = np.random.rand(vocab_size, hidden_size)*0.01 # hidden to output
24 bh = np.zeros((hidden_size, 1)) # hidden bias
25 by = np.zeros((vocab_size, 1)) # output bias
26
27 def lossFun(inputs, targets, hprev):
28     """
29     inputs,targets are both list of integers.
30     hprev is Hx1 array of initial hidden state
31     returns the loss, gradients on model parameters, and last hidden state
32     """
33     xs, hs, ys, ps = [], [], [], []
34     hs[-1] = np.copy(hprev)
35     loss = 0
36     # forward pass
37     for t in xrange(len(inputs)):
38         xs[t] = np.zeros((vocab_size,1)) # encode in 1-of-k representation
39         xs[t][inputs[t]] = 1
40         hs[t] = np.tanh(np.dot(wkh, xs[t]) + np.dot(whh, hs[-1]) + bh) # hidden state
41         ys[t] = np.dot(why, hs[t]) - by # unnormalized log probabilities for next chars
42         ps[t] = np.exp(ys[t]) / np.sum(np.exp(ys[t])) # probabilities for next chars
43         loss += -np.log(ps[t][targets[t]]) # softmax (cross-entropy loss)
44
45         # backward pass: compute gradients going backwards
46         dwhx, dwhh, dwhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
47         dbh, dby = np.zeros_like(bh), np.zeros_like(by)
48         dhnext = np.zeros_like(hs[0])
49         for t2 in reversed(xrange(len(inputs))):
50             dy = np.copy(ps[t2])
51             dy[targets[t2]] -= 1 # backprop into y
52             dyb = -np.dot(dy, hs[t2].T)
53             dh = np.dot(why.T, dy) + dhnext # backprop into h
54             ddraw = (i - hs[t2].T) * dh # backprop through tanh nonlinearity
55             dbh += ddraw
56             dwhx += np.dot(ddraw, xs[t2].T)
57             dwhh += np.dot(ddraw, hs[t2].T)
58             dhnext = np.dot(why.T, ddraw)
59             for dparam in [dwhx, dwhh, dwhy, dbh, dby]:
60                 np.clip(dparam, -5, 5, out=dparam) # clip to mitigate exploding gradients
61
62     return loss, dwhx, dwhh, dwhy, dbh, dby, hs[len(inputs)-1]
```

```
63 def sample(h, seed_ix, n):
64     """
65     sample a sequence of integers from the model
66     h is memory state, seed_ix is seed letter for first time step
67     """
68     x = np.zeros((vocab_size, 1))
69     x[seed_ix] = 1
70     ixes = []
71     for t in xrange(n):
72         h = np.tanh(np.dot(wkh, x) + np.dot(whh, h) + bh)
73         y = np.dot(why, h) + by
74         p = np.exp(y) / np.sum(np.exp(y))
75         ix = np.random.choice(range(vocab_size), p=p.ravel())
76         x = np.zeros((vocab_size, 1))
77         x[ix] = 1
78         ixes.append(ix)
79
80     return ixes
81
82 n, p = 0, 0
83 mxwh, mwhh, mwhy = np.zeros_like(wkh), np.zeros_like(whh), np.zeros_like(why)
84 mbh, mbp = np.zeros_like(bh), np.zeros_like(by) # memory variables for Adagrad
85 smooth_loss = -np.log(1.0/vocab_size)*seq_length # loss at iteration 0
86 while True:
87     # prepare inputs (we're sweeping from left to right in steps seq_length long)
88     if p+seq_length >= len(data) or n == 0:
89         hprev = np.zeros((hidden_size,1)) # reset RNN memory
90         p = 0 # go from start of data
91     inputs = [char_to_ix[ch] for ch in data[p:p+seq_length]]
92     targets = [char_to_ix[ch] for ch in data[p+1:p+seq_length+1]]
93
94     # sample from the model now and then
95     if n % 100 == 0:
96         sample_ix = sample(hprev, inputs[0], 200)
97         txt = ''.join(ix_to_char[ix] for ix in sample_ix)
98         print('----\n%s\n----' % (txt,))
99
100    # forward seq_length characters through the net and fetch gradient
101    loss, dwhx, dwhh, dwhy, dbh, dby, hprev = lossFun(inputs, targets, hprev)
102    smooth_loss = smooth_loss * .999 + loss * .001
103    if n % 100 == 0: print('iter %d, loss: %f' % (n, smooth_loss)) # print progress
104
105    # perform parameter update with Adagrad
106    for param, dparam, mem in zip([wkh, whh, why, bh, by],
107                                 [dwhx, dwhh, dwhy, dbh, dby],
108                                 [mxwh, mwhh, mwhy, mbh, mbp]):
109        mem += dparam * dparam
110        param -= learning_rate * param / np.sqrt(mem + 1e-8) # adagrad update
111
112    p += seq_length # move data pointer
113    n += 1 # iteration counter
```

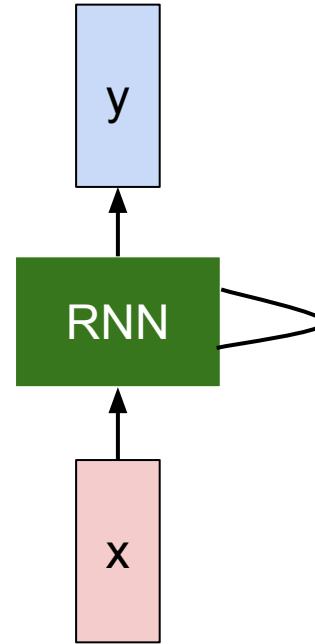
(<https://gist.github.com/karpathy/d4dee566867f8291f086>)

THE SONNETS

by William Shakespeare

From fairest creatures we desire increase,
That thereby beauty's rose might never die,
But as the riper should by time decease,
His tender heir might bear his memory:
But thou, contracted to thine own bright eyes,
Feed'st thy light's flame with self-substantial fuel,
Making a famine where abundance lies,
Thyself thy foe, to thy sweet self too cruel:
Thou that art now the world's fresh ornament,
And only herald to the gaudy spring,
Within thine own bud buriest thy content,
And tender churl mak'st waste in niggarding:
 Pity the world, or else this glutton be,
 To eat the world's due, by the grave and thee.

When forty winters shall besiege thy brow,
And dig deep trenches in thy beauty's field,
Thy youth's proud livery so gazed on now,
Will be a tatter'd weed of small worth held:
Then being asked, where all thy beauty lies,
Where all the treasure of thy lusty days;
To say, within thine own deep sunken eyes,
Were an all-eating shame, and thriftless praise.
How much more praise deserv'd thy beauty's use,
If thou couldst answer 'This fair child of mine
Shall sum my count, and make my old excuse,'
Proving his beauty by succession thine!
 This were to be new made when thou art old,
 And see thy blood warm when thou feel'st it cold.



at first:

tyntd-iafhatawiaoihrdemot lytdws e ,tfti, astai f ogoh eoase rrranbyne 'nhthnee e
plia tkldrgd t o idoe ns,smtt h ne etie h,hregtrs nigtike,aoaenns lng

↓ train more

"Tmont thithey" fomesscerliund
Keushey. Thom here
sheulke, anmerenith ol sivh I lalterthend Bleipile shuwyl fil on aseterlome
coaniogennc Phe lism thond hon at. MeiDimorotion in ther thize."

↓ train more

Aftair fall unsuch that the hall for Prince Velzonski's that me of
her hearly, and behs to so arwage fiving were to it beloge, pavu say falling misfort
how, and Gogition is so overelical and ofter.

↓ train more

"Why do what that day," replied Natasha, and wishing to himself the fact the
princess, Princess Mary was easier, fed in had oftened him.
Pierre aking his soul came to the packs and drove up his father-in-law women.

PANDARUS:

Alas, I think he shall be come approached and the day
When little strain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:

They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENTIO:

Well, your wit is in the care of side and that.

Second Lord:

They would be ruled after this chamber, and
my fair nues begun out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:

Come, sir, I will make did behold your worship.

VIOLA:

I'll drink it.

VIOLA:

Why, Salisbury must find his flesh and thought
That which I am not aps, not a man and in fire,
To show the reining of the raven and the wars
To grace my hand reproach within, and not a fair are hand,
That Caesar and my goodly father's world;
When I was heaven of presence and our fleets,
We spare with hours, but cut thy council I am great,
Murdered and by thy master's ready there
My power to give thee but so much as hell:
Some service in the noble bondman here,
Would show him to her wine.

KING LEAR:

O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

The Stacks Project: open source algebraic geometry textbook

The screenshot shows the homepage of the Stacks Project. At the top, there is a navigation bar with links: home, about, tags explained, tag lookup, browse, search, bibliography, recent comments, blog, and add slogans. Below the navigation bar, there is a section titled "Browse chapters". This section contains a table with two columns: "Part" and "Chapter". The "Part" column lists "Preliminaries", "Algebraic Spaces", "Topics in Scheme Theory", "Deformation Theory", "Algebraic Stacks", and "Miscellany". The "Chapter" column lists numbered chapters from 1 to 10, each with three download links: "online", "TeX", and "pdf". To the right of the table, there is a sidebar with sections for "Parts" and "Statistics". The "Parts" section lists the same categories as the table. The "Statistics" section provides information about the project's size: 455910 lines of code, 14221 tags (56 inactive tags), and 2366 sections.

Part	Chapter	online	TeX	view pdf
Preliminaries	1. Introduction	online	tex	pdf
	2. Conventions	online	tex	pdf
	3. Set Theory	online	tex	pdf
	4. Categories	online	tex	pdf
	5. Topology	online	tex	pdf
	6. Sheaves on Spaces	online	tex	pdf
	7. Sites and Sheaves	online	tex	pdf
	8. Stacks	online	tex	pdf
	9. Fields	online	tex	pdf
	10. Commutative Algebra	online	tex	pdf

Latex source

<http://stacks.math.columbia.edu/>

The stacks project is licensed under the [GNU Free Documentation License](#)

For $\bigoplus_{n=1,\dots,m} \mathcal{L}_{m,n} = 0$, hence we can find a closed subset \mathcal{H} in \mathcal{H} and any sets \mathcal{F} on X , U is a closed immersion of S , then $U \rightarrow T$ is a separated algebraic space.

Proof. Proof of (1). It also start we get

$$S = \text{Spec}(R) = U \times_X U \times_X U$$

and the comparicoly in the fibre product covering we have to prove the lemma generated by $\coprod Z \times_U U \rightarrow V$. Consider the maps M along the set of points Sch_{fppf} and $U \rightarrow U$ is the fibre category of S in U in Section, ?? and the fact that any U affine, see Morphisms, Lemma ???. Hence we obtain a scheme S and any open subset $W \subset U$ in $\text{Sh}(G)$ such that $\text{Spec}(R') \rightarrow S$ is smooth or an

$$U = \bigcup U_i \times_{S_i} U_i$$

which has a nonzero morphism we may assume that f_i is of finite presentation over S . We claim that $\mathcal{O}_{X,x}$ is a scheme where $x, x', s'' \in S'$ such that $\mathcal{O}_{X,x'} \rightarrow \mathcal{O}'_{X',x'}$ is separated. By Algebra, Lemma ?? we can define a map of complexes $\text{GL}_{S'}(x'/S'')$ and we win. \square

To prove study we see that $\mathcal{F}|_U$ is a covering of \mathcal{X}' , and \mathcal{T}_i is an object of $\mathcal{F}_{X/S}$ for $i > 0$ and \mathcal{F}_p exists and let \mathcal{F}_i be a presheaf of \mathcal{O}_X -modules on \mathcal{C} as a \mathcal{F} -module. In particular $\mathcal{F} = U/\mathcal{F}$ we have to show that

$$\widetilde{M}^\bullet = \mathcal{I}^\bullet \otimes_{\text{Spec}(k)} \mathcal{O}_{S,s} - i_X^{-1} \mathcal{F}$$

is a unique morphism of algebraic stacks. Note that

$$\text{Arrows} = (\text{Sch}/S)_{fppf}^{\text{opp}}, (\text{Sch}/S)_{fppf}$$

and

$$V = \Gamma(S, \mathcal{O}) \rightarrow (U, \text{Spec}(A))$$

is an open subset of X . Thus U is affine. This is a continuous map of X is the inverse, the groupoid scheme S .

Proof. See discussion of sheaves of sets. \square

The result for prove any open covering follows from the less of Example ???. It may replace S by $X_{\text{spaces},\text{étale}}$ which gives an open subspace of X and T equal to S_{Zar} , see Descent, Lemma ???. Namely, by Lemma ?? we see that R is geometrically regular over S .

Lemma 0.1. Assume (3) and (3) by the construction in the description.

Suppose $X = \lim |X|$ (by the formal open covering X and a single map $\underline{\text{Proj}}_X(\mathcal{A}) = \text{Spec}(B)$ over U compatible with the complex

$$\text{Set}(\mathcal{A}) = \Gamma(X, \mathcal{O}_{X,\mathcal{O}_X}).$$

When in this case of to show that $\mathcal{Q} \rightarrow \mathcal{C}_{Z/X}$ is stable under the following result in the second conditions of (1), and (3). This finishes the proof. By Definition ?? (without element is when the closed subschemes are catenary. If T is surjective we may assume that T is connected with residue fields of S . Moreover there exists a closed subspace $Z \subset X$ of X where U in X' is proper (some defining as a closed subset of the uniqueness it suffices to check the fact that the following theorem

(1) f is locally of finite type. Since $S = \text{Spec}(R)$ and $Y = \text{Spec}(R)$.

Proof. This is form all sheaves of sheaves on X . But given a scheme U and a surjective étale morphism $U \rightarrow X$. Let $U \cap U = \coprod_{i=1,\dots,n} U_i$ be the scheme X over S at the schemes $X_i \rightarrow X$ and $U = \lim_i X_i$. \square

The following lemma surjective restrocomposes of this implies that $\mathcal{F}_{x_0} = \mathcal{F}_{x_0} = \mathcal{F}_{x,\dots,x_0}$.

Lemma 0.2. Let X be a locally Noetherian scheme over S , $E = \mathcal{F}_{X/S}$. Set $\mathcal{I} = \mathcal{J}_1 \subset \mathcal{I}'_n$. Since $\mathcal{I}^n \subset \mathcal{I}^n$ are nonzero over $i_0 \leq p$ is a subset of $\mathcal{J}_{n,0} \circ \mathcal{A}_2$ works.

Lemma 0.3. In Situation ???. Hence we may assume $q' = 0$.

Proof. We will use the property we see that p is the next functor (??). On the other hand, by Lemma ?? we see that

$$D(\mathcal{O}_{X'}) = \mathcal{O}_X(D)$$

where K is an F -algebra where δ_{n+1} is a scheme over S . \square

Proof. Omitted. □

Lemma 0.1. Let \mathcal{C} be a set of the construction.

Let \mathcal{C} be a gerber covering. Let \mathcal{F} be a quasi-coherent sheaves of \mathcal{O} -modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

Proof. This is an algebraic space with the composition of sheaves \mathcal{F} on $X_{\text{étale}}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{\text{morph}_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where \mathcal{G} defines an isomorphism $\mathcal{F} \rightarrow \mathcal{F}$ of \mathcal{O} -modules. □

Lemma 0.2. This is an integer \mathcal{Z} is injective.

Proof. See Spaces, Lemma ??.

Lemma 0.3. Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \rightarrow Y' \rightarrow Y \rightarrow Y \rightarrow Y' \times_X Y \rightarrow X.$$

be a morphism of algebraic spaces over S and Y .

Proof. Let X be a nonzero scheme of X . Let X be an algebraic space. Let \mathcal{F} be a quasi-coherent sheaf of \mathcal{O}_X -modules. The following are equivalent

- (1) \mathcal{F} is an algebraic space over S .
- (2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram

$$\begin{array}{ccccc}
 S & \xrightarrow{\quad} & & & \\
 \downarrow & & & & \\
 \xi & \longrightarrow & \mathcal{O}_{X'} & \nearrow & \\
 \text{gor}_s & & & & \\
 & & & & \\
 & & = \alpha' \longrightarrow & & X \\
 & & \downarrow & & \downarrow \\
 & & = \alpha' \longrightarrow \alpha & & \text{Spec}(K_\psi) \qquad \text{Mor}_{\text{Sets}} \qquad d(\mathcal{O}_{X_{X/k}}, \mathcal{G})
 \end{array}$$

is a limit. Then \mathcal{G} is a finite type and assume S is a flat and \mathcal{F} and \mathcal{G} is a finite type f_* . This is of finite type diagrams, and

- the composition of \mathcal{G} is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \text{Spec}(R)$ and \mathcal{F} is a finite type representable by algebraic space. The property \mathcal{F} is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U . □

Proof. This is clear that \mathcal{G} is a finite presentation, see Lemmas ??.

A reduced above we conclude that U is an open covering of \mathcal{C} . The functor \mathcal{F} is a “field”

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\bar{x}} \dashrightarrow^{-1} (\mathcal{O}_{X_{\text{étale}}}) \longrightarrow \mathcal{O}_{X_{\bar{x}}}^{-1} \mathcal{O}_{X_{\lambda}}(\mathcal{O}_{X_{\eta}}^{\text{v}})$$

is an isomorphism of covering of \mathcal{O}_{X_i} . If \mathcal{F} is the unique element of \mathcal{F} such that X is an isomorphism.

The property \mathcal{F} is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme \mathcal{O}_X -algebra with \mathcal{F} are opens of finite type over S .

If \mathcal{F} is a scheme theoretic image points. □

If \mathcal{F} is a finite direct sum \mathcal{O}_{X_k} is a closed immersion, see Lemma ???. This is a sequence of \mathcal{F} is a similar morphism.

This repository Search

Explore Gist Blog Help

karpathy + - ⌂ ⚙ ⌂

torvalds / linux

Watch 3,711 Star 23,054 Fork 9,141

Linux kernel source tree

520,037 commits 1 branch 420 releases 5,039 contributors

branch: master / linux / +

Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux ...

torvalds authored 9 hours ago latest commit 4b1786927d

Category	Commit Message	Time Ago
Documentation	Merge git://git.kernel.org/pub/scm/linux/kernel/git/nab/target-pending	6 days ago
arch	Merge branch 'x86-urgent-for-linus' of git://git.kernel.org/pub/scm/l...	a day ago
block	block: discard bdi_unregister() in favour of bdi_destroy()	9 days ago
crypto	Merge git://git.kernel.org/pub/scm/linux/kernel/git/herbert/crypto-2.6	10 days ago
drivers	Merge branch 'drm-fixes' of git://people.freedesktop.org/~airlied/linux	9 hours ago
firmware	firmware/ihex2fw.c: restore missing default in switch statement	2 months ago
fs	vfs: read file_handle only once in handle_to_path	4 days ago
include	Merge branch 'perl-urgent-for-linus' of git://git.kernel.org/pub/scm/...	a day ago
init	init: fix regression by supporting devices with major:minor:offset fo...	a month ago
ipc	ipc: fix race between NewRequest and multi-client transaction processing	a month ago

Pulse Graphs

HTTPS clone URL <https://github.com/torvalds/linux>

You can clone with HTTPS, SSH, or Subversion.

Clone in Desktop Download ZIP

```
static void do_command(struct seq_file *m, void *v)
{
    int column = 32 << (cmd[2] & 0x80);
    if (state)
        cmd = (int)(int_state ^ (in_8(&ch->ch_flags) & Cmd) ? 2 : 1);
    else
        seq = 1;
    for (i = 0; i < 16; i++) {
        if (k & (1 << 1))
            pipe = (in_use & UMXTHREAD_UNCCA) +
                ((count & 0x00000000fffffff8) & 0x0000000f) << 8;
        if (count == 0)
            sub(pid, ppc_md.kexec_handle, 0x20000000);
        pipe_set_bytes(i, 0);
    }
    /* Free our user pages pointer to place camera if all dash */
    subsystem_info = &of_changes[PAGE_SIZE];
    rek_controls(offset, idx, &soffset);
    /* Now we want to deliberately put it to device */
    control_check_polarity(&context, val, 0);
    for (i = 0; i < COUNTER; i++)
        seq_puts(s, "policy ");
}
```

Generated C code

```
/*
 * Copyright (c) 2006-2010, Intel Mobile Communications. All rights reserved.
 *
 * This program is free software; you can redistribute it and/or modify it
 * under the terms of the GNU General Public License version 2 as published by
 * the Free Software Foundation.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 *
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software Foundation,
 * Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
 */

#include <linux/kexec.h>
#include <linux/errno.h>
#include <linux/io.h>
#include <linux/platform_device.h>
#include <linux/multi.h>
#include <linux/ckevent.h>

#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/seteew.h>
#include <asm/pgproto.h>
```

```
#include <asm/io.h>
#include <asm/prom.h>
#include <asm/e820.h>
#include <asm/system_info.h>
#include <asm/seteew.h>
#include <asm/pgproto.h>

#define REG_PG      vesa_slot_addr_pack
#define PFM_NOCOMP  AFSR(0, load)
#define STACK_DDR(type)      (func)

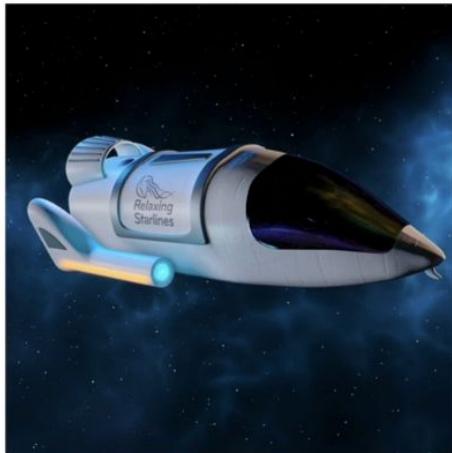
#define SWAP_ALLOCATE(nr)      (e)
#define emulate_sigs()  arch_get_unaligned_child()
#define access_rw(TST)  asm volatile("movd %esp, %0, %3" : : "r" (0)); \
    if (__type & DO_READ)

static void stat_PC_SEC __read_mostly offsetof(struct seq_argsqueue, \
    pC>[1]);

static void
os_prefix(unsigned long sys)
{
#ifdef CONFIG_PREEMPT
    PUT_PARAM_RAID(2, sel) = get_state_state();
    set_pid_sum((unsigned long)state, current_state_str(),
                (unsigned long)-1->lr_full, low;
}

```

OpenAI Codex



```
/* Add this image of a rocketship:  
https://i1.sndcdn.com/artworks-j8xjG7zc1wmTe07b-06183w-t500x500.jpg */  
var rocketship =  
document.createElement('img');  
rocketship.src =  
https://i1.sndcdn.com/artworks-j8xjG7zc1wmTe07b-06183w-t500x500.jpg;  
document.body.appendChild(rocketship);
```

Add this image of a rocketship:

<https://i1.sndcdn.com/artworks-j8xjG7zc1wmTe07b-06183w-t500x500.jpg>



OpenAI GPT-2 generated text

[source](#)

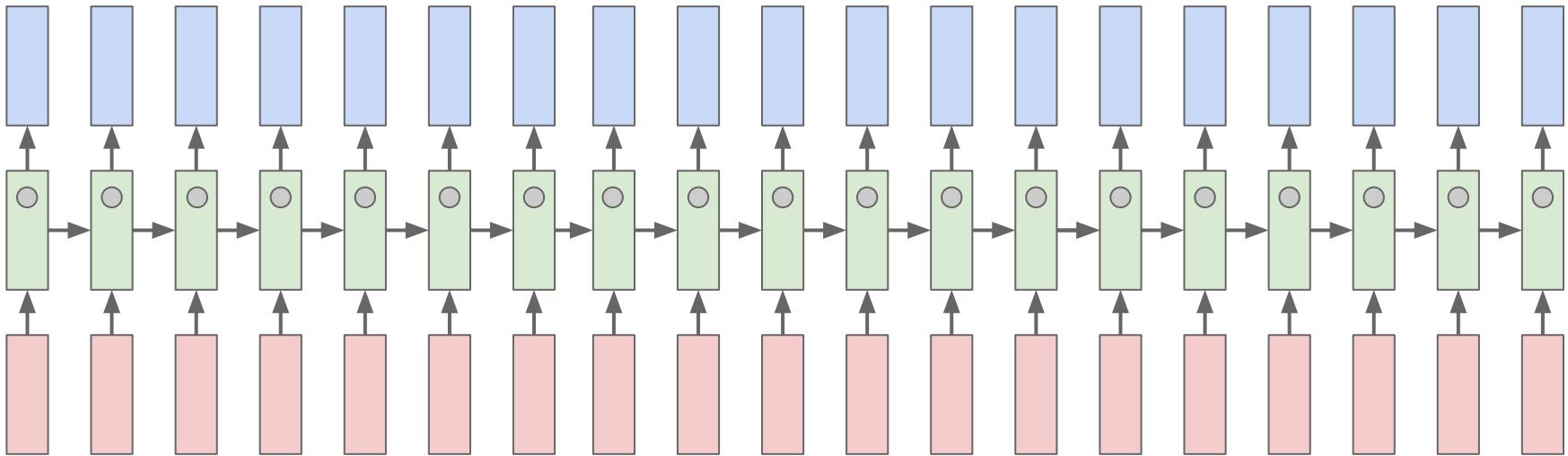
Input: In a shocking finding, scientist discovered a herd of unicorns living in a remote, previously unexplored valley, in the Andes Mountains. Even more surprising to the researchers was the fact that the unicorns spoke perfect English.

Output: The scientist named the population, after their distinctive horn, Ovid's Unicorn. These four-horned, silver-white unicorns were previously unknown to science.

Now, after almost two centuries, the mystery of what sparked this odd phenomenon is finally solved.

Dr. Jorge Pérez, an evolutionary biologist from the University of La Paz, and several companions, were exploring the Andes Mountains when they found a small valley, with no other animals or humans. Pérez noticed that the valley had what appeared to be a natural fountain, surrounded by two peaks of rock and silver snow.

Searching for interpretable cells



Searching for interpretable cells

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

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

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

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

quote detection cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

Cell sensitive to position in line:

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

line length tracking cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

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

if statement cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

Cell that turns on inside comments and quotes:

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

quote/comment cell

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

Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

Searching for interpretable cells

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

code depth cell

Karpathy, Johnson, and Fei-Fei: Visualizing and Understanding Recurrent Networks, ICLR Workshop 2016
Figures copyright Karpathy, Johnson, and Fei-Fei, 2015; reproduced with permission

RNN tradeoffs

RNN Advantages:

- Can process any length input
- Computation for step t can (in theory) use information from many steps back
- Model size doesn't increase for longer input
- Same weights applied on every timestep, so there is symmetry in how inputs are processed.

RNN Disadvantages:

- Recurrent computation is slow
- In practice, difficult to access information from many steps back

Image Captioning

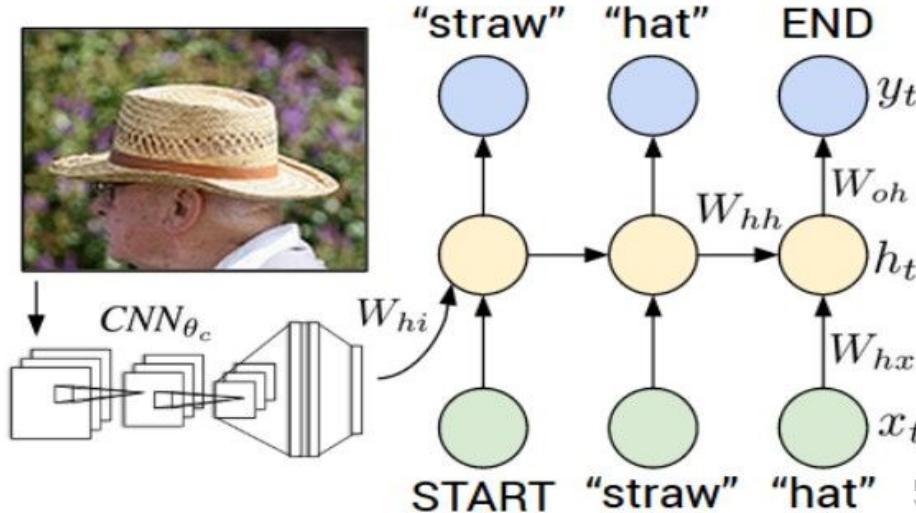


Figure from Karpathy et al, "Deep Visual-Semantic Alignments for Generating Image Descriptions", CVPR 2015; figure copyright IEEE, 2015.
Reproduced for educational purposes.

Explain Images with Multimodal Recurrent Neural Networks, Mao et al.

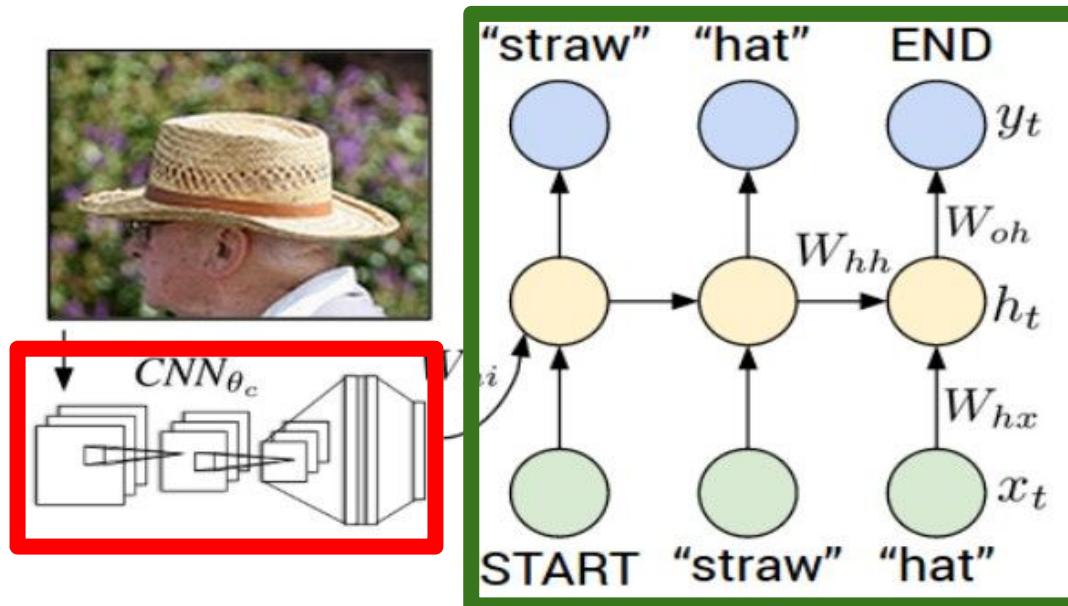
Deep Visual-Semantic Alignments for Generating Image Descriptions, Karpathy and Fei-Fei

Show and Tell: A Neural Image Caption Generator, Vinyals et al.

Long-term Recurrent Convolutional Networks for Visual Recognition and Description, Donahue et al.

Learning a Recurrent Visual Representation for Image Caption Generation, Chen and Zitnick

Recurrent Neural Network



Convolutional Neural Network

test image



[This image](#) is [CC0 public domain](#)

image



conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC-1000

softmax



test image

image



test image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

conv-512

conv-512

maxpool

FC-4096

FC-4096

FC1000

softmax

X

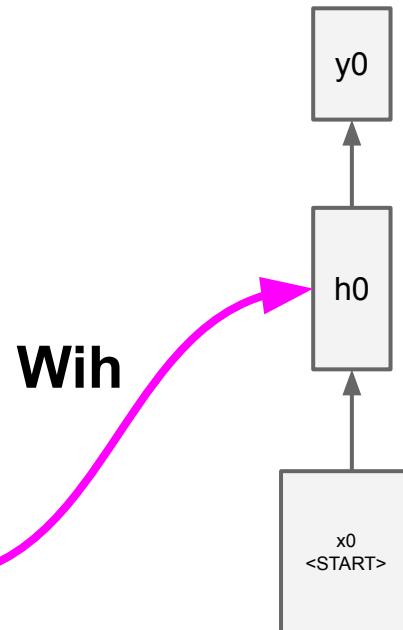


test image





test image



before:

$$h = \tanh(W_{xh} * x + W_{hh} * h)$$

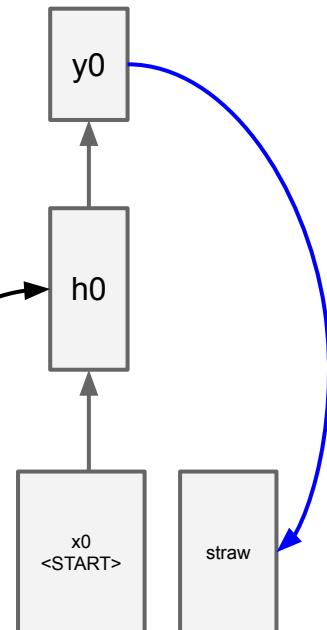
now:

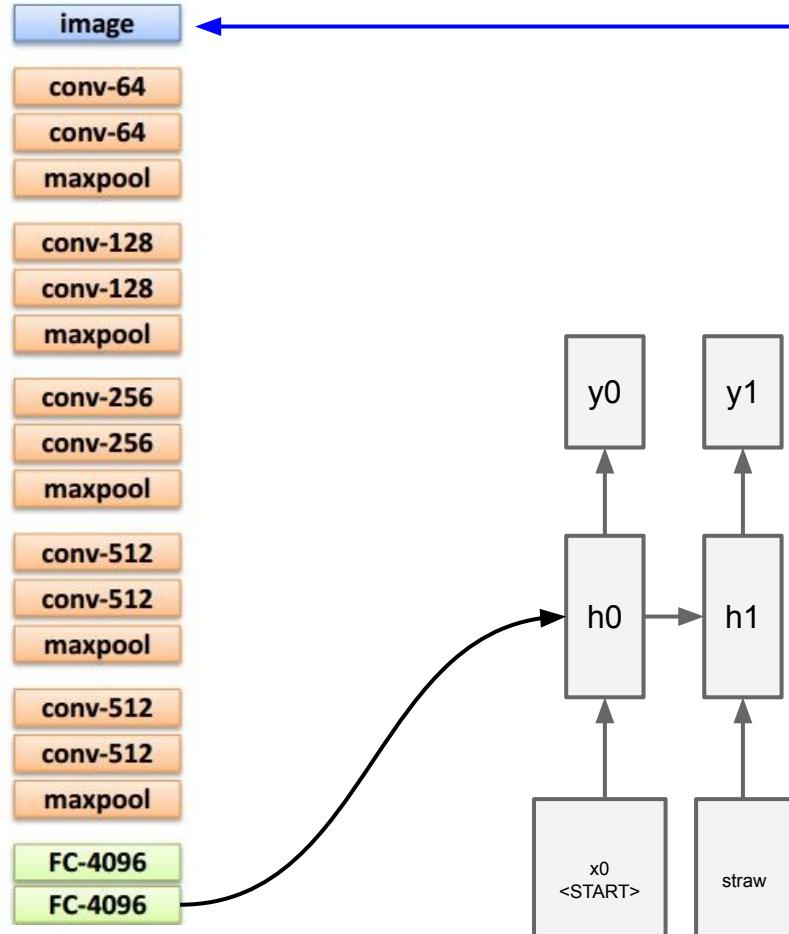
$$h = \tanh(W_{xh} * x + W_{hh} * h + W_{ih} * v)$$



test image

sample!

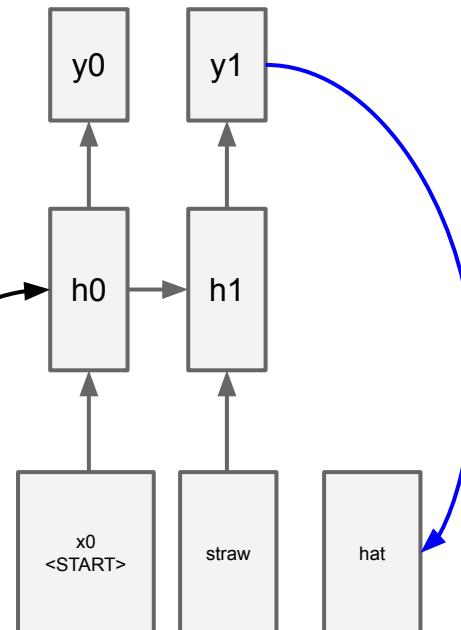




test image



test image



sample!

image



test image

conv-64

conv-64

maxpool

conv-128

conv-128

maxpool

conv-256

conv-256

maxpool

conv-512

conv-512

maxpool

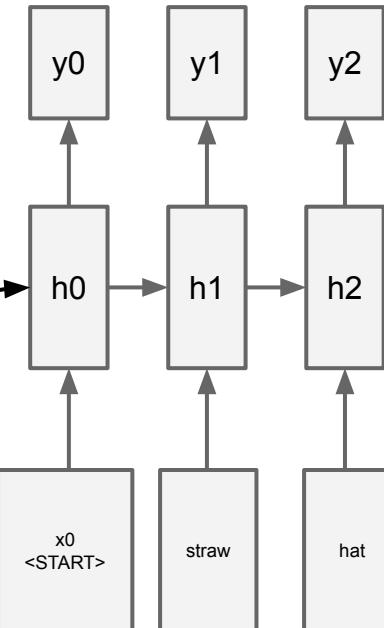
conv-512

conv-512

maxpool

FC-4096

FC-4096





test image

sample
<END> token
=> finish.

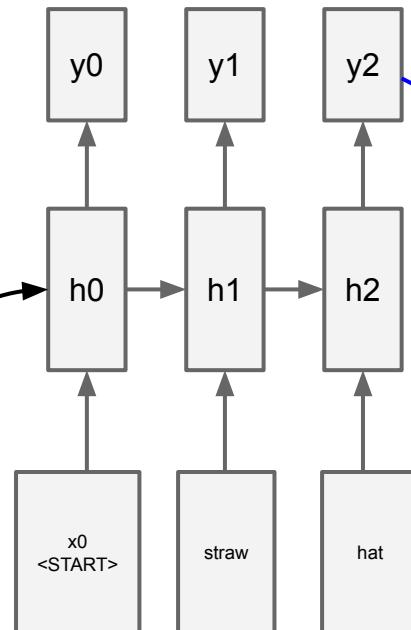


Image Captioning: Example Results



A cat sitting on a suitcase on the floor



A cat is sitting on a tree branch



A dog is running in the grass with a frisbee



A white teddy bear sitting in the grass



Two people walking on the beach with surfboards



A tennis player in action on the court



Two giraffes standing in a grassy field



A man riding a dirt bike on a dirt track

Image Captioning: Failure Cases



A woman is holding a cat in her hand



A person holding a computer mouse on a desk



A woman standing on a beach holding a surfboard



A bird is perched on a tree branch



A man in a baseball uniform throwing a ball

Visual Question Answering (VQA)



Q: What endangered animal is featured on the truck?

- A: A bald eagle.
- A: A sparrow.
- A: A humming bird.
- A: A raven.



Q: Where will the driver go if turning right?

- A: Onto 24 1/4 Rd.
- A: Onto 25 1/4 Rd.
- A: Onto 23 1/4 Rd.
- A: Onto Main Street.



Q: When was the picture taken?

- A: During a wedding.
- A: During a bar mitzvah.
- A: During a funeral.
- A: During a Sunday church service



Q: Who is under the umbrella?

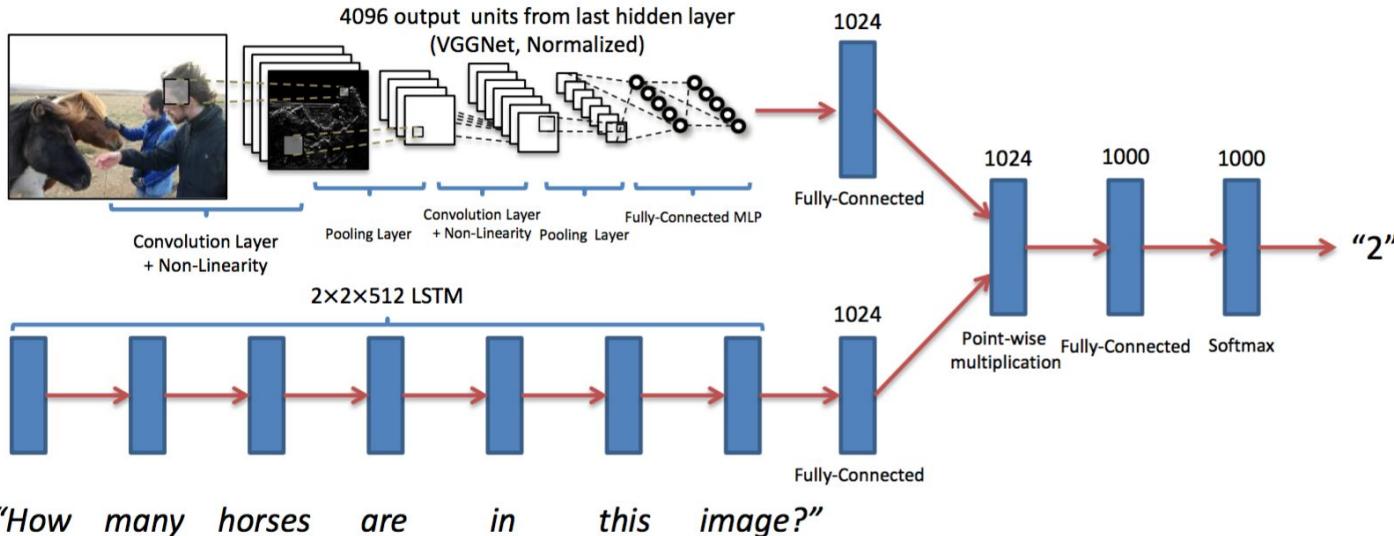
- A: Two women.
- A: A child.
- A: An old man.
- A: A husband and a wife.

Agrawal et al, "VQA: Visual Question Answering", ICCV 2015

Zhu et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2016

Figure from Zhu et al, copyright IEEE 2016. Reproduced for educational purposes.

Visual Question Answering (VQA)



Agrawal et al, "Visual 7W: Grounded Question Answering in Images", CVPR 2015
Figures from Agrawal et al, copyright IEEE 2015. Reproduced for educational purposes.

Visual Dialog: Conversations about images

Visual Dialog

A cat drinking water out of a coffee mug.

What color is the mug?

White and red

No, something is there can't tell what it is

Are there any pictures on it?

Yes, they are

Is the mug and cat on a table?

Are there other items on the table?

Yes, magazines, books, toaster and basket, and a plate

C Start typing question here ... >

Das et al, "Visual Dialog", CVPR 2017
Figures from Das et al, copyright IEEE 2017. Reproduced with permission.

Visual Language Navigation: Go to the living room

Agent encodes instructions in language and uses an RNN to generate a series of movements as the visual input changes after each move.

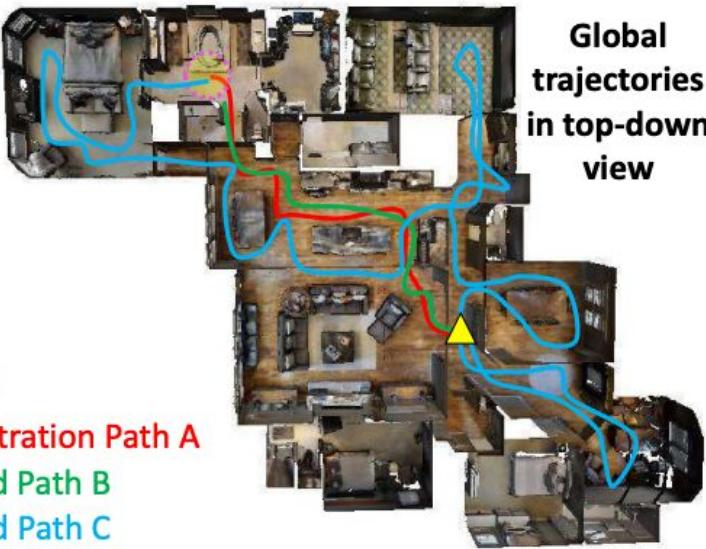
Instruction

Turn right and head towards the *kitchen*. Then turn left, pass a *table* and enter the *hallway*. Walk down the hallway and turn into the *entry way* to your right *without doors*. Stop in front of the *toilet*.

Local visual scene



Global trajectories in top-down view



Initial Position

Target Position

Demonstration Path A

Executed Path B

Executed Path C

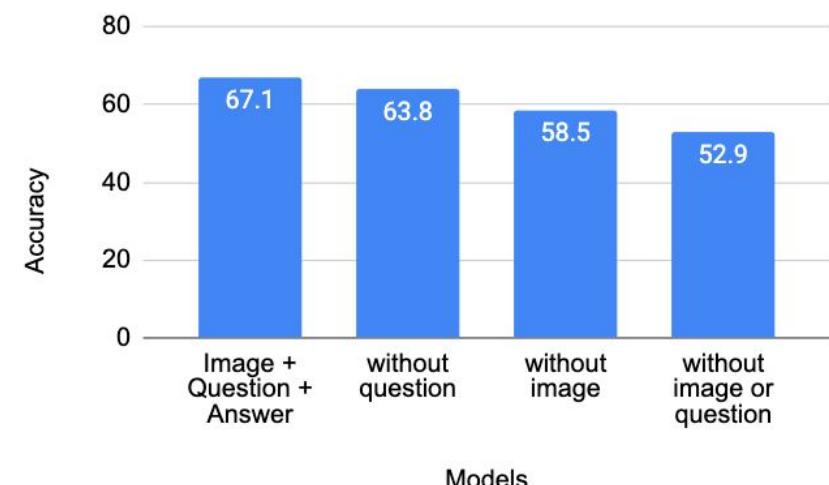
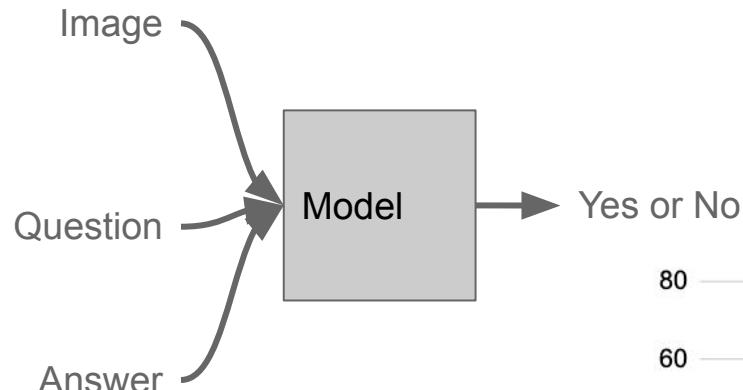
Wang et al, "Reinforced Cross-Modal Matching and Self-Supervised Imitation Learning for Vision-Language Navigation", CVPR 2018
Figures from Wang et al, copyright IEEE 2017. Reproduced with permission.

Visual Question Answering: Dataset Bias

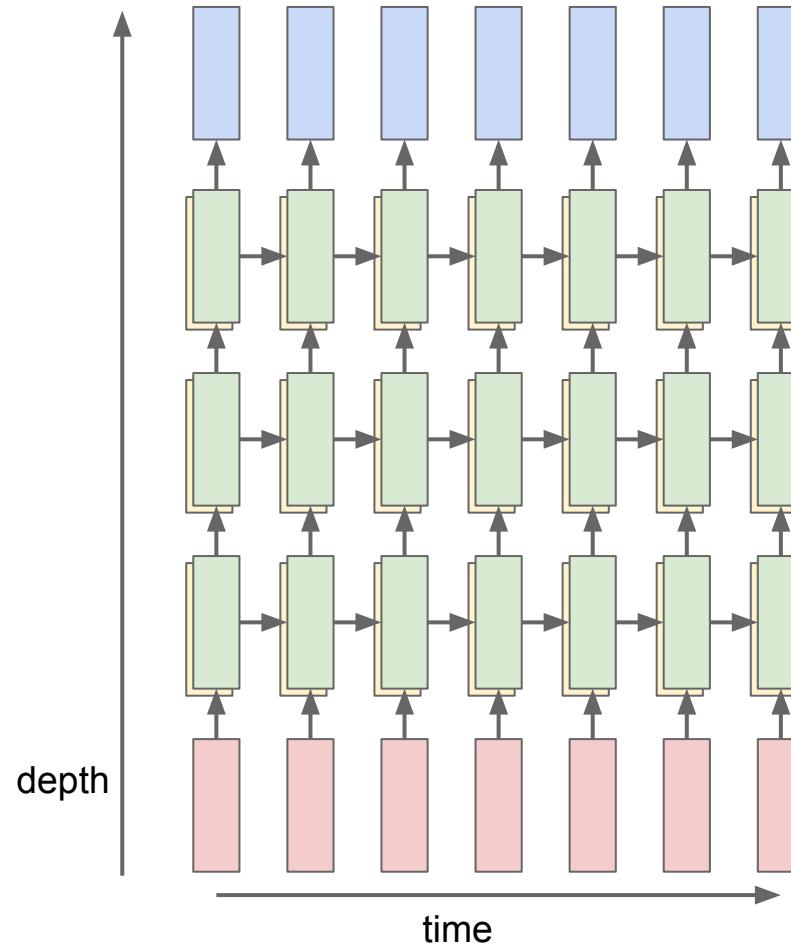


What is the dog
playing with?

Frisbee



Multilayer RNNs



Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

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

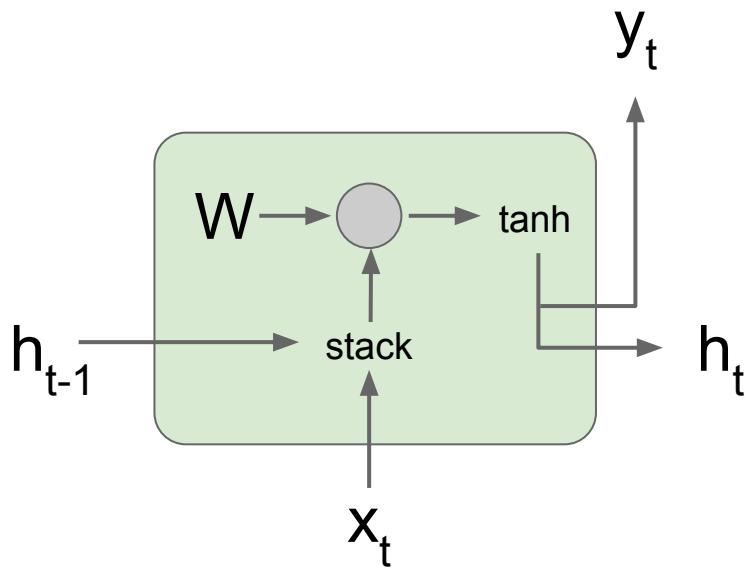
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

Vanilla RNN Gradient Flow

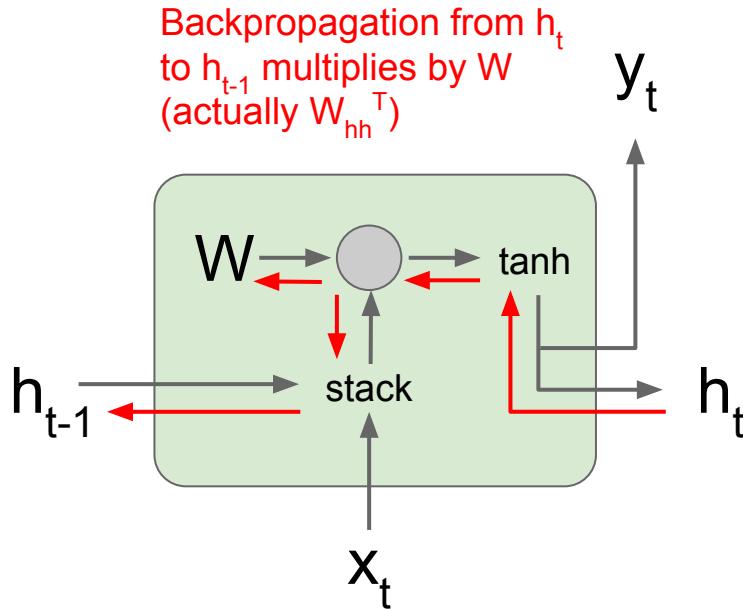
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Vanilla RNN Gradient Flow

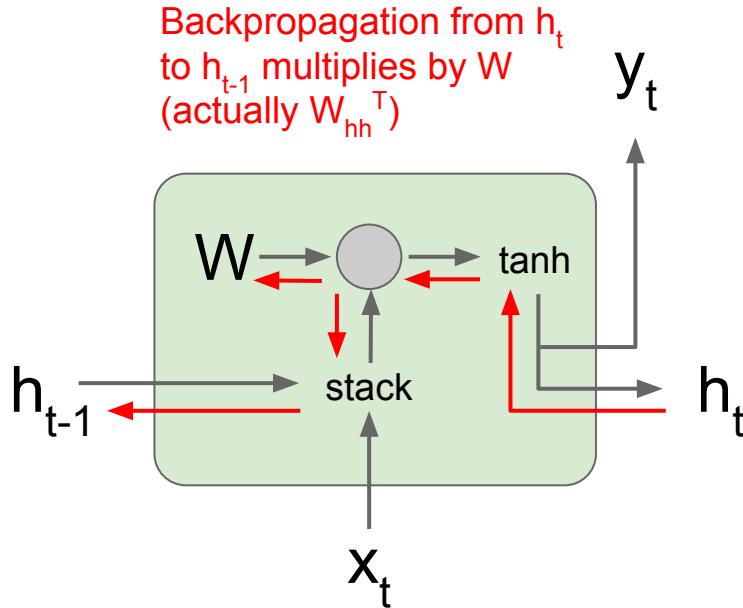
Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

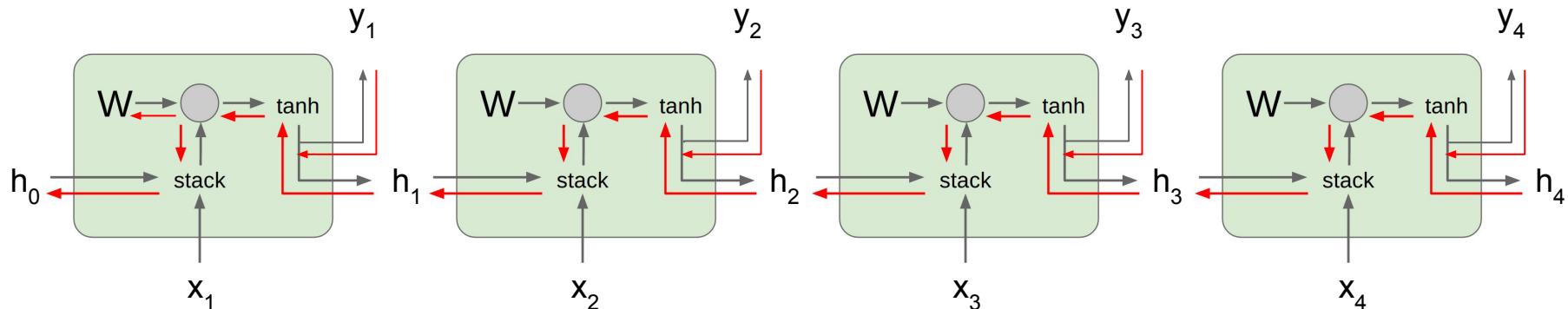


$$\begin{aligned} h_t &= \tanh(W_{hh}h_{t-1} + W_{xh}x_t) \\ &= \tanh \left(\begin{pmatrix} W_{hh} & W_{hx} \end{pmatrix} \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \\ &= \tanh \left(W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix} \right) \end{aligned}$$

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

Vanilla RNN Gradient Flow

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

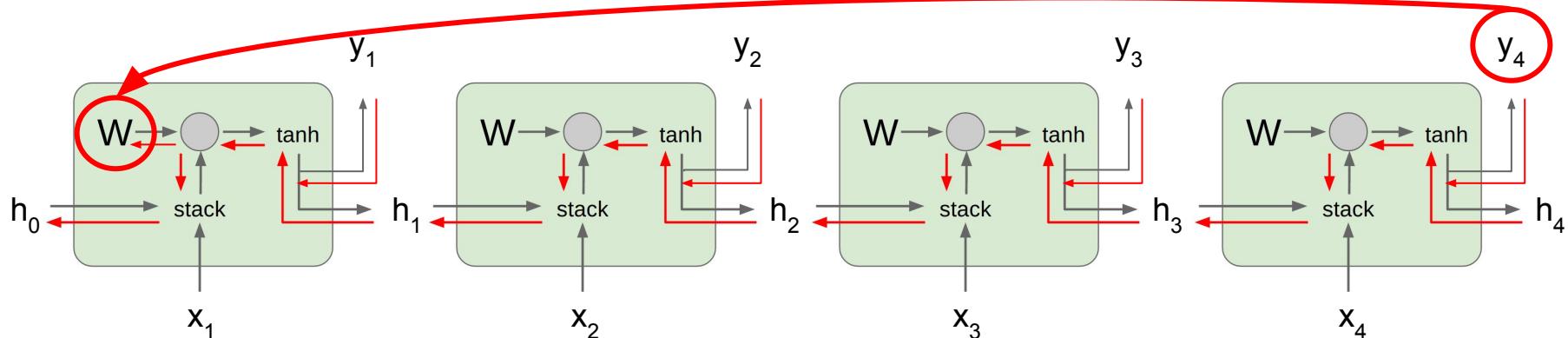


$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



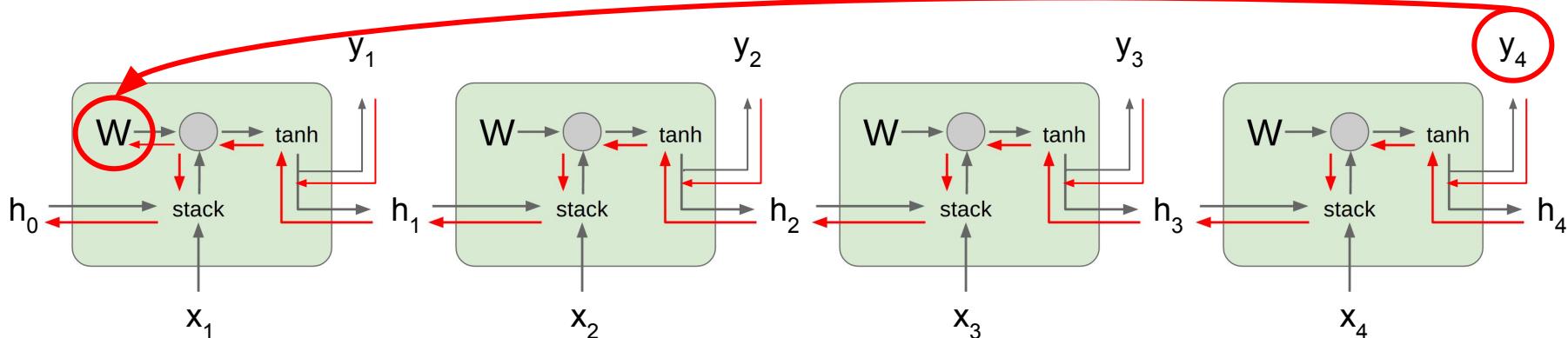
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



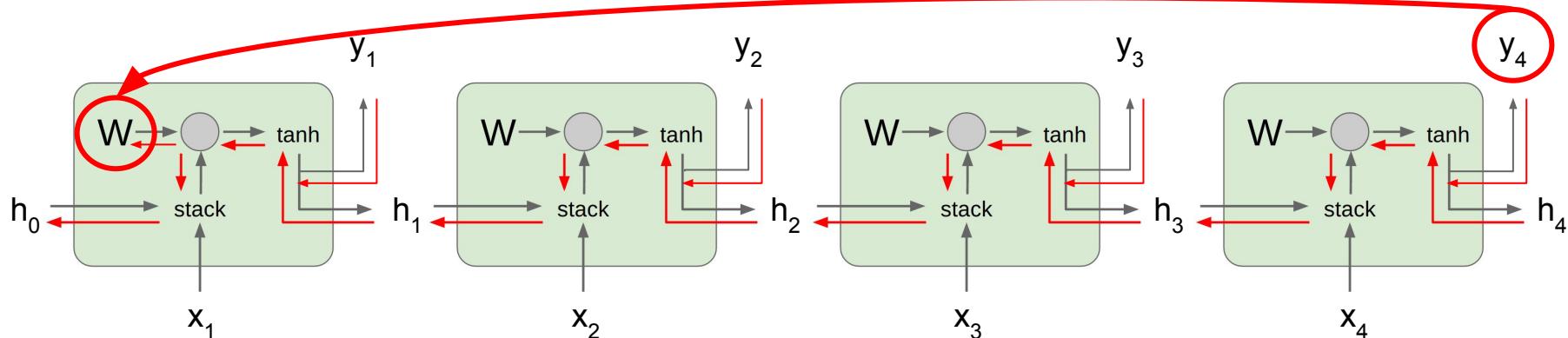
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

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

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

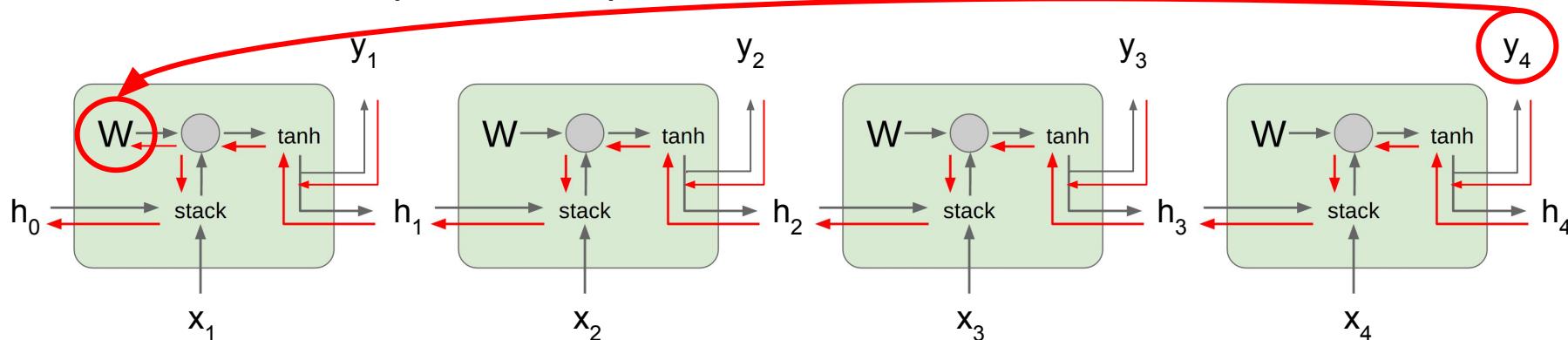
$$\frac{\partial h_t}{\partial h_{t-1}} = \tanh'(W_{hh}h_{t-1} + W_{xh}x_t)W_{hh}$$

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

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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

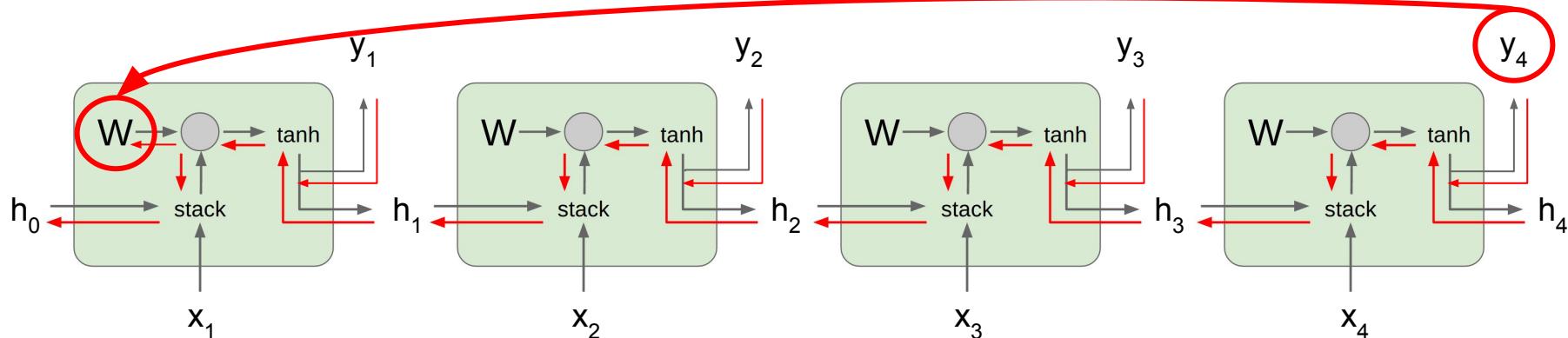
Almost always < 1
Vanishing gradients

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

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



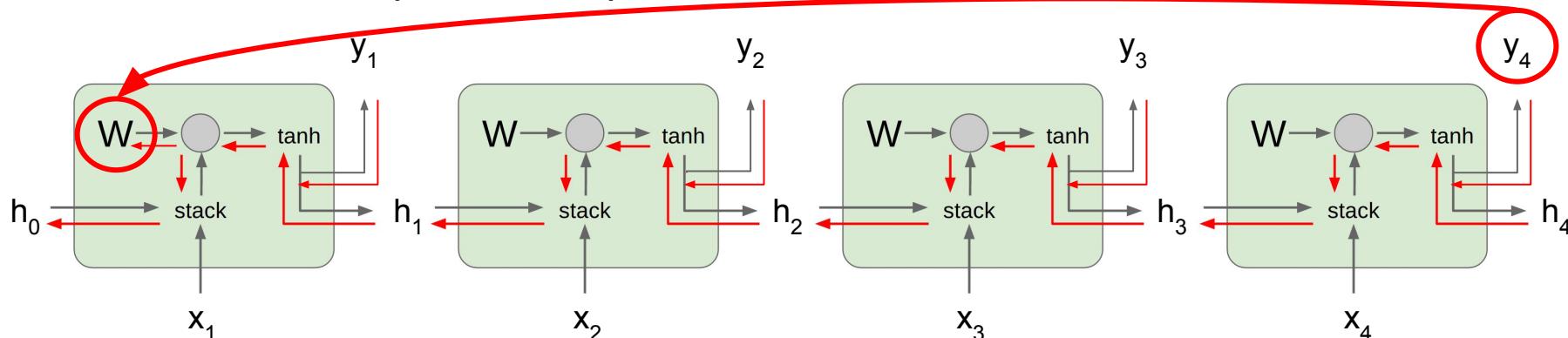
$$\frac{\partial L}{\partial W} = \sum_{t=1}^T \frac{\partial L_t}{\partial W}$$

What if we assumed no non-linearity?

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



What if we assumed no non-linearity?

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

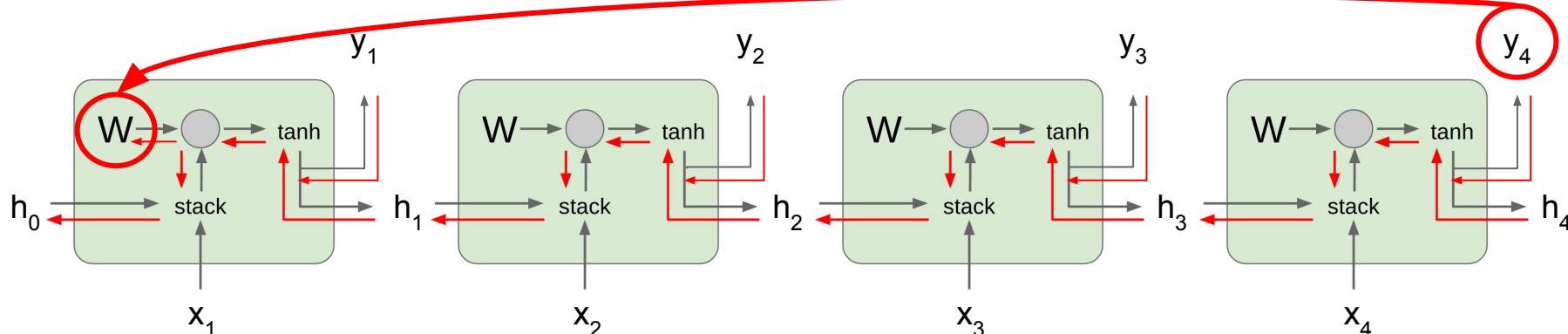
Largest singular value > 1 :
Exploding gradients

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

Largest singular value < 1 :
Vanishing gradients

Vanilla RNN Gradient Flow

Gradients over multiple time steps:



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

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013

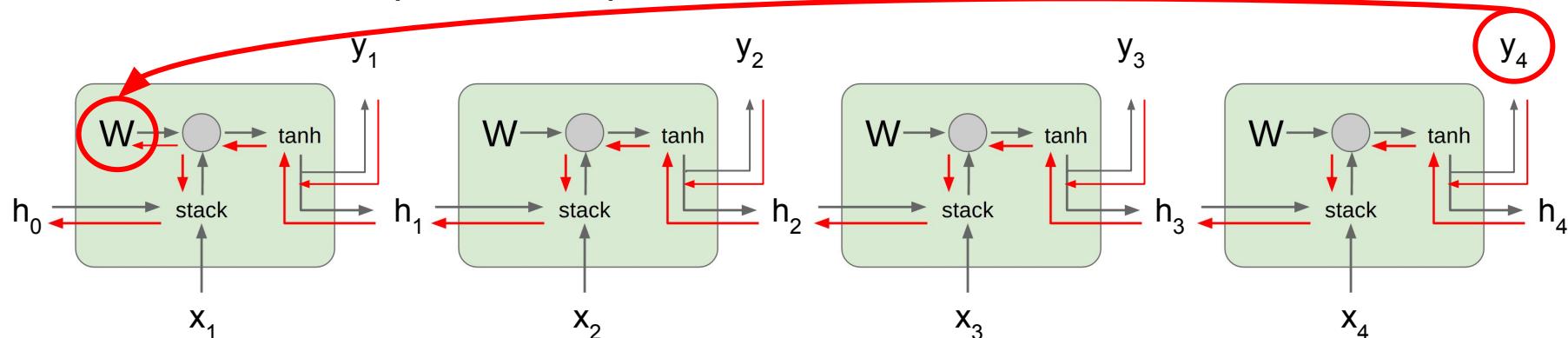
→ **Gradient clipping:**
Scale gradient if its norm is too big

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

Vanilla RNN Gradient Flow

Gradients over multiple time steps:

Bengio et al, "Learning long-term dependencies with gradient descent is difficult", IEEE Transactions on Neural Networks, 1994
Pascanu et al, "On the difficulty of training recurrent neural networks", ICML 2013



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)

Vanilla RNN

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

LSTM

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

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

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

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

Long Short Term Memory (LSTM)

Vanilla RNN

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

LSTM

Four gates

Cell state

Hidden state

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

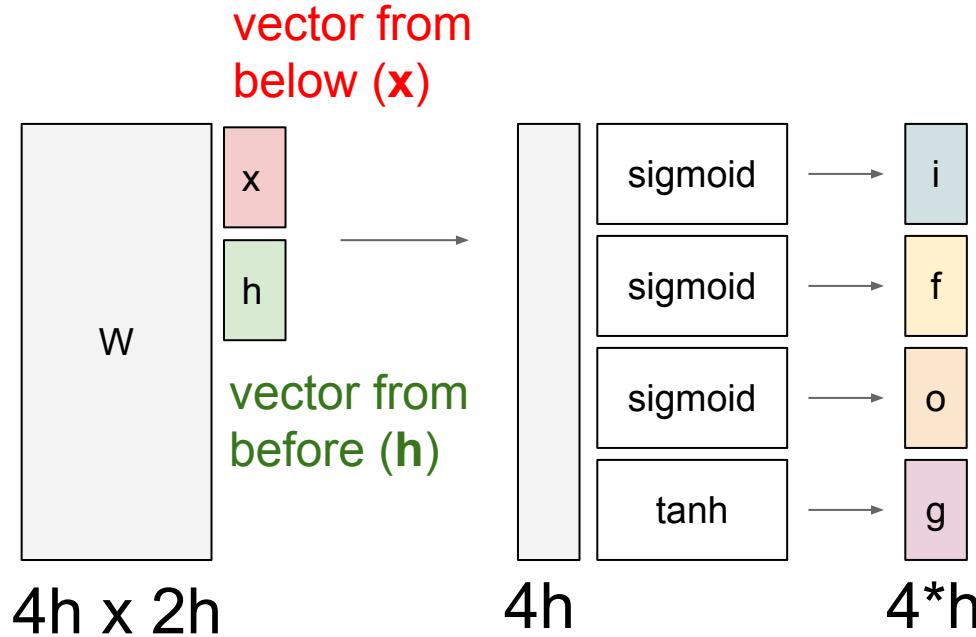
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Hochreiter and Schmidhuber, "Long Short Term Memory", Neural Computation 1997

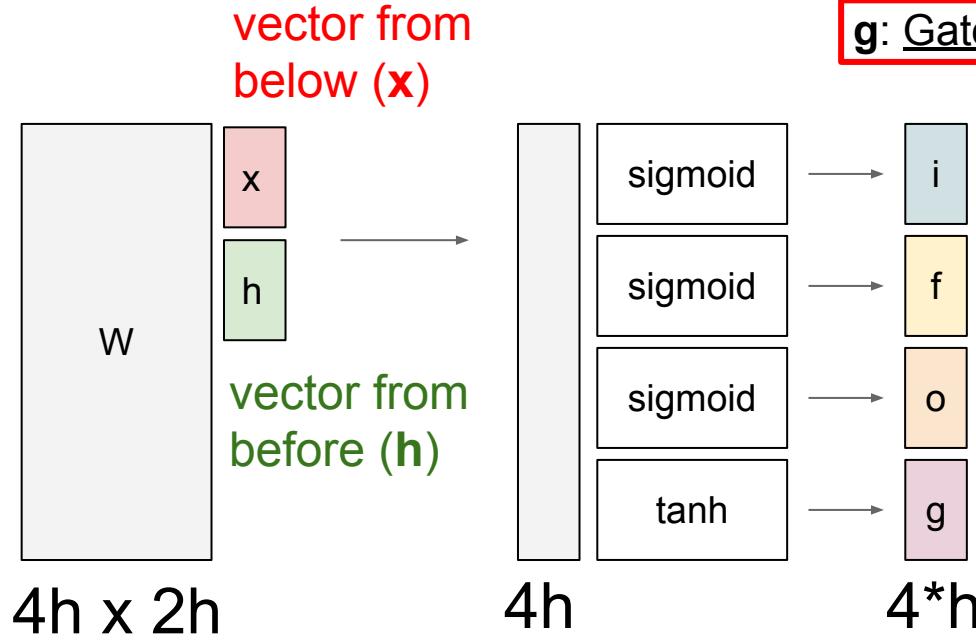
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



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

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

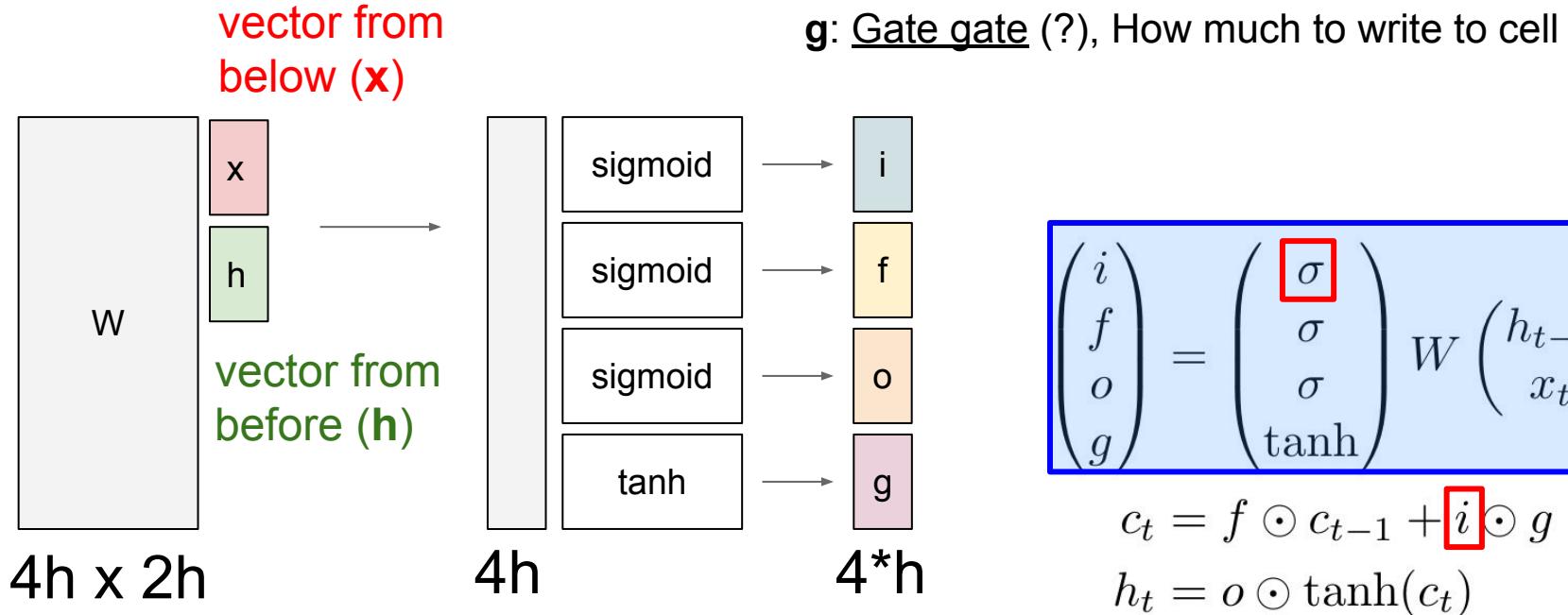
$$c_t = f \odot c_{t-1} + i \odot g$$

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

i: Input gate, whether to write to cell



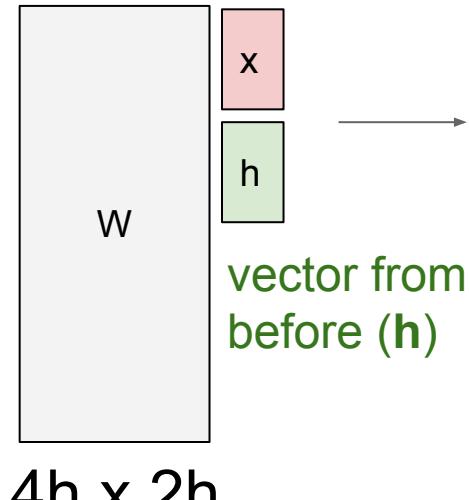
Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]

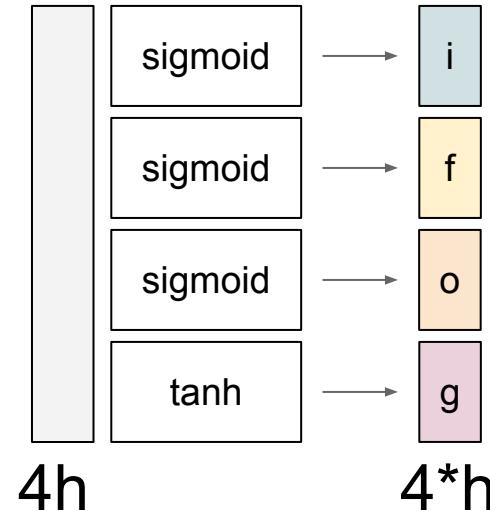
i: Input gate, whether to write to cell
f: Forget gate, Whether to erase cell

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

vector from
below (x)



vector from
before (h)



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

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

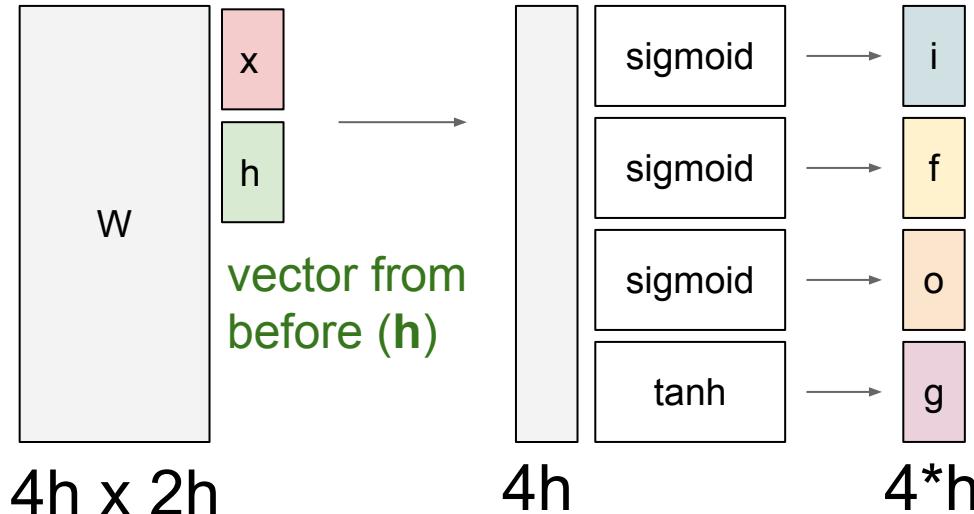
$$h_t = o \odot \tanh(c_t)$$

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

vector from
below (x)



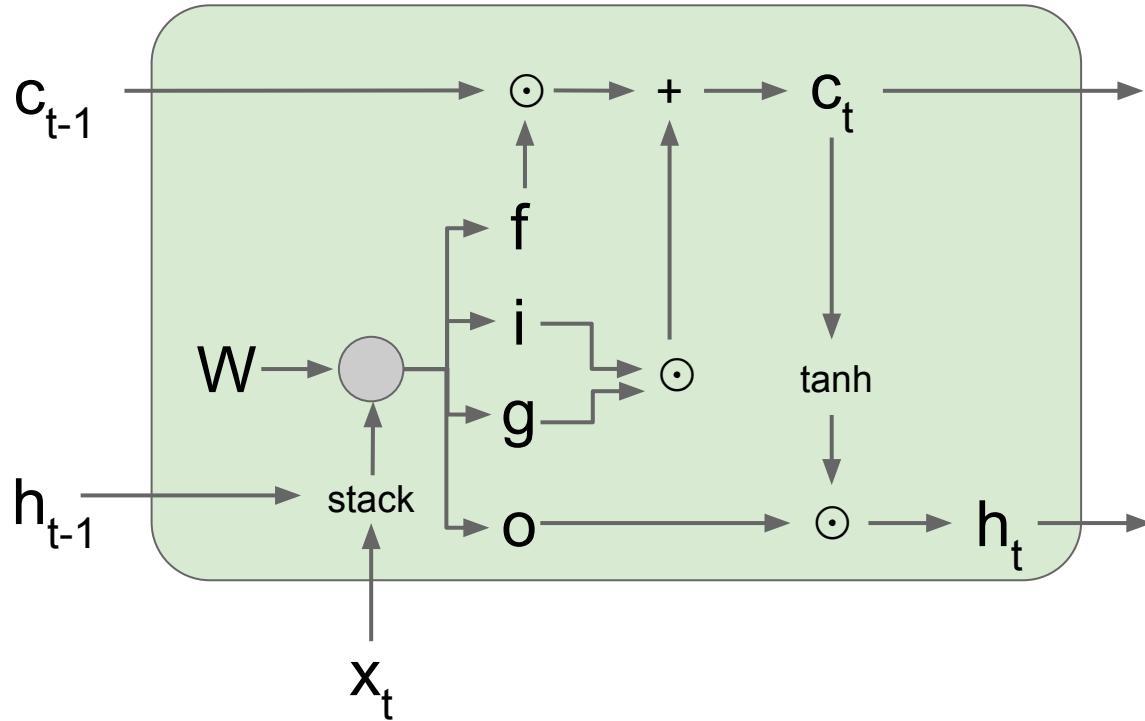
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

Long Short Term Memory (LSTM)

[Hochreiter et al., 1997]



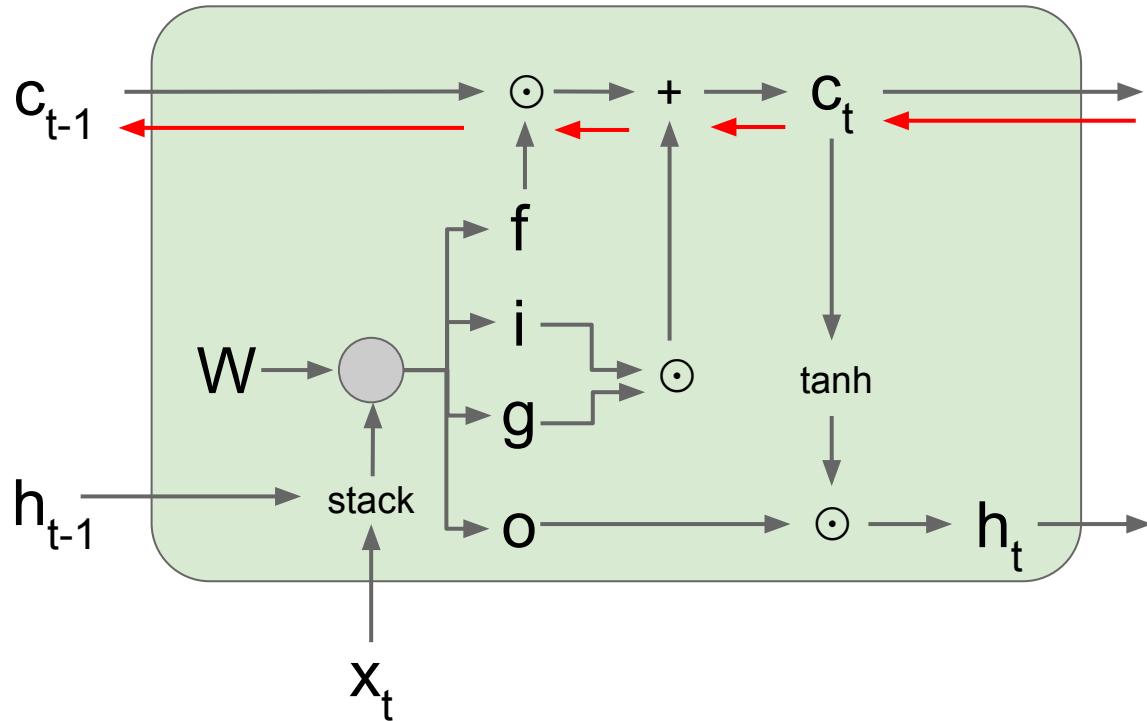
$$\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \sigma \\ \sigma \\ \sigma \\ \tanh \end{pmatrix} W \begin{pmatrix} h_{t-1} \\ x_t \end{pmatrix}$$

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

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

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

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

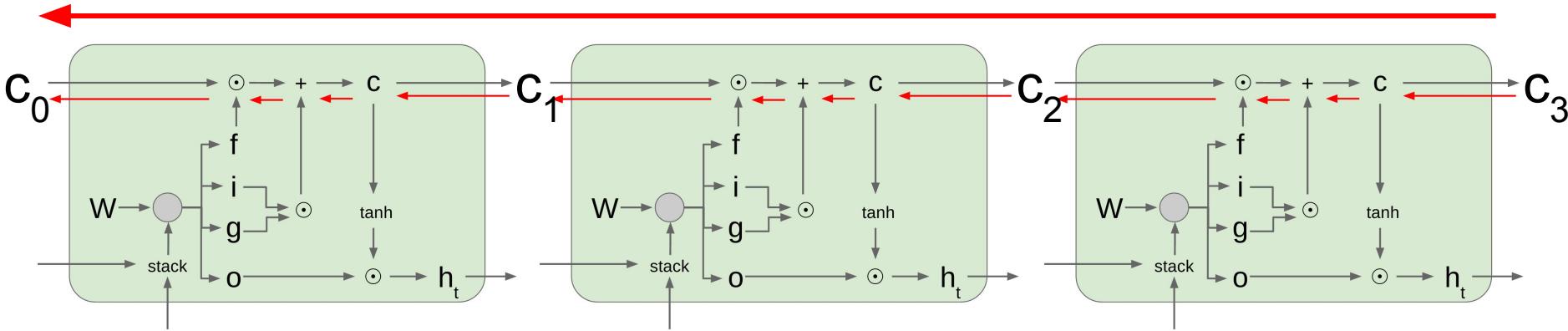
$$c_t = f \odot c_{t-1} + i \odot g$$

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

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

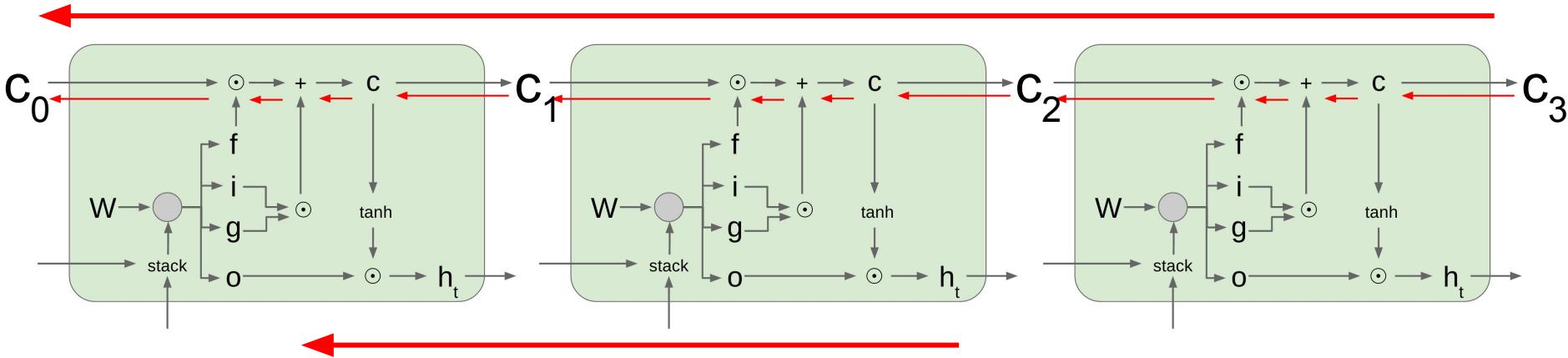
- e.g. **if the $f = 1$ and the $i = 0$** , then the information of that cell is preserved indefinitely.
- By contrast, it's harder for vanilla RNN to learn a recurrent weight matrix W_h that preserves info in hidden state

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

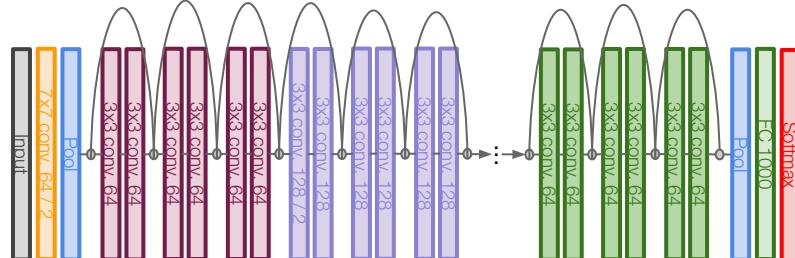
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



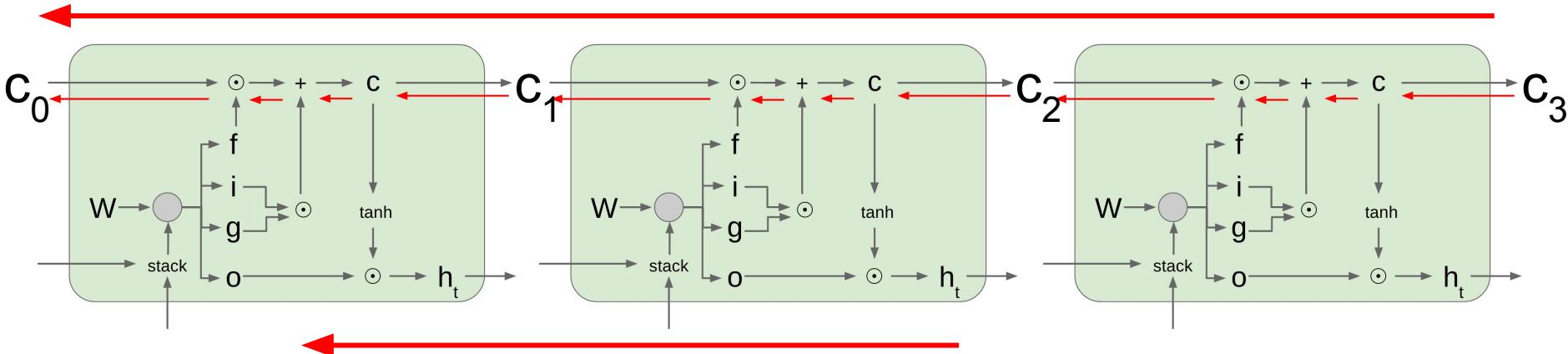
Similar to ResNet!



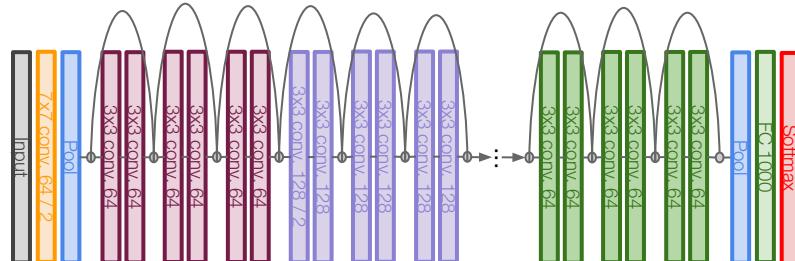
Long Short Term Memory (LSTM): Gradient Flow

[Hochreiter et al., 1997]

Uninterrupted gradient flow!



Similar to ResNet!



In between:
Highway Networks

$$g = T(x, W_T)$$

$$y = g \odot H(x, W_H) + (1 - g) \odot x$$

Srivastava et al., "Highway Networks",
ICML DL Workshop 2015

Other RNN Variants

GRU [*Learning phrase representations using rnn encoder-decoder for statistical machine translation*, Cho et al. 2014]

$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$

$$\tilde{h}_t = \tanh(W_{xh}x_t + W_{hh}(r_t \odot h_{t-1}) + b_h)$$

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

[*LSTM: A Search Space Odyssey*, Greff et al., 2015]

[*An Empirical Exploration of Recurrent Network Architectures*, Jozefowicz et al., 2015]

MUT1:

$$z = \text{sigm}(W_{xz}x_t + b_z)$$

$$r = \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r)$$

$$h_{t+1} = \tanh(W_{hh}(r \odot h_t) + \tanh(x_t) + b_h) \odot z + h_t \odot (1 - z)$$

MUT2:

$$z = \text{sigm}(W_{xz}x_t + W_{hx}h_t + b_z)$$

$$r = \text{sigm}(x_t + W_{hr}h_t + b_r)$$

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

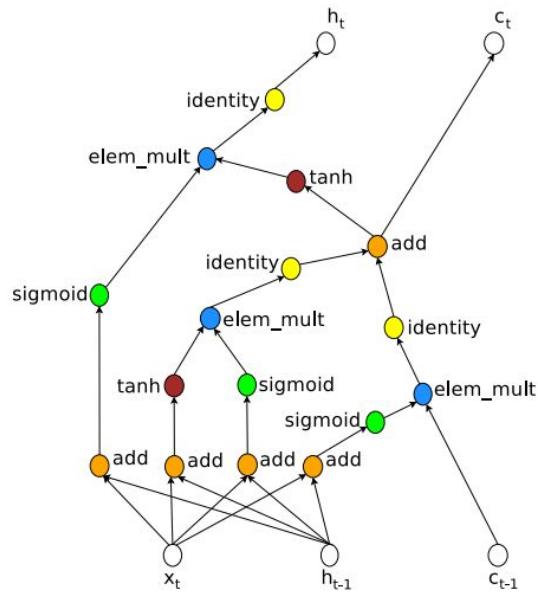
MUT3:

$$z = \text{sigm}(W_{xz}x_t + W_{hz}\tanh(h_t) + b_z)$$

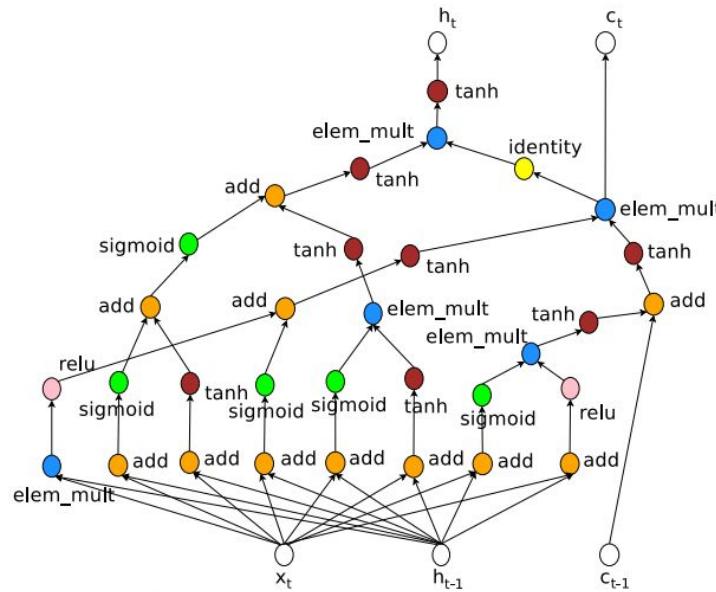
$$r = \text{sigm}(W_{xr}x_t + W_{hr}h_t + b_r)$$

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

Neural Architecture Search for RNN architectures



LSTM cell



Cell they found

Zoph et Le, "Neural Architecture Search with Reinforcement Learning", ICLR 2017
Figures copyright Zoph et al, 2017. Reproduced with permission.

Summary

- RNNs allow a lot of flexibility in architecture design
- Vanilla RNNs are simple but don't work very well
- Common to use LSTM or GRU: their additive interactions improve gradient flow
- Backward flow of gradients in RNN can explode or vanish. Exploding is controlled with gradient clipping. Vanishing is controlled with additive interactions (LSTM)
- Better/simpler architectures are a hot topic of current research, as well as new paradigms for reasoning over sequences
- Better understanding (both theoretical and empirical) is needed.

Next time: Attention and Transformers

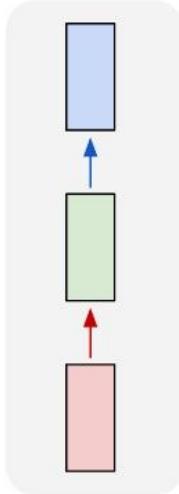
Lecture 11: Attention and Transformers

Administrative

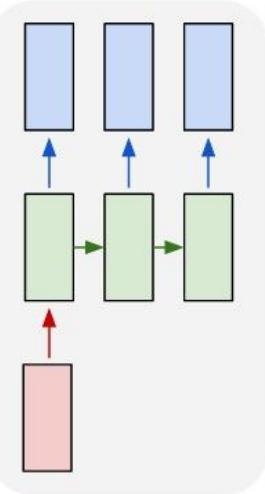
- Project proposal grades released.
Check feedback on GradeScope!
- Project milestone due May 7th Saturday 11:59pm PT
Check Ed and course website for requirements

Last Time: Recurrent Neural Networks

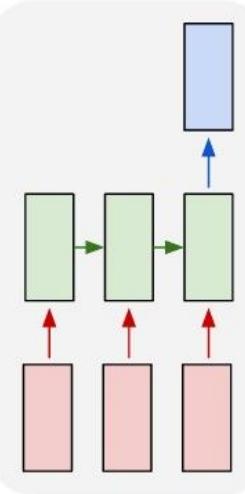
one to one



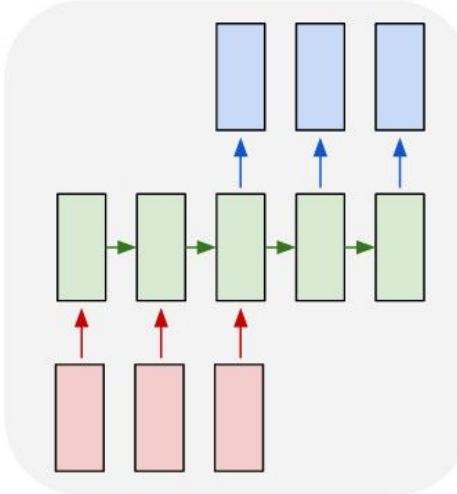
one to many



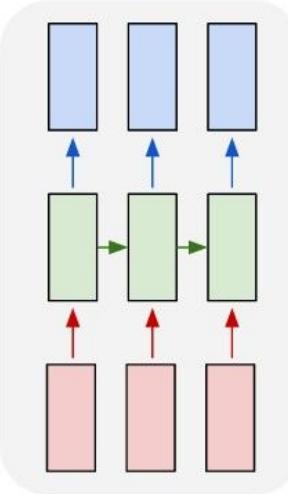
many to one



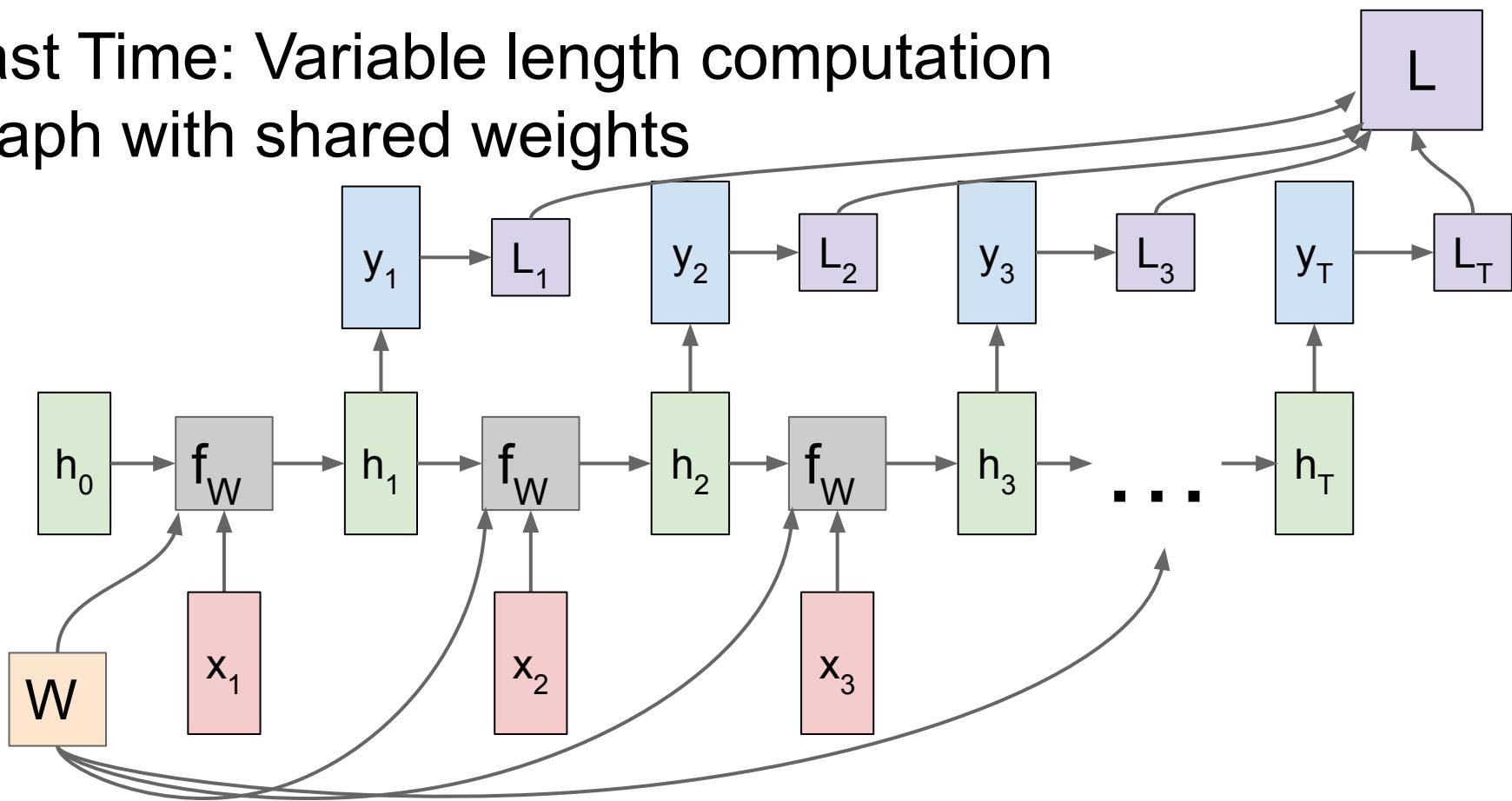
many to many



many to many



Last Time: Variable length computation graph with shared weights

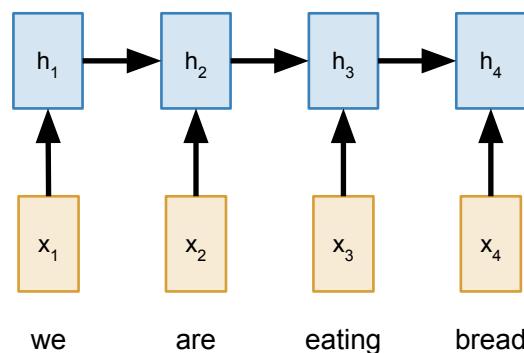


Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

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



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

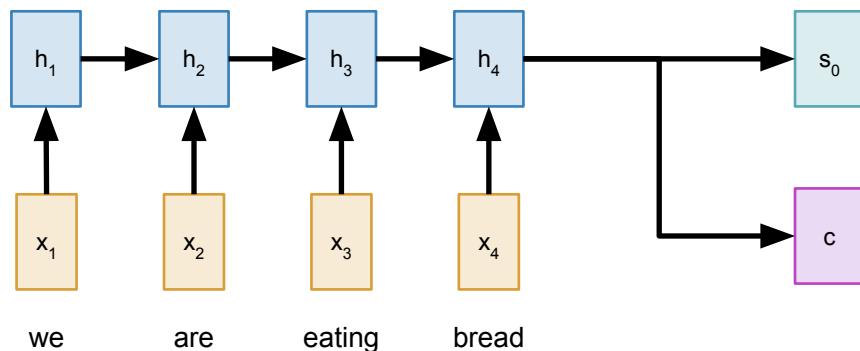
Output: Sequence y_1, \dots, y_T

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al, "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

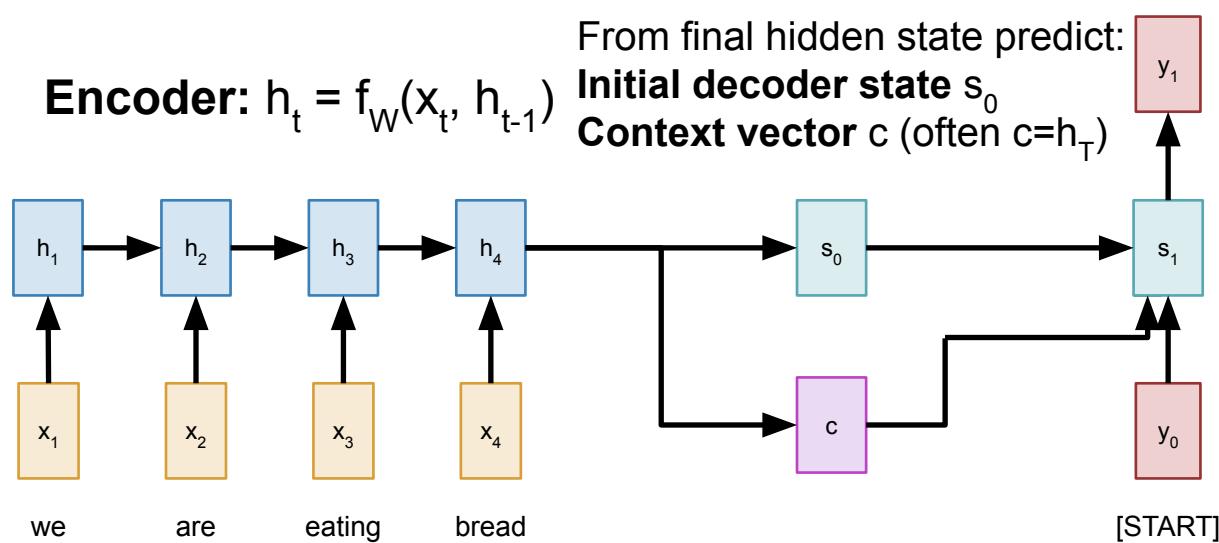
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sutskever et al., "Sequence to sequence learning with neural networks", NeurIPS 2014

Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

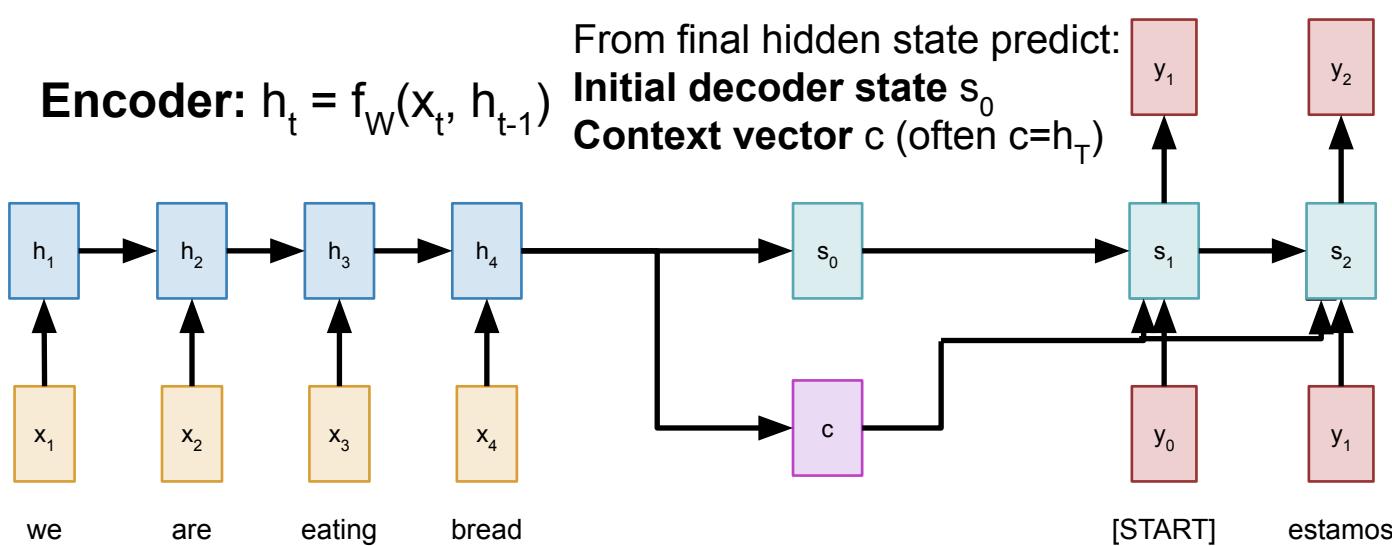
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

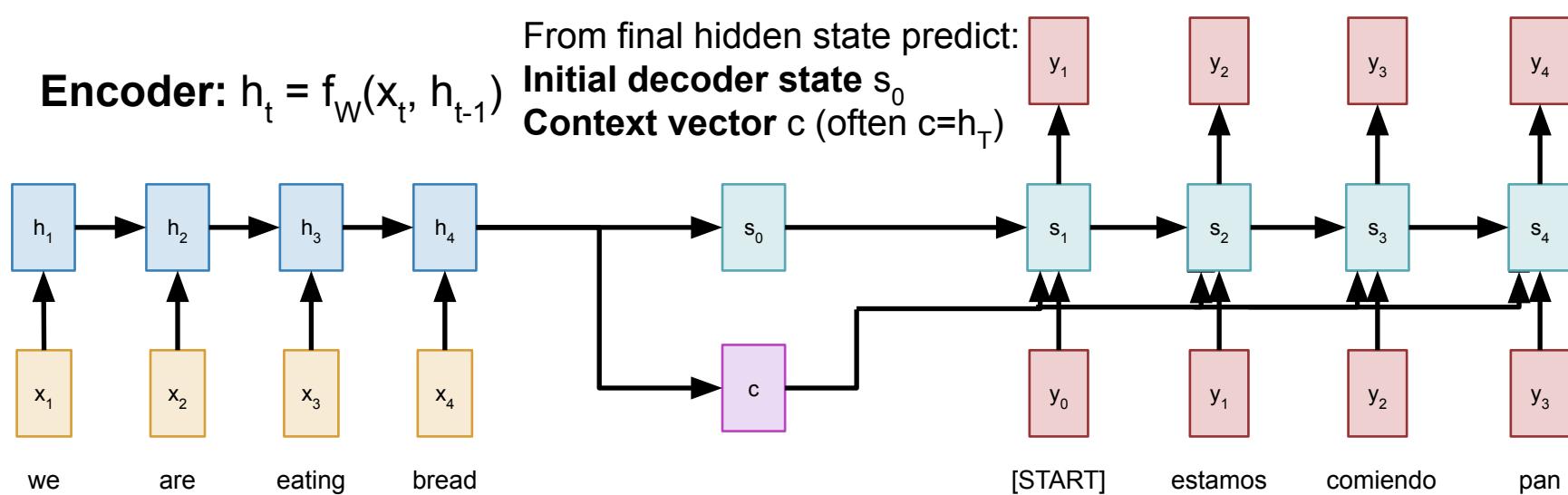
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

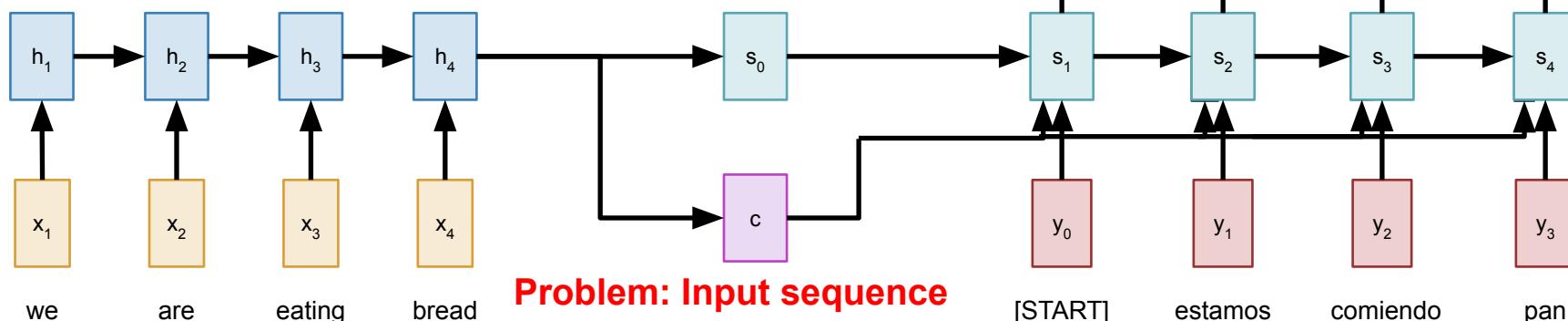
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



**Problem: Input sequence
bottlenecked through
fixed-sized vector. What if
 $T=1000$?**

Sutskever et al, "Sequence to sequence learning with neural networks", NIPS 2014

Sequence to Sequence with RNNs

Input: Sequence x_1, \dots, x_T

Output: Sequence y_1, \dots, y_T

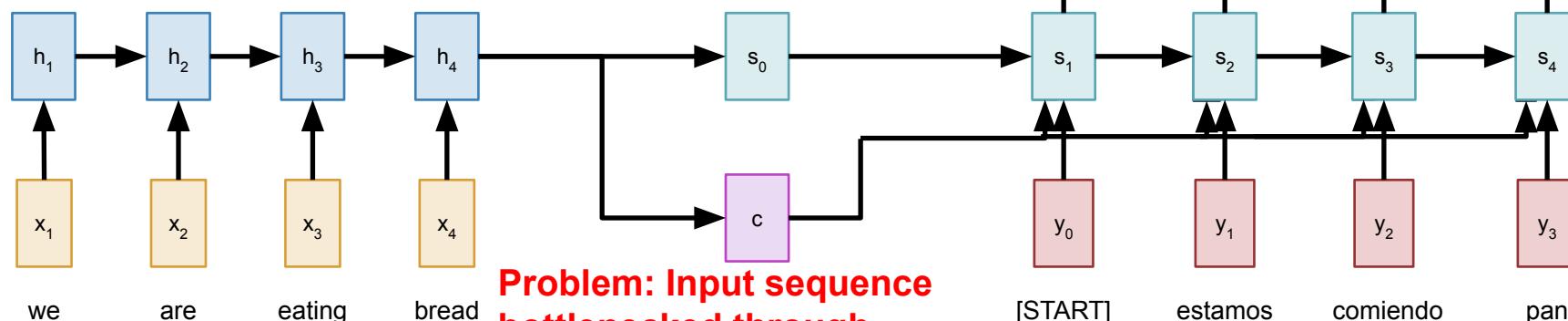
Decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c)$

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

From final hidden state predict:

Initial decoder state s_0

Context vector c (often $c=h_T$)



Problem: Input sequence bottlenecked through fixed-sized vector. What if $T=1000$?

estamos comiendo pan [STOP]

[START] estamos comiendo pan

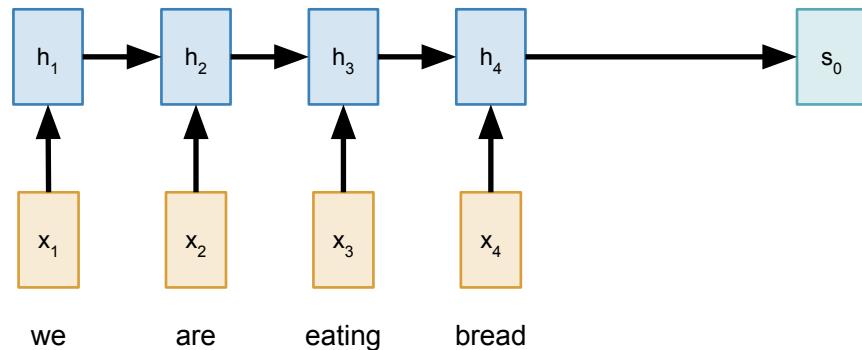
Idea: use new context vector at each step of decoder!

Sequence to Sequence with RNNs and Attention

Input: Sequence x_1, \dots, x_T

Output: Sequence $y_1, \dots, y_{T'}$

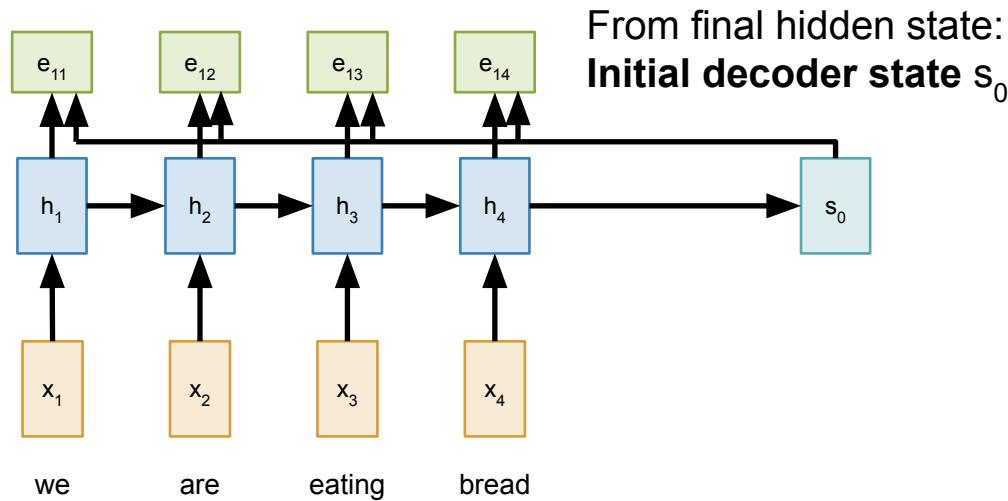
Encoder: $h_t = f_W(x_t, h_{t-1})$ From final hidden state:
Initial decoder state s_0



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

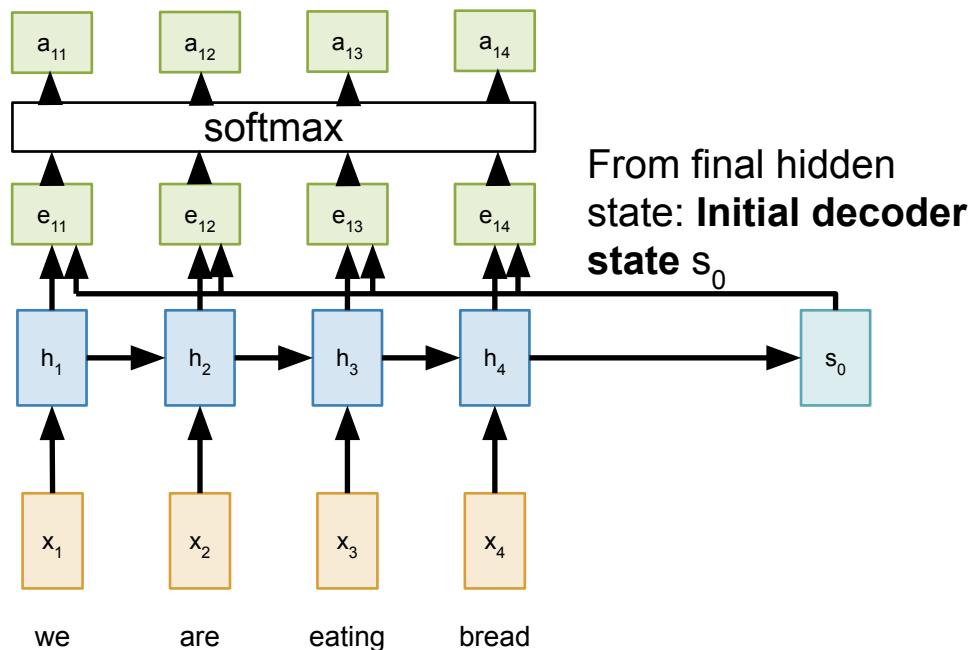
Sequence to Sequence with RNNs and Attention

Compute (scalar) **alignment scores**
 $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$ (f_{att} is an MLP)



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention



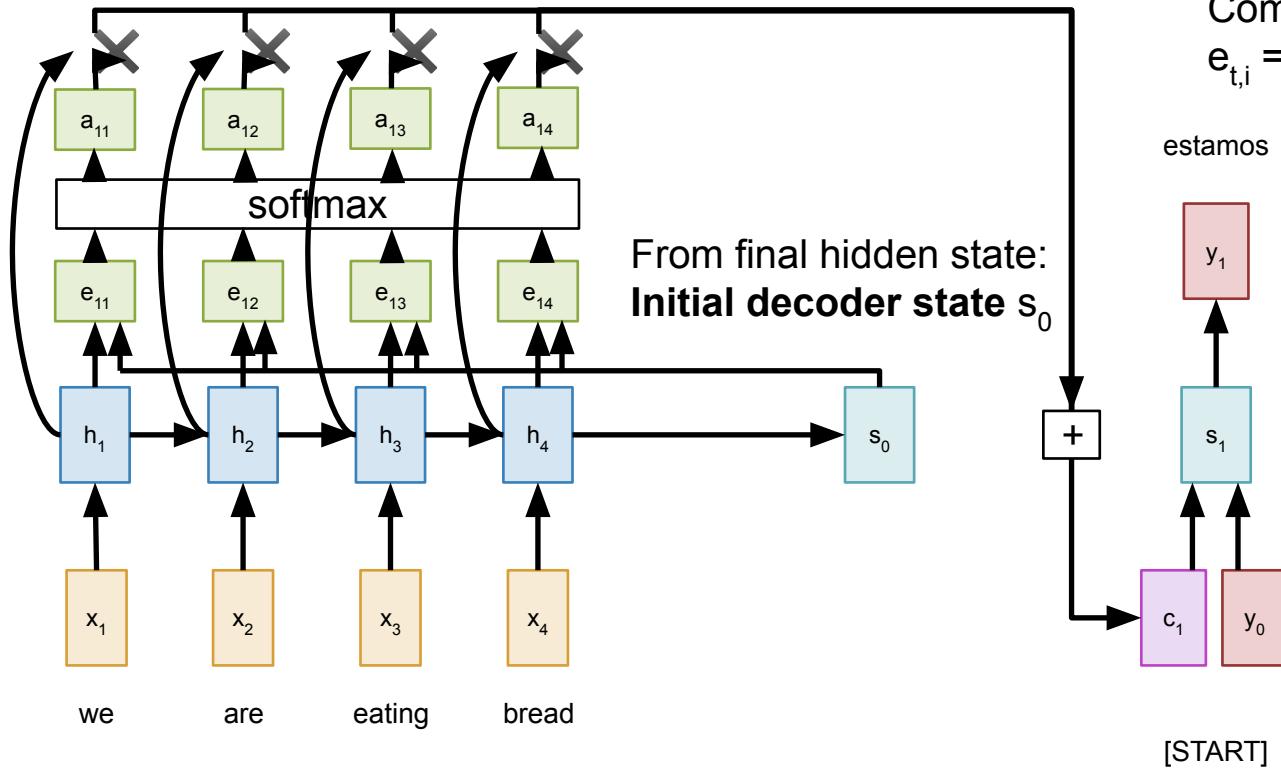
Compute (scalar) **alignment scores**
 $e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

Normalize alignment scores
to get **attention weights**
 $0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$

From final hidden
state: **Initial decoder**
state s_0

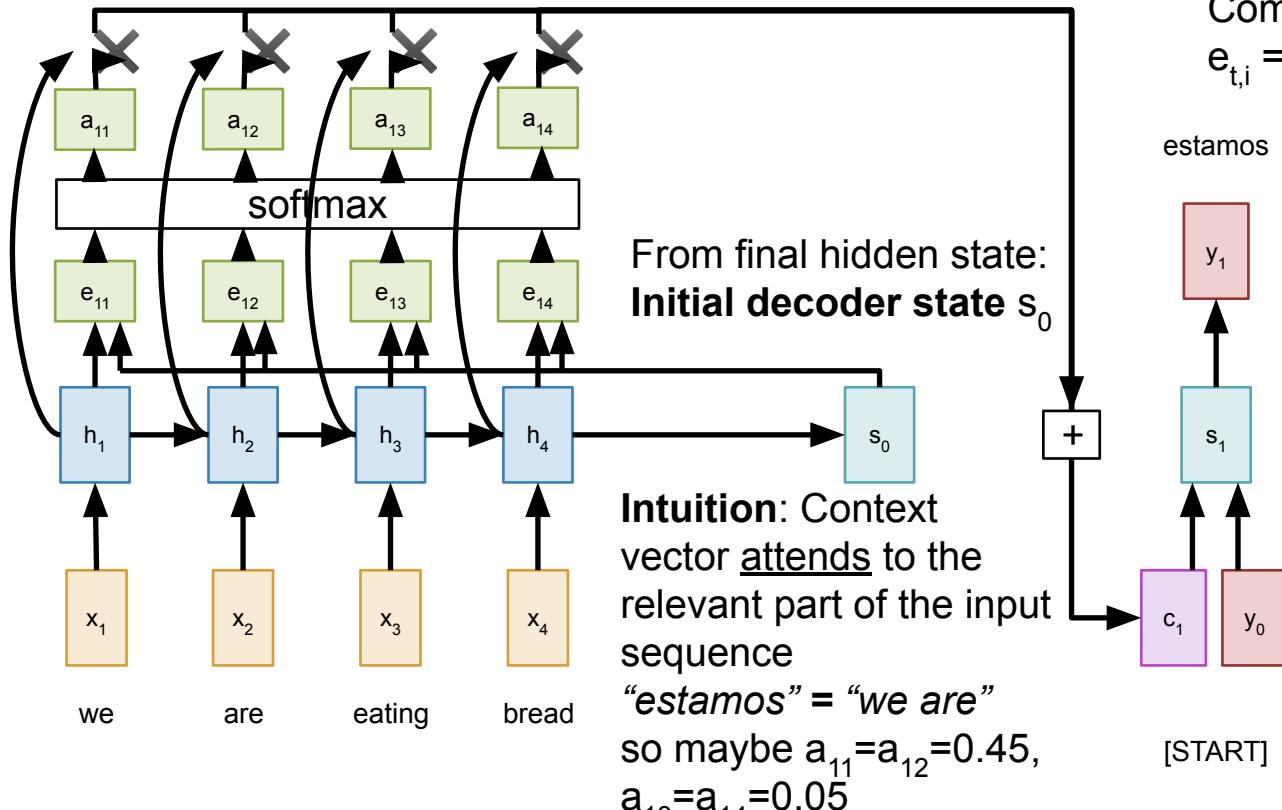
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention



Compute (scalar) **alignment scores**
 $e_{t,i} = f_{\text{att}}(s_{t-1}, h_i)$ (f_{att} is an MLP)

estamos

Normalize alignment scores
to get **attention weights**

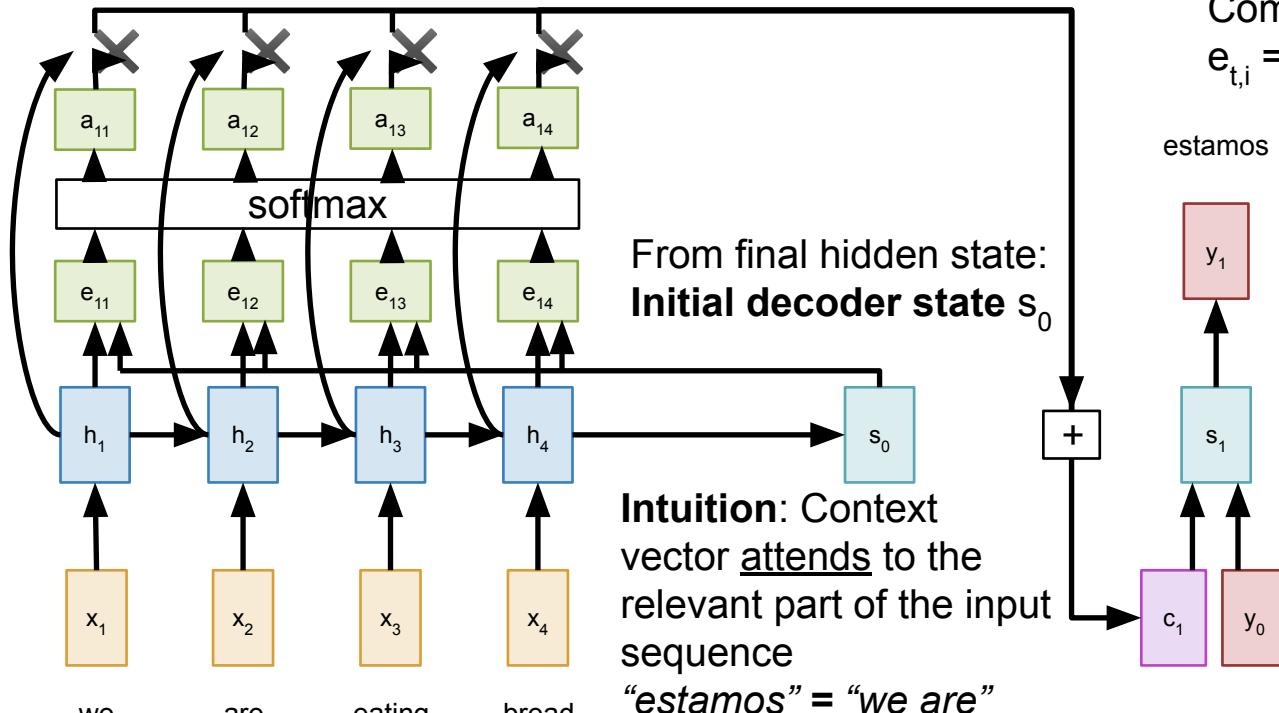
$$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$$

Compute context vector as
linear combination of hidden
states

$$c_t = \sum_i a_{t,i} h_i$$

Use context vector in
decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

Sequence to Sequence with RNNs and Attention



Compute (scalar) **alignment scores**
 $e_{t,i} = f_{att}(s_{t-1}, h_i)$ (f_{att} is an MLP)

estamos

Normalize alignment scores
to get **attention weights**

$$0 < a_{t,i} < 1 \quad \sum_i a_{t,i} = 1$$

Compute context vector as
linear combination of hidden
states

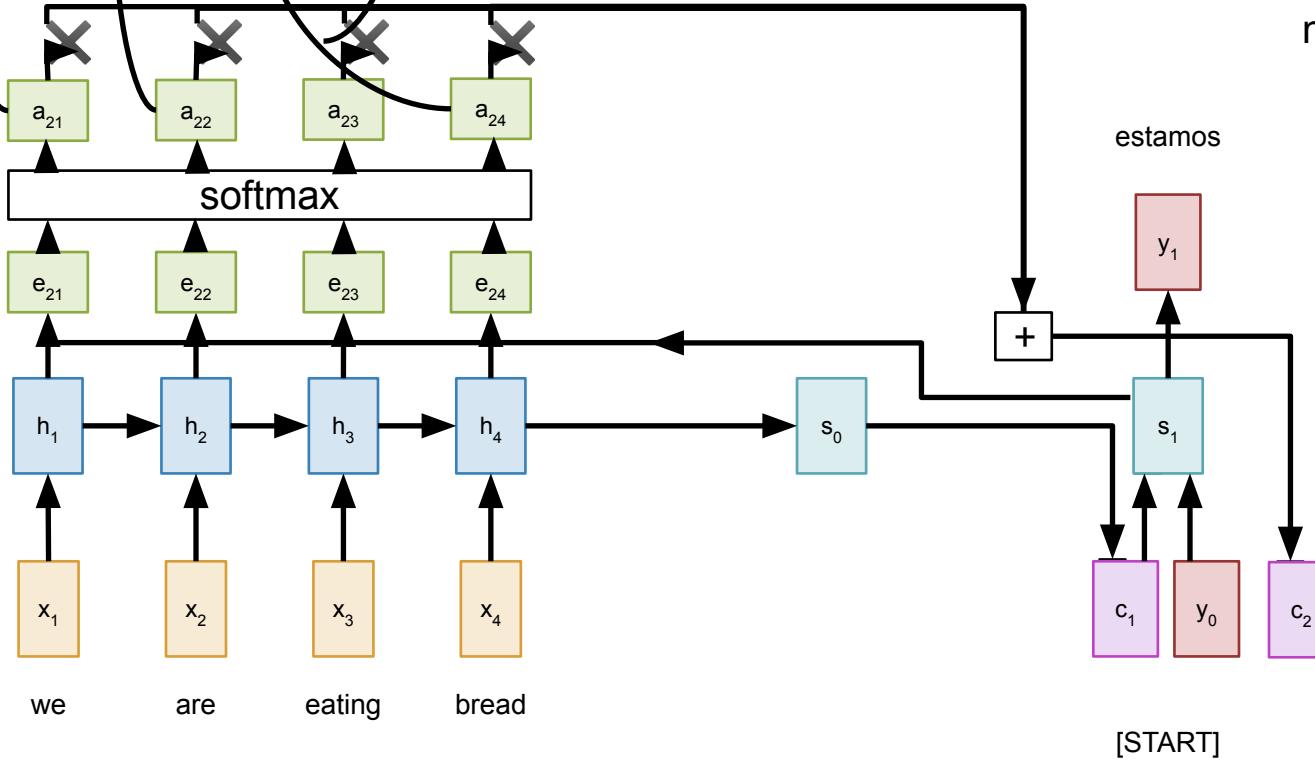
$$c_t = \sum_i a_{t,i} h_i$$

Use context vector in
decoder: $s_t = g_U(y_{t-1}, s_{t-1}, c_t)$

**This is all differentiable! No
supervision on attention
weights – backprop through
everything**

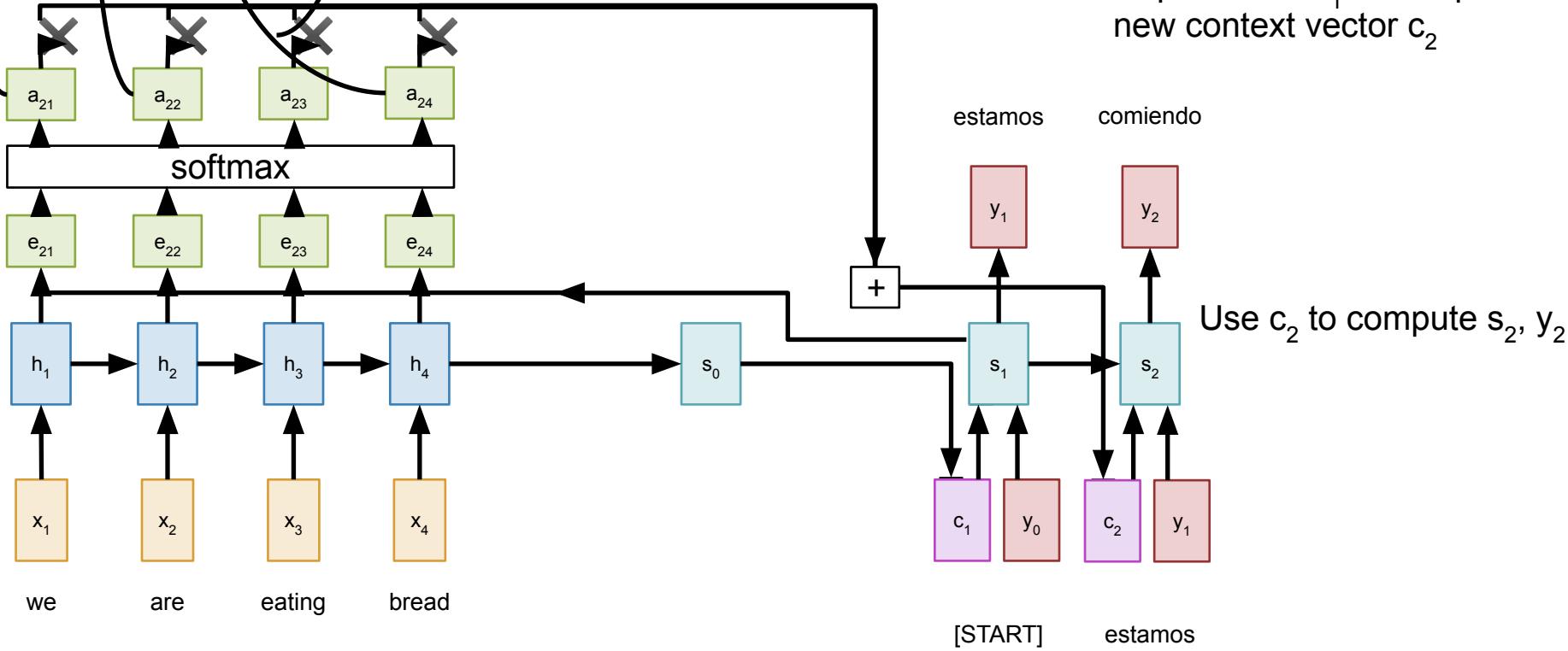
Sequence to Sequence with RNNs and Attention

Repeat: Use s_1 to compute new context vector c_2



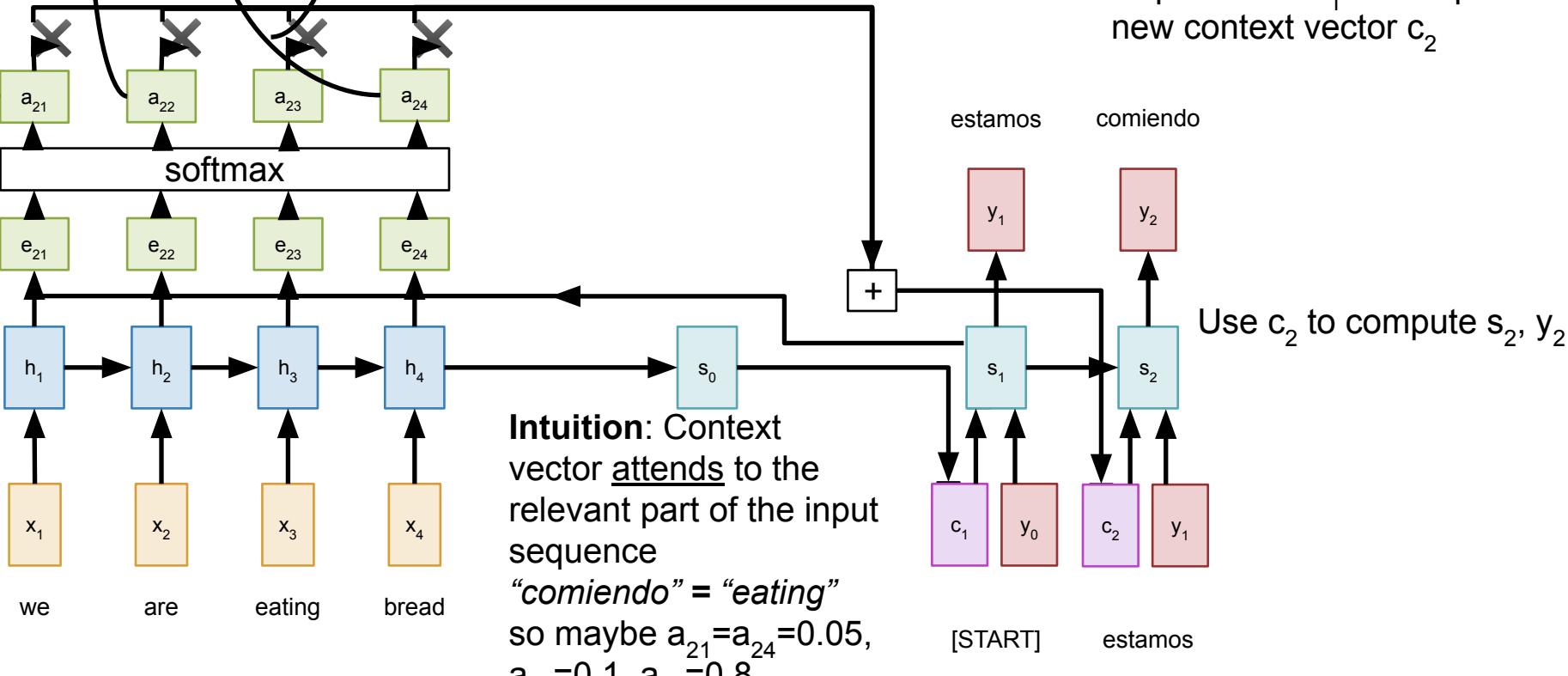
Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention

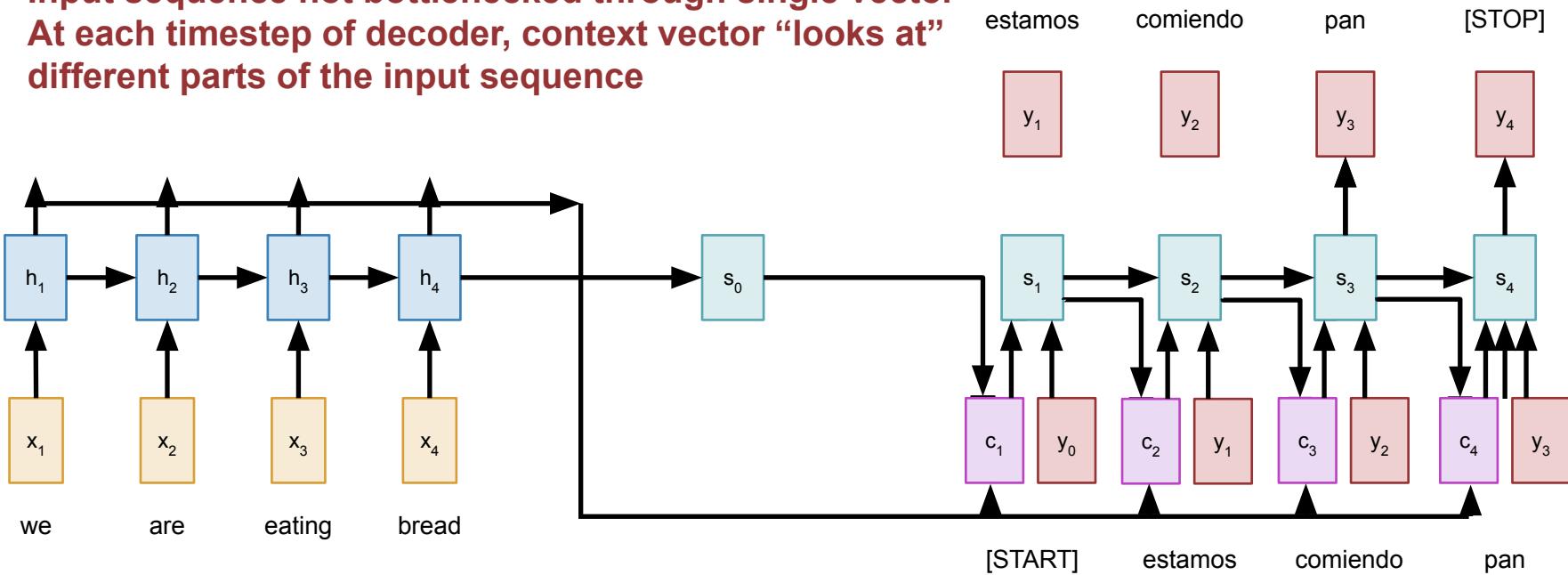


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Sequence to Sequence with RNNs and Attention

Use a different context vector in each timestep of decoder

- Input sequence not bottlenecked through single vector
- At each timestep of decoder, context vector “looks at” different parts of the input sequence



Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

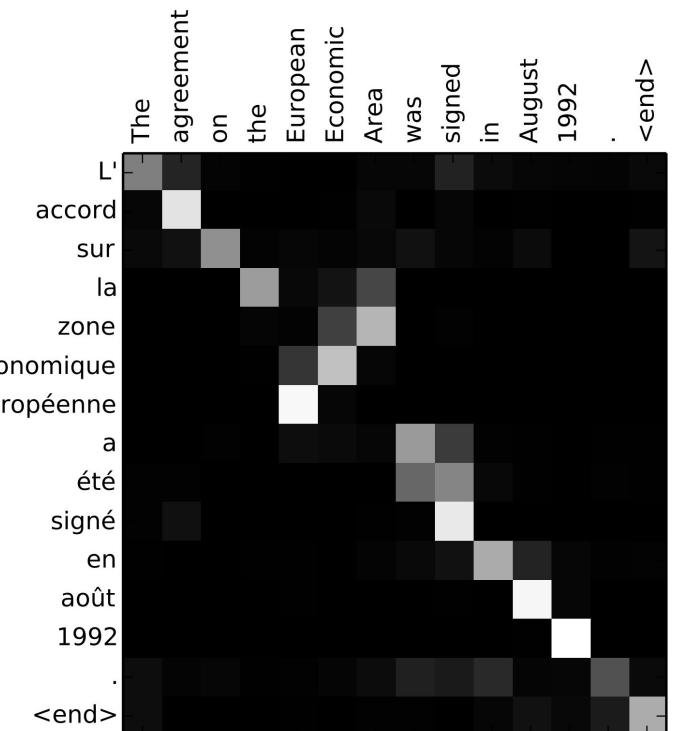
Sequence to Sequence with RNNs and Attention

Example: English to French translation

Input: “The agreement on the European Economic Area was signed in August 1992.”

Output: “L'accord sur la zone économique européenne a été signé en août 1992.”

Visualize attention weights $a_{t,i}$



Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence to Sequence with RNNs and Attention

Example: English to French translation

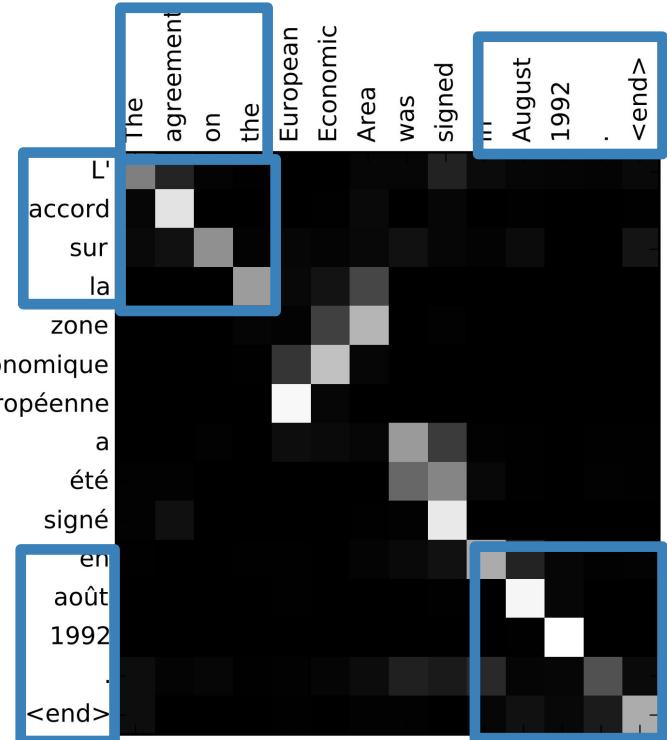
Input: “**The agreement on the European Economic Area was signed in August 1992.**”

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

Diagonal attention means words correspond in order

Diagonal attention means words correspond in order

Visualize attention weights $a_{t,i}$



Sequence to Sequence with RNNs and Attention

Example: English to French translation

Input: “**The agreement on the European Economic Area** was signed in **August 1992**.”

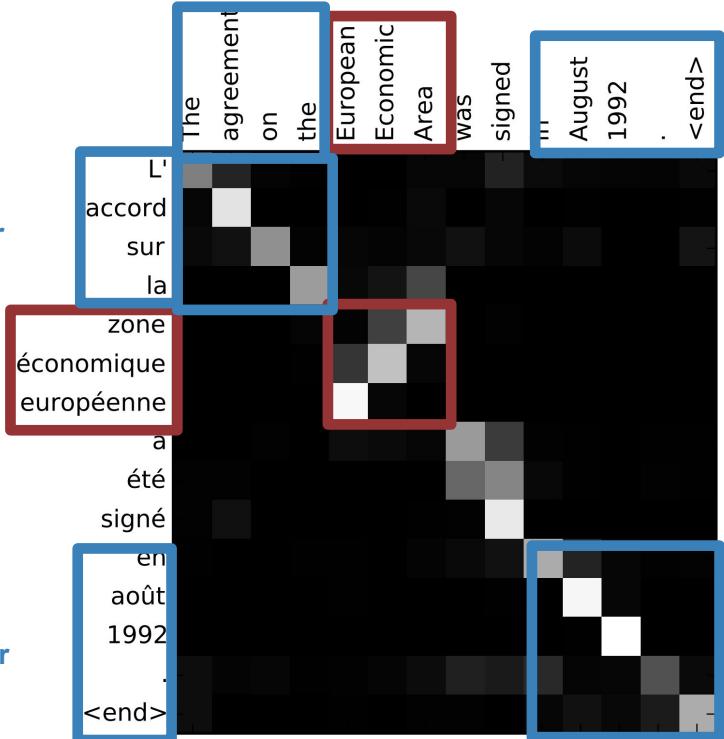
Output: “**L'accord sur la zone économique européenne** a été signé en **août 1992**.”

Diagonal attention means words correspond in order

Attention figures out different word orders

Diagonal attention means words correspond in order

Visualize attention weights $a_{t,i}$

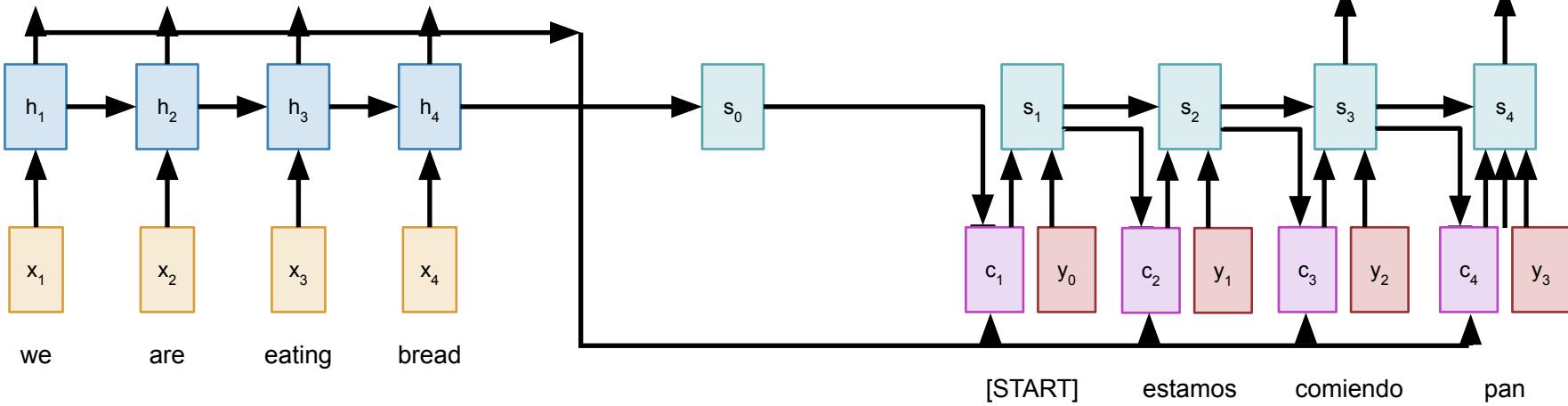


Bahdanau et al, “Neural machine translation by jointly learning to align and translate”, ICLR 2015

Sequence to Sequence with RNNs and Attention

The decoder doesn't use the fact that h_i form an ordered sequence – it just treats them as an unordered set $\{h_i\}$

Can use similar architecture given any set of input hidden vectors $\{h_i\}$!

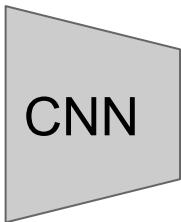
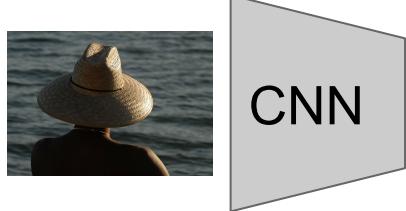


Bahdanau et al, "Neural machine translation by jointly learning to align and translate", ICLR 2015

Image Captioning using spatial features

Input: Image I

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



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

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

Image Captioning using spatial features

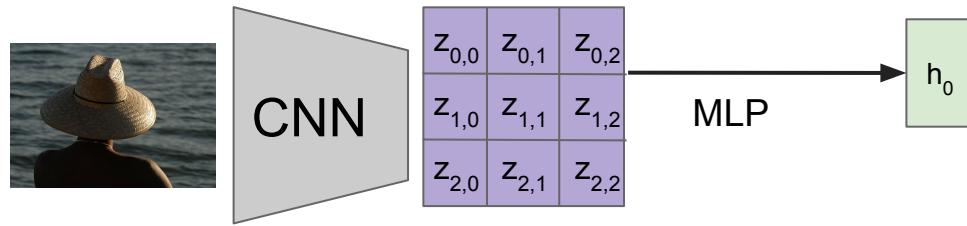
Input: Image I

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

Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{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

Image Captioning using spatial features

Input: Image I

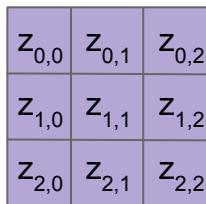
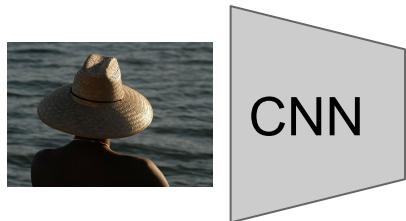
Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

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

Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

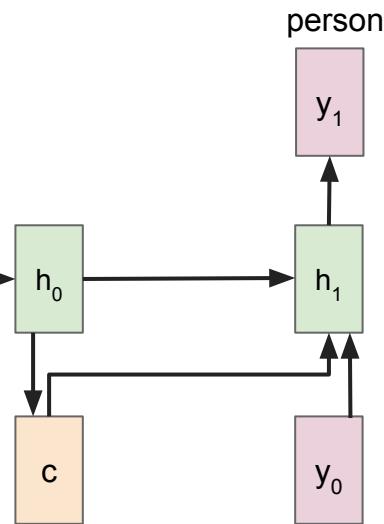
$f_w(\cdot)$ is an MLP



Extract spatial
features from a
pretrained CNN

Features:
 $H \times W \times D$

MLP



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

[START]

Image Captioning using spatial features

Input: Image I

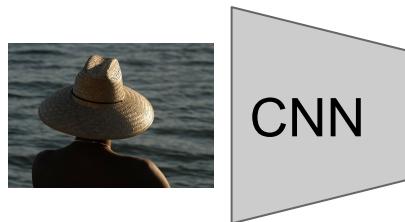
Output: Sequence $\mathbf{y} = y_1, y_2, \dots, y_T$

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

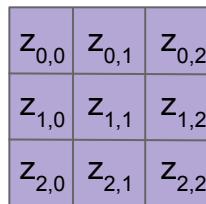
Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP

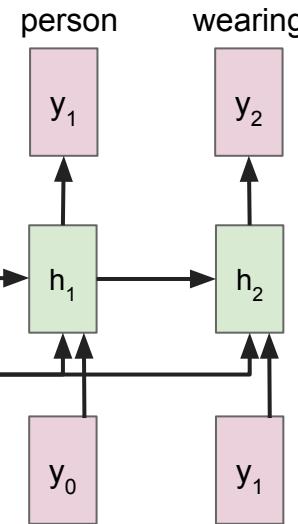


Extract spatial
features from a
pretrained CNN



Features:
 $H \times W \times D$

MLP



[START]

person

Image Captioning using spatial features

Input: Image I

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

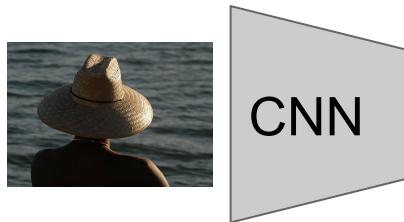
Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$

where context vector c is often $c = h_0$

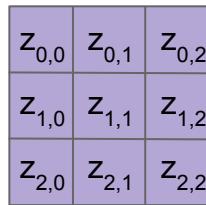
Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$f_w(\cdot)$ is an MLP



Extract spatial
features from a
pretrained CNN

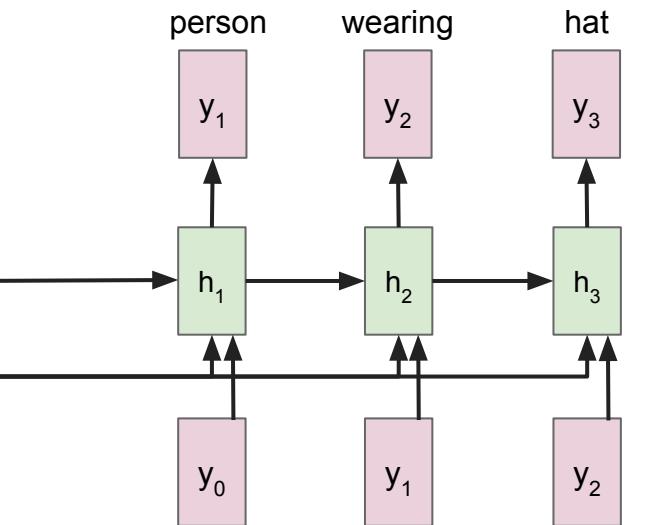


Features:
 $H \times W \times D$

MLP



c



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

[START]

person

wearing

Image Captioning using spatial features

Input: Image I

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

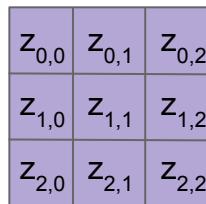
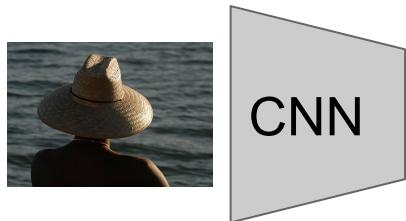
Decoder: $y_t = g_v(y_{t-1}, h_{t-1}, c)$

where context vector c is often $c = h_0$

Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

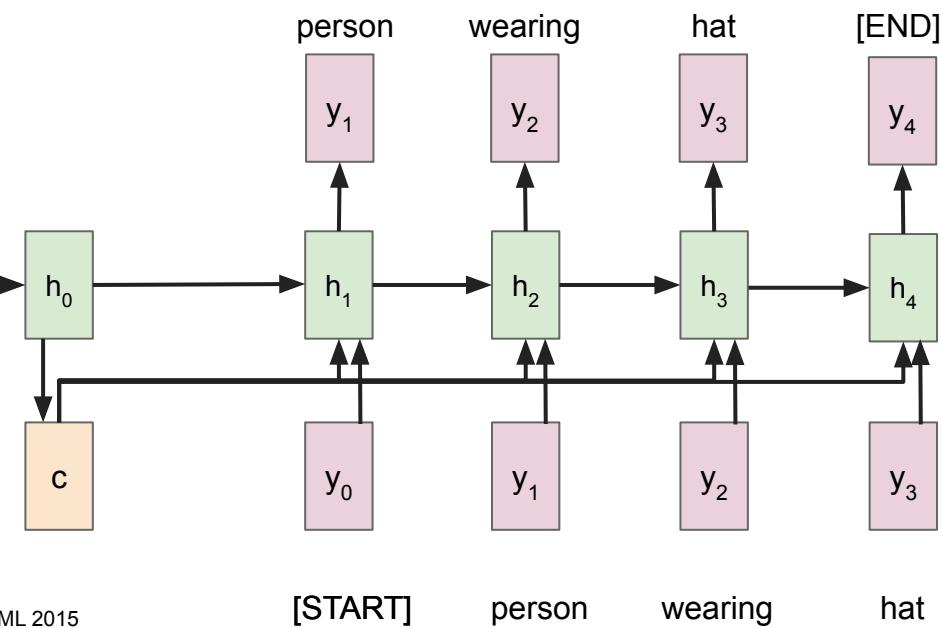
$f_w(\cdot)$ is an MLP



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

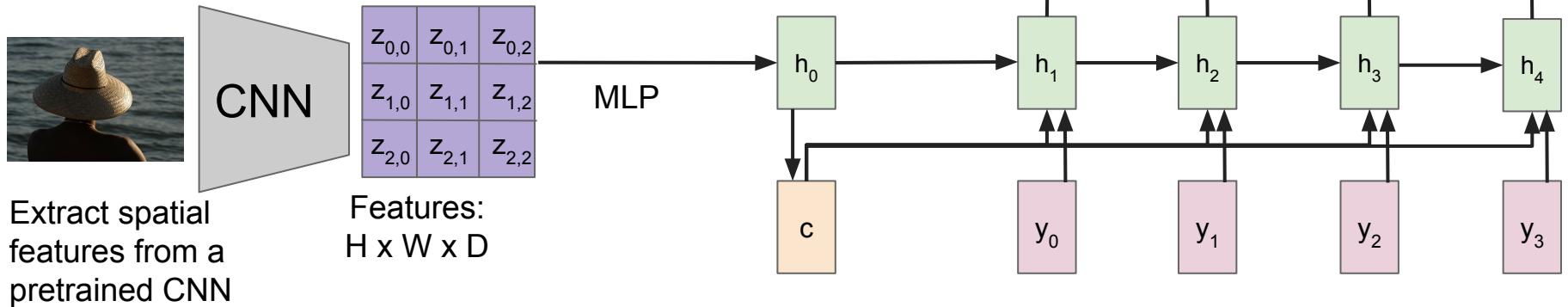
[START] person wearing hat

Image Captioning using spatial features

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



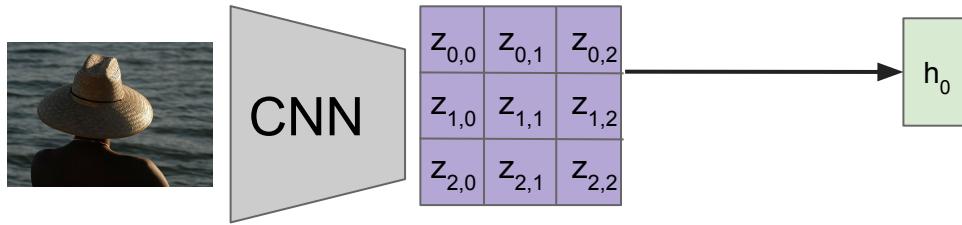
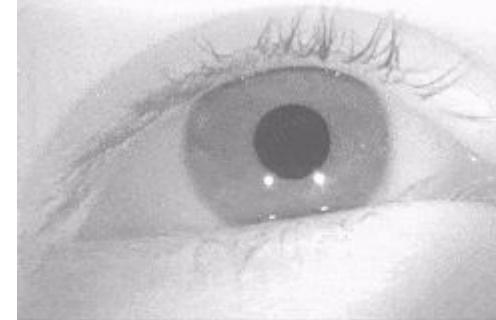
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and **Attention**

[gif source](#)

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

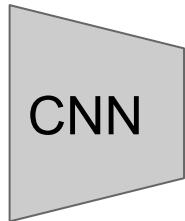
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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

Features:
 $H \times W \times D$

Alignment scores:

$H \times W$

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

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

h_0

h_0

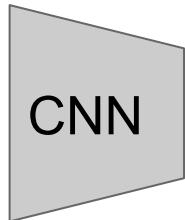
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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:

$H \times W$

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

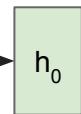
Normalize to get attention weights:

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

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

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

Features:
 $H \times W \times D$



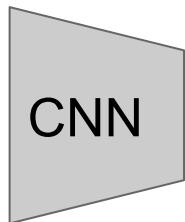
Xu et al, "Show, Attend and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015

Image Captioning with RNNs and Attention

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:

$$H \times W$$

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

Normalize to get attention weights:

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

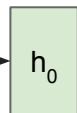
$0 < a_{t,i,j} < 1$,
attention values sum to 1

Compute context vector:

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

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

Features:
 $H \times W \times D$



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

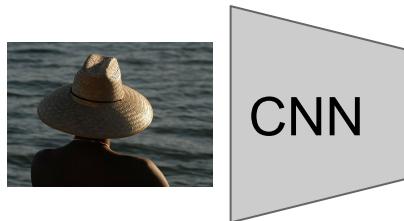
Image Captioning with RNNs and Attention

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

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

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

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

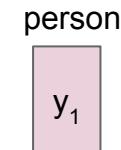


Extract spatial features from a pretrained CNN

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

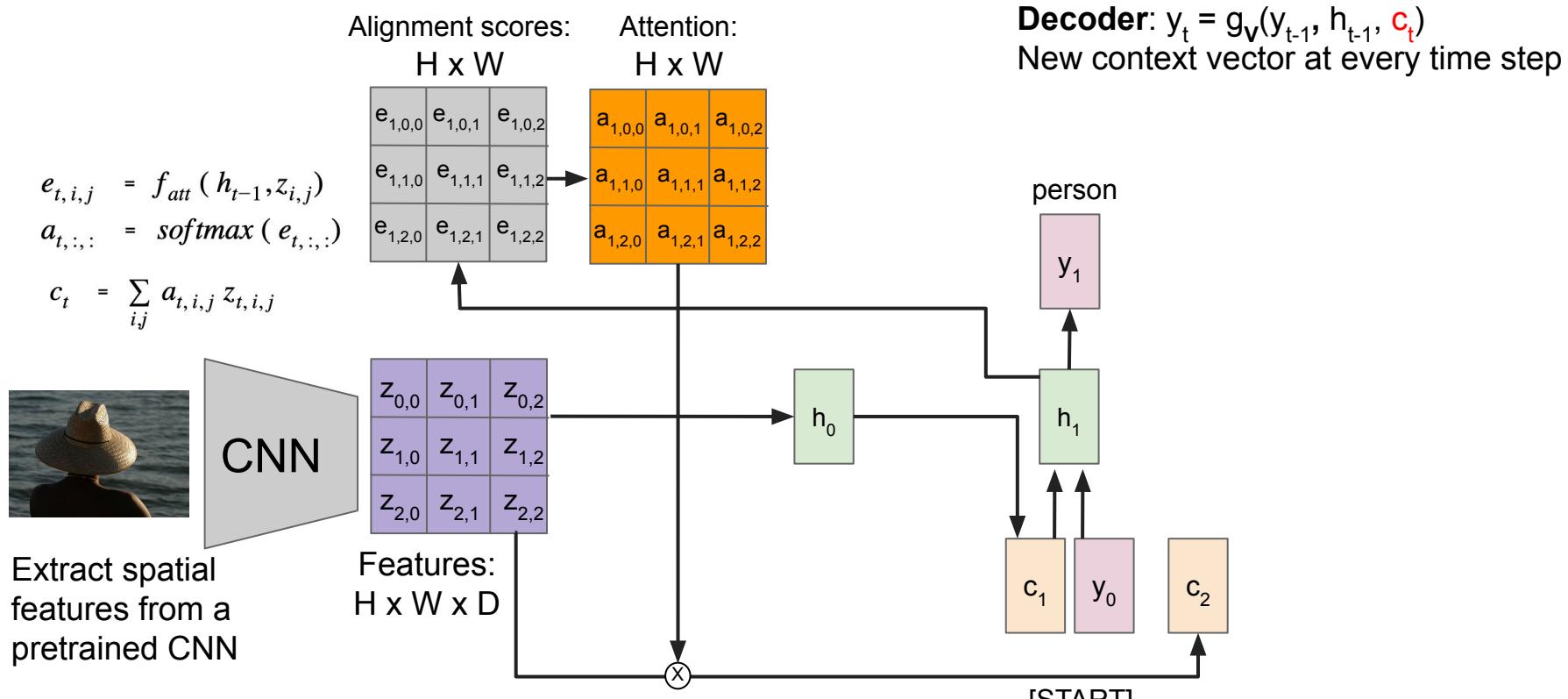
Features:
 $H \times W \times D$

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

Image Captioning with RNNs and Attention



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

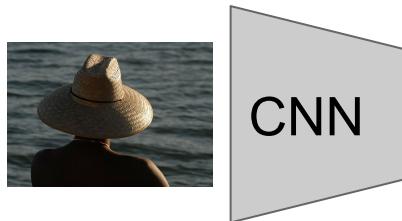
Image Captioning with RNNs and Attention

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

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

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

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

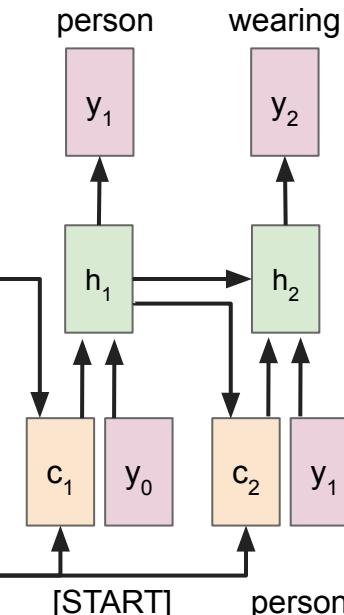


Extract spatial features from a pretrained CNN

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

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

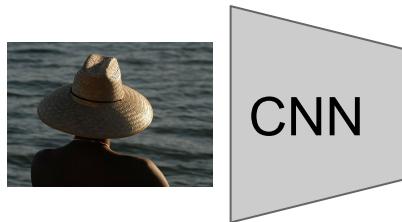
Image Captioning with RNNs and Attention

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

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

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

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



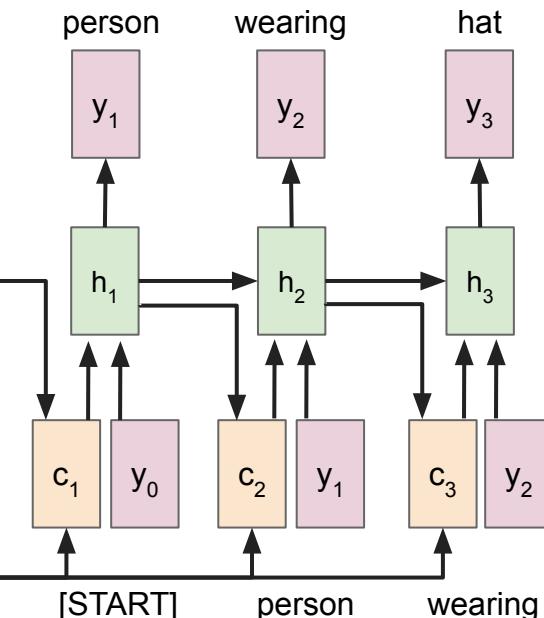
Extract spatial features from a pretrained CNN

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

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

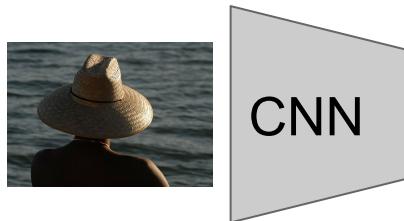
Image Captioning with RNNs and Attention

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

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

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

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



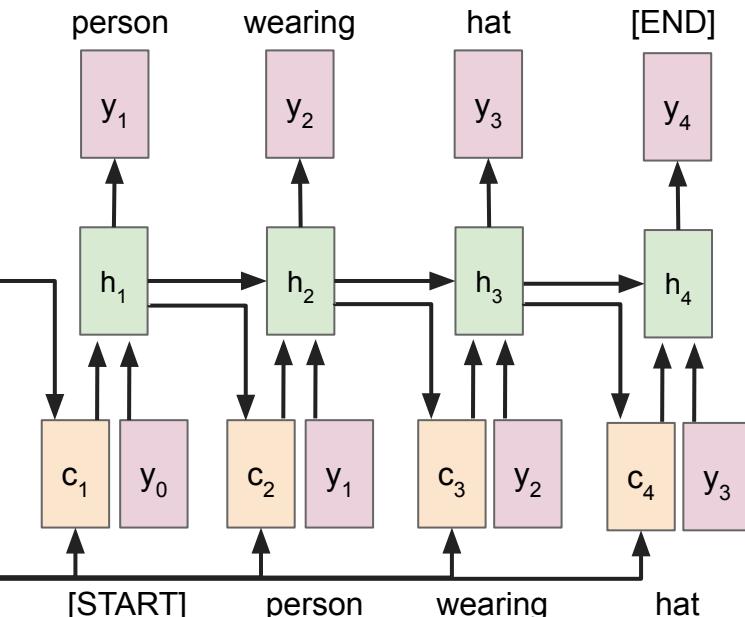
Extract spatial features from a pretrained CNN

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

Features:
 $H \times W \times D$

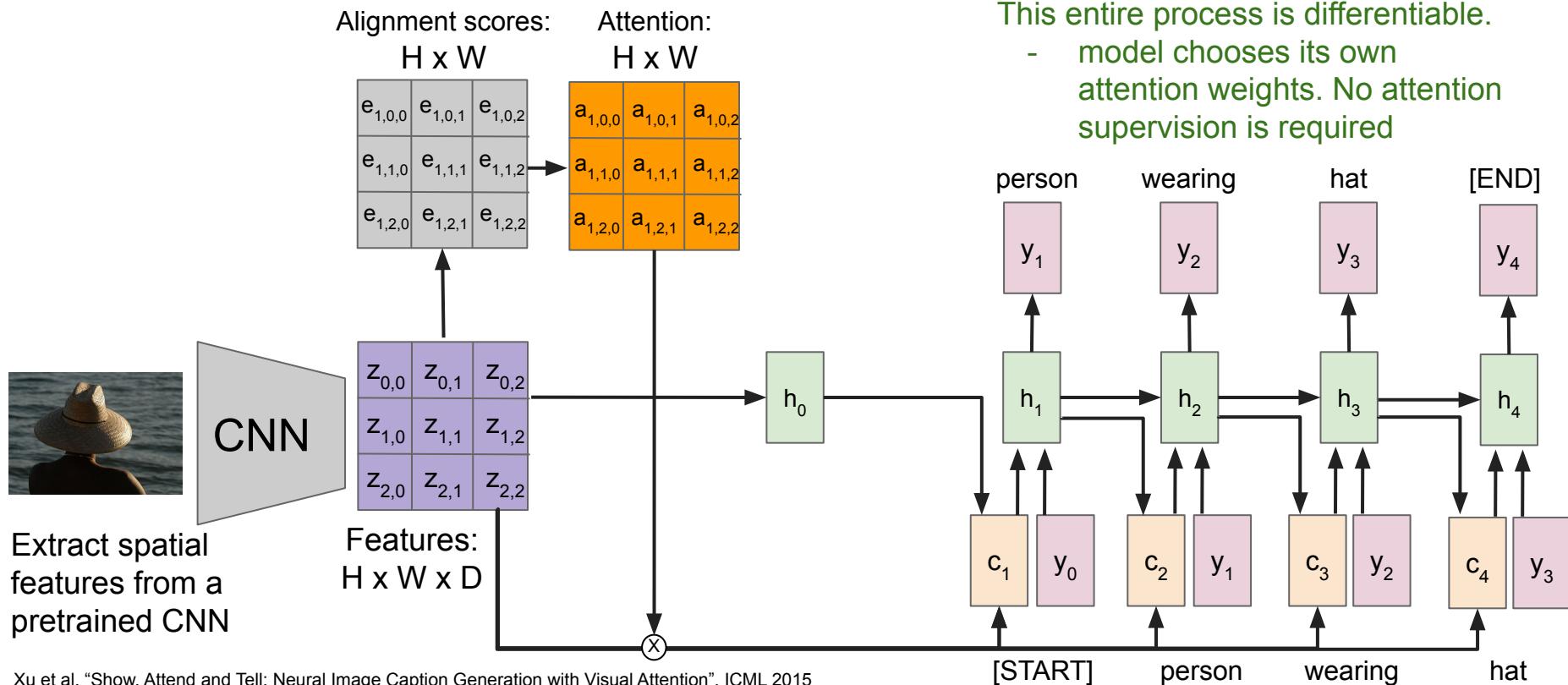
$$\text{Decoder: } y_t = g_v(y_{t-1}, h_{t-1}, c_t)$$

New context vector at every time step



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

Image Captioning with RNNs and Attention



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

Image Captioning with Attention

Soft attention



Hard attention
(requires
reinforcement
learning)



A

bird

flying

over

a

body

of

water

.

Xu et al, "Show, Attend, and Tell: Neural Image Caption Generation with Visual Attention", ICML 2015
Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio, 2015. Reproduced with permission.

Image Captioning with Attention



A woman is throwing a frisbee in a park.



A dog is standing on a hardwood floor.



A stop sign is on a road with a mountain in the background.



A little girl sitting on a bed with a teddy bear.



A group of people sitting on a boat in the water.

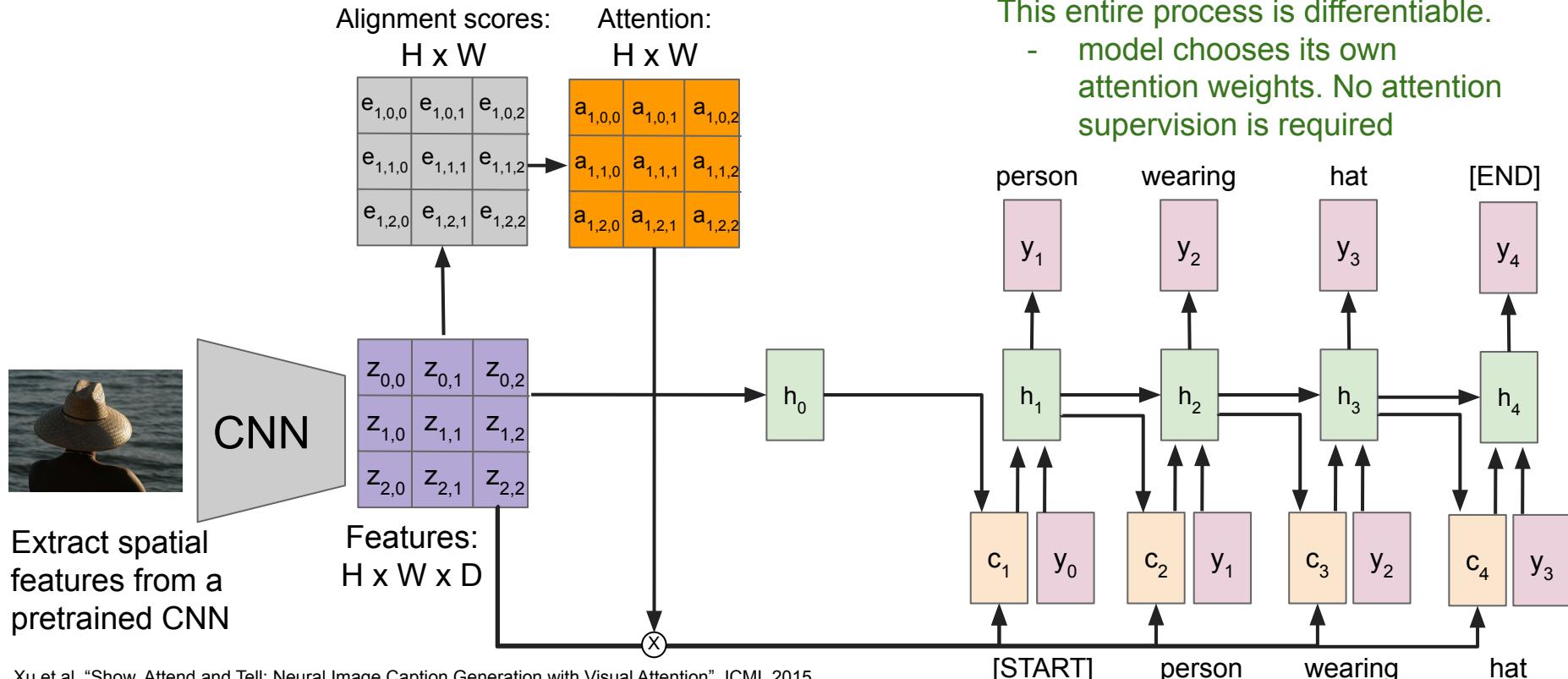


A giraffe standing in a forest with trees in the background.

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

Figure copyright Kelvin Xu, Jimmy Lei Ba, Jamie Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhutdinov, Richard S. Zemel, and Yoshua Bengio, 2015. Reproduced with permission.

Image Captioning with RNNs and Attention



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

Attention we just saw in image captioning

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

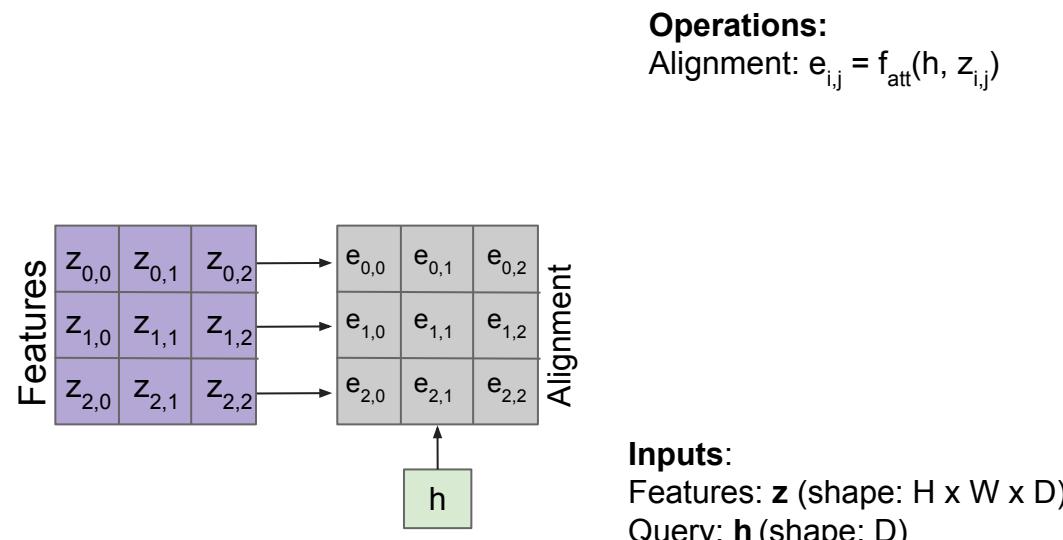
h

Inputs:

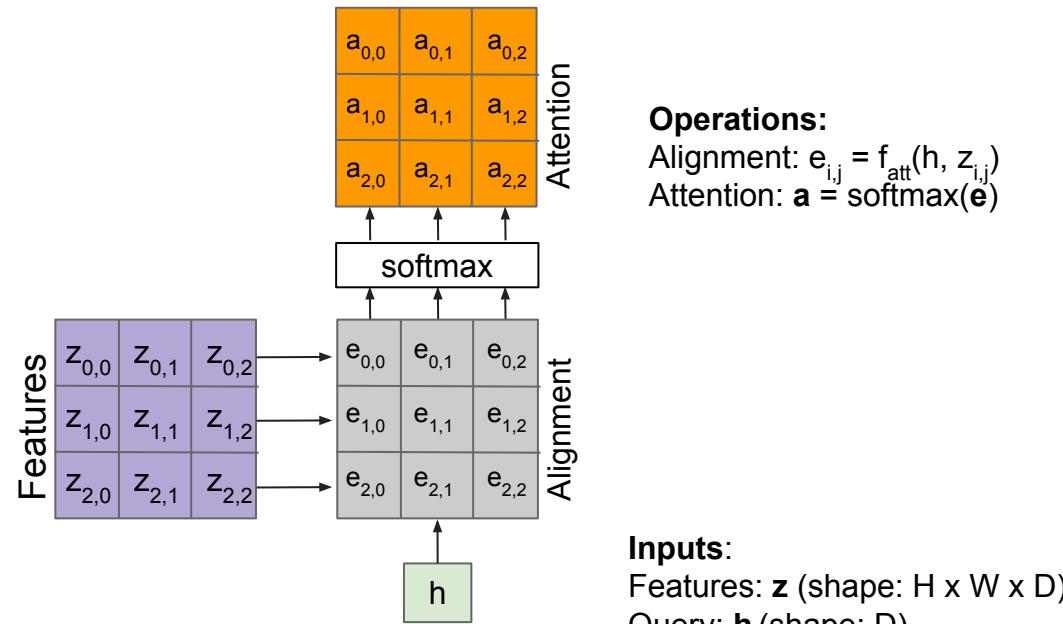
Features: \mathbf{z} (shape: $H \times W \times D$)

Query: \mathbf{h} (shape: D)

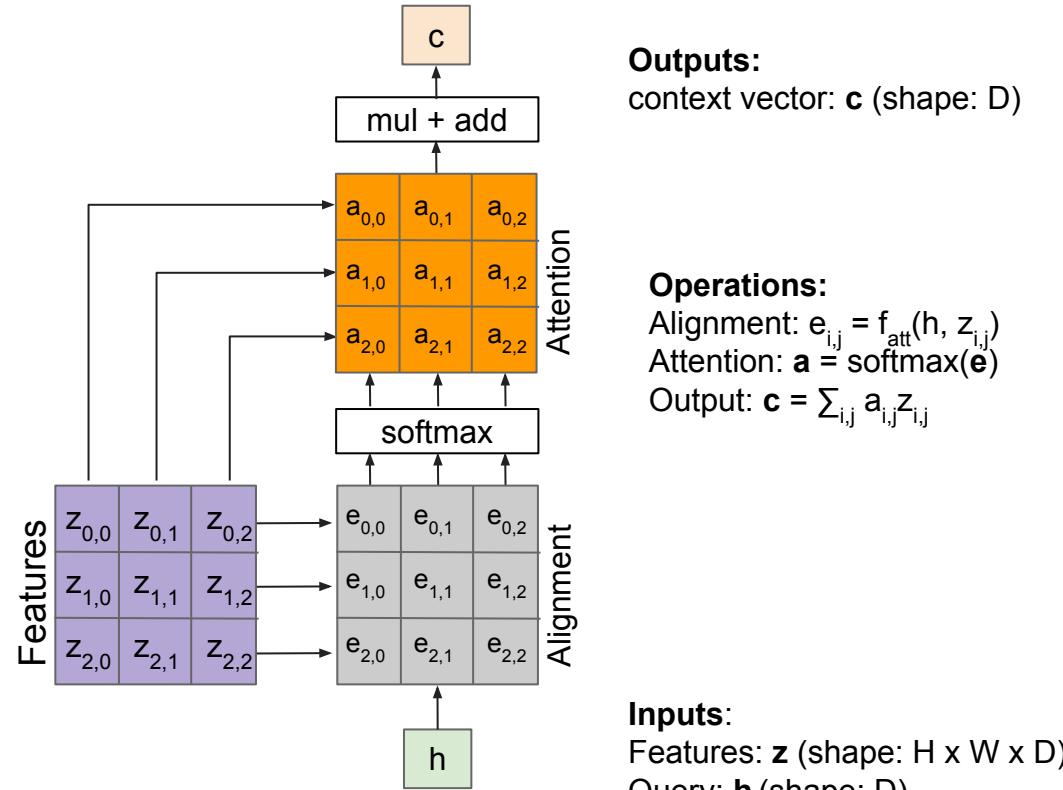
Attention we just saw in image captioning



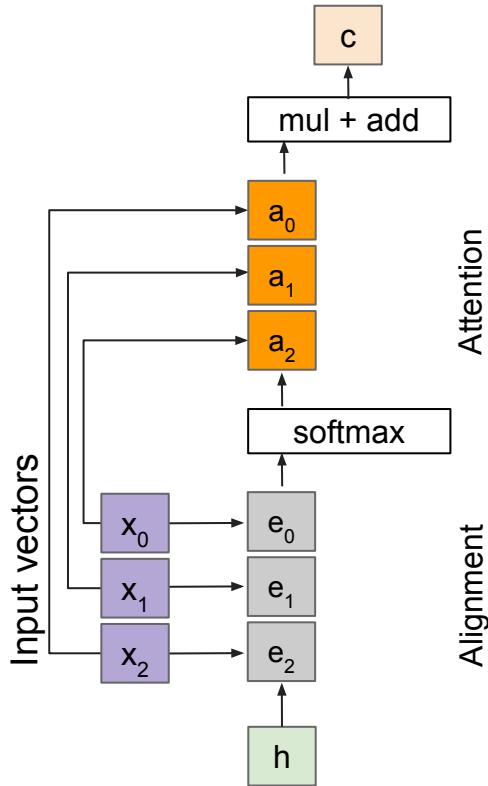
Attention we just saw in image captioning



Attention we just saw in image captioning



General attention layer



Outputs:

context vector: \mathbf{c} (shape: D)

Operations:

Alignment: $\mathbf{e}_i = f_{att}(\mathbf{h}, \mathbf{x}_i)$

Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

Output: $\mathbf{c} = \sum_i \mathbf{a}_i \mathbf{x}_i$

Inputs:

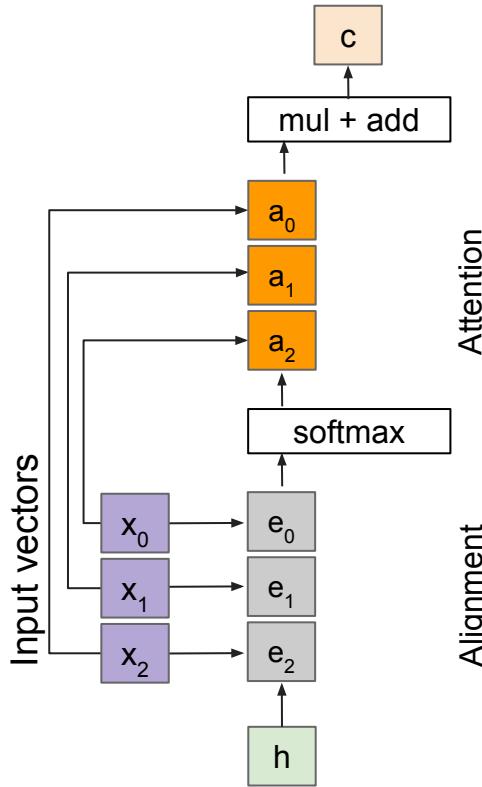
Input vectors: \mathbf{x} (shape: N x D)

Query: \mathbf{h} (shape: D)

Attention operation is **permutation invariant**.

- Doesn't care about ordering of the features
- Stretch $H \times W = N$ into N vectors

General attention layer



Outputs:

context vector: c (shape: D)

Operations:

Alignment: $e_i = h \cdot x_i$

Attention: $a = \text{softmax}(e)$

Output: $c = \sum_i a_i x_i$

Change $f_{\text{att}}(\cdot)$ to a simple dot product

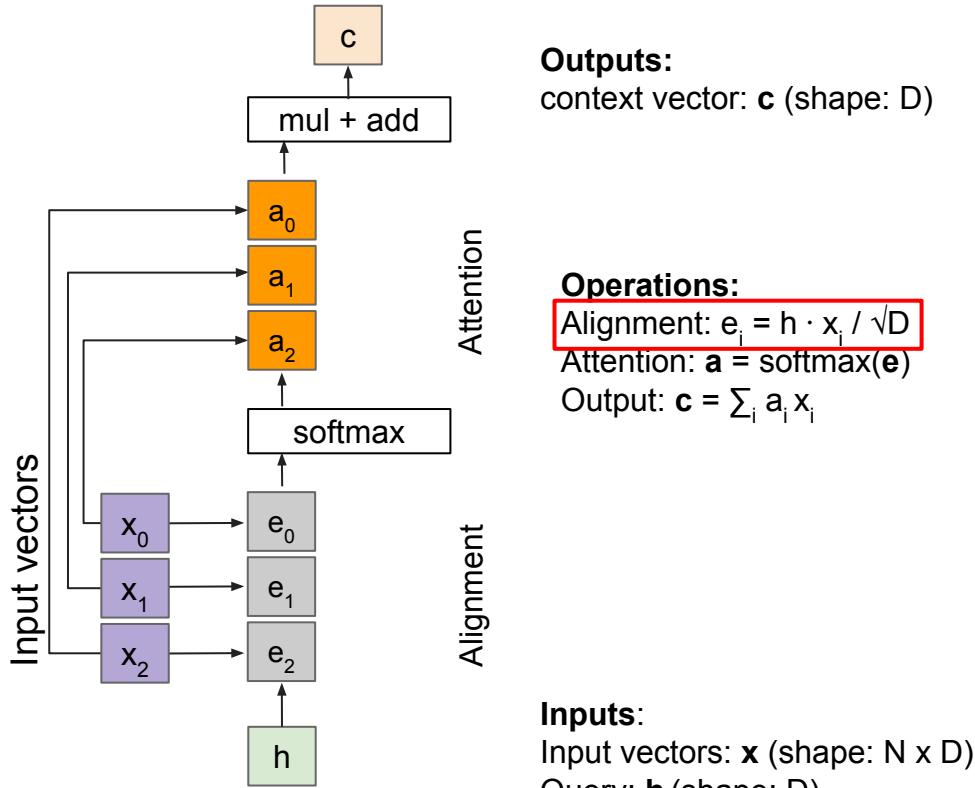
- only works well with key & value transformation trick (will mention in a few slides)

Inputs:

Input vectors: x (shape: N x D)

Query: h (shape: D)

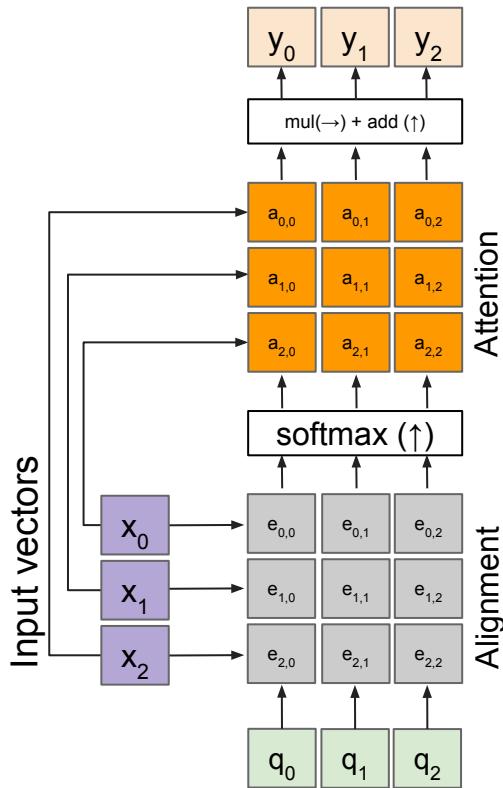
General attention layer



Change $f_{\text{att}}(\cdot)$ to a **scaled** simple dot product

- Larger dimensions means more terms in the dot product sum.
- So, the variance of the logits is higher. Large magnitude vectors will produce much higher logits.
- So, the post-softmax distribution has lower-entropy, assuming logits are IID.
- Ultimately, these large magnitude vectors will cause softmax to peak and assign very little weight to all others
- Divide by \sqrt{D} to reduce effect of large magnitude vectors

General attention layer



Outputs:

context vectors: \mathbf{y} (shape: D)

Multiple query vectors

- each query creates a new output context vector

Operations:

$$\text{Alignment: } e_{i,j} = q_j \cdot x_i / \sqrt{D}$$

$$\text{Attention: } \mathbf{a} = \text{softmax}(\mathbf{e})$$

$$\text{Output: } y_j = \sum_i a_{i,j} x_i$$

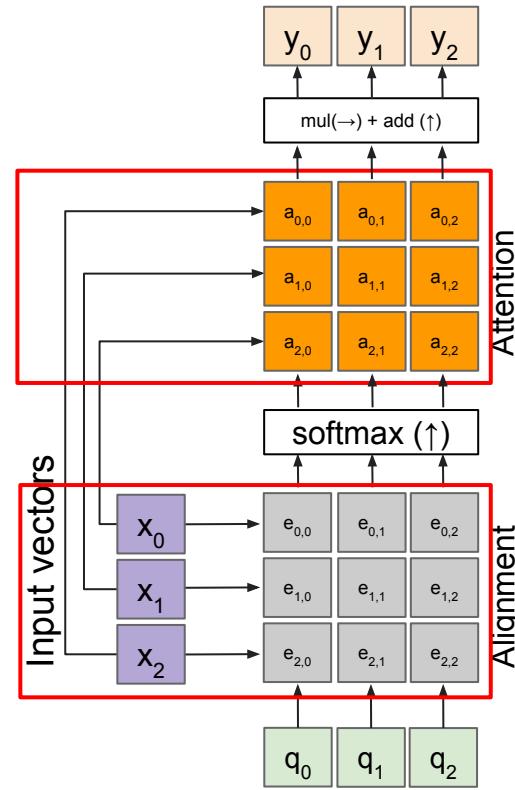
Multiple query vectors

Inputs:

Input vectors: \mathbf{x} (shape: N x D)

Queries: \mathbf{q} (shape: M x D)

General attention layer



Outputs:

context vectors: \mathbf{y} (shape: D)

Operations:

Alignment: $e_{i,j} = \mathbf{q}_j \cdot \mathbf{x}_i / \sqrt{D}$

Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

Output: $\mathbf{y}_j = \sum_i a_{i,j} \mathbf{x}_i$

Notice that the input vectors are used for both the alignment as well as the attention calculations.

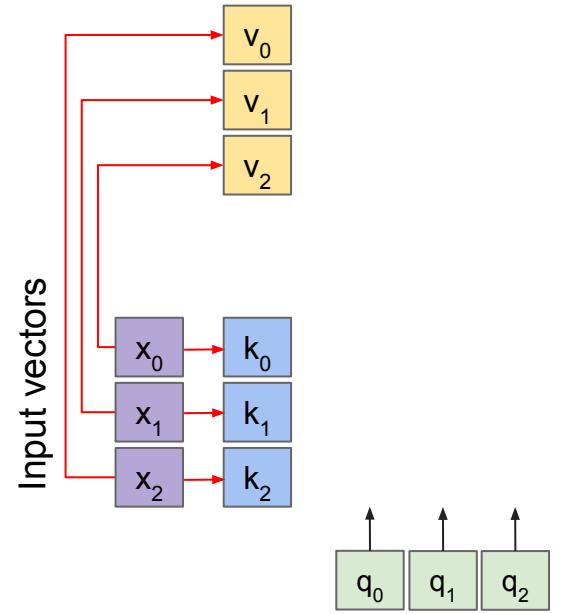
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

Inputs:

Input vectors: \mathbf{x} (shape: N x D)

Queries: \mathbf{q} (shape: M x D)

General attention layer



Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$

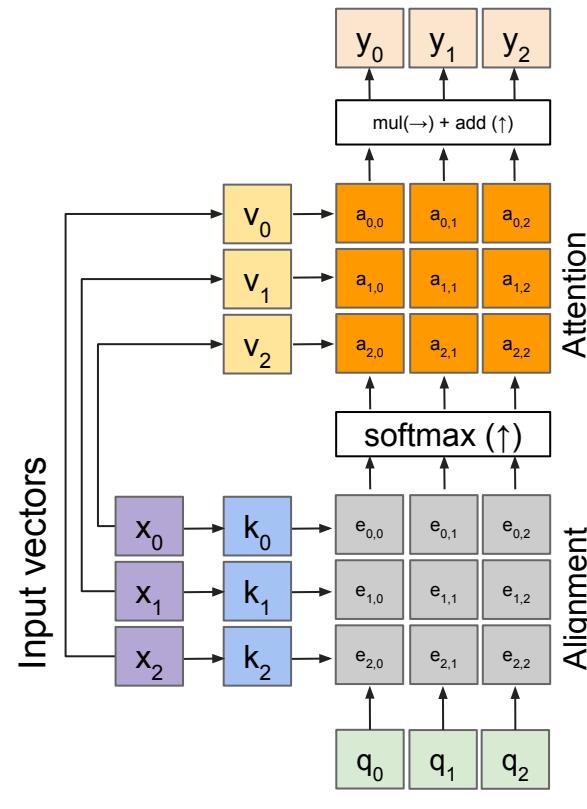
Notice that the input vectors are used for both the alignment as well as the attention calculations.

- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

Inputs:

Input vectors: \mathbf{x} (shape: $N \times D$)
Queries: \mathbf{q} (shape: $M \times D_k$)

General attention layer



Outputs:

context vectors: \mathbf{y} (shape: D_y)

The input and output dimensions can now change depending on the key and value FC layers

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$

Value vectors: $\mathbf{v} = \mathbf{x}W_v$

Alignment: $e_{i,j} = \mathbf{q}_j \cdot \mathbf{k}_i / \sqrt{D_k}$

Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

Output: $\mathbf{y}_j = \sum_i a_{i,j} \mathbf{v}_i$

Notice that the input vectors are used for both the alignment as well as the attention calculations.

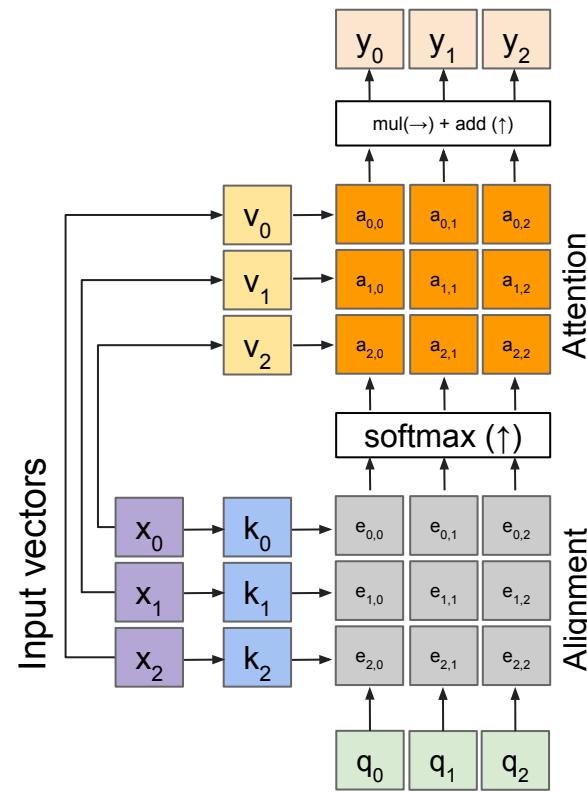
- We can add more expressivity to the layer by adding a different FC layer before each of the two steps.

Inputs:

Input vectors: \mathbf{x} (shape: $N \times D$)

Queries: \mathbf{q} (shape: $M \times D_k$)

General attention layer



Outputs:

context vectors: \mathbf{y} (shape: D_y)

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$

Value vectors: $\mathbf{v} = \mathbf{x}W_v$

Alignment: $e_{i,j} = q_i \cdot k_j / \sqrt{D_k}$

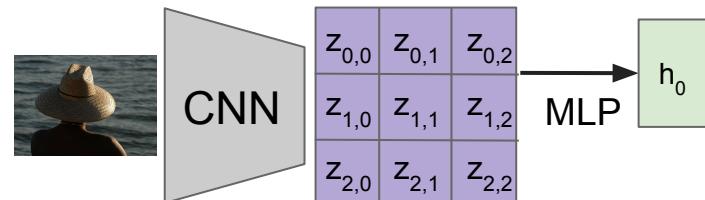
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

Output: $y_j = \sum_i a_{i,j} v_i$

Recall that the query vector was a function of the input vectors

Encoder: $h_0 = f_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features
 $f_w(\cdot)$ is an MLP

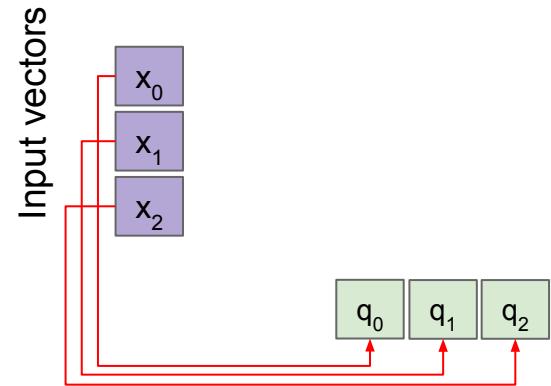


Inputs:

Input vectors: \mathbf{x} (shape: $N \times D$)

Queries: \mathbf{q} (shape: $M \times D_k$)

Self attention layer



Operations:

Key vectors: $k = xW_k$

Value vectors: $v = xW_v$

Query vectors: $q = xW_q$

Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$

Attention: $a = \text{softmax}(e)$

Output: $y_j = \sum_i a_{i,j} v_i$

We can calculate the query vectors from the input vectors, therefore, defining a "self-attention" layer.

Instead, query vectors are calculated using a FC layer.

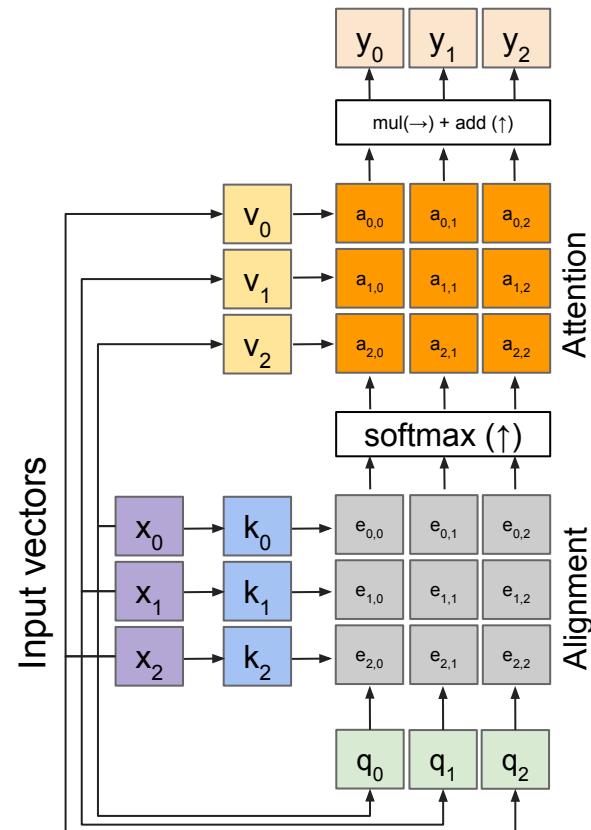
No input query vectors anymore

Inputs:

Input vectors: x (shape: $N \times D$)

Queries: q (shape: $M \times D_K$)

Self attention layer



Outputs:

context vectors: \mathbf{y} (shape: D_y)

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$

Value vectors: $\mathbf{v} = \mathbf{x}W_v$

Query vectors: $\mathbf{q} = \mathbf{x}W_q$

Alignment: $\mathbf{e}_{i,j} = \mathbf{q}_j \cdot \mathbf{k}_i / \sqrt{D}$

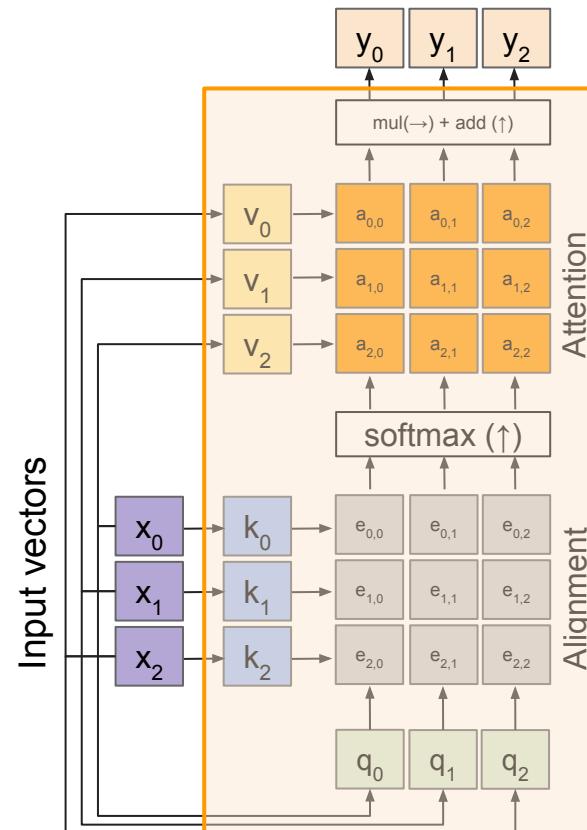
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

Output: $\mathbf{y}_j = \sum_i \mathbf{a}_{i,j} \mathbf{v}_i$

Inputs:

Input vectors: \mathbf{x} (shape: $N \times D$)

Self attention layer - attends over sets of inputs



Outputs:

context vectors: \mathbf{y} (shape: D_y)

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$

Value vectors: $\mathbf{v} = \mathbf{x}W_v$

Query vectors: $\mathbf{q} = \mathbf{x}W_q$

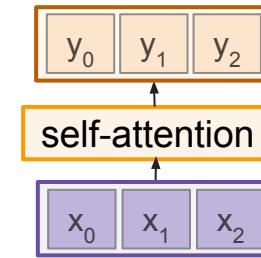
Alignment: $e_{i,j} = q_j \cdot k_i / \sqrt{D}$

Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

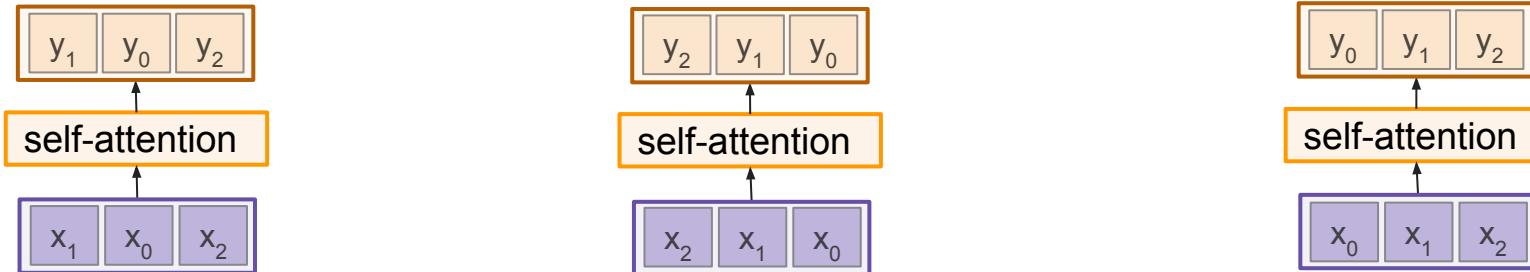
Output: $y_j = \sum_i a_{i,j} v_i$

Inputs:

Input vectors: \mathbf{x} (shape: $N \times D$)



Self attention layer - attends over sets of inputs

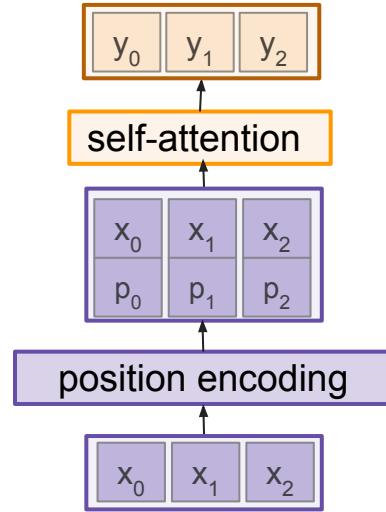


Permutation equivariant

Self-attention layer doesn't care about the orders of the inputs!

Problem: How can we encode ordered sequences like language or spatially ordered image features?

Positional encoding



Concatenate/add special positional encoding p_j to each input vector x_j

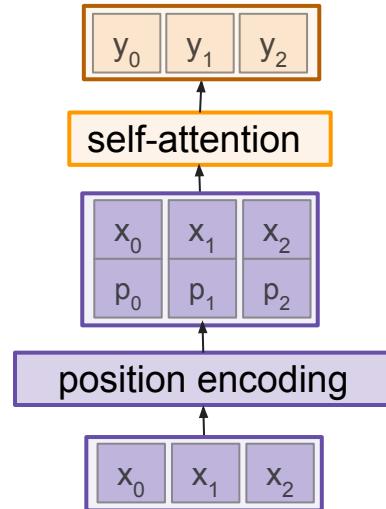
We use a function $pos: N \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

So, $p_j = pos(j)$

Desiderata of $pos(\cdot)$:

1. It should output a **unique** encoding for each time-step (word's position in a sentence)
2. **Distance** between any two time-steps should be consistent across sentences with different lengths.
3. Our model should generalize to **longer** sentences without any efforts. Its values should be bounded.
4. It must be **deterministic**.

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

We use a function $pos: N \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

So, $p_j = pos(j)$

Options for $pos(\cdot)$

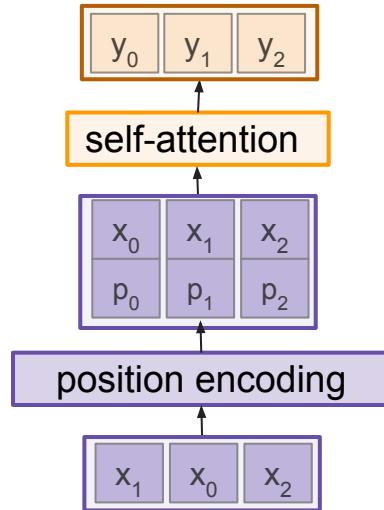
1. Learn a lookup table:
 - Learn parameters to use for $pos(t)$ for $t \in [0, T]$
 - Lookup table contains $T \times d$ parameters.

Desiderata of $pos(\cdot)$:

1. It should output a **unique** encoding for each time-step (word's position in a sentence)
2. **Distance** between any two time-steps should be consistent across sentences with different lengths.
3. Our model should generalize to **longer** sentences without any efforts. Its values should be bounded.
4. It must be **deterministic**.

Vaswani et al, "Attention is all you need", NeurIPS 2017

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

We use a function $pos: N \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

So, $p_j = pos(j)$

Options for $pos(\cdot)$

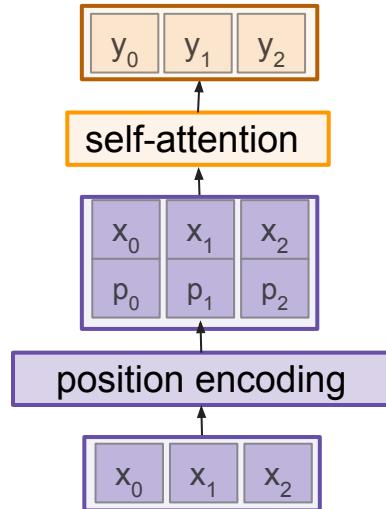
1. Learn a lookup table:
 - Learn parameters to use for $pos(t)$ for $t \in [0, T]$
 - Lookup table contains $T \times d$ parameters.
2. Design a fixed function with the desiderata

$$p(t) = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \vdots \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}_d$$

where $\omega_k = \frac{1}{10000^{2k/d}}$

Vaswani et al, "Attention is all you need", NeurIPS 2017

Positional encoding



Concatenate special positional encoding p_j to each input vector x_j

We use a function $pos: \mathbb{N} \rightarrow \mathbb{R}^d$ to process the position j of the vector into a d -dimensional vector

So, $p_j = pos(j)$

Options for $pos(\cdot)$

1. Learn a lookup table:
 - Learn parameters to use for $pos(t)$ for $t \in [0, T]$
 - Lookup table contains $T \times d$ parameters.
2. Design a fixed function with the desiderata

Intuition:

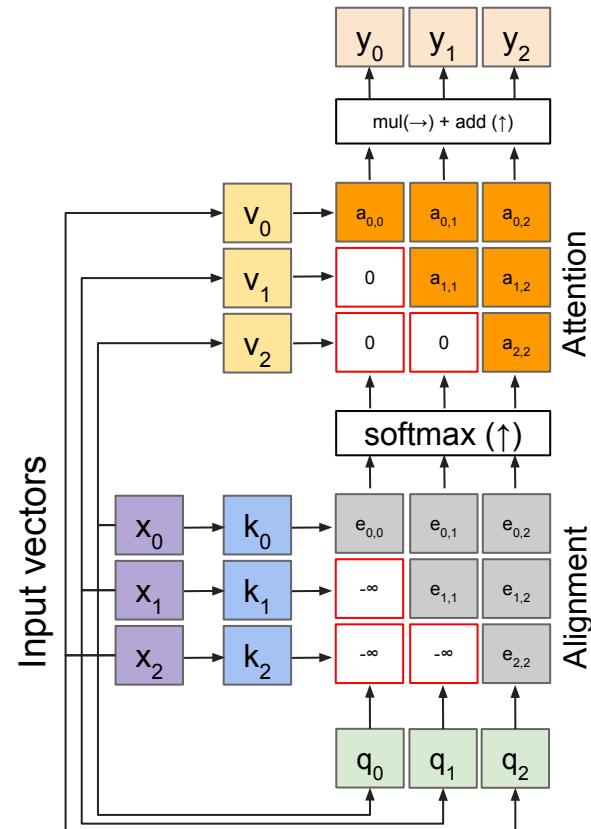
$$p(t) = \begin{bmatrix} \sin(\omega_1 \cdot t) \\ \cos(\omega_1 \cdot t) \\ \sin(\omega_2 \cdot t) \\ \cos(\omega_2 \cdot t) \\ \vdots \\ \sin(\omega_{d/2} \cdot t) \\ \cos(\omega_{d/2} \cdot t) \end{bmatrix}_d$$

where $\omega_k = \frac{1}{10000^{2k/d}}$

Vaswani et al, "Attention is all you need", NeurIPS 2017

[image source](#)

Masked self-attention layer



Outputs:

context vectors: \mathbf{y} (shape: D_y)

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}W_k$
Value vectors: $\mathbf{v} = \mathbf{x}W_v$
Query vectors: $\mathbf{q} = \mathbf{x}W_q$
Alignment: $e_{i,j} = \mathbf{q}_j \cdot \mathbf{k}_i / \sqrt{D}$
Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$
Output: $\mathbf{y}_j = \sum_i a_{i,j} \mathbf{v}_i$

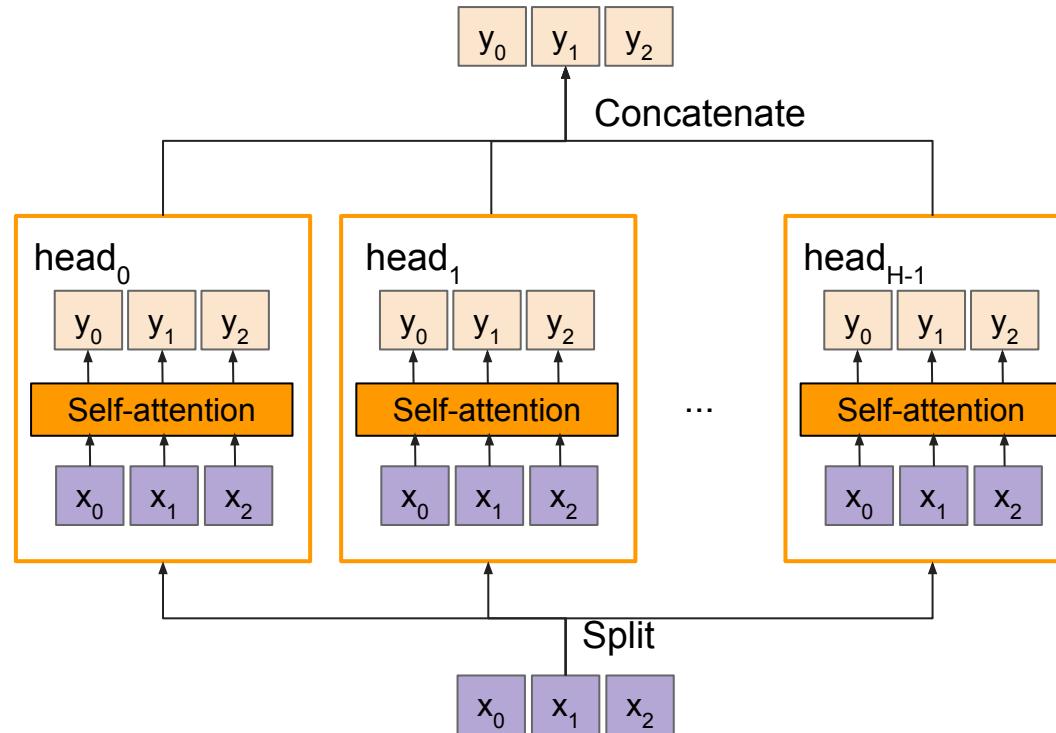
- Prevent vectors from looking at future vectors.
- Manually set alignment scores to $-\infty$

Inputs:

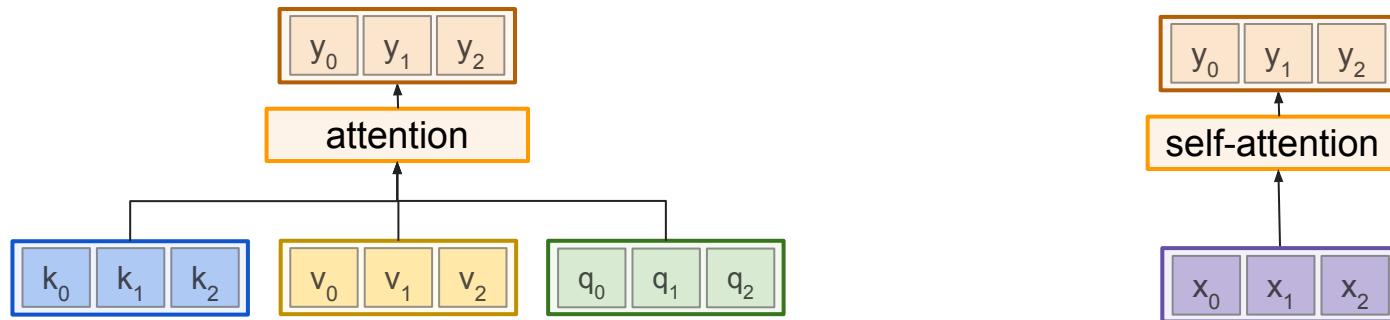
Input vectors: \mathbf{x} (shape: $N \times D$)

Multi-head self attention layer

- Multiple self-attention heads in parallel



General attention versus self-attention



Example: CNN with Self-Attention

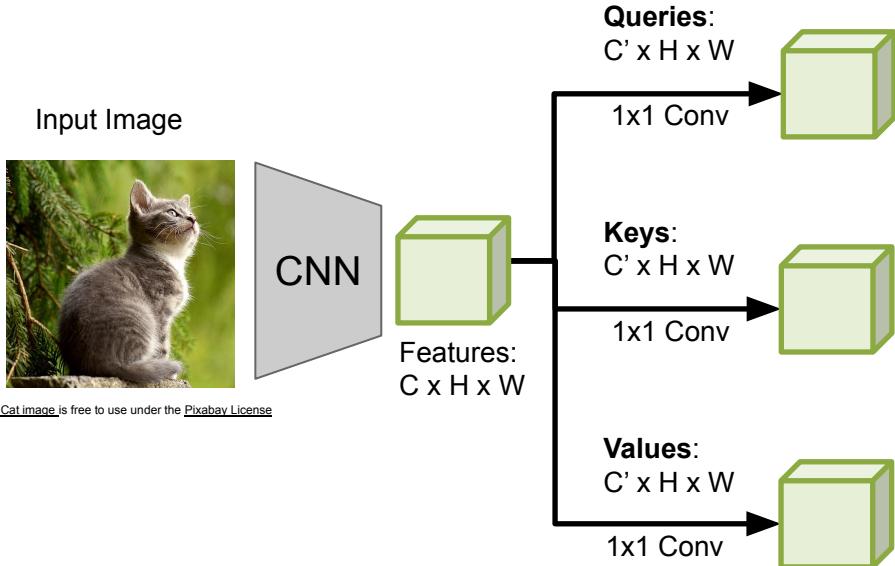
Input Image



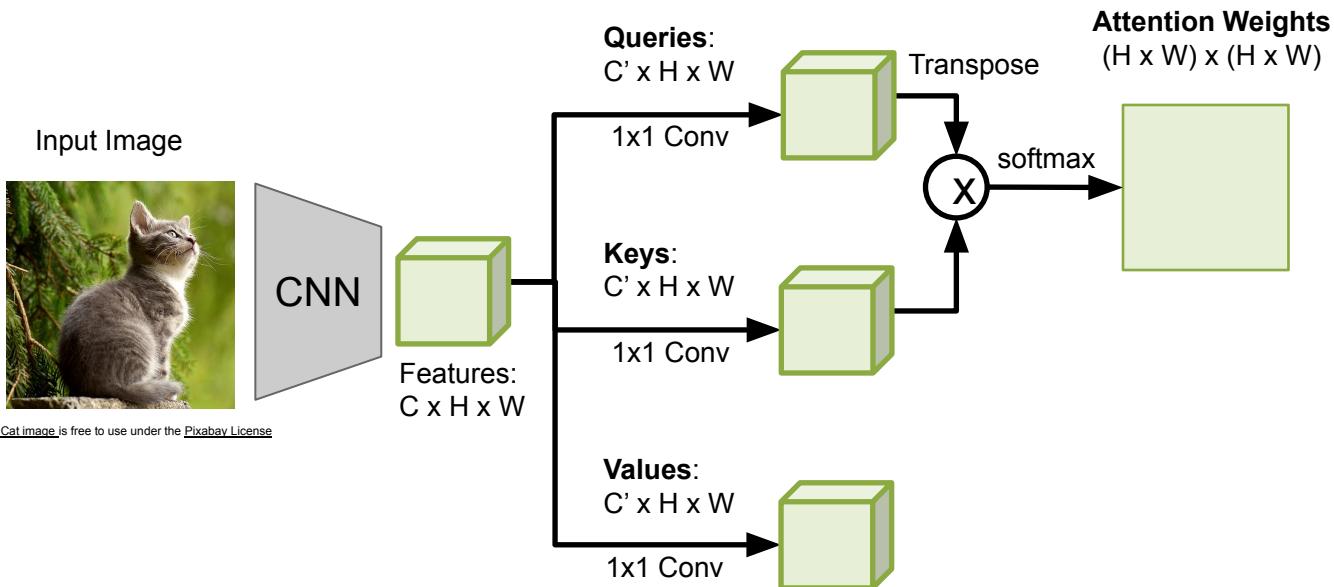
Features:
 $C \times H \times W$

[Cat image](#) is free to use under the [Pixabay License](#)

Example: CNN with Self-Attention

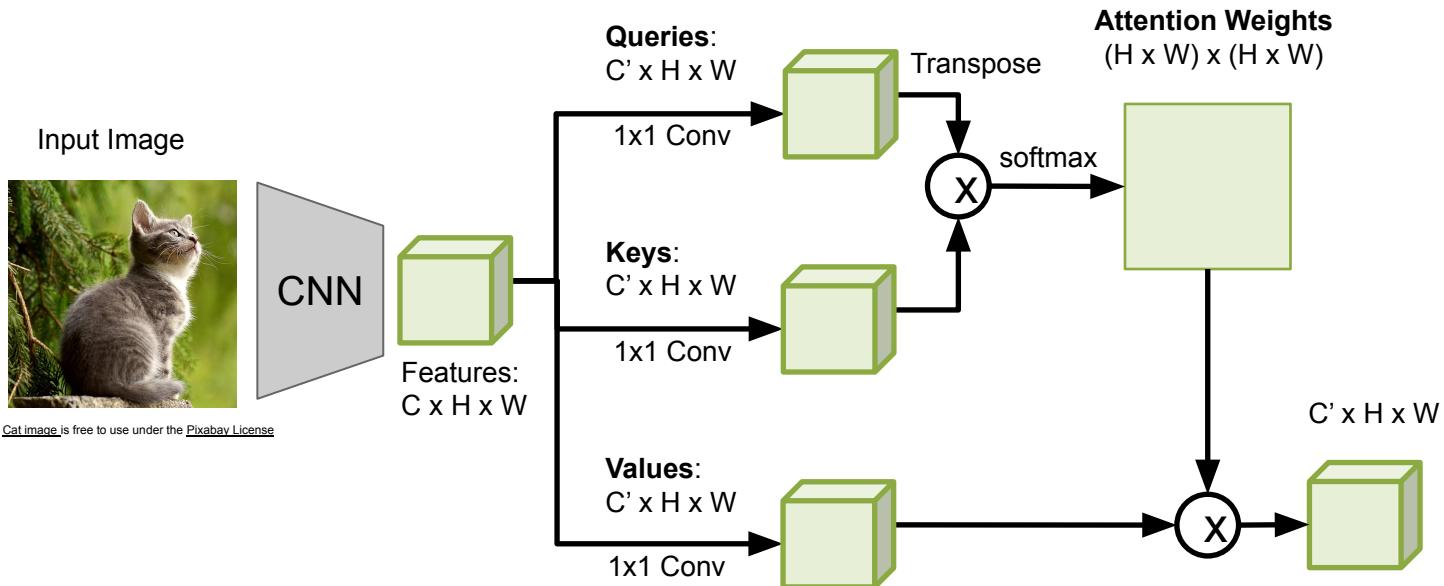


Example: CNN with Self-Attention



[Cat image](#) is free to use under the [Pixabay License](#)

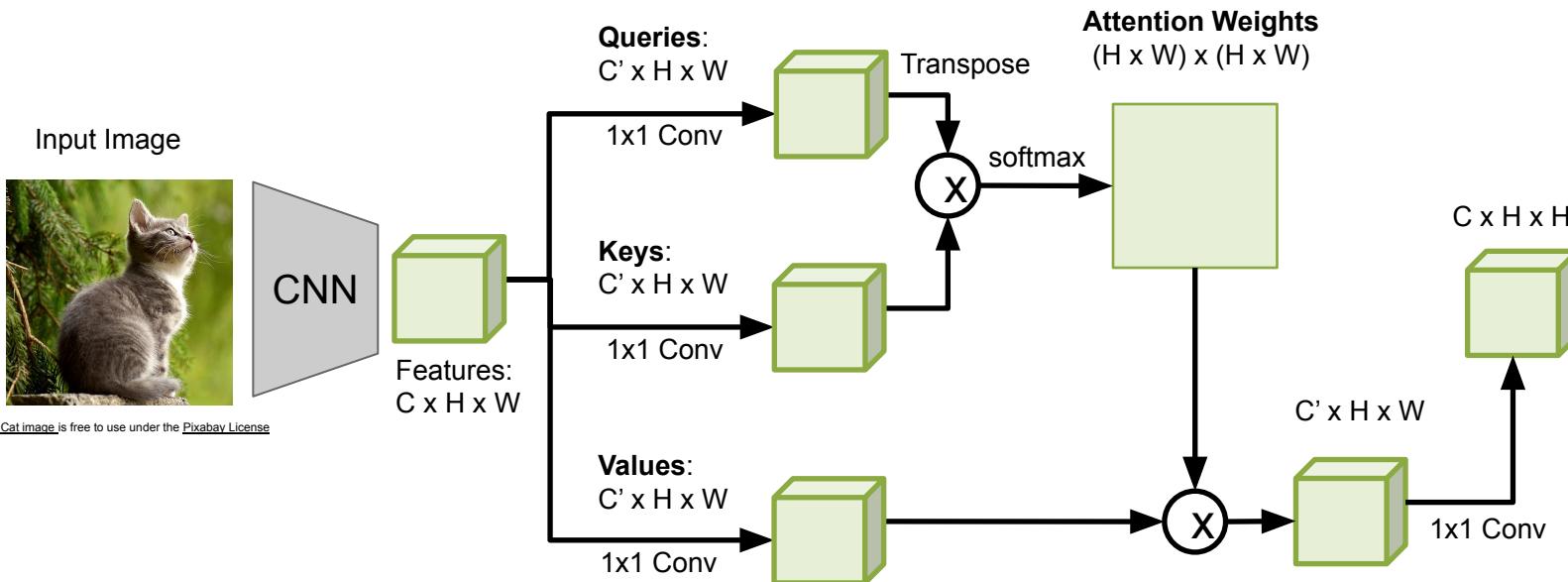
Example: CNN with Self-Attention



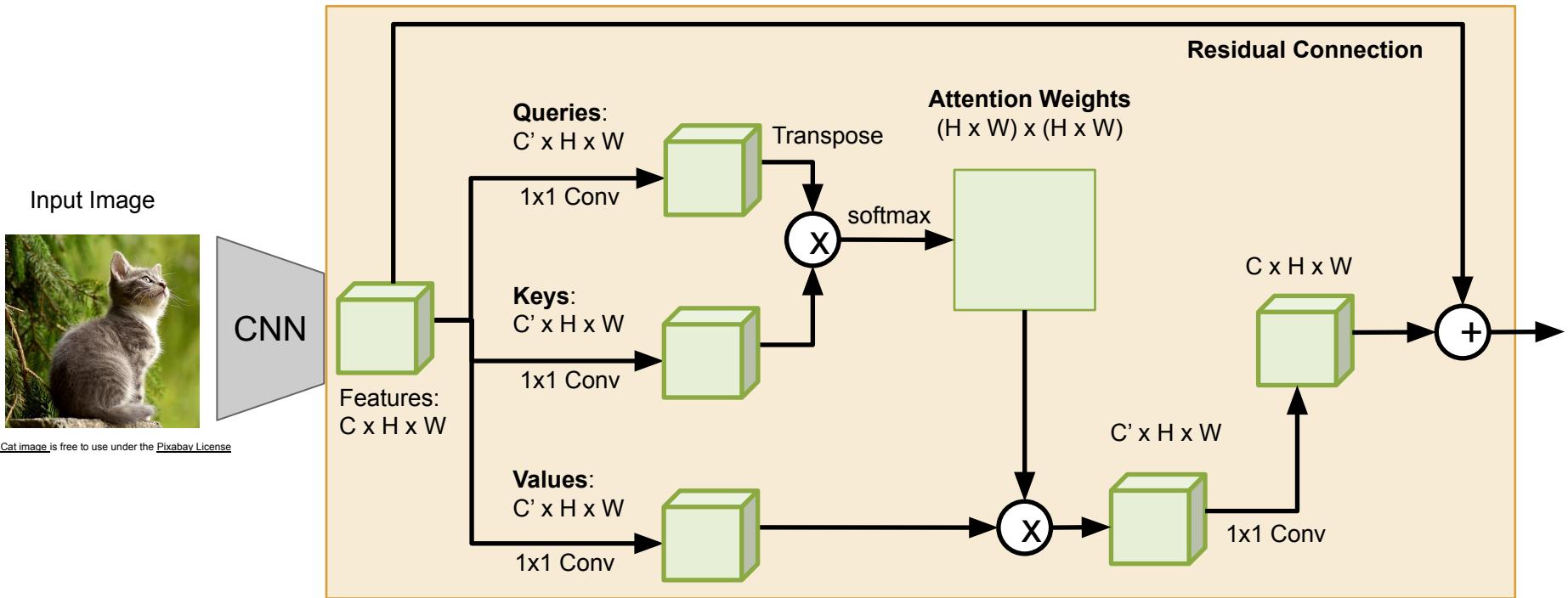
Zhang et al. "Self-Attention Generative Adversarial Networks". ICML 2018

Slide credit: Justin Johnson

Example: CNN with Self-Attention



Example: CNN with Self-Attention



Self-Attention Module

Zhang et al, "Self-Attention Generative Adversarial Networks", ICML 2018

Slide credit: Justin Johnson

Comparing RNNs to Transformer

RNNs

- (+) LSTMs work reasonably well for long sequences.
- (-) Expects an ordered sequences of inputs
- (-) Sequential computation: subsequent hidden states can only be computed after the previous ones are done.

Transformer:

- (+) Good at long sequences. Each attention calculation looks at all inputs.
- (+) Can operate over unordered sets or ordered sequences with positional encodings.
- (+) Parallel computation: All alignment and attention scores for all inputs can be done in parallel.
- (-) Requires a lot of memory: $N \times M$ alignment and attention scalers need to be calculated and stored for a single self-attention head. (but GPUs are getting bigger and better)

Attention Is All You Need

Ashish Vaswani*
Google Brain
avaswani@google.com

Noam Shazeer*
Google Brain
noam@google.com

Niki Parmar*
Google Research
nikip@google.com

Jakob Uszkoreit*
Google Research
usz@google.com

Llion Jones*
Google Research
llion@google.com

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

Lukasz Kaiser*
Google Brain
lukaszkaiser@google.com

Illia Polosukhin* ‡
illia.polosukhin@gmail.com

“ImageNet Moment for Natural Language Processing”

Pretraining:

Download a lot of text from the internet

Train a giant Transformer model for language modeling

Finetuning:

Fine-tune the Transformer on your own NLP task

On the Opportunities and Risks of Foundation Models

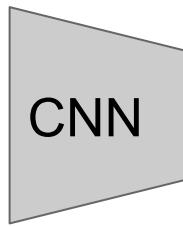
Rishi Bommasani* Drew A. Hudson Ehsan Adeli Russ Altman Simran Arora
Sydney von Arx Michael S. Bernstein Jeannette Bohg Antoine Bosselut Emma Brunskill
Erik Brynjolfsson Shyamal Buch Dallas Card Rodrigo Castellon Niladri Chatterji
Annie Chen Kathleen Creel Jared Quincy Davis Dorottya Demszky Chris Donahue
Moussa Doumbouya Esin Durmus Stefano Ermon John Etchemendy Kawin Ethayarajh
Li Fei-Fei Chelsea Finn Trevor Gale Lauren Gillespie Karan Goel Noah Goodman
Shelby Grossman Neel Guha Tatsunori Hashimoto Peter Henderson John Hewitt
Daniel E. Ho Jenny Hong Kyle Hsu Jing Huang Thomas Icard Saahil Jain
Dan Jurafsky Pratyusha Kalluri Siddharth Karamcheti Geoff Keeling Fereshte Khani
Omar Khattab Pang Wei Koh Mark Krass Ranjay Krishna Rohith Kuditipudi
Ananya Kumar Faisal Ladhak Mina Lee Tony Lee Jure Leskovec Isabelle Levent
Xiang Lisa Li Xuechen Li Tengyu Ma Ali Malik Christopher D. Manning
Suvir Mirchandani Eric Mitchell Zanele Munyikwa Suraj Nair Avanika Narayan
Deepak Narayanan Ben Newman Allen Nie Juan Carlos Niebles Hamed Nilforoshan
Julian Nyarko Giray O gut Laurel Orr Isabel Papadimitriou Joon Sung Park Chris Piech
Eva Portelance Christopher Potts Aditi Raghunathan Rob Reich Hongyu Ren
Frieda Rong Yusuf Roohani Camilo Ruiz Jack Ryan Christopher Ré Dorsa Sadigh
Shiori Sagawa Keshav Santhanam Andy Shih Krishnan Srinivasan Alex Tamkin
Rohan Taori Armin W. Thomas Florian Tramèr Rose E. Wang William Wang Bohan Wu
Jiajun Wu Yuhuai Wu Sang Michael Xie Michihiro Yasunaga Jiaxuan You Matei Zaharia
Michael Zhang Tianyi Zhang Xikun Zhang Yuhui Zhang Lucia Zheng Kaitlyn Zhou
Percy Liang*¹

Center for Research on Foundation Models (CRFM)
Stanford Institute for Human-Centered Artificial Intelligence (HAI)
Stanford University

Image Captioning using Transformers

Input: Image I

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



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:
 $H \times W \times D$

Image Captioning using Transformers

Input: Image I

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

Encoder: $\mathbf{c} = T_w(\mathbf{z})$

where \mathbf{z} is spatial CNN features

$T_w(\cdot)$ is the transformer encoder

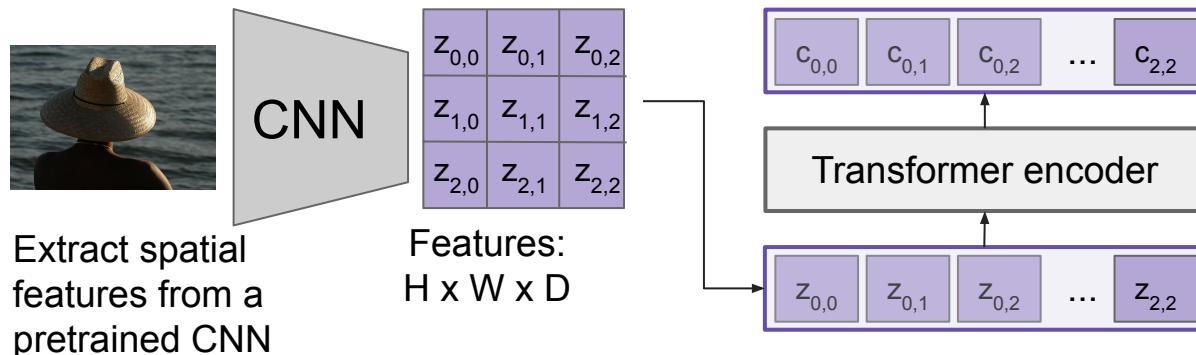


Image Captioning using Transformers

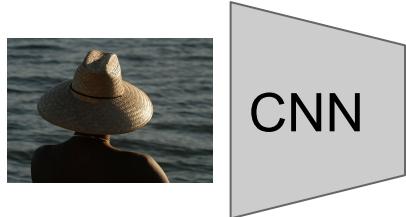
Input: Image I

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

Decoder: $y_t = T_D(\mathbf{y}_{0:t-1}, \mathbf{c})$
where $T_D(\cdot)$ is the transformer decoder

Encoder: $\mathbf{c} = T_W(\mathbf{z})$

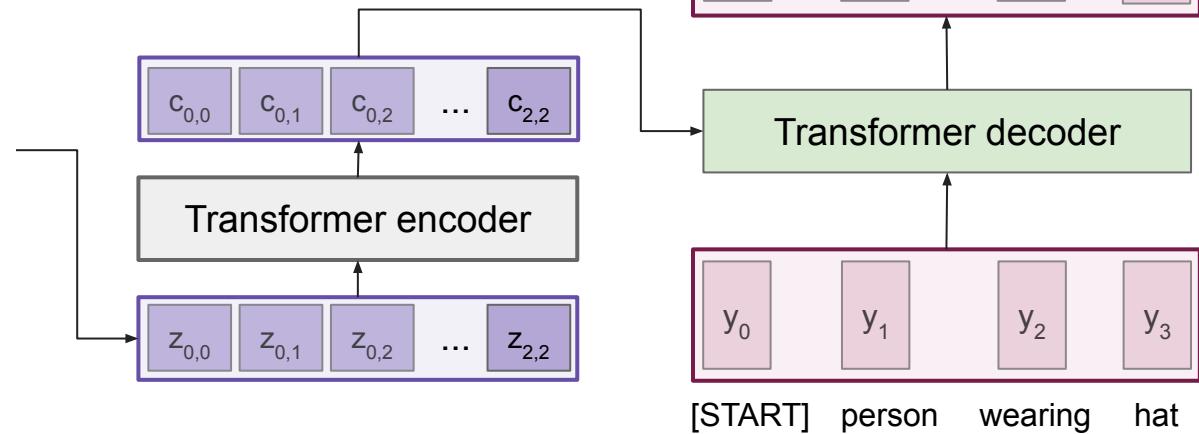
where \mathbf{z} is spatial CNN features
 $T_W(\cdot)$ is the transformer encoder



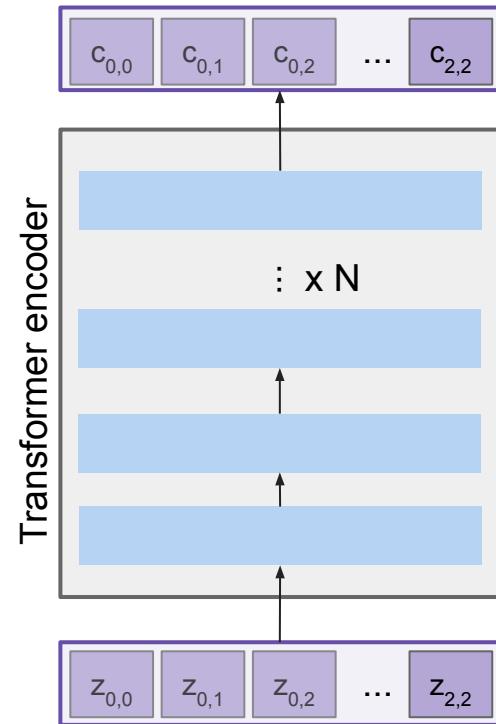
Extract spatial
features from a
pretrained CNN

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

Features:
 $H \times W \times D$



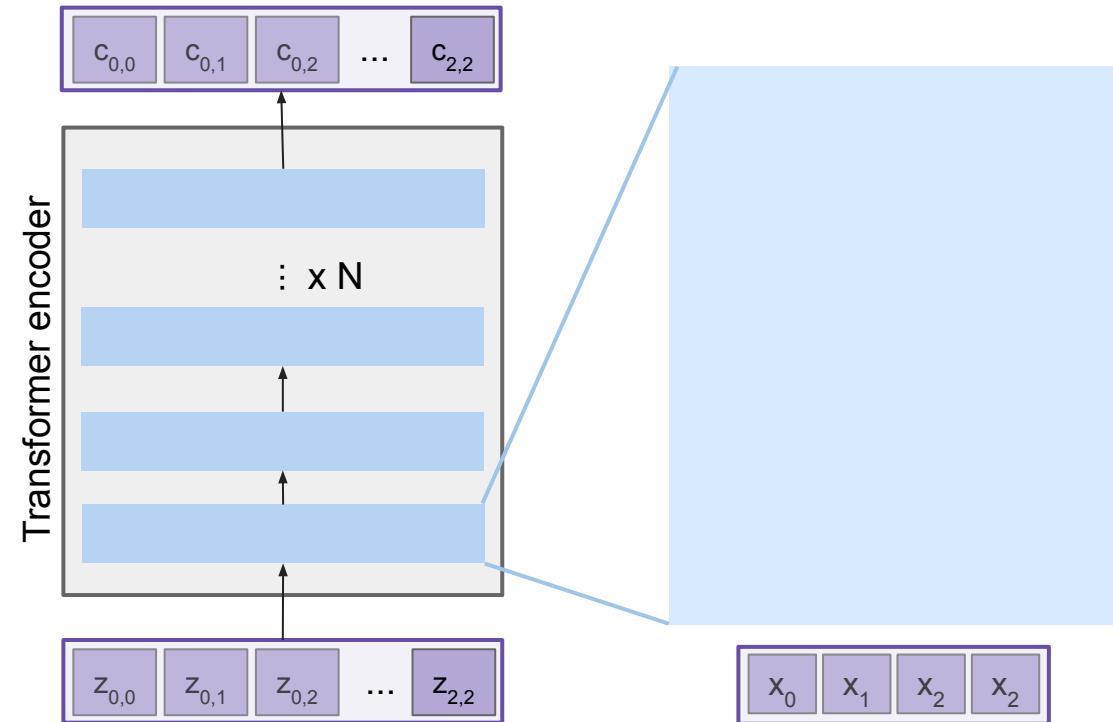
The Transformer encoder block



Made up of N encoder blocks.

In vaswani et al. $N = 6$, $D_q = 512$

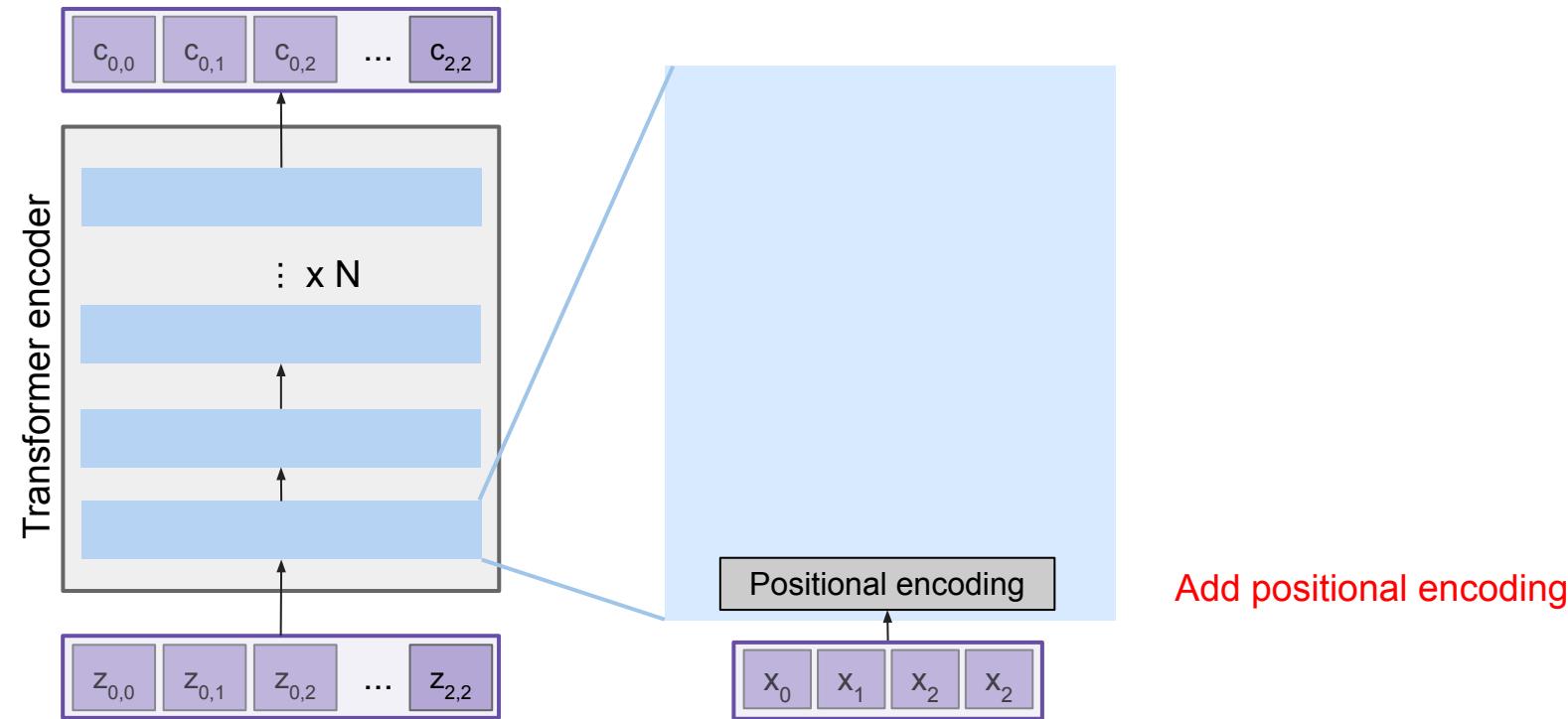
The Transformer encoder block



Let's dive into one encoder block

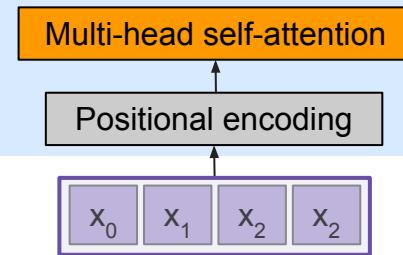
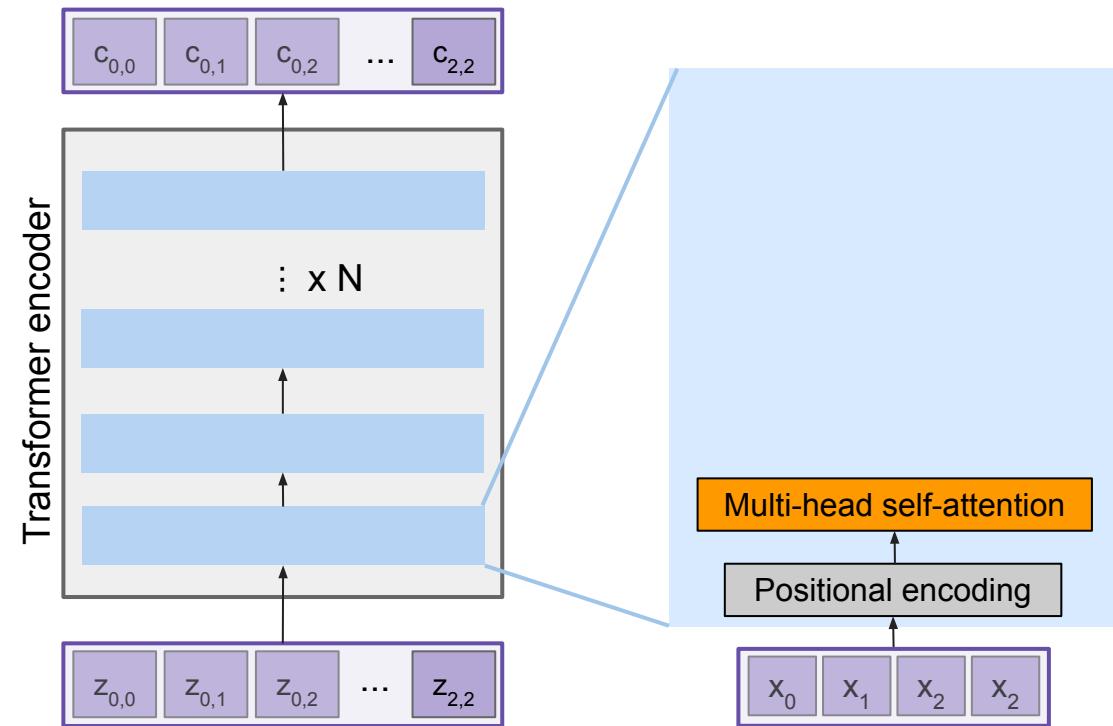
Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer encoder block



Vaswani et al, "Attention is all you need", NeurIPS 2017

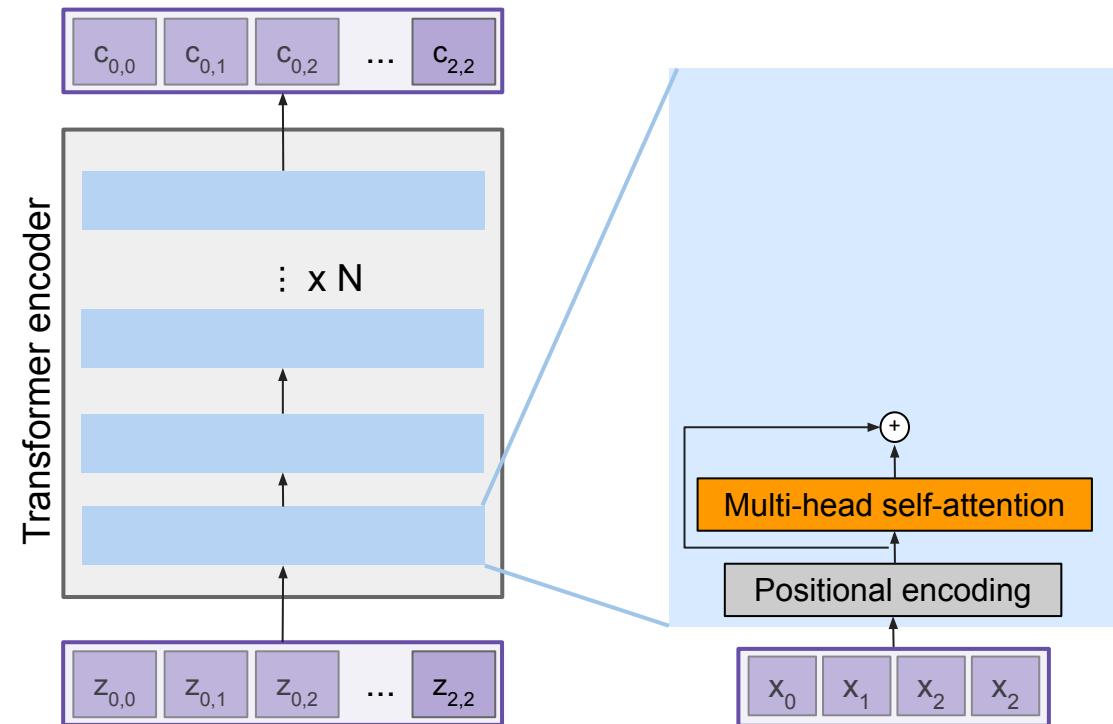
The Transformer encoder block



Attention attends over all the vectors

Add positional encoding

The Transformer encoder block



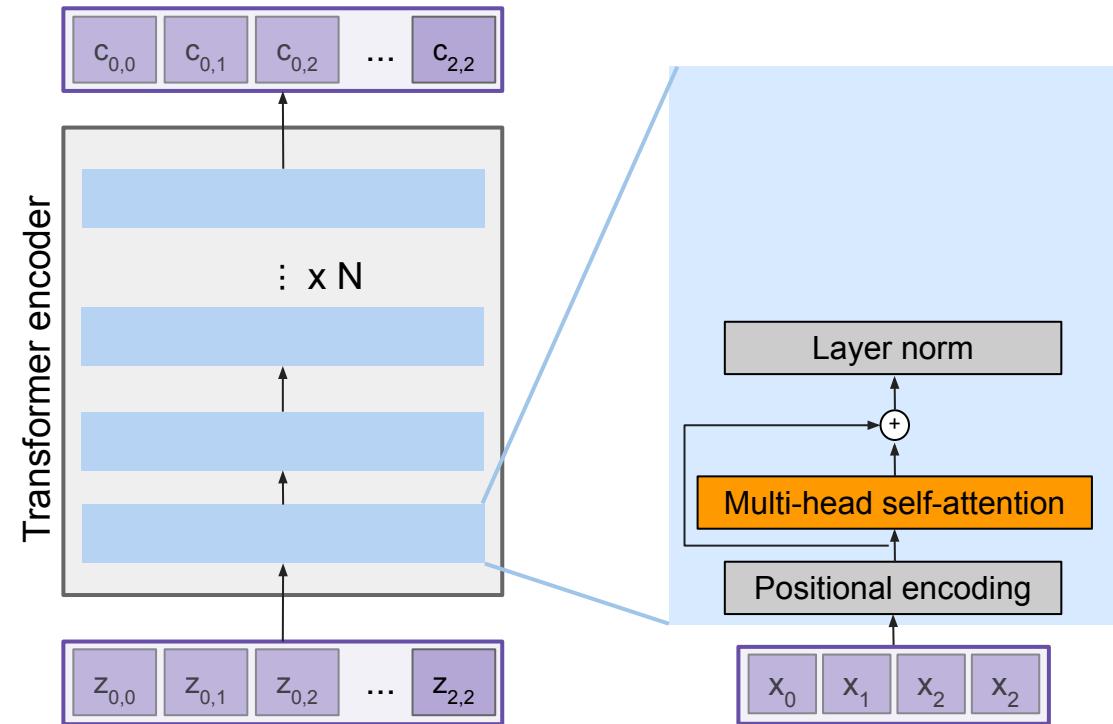
Residual connection

Attention attends over all the vectors

Add positional encoding

Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer encoder block



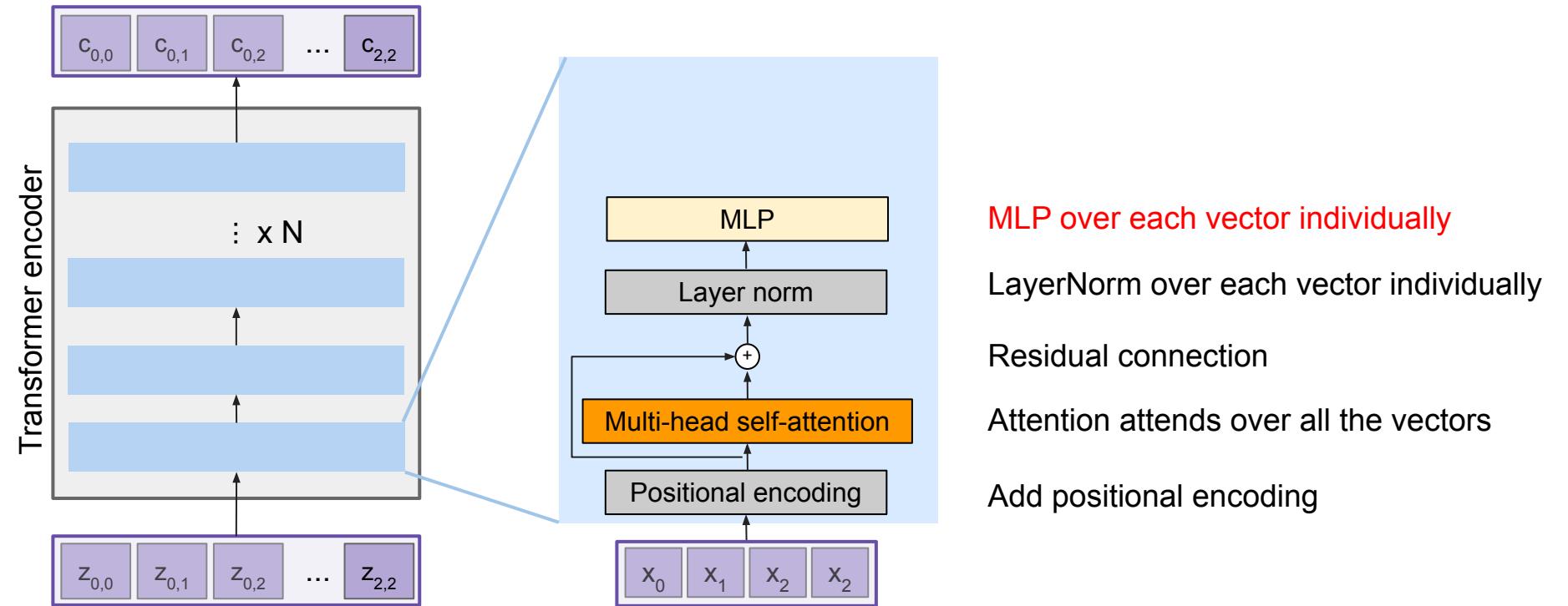
LayerNorm over each vector individually

Residual connection

Attention attends over all the vectors

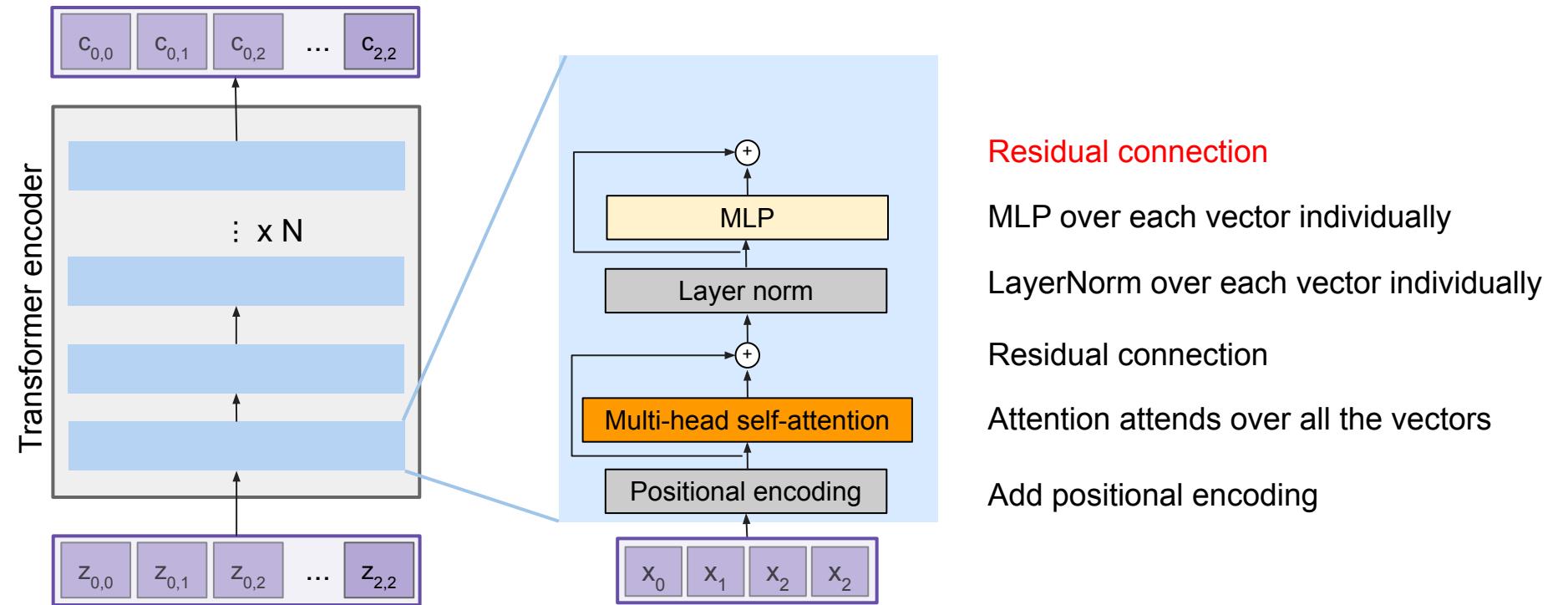
Add positional encoding

The Transformer encoder block



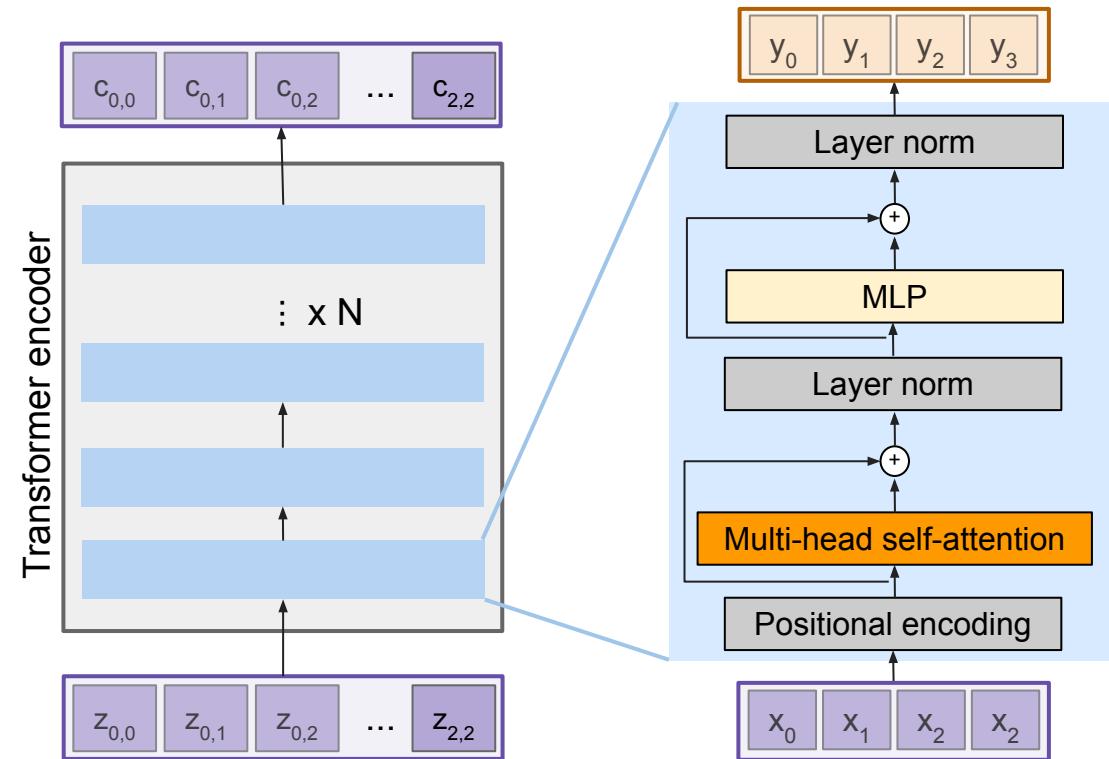
Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer encoder block



Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer encoder block



Transformer Encoder Block:

Inputs: Set of vectors \mathbf{x}

Outputs: Set of vectors \mathbf{y}

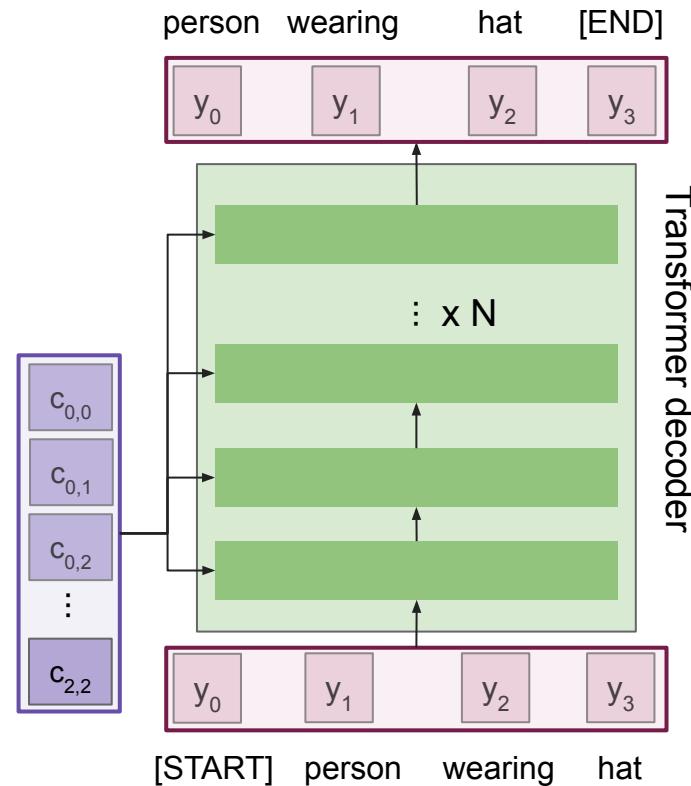
Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

Vaswani et al, "Attention is all you need", NeurIPS 2017

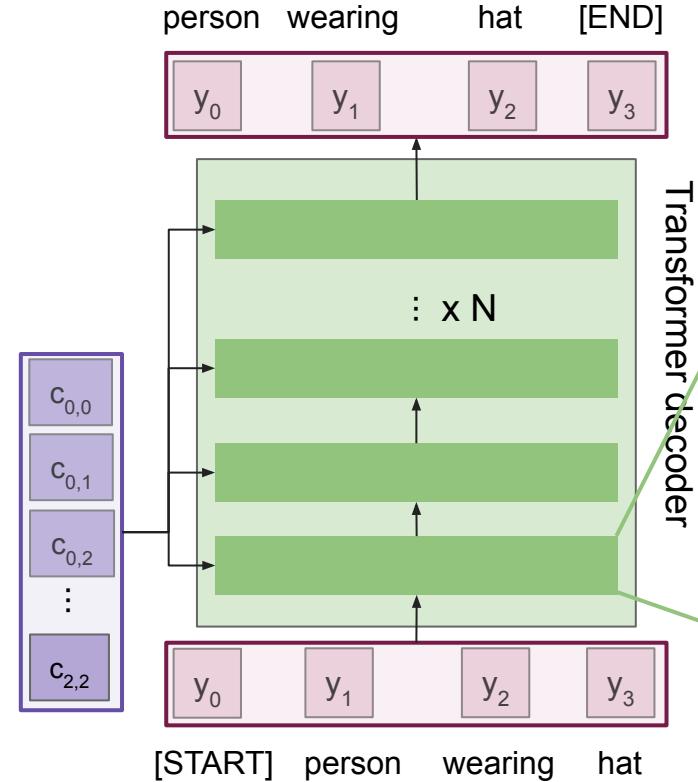
The Transformer decoder block



Made up of N decoder blocks.

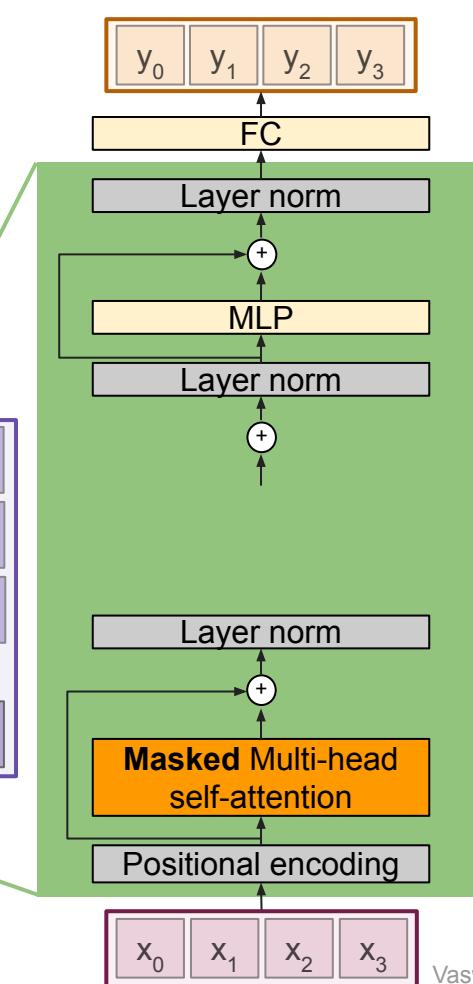
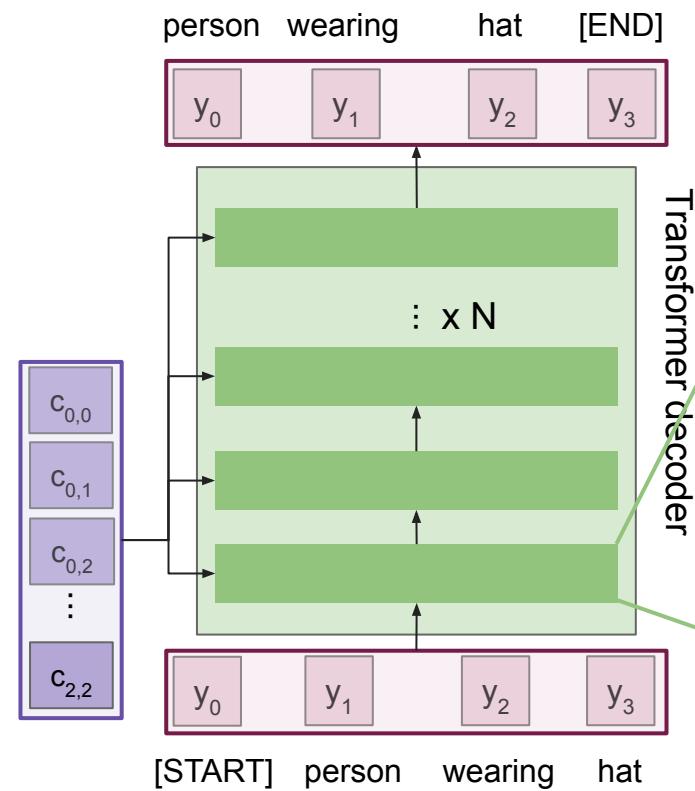
In vaswani et al. $N = 6, D_q = 512$

The Transformer decoder block



Let's dive into the transformer decoder block

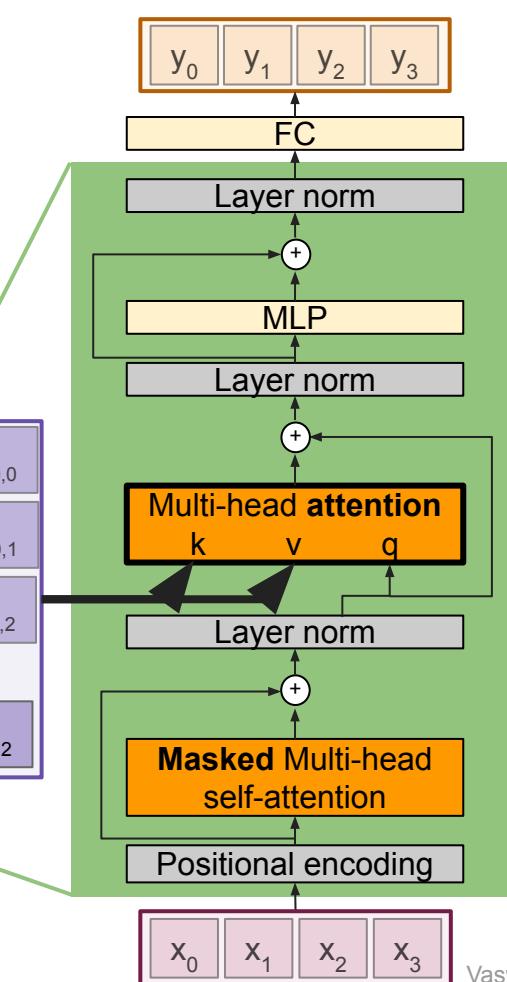
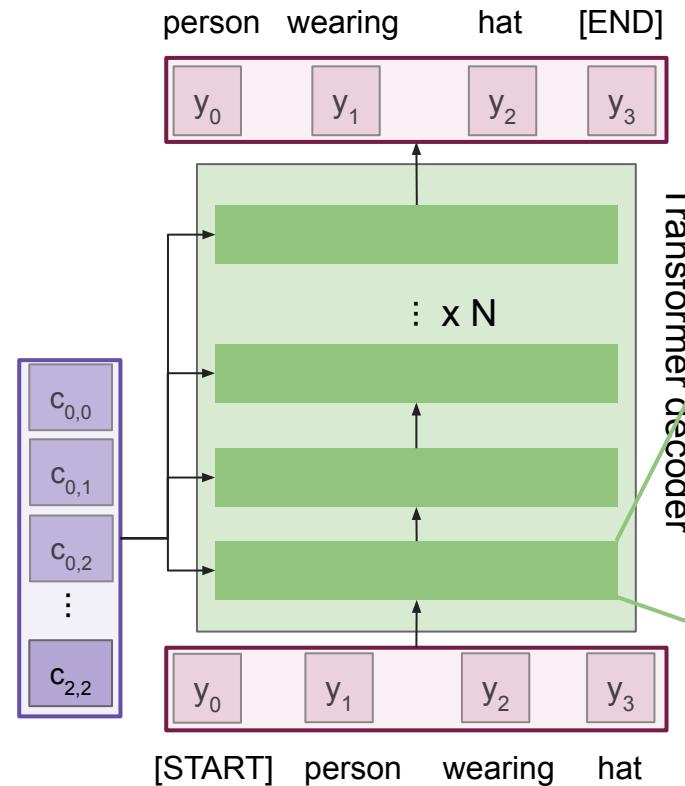
The Transformer Decoder block



Most of the network is the same the transformer encoder.

Vaswani et al, "Attention is all you need", NeurIPS 2017

The Transformer Decoder block

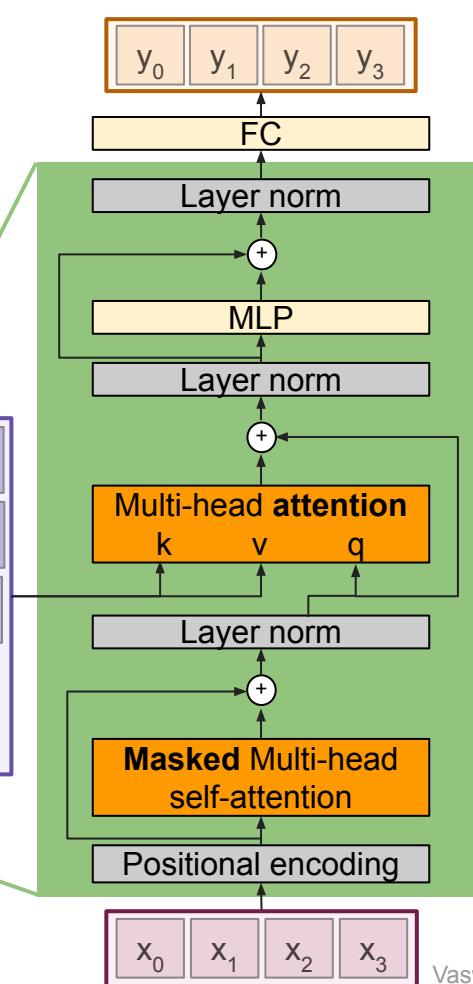
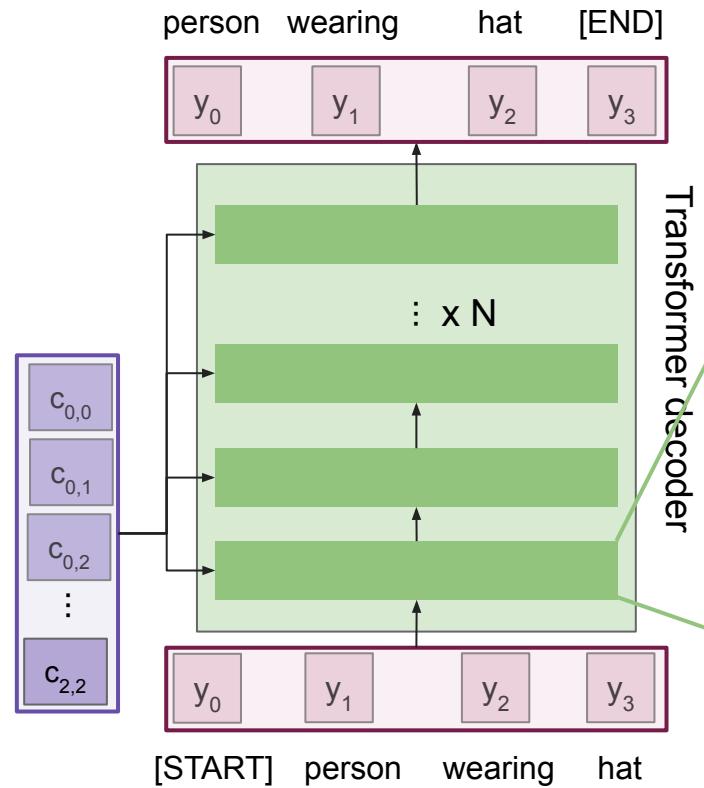


Multi-head attention block attends over the transformer encoder outputs.

For image captions, this is how we inject image features into the decoder.

Vaswani et al., "Attention is all you need", NeurIPS 2017

The Transformer Decoder block



Transformer Decoder Block:

Inputs: Set of vectors \mathbf{x} and Set of context vectors \mathbf{c} .

Outputs: Set of vectors \mathbf{y} .

Masked Self-attention only interacts with past inputs.

Multi-head attention block is NOT self-attention. It attends over encoder outputs.

Highly scalable, highly parallelizable, but high memory usage.

Vaswani et al., "Attention is all you need", NeurIPS 2017

Image Captioning using transformers

- No recurrence at all

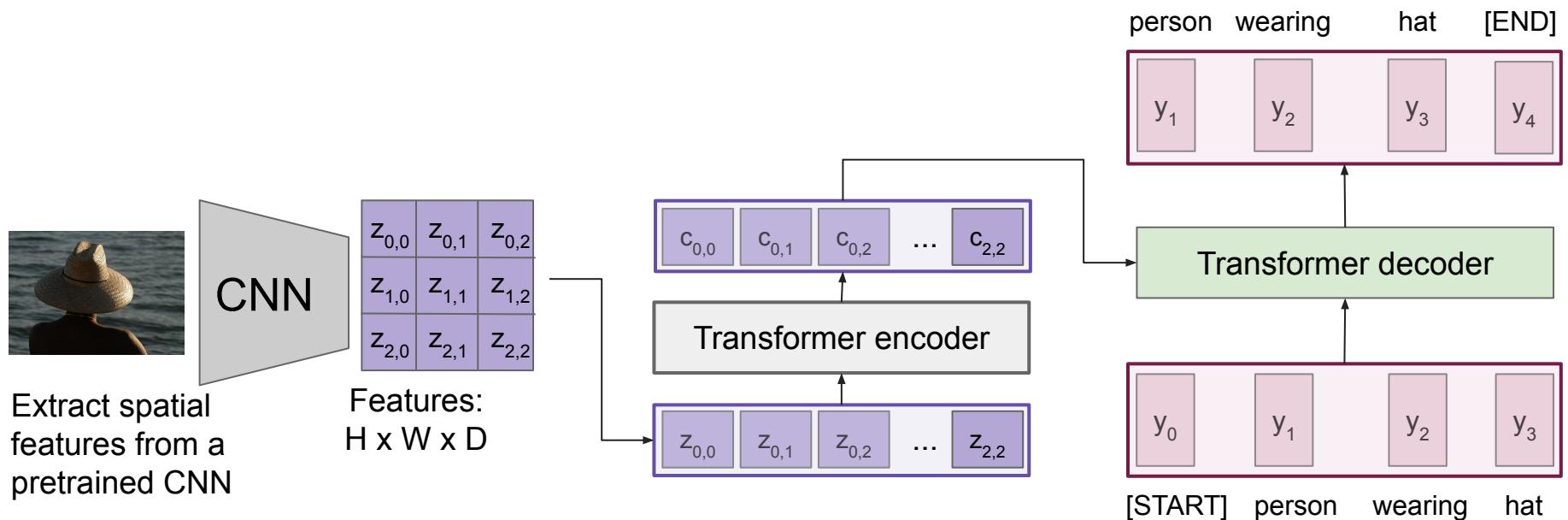
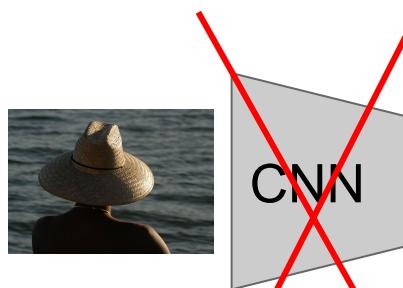


Image Captioning using transformers

- Perhaps we don't need convolutions at all?



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:
 $H \times W \times D$

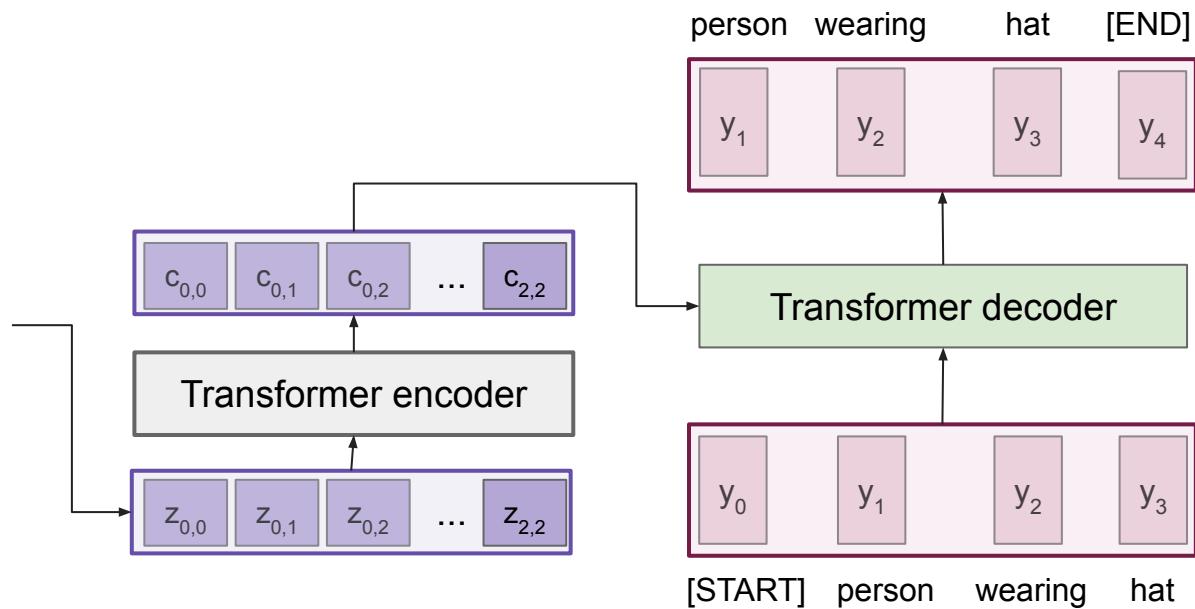
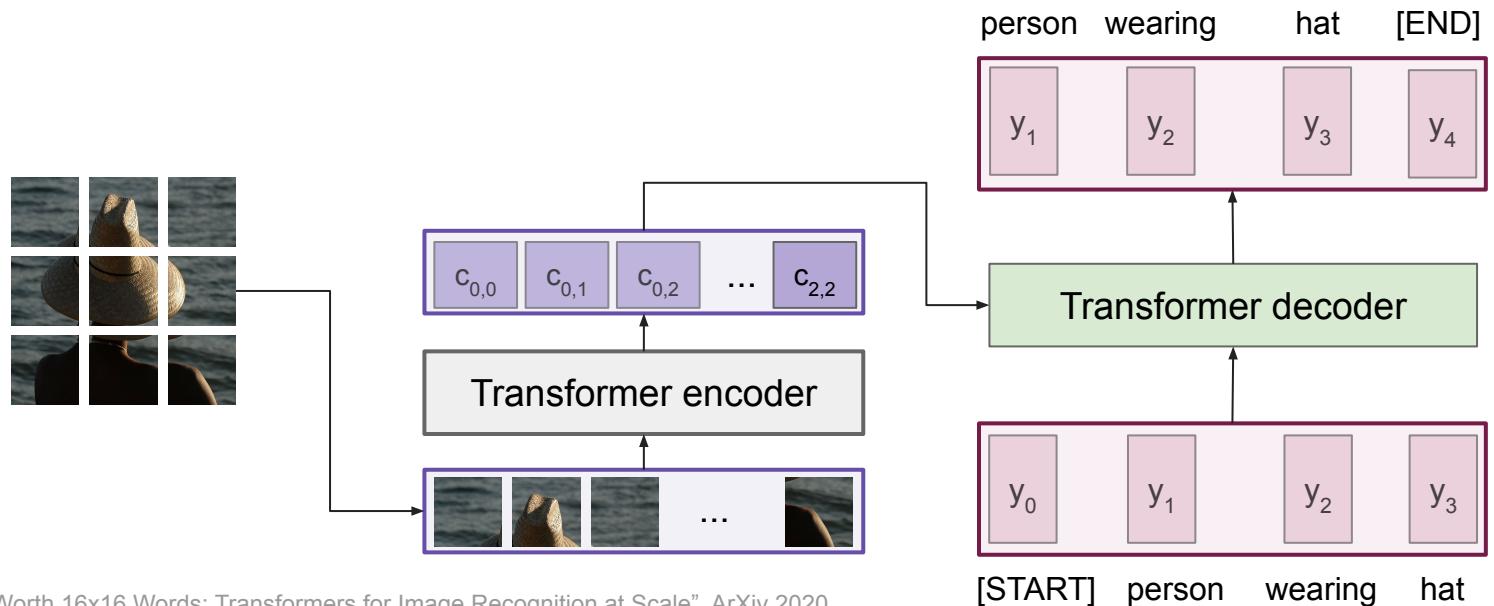


Image Captioning using **ONLY** transformers

- Transformers from pixels to language



Dosovitskiy et al., "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020
[Colab link](#) to an implementation of vision transformers

Vision Transformers vs. ResNets

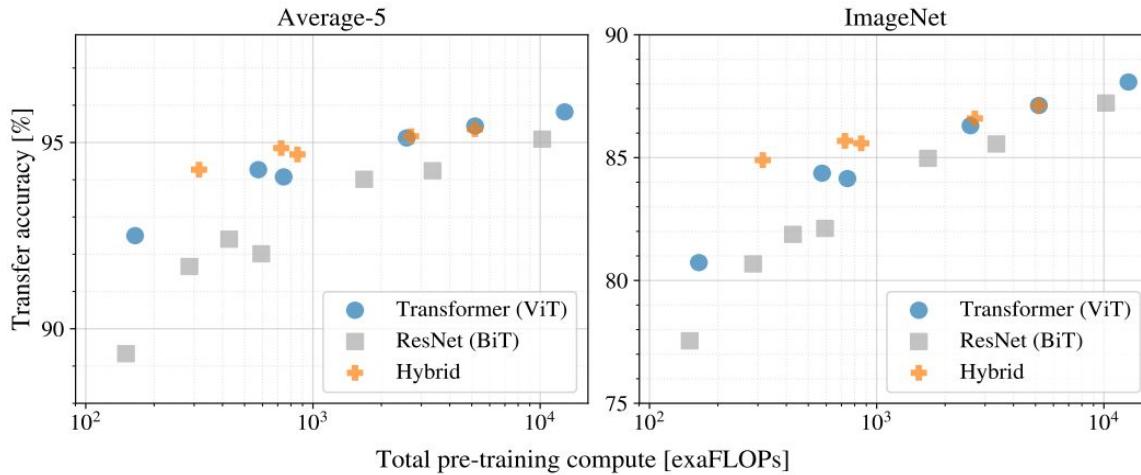
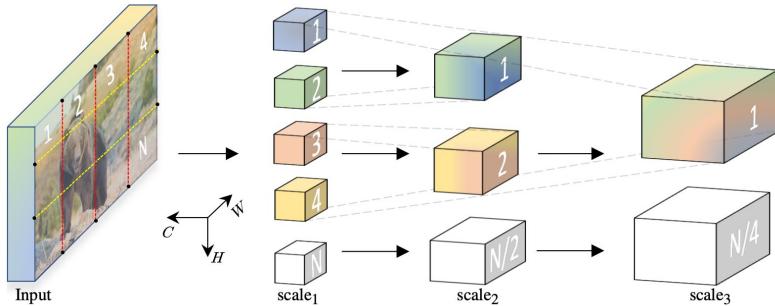


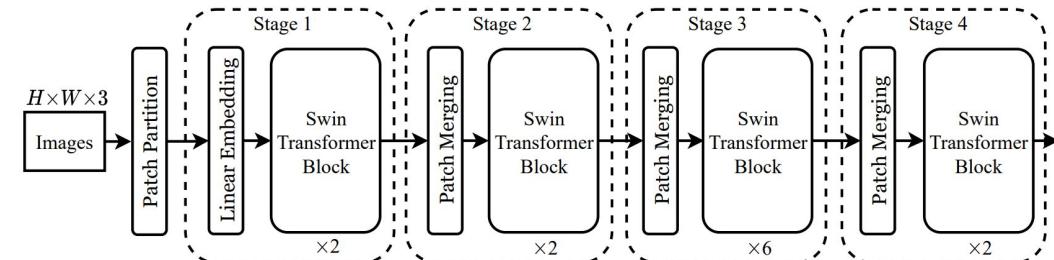
Figure 5: Performance versus cost for different architectures: Vision Transformers, ResNets, and hybrids. Vision Transformers generally outperform ResNets with the same computational budget. Hybrids improve upon pure Transformers for smaller model sizes, but the gap vanishes for larger models.

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ArXiv 2020
[Colab link](#) to an implementation of vision transformers

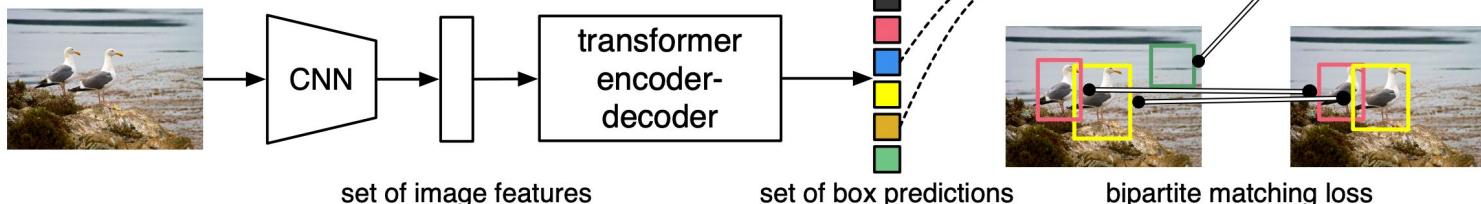
Vision Transformers



Fan et al, "Multiscale Vision Transformers", ICCV 2021



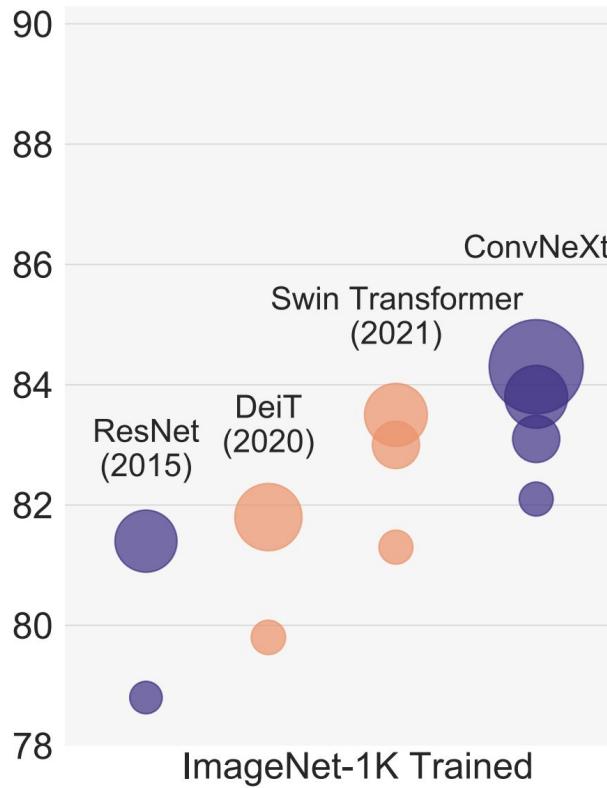
Liu et al, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows", CVPR 2021



Carion et al, "End-to-End Object Detection with Transformers", ECCV 2020

ConvNets strike back!

ImageNet-1K Acc.



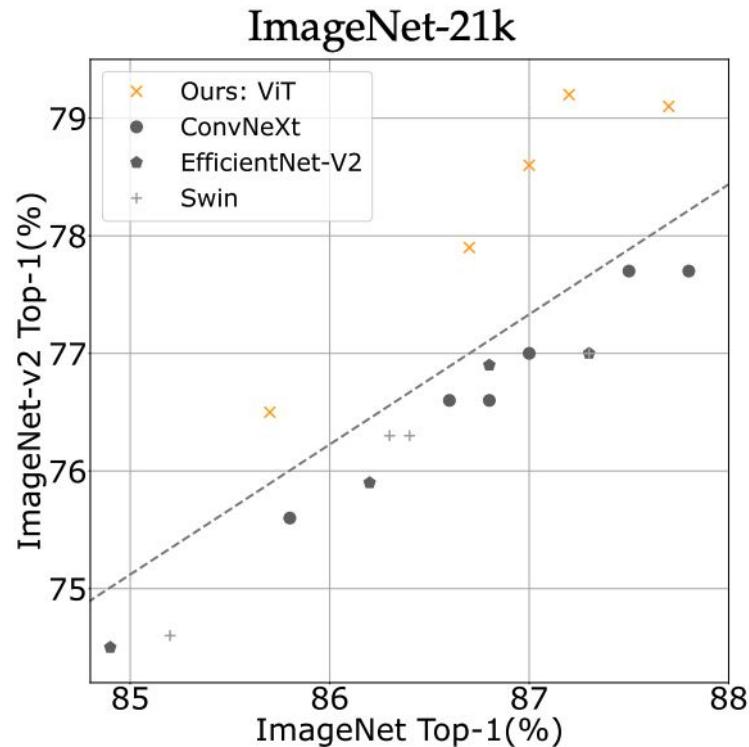
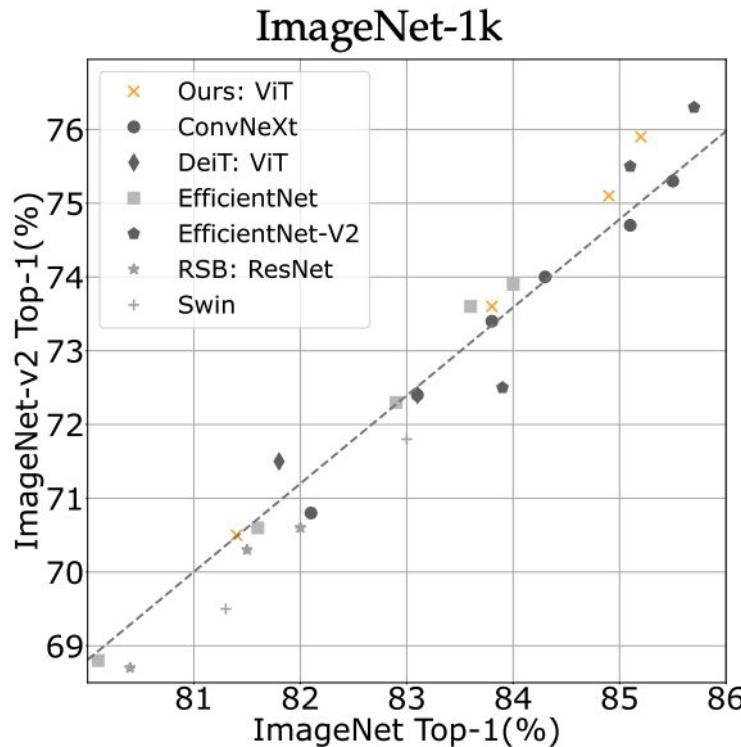
ImageNet-22K Pre-trained

Diameter
4 8 16 256 GFLOPs

A ConvNet for the 2020s. Liu et al. CVPR 2022

DeiT III: Revenge of the ViT

Hugo Touvron^{*,†} Matthieu Cord[†] Hervé Jégou^{*}



Summary

- Adding **attention** to RNNs allows them to "attend" to different parts of the input at every time step
- The **general attention layer** is a new type of layer that can be used to design new neural network architectures
- **Transformers** are a type of layer that uses **self-attention** and layer norm.
 - o It is highly **scalable** and highly **parallelizable**
 - o **Faster** training, **larger** models, **better** performance across vision and language tasks
 - o They are quickly replacing RNNs, LSTMs, and may(?) even replace convolutions.

Next time: Video Understanding

Lecture 12: Video Understanding

Administrative

- Project milestone due May 7th Saturday 11:59pm PT
- Check Ed and course website for requirements

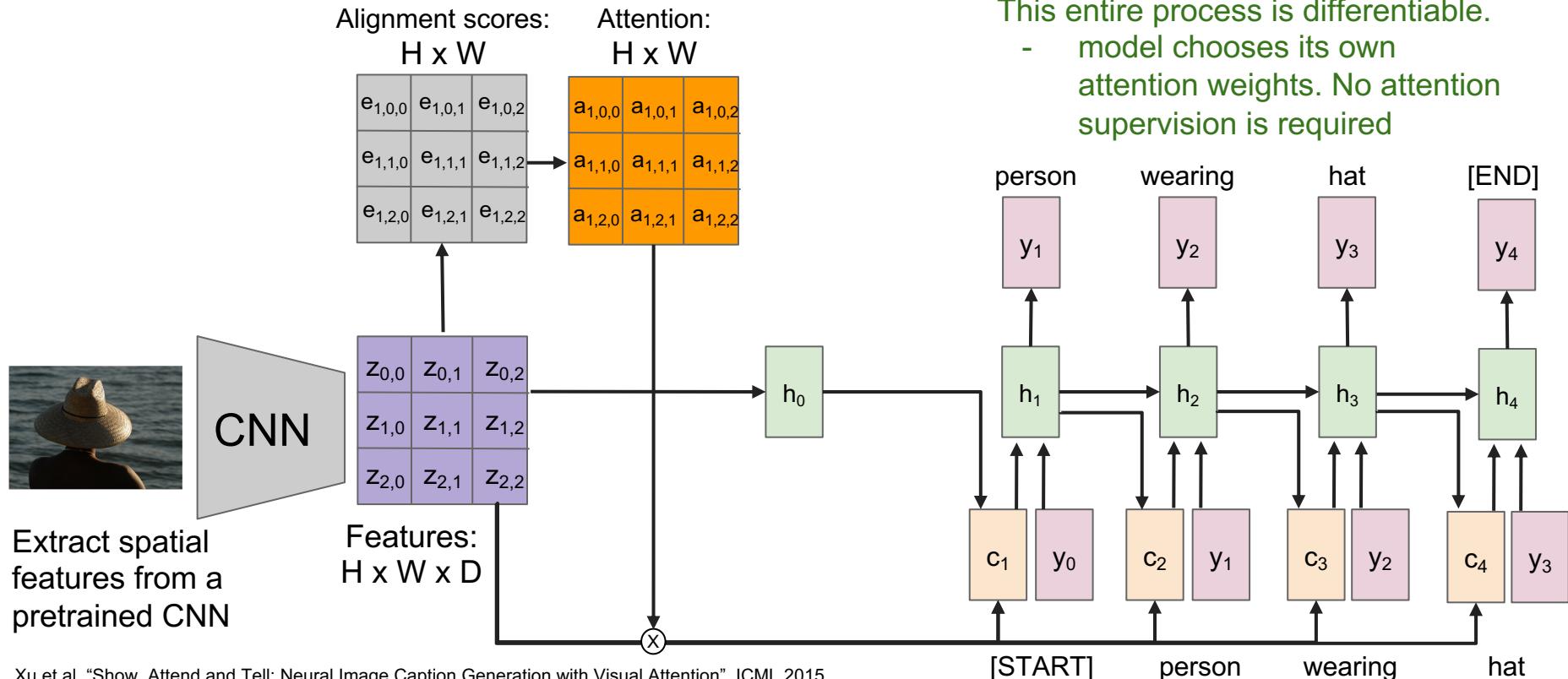
Administrative

- In-Class Midterm next Tuesday May 6th 1:30-3:00pm PT
Check Ed for details!
- Check sample midterm and solutions
Midterm review session tomorrow!
- 20% of your grade

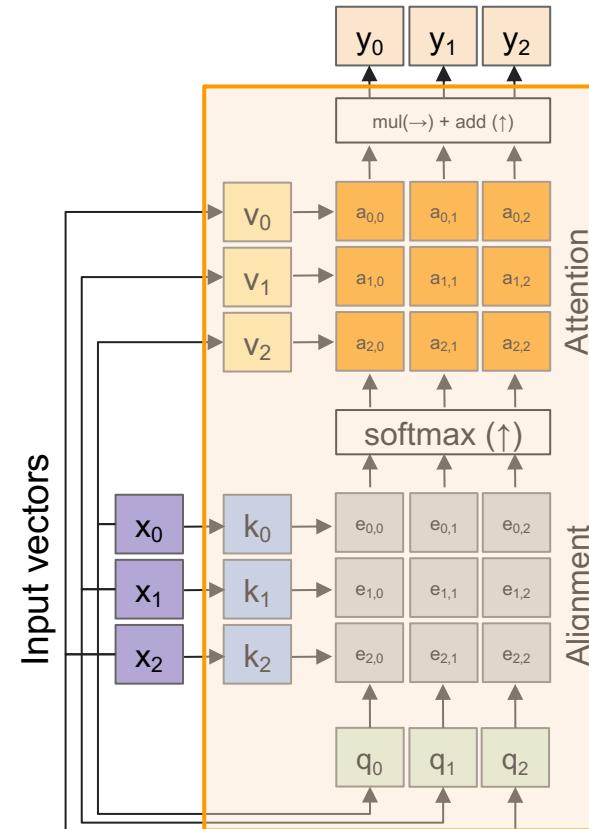
Administrative

- Assignment 3 will be released on May 6th after the midterm

Last time: Image Captioning with RNNs and Attention



Last time: Self-Attention



Outputs:

context vectors: \mathbf{y} (shape: D_y)

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$

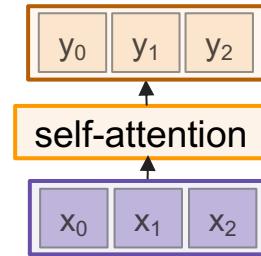
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$

Query vectors: $\mathbf{q} = \mathbf{x}\mathbf{W}_q$

Alignment: $e_{i,j} = \mathbf{q}_j \cdot \mathbf{k}_i / \sqrt{D}$

Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

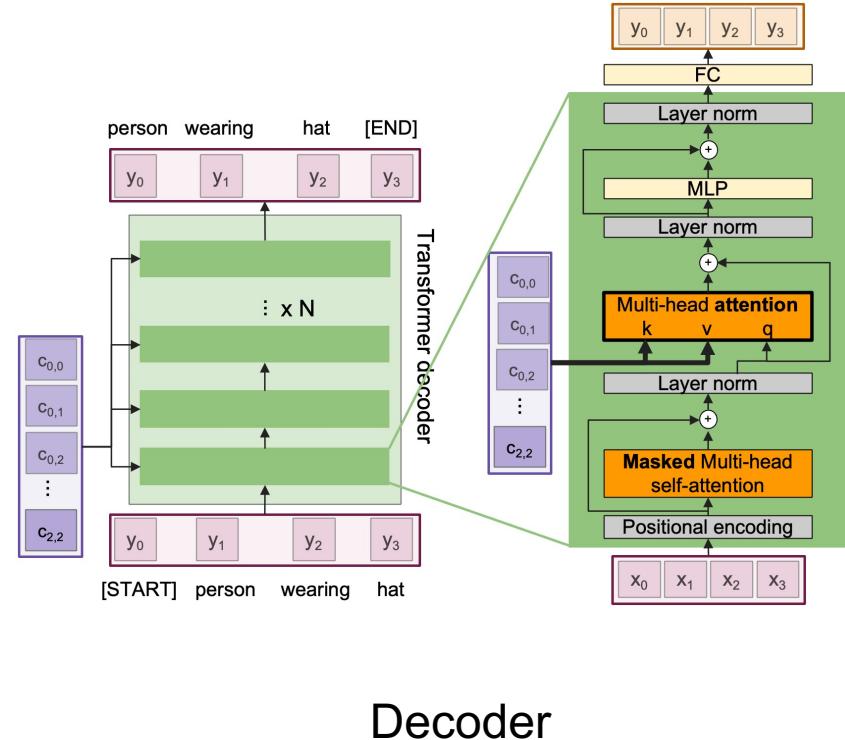
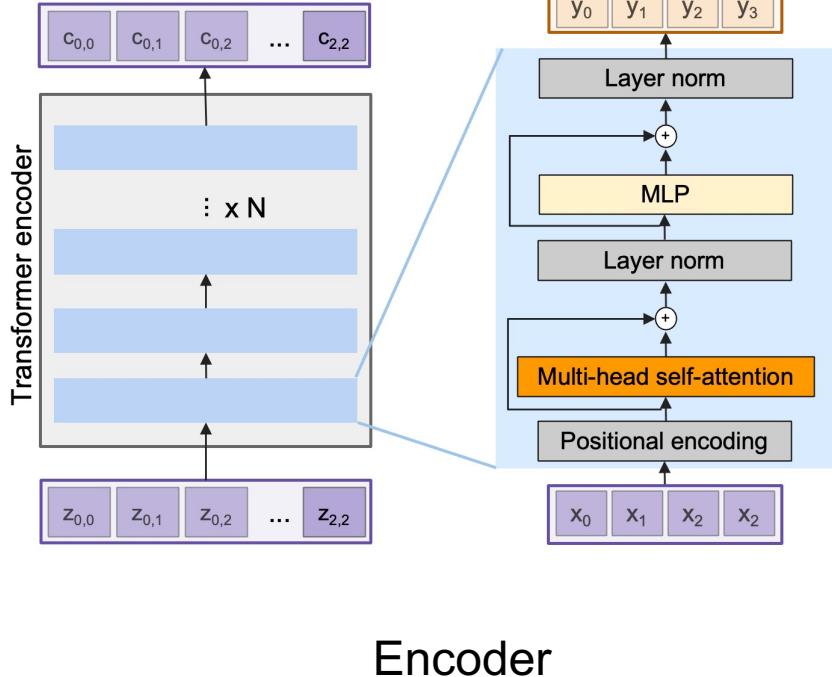
Output: $\mathbf{y}_j = \sum_i a_{i,j} \mathbf{v}_i$



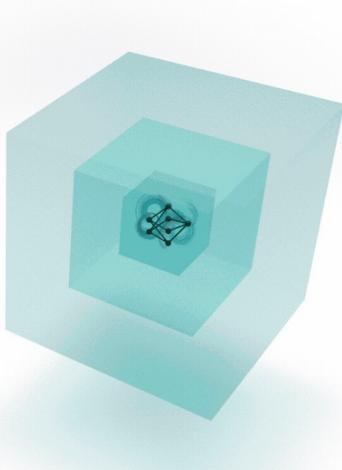
Inputs:

Input vectors: \mathbf{x} (shape: $N \times D$)

Last time: Transformer



Last time: Large foundation models



OPT-175B: 175 billion-parameter language model

Open sourced:

<https://github.com/facebookresearch/metaseq/tree/main/projects/OPT>

OPT: Open Pre-trained Transformer Language Models, Zhang et al. 2022

<https://ai.facebook.com/blog/democratizing-access-to-large-scale-language-models-with-opt-175b/>

Recall: (2D) Image classification



(assume given a set of possible labels)
{dog, cat, truck, plane, ...}



cat

This image by [Nikita](#) is
licensed under [CC-BY 2.0](#)

Recall: (2D) Detection and Segmentation

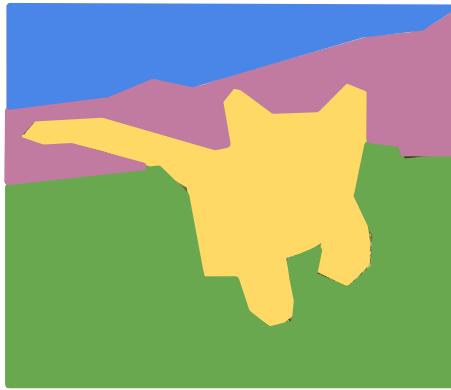
Classification



CAT

No spatial extent

Semantic Segmentation



GRASS, CAT,
TREE, SKY

No objects, just pixels

Object Detection



DOG, DOG, CAT

Multiple Objects

Instance Segmentation



DOG, DOG, CAT

[This image](#) is CC0 public domain

Living room

Dog

Baby



Today: **Video** = 2D + Time

A video is a **sequence** of images

4D tensor: $T \times 3 \times H \times W$
(or $3 \times T \times H \times W$)



[This image](#) is CC0 public domain

Example task: Video Classification



Input video:
 $T \times 3 \times H \times W$

Swimming
Running
Jumping
Eating
Standing

Example task: Video Classification



Images: Recognize **objects**



Dog
Cat
Fish
Truck



Videos: Recognize **actions**



Swimming
Running
Jumping
Eating
Standing

[Running video](#) is in the [public domain](#)

Problem: Videos are big!

Videos are ~30 frames per second (fps)



Size of uncompressed video
(3 bytes per pixel):

SD (640 x 480): **~1.5 GB per minute**
HD (1920 x 1080): **~10 GB per minute**

Input video:

$T \times 3 \times H \times W$

Slide credit: Justin Johnson

Problem: Videos are big!

Videos are ~30 frames per second (fps)



Input video:
 $T \times 3 \times H \times W$

Size of uncompressed video
(3 bytes per pixel):

SD (640 x 480): **~1.5 GB per minute**
HD (1920 x 1080): **~10 GB per minute**

Solution: Train on short **clips**:
low fps and low spatial resolution
e.g. $T = 16$, $H=W=112$
(3.2 seconds at 5 fps, 588 KB)

Slide credit: Justin Johnson

Training on Clips

Raw video: Long, high FPS



Slide credit: Justin Johnson

Training on Clips

Raw video: Long, high FPS



Training: Train model to classify short **clips** with low FPS



Slide credit: Justin Johnson

Training on Clips

Raw video: Long, high FPS



Training: Train model to classify short **clips** with low FPS



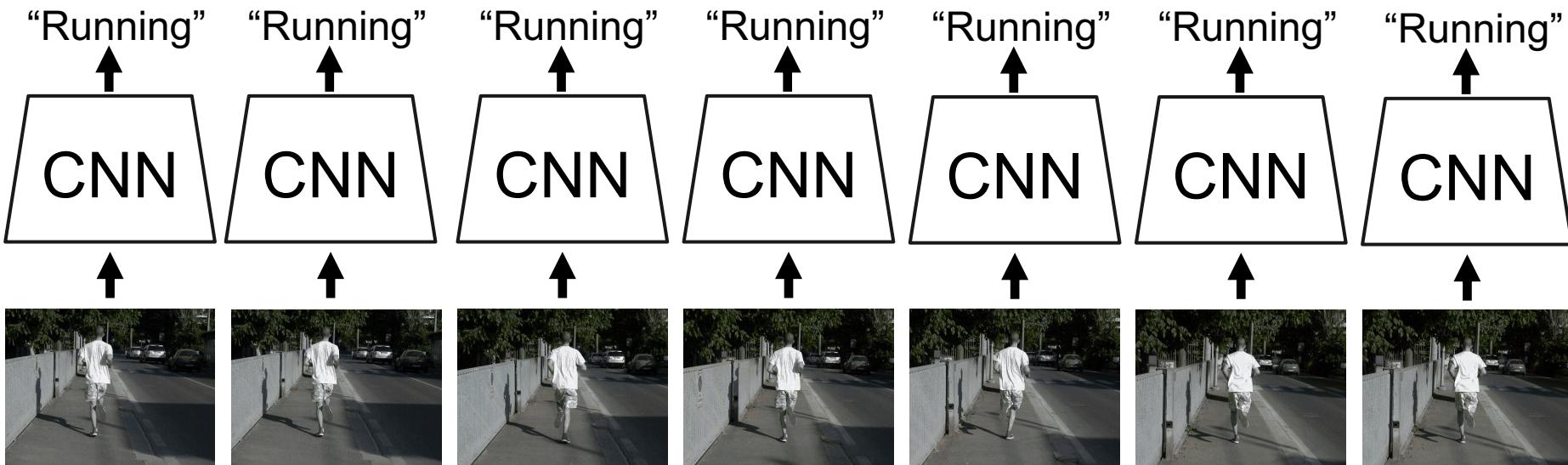
Testing: Run model on different clips, average predictions



Slide credit: Justin Johnson

Video Classification: Single-Frame CNN

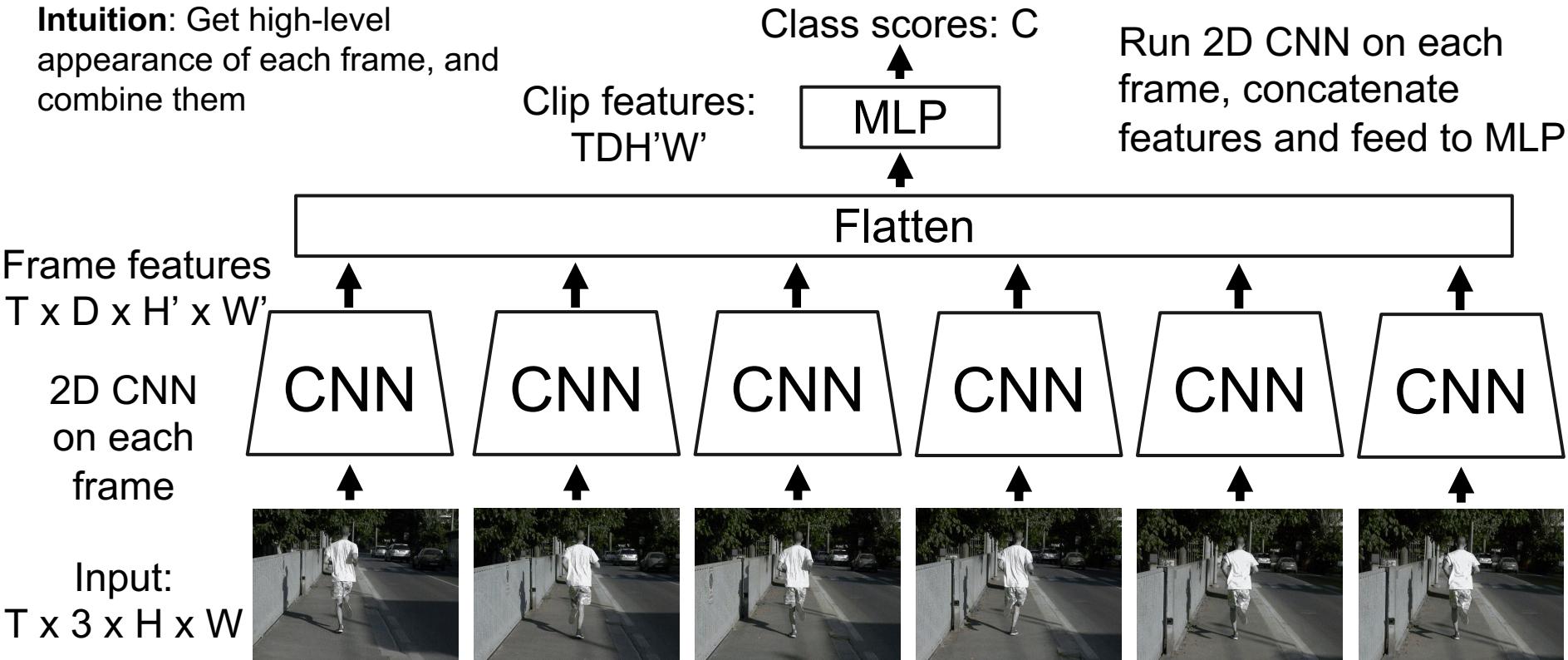
Simple idea: train normal 2D CNN to classify video frames independently!
(Average predicted probs at test-time)
Often a **very** strong baseline for video classification



Slide credit: Justin Johnson

Video Classification: Late Fusion (with FC layers)

Intuition: Get high-level appearance of each frame, and combine them

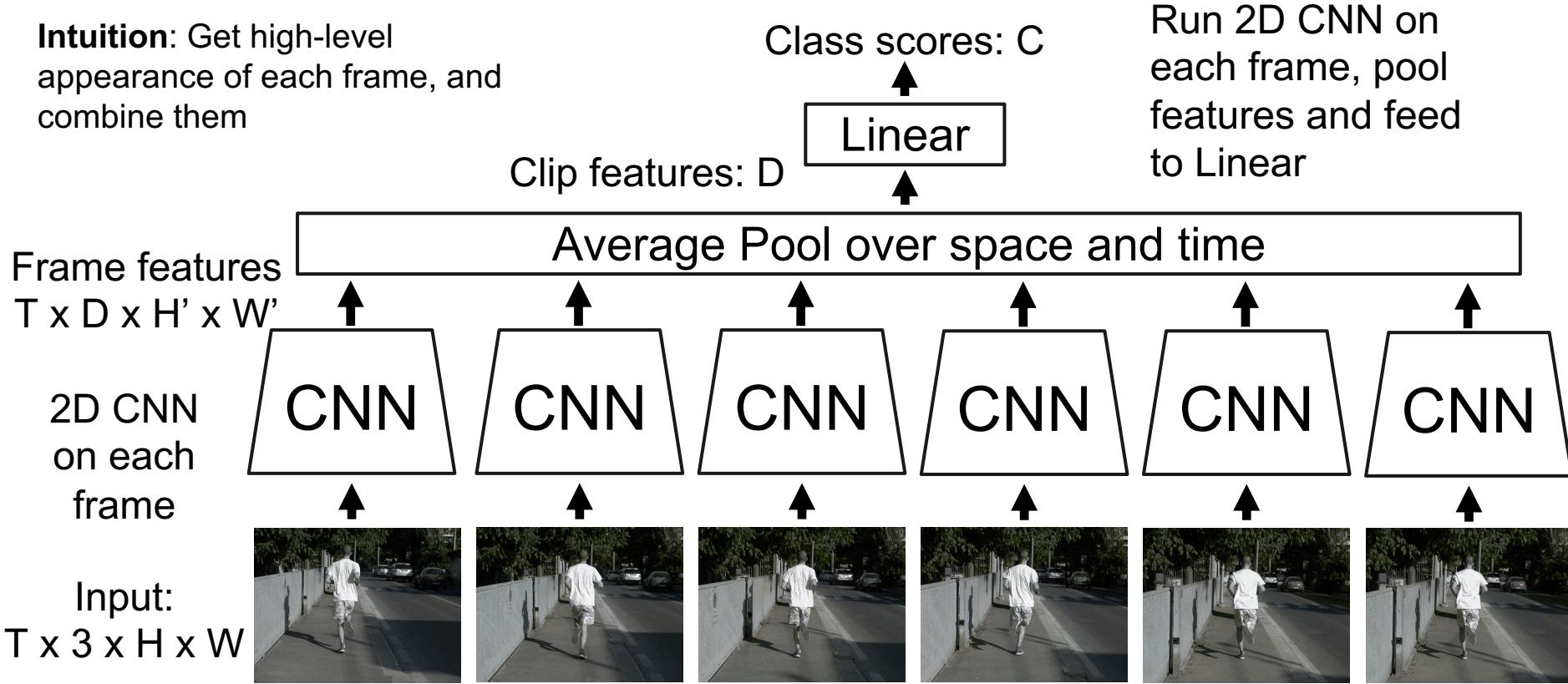


Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Slide credit: Justin Johnson

Video Classification: Late Fusion (with pooling)

Intuition: Get high-level appearance of each frame, and combine them



Slide credit: Justin Johnson

Video Classification: Late Fusion (with pooling)

Intuition: Get high-level appearance of each frame, and combine them

Problem: Hard to compare low-level motion between frames

Class scores: C

Linear

Run 2D CNN on each frame, pool features and feed to Linear

Frame features

$T \times D \times H' \times W'$

2D CNN
on each
frame

Average Pool over space and time

Input:
 $T \times 3 \times H \times W$



Slide credit: Justin Johnson

Video Classification: Early Fusion

Intuition: Compare frames with very first conv layer, after that normal 2D CNN

First 2D convolution collapses all temporal information:
Input: $3T \times H \times W$
Output: $D \times H \times W$

Reshape:
 $3T \times H \times W$

Input:
 $T \times 3 \times H \times W$



Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Slide credit: Justin Johnson

Video Classification: Early Fusion

Intuition: Compare frames with very first conv layer, after that normal 2D CNN

Problem: One layer of temporal processing may not be enough!

First 2D convolution collapses all temporal information:

Input: $3T \times H \times W$

Output: $D \times H \times W$

Reshape:
 $3T \times H \times W$

Input:
 $T \times 3 \times H \times W$



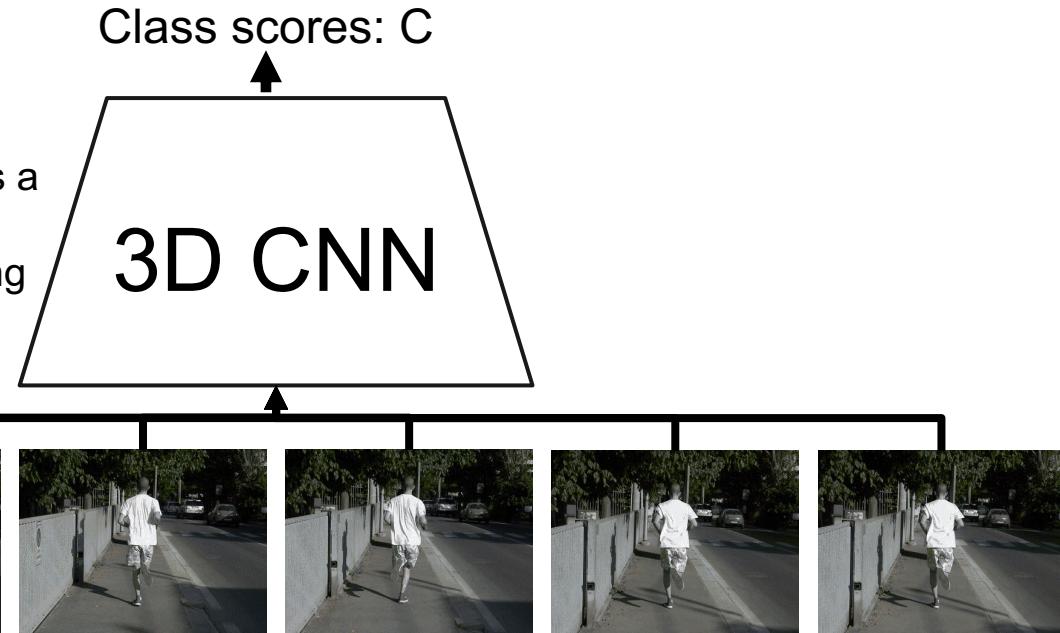
Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Slide credit: Justin Johnson

Video Classification: 3D CNN

Intuition: Use 3D versions of convolution and pooling to slowly fuse temporal information over the course of the network

Each layer in the network is a 4D tensor: $D \times T \times H \times W$
Use 3D conv and 3D pooling operations



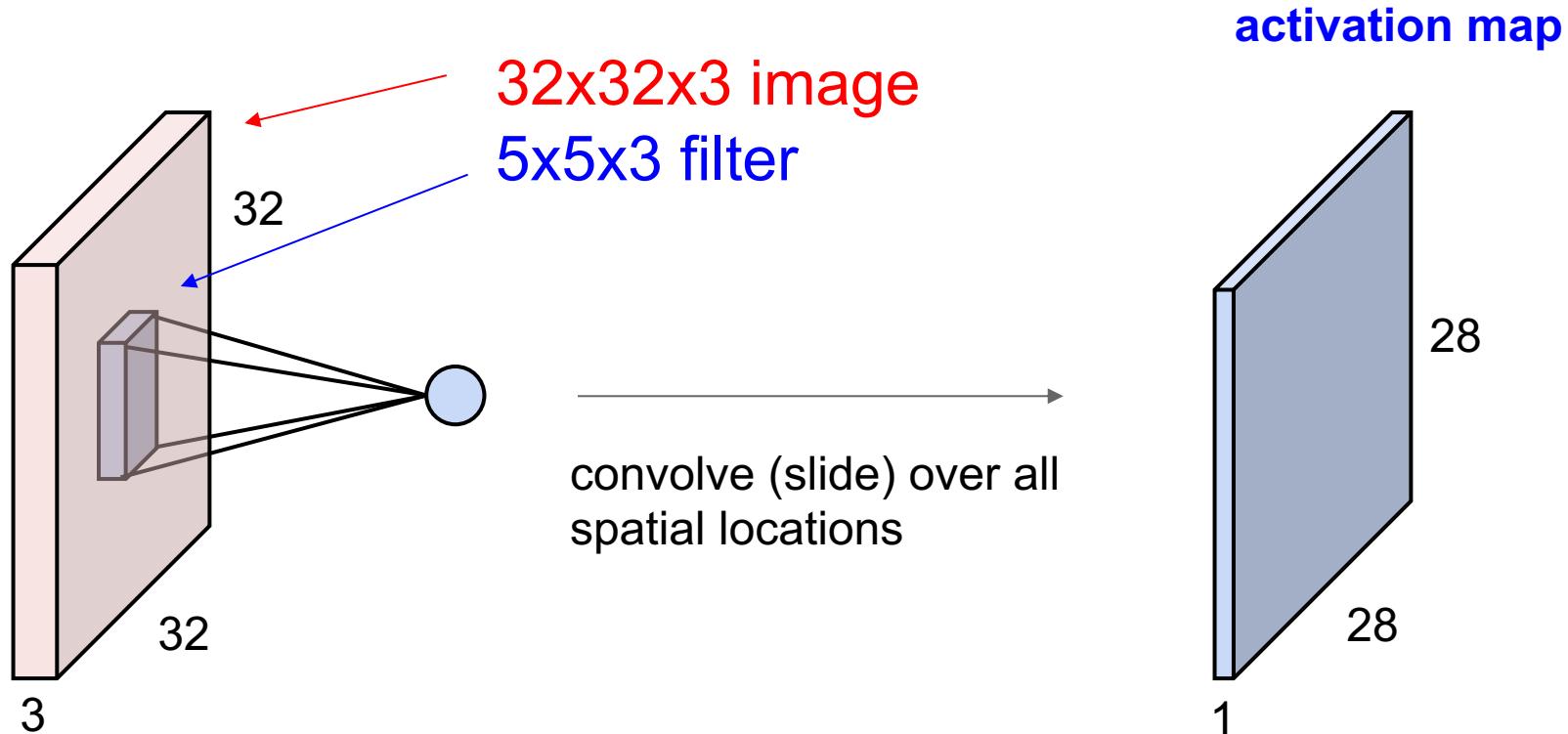
Input:
 $3 \times T \times H \times W$



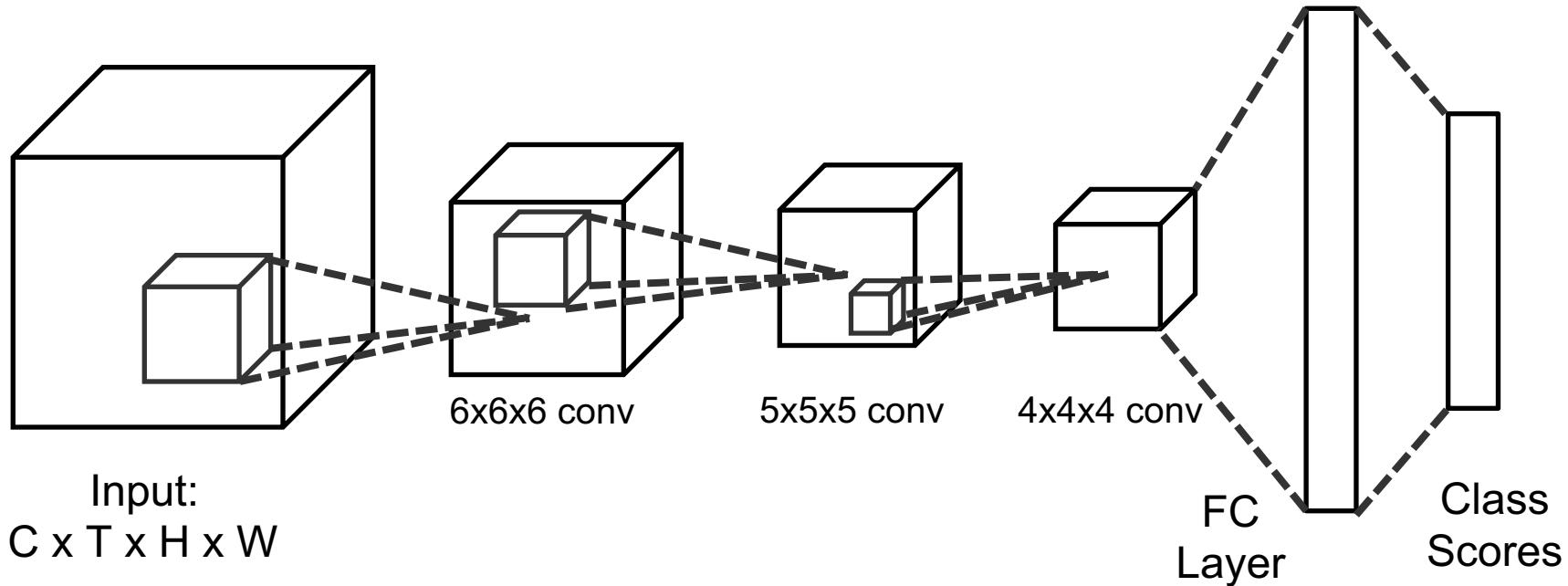
Ji et al, "3D Convolutional Neural Networks for Human Action Recognition", TPAMI 2010 ; Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014

Slide credit: Justin Johnson

Convolution Layer



3D Convolution



Early Fusion vs Late Fusion vs 3D CNN

Late
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3

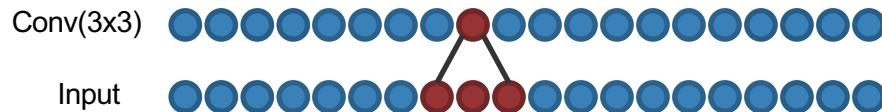
(Small example
architectures, in
practice much
bigger)

Slide credit: Justin Johnson

Early Fusion vs Late Fusion vs 3D CNN

Late
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3



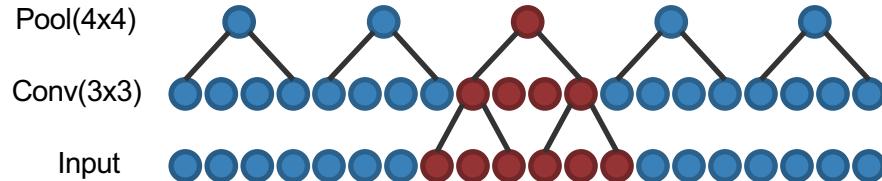
(Small example
architectures, in
practice much
bigger)

Slide credit: Justin Johnson

Early Fusion vs Late Fusion vs 3D CNN

Late
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6



(Small example
architectures, in
practice much
bigger)

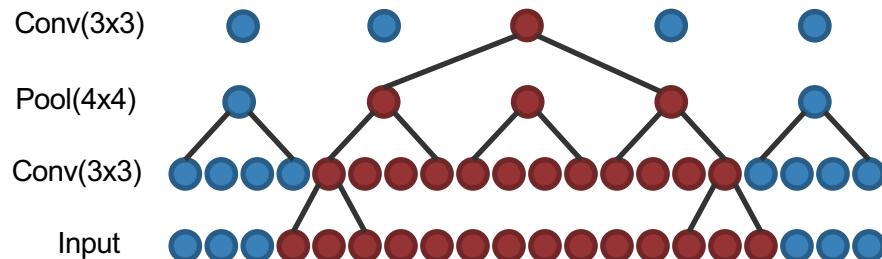
Slide credit: Justin Johnson

Early Fusion vs Late Fusion vs 3D CNN

Late
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14

Build slowly in space



(Small example
architectures, in
practice much
bigger)

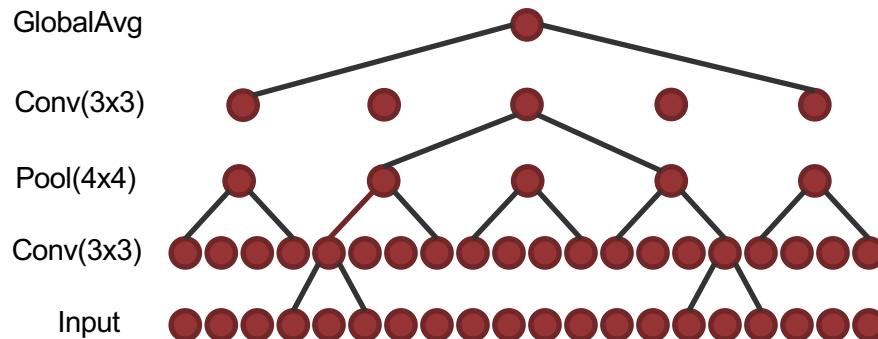
Slide credit: Justin Johnson

Early Fusion vs Late Fusion vs 3D CNN

Late
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14
GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64

Build slowly in space,
All-at-once in time at end



(Small example
architectures, in
practice much
bigger)

Slide credit: Justin Johnson

Early Fusion vs Late Fusion vs 3D CNN

Late
Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14
GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3*20->12)	12 x 64 x 64	20 x 3 x 3
Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6
Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14
GlobalAvgPool	24 x 1 x 1	20 x 64 x 64

Build slowly in space,
All-at-once in time at end

Early
Fusion

Build slowly in space,
All-at-once in time at start

(Small example
architectures, in
practice much
bigger)

Slide credit: Justin Johnson

Early Fusion vs Late Fusion vs 3D CNN

Late Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
	12 x 20 x 64 x 64	1 x 3 x 3
	12 x 20 x 16 x 16	1 x 6 x 6
	24 x 20 x 16 x 16	1 x 14 x 14
	24 x 1 x 1 x 1	20 x 64 x 64
Input	3 x 20 x 64 x 64	
	12 x 64 x 64	20 x 3 x 3
	12 x 16 x 16	20 x 6 x 6
	24 x 16 x 16	20 x 14 x 14
	24 x 1 x 1	20 x 64 x 64
Input	3 x 20 x 64 x 64	
	12 x 20 x 64 x 64	3 x 3 x 3
	12 x 5 x 16 x 16	6 x 6 x 6
	24 x 5 x 16 x 16	14 x 14 x 14
	24 x 1 x 1	20 x 64 x 64

Build slowly in space,
All-at-once in time at end

Early Fusion

Build slowly in space,
All-at-once in time at start

3D CNN

Build slowly in space,
Build slowly in time
"Slow Fusion"

(Small example
architectures, in
practice much
bigger)

Slide credit: Justin Johnson

Early Fusion vs Late Fusion vs 3D CNN

What is the difference?

Late Fusion

Layer	Size (C x T x H x W)	Receptive Field (T x H x W)
Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3->12)	12 x 20 x 64 x 64	1 x 3 x 3
Pool2D(4x4)	12 x 20 x 16 x 16	1 x 6 x 6
Conv2D(3x3, 12->24)	24 x 20 x 16 x 16	1 x 14 x 14
GlobalAvgPool	24 x 1 x 1 x 1	20 x 64 x 64

Build slowly in space,
All-at-once in time at end

Early Fusion

Input	3 x 20 x 64 x 64	
Conv2D(3x3, 3*20->12)	12 x 64 x 64	20 x 3 x 3
Pool2D(4x4)	12 x 16 x 16	20 x 6 x 6
Conv2D(3x3, 12->24)	24 x 16 x 16	20 x 14 x 14
GlobalAvgPool	24 x 1 x 1	20 x 64 x 64

Build slowly in space,
All-at-once in time at start

3D CNN

Input	3 x 20 x 64 x 64	
Conv3D(3x3x3, 3->12)	12 x 20 x 64 x 64	3 x 3 x 3
Pool3D(4x4x4)	12 x 5 x 16 x 16	6 x 6 x 6
Conv3D(3x3x3, 12->24)	24 x 5 x 16 x 16	14 x 14 x 14
GlobalAvgPool	24 x 1 x 1	20 x 64 x 64

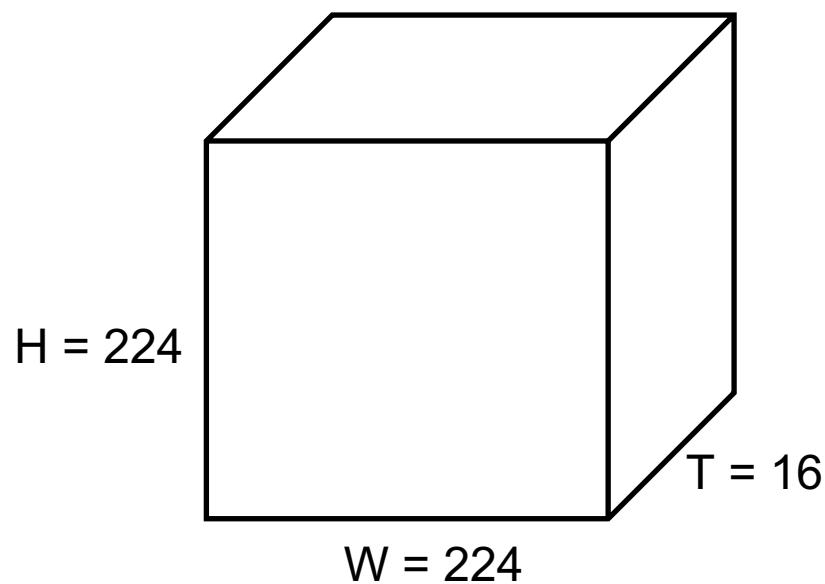
Build slowly in space,
Build slowly in time
"Slow Fusion"

(Small example
architectures, in
practice much
bigger)

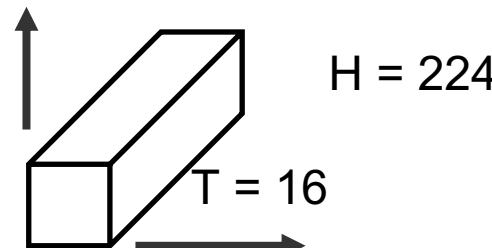
Slide credit: Justin Johnson

2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: $C_{in} \times T \times H \times W$
(3D grid with C_{in} -dim
feat at each point)

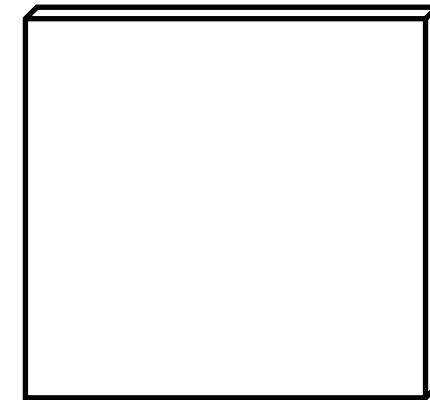


Weight:
 $C_{out} \times C_{in} \times T \times 3 \times 3$
Slide over x and y



C_{out} different filters

Output:
 $C_{out} \times H \times W$
2D grid with C_{out} -dim
feat at each point



$W = 224$

Slide credit: Justin Johnson

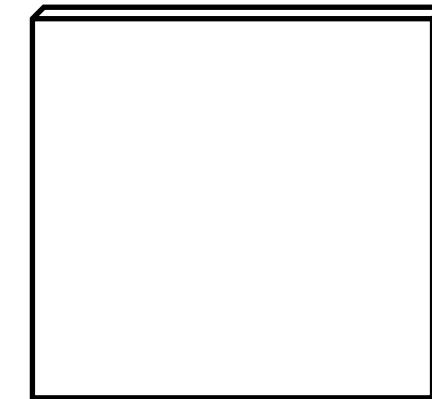
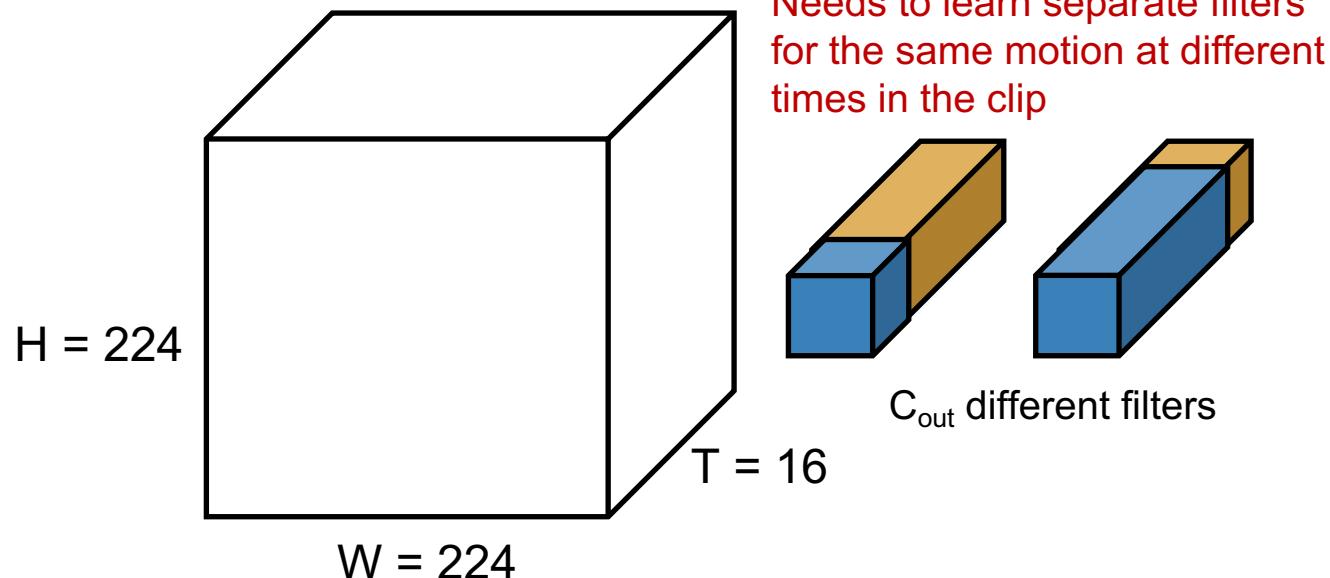
2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: $C_{in} \times T \times H \times W$
(3D grid with C_{in} -dim
feat at each point)

Weight:
 $C_{out} \times C_{in} \times T \times 3 \times 3$
Slide over x and y

No temporal shift-invariance!
Needs to learn separate filters
for the same motion at different
times in the clip

Output:
 $C_{out} \times H \times W$
2D grid with C_{out} -dim
feat at each point



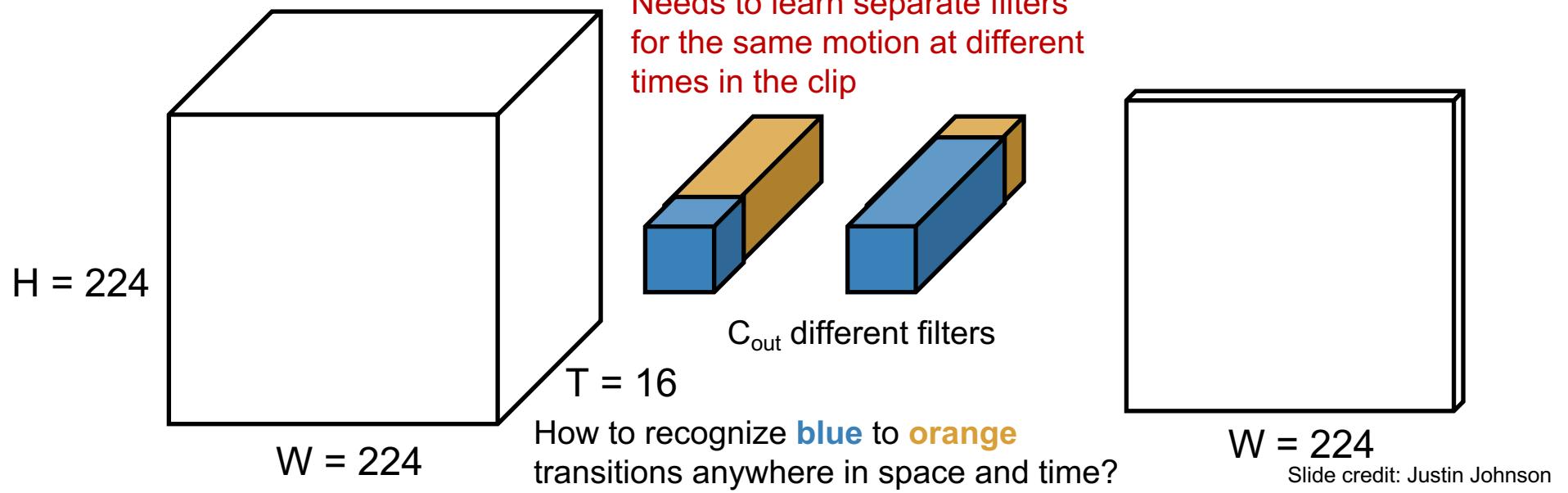
Slide credit: Justin Johnson

2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: $C_{in} \times T \times H \times W$
(3D grid with C_{in} -dim
feat at each point)

Weight:
 $C_{out} \times C_{in} \times T \times 3 \times 3$
Slide over x and y

Output:
 $C_{out} \times H \times W$
2D grid with C_{out} -dim
feat at each point

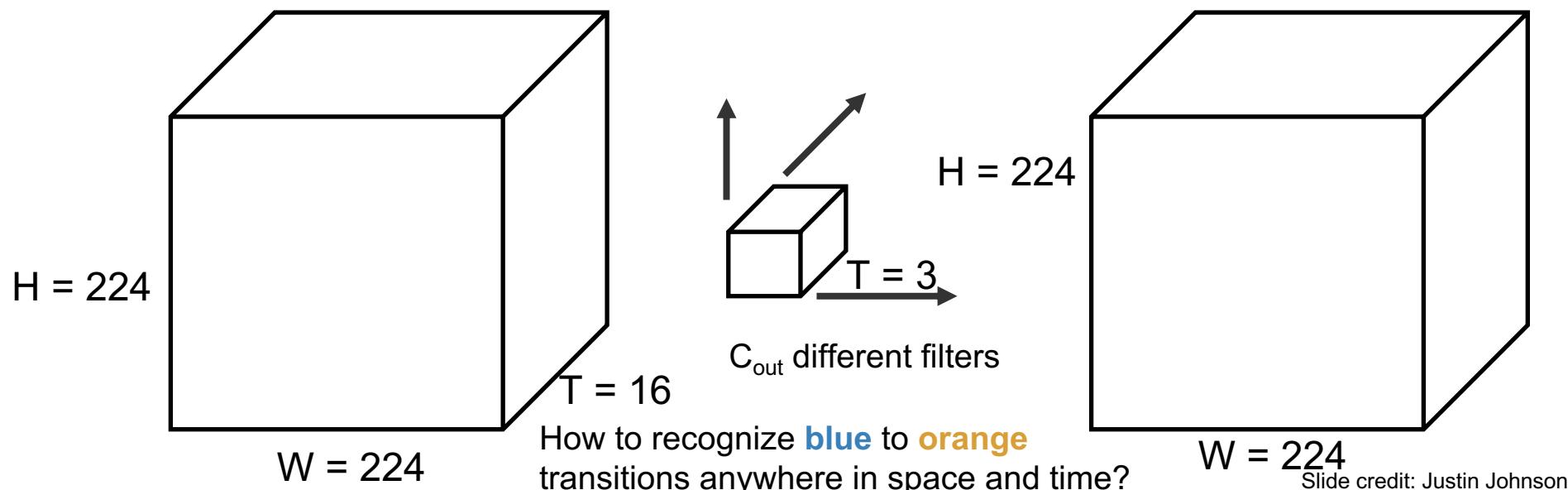


2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: $C_{in} \times T \times H \times W$
(3D grid with C_{in} -dim
feat at each point)

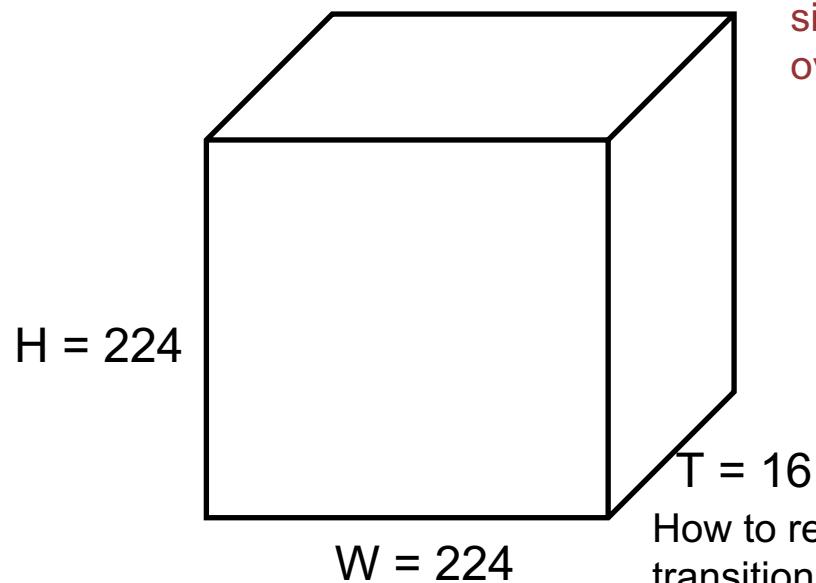
Weight:
 $C_{out} \times C_{in} \times 3 \times 3 \times 3$
Slide over x and y

Output:
 $C_{out} \times T \times H \times W$
3D grid with C_{out} -dim
feat at each point



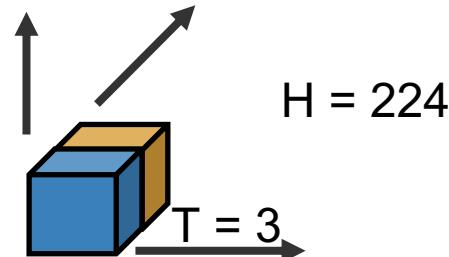
2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: $C_{in} \times T \times H \times W$
(3D grid with C_{in} -dim
feat at each point)



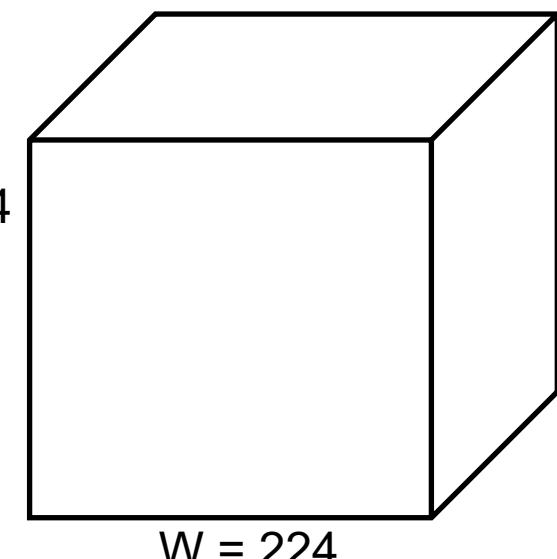
Weight:
 $C_{out} \times C_{in} \times 3 \times 3 \times 3$
Slide over x and y

Temporal shift-invariant
since each filter slides
over time!



How to recognize **blue** to **orange**
transitions anywhere in space and time?

Output:
 $C_{out} \times T \times H \times W$
3D grid with C_{out} -dim
feat at each point



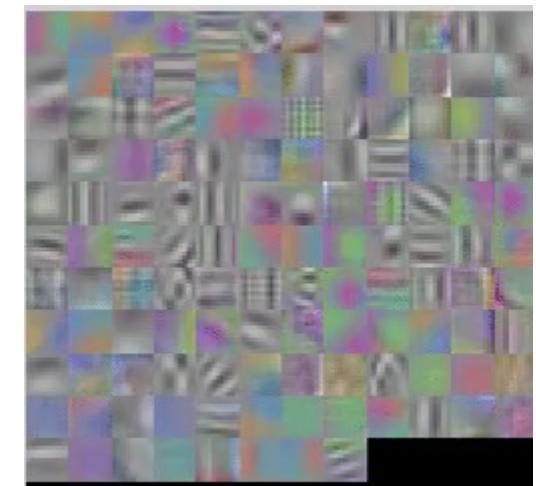
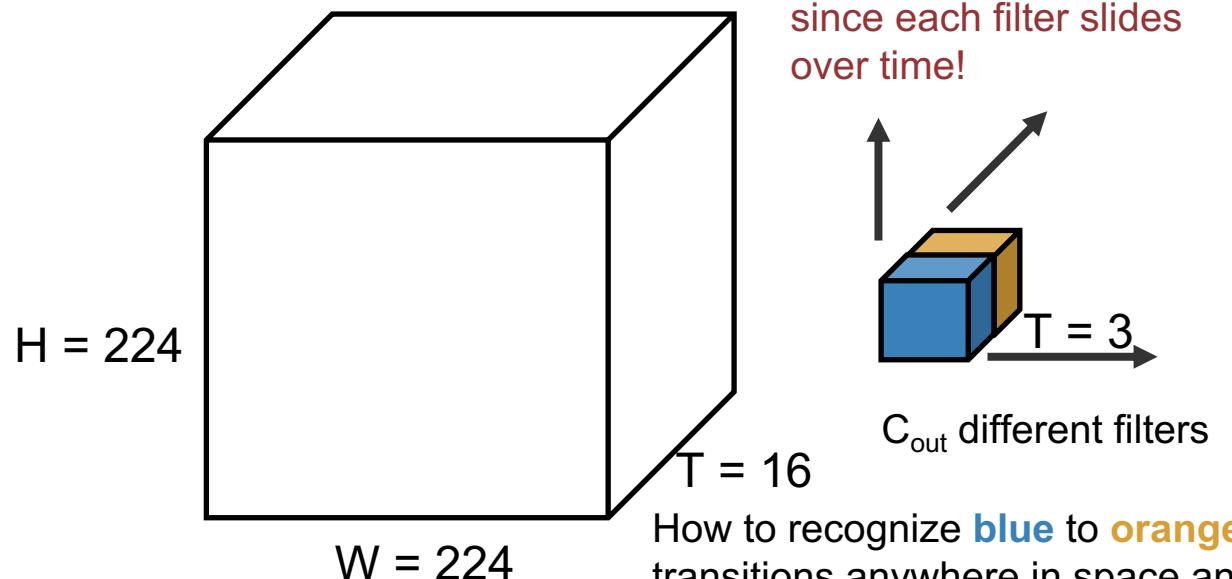
Slide credit: Justin Johnson

2D Conv (Early Fusion) vs 3D Conv (3D CNN)

Input: $C_{in} \times T \times H \times W$
(3D grid with C_{in} -dim
feat at each point)

Weight:
 $C_{out} \times C_{in} \times 3 \times 3 \times 3$
Slide over x and y

First-layer filters have shape
3 (RGB) x 4 (frames) x 5 x 5
(space)
Can visualize as video clips!



Slide credit: Justin Johnson

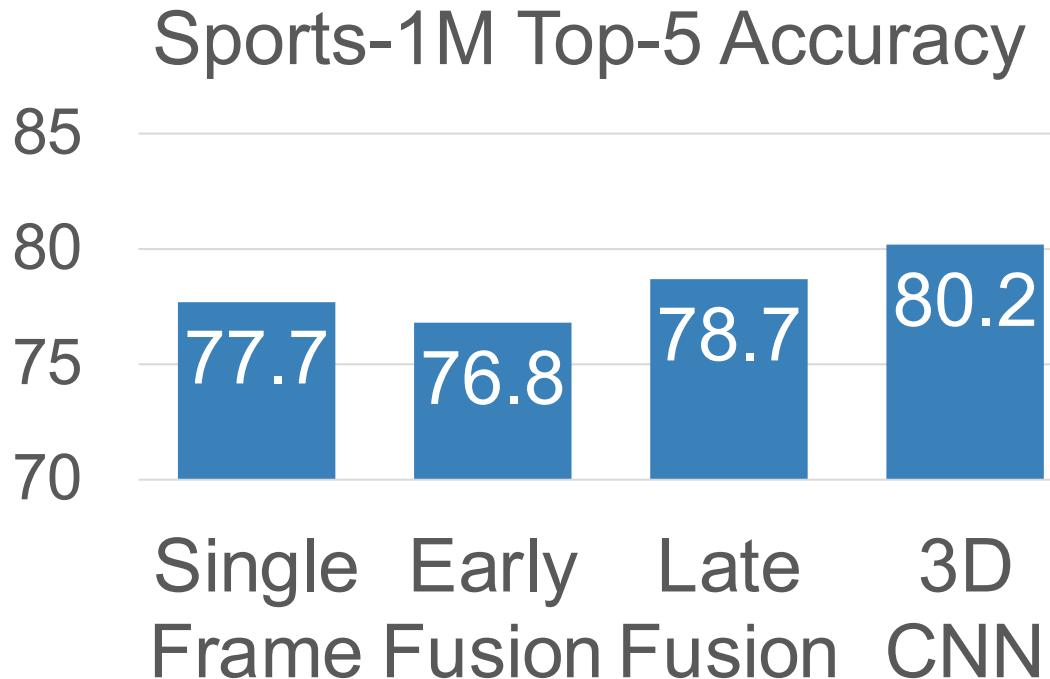
Example Video Dataset: Sports-1M



1 million YouTube videos
annotated with labels for 487
different types of sports

Ground Truth
Correct prediction
Incorrect prediction

Early Fusion vs Late Fusion vs 3D CNN



Single Frame
model works well
– always try this
first!

3D CNNs have
improved a lot
since 2014!

C3D: The VGG of 3D CNNs

3D CNN that uses all 3x3x3 conv
and 2x2x2 pooling
(except Pool1 which is 1x2x2)

Released model pretrained on
Sports-1M: Many people used this
as a video feature extractor

Layer	Size
Input	3 x 16 x 112 x 112
Conv1 (3x3x3)	64 x 16 x 112 x 112
Pool1 (1x2x2)	64 x 16 x 56 x 56
Conv2 (3x3x3)	128 x 16 x 56 x 56
Pool2 (2x2x2)	128 x 8 x 28 x 28
Conv3a (3x3x3)	256 x 8 x 28 x 28
Conv3b (3x3x3)	256 x 8 x 28 x 28
Pool3 (2x2x2)	256 x 4 x 14 x 14
Conv4a (3x3x3)	512 x 4 x 14 x 14
Conv4b (3x3x3)	512 x 4 x 14 x 14
Pool4 (2x2x2)	512 x 2 x 7 x 7
Conv5a (3x3x3)	512 x 2 x 7 x 7
Conv5b (3x3x3)	512 x 2 x 7 x 7
Pool5	512 x 1 x 3 x 3
FC6	4096
FC7	4096
FC8	C

Tran et al, "Learning Spatiotemporal Features with 3D Convolutional Networks", ICCV 2015

C3D: The VGG of 3D CNNs

3D CNN that uses all 3x3x3 conv
and 2x2x2 pooling
(except Pool1 which is 1x2x2)

Released model pretrained on
Sports-1M: Many people used this
as a video feature extractor

Problem: 3x3x3 conv is very
expensive!

AlexNet: 0.7 GFLOP

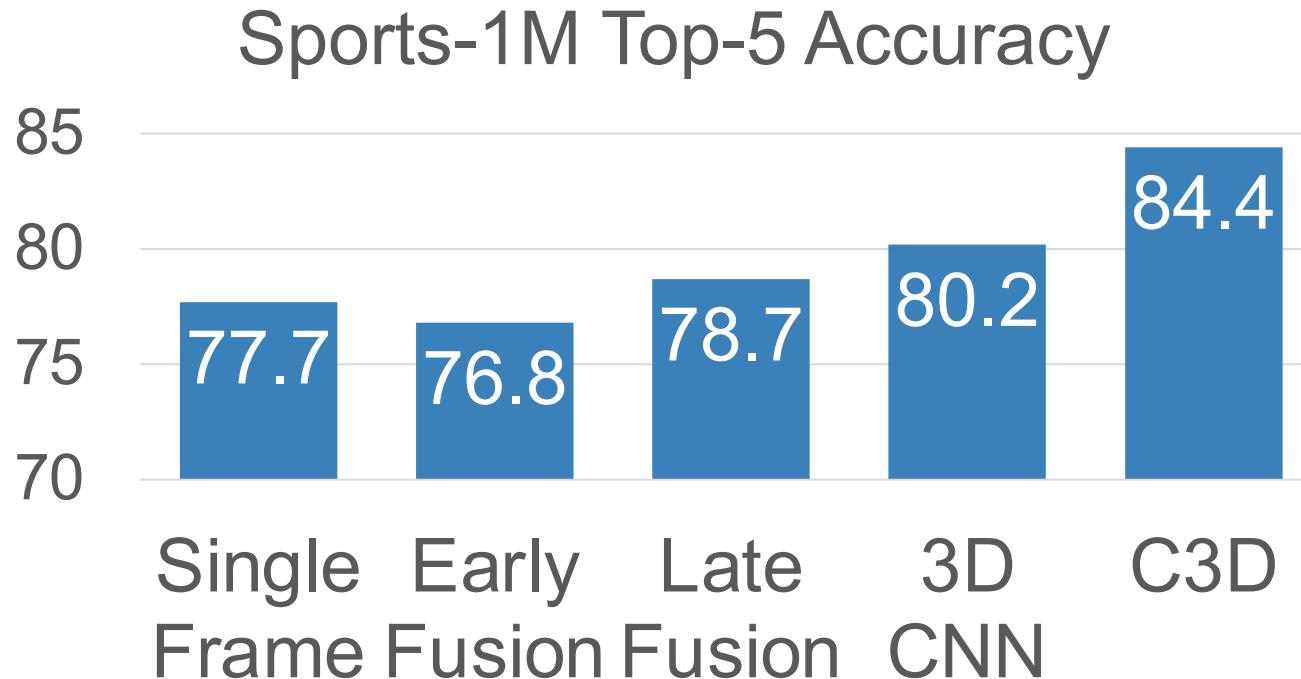
VGG-16: 13.6 GFLOP

C3D: **39.5 GFLOP (2.9x VGG!)**

Layer	Size	MFLOPs
Input	3 x 16 x 112 x 112	
Conv1 (3x3x3)	64 x 16 x 112 x 112	1.04
Pool1 (1x2x2)	64 x 16 x 56 x 56	
Conv2 (3x3x3)	128 x 16 x 56 x 56	11.10
Pool2 (2x2x2)	128 x 8 x 28 x 28	
Conv3a (3x3x3)	256 x 8 x 28 x 28	5.55
Conv3b (3x3x3)	256 x 8 x 28 x 28	11.10
Pool3 (2x2x2)	256 x 4 x 14 x 14	
Conv4a (3x3x3)	512 x 4 x 14 x 14	2.77
Conv4b (3x3x3)	512 x 4 x 14 x 14	5.55
Pool4 (2x2x2)	512 x 2 x 7 x 7	
Conv5a (3x3x3)	512 x 2 x 7 x 7	0.69
Conv5b (3x3x3)	512 x 2 x 7 x 7	0.69
Pool5	512 x 1 x 3 x 3	
FC6	4096	0.51
FC7	4096	0.45
FC8	C	0.05

Slide credit: Justin Johnson

Early Fusion vs Late Fusion vs 3D CNN

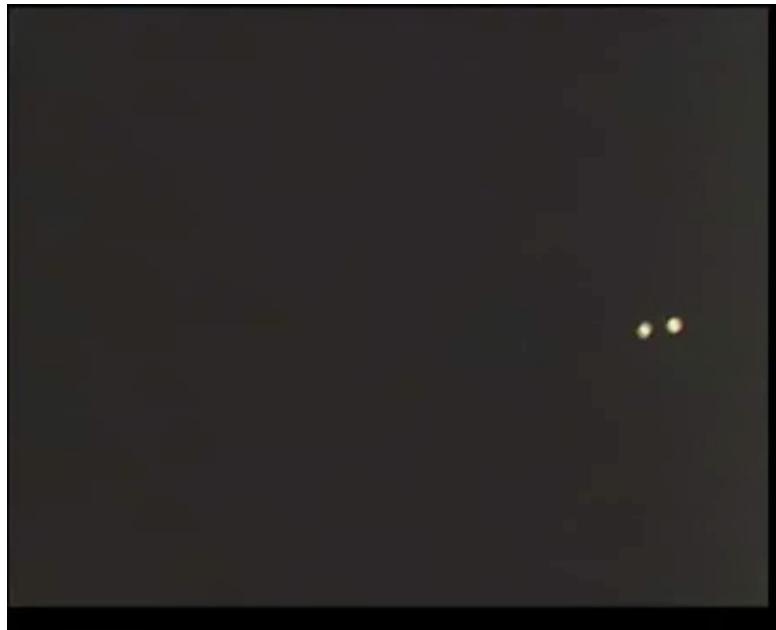


Karpathy et al, "Large-scale Video Classification with Convolutional Neural Networks", CVPR 2014
Tran et al, "Learning Spatiotemporal Features with 3D Convolutional Networks", ICCV 2015

Slide credit: Justin Johnson

Recognizing Actions from Motion

We can easily recognize actions using only **motion information**



Johansson, "Visual perception of biological motion and a model for its analysis." Perception & Psychophysics. 14(2):201-211. 1973.

Slide credit: Justin Johnson

Measuring Motion: Optical Flow

Image at frame t

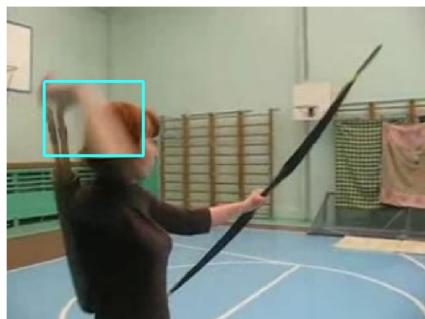


Image at frame t+1

Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

Slide credit: Justin Johnson

Measuring Motion: Optical Flow

Image at frame t

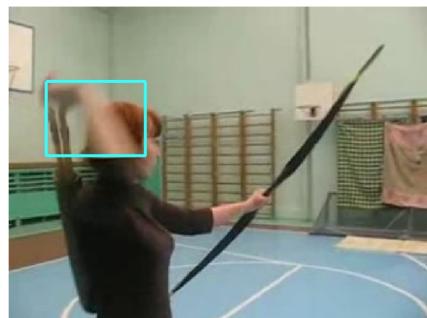
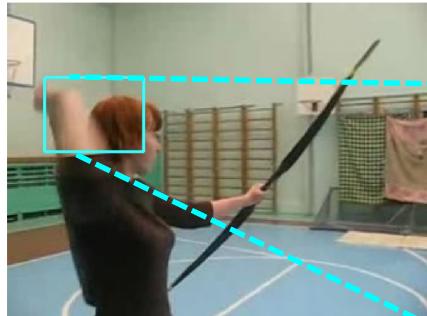
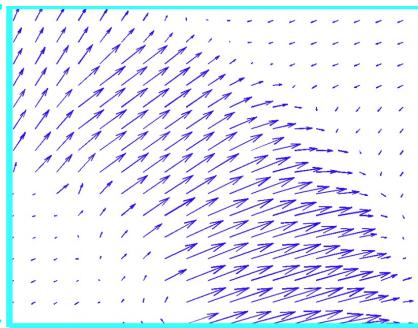


Image at frame $t+1$

Optical flow gives a displacement field F between images I_t and I_{t+1}



Tells where each pixel will move in the next frame:
 $F(x, y) = (dx, dy)$
 $I_{t+1}(x+dx, y+dy) = I_t(x, y)$

Measuring Motion: Optical Flow

Image at frame t

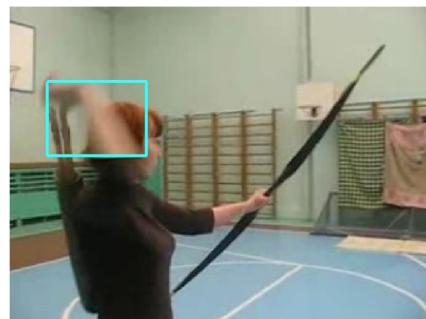
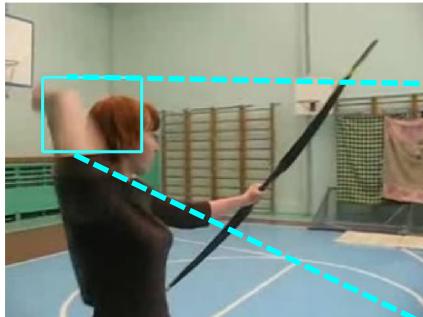
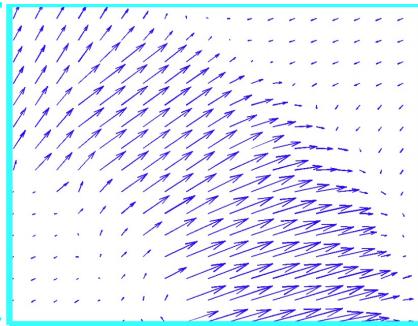


Image at frame $t+1$

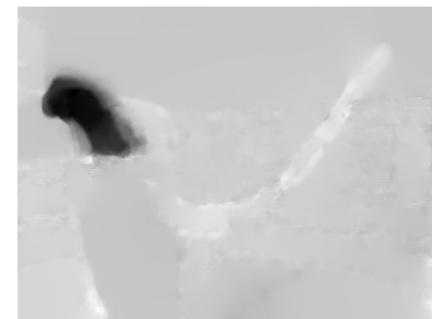
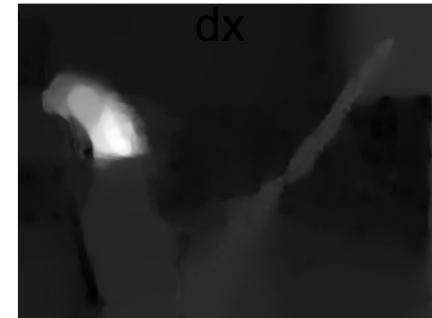
Optical flow gives a displacement field F between images I_t and I_{t+1}



Tells where each pixel will move in the next frame:
 $F(x, y) = (dx, dy)$
 $I_{t+1}(x+dx, y+dy) = I_t(x, y)$

Optical Flow highlights local motion

Horizontal flow



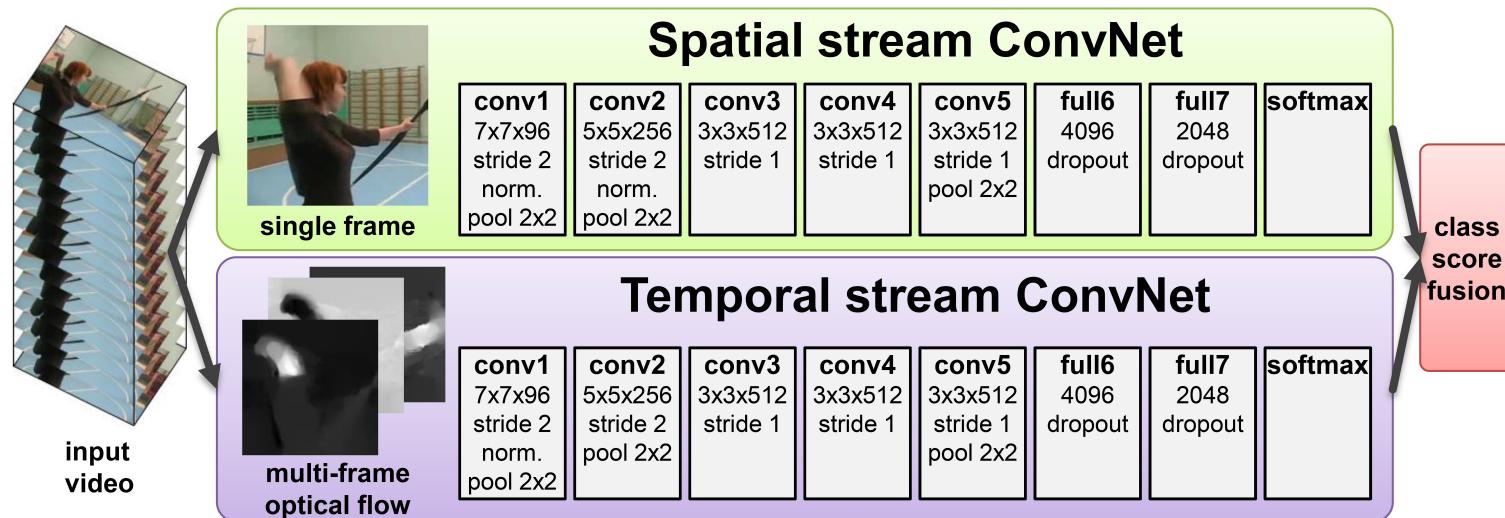
Vertical Flow dy

Slide credit: Justin Johnson

Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

Separating Motion and Appearance: Two-Stream Networks

Input: Single Image
 $3 \times H \times W$



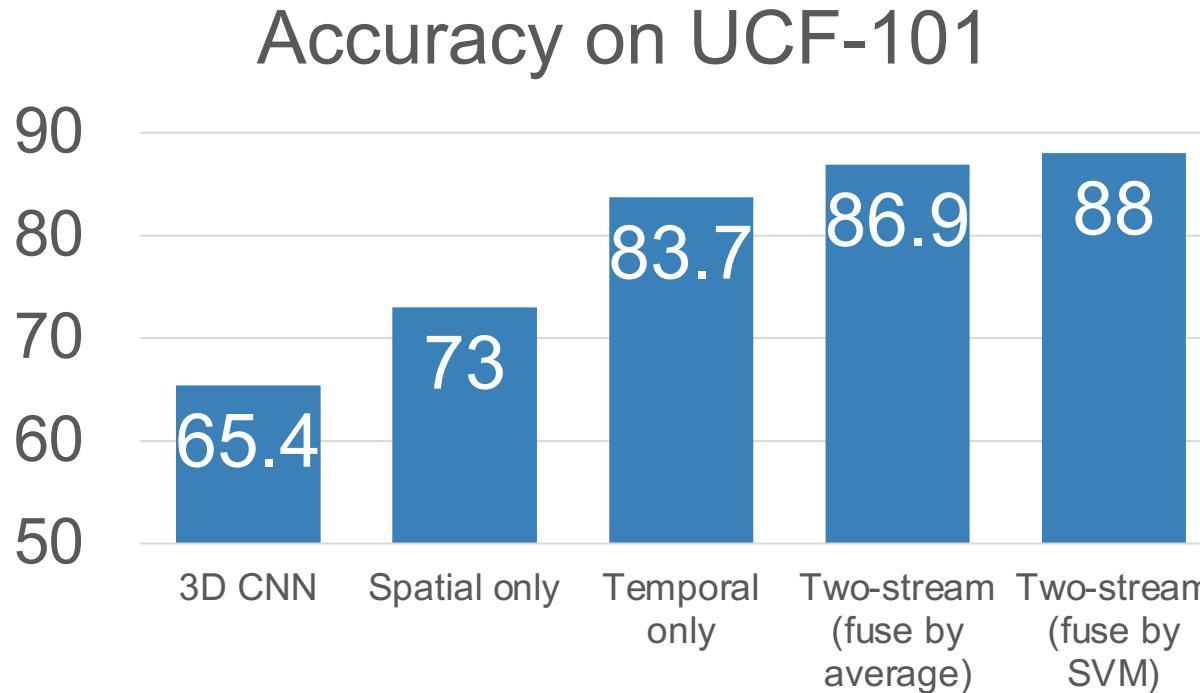
Input: Stack of optical flow:
 $[2^*(T-1)] \times H \times W$

Early fusion: First 2D conv
processes all flow images

Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

Slide credit: Justin Johnson

Separating Motion and Appearance: Two-Stream Networks

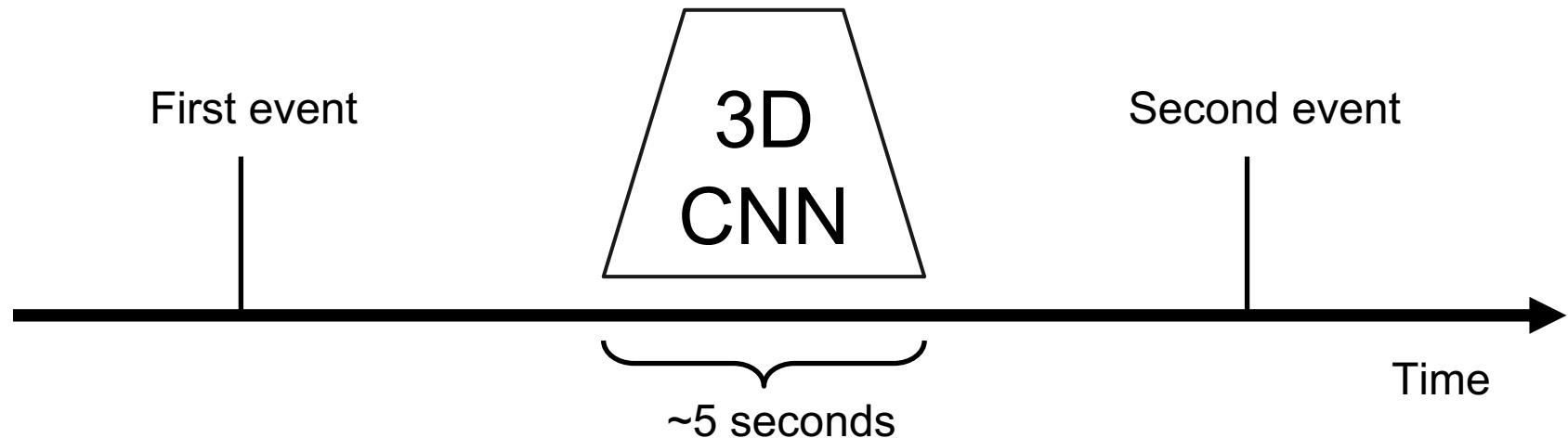


Simonyan and Zisserman, "Two-stream convolutional networks for action recognition in videos", NeurIPS 2014

Slide credit: Justin Johnson

Modeling long-term temporal structure

So far all our temporal CNNs only model local motion between frames in very short clips of ~2-5 seconds. What about long-term structure?

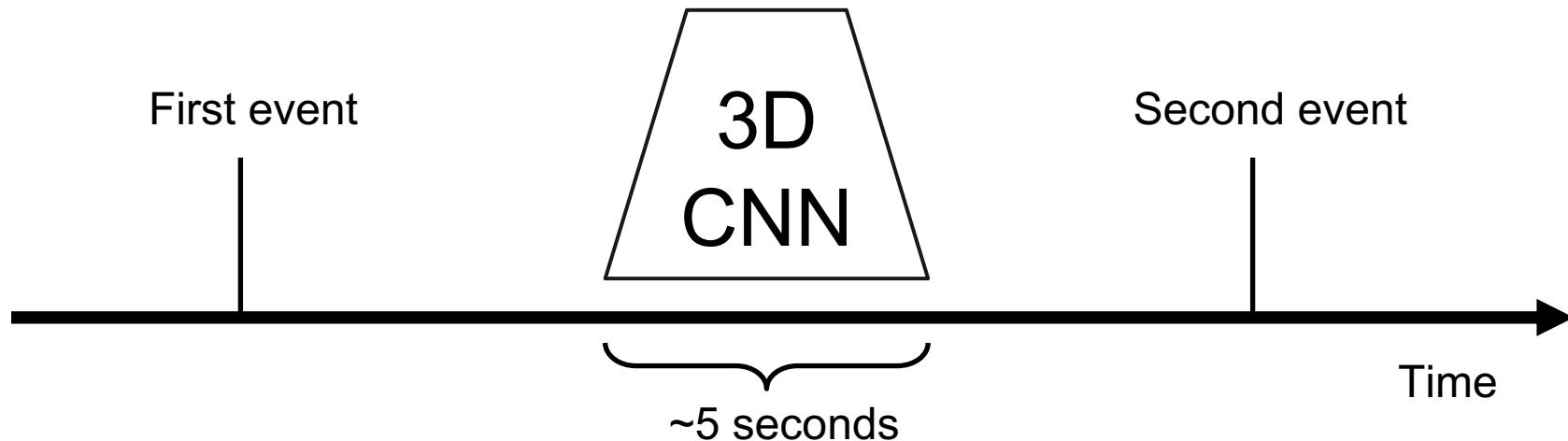


Slide credit: Justin Johnson

Modeling long-term temporal structure

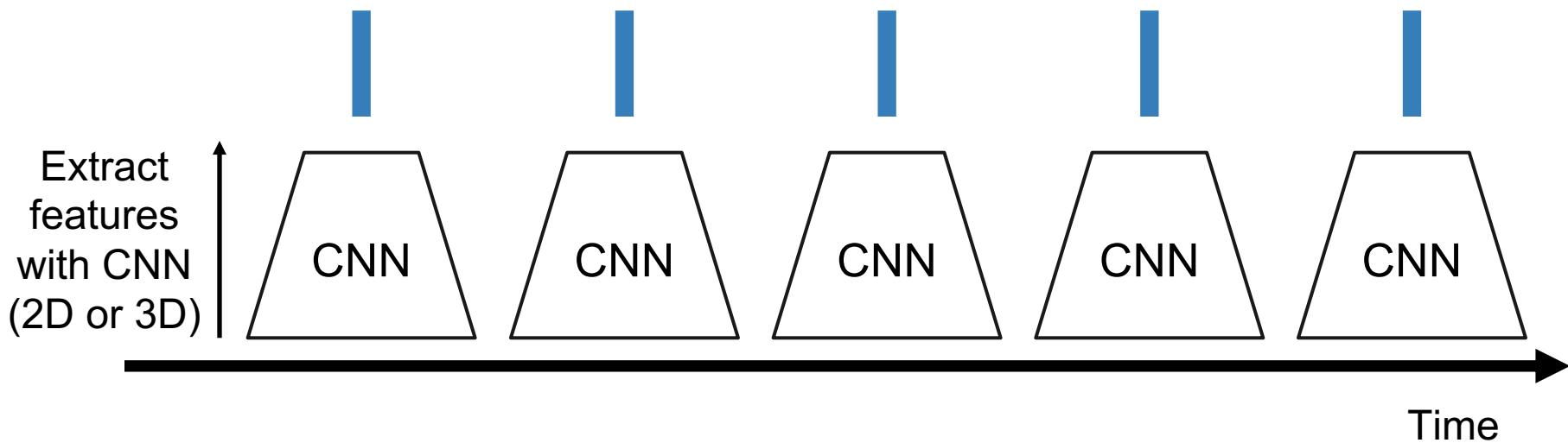
So far all our temporal CNNs only model local motion between frames in very short clips of ~2-5 seconds. What about long-term structure?

We know how to handle sequences! How about recurrent networks?



Slide credit: Justin Johnson

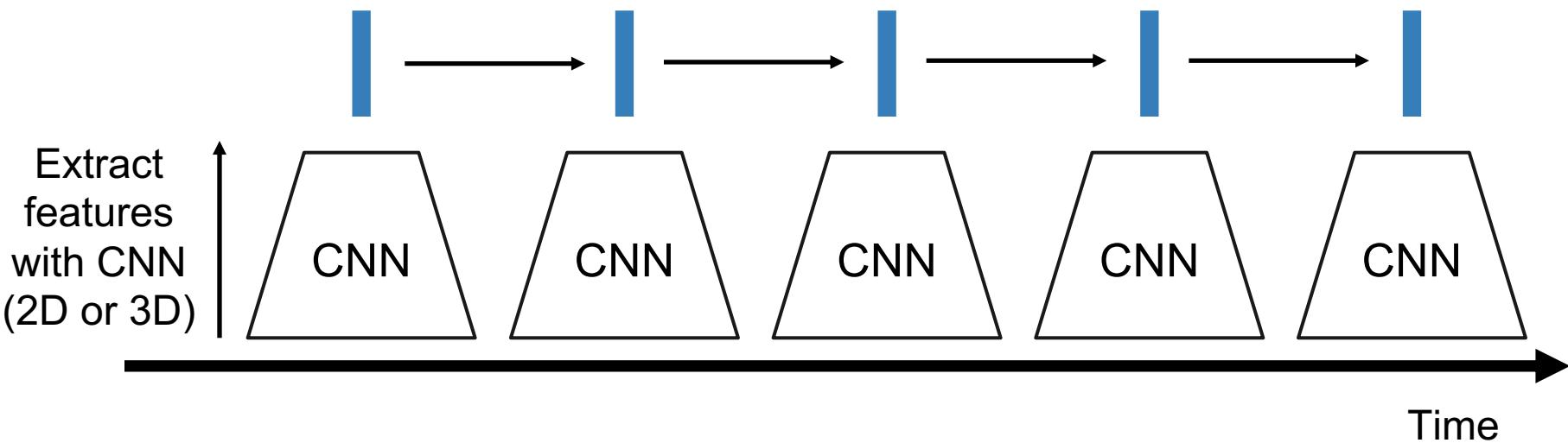
Modeling long-term temporal structure



Slide credit: Justin Johnson

Modeling long-term temporal structure

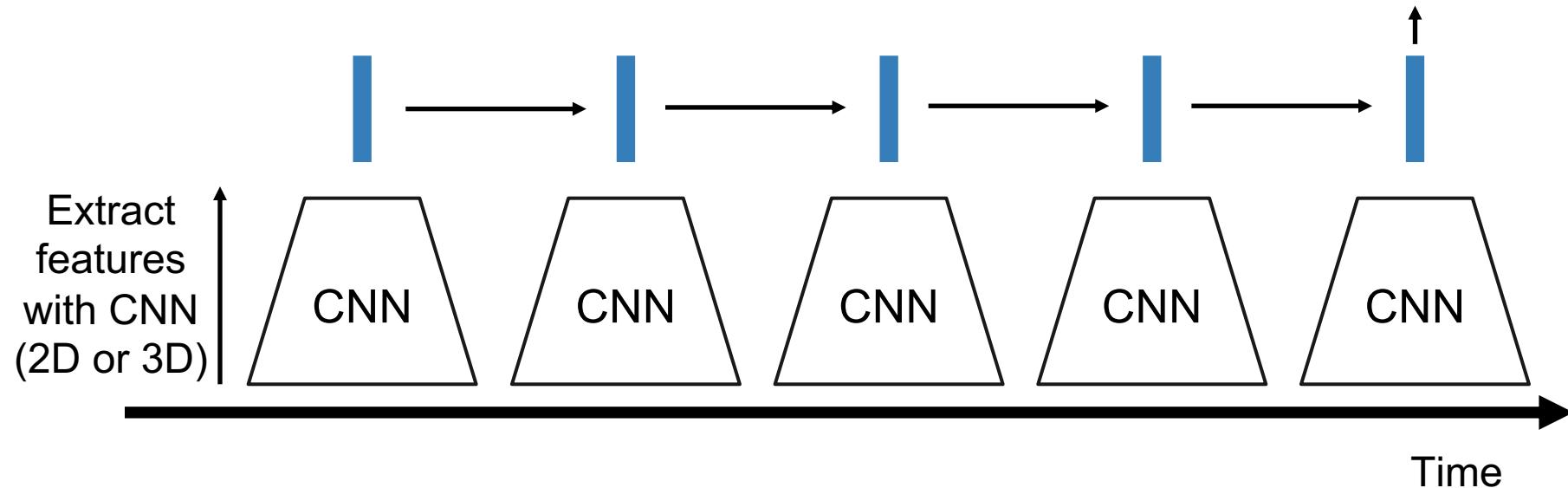
Process local features using recurrent network (e.g. LSTM)



Slide credit: Justin Johnson

Modeling long-term temporal structure

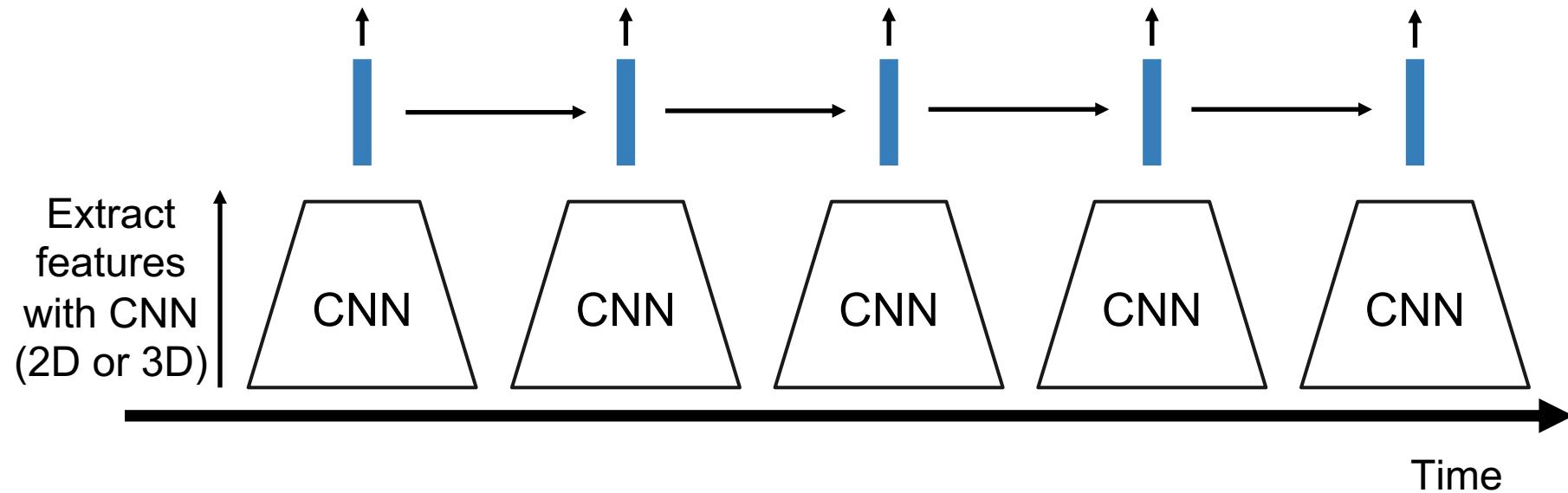
Process local features using recurrent network (e.g. LSTM)
Many to one: One output at end of video



Slide credit: Justin Johnson

Modeling long-term temporal structure

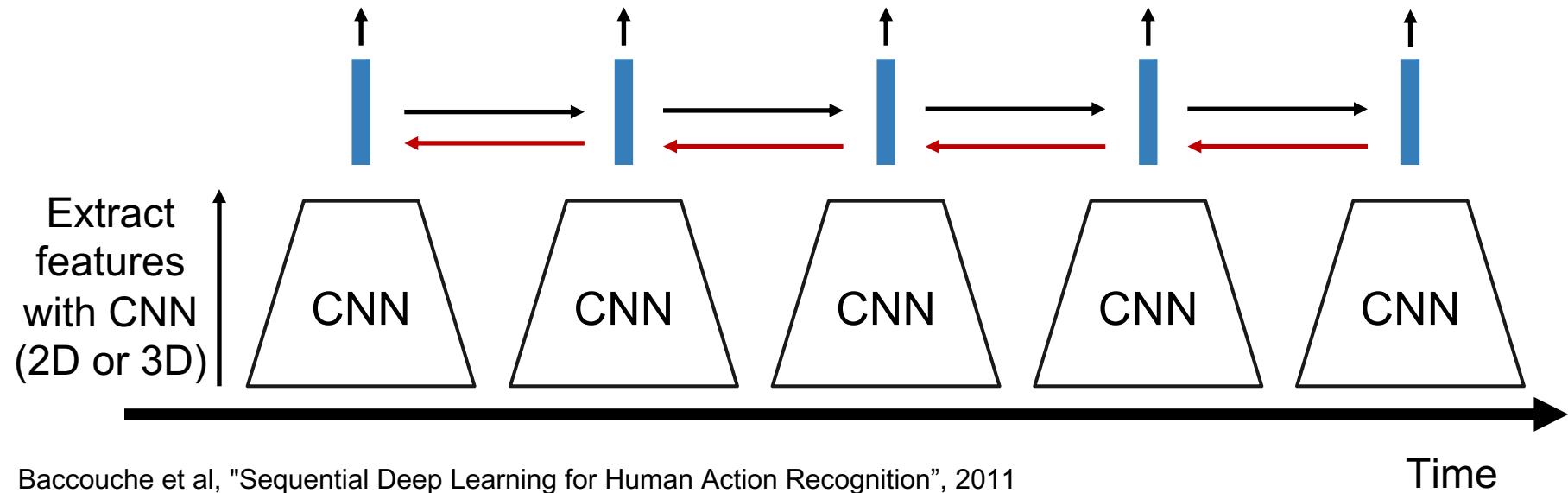
Process local features using recurrent network (e.g. LSTM)
Many to many: one output per video frame



Slide credit: Justin Johnson

Modeling long-term temporal structure

Sometimes don't backprop to CNN to save memory; pretrain and use it as a feature extractor



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

Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

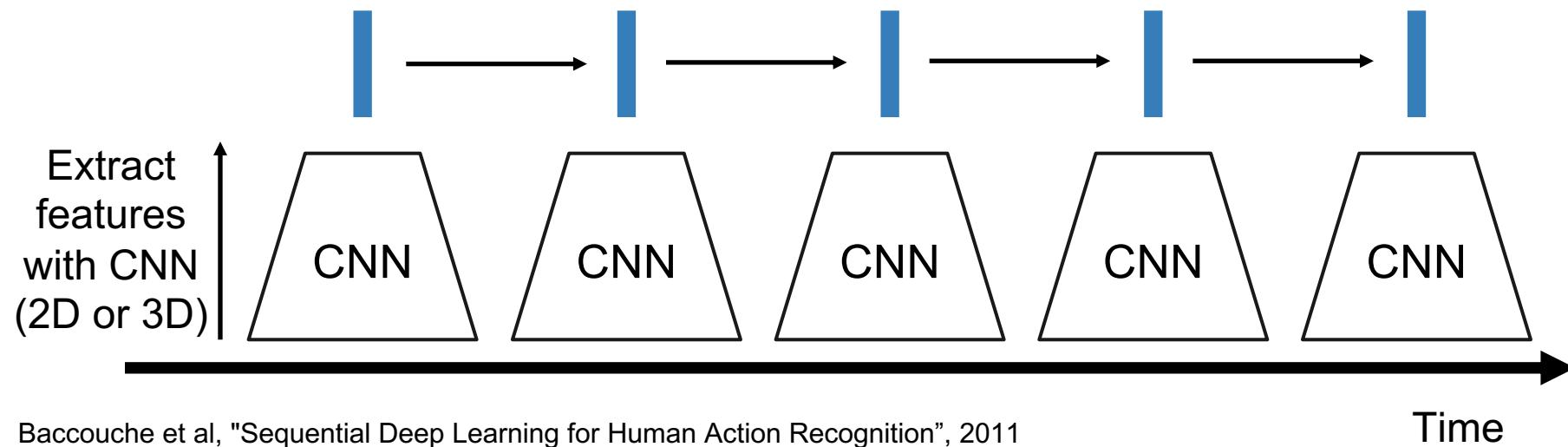
Slide credit: Justin Johnson

Modeling long-term temporal structure

Inside CNN: Each value is a function of a fixed temporal window (local temporal structure)

Inside RNN: Each vector is a function of all previous vectors (global temporal structure)

Can we merge both approaches?



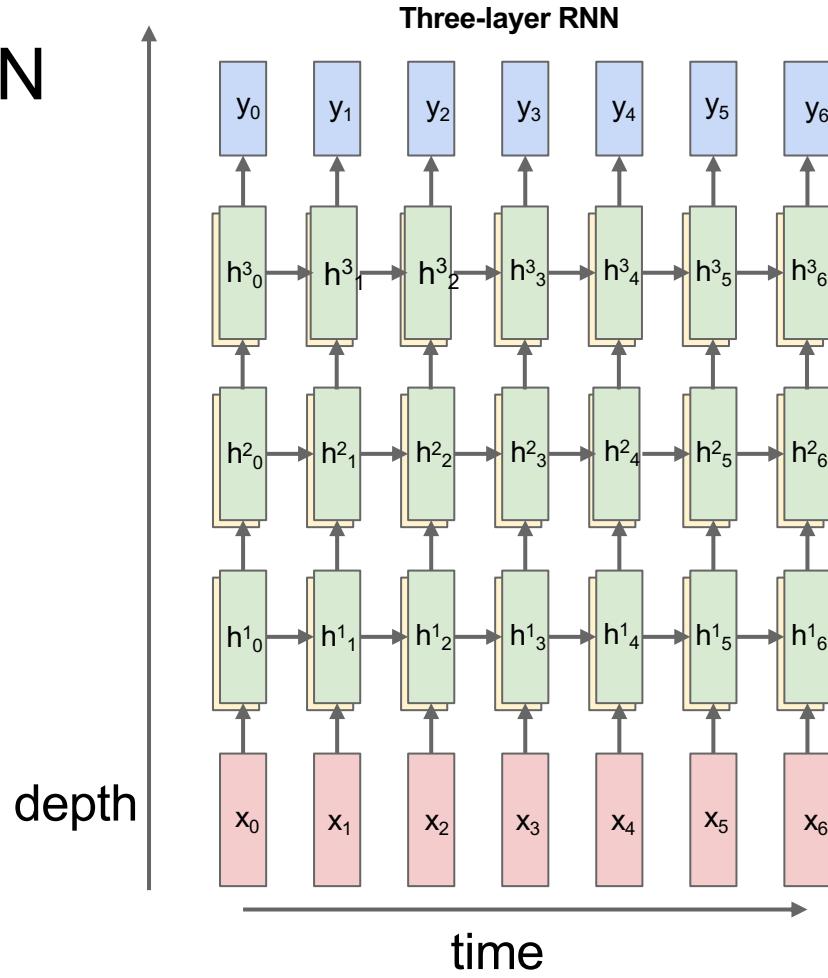
Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011

Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

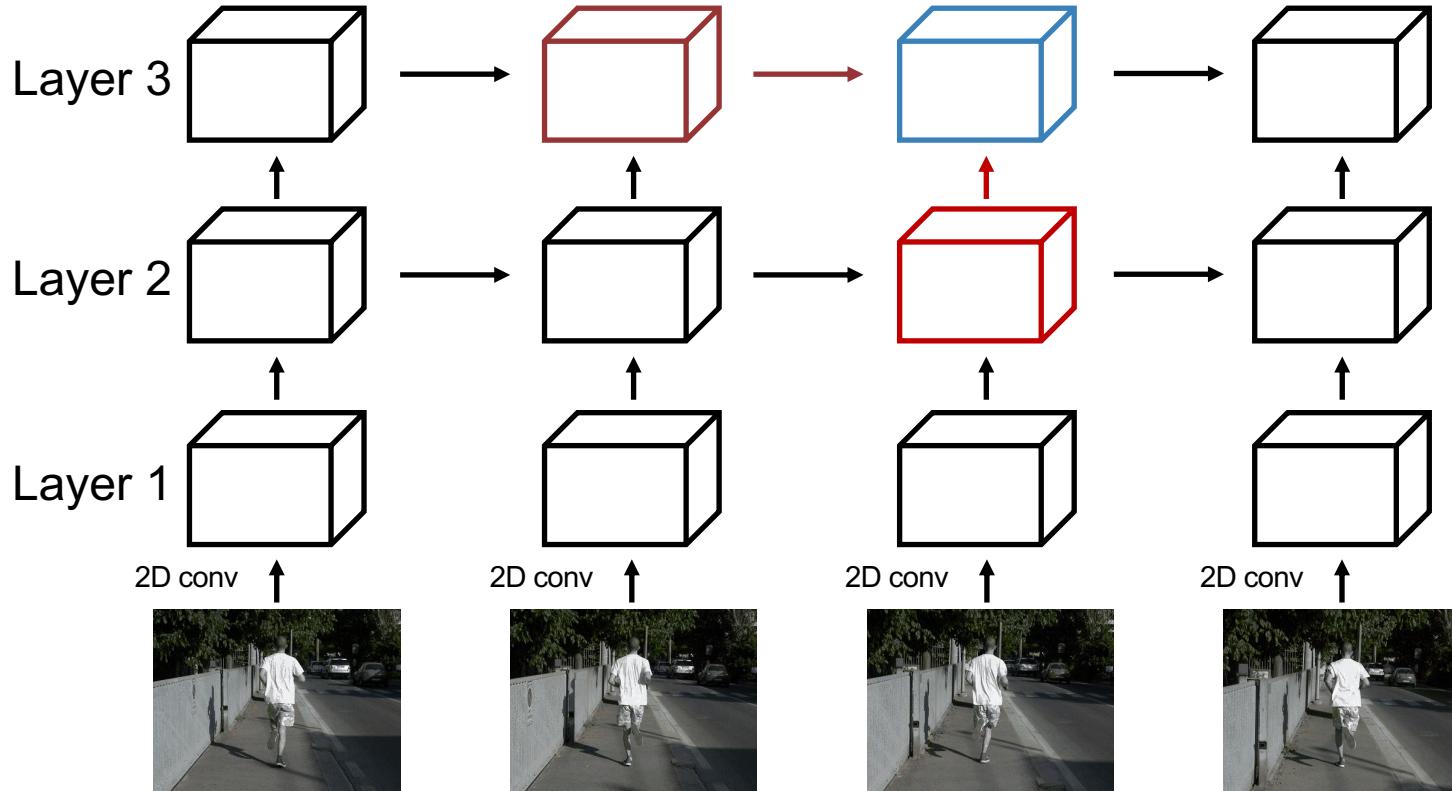
Slide credit: Justin Johnson

Recall: Multi-layer RNN

We can use a similar structure to process videos!



Recurrent Convolutional Network



Entire network uses 2D feature maps:
 $C \times H \times W$

Each depends on two inputs:

1. Same layer, previous timestep
2. Prev layer, same timestep

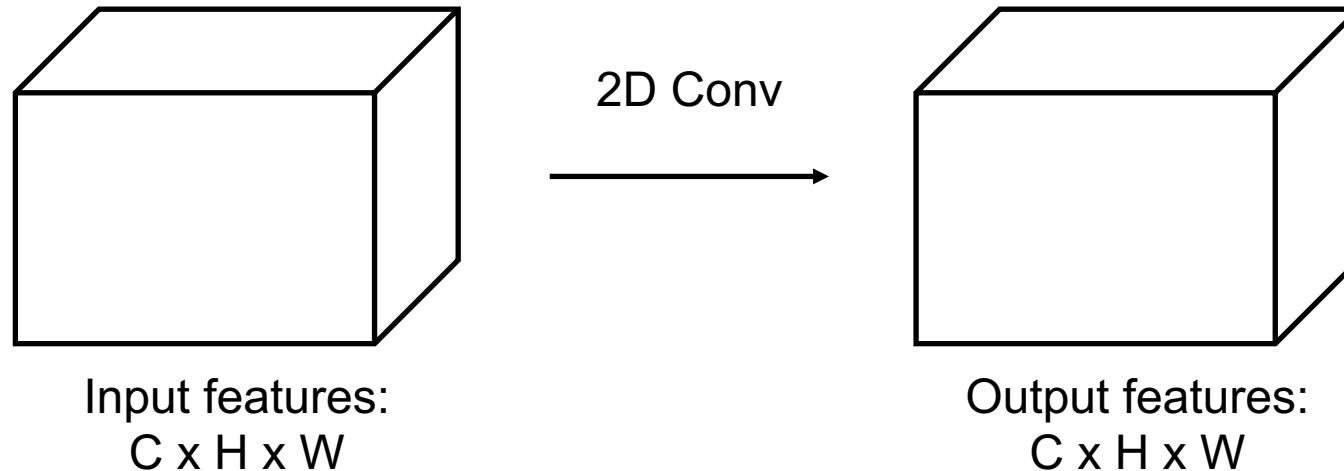
Use different weights at each layer, share weights across time

Ballas et al., "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

Slide credit: Justin Johnson

Recurrent Convolutional Network

Normal 2D CNN:



Slide credit: Justin Johnson

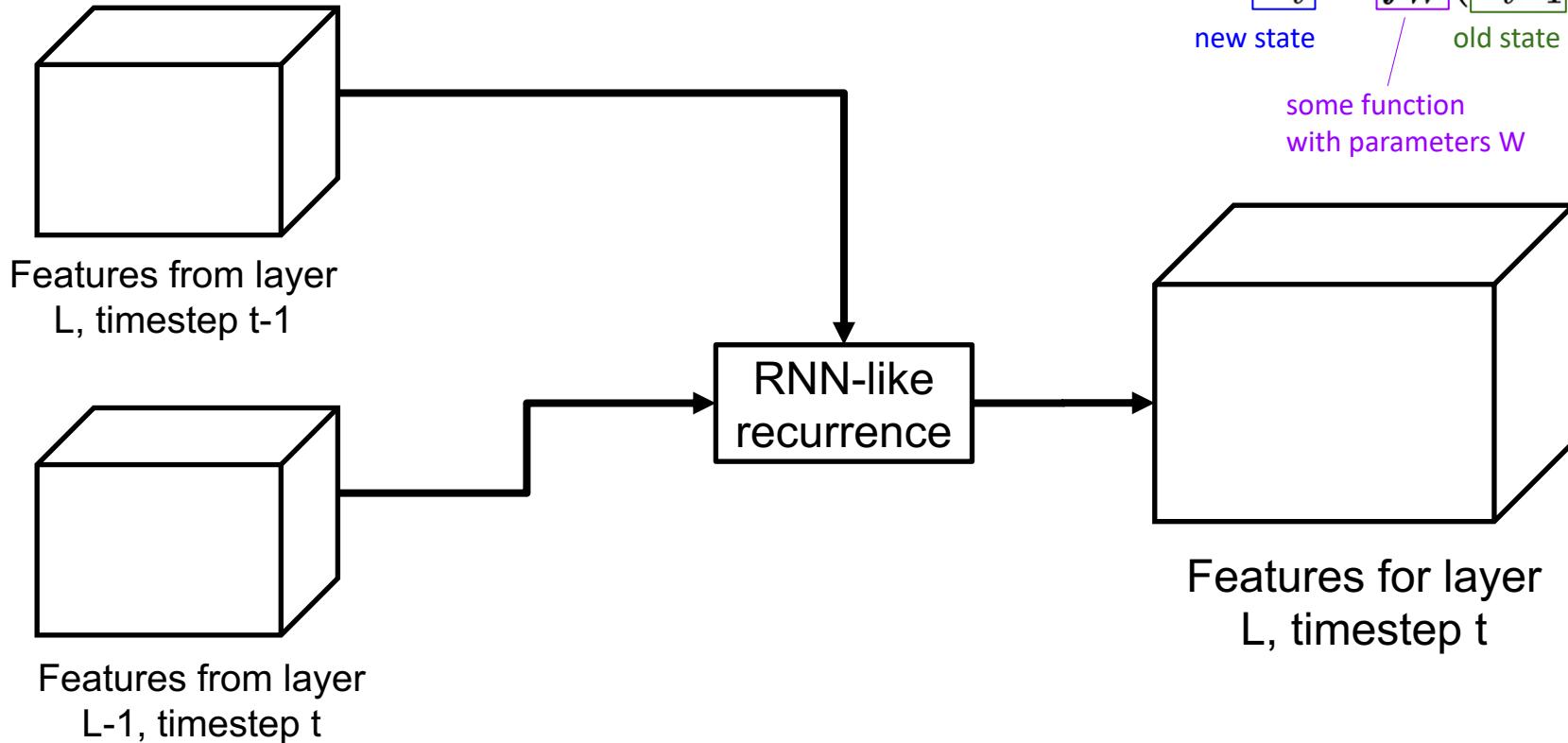
Recurrent Convolutional Network

Recall: Recurrent Network

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

new state / old state

some function
with parameters W



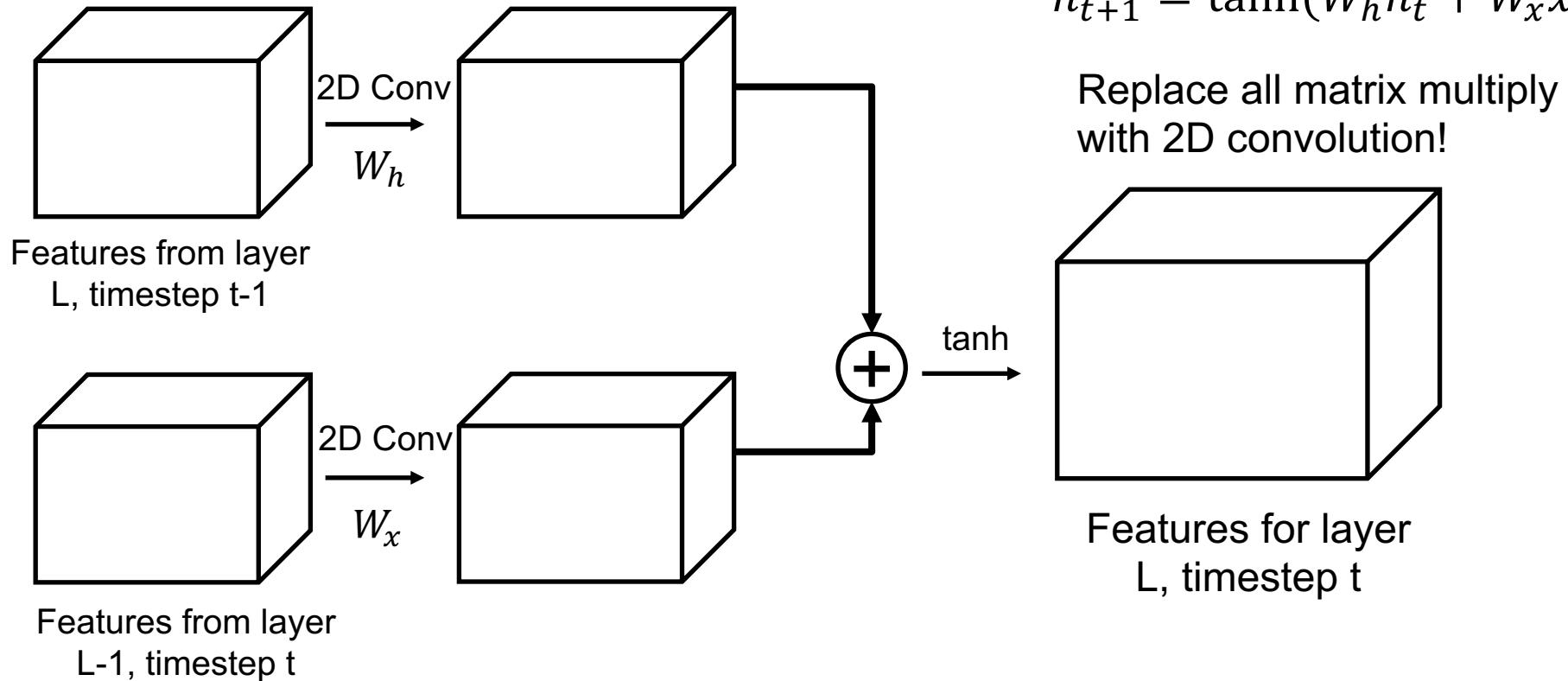
Ballas et al. "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

Slide credit: Justin Johnson

Recurrent Convolutional Network

Recall: Vanilla RNN

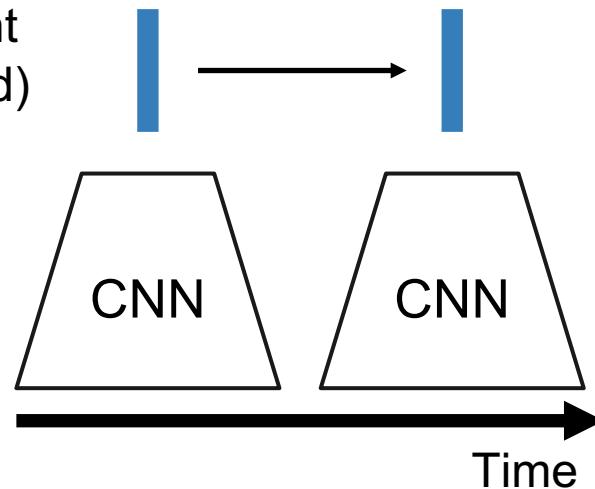
$$h_{t+1} = \tanh(W_h h_t + W_x x)$$



Ballas et al. "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

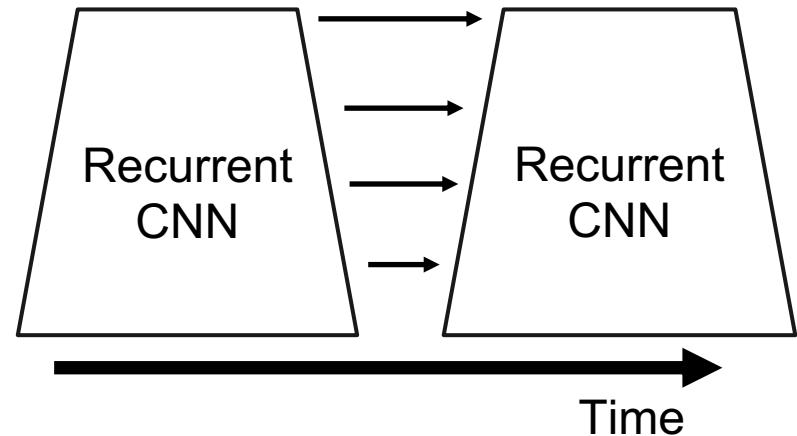
Modeling long-term temporal structure

RNN: Infinite
temporal extent
(fully-connected)



CNN: finite
temporal extent
(convolutional)

Recurrent CNN: Infinite
temporal extent
(convolutional)



Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011
Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

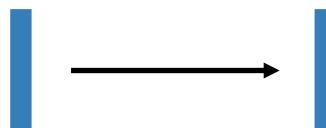
Ballas et al, "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

Slide credit: Justin Johnson

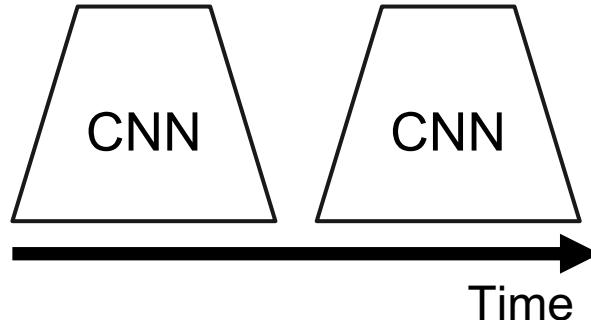
Modeling long-term temporal structure

Problem: RNNs are slow for long sequences (can't be parallelized)

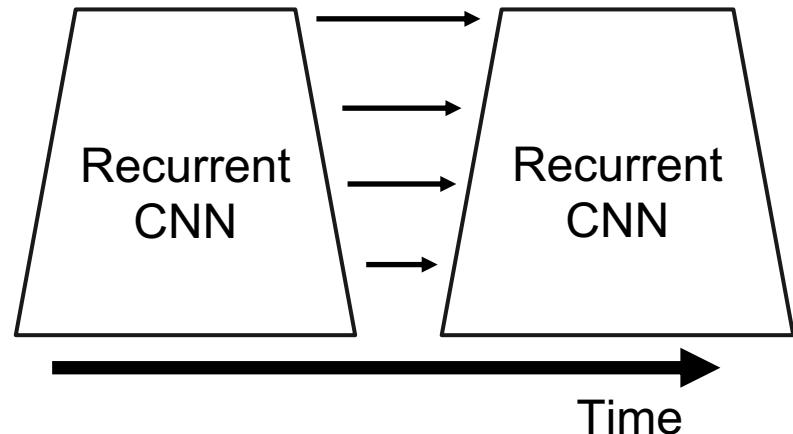
RNN: Infinite temporal extent (fully-connected)



CNN: finite temporal extent (convolutional)



Recurrent CNN: Infinite temporal extent (convolutional)



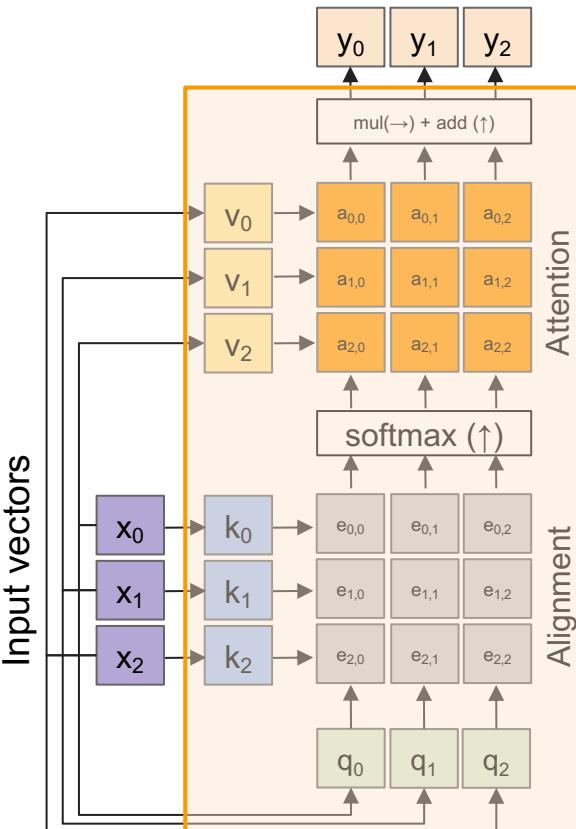
Baccouche et al, "Sequential Deep Learning for Human Action Recognition", 2011

Donahue et al, "Long-term recurrent convolutional networks for visual recognition and description", CVPR 2015

Ballas et al, "Delving Deeper into Convolutional Networks for Learning Video Representations", ICLR 2016

Slide credit: Justin Johnson

Recall: Self-Attention



Outputs:

context vectors: \mathbf{y} (shape: D_y)

Operations:

Key vectors: $\mathbf{k} = \mathbf{x}\mathbf{W}_k$

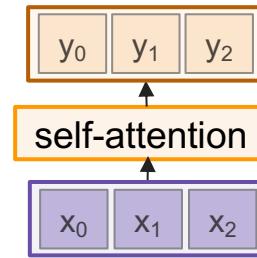
Value vectors: $\mathbf{v} = \mathbf{x}\mathbf{W}_v$

Query vectors: $\mathbf{q} = \mathbf{x}\mathbf{W}_q$

Alignment: $e_{i,j} = \mathbf{q}_j \cdot \mathbf{k}_i / \sqrt{D}$

Attention: $\mathbf{a} = \text{softmax}(\mathbf{e})$

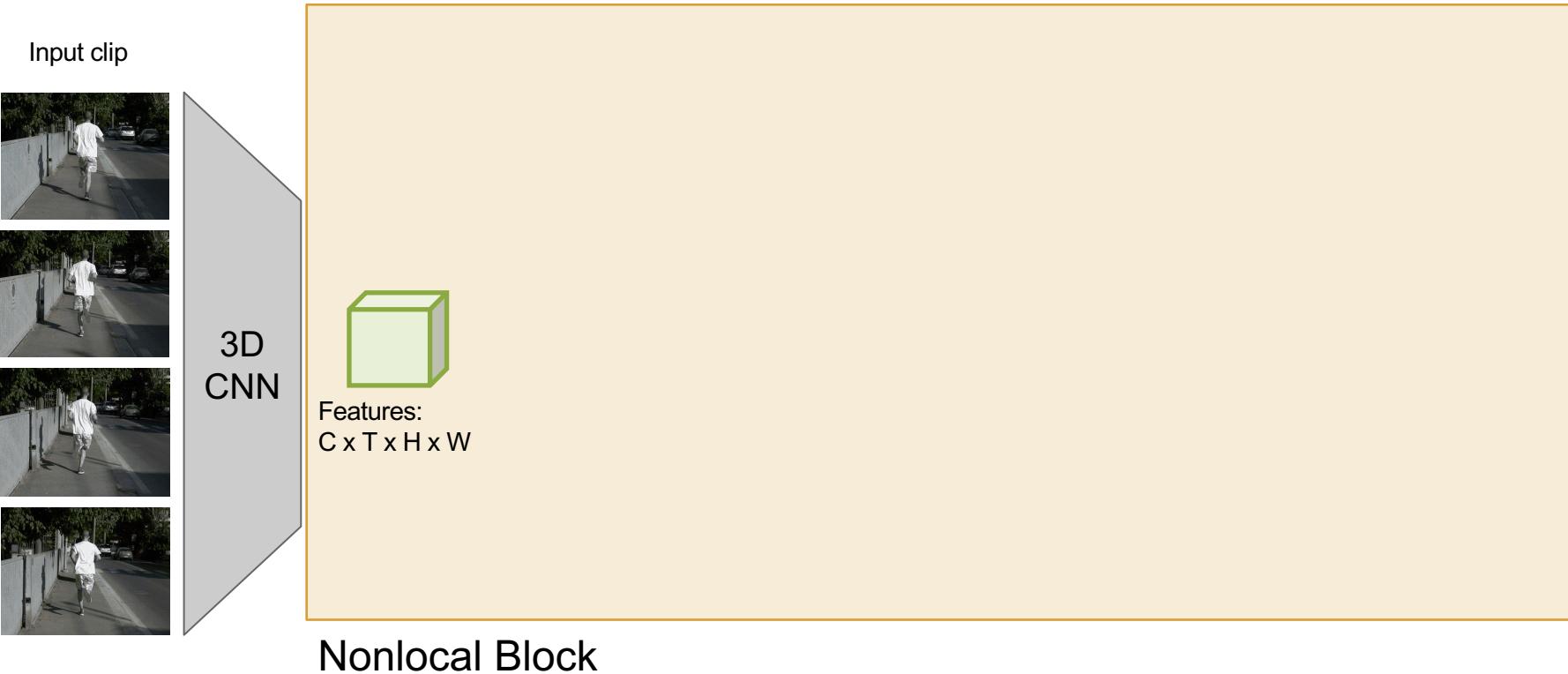
Output: $y_j = \sum_i a_{i,j} \mathbf{v}_i$



Inputs:

Input vectors: \mathbf{x} (shape: $N \times D$)

Spatio-Temporal Self-Attention (Nonlocal Block)



Wang et al, "Non-local neural networks", CVPR 2018

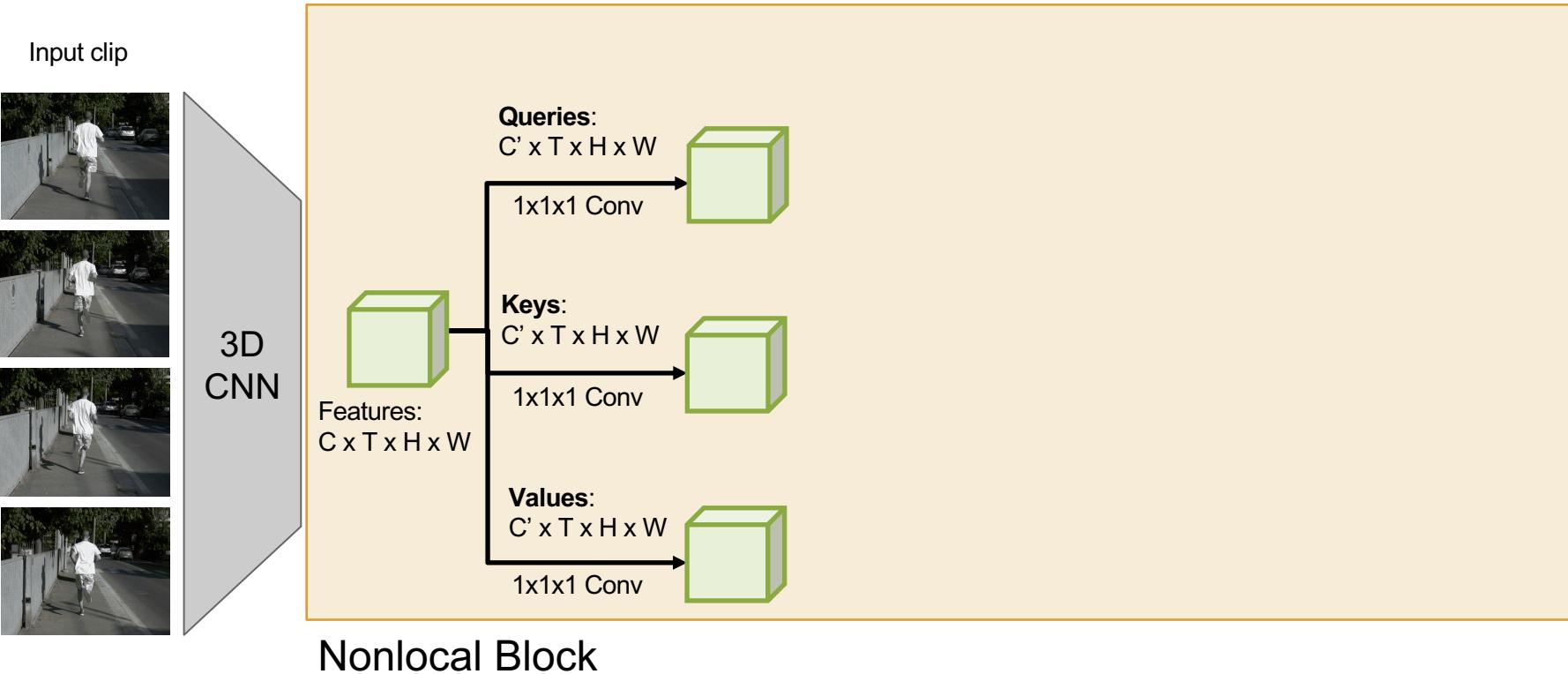
Fei-Fei Li, Jiajun Wu, Ruohan Gao

Lecture 12 - 71

Slide credit: Justin Johnson

May 5, 2022

Spatio-Temporal Self-Attention (Nonlocal Block)



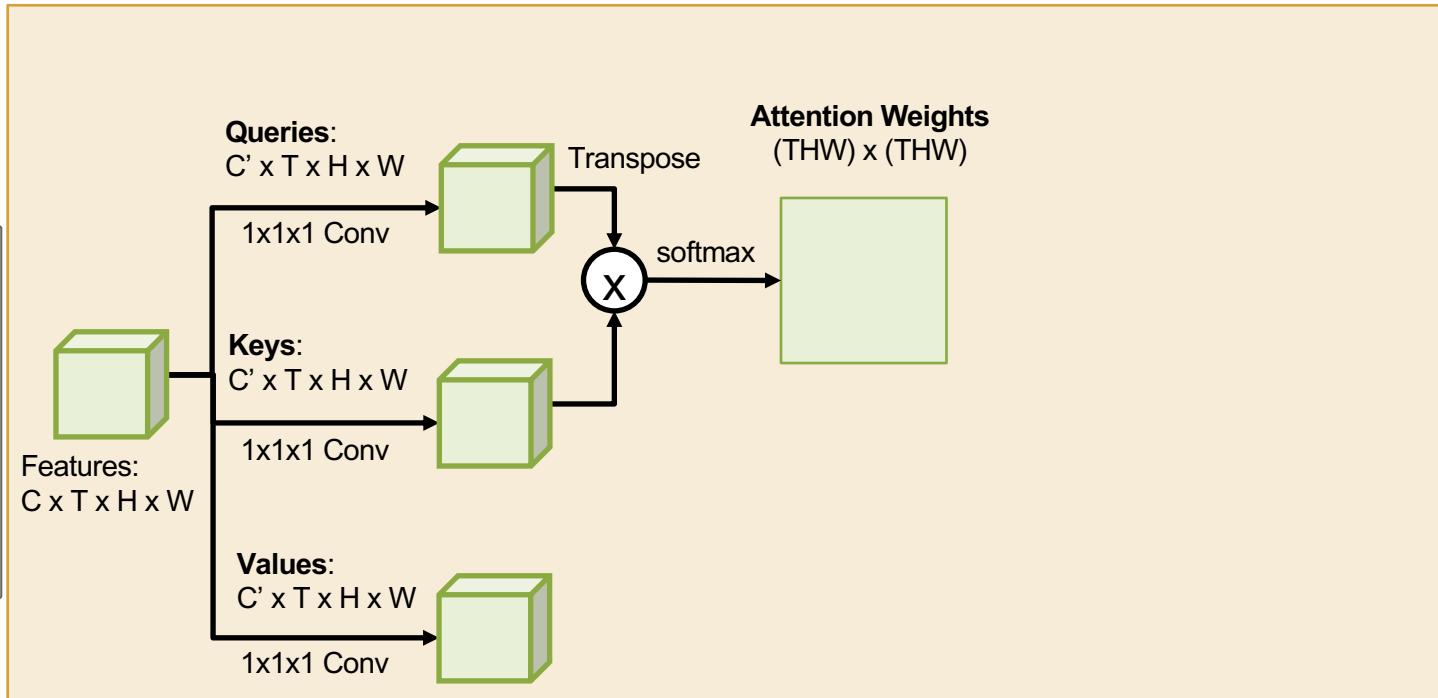
Wang et al, "Non-local neural networks", CVPR 2018

Spatio-Temporal Self-Attention (Nonlocal Block)

Input clip



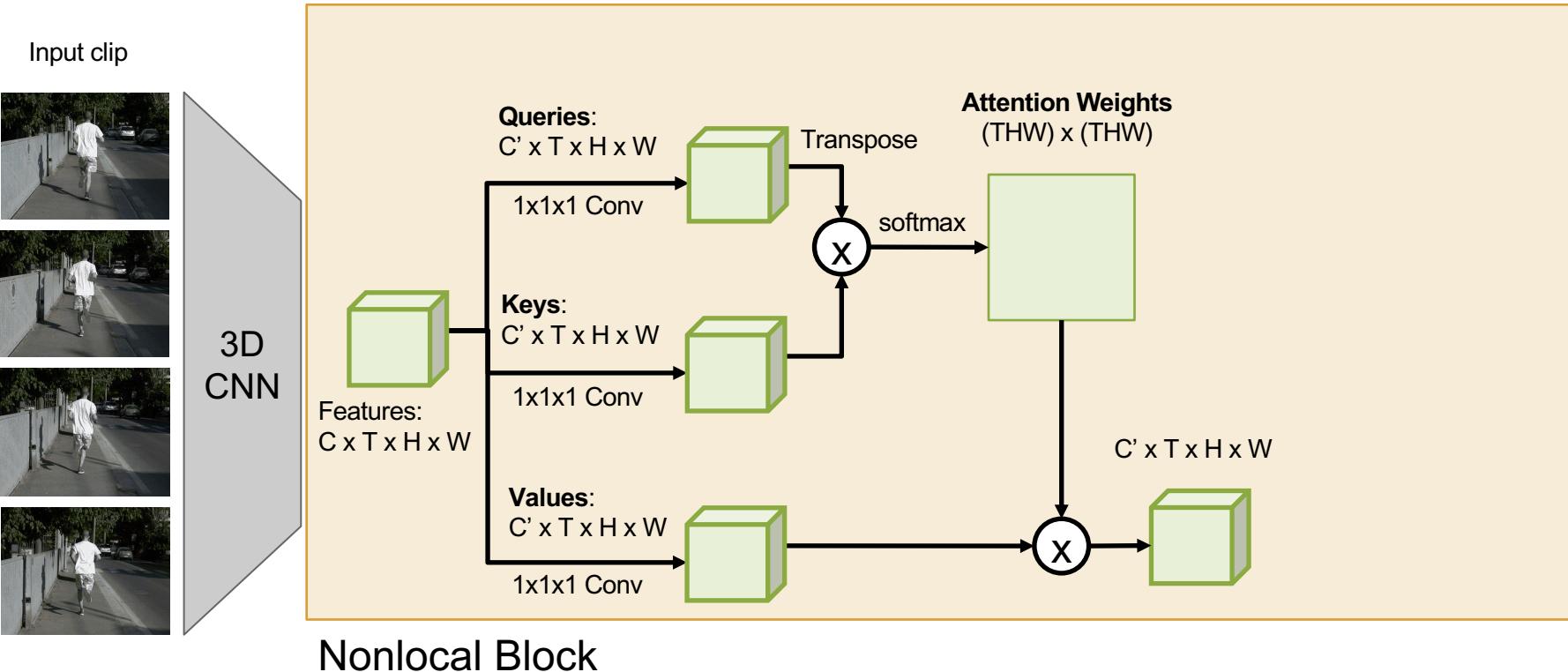
3D
CNN



Nonlocal Block

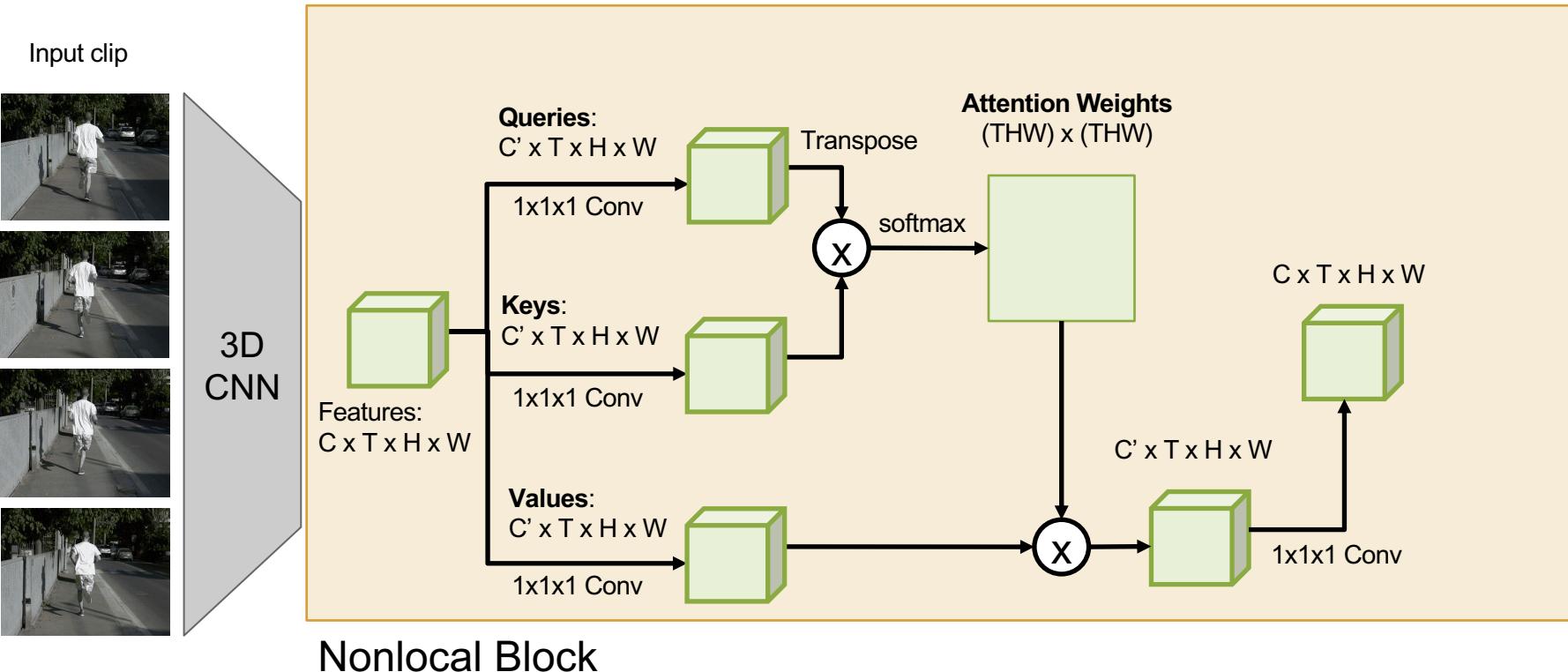
Wang et al, "Non-local neural networks", CVPR 2018

Spatio-Temporal Self-Attention (Nonlocal Block)



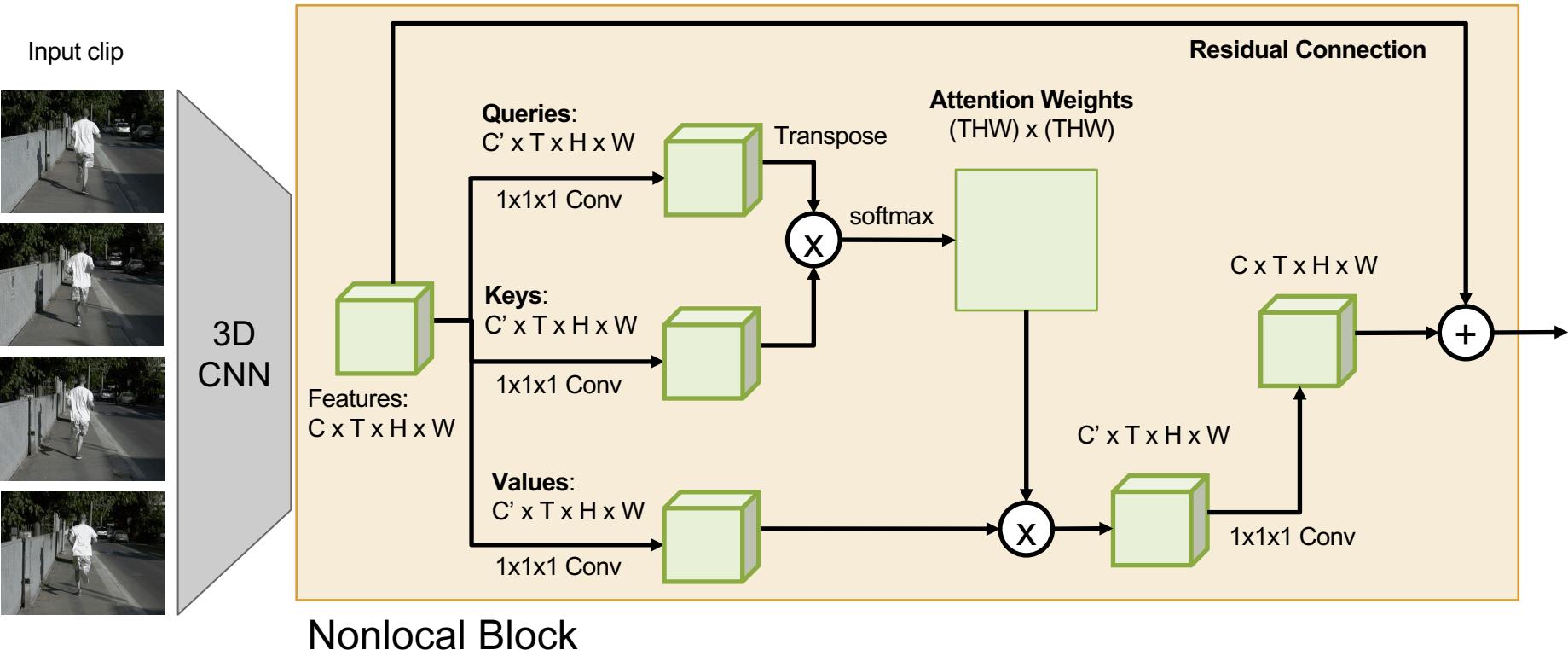
Wang et al, "Non-local neural networks", CVPR 2018

Spatio-Temporal Self-Attention (Nonlocal Block)



Wang et al, "Non-local neural networks", CVPR 2018

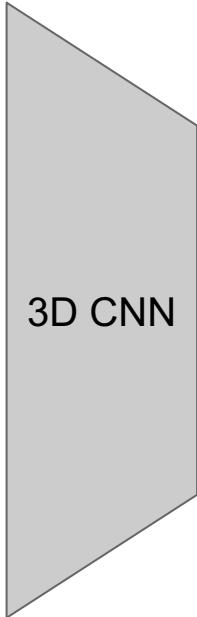
Spatio-Temporal Self-Attention (Nonlocal Block)



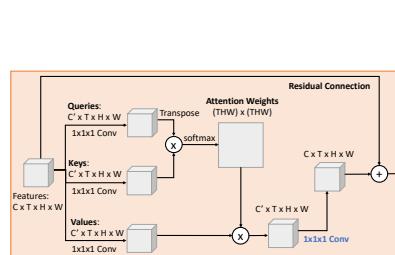
Wang et al, "Non-local neural networks", CVPR 2018

Spatio-Temporal Self-Attention (Nonlocal Block)

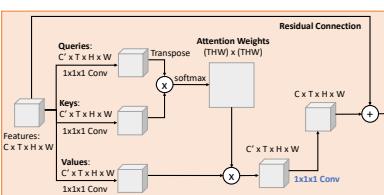
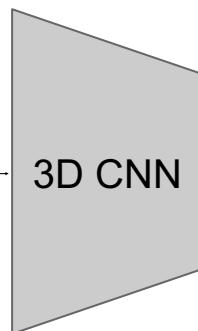
Input clip



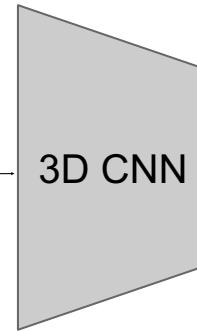
We can add nonlocal blocks into existing 3D CNN architectures.
But what is the best 3D CNN architecture?



Nonlocal Block



Nonlocal Block



Running

Slide credit: Justin Johnson

Wang et al, "Non-local neural networks", CVPR 2018

Fei-Fei Li, Jiajun Wu, Ruohan Gao

Lecture 12 - 77

May 5, 2022

Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images. Can we reuse image architectures for video?

Idea: take a 2D CNN architecture.

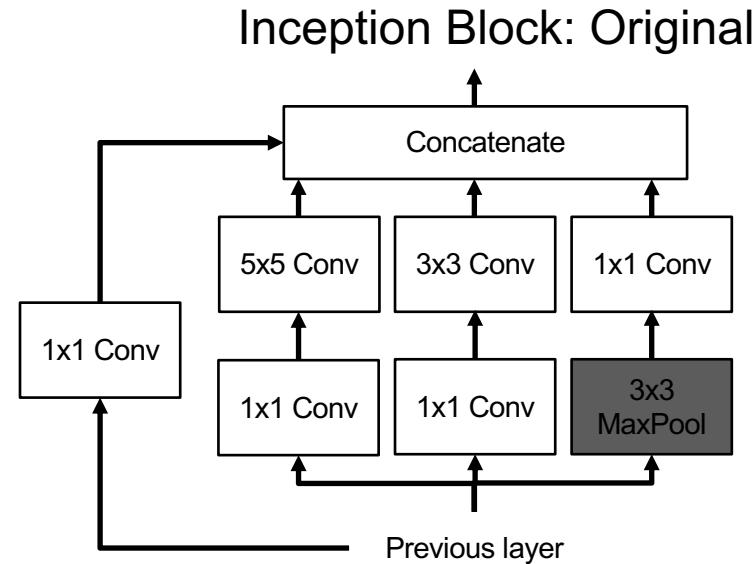
Replace each 2D $K_h \times K_w$ conv/pool layer with a 3D $K_t \times K_h \times K_w$ version

Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images. Can we reuse image architectures for video?

Idea: take a 2D CNN architecture.

Replace each $2D\ K_h \times K_w$ conv/pool layer with a $3D\ K_t \times K_h \times K_w$ version

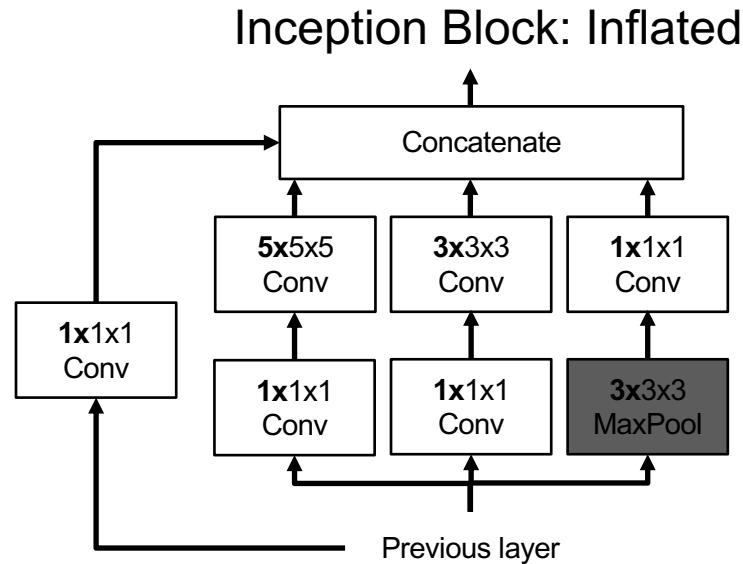


Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images. Can we reuse image architectures for video?

Idea: take a 2D CNN architecture.

Replace each $2D K_h \times K_w$ conv/pool layer with a $3D K_t \times K_h \times K_w$ version



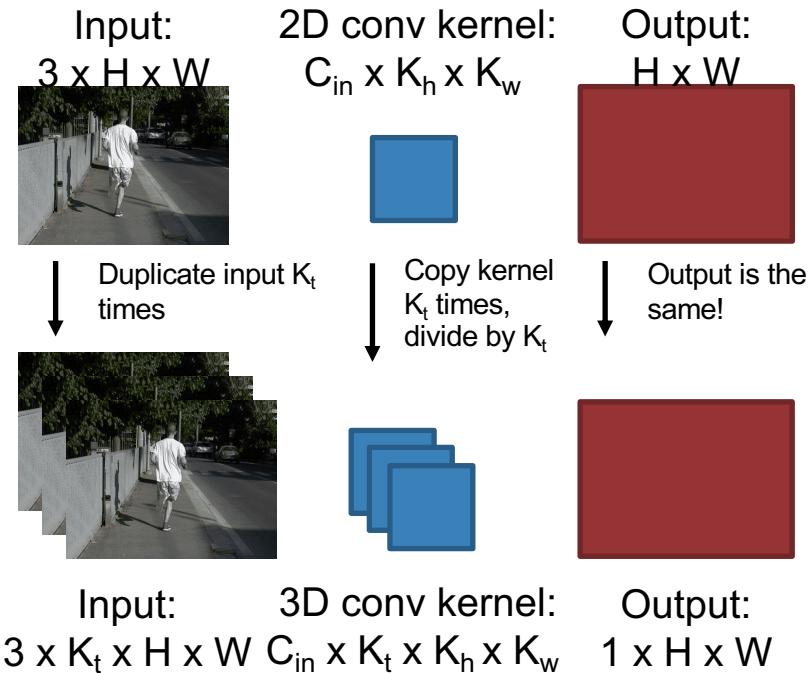
Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images. Can we reuse image architectures for video?

Idea: take a 2D CNN architecture.

Replace each 2D $K_h \times K_w$ conv/pool layer with a 3D $K_t \times K_h \times K_w$ version

Can use weights of 2D conv to initialize 3D conv: copy K_t times in space and divide by K_t
This gives the same result as 2D conv given “constant” video input



Carreira and Zisserman, “Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset”, CVPR 2017

Inflating 2D Networks to 3D (I3D)

There has been a lot of work on architectures for images. Can we reuse image architectures for video?

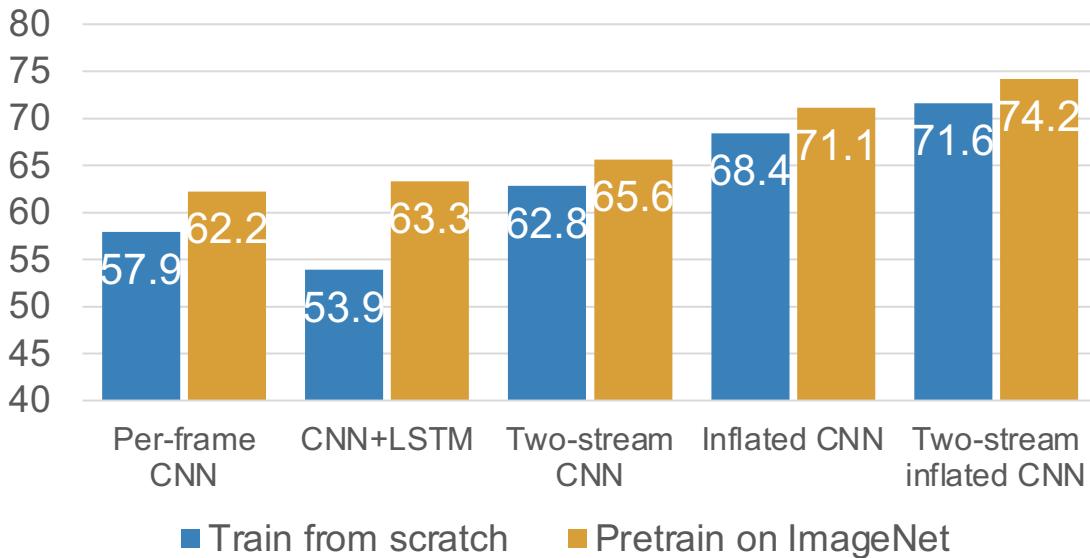
Idea: take a 2D CNN architecture.

Replace each 2D $K_h \times K_w$ conv/pool layer with a 3D $K_t \times K_h \times K_w$ version

Can use weights of 2D conv to initialize 3D conv: copy K_t times in space and divide by K_t

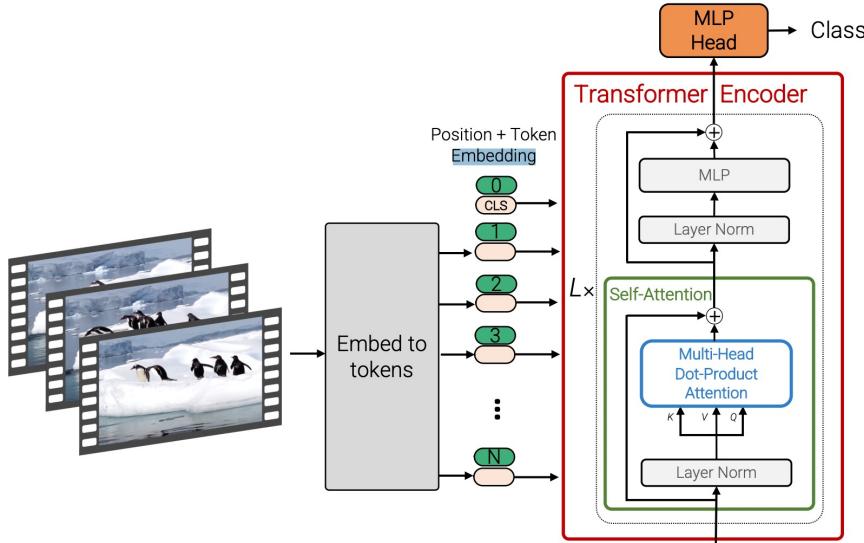
This gives the same result as 2D conv given “constant” video input

Top-1 Accuracy on Kinetics-400

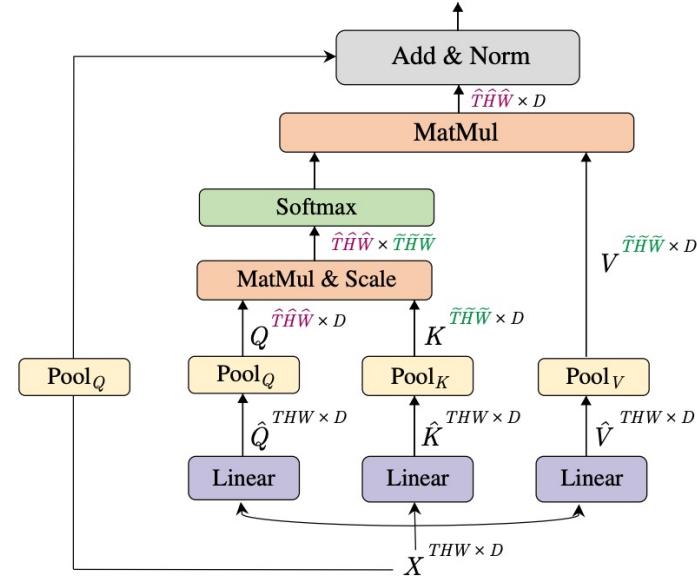


Vision Transformers for Video

Factorized attention: Attend over space / time



Pooling module: Reduce number of tokens



Bertasius et al, "Is Space-Time Attention All You Need for Video Understanding?", ICML 2021

Arnab et al, "ViViT: A Video Vision Transformer", ICCV 2021

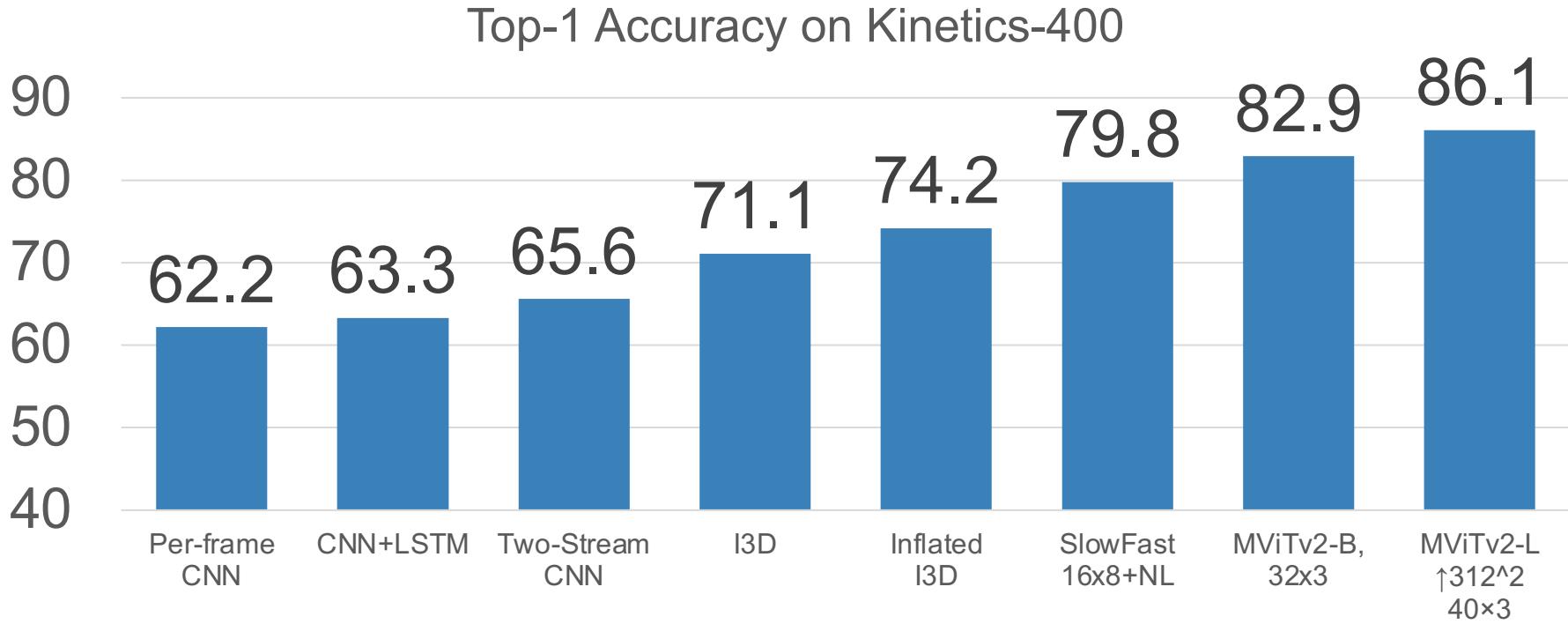
Neimark et al, "Video Transformer Network", ICCV 2021

Fan et al, "Multiscale Vision Transformers", ICCV 2021

Li et al, "MViTv2: Improved Multiscale Vision Transformers for Classification and Detection", CVPR 2022

Slide credit: Justin Johnson

Vision Transformers for Video



Li et al, "MViTv2: Improved Multiscale Vision Transformers for Classification and Detection", CVPR 2022

Slide credit: Justin Johnson

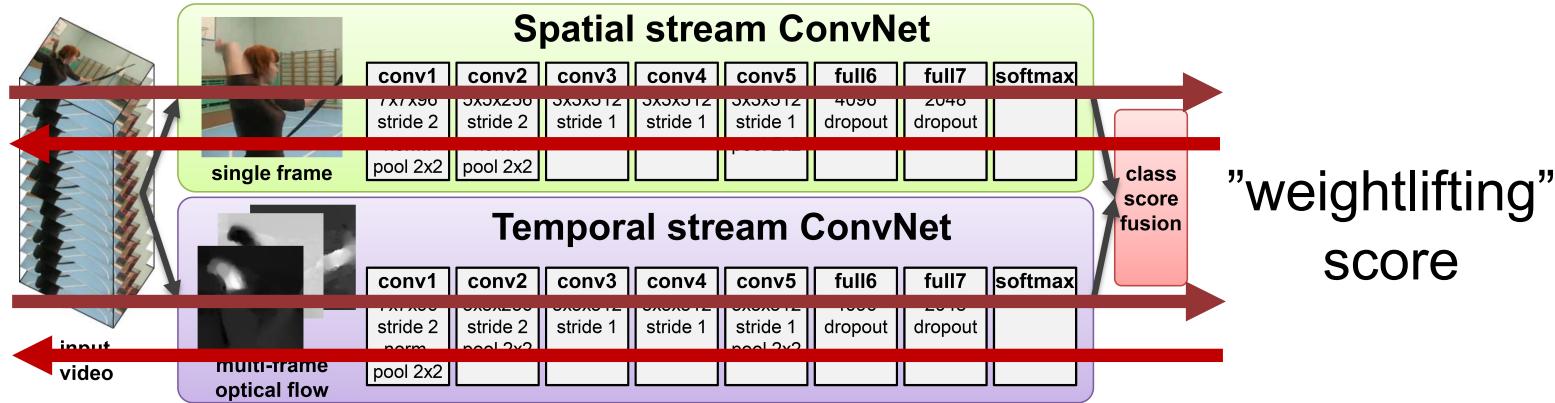
Visualizing Video Models

Image



Flow

Forward: Compute class score



"weightlifting"
score

Backward: Compute gradient

Add a term to encourage spatially smooth flow; tune penalty to pick out “slow” vs “fast” motion

Figure credit: Simonyan and Zisserman, “Two-stream convolutional networks for action recognition in videos”, NeurIPS 2014

Feichtenhofer et al, “What have we learned from deep representations for action recognition?”, CVPR 2018
Feichtenhofer et al, “Deep insights into convolutional networks for video recognition?”, IJCV 2019.

Slide credit: Justin Johnson

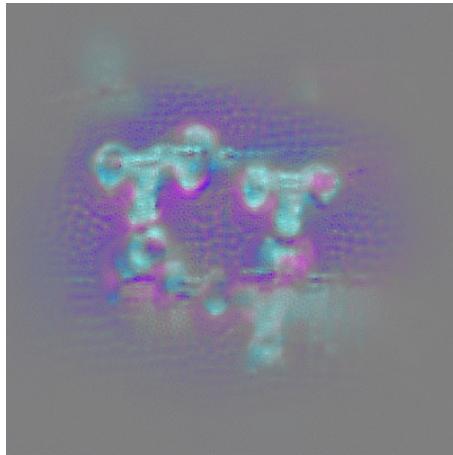
Can you guess the action?

Feichtenhofer et al, "What have we learned from deep representations for action recognition?", CVPR 2018
Feichtenhofer et al, "Deep insights into convolutional networks for video recognition?", IJCV 2019.
Slide credit: Christoph Feichtenhofers

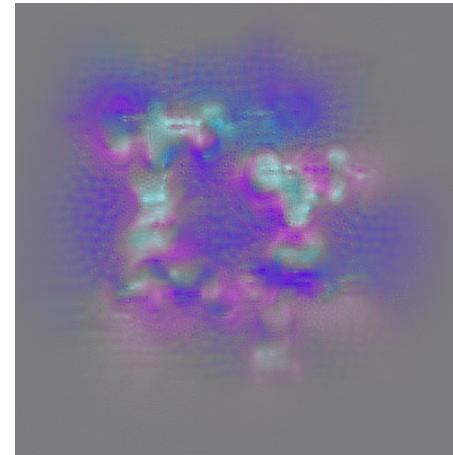
Appearance



“Slow” motion

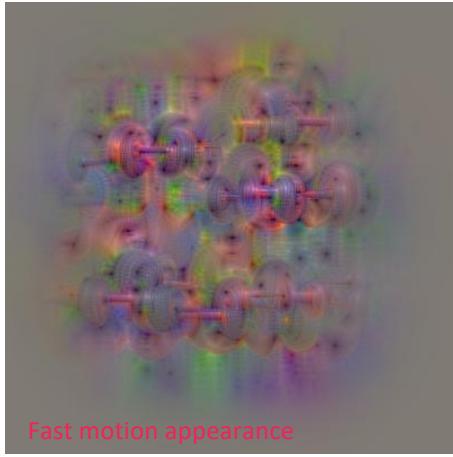


“Fast” motion

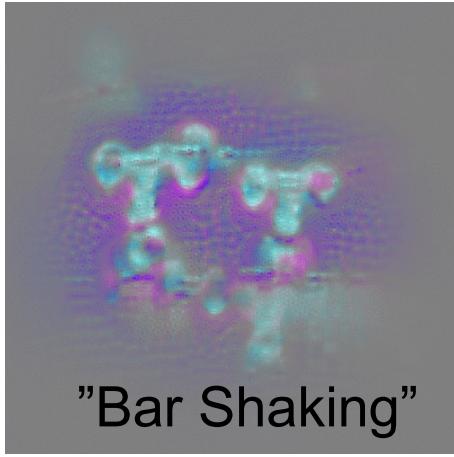


Can you guess the action? Weightlifting

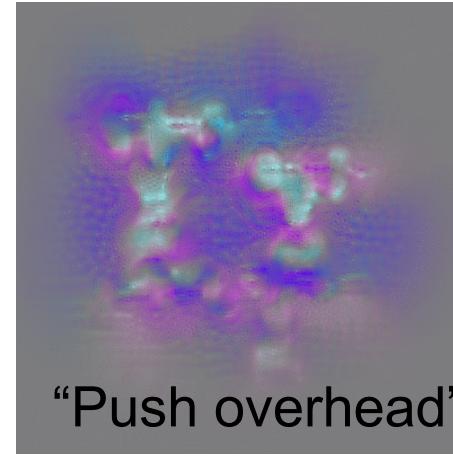
Appearance



“Slow” motion



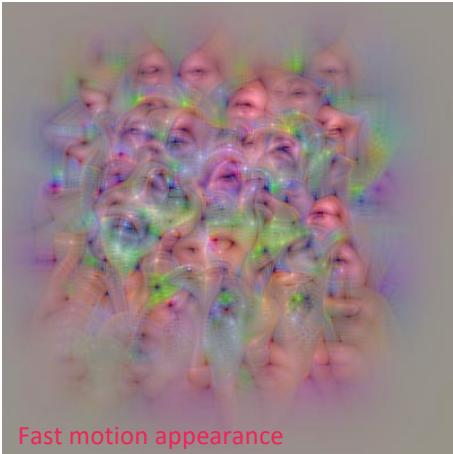
“Fast” motion



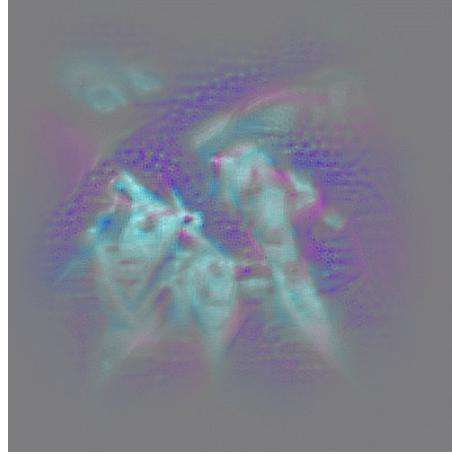
Slide credit: Justin Johnson

Can you guess the action?

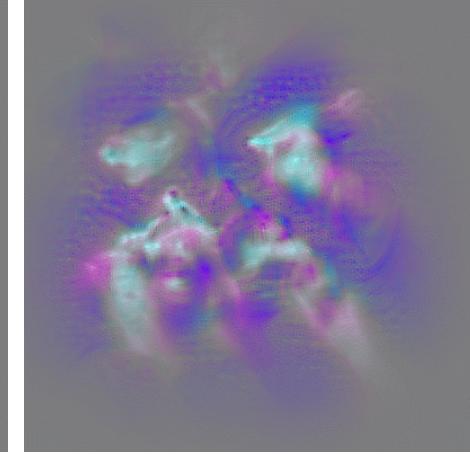
Appearance



“Slow” motion



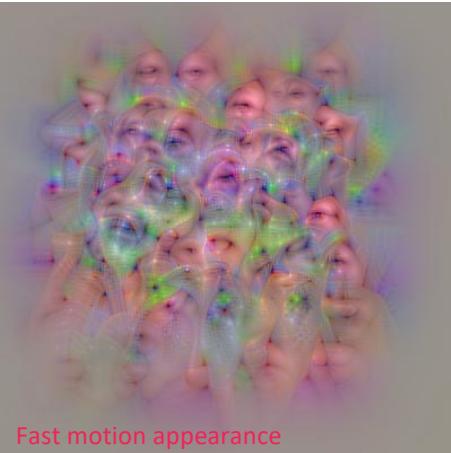
“Fast” motion



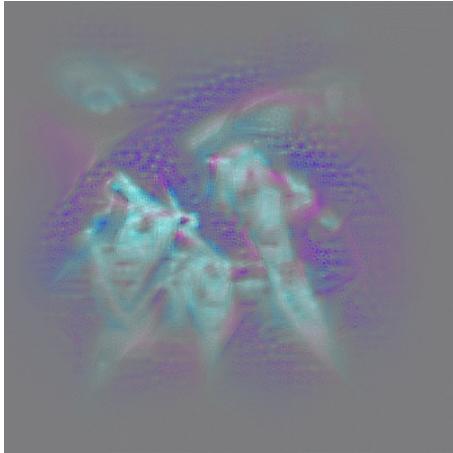
Slide credit: Justin Johnson

Can you guess the action? Apply Eye Makeup

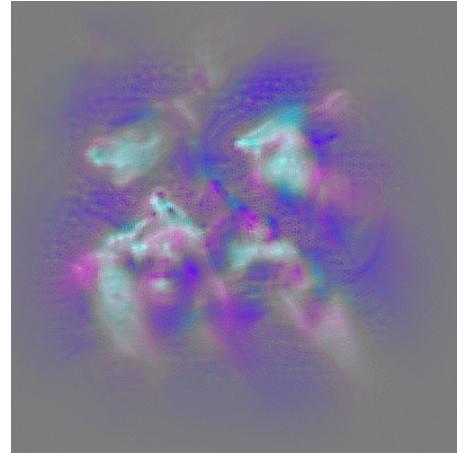
Appearance



“Slow” motion



“Fast” motion



Slide credit: Justin Johnson

So far: Classify short clips



Videos: Recognize **actions**

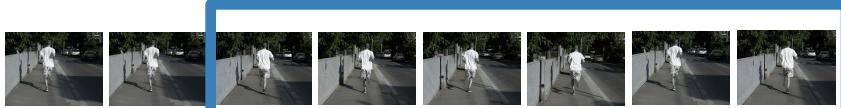


Swimming
Running
Jumping
Eating
Standing

Temporal Action Localization

Given a long untrimmed video sequence, identify frames corresponding to different actions

Running



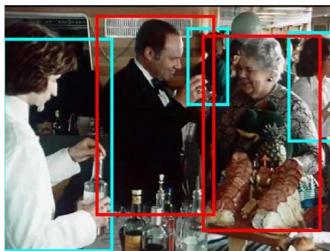
Jumping



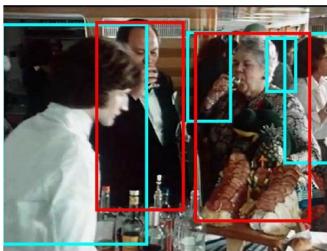
Can use architecture similar to Faster R-CNN:
first generate **temporal proposals** then **classify**

Spatio-Temporal Detection

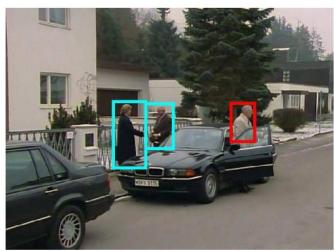
Given a long untrimmed video, detect all the people in both space and time and classify the activities they are performing.
Some examples from AVA Dataset:



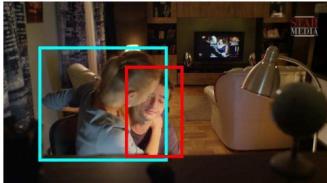
clink glass → drink



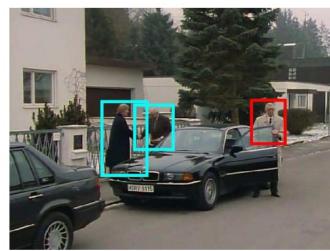
open → close



grab (a person) → hug



look at phone → answer phone



Today: Temporal Stream



3D CNN, Two-Stream Neural Network, Spatial-Temporal Self-Attention.....



Ba Ba Ba

...

(McGurk & McDonald 1976)

Fa Fa Fa ...

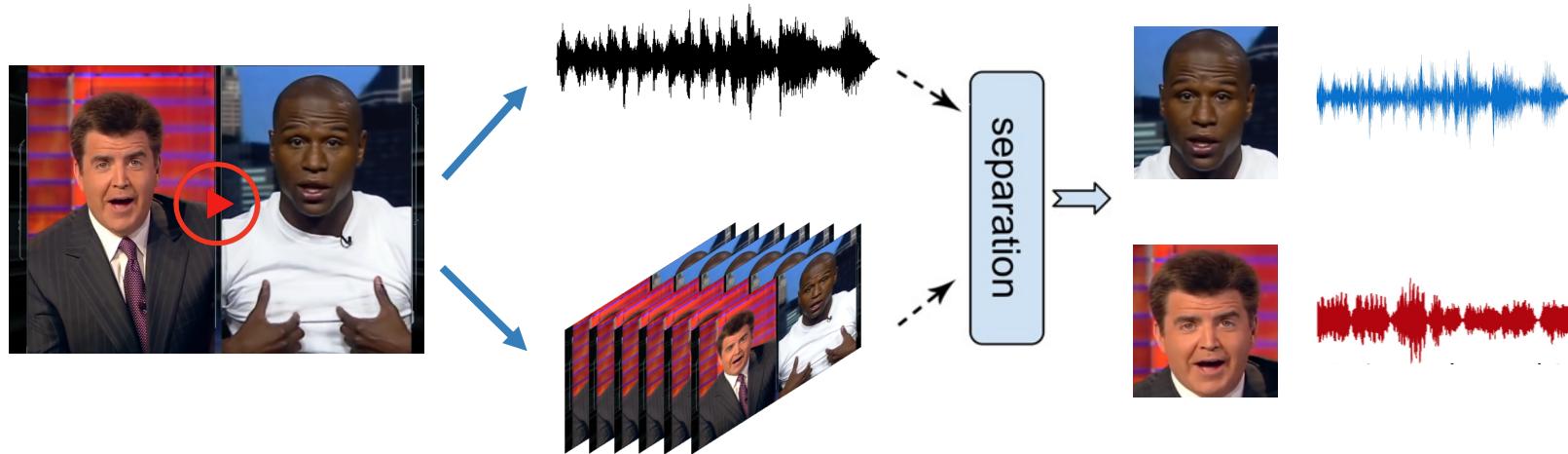
(McGurk & McDonald 1976)



Video source: BBC

(McGurk & McDonald 1976)

Visually-guided audio source separation



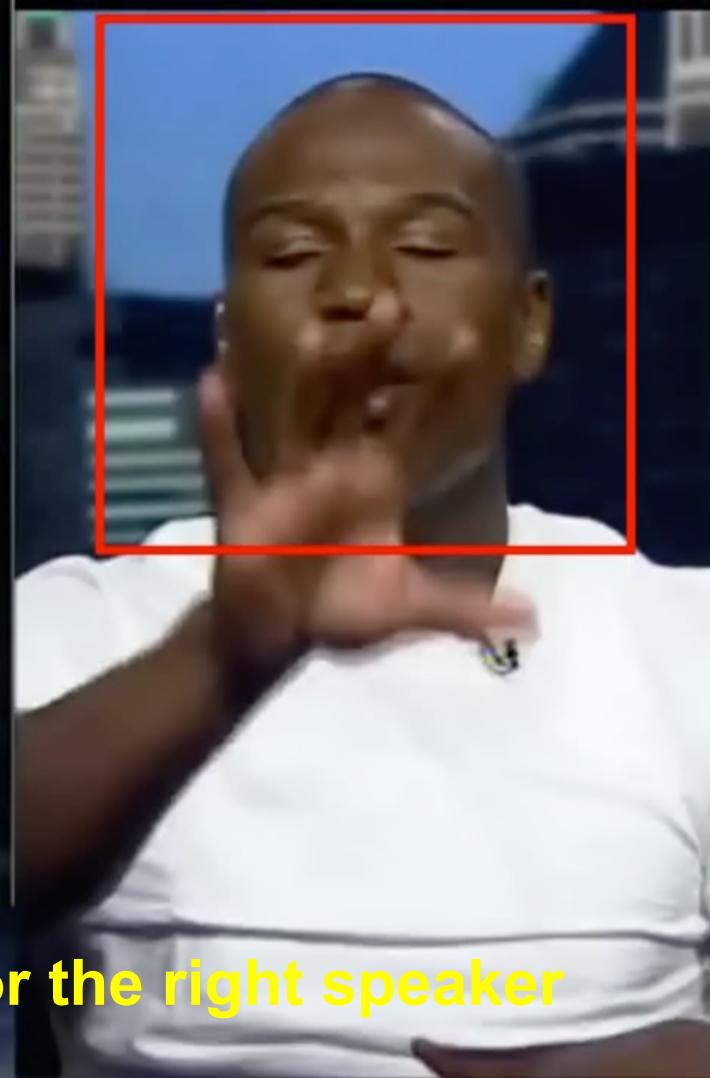
[Gao et al. ECCV 2018, Afouras et al. Interspeech'18, Gabby et al. Interspeech'18, Owens & Efros ECCV'18, Ephrat et al. SIGGRAPH'18, Zhao et al. ECCV 2018, Gao & Grauman ICCV 2019, Zhao et al. ICCV 2019, Xu et al. ICCV 2019, Gan et al. CVPR 2020, Gao et al. CVPR 2021]



Speech mixture



Separated voice for the left speaker



Separated voice for the right speaker

Musical instruments source separation

Train on 100,000 unlabeled multi-source video clips,
then separate audio for novel video.



original video
(before separation)

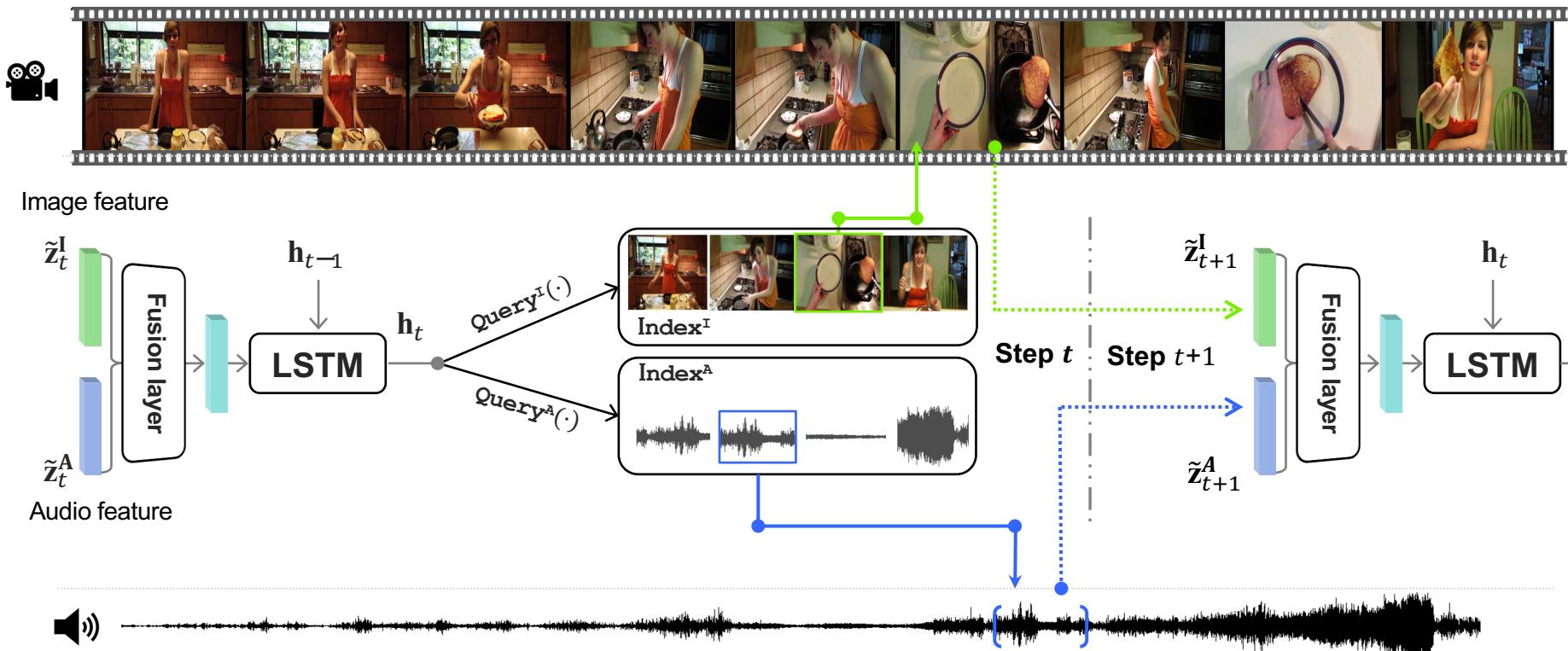
object detections:
violin & flute

Gao & Grauman, Co-Separating Sounds of Visual Objects, ICCV 2019

Fei-Fei Li, Jiajun Wu, Ruohan Gao

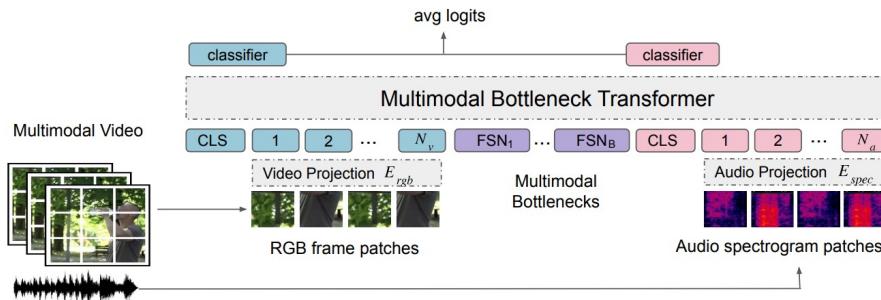
Lecture 12 - 102 May 5, 2022

Audio as a preview mechanism for efficient action recognition in videos

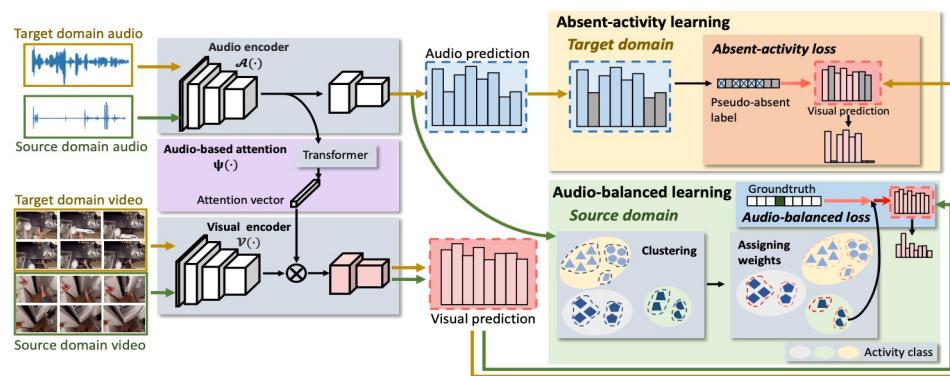


Gao et al., Listen to Look: Action Recognition by Previewing Audio, CVPR 2020

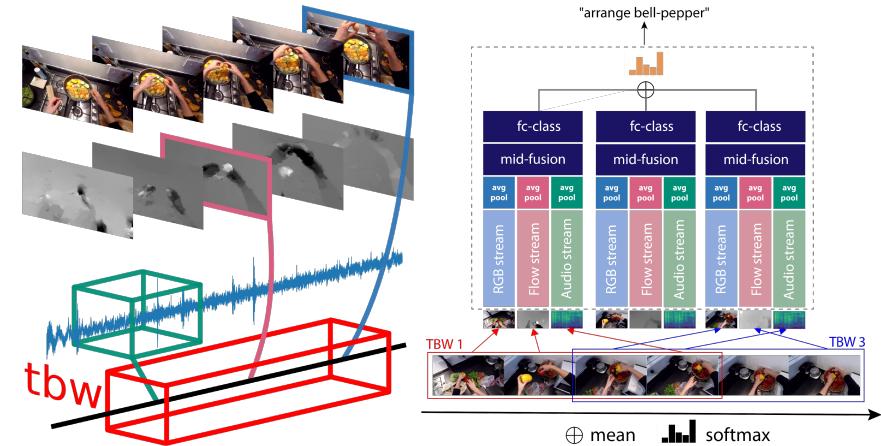
Multimodal Video Understanding



Attention Bottlenecks for Multimodal Fusion, Nagrani et al. NeurIPS 2021

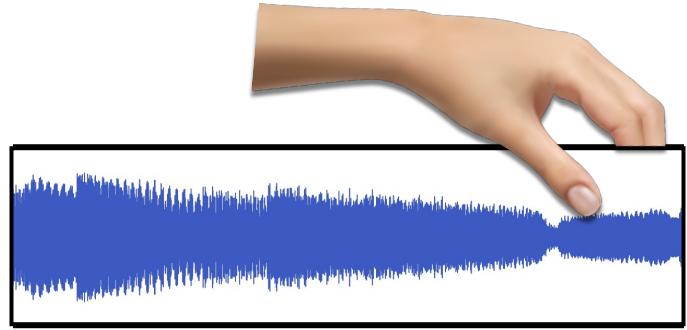


Audio-Adaptive Activity Recognition Across Video Domains, Yunhua et al. CVPR 2022



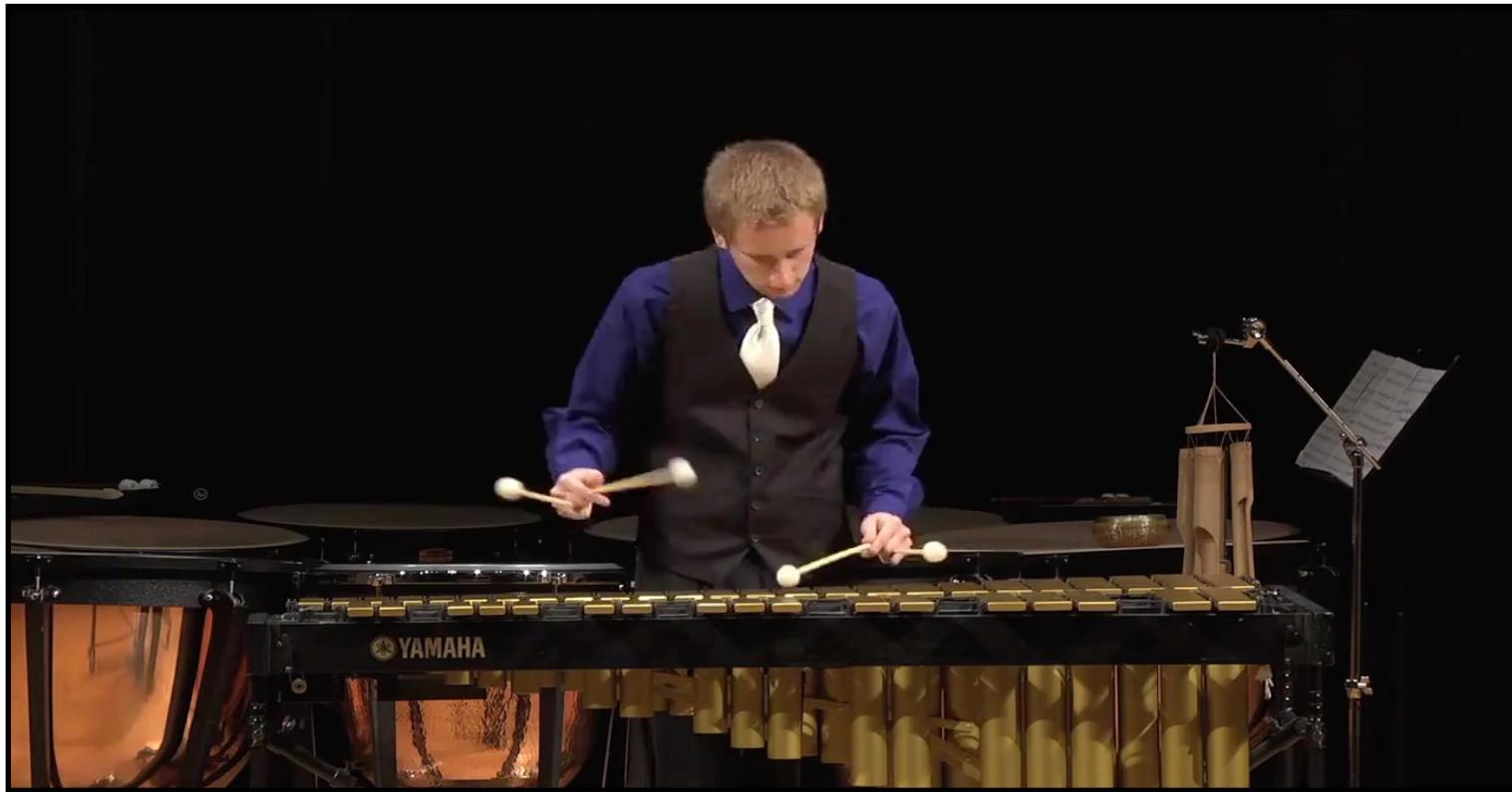
EPIC-Fusion: Audio-Visual Temporal Binding for Egocentric Action Recognition, Kazakos et al., ICCV 2019

Learning audio-visual synchronization



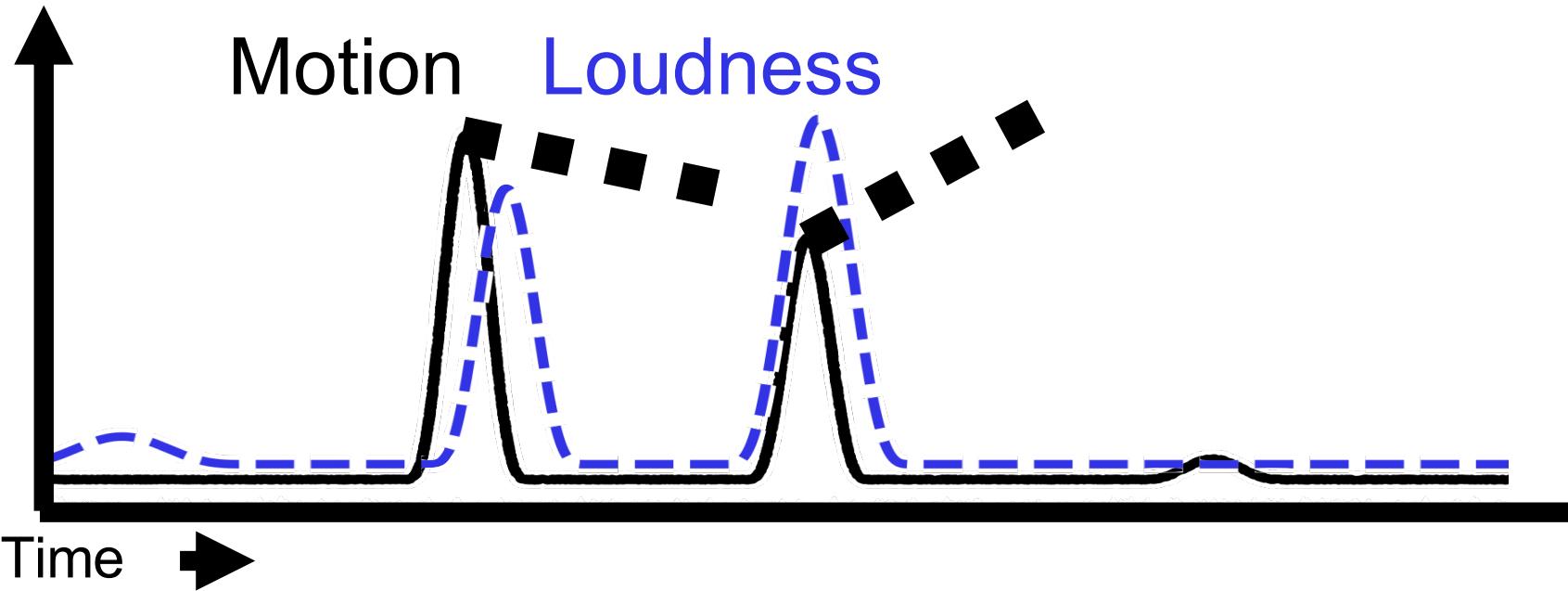
Owens & Efros, *Audio-visual scene analysis with self-supervised multisensory features*, ECCV 2018
Korbar et al., *Co-training of audio and video representations from self-supervised temporal synchronization*, NeurIPS 2018

Learning audio-visual synchronization



Owens & Efros, *Audio-visual scene analysis with self-supervised multisensory features*, ECCV 2018

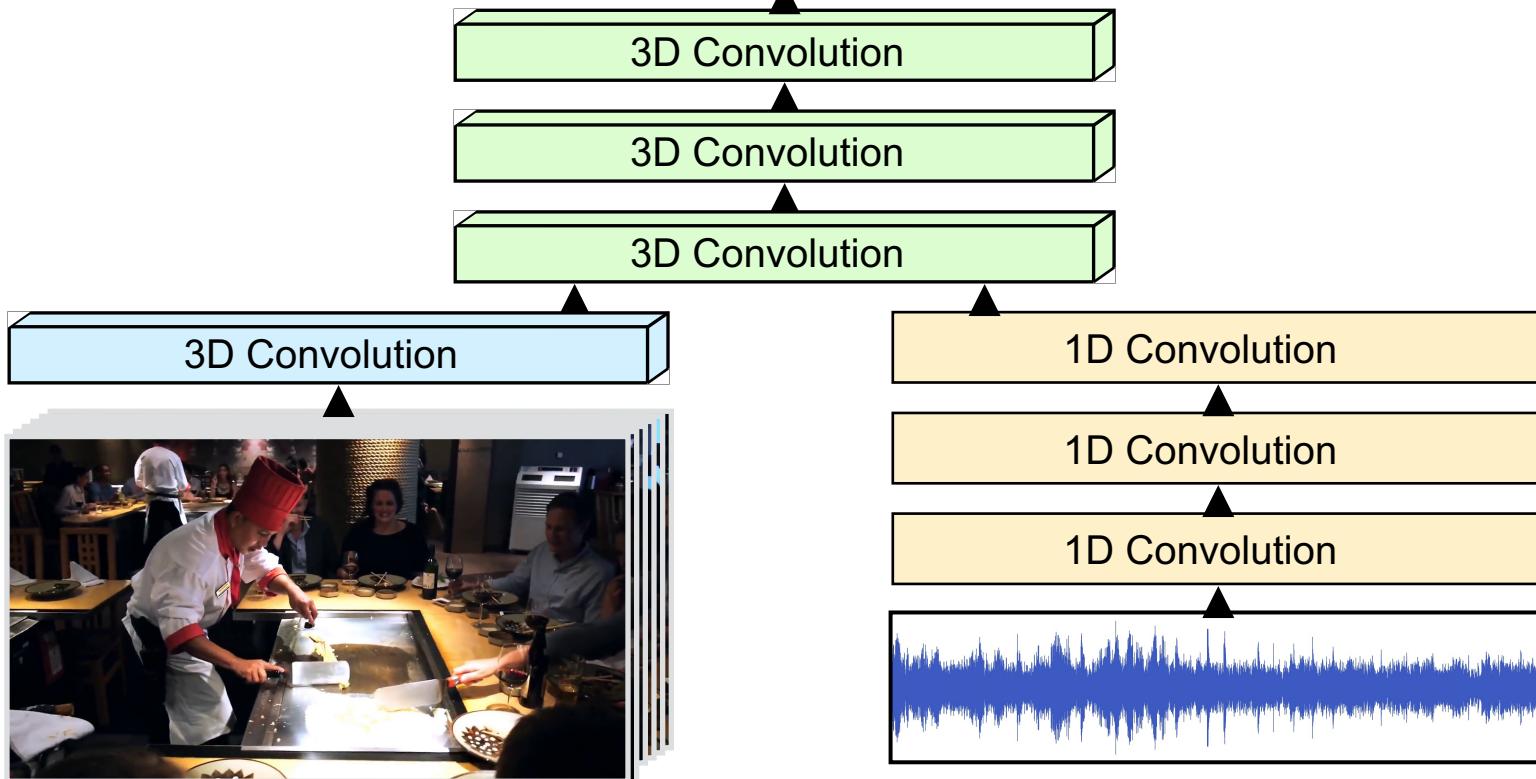
Learning audio-visual synchronization



Slide Credit: Andrew Owens

Learning audio-visual synchronization

Aligned vs. misaligned



Owens & Efros, *Audio-visual scene analysis with self-supervised multisensory features*, ECCV 2018

Top responses in test set



Owens & Efros, *Audio-visual scene analysis with self-supervised multisensory features*, ECCV 2018

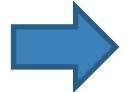
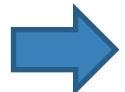
Sound source localization

Top responses per category
(speech examples omitted)



Owens & Efros, *Audio-visual scene analysis with self-supervised multisensory features*, ECCV 2018
Arandjelović and Zisserman, ECCV 2018; Senocak et al. CVPR 2018; Kidron et al. CVPR 2005 ...

CS231n: Deep Learning for Computer Vision

- Deep Learning Basics (Lecture 2 – 4)
- Perceiving and Understanding the Visual World (Lecture 5 – 12)

- Reconstructing and Interacting with the Visual World (Lecture 13 – 16)

- Human-Centered Artificial Intelligence (Lecture 17 – 18)

Next time: Generative Models

Lecture 13: Generative Models

Administrative

- A3 is out. Due May 25.
- Milestone was due May 10th
 - Read website page for milestone requirements.
 - Need to Finish data preprocessing and initial results by then.
- Midterm and A2 grades will be out this week

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.



→ Cat

Classification

This image is CC0 public domain

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.



A cat sitting on a suitcase on the floor

Image captioning

Caption generated using [neuraltalk2](#).
[Image](#) is CC0 Public domain.

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.



DOG, DOG, CAT

Object Detection

This image is CC0 public domain

Supervised vs Unsupervised Learning

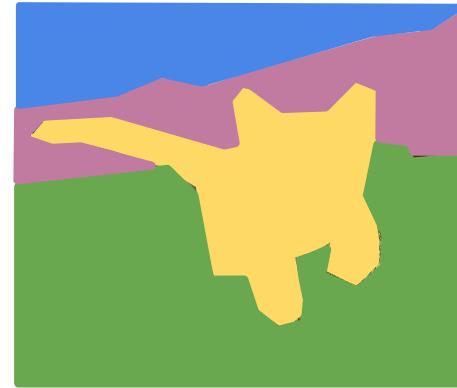
Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.



GRASS, CAT,
TREE, SKY

Semantic Segmentation

Supervised vs Unsupervised Learning

Unsupervised Learning

Data: x

Just data, **no labels!**

Goal: Learn some underlying
hidden *structure* of the data

Examples: Clustering,
dimensionality reduction, feature
learning, density estimation, etc.

Supervised vs Unsupervised Learning

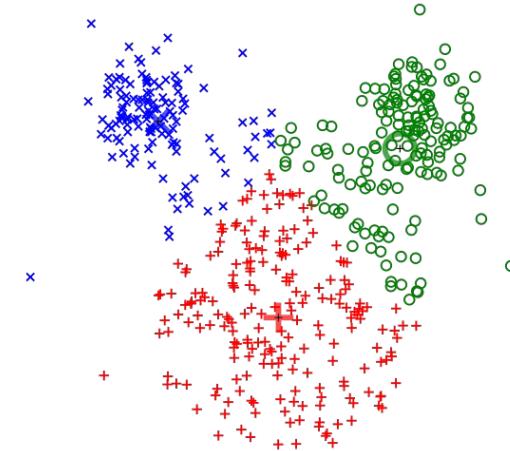
Unsupervised Learning

Data: x

Just data, no labels!

Goal: Learn some underlying
hidden *structure* of the data

Examples: Clustering,
dimensionality reduction, density
estimation, etc.



K-means clustering

This image is CC0 public domain

Supervised vs Unsupervised Learning

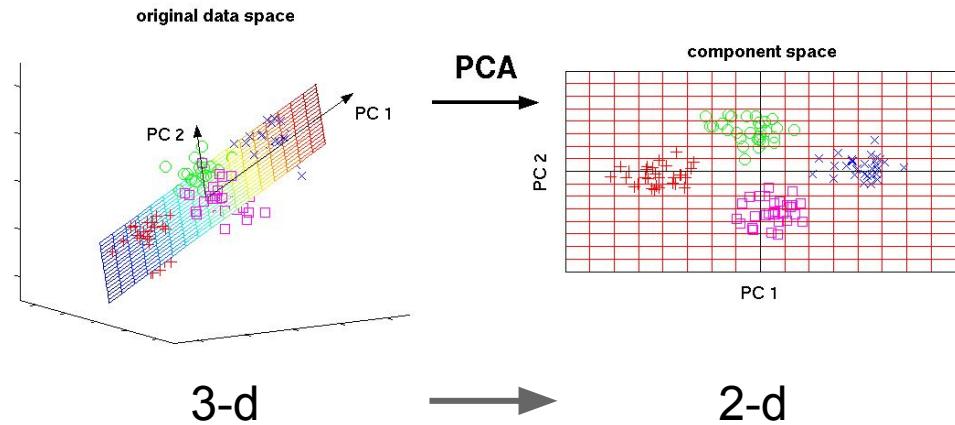
Unsupervised Learning

Data: x

Just data, no labels!

Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, density estimation, etc.



Principal Component Analysis
(Dimensionality reduction)

This image from Matthias Scholz
is CC0 public domain

Supervised vs Unsupervised Learning

Unsupervised Learning

Data: x

Just data, no labels!

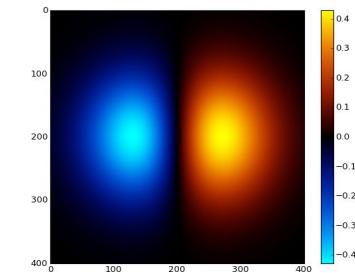
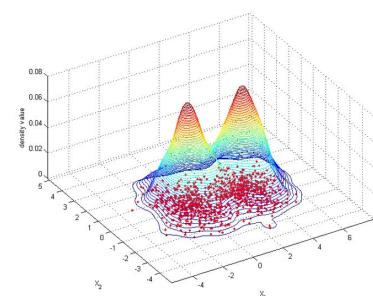
Goal: Learn some underlying hidden *structure* of the data

Examples: Clustering, dimensionality reduction, density estimation, etc.



Figure copyright Ian Goodfellow, 2016. Reproduced with permission.

1-d density estimation



2-d density estimation

Modeling $p(x)$

2-d density images [left](#) and [right](#) are [CC0 public domain](#)

Supervised vs Unsupervised Learning

Supervised Learning

Data: (x, y)

x is data, y is label

Goal: Learn a *function* to map $x \rightarrow y$

Examples: Classification,
regression, object detection,
semantic segmentation, image
captioning, etc.

Unsupervised Learning

Data: x

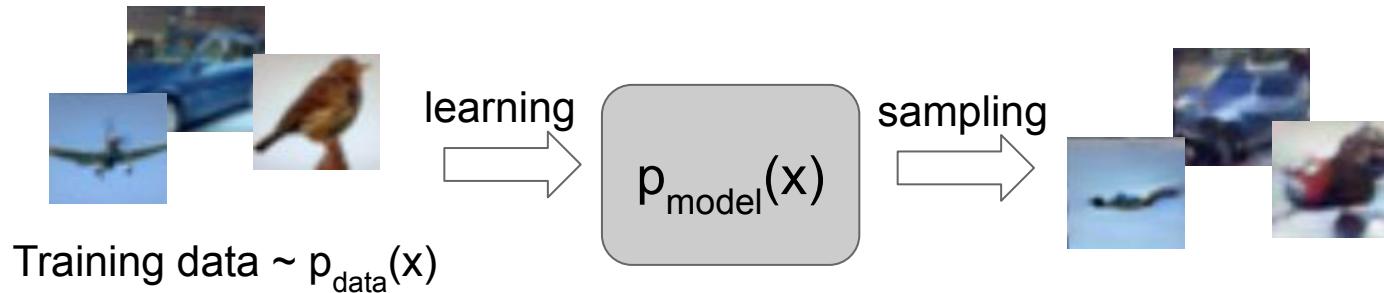
Just data, no labels!

Goal: Learn some underlying
hidden *structure* of the data

Examples: Clustering,
dimensionality reduction, density
estimation, etc.

Generative Modeling

Given training data, generate new samples from same distribution

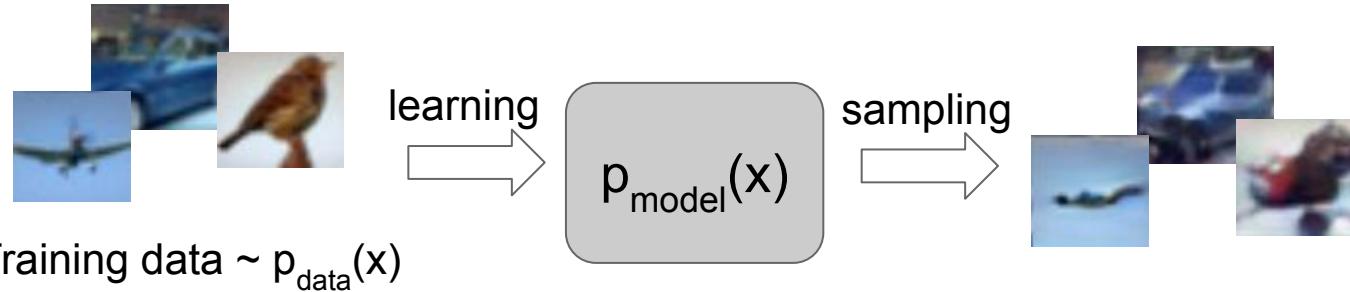


Objectives:

1. Learn $p_{\text{model}}(x)$ that approximates $p_{\text{data}}(x)$
2. **Sampling new x from $p_{\text{model}}(x)$**

Generative Modeling

Given training data, generate new samples from same distribution



Formulate as density estimation problems:

- **Explicit density estimation:** explicitly define and solve for $p_{\text{model}}(x)$
- **Implicit density estimation:** learn model that can sample from $p_{\text{model}}(x)$ **without explicitly defining it.**

Why Generative Models?



- Realistic samples for artwork, super-resolution, colorization, etc.
- Learn useful features for downstream tasks such as classification.
- Getting insights from high-dimensional data (physics, medical imaging, etc.)
- Modeling physical world for simulation and planning (robotics and reinforcement learning applications)
- Many more ...

Figures from L-R are copyright: (1) [Alec Radford et al. 2016](#); (2) [Philip Isola et al. 2017](#). Reproduced with authors permission (3) [BAIR Blog](#).

Taxonomy of Generative Models

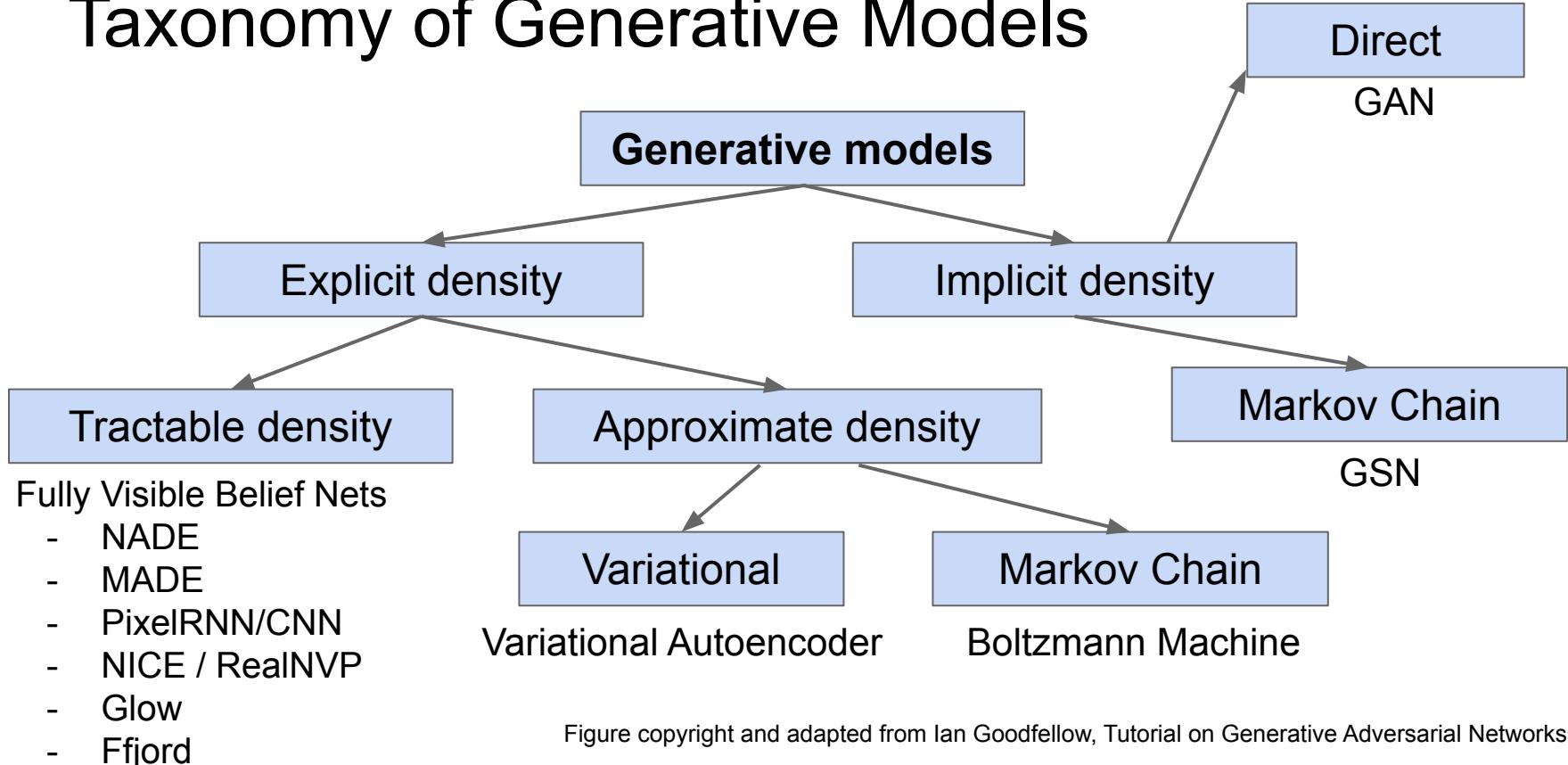


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Taxonomy of Generative Models

Today: discuss 3 most popular types of generative models today

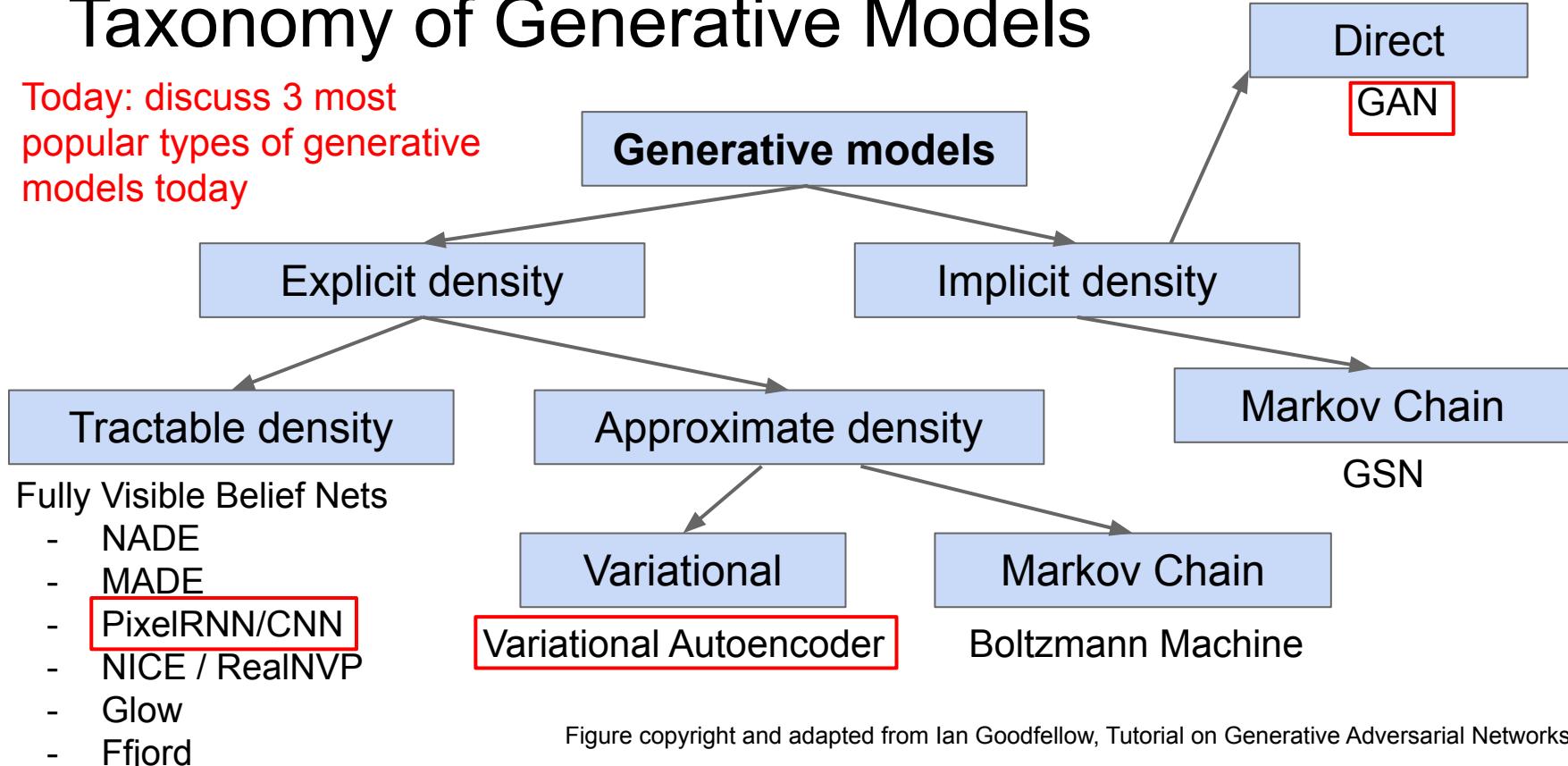


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

PixelRNN and PixelCNN

(A very brief overview)

Fully visible belief network (FVBN)

Explicit density model

$$p(x) = p(x_1, x_2, \dots, x_n)$$



Likelihood of
image x



Joint likelihood of each
pixel in the image

Fully visible belief network (FVBN)

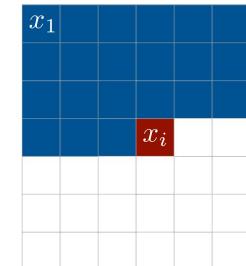
Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^n p(x_i|x_1, \dots, x_{i-1})$$

↑ ↑

Likelihood of image x Probability of i 'th pixel value given all previous pixels



Then maximize likelihood of training data

Fully visible belief network (FVBN)

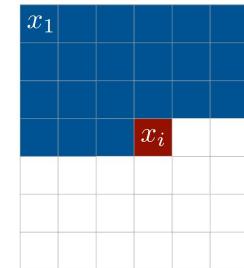
Explicit density model

Use chain rule to decompose likelihood of an image x into product of 1-d distributions:

$$p(x) = \prod_{i=1}^n p(x_i|x_1, \dots, x_{i-1})$$

↑
Likelihood of
image x

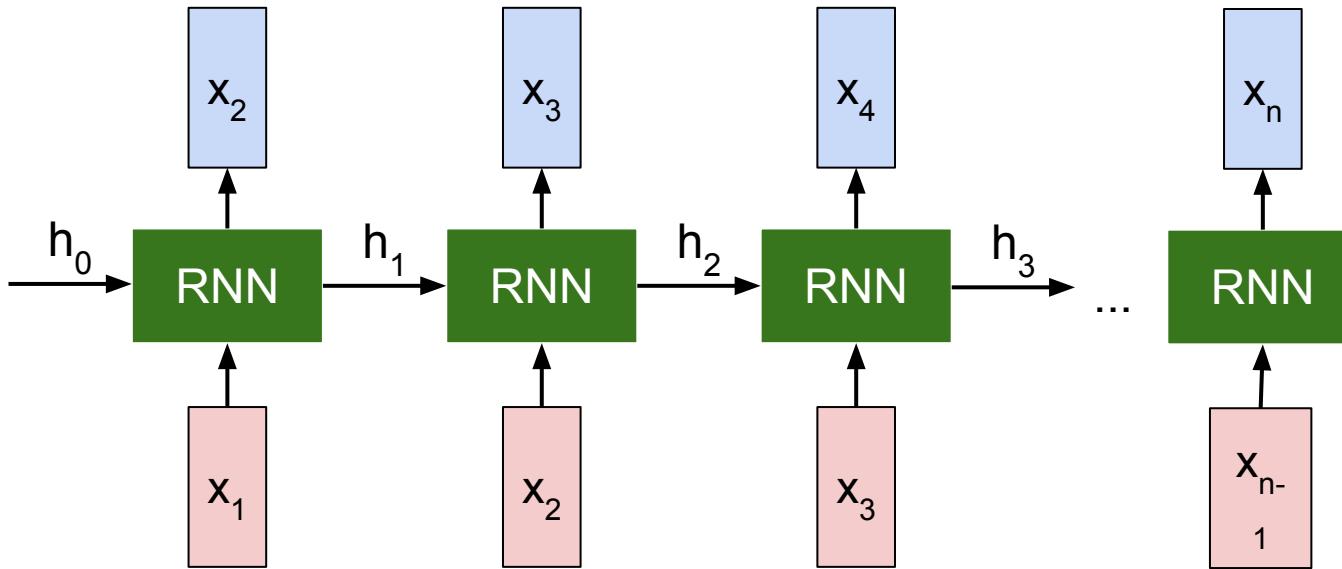
↑
Probability of i 'th pixel value
given all previous pixels



Complex distribution over pixel
values => Express using a neural
network!

Then maximize likelihood of training data

Recurrent Neural Network

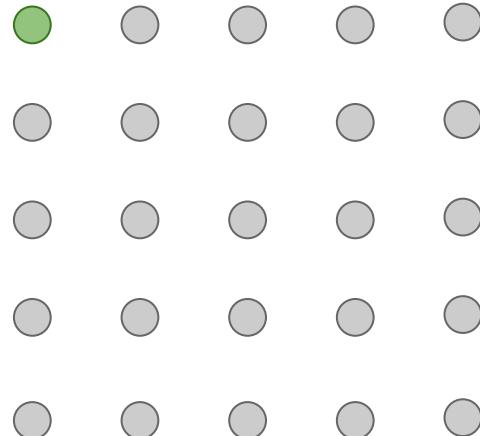


$$p(x_i | x_1, \dots, x_{i-1})$$

PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner



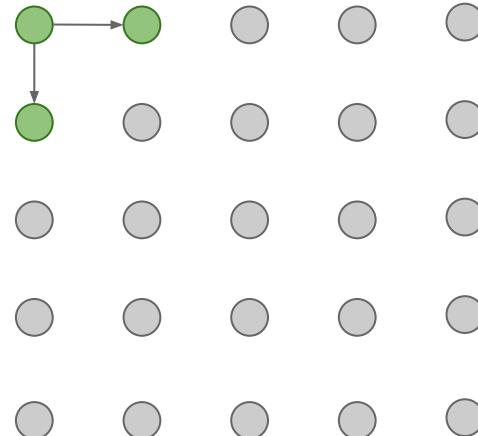
Dependency on previous pixels modeled
using an RNN (LSTM)

PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled
using an RNN (LSTM)

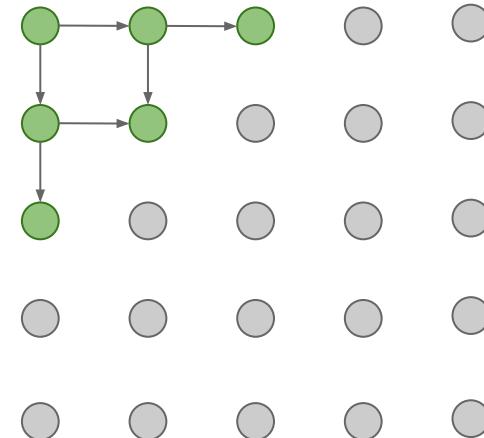


PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled
using an RNN (LSTM)



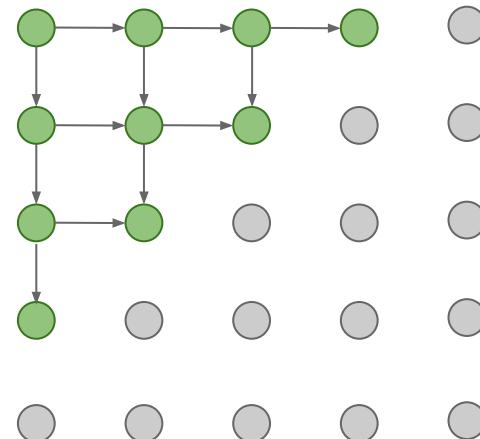
PixelRNN

[van der Oord et al. 2016]

Generate image pixels starting from corner

Dependency on previous pixels modeled using an RNN (LSTM)

Drawback: sequential generation is slow in both training and inference!



PixelCNN

[van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region
(masked convolution)

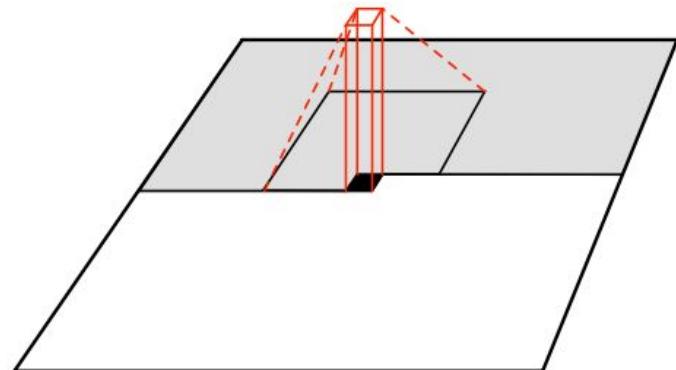


Figure copyright van der Oord et al., 2016. Reproduced with permission.

PixelCNN

[van der Oord et al. 2016]

Still generate image pixels starting from corner

Dependency on previous pixels now modeled using a CNN over context region (masked convolution)

Training is faster than PixelRNN
(can parallelize convolutions since context region values known from training images)

Generation is still slow:

For a 32x32 image, we need to do forward passes of the network 1024 times for a single image

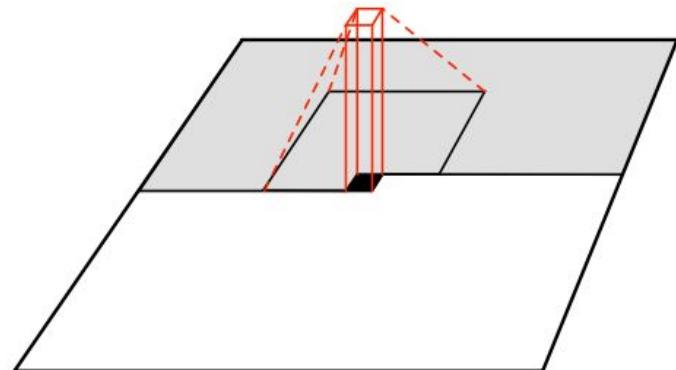
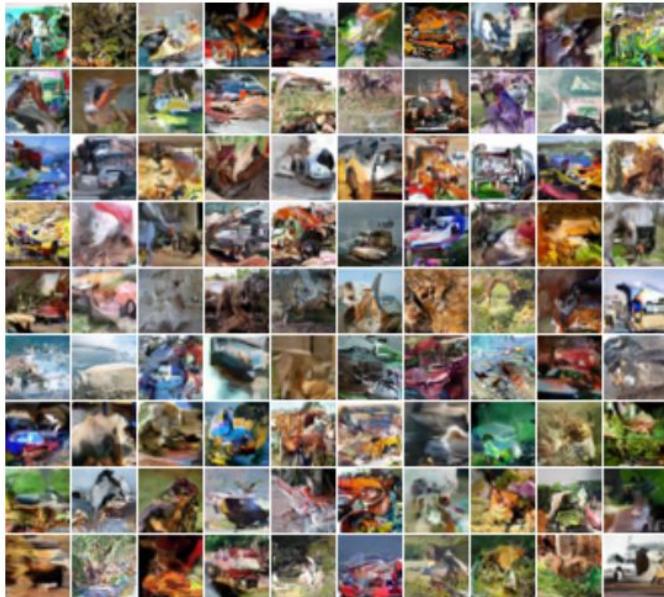
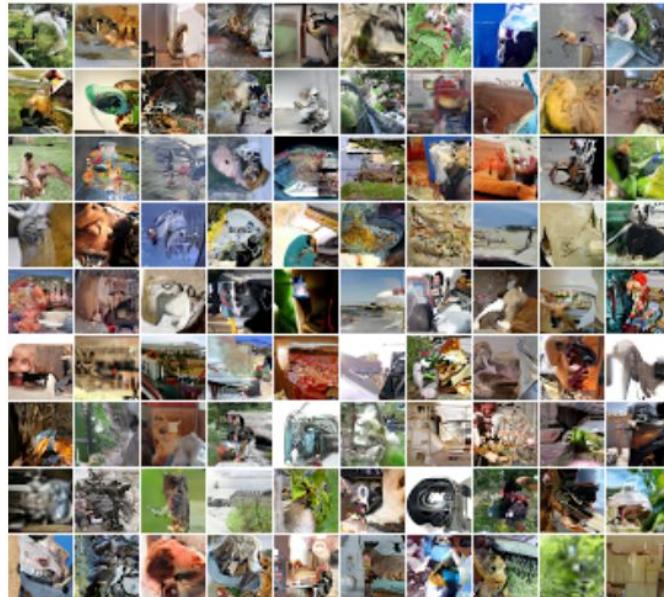


Figure copyright van der Oord et al., 2016. Reproduced with permission.

Generation Samples



32x32 CIFAR-10



32x32 ImageNet

Figures copyright Aaron van der Oord et al., 2016. Reproduced with permission.

PixelRNN and PixelCNN

Pros:

- Can explicitly compute likelihood $p(x)$
- Easy to optimize
- Good samples

Con:

- Sequential generation => slow

Improving PixelCNN performance

- Gated convolutional layers
- Short-cut connections
- Discretized logistic loss
- Multi-scale
- Training tricks
- Etc...

See

- Van der Oord et al. NIPS 2016
- Salimans et al. 2017
(PixelCNN++)

Taxonomy of Generative Models

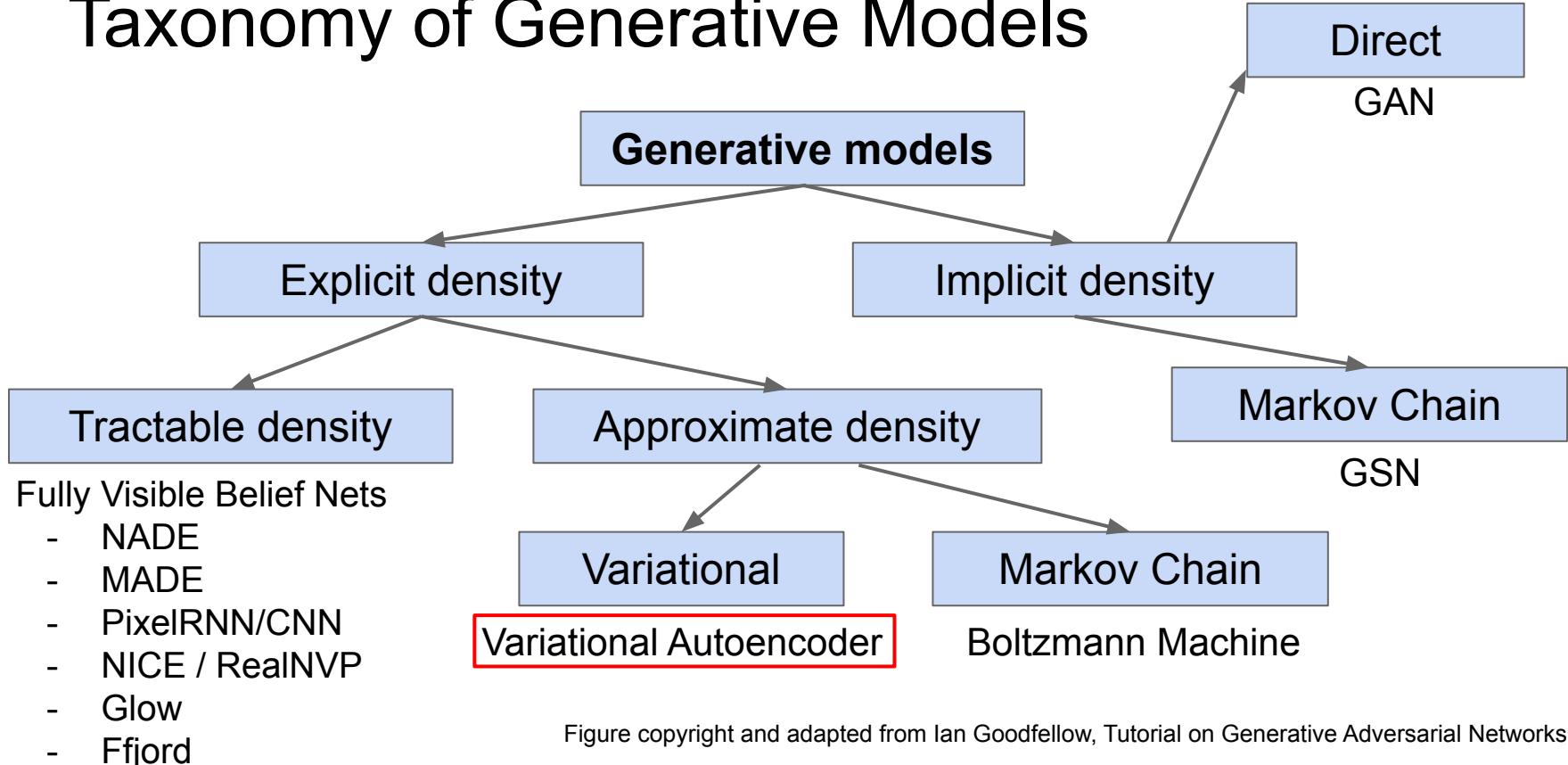


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Variational Autoencoders (VAE)

So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i|x_1, \dots, x_{i-1})$$

So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i|x_1, \dots, x_{i-1})$$

Variational Autoencoders (VAEs) define intractable density function with latent \mathbf{z} :

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

No dependencies among pixels, can generate all pixels at the same time!

Cannot optimize directly, derive and optimize lower bound on likelihood instead

So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i|x_1, \dots, x_{i-1})$$

Variational Autoencoders (VAEs) define intractable density function with latent \mathbf{z} :

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

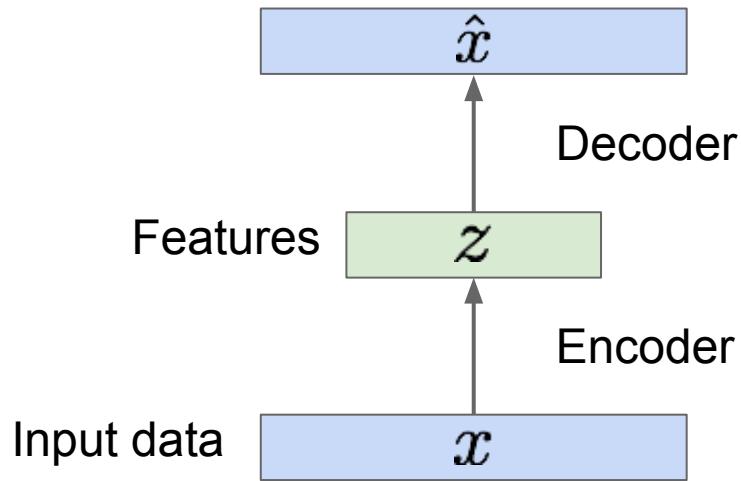
No dependencies among pixels, can generate all pixels at the same time!

Cannot optimize directly, derive and optimize lower bound on likelihood instead

Why latent \mathbf{z} ?

Some background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

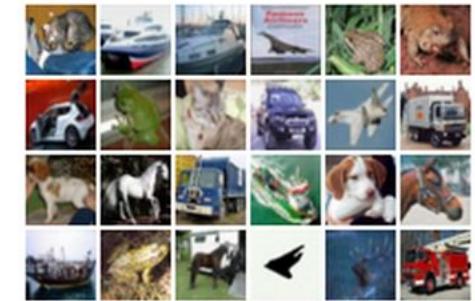
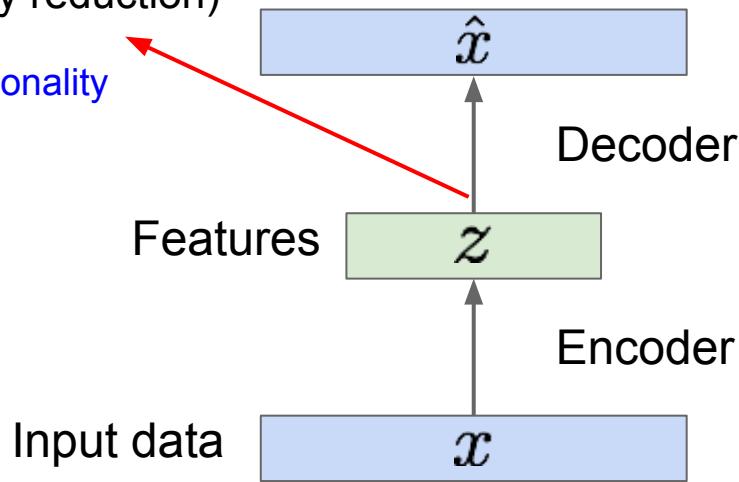


Some background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?



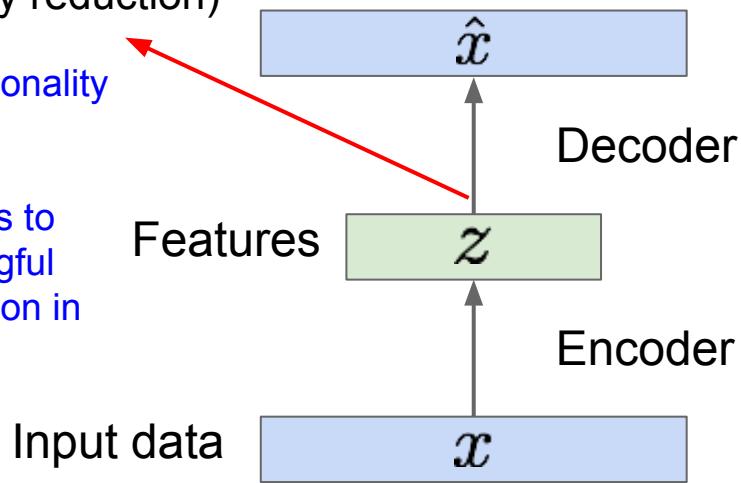
Some background first: Autoencoders

Unsupervised approach for learning a lower-dimensional feature representation from unlabeled training data

z usually smaller than x
(dimensionality reduction)

Q: Why dimensionality reduction?

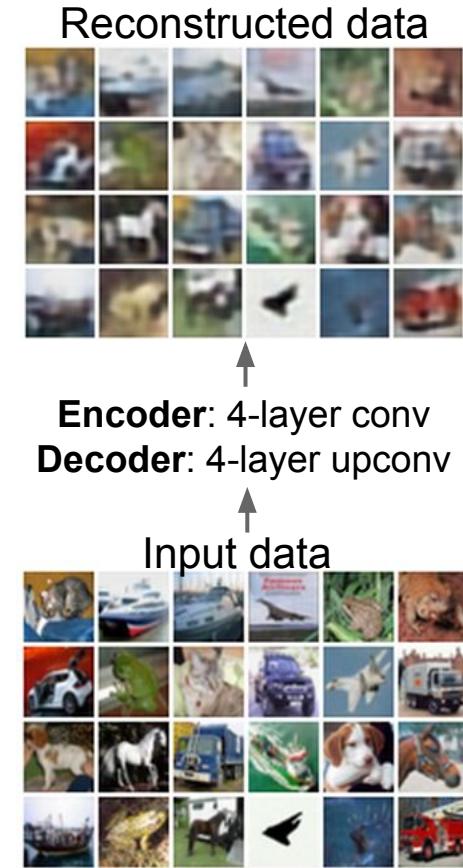
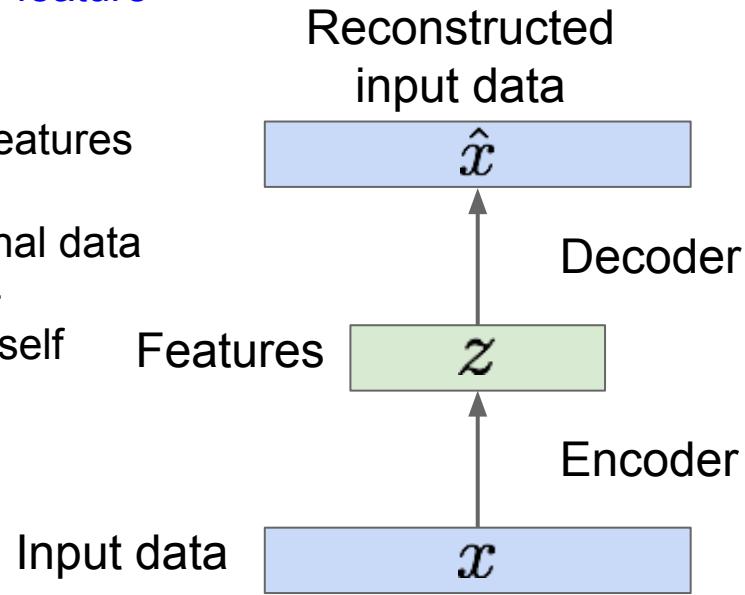
A: Want features to capture meaningful factors of variation in data



Some background first: Autoencoders

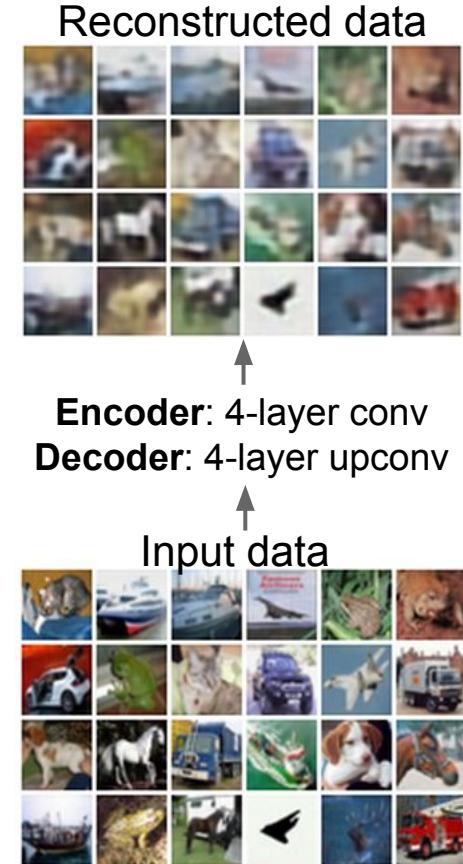
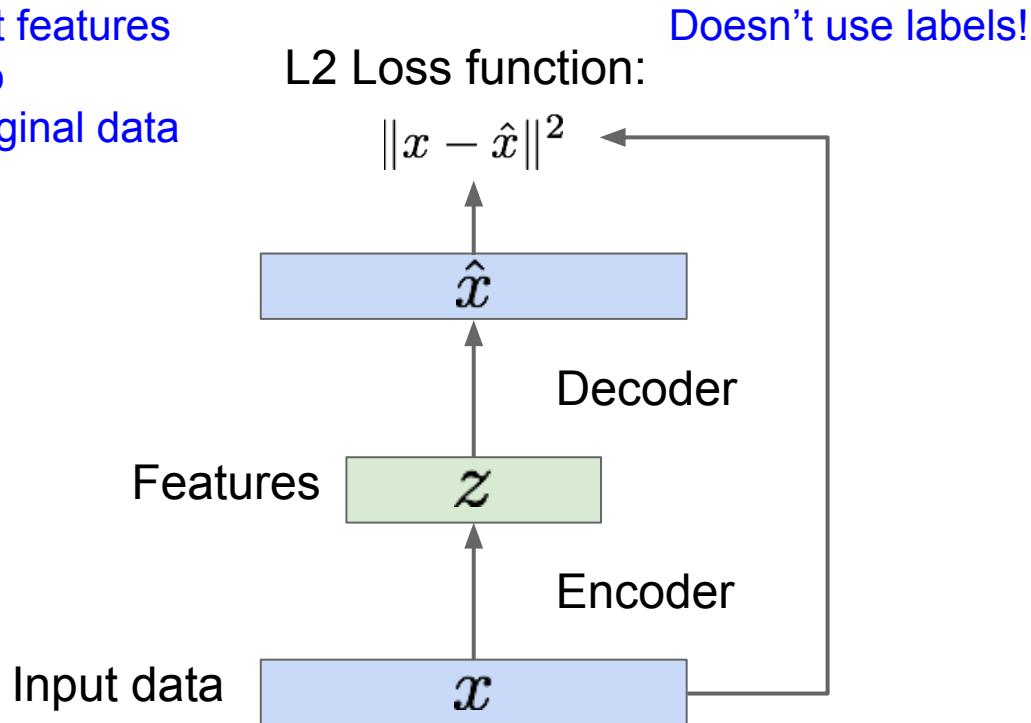
How to learn this feature representation?

Train such that features can be used to reconstruct original data
“Autoencoding” - encoding input itself

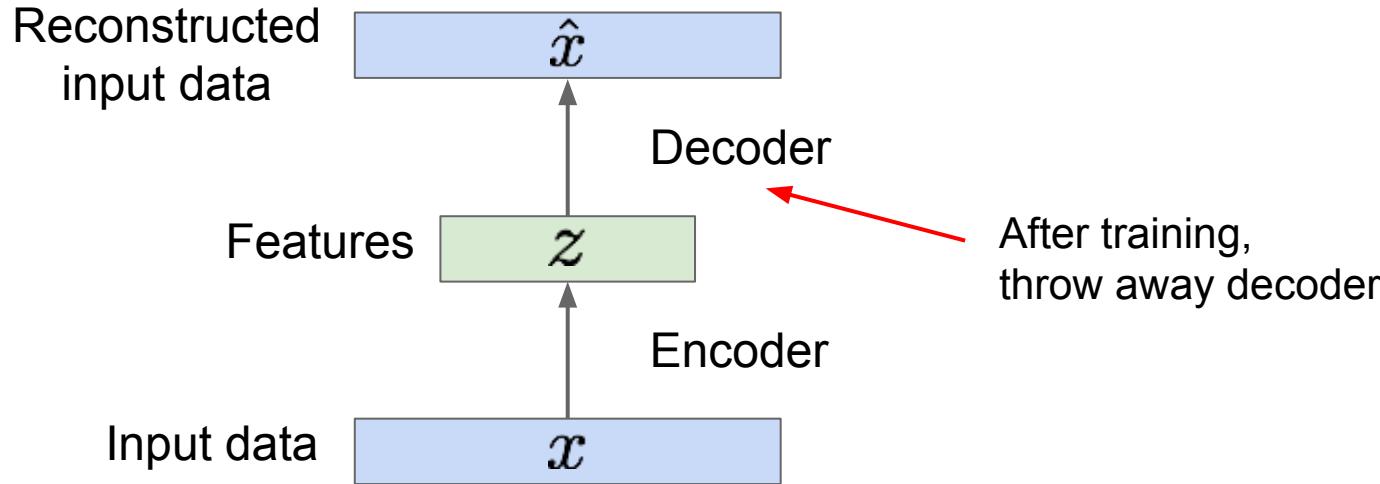


Some background first: Autoencoders

Train such that features
can be used to
reconstruct original data

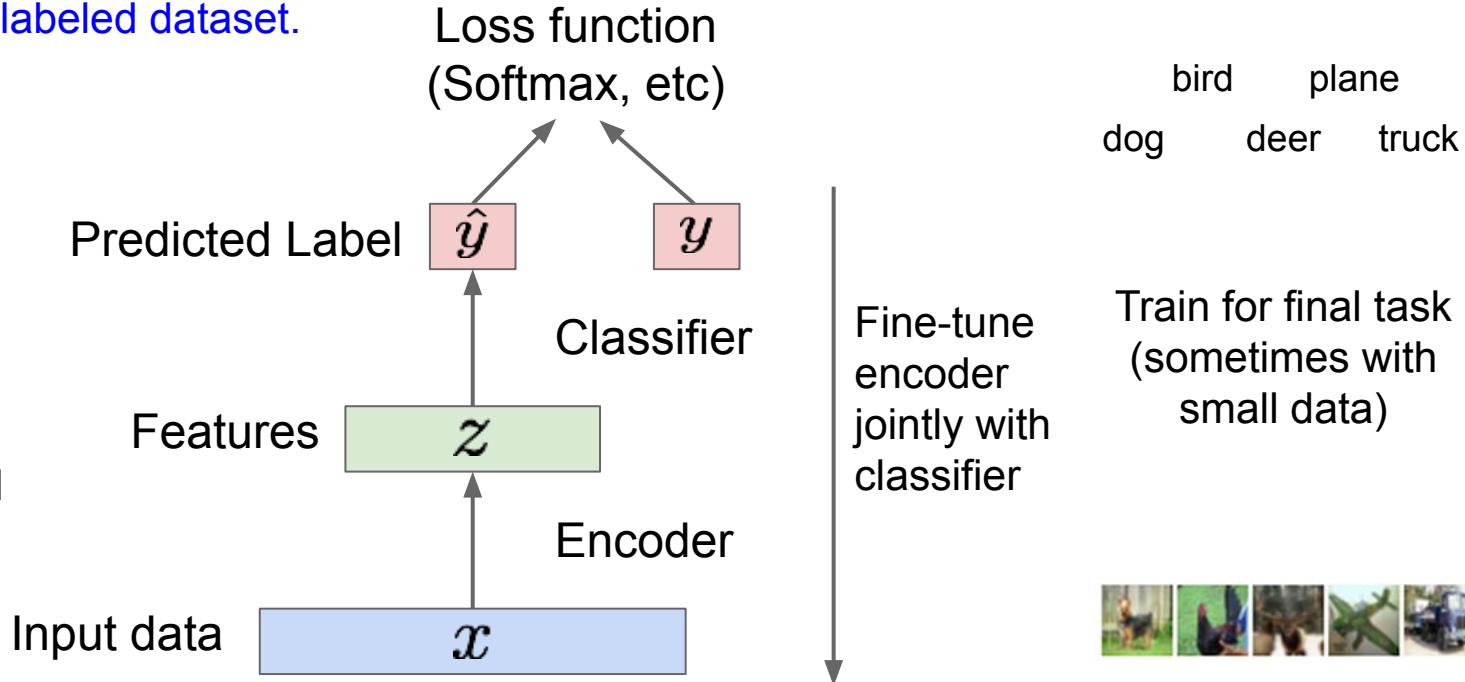


Some background first: Autoencoders

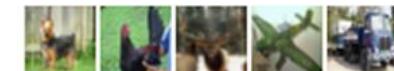


Some background first: Autoencoders

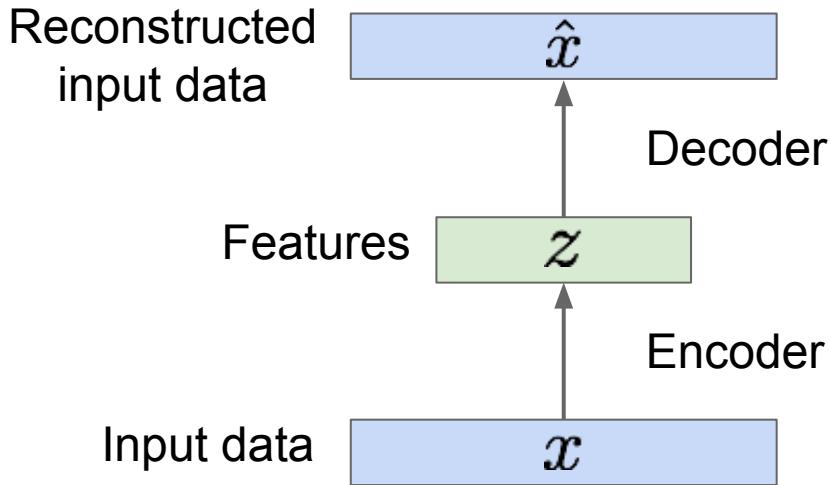
Transfer from large, unlabeled dataset to small, labeled dataset.



Encoder can be used to initialize a **supervised** model



Some background first: Autoencoders



Autoencoders can reconstruct data, and can learn features to initialize a supervised model

Features capture factors of variation in training data.

But we can't generate new images from an autoencoder because we don't know the space of z .

How do we make autoencoder a generative model?

Variational Autoencoders

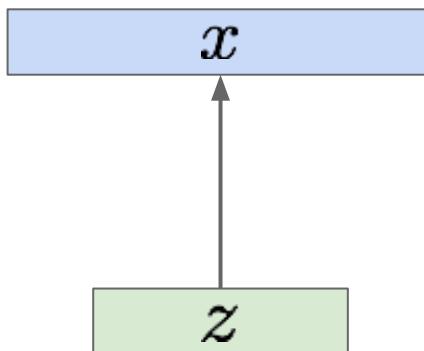
Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data $\{x^{(i)}\}_{i=1}^N$ is generated from the distribution of unobserved (latent) representation z

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$



Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$

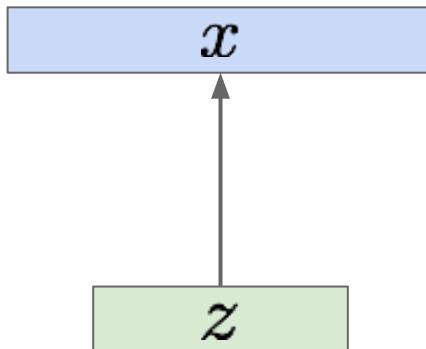
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

Probabilistic spin on autoencoders - will let us sample from the model to generate data!

Assume training data $\{x^{(i)}\}_{i=1}^N$ is generated from the distribution of unobserved (latent) representation z

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$



Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$

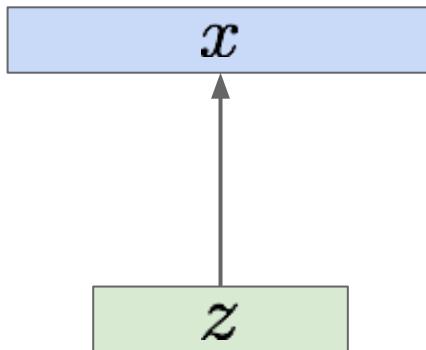
Intuition (remember from autoencoders!):
 x is an image, z is latent factors used to
generate x : attributes, orientation, etc.

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

We want to estimate the true parameters θ^* of this generative model given training data x .

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$

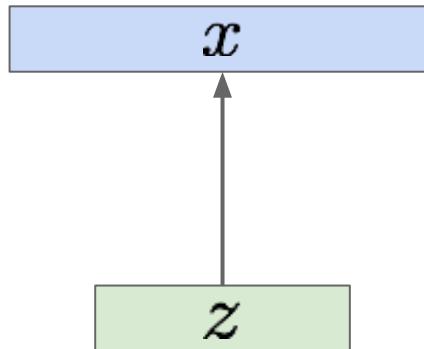


Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$



Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$

We want to estimate the true parameters θ^* of this generative model given training data x .

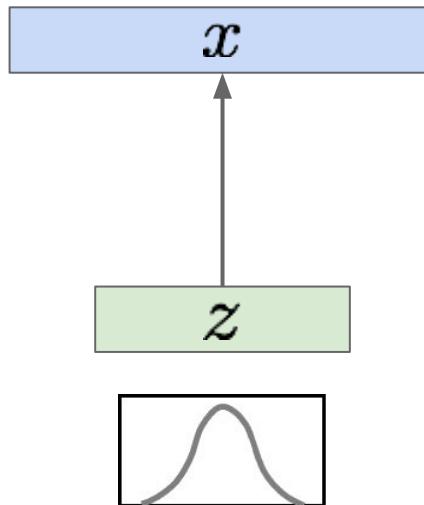
How should we represent this model?

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders

Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$

Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x .

How should we represent this model?

Choose prior $p(z)$ to be simple, e.g.
Gaussian. Reasonable for latent attributes,
e.g. pose, how much smile.

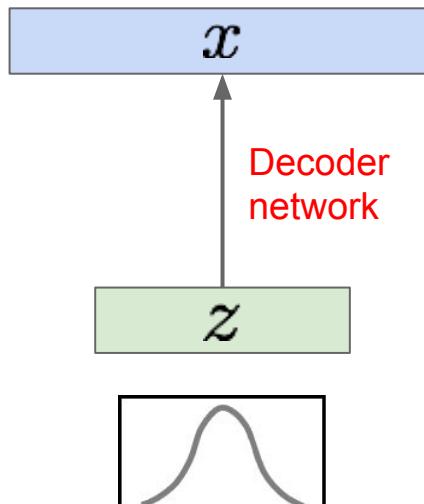
Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders



Sample from
true conditional
 $p_{\theta^*}(x \mid z^{(i)})$

Sample from
true prior
 $z^{(i)} \sim p_{\theta^*}(z)$



We want to estimate the true parameters θ^* of this generative model given training data x .

How should we represent this model?

Choose prior $p(z)$ to be simple, e.g. Gaussian. Reasonable for latent attributes, e.g. pose, how much smile.

Conditional $p(x|z)$ is complex (generates image) => represent with neural network

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

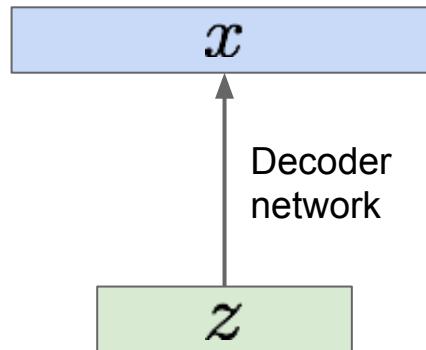
Variational Autoencoders

Sample from
true conditional

$$p_{\theta^*}(x \mid z^{(i)})$$

Sample from
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



We want to estimate the true parameters θ^* of this generative model given training data x .

How to train the model?

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

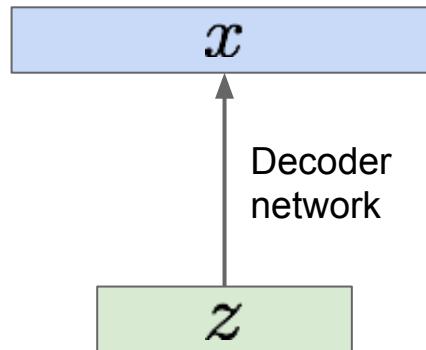
Variational Autoencoders

Sample from
true conditional

$$p_{\theta^*}(x \mid z^{(i)})$$

Sample from
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



We want to estimate the true parameters θ^* of this generative model given training data x .

How to train the model?

Learn model parameters to maximize likelihood of training data

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

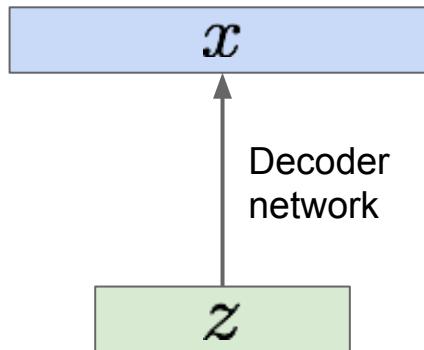
Variational Autoencoders

Sample from
true conditional

$$p_{\theta^*}(x \mid z^{(i)})$$

Sample from
true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



We want to estimate the true parameters θ^* of this generative model given training data x .

How to train the model?

Learn model parameters to maximize likelihood of training data

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Q: What is the problem with this?

Intractable!

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

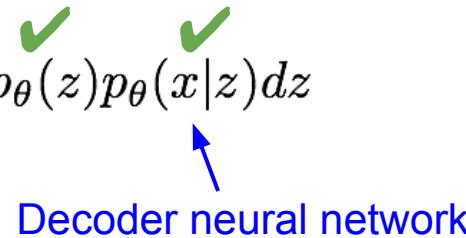
Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

↑
Simple Gaussian prior

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$



Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$



Intractable to compute $p(x|z)$ for every z !

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Intractability

Data likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$



Intractable to compute $p(x|z)$ for every z !

$$\log p(x) \approx \log \frac{1}{k} \sum_{i=1}^k p(x|z^{(i)}), \text{ where } z^{(i)} \sim p(z)$$

Monte Carlo estimation is too high variance

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Intractability

Data likelihood: $p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$

Posterior density: $p_{\theta}(z|x) = p_{\theta}(x|z)p_{\theta}(z)/p_{\theta}(x)$

Intractable data likelihood

Variational Autoencoders: Intractability

Data likelihood: $p_\theta(x) = \int p_\theta(z)p_\theta(x|z)dz$

Posterior density also intractable: $p_\theta(z|x) = p_\theta(x|z)p_\theta(z)/p_\theta(x)$

Solution: In addition to modeling $p_\theta(x|z)$, learn $q_\phi(z|x)$ that approximates the true posterior $p_\theta(z|x)$.

Will see that the approximate posterior allows us to derive a lower bound on the data likelihood that is tractable, which we can optimize.

Variational inference is to approximate the unknown posterior distribution from only the observed data x

Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} [\log p_{\theta}(x^{(i)})] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$

Variational Autoencoders

$$\log p_{\theta}(x^{(i)}) = \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} [\log p_{\theta}(x^{(i)})] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z)$$



Taking expectation wrt. z
(using encoder network) will
come in handy later

Variational Autoencoders

$$\begin{aligned}\log p_{\theta}(x^{(i)}) &= \mathbf{E}_{z \sim q_{\phi}(z|x^{(i)})} \left[\log p_{\theta}(x^{(i)}) \right] \quad (p_{\theta}(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_{\theta}(x^{(i)} | z)p_{\theta}(z)}{p_{\theta}(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})\end{aligned}$$

Variational Autoencoders

$$\begin{aligned}\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} \left[\log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant})\end{aligned}$$

Variational Autoencoders

$$\begin{aligned}\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} \left[\log p_\theta(x^{(i)}) \right] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms})\end{aligned}$$

Variational Autoencoders

$$\begin{aligned}\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z)) + D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z | x^{(i)}))\end{aligned}$$



The expectation wrt. z (using encoder network) let us write nice KL terms

Variational Autoencoders

$$\begin{aligned}\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z)) + D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z | x^{(i)}))\end{aligned}$$



Decoder network gives $p_\theta(x|z)$, can compute estimate of this term through sampling (need some trick to differentiate through sampling).



This KL term (between Gaussians for encoder and z prior) has nice closed-form solution!



$p_\theta(z|x)$ intractable (saw earlier), can't compute this KL term :(But we know KL divergence always ≥ 0 .

Variational Autoencoders

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

We want to
maximize the
data
likelihood

$$\uparrow$$
$$= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

$$= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z)) + D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z | x^{(i)}))$$

Decoder network gives $p_\theta(x|z)$, can
compute estimate of this term through
sampling.

This KL term (between
Gaussians for encoder and z
prior) has nice closed-form
solution!

$p_\theta(z|x)$ intractable (saw
earlier), can't compute this KL
term :(But we know KL
divergence always ≥ 0 .

Variational Autoencoders

We want to
maximize the
data
likelihood

$$\begin{aligned}\log p_\theta(x^{(i)}) &= \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule}) \\ &= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant}) \\ &= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms}) \\ &= \underbrace{\mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z | x^{(i)}))}_{\geq 0}\end{aligned}$$

Tractable lower bound which we can take
gradient of and optimize! ($p_\theta(x|z)$ differentiable,
KL term differentiable)

Variational Autoencoders

$$\log p_\theta(x^{(i)}) = \mathbf{E}_{z \sim q_\phi(z|x^{(i)})} [\log p_\theta(x^{(i)})] \quad (p_\theta(x^{(i)}) \text{ Does not depend on } z)$$

$$= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \right] \quad (\text{Bayes' Rule})$$

Decoder:
reconstruct
the input data

$$= \mathbf{E}_z \left[\log \frac{p_\theta(x^{(i)} | z)p_\theta(z)}{p_\theta(z | x^{(i)})} \frac{q_\phi(z | x^{(i)})}{q_\phi(z | x^{(i)})} \right] \quad (\text{Multiply by constant})$$

$$= \mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z)} \right] + \mathbf{E}_z \left[\log \frac{q_\phi(z | x^{(i)})}{p_\theta(z | x^{(i)})} \right] \quad (\text{Logarithms})$$

$$= \underbrace{\mathbf{E}_z [\log p_\theta(x^{(i)} | z)] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)} + \underbrace{D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z | x^{(i)}))}_{\geq 0}$$

Encoder:
make approximate
posterior distribution
close to prior

Tractable lower bound which we can take
gradient of and optimize! ($p_\theta(x|z)$ differentiable,
KL term differentiable)

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

Let's look at computing the KL divergence between the estimated posterior and the prior given some data

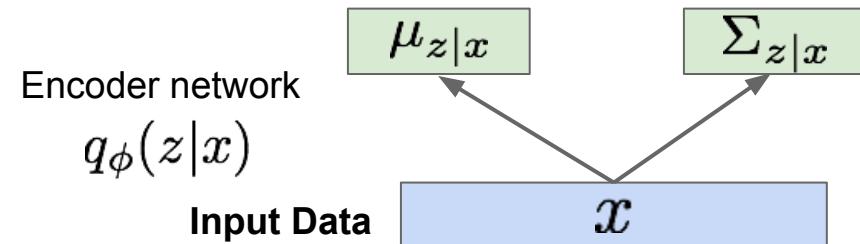
Input Data

x

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

$$D_{KL}(\mathcal{N}(\mu_{z|x}, \Sigma_{z|x}) || \mathcal{N}(0, I))$$

Have analytical solution

Make approximate posterior distribution close to prior

Encoder network

$$q_\phi(z|x)$$

Input Data

$$\mu_{z|x}$$

$$\Sigma_{z|x}$$

Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))}_{\mathcal{L}(x^{(i)}, \theta, \phi)}$$

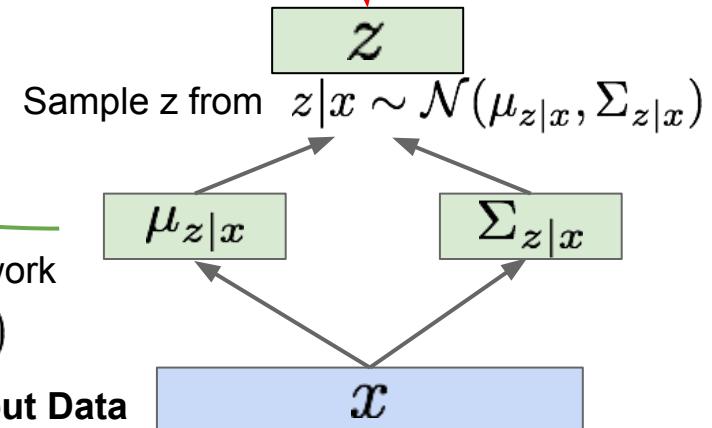
Make approximate posterior distribution close to prior

Encoder network

$$q_\phi(z|x)$$

Input Data

Not part of the computation graph!



Variational Autoencoders

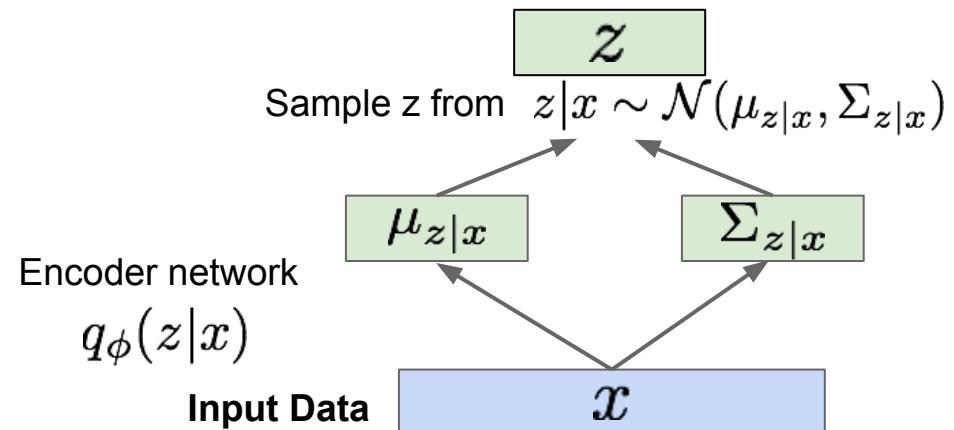
Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

Reparameterization trick to make sampling differentiable:

$$\text{Sample } \epsilon \sim \mathcal{N}(0, I)$$

$$z = \mu_{z|x} + \epsilon \sigma_{z|x}$$



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

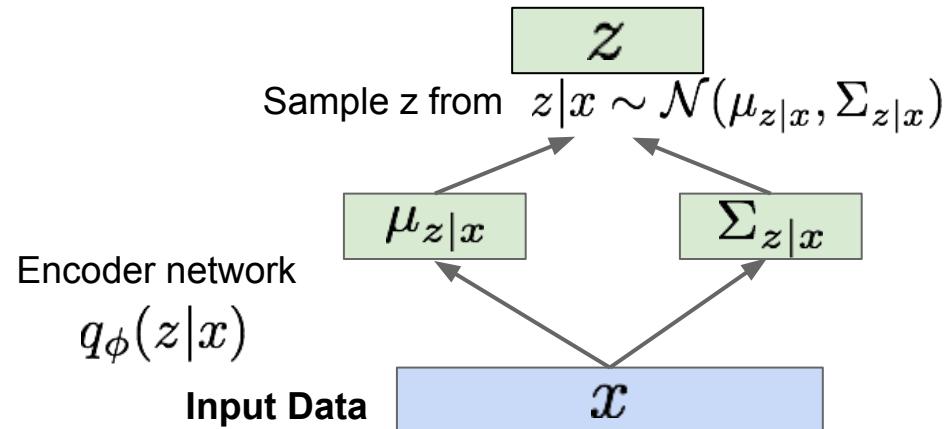
$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

Reparameterization trick to make sampling differentiable:

$$z = \mu_{z|x} + \epsilon \sigma_{z|x}$$

Sample $\epsilon \sim \mathcal{N}(0, I)$ Input to the graph

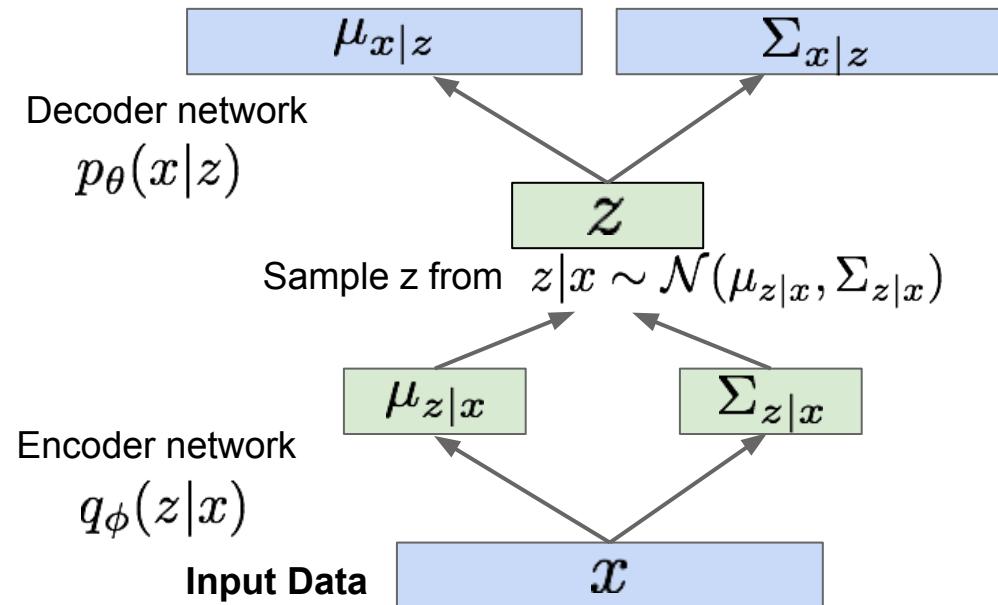
Part of computation graph



Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\underbrace{\mathbf{E}_z \left[\log p_\theta(x^{(i)} | z) \right]}_{\mathcal{L}(x^{(i)}, \theta, \phi)} - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

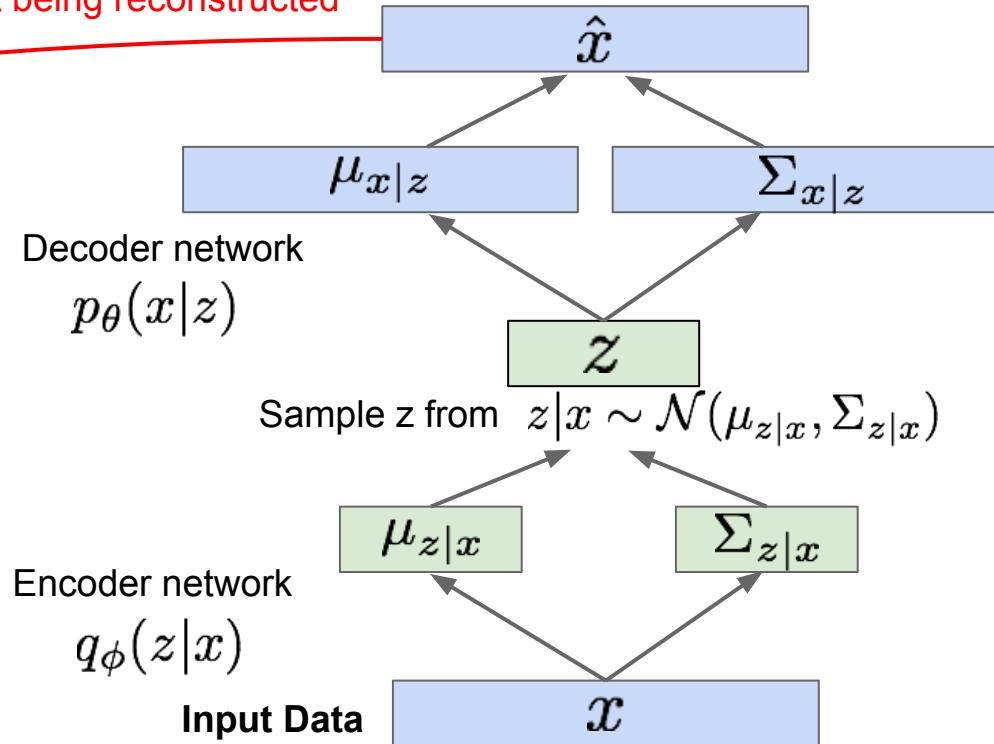


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\mathcal{L}(x^{(i)}, \theta, \phi) = \mathbb{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z))$$

Maximize likelihood of original input being reconstructed

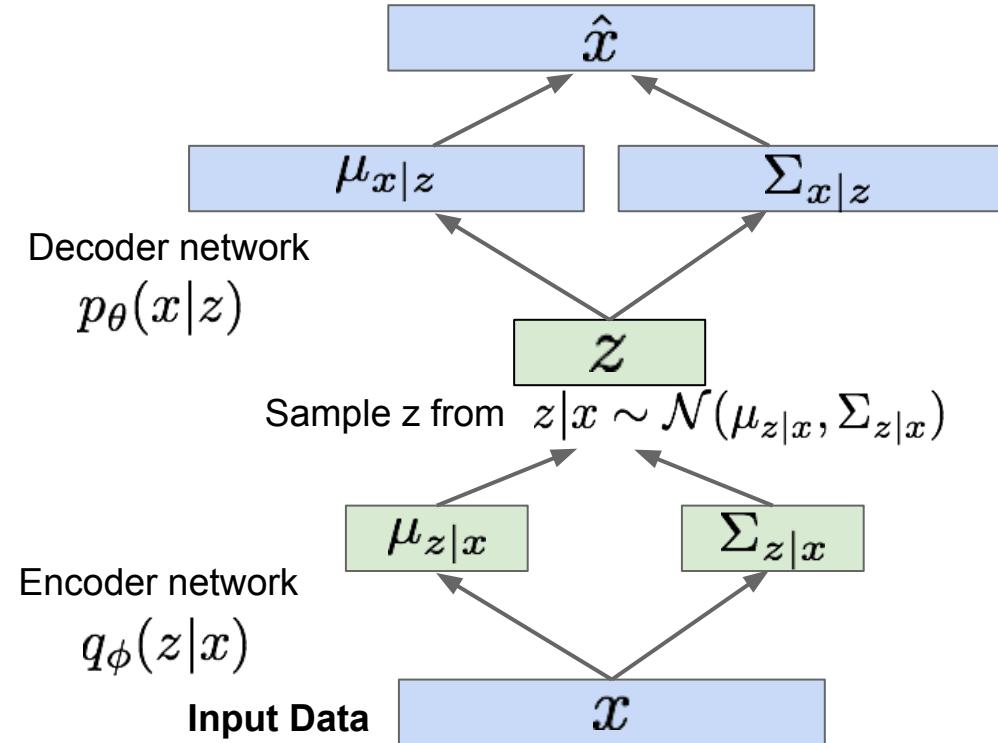


Variational Autoencoders

Putting it all together: maximizing the likelihood lower bound

$$\mathbb{E}_z \left[\log p_\theta(x^{(i)} | z) \right] - D_{KL}(q_\phi(z | x^{(i)}) || p_\theta(z)) \\ \mathcal{L}(x^{(i)}, \theta, \phi)$$

For every minibatch of input data: compute this forward pass, and then backprop!



Variational Autoencoders: Generating Data!

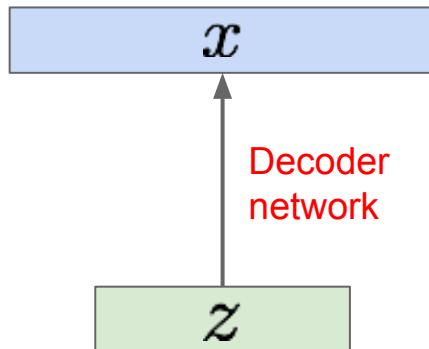
Our assumption about data generation process

Sample from true conditional

$$p_{\theta^*}(x \mid z^{(i)})$$

Sample from true prior

$$z^{(i)} \sim p_{\theta^*}(z)$$



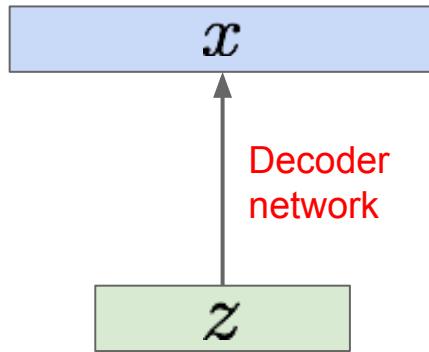
Kingma and Welling, “Auto-Encoding Variational Bayes”, ICLR 2014

Variational Autoencoders: Generating Data!

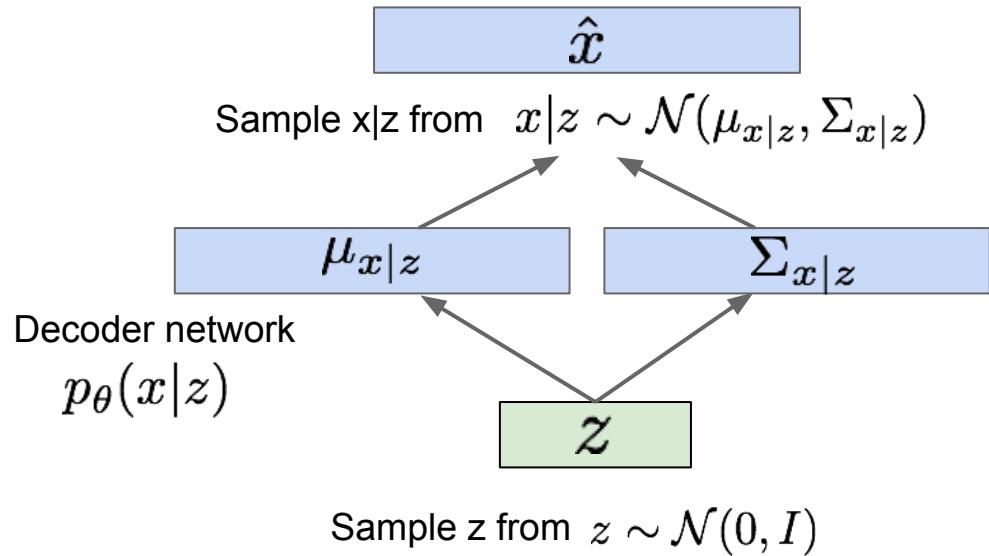
Our assumption about data generation process

Sample from true conditional
 $p_{\theta^*}(x | z^{(i)})$

Sample from true prior
 $z^{(i)} \sim p_{\theta^*}(z)$



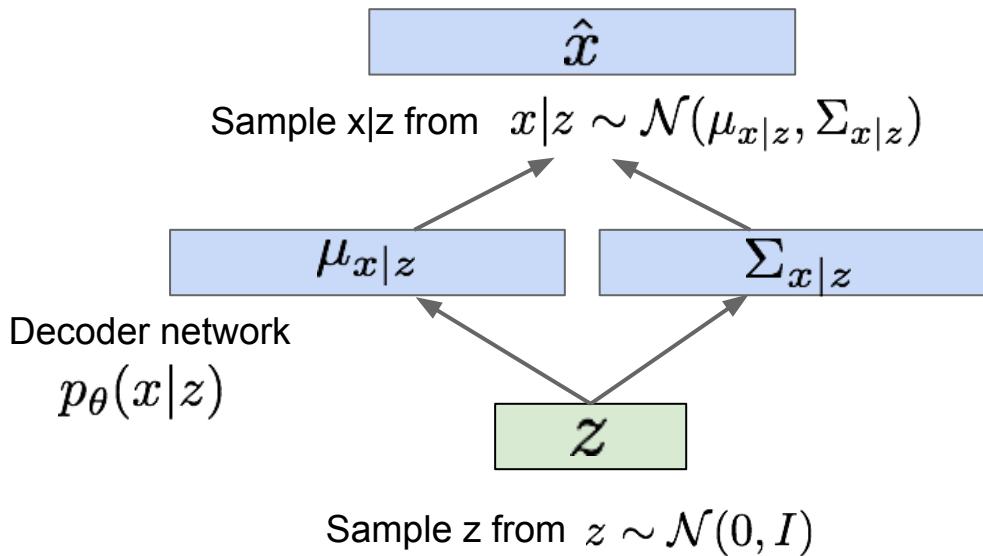
Now given a trained VAE:
use decoder network & sample z from prior!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Generating Data!

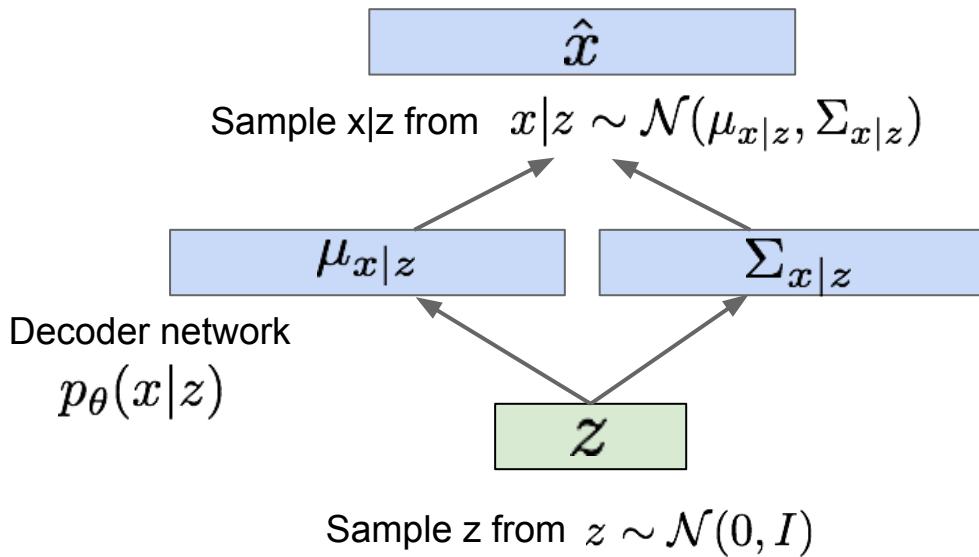
Use decoder network. Now sample z from prior!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

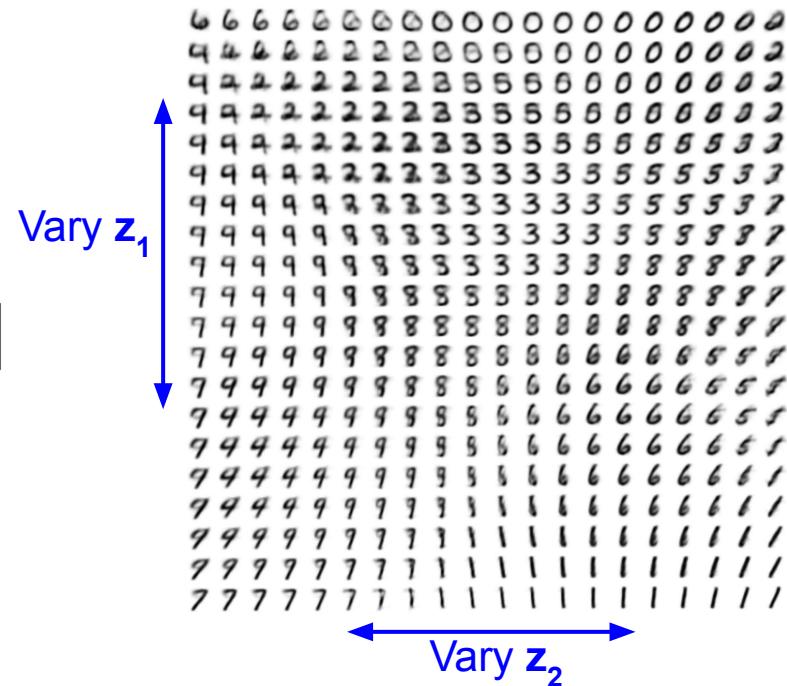
Variational Autoencoders: Generating Data!

Use decoder network. Now sample z from prior!



Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Data manifold for 2-d z



Variational Autoencoders: Generating Data!

Diagonal prior on z
=> independent
latent variables

Different
dimensions of z
encode
interpretable factors
of variation

Degree of smile
Vary z_1



Vary z_2 Head pose

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Generating Data!

Diagonal prior on z
=> independent
latent variables

Different
dimensions of z
encode
interpretable factors
of variation

Also good feature representation that
can be computed using $q_\phi(z|x)$!

Degree of smile
Vary z_1



Vary z_2 Head pose

Kingma and Welling, "Auto-Encoding Variational Bayes", ICLR 2014

Variational Autoencoders: Generating Data!



32x32 CIFAR-10



Labeled Faces in the Wild

Figures copyright (L) Dirk Kingma et al. 2016; (R) Anders Larsen et al. 2017. Reproduced with permission.

Variational Autoencoders

Probabilistic spin to traditional autoencoders => allows generating data

Defines an intractable density => derive and optimize a (variational) lower bound

Pros:

- Principled approach to generative models
- Interpretable latent space.
- Allows inference of $q(z|x)$, can be useful feature representation for other tasks

Cons:

- Maximizes lower bound of likelihood: okay, but not as good evaluation as PixelRNN/PixelCNN
- Samples blurrier and lower quality compared to state-of-the-art (GANs)

Active areas of research:

- More flexible approximations, e.g. richer approximate posterior instead of diagonal Gaussian, e.g., Gaussian Mixture Models (GMMs), Categorical Distributions.
- Learning disentangled representations.

Taxonomy of Generative Models

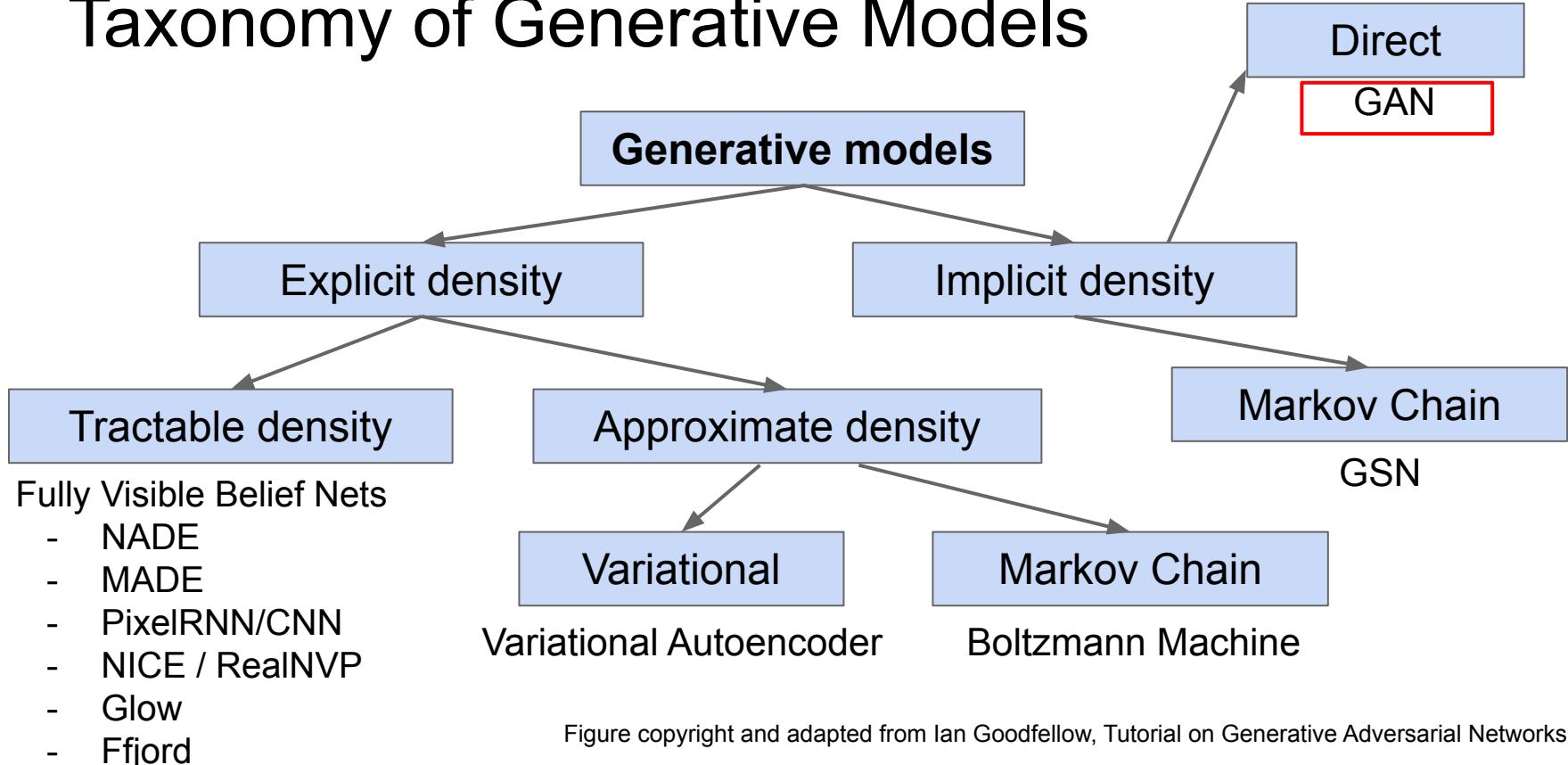


Figure copyright and adapted from Ian Goodfellow, Tutorial on Generative Adversarial Networks, 2017.

Generative Adversarial Networks (GANs)

So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i|x_1, \dots, x_{i-1})$$

VAEs define intractable density function with latent \mathbf{z} :

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i|x_1, \dots, x_{i-1})$$

VAEs define intractable density function with latent \mathbf{z} :

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?

So far...

PixelRNN/CNNs define tractable density function, optimize likelihood of training data:

$$p_{\theta}(x) = \prod_{i=1}^n p_{\theta}(x_i|x_1, \dots, x_{i-1})$$

VAEs define intractable density function with latent \mathbf{z} :

$$p_{\theta}(x) = \int p_{\theta}(z)p_{\theta}(x|z)dz$$

Cannot optimize directly, derive and optimize lower bound on likelihood instead

What if we give up on explicitly modeling density, and just want ability to sample?

GANs: not modeling any explicit density function!

Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Generative Adversarial Networks

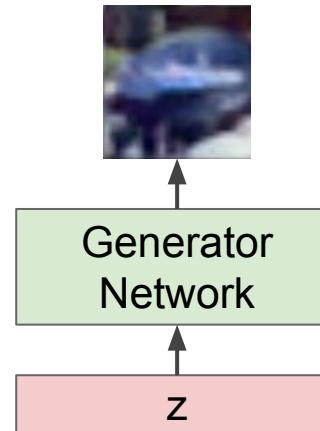
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

Output: Sample from
training distribution

Input: Random noise



Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

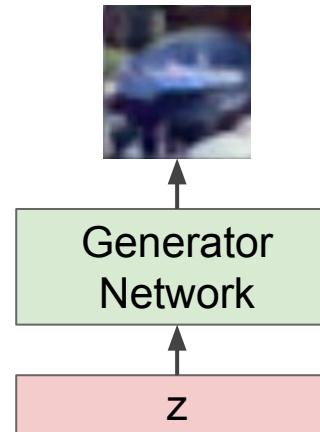
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution

Input: Random noise



Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

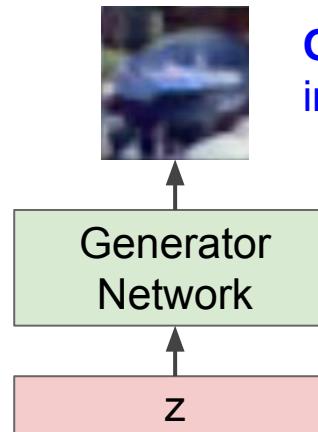
Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Output: Sample from training distribution

Input: Random noise



Objective: generated images should look “real”

Generative Adversarial Networks

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Problem: Want to sample from complex, high-dimensional training distribution. No direct way to do this!

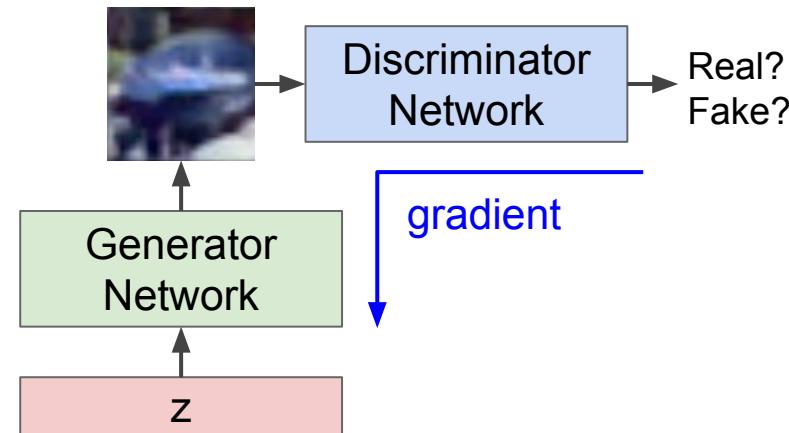
Solution: Sample from a simple distribution we can easily sample from, e.g. random noise. Learn transformation to training distribution.

But we don't know which sample z maps to which training image -> can't learn by reconstructing training images

Solution: Use a discriminator network to tell whether the generated image is within data distribution (“real”) or not

Output: Sample from training distribution

Input: Random noise



Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Discriminator network: try to distinguish between real and fake images

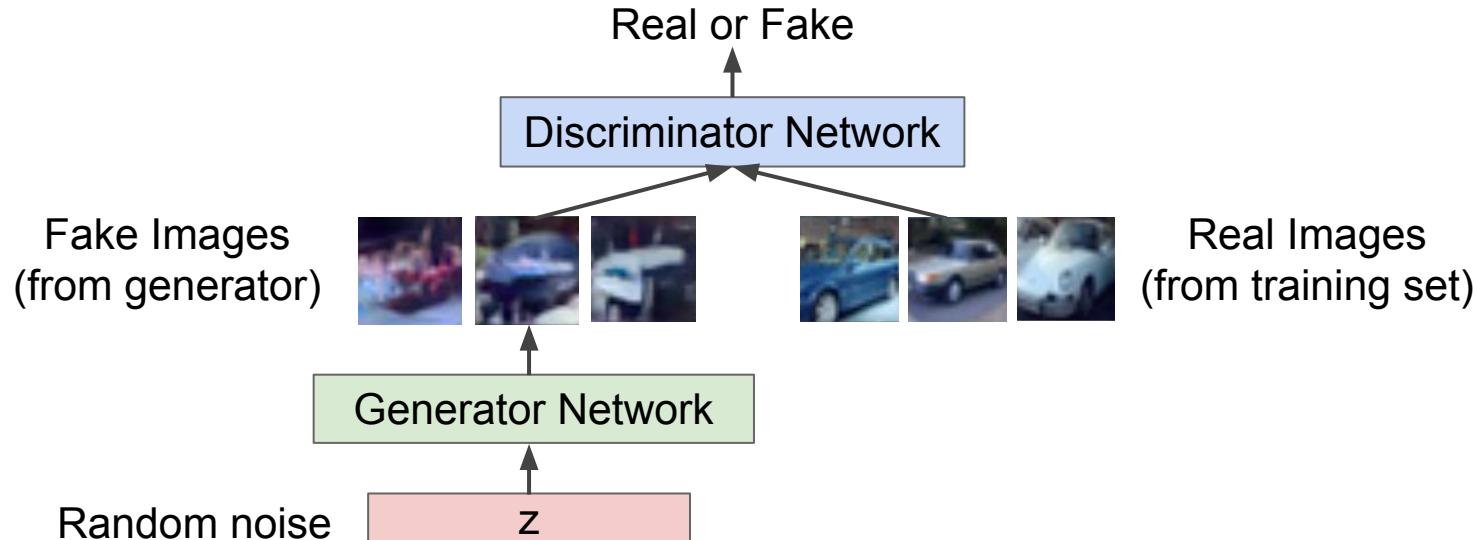
Generator network: try to fool the discriminator by generating real-looking images

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Discriminator network: try to distinguish between real and fake images

Generator network: try to fool the discriminator by generating real-looking images



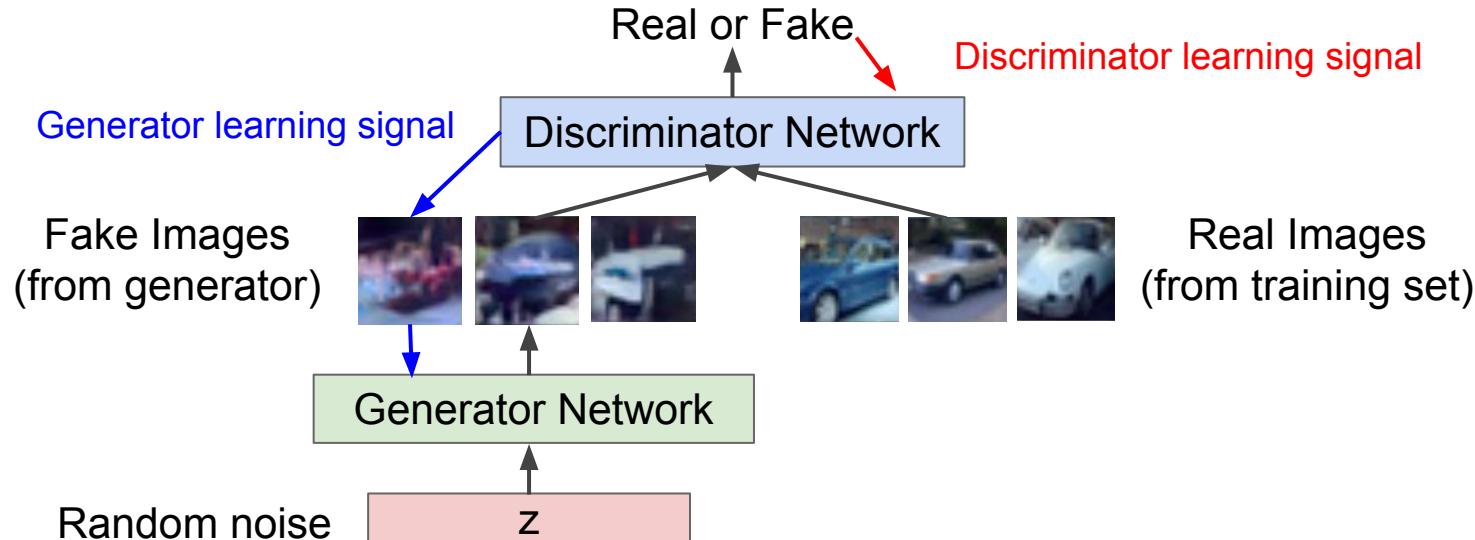
Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Discriminator network: try to distinguish between real and fake images

Generator network: try to fool the discriminator by generating real-looking images



Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Discriminator network: try to distinguish between real and fake images

Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Generator objective
Discriminator objective

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Discriminator network: try to distinguish between real and fake images

Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)}) \right]$$

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Discriminator network: try to distinguish between real and fake images

Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + \mathbb{E}_{z \sim p(z)} \log (1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)}) \right]$$


Training GANs: Two-player game

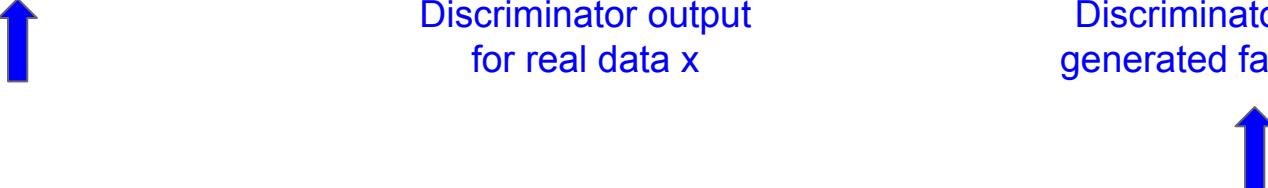
Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Discriminator network: try to distinguish between real and fake images

Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)}) \right]$$


Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Discriminator network: try to distinguish between real and fake images

Generator network: try to fool the discriminator by generating real-looking images

Train jointly in **minimax game**

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log \underbrace{D_{\theta_d}(x)}_{\text{Discriminator output for real data } x} + \mathbb{E}_{z \sim p(z)} \log(1 - \underbrace{D_{\theta_d}(G_{\theta_g}(z))}_{\text{Discriminator output for generated fake data } G(z)}) \right]$$

Discriminator outputs likelihood in (0,1) of real image

- Discriminator (θ_d) wants to **maximize objective** such that $D(x)$ is close to 1 (real) and $D(G(z))$ is close to 0 (fake)
- Generator (θ_g) wants to **minimize objective** such that $D(G(z))$ is close to 1 (discriminator is fooled into thinking generated $G(z)$ is real)

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

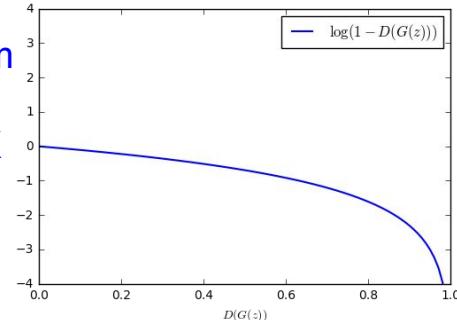
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).



Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. **Gradient ascent** on discriminator

$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. **Gradient descent** on generator

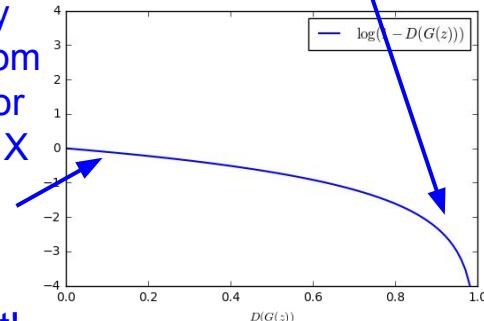
$$\min_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z)))$$

In practice, optimizing this generator objective does not work well!

When sample is likely fake, want to learn from it to improve generator (move to the right on X axis).

But gradient in this region is relatively flat!

Gradient signal dominated by region where sample is already good



Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Minimax objective function:

$$\min_{\theta_g} \max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

Alternate between:

1. Gradient ascent on discriminator

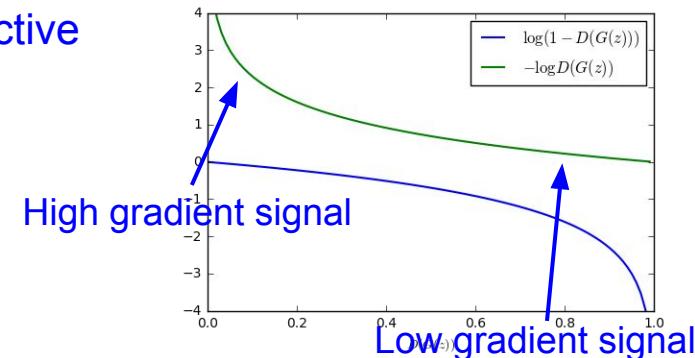
$$\max_{\theta_d} \left[\mathbb{E}_{x \sim p_{data}} \log D_{\theta_d}(x) + \mathbb{E}_{z \sim p(z)} \log(1 - D_{\theta_d}(G_{\theta_g}(z))) \right]$$

2. Instead: Gradient ascent on generator, different objective

$$\max_{\theta_g} \mathbb{E}_{z \sim p(z)} \log(D_{\theta_d}(G_{\theta_g}(z)))$$

Instead of minimizing likelihood of discriminator being correct, now maximize likelihood of discriminator being wrong.

Same objective of fooling discriminator, but now higher gradient signal for bad samples => works much better! Standard in practice.



Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

Putting it together: GAN training algorithm

for number of training iterations **do**

for k steps **do**

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Sample minibatch of m examples $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\mathbf{x})$.
- Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D_{\theta_d}(\mathbf{x}^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}^{(i)}))) \right]$$

end for

- Sample minibatch of m noise samples $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$ from noise prior $p_g(\mathbf{z})$.
- Update the generator by ascending its stochastic gradient (improved objective):

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(\mathbf{z}^{(i)})))$$

end for

Training GANs: Two-player game

Ian Goodfellow et al., "Generative Adversarial Nets", NIPS 2014

Putting it together: GAN training algorithm

Some find $k=1$ more stable, others use $k > 1$, no best rule.

```
for number of training iterations do
    for k steps do
        • Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
        • Sample minibatch of  $m$  examples  $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$  from data generating distribution  $p_{\text{data}}(\mathbf{x})$ .
        • Update the discriminator by ascending its stochastic gradient:
```

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^m \left[\log D_{\theta_d}(\mathbf{x}^{(i)}) + \log(1 - D_{\theta_d}(G_{\theta_g}(\mathbf{z}^{(i)}))) \right]$$

```
end for
• Sample minibatch of  $m$  noise samples  $\{\mathbf{z}^{(1)}, \dots, \mathbf{z}^{(m)}\}$  from noise prior  $p_g(\mathbf{z})$ .
• Update the generator by ascending its stochastic gradient (improved objective):
```

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^m \log(D_{\theta_d}(G_{\theta_g}(\mathbf{z}^{(i)})))$$

```
end for
```

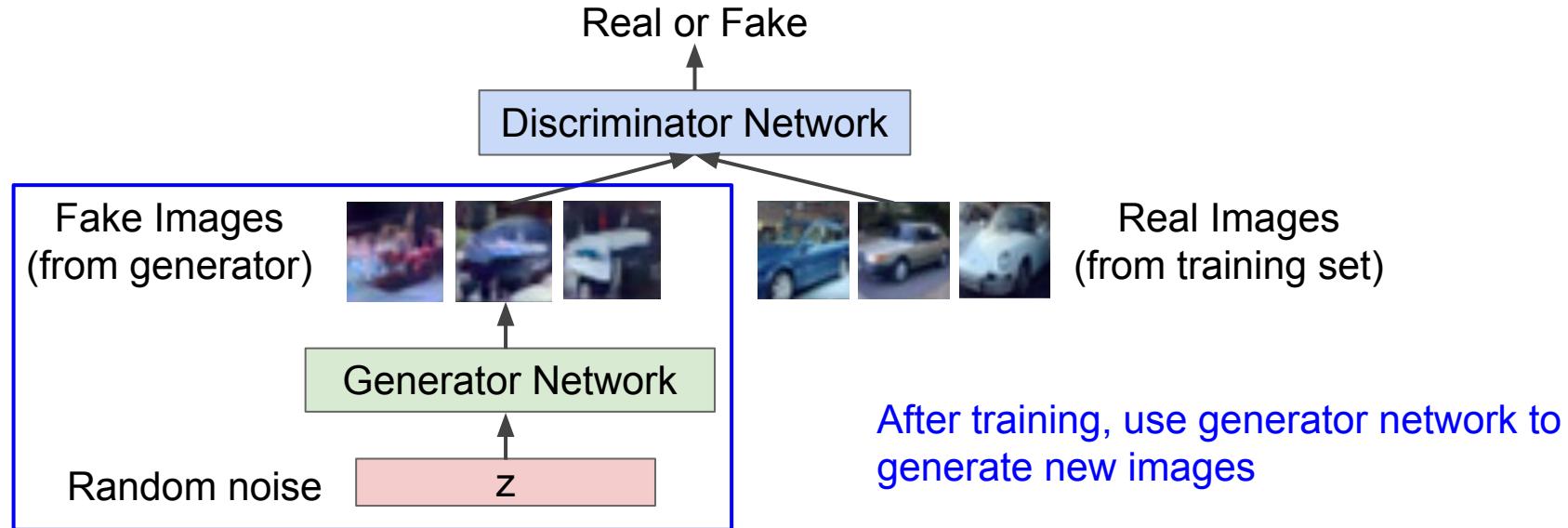
Arjovsky et al. "Wasserstein gan." arXiv preprint arXiv:1701.07875 (2017)

Berthelot, et al. "Began: Boundary equilibrium generative adversarial networks." arXiv preprint arXiv:1703.10717 (2017)

Training GANs: Two-player game

Ian Goodfellow et al., “Generative Adversarial Nets”, NIPS 2014

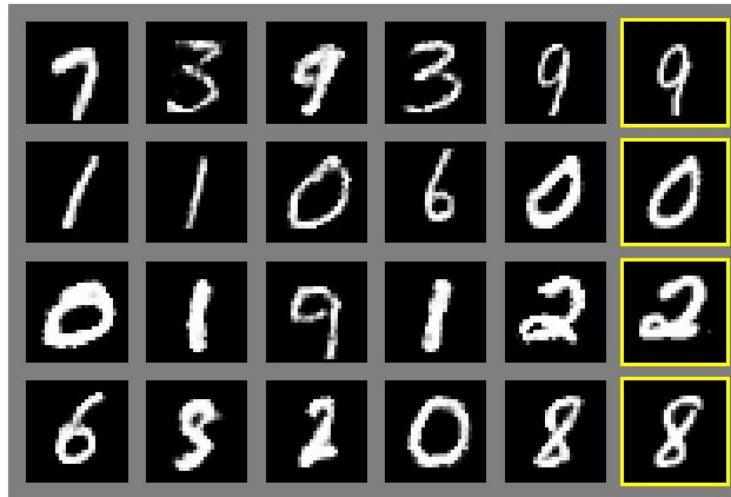
Generator network: try to fool the discriminator by generating real-looking images
Discriminator network: try to distinguish between real and fake images



Fake and real images copyright Emily Denton et al. 2015. Reproduced with permission.

Generative Adversarial Nets

Generated samples



Nearest neighbor from training set

Figures copyright Ian Goodfellow et al., 2014. Reproduced with permission.

Generative Adversarial Nets

Generated samples (CIFAR-10)



Nearest neighbor from training set

Figures copyright Ian Goodfellow et al., 2014. Reproduced with permission.

Generative Adversarial Nets: Convolutional Architectures

Generator is an upsampling network with fractionally-strided convolutions
Discriminator is a convolutional network

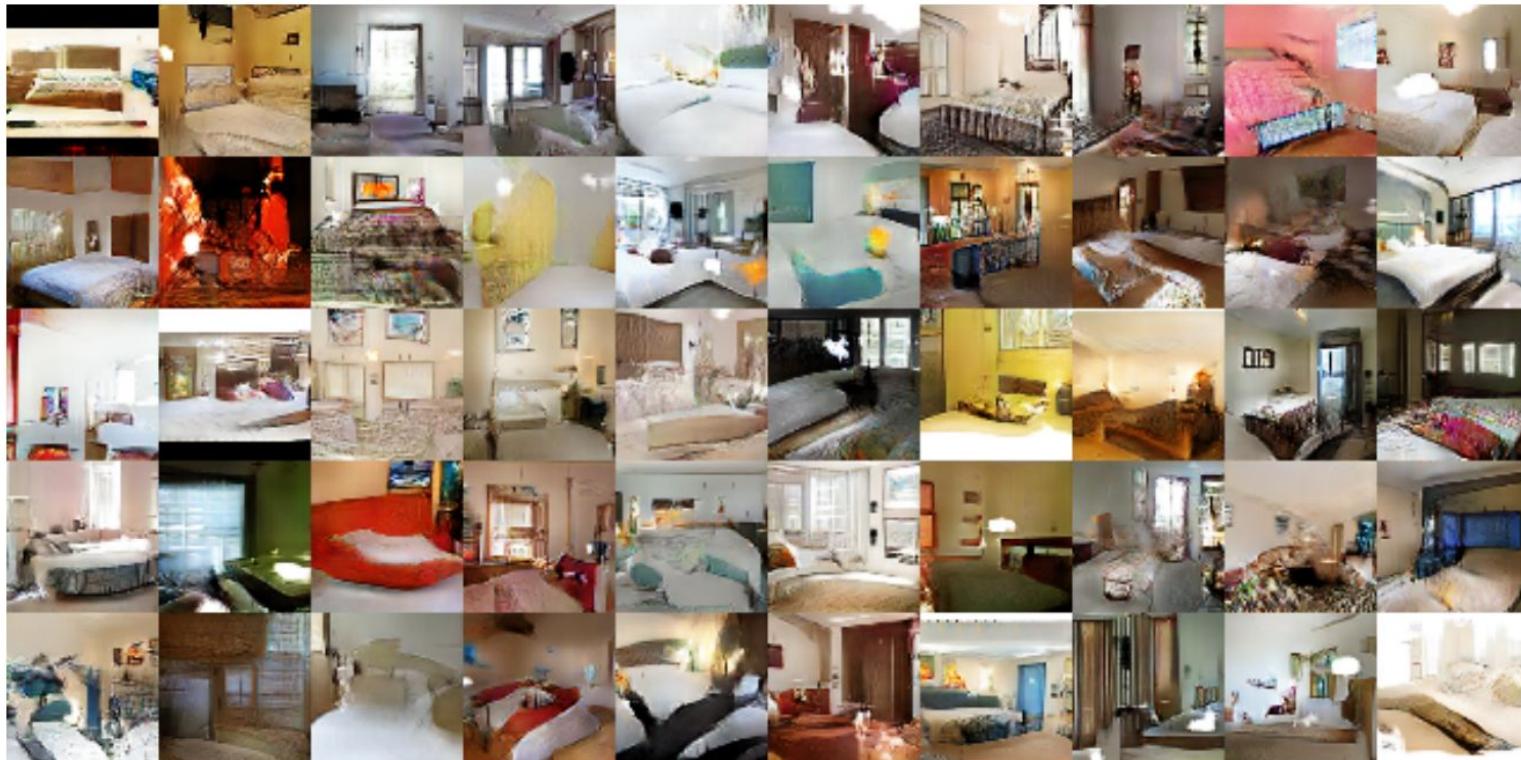
Architecture guidelines for stable Deep Convolutional GANs

- Replace any pooling layers with strided convolutions (discriminator) and fractional-strided convolutions (generator).
- Use batchnorm in both the generator and the discriminator.
- Remove fully connected hidden layers for deeper architectures.
- Use ReLU activation in generator for all layers except for the output, which uses Tanh.
- Use LeakyReLU activation in the discriminator for all layers.

Radford et al, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks”, ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Samples
from the
model look
much
better!



Radford et al,
ICLR 2016

Generative Adversarial Nets: Convolutional Architectures

Interpolating
between
random
points in latent
space



Radford et al,
ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math

Smiling woman



Neutral woman



Neutral man



Samples
from the
model

Radford et al, ICLR 2016

Generative Adversarial Nets: Interpretable Vector Math

Radford et al, ICLR 2016

Smiling woman Neutral woman Neutral man

Samples
from the
model



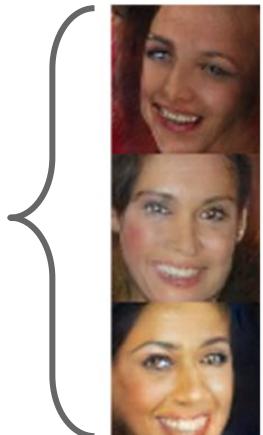
Average Z
vectors, do
arithmetic



Generative Adversarial Nets: Interpretable Vector Math

Smiling woman Neutral woman Neutral man

Samples
from the
model



Average Z
vectors, do
arithmetic



Radford et al, ICLR 2016

Smiling Man

Generative Adversarial Nets: Interpretable Vector Math

Glasses man



No glasses man

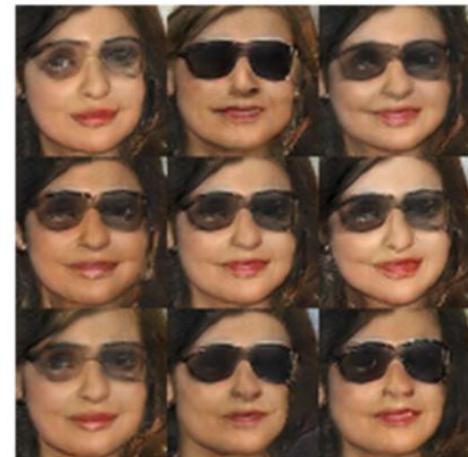


No glasses woman



Radford et al,
ICLR 2016

Woman with glasses



2017: Explosion of GANs

“The GAN Zoo”

- GAN - Generative Adversarial Networks
- 3D-GAN - Learning a Probabilistic Latent Space of Object Shapes via 3D Generative-Adversarial Modeling
- acGAN - Face Aging With Conditional Generative Adversarial Networks
- AC-GAN - Conditional Image Synthesis With Auxiliary Classifier GANs
- AdAGAN - AdaGAN: Boosting Generative Models
- AEGAN - Learning Inverse Mapping by Autoencoder based Generative Adversarial Nets
- AffGAN - Amortised MAP Inference for Image Super-resolution
- AL-CGAN - Learning to Generate Images of Outdoor Scenes from Attributes and Semantic Layouts
- ALI - Adversarially Learned Inference
- AM-GAN - Generative Adversarial Nets with Labeled Data by Activation Maximization
- AnoGAN - Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery
- ArtGAN - ArtGAN: Artwork Synthesis with Conditional Categorical GANs
- b-GAN - b-GAN: Unified Framework of Generative Adversarial Networks
- Bayesian GAN - Deep and Hierarchical Implicit Models
- BEGAN - BEGAN: Boundary Equilibrium Generative Adversarial Networks
- BiGAN - Adversarial Feature Learning
- BS-GAN - Boundary-Seeking Generative Adversarial Networks
- CGAN - Conditional Generative Adversarial Nets
- CaloGAN - CaloGAN: Simulating 3D High Energy Particle Showers in Multi-Layer Electromagnetic Calorimeters with Generative Adversarial Networks
- CCGAN - Semi-Supervised Learning with Context-Conditional Generative Adversarial Networks
- CatGAN - Unsupervised and Semi-supervised Learning with Categorical Generative Adversarial Networks
- CoGAN - Coupled Generative Adversarial Networks

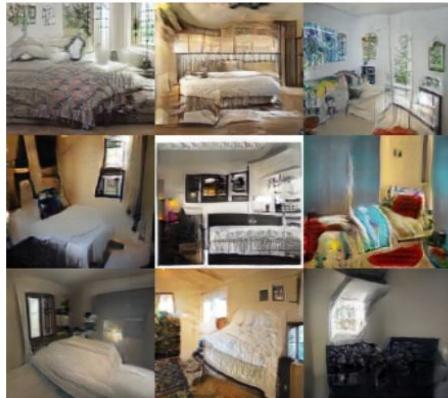
See also: <https://github.com/soumith/ganhacks> for tips and tricks for trainings GANs

- Context-RNN-GAN - Contextual RNN-GANs for Abstract Reasoning Diagram Generation
- C-RNN-GAN - C-RNN-GAN: Continuous recurrent neural networks with adversarial training
- CS-GAN - Improving Neural Machine Translation with Conditional Sequence Generative Adversarial Nets
- CVAE-GAN - CVAE-GAN: Fine-Grained Image Generation through Asymmetric Training
- CycleGAN - Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks
- DTN - Unsupervised Cross-Domain Image Generation
- DCGAN - Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks
- DiscoGAN - Learning to Discover Cross-Domain Relations with Generative Adversarial Networks
- DR-GAN - Disentangled Representation Learning GAN for Pose-Invariant Face Recognition
- DualGAN - DualGAN: Unsupervised Dual Learning for Image-to-Image Translation
- EBGAN - Energy-based Generative Adversarial Network
- f-GAN - f-GAN: Training Generative Neural Samplers using Variational Divergence Minimization
- FF-GAN - Towards Large-Pose Face Frontalization in the Wild
- GAWWN - Learning What and Where to Draw
- GeneGAN - GeneGAN: Learning Object Transfiguration and Attribute Subspace from Unpaired Data
- Geometric GAN - Geometric GAN
- GoGAN - Gang of GANs: Generative Adversarial Networks with Maximum Margin Ranking
- GP-GAN - GP-GAN: Towards Realistic High-Resolution Image Blending
- IAN - Neural Photo Editing with Introspective Adversarial Networks
- iGAN - Generative Visual Manipulation on the Natural Image Manifold
- IcGAN - Invertible Conditional GANs for image editing
- ID-CGAN - Image De-raining Using a Conditional Generative Adversarial Network
- Improved GAN - Improved Techniques for Training GANs
- InfoGAN - InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets
- LAGAN - Learning Particle Physics by Example: Location-Aware Generative Adversarial Networks for Physics Synthesis
- LAPGAN - Deep Generative Image Models using a Laplacian Pyramid of Adversarial Networks

<https://github.com/hindupuravinash/the-gan-zoo>

2017: Explosion of GANs

Better training and generation



LSGAN, Zhu 2017.



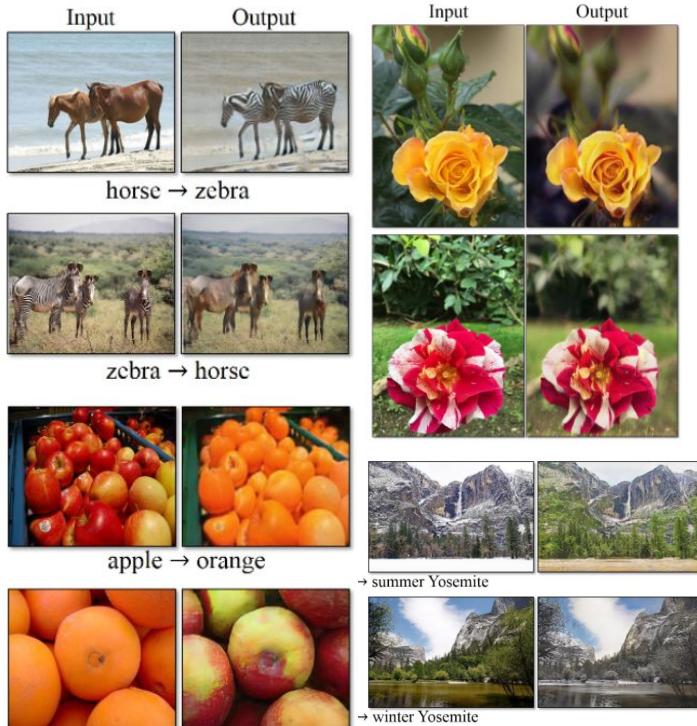
Wasserstein GAN,
Arjovsky 2017.
Improved Wasserstein
GAN, Gulrajani 2017.



Progressive GAN, Karras 2018.

2017: Explosion of GANs

Source->Target domain transfer



CycleGAN. Zhu et al. 2017.

Text -> Image Synthesis

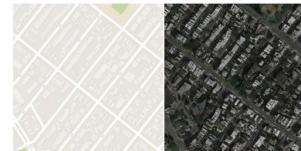
this small bird has a pink breast and crown, and black primaries and secondaries.

this magnificent fellow is almost all black with a red crest, and white cheek patch.



Reed et al. 2017.

Many GAN applications



Pix2pix. Isola 2017. Many examples at <https://phillipi.github.io/pix2pix/>

2019: BigGAN

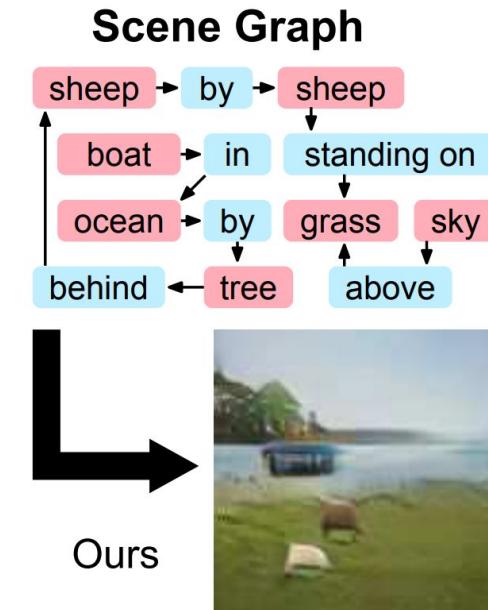


Brock et al., 2019

Scene graphs to GANs

Specifying exactly what kind of image you want to generate.

The explicit structure in scene graphs provides better image generation for complex scenes.

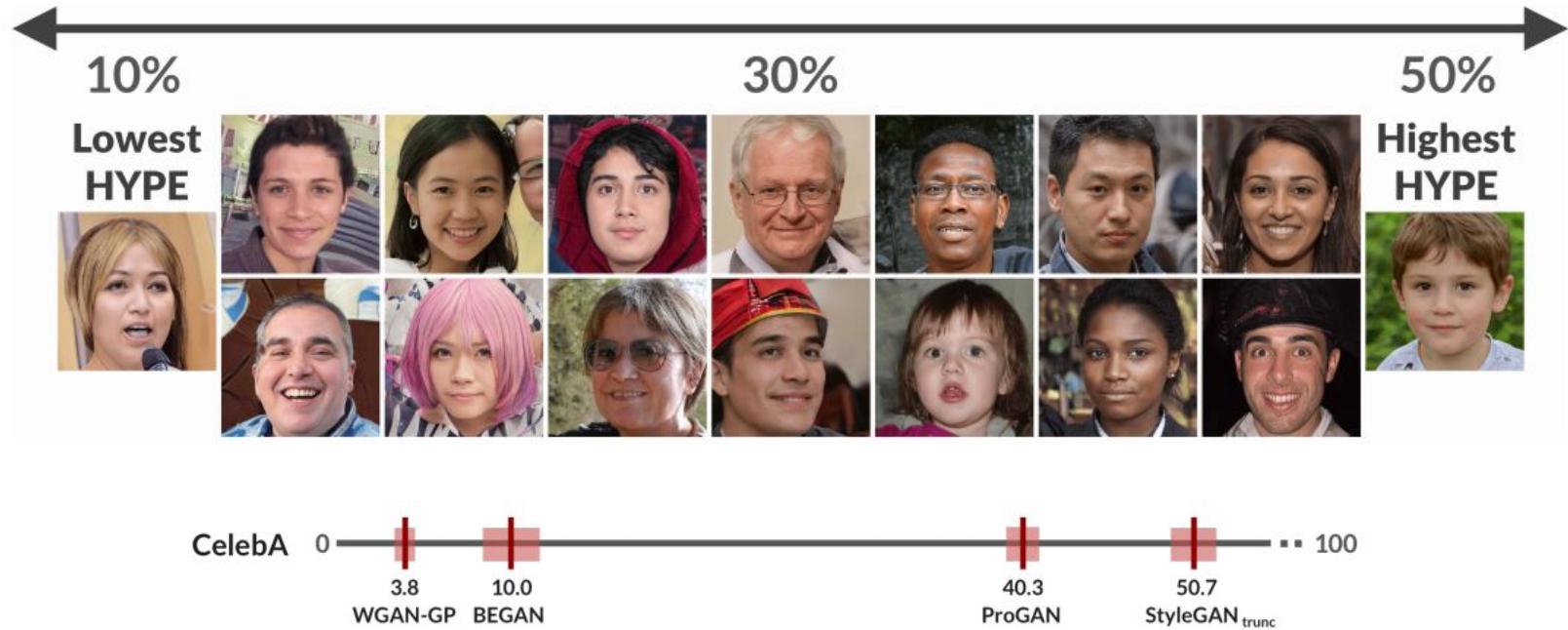


Figures copyright 2019. Reproduced with permission.

Johnson et al. Image Generation from Scene Graphs, CVPR 2019

HYPE: Human eYe Perceptual Evaluations

hype.stanford.edu



Zhou, Gordon, Krishna et al. HYPE: Human eYe Perceptual Evaluations, NeurIPS 2019

Figures copyright 2019. Reproduced with permission.

Summary: GANs

Don't work with an explicit density function

Take game-theoretic approach: learn to generate from training distribution through 2-player game

Pros:

- Beautiful, state-of-the-art samples!

Cons:

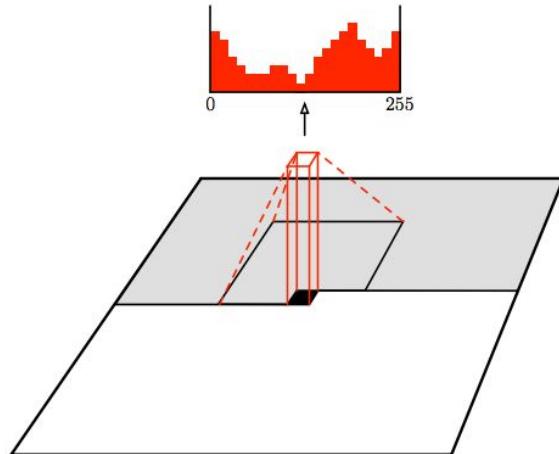
- Trickier / more unstable to train
- Can't solve inference queries such as $p(x)$, $p(z|x)$

Active areas of research:

- Better loss functions, more stable training (Wasserstein GAN, LSGAN, many others)
- Conditional GANs, GANs for all kinds of applications

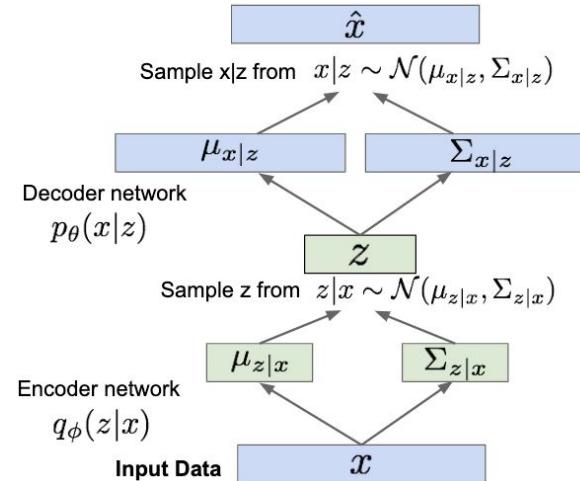
Summary

Autoregressive models: PixelRNN, PixelCNN



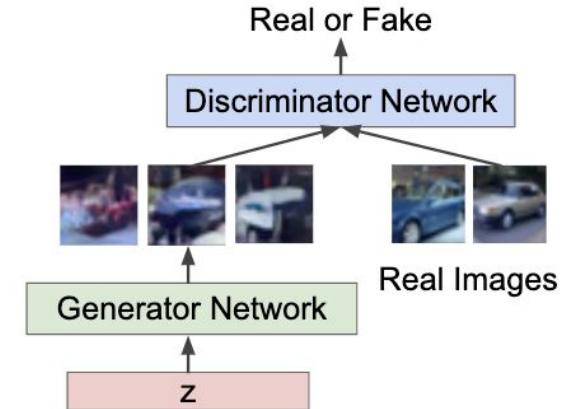
Van der Oord et al, "Conditional image generation with pixelCNN decoders", NIPS 2016

Variational Autoencoders



Kingma and Welling, "Auto-encoding variational bayes", ICLR 2013

Generative Adversarial Networks (GANs)



Goodfellow et al, "Generative Adversarial Nets", NIPS 2014

Useful Resources on Generative Models

CS 236: [Deep Generative Models](#) (Stanford)

CS 294-158 [Deep Unsupervised Learning](#) (Berkeley)

Next: Self-Supervised Learning