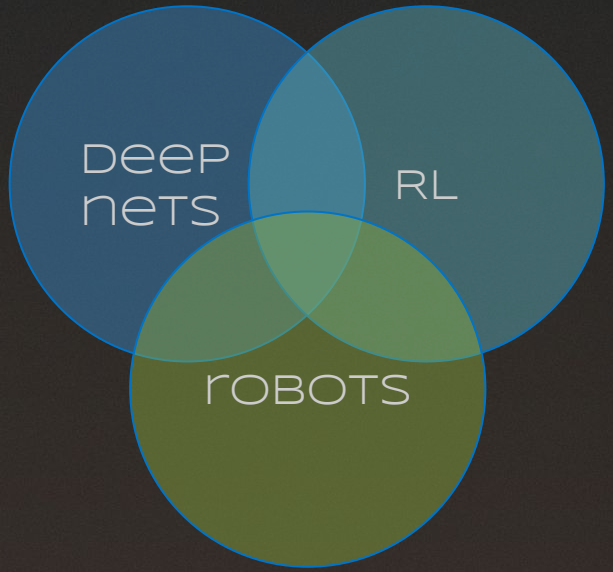
Beyond Image Recognition, End-to-End Learning, Embeddings

Raia Hadsell
Staff Research Scientist
DeepMind

DeepMind

Raia Hadsell

- philosophy
- circuit design
- robots with DL vision (phd, Yann LeCun)
- more robots



Now:

- deep learning
- reinforcement learning
- end-to-end robots

DeepMind

Overview

1. End-to-end learning with more complex models and tasks
    a. Beyond classification: detection, segmentation, videos
    b. End-to-end case study: spatial transformer networks
2. Learning without labels: embeddings and manifolds
3. Putting many things together - RL + sequence learning + auxiliary losses

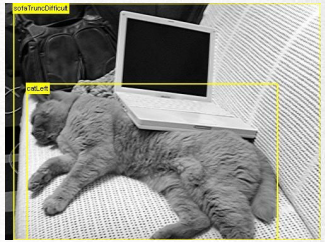
Beyond ImageNet Classification

detection, multi-label, segmentation, video, etc

Pretraining

- Training large models on large datasets takes a lot of time
  - several weeks on multiple GPUs for ImageNet ConvNets
- Network trained on a large diverse dataset (ImageNet) should be useful for other data
  - similar classes
  - similar domain

ImageNet cat



PASCAL VOC cat

- Make use of trained models :
  - Take a pretrained model
  - Plug into another network
  - Train remaining layers
  - Keep pre-trained weights fixed or slowly updated (fine-tuning)

Pretraining of Image Classification Nets

- Train ImageNet classification ConvNet
- Replace the last layer with a new one
  - #outputs = #classes in a new dataset
- Train the last layer

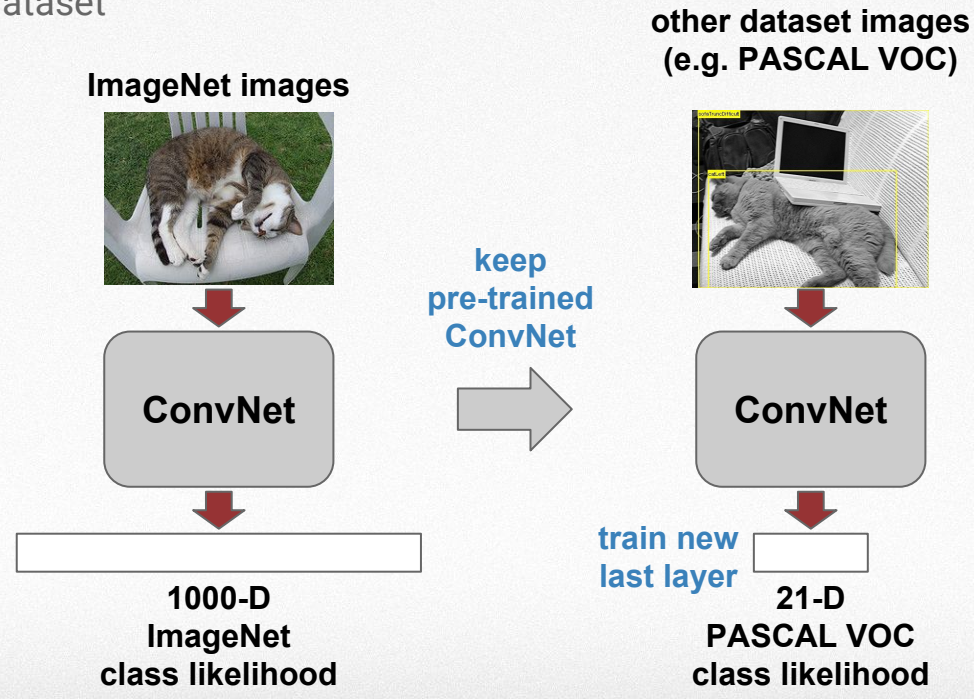
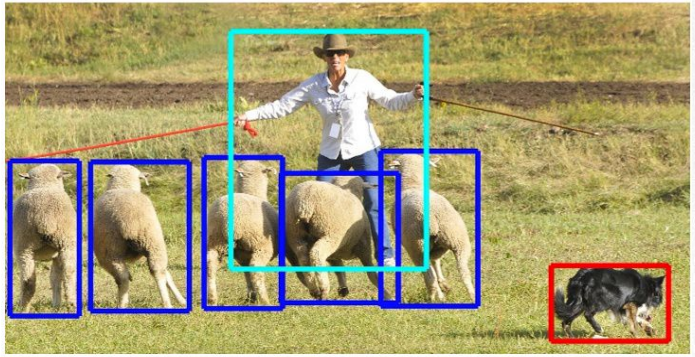


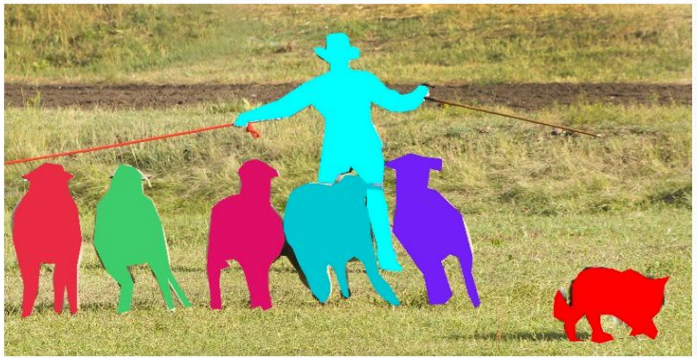
Image Recognition Tasks

(a) Image classification



(b) Object localisation/detection

(c) Semantic segmentation

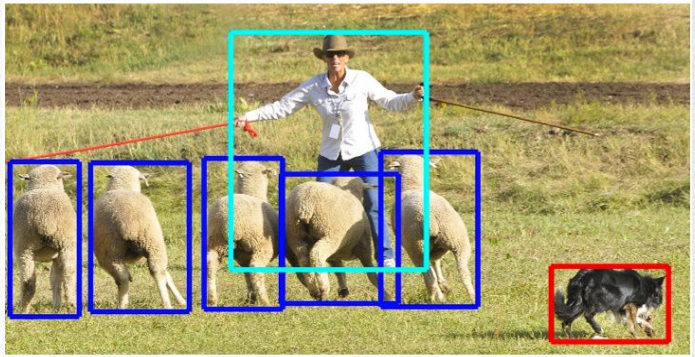


(d) Instance segmentation

Object Detection with ConvNets

Popular Approaches

- Detection as classification - classify all bounding boxes
  - too slow if done separately, but can be optimised
  - the same object might be detected multiple times



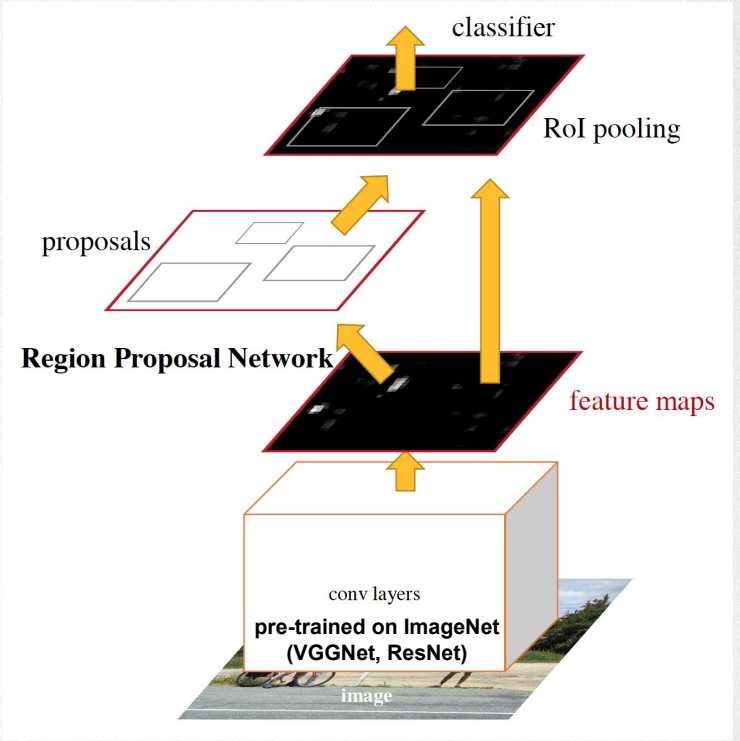
- Detection as bounding box prediction
  - regress 4 numbers - box coordinates (MSE loss)
  - the number of boxes is unknown

- Bounding box proposal
  - separate module (which can also be a ConvNet) predicts which boxes might contain objects
  - proposals are then passed through a classifier

Faster R-CNN (region-CNN): Overview

Ren et al., 2015

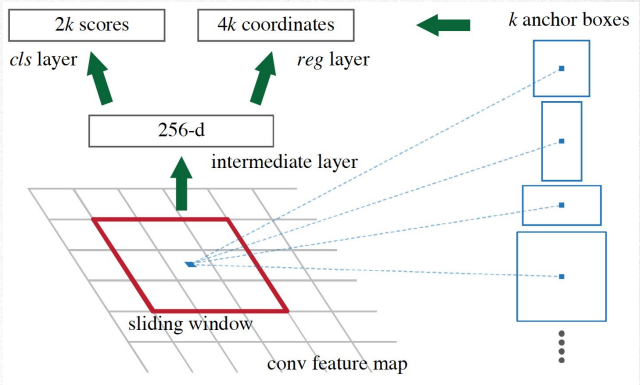
- State of the art proposal-based object detection
- Conv layers pre-trained on ImageNet
- Proposal stage
  - region proposal sub-network produces possible object locations
- Final stage
  - classifies proposals into classes
  - further refines their location



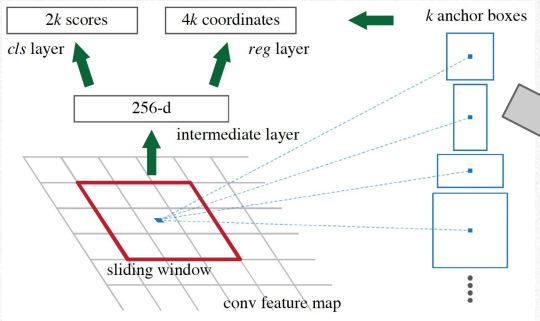
Faster R-CNN: Overview

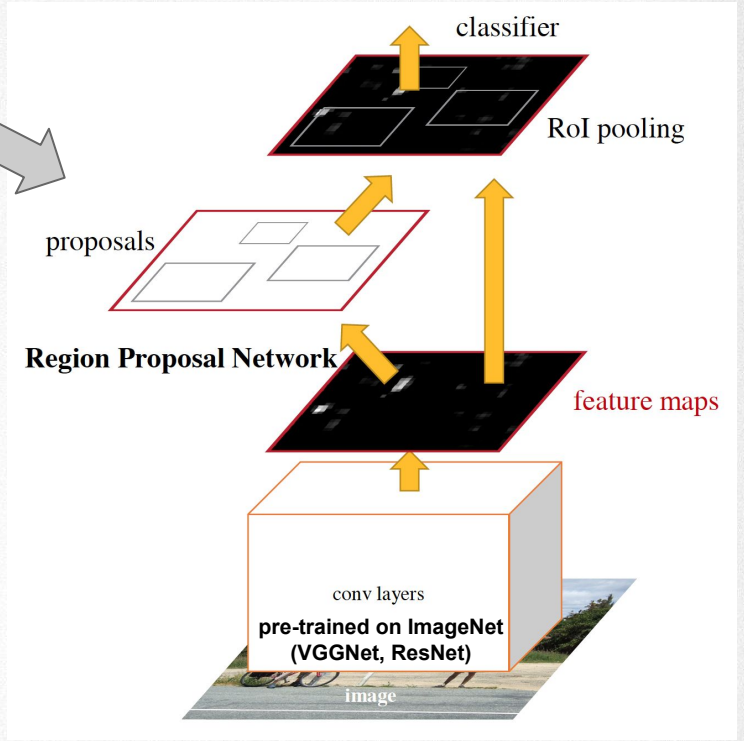
Region proposal network

- Objective
  - for each location, for each of the k=9 anchors
    - predict proposal box location wrt the anchor
    - predict if the proposal contains *some* object
- Configuration: two conv layers
  - first: 3×3 conv with 256 channels
  - second: 1×1 conv with 6k output channels
    - 6 numbers for each of the k proposals



Faster R-CNN: Overview





- Trained end-to-end, note differentiability req.
- Pre-trained conv layers are fine-tuned with a lower learning rate

Semantic Segmentation with ConvNets

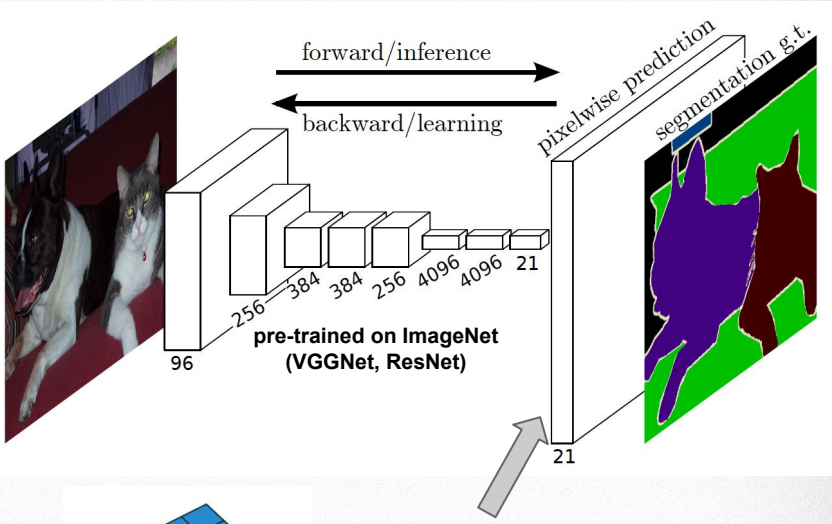
- Label each pixel into one of the object classes
- Popular approach: classify each pixel
- Network outputs a full resolution map S with #classes channels
  - $S_{i,j,c}$ - likelihood of pixel (i, j) having class c

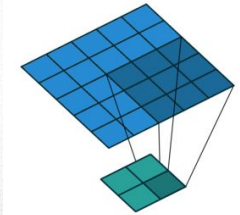
- Many architectural variations
- Popular resolution-preserving building blocks (cf. Karen Simoyan's talk)
  - pooling → transposed conv
  - conv/dilated conv, stride=1

Fully Convolutional Networks

Shelhamer et al., 2014

- ConvNet w/o linear layers (“fully convolutional”)
  - pooling layers are compensated by transposed conv in the end
  - pre-trained on ImageNet classification
- Penultimate conv layer has 21 channel
  - 20 classes & background



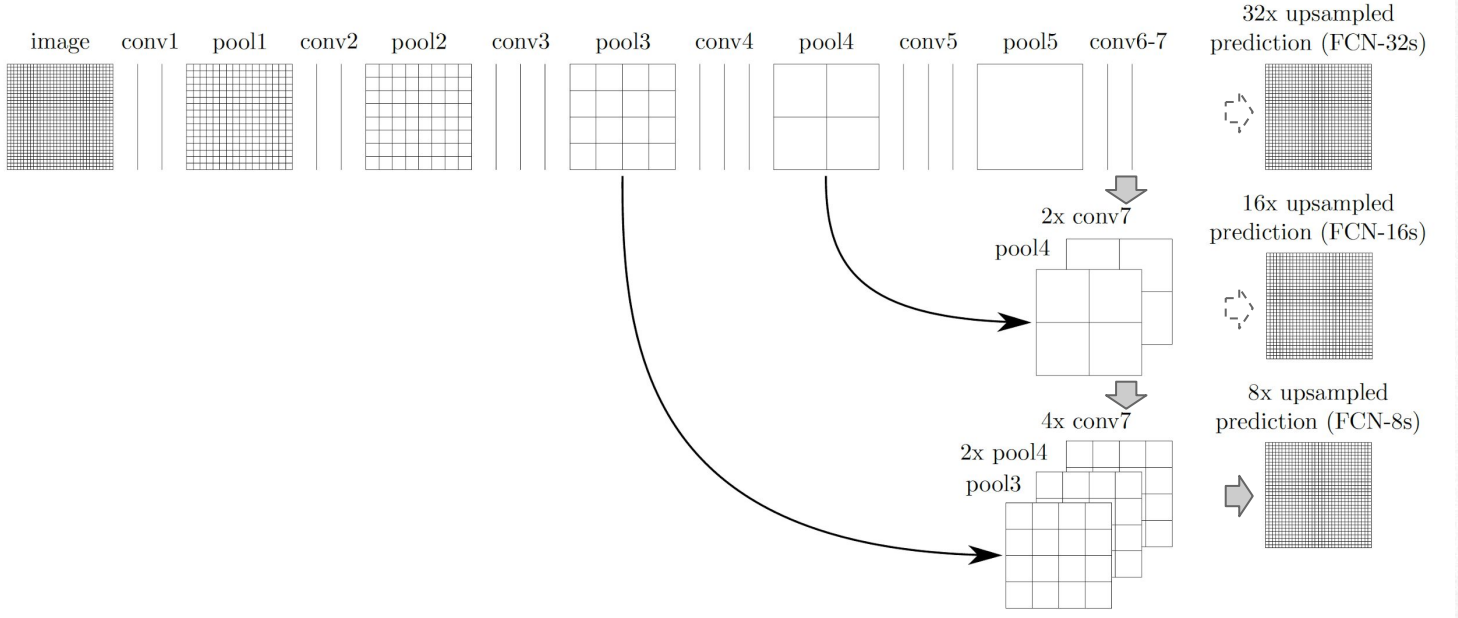


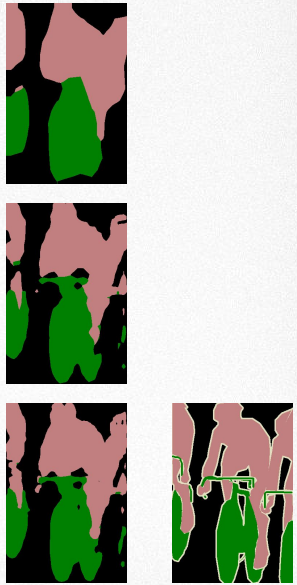
transposed conv increases spatial resolution

Fully Convolutional Networks

Combining high- and low-level cues

- skip connections from shallow (hi-res) layers to deep (low-res) layers
- fuse information from different layers



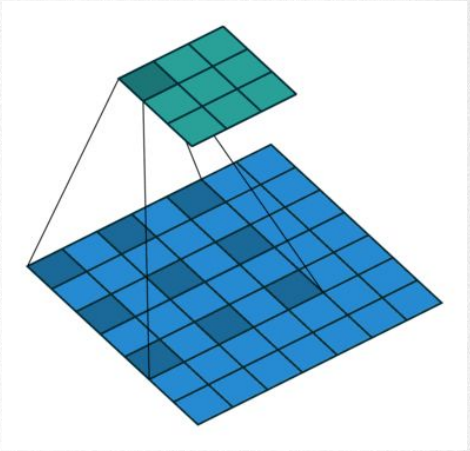


ground truth

Image Segmentation

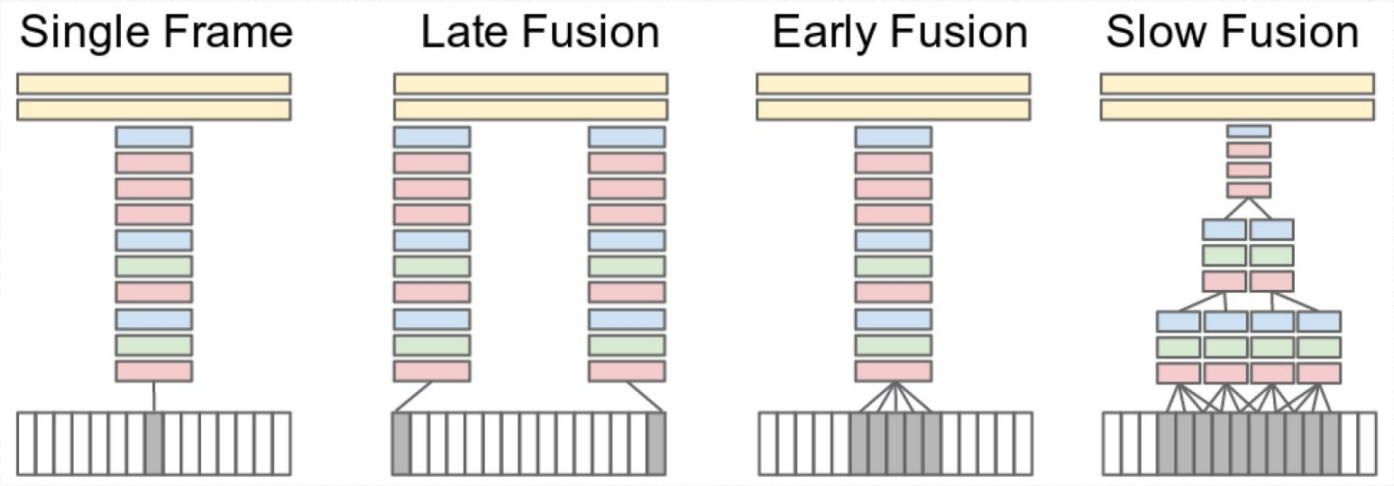
with dilated convolutions [Yu & Koltun, 2015]

- Recall: dilated convolution expands conv kernel
  - large receptive field
- ConvNet w/o linear layers
  - pre-trained on ImageNet classification (again!)
  - pooling removed
  - conv replaced with dilated conv
    - same receptive field
    - no resolution decrease
- Simpler and higher accuracy than the previous method



Video Classification with ConvNets

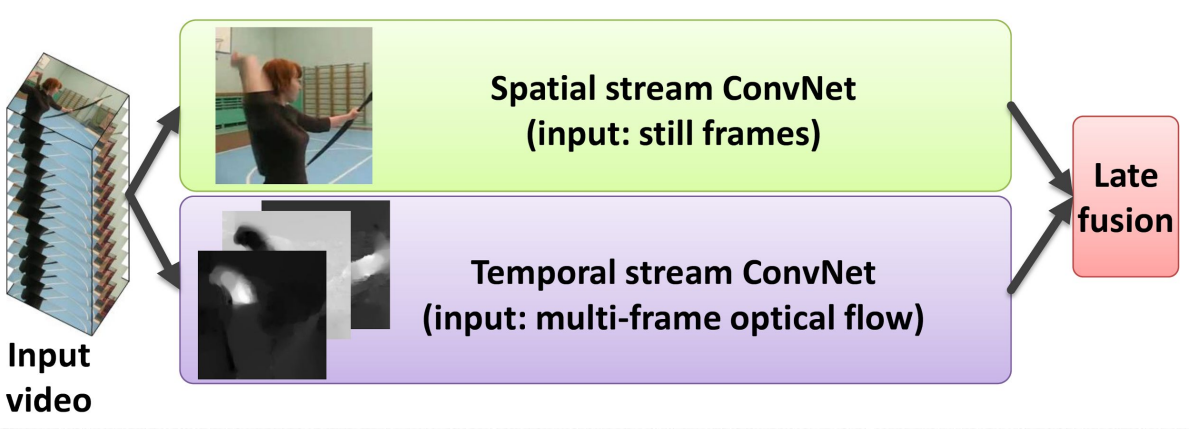
- ConvNets are applicable not only to images, but to other structured domains
- Multiple ways to incorporate temporal component
  - stack multiple frames into a volume [Karpathy et al., 2014]



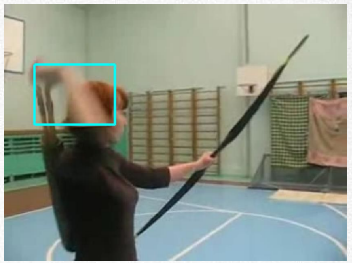
fusing information over multiple frames

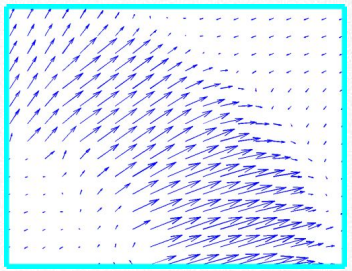
Two-Stream ConvNet for Video

Simonyan & Zisserman, 2014



- Appearance and motion are processed separately
- Spatial stream ConvNet
  - input: RGB frame
- Temporal stream ConvNet
  - input: motion vector field between several frames





optical flow

End-to-end learning deep dive: Spatial Transformers

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015

Convolutional nets have pooling layers to accommodate spatial translations

But this does not give true invariance, especially in intermediate feature layers, and they are hard-coded in the architecture.

For example - rotations must be learned.

A better idea - learn to transform the input so that it can be understood.

End-to-end learning: the Spatial Transformer Net

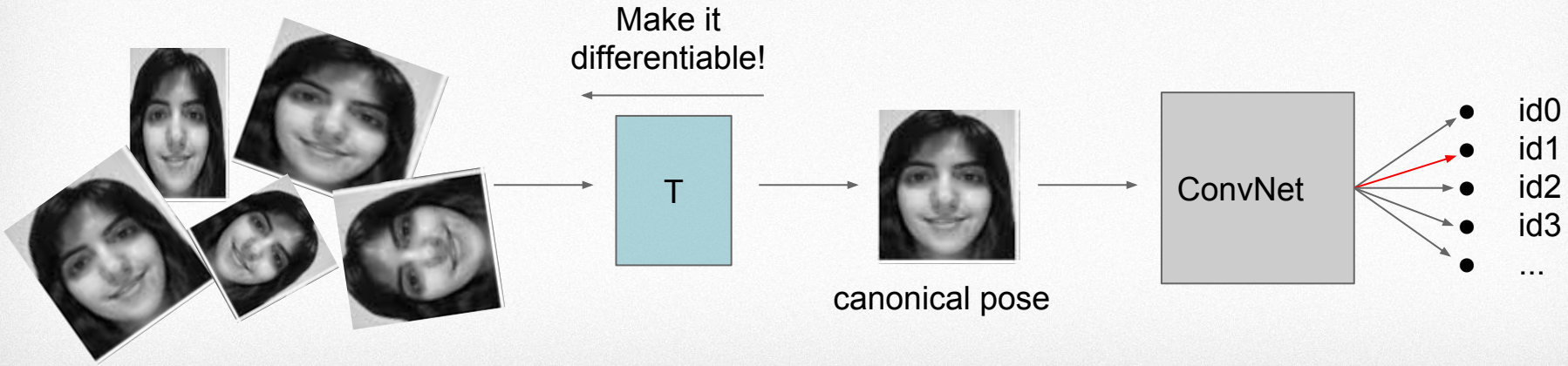
Jaderberg et al, 2015

Convolutional nets have pooling layers to accommodate spatial translations

But this does not give true invariance, especially in intermediate feature layers, and they are hard-coded in the architecture.

For example - rotations must be learned.

A better idea - learn to transform the input so that it can be understood.



End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015

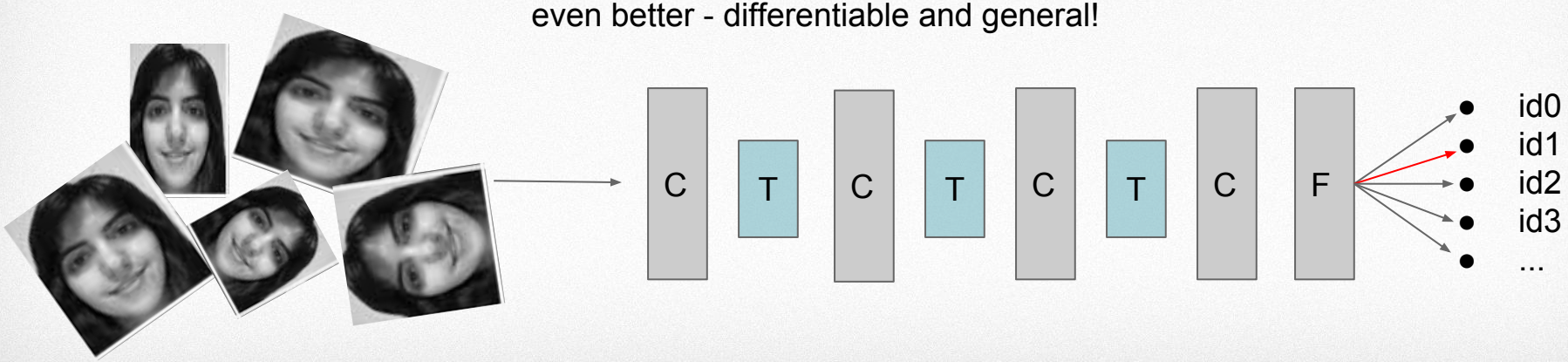
Convolutional nets have pooling layers to accommodate spatial translations

But this does not give true invariance, especially in intermediate feature layers, and they are hard-coded in the architecture.

For example - rotations must be learned.

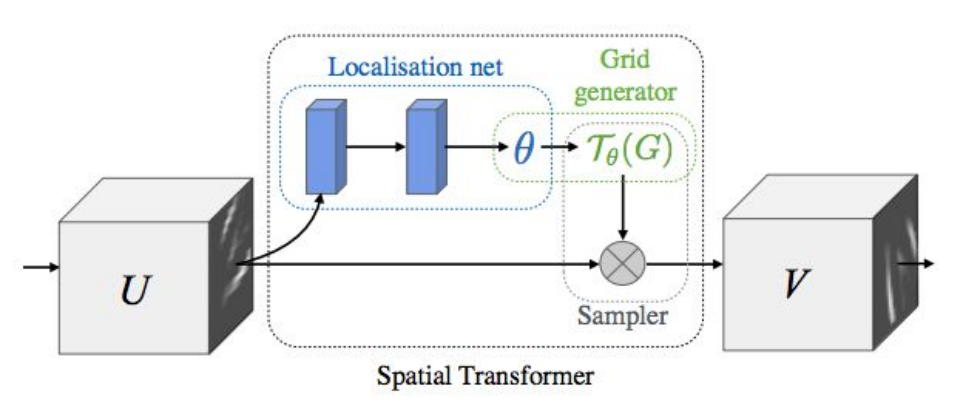
A better idea - learn to transform the input so that it can be understood.

even better - differentiable and general!



End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015

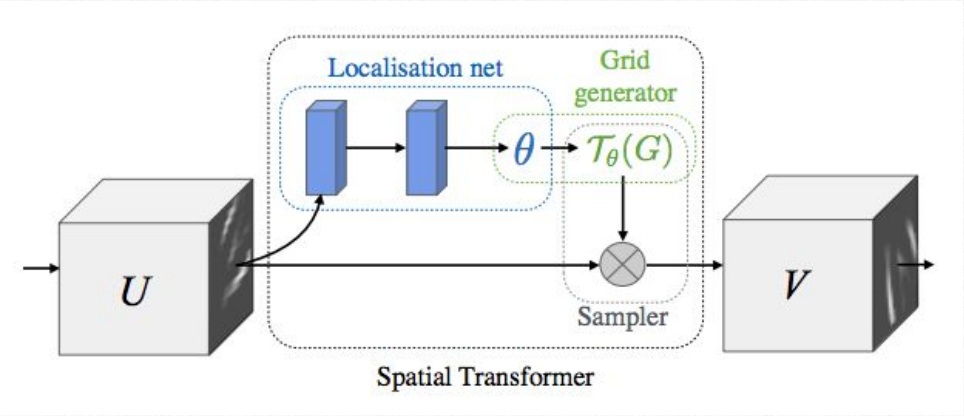


Differentiable components:

1. *Localisation net*: trainable function maps input features to transformation
2. *Grid generator*: creates a sampling grid
3. *Sampler*: produces the transformed feature map

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015



Differentiable components:

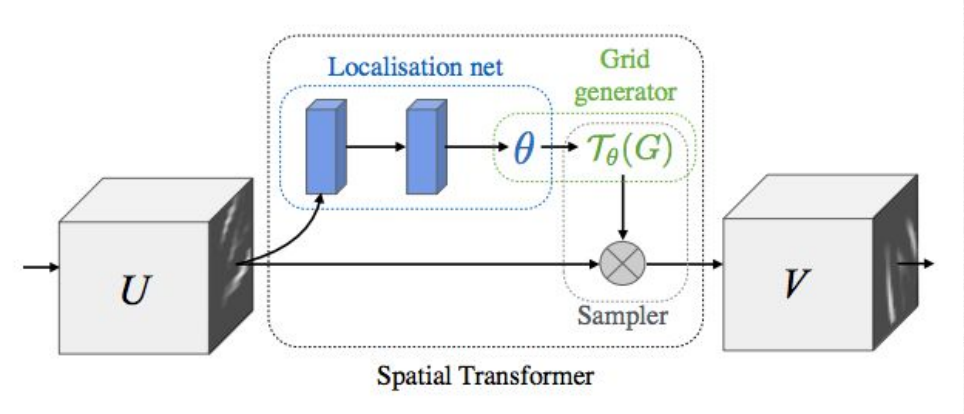
1. *Localisation net*

   trainable function maps input to transformation: $\theta = f_{loc}(U)$

   forward pass: normal inference $U \in \mathbb{R}^{H \times W \times C} \longrightarrow \theta \in \mathbb{R}^6$

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015



Differentiable components:

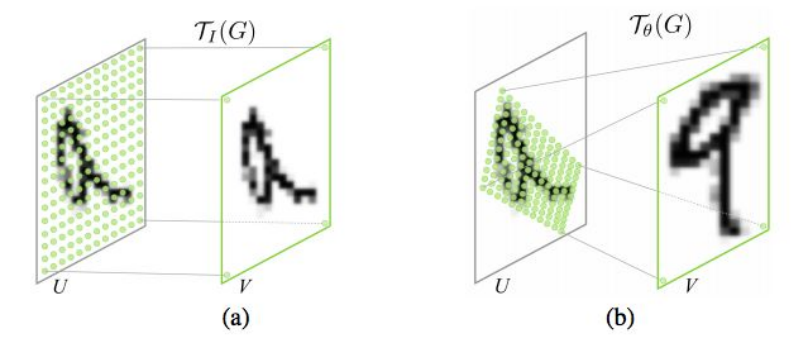
2. *Grid Generator* $\mathcal{T}_\theta(G)$

   forward pass: parameterised grid sampler generates an output map

$$\theta \in \mathbb{R}^6 \longrightarrow V \in \mathbb{R}^{H \times W \times C}$$

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015



Differentiable components:

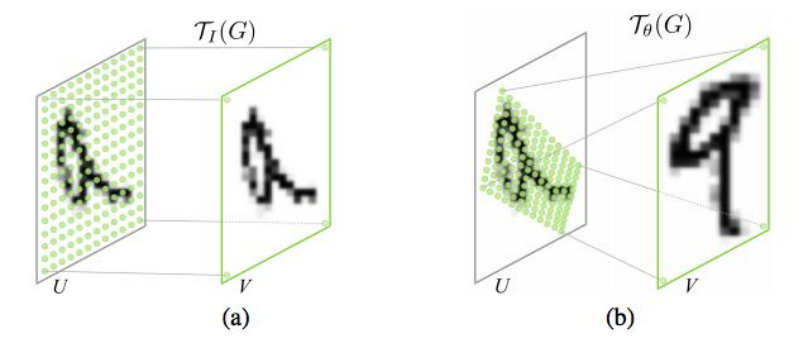
2. *Grid Generator* $\mathcal{T}_\theta(G)$

   forward pass for affine transforms:

$$\begin{pmatrix} x_i^s \\ y_i^s \end{pmatrix} = \mathcal{T}_\theta(G_i) = \mathtt{A}_\theta \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix} = \begin{bmatrix} \theta_{11} & \theta_{12} & \theta_{13} \\ \theta_{21} & \theta_{22} & \theta_{23} \end{bmatrix} \begin{pmatrix} x_i^t \\ y_i^t \\ 1 \end{pmatrix}$$

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015



Differentiable components:

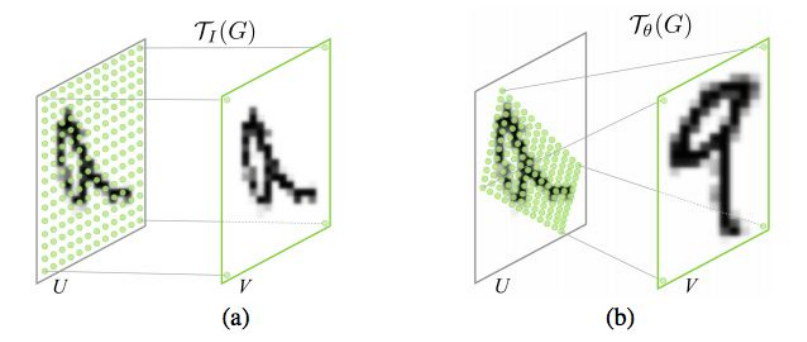
3. *Sampler:* each (x_i^s, y_i^s) in $\mathcal{T}_\theta(G_i)$ defines where a sampling kernel $k()$ is applied to U to get V

forward pass (general formula):

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c k(x_i^s - m; \Phi_x) k(y_i^s - n; \Phi_y) \quad \forall i \in [1 \ldots H'W'], \forall c \in [1 \ldots C]$$

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015



Differentiable components:

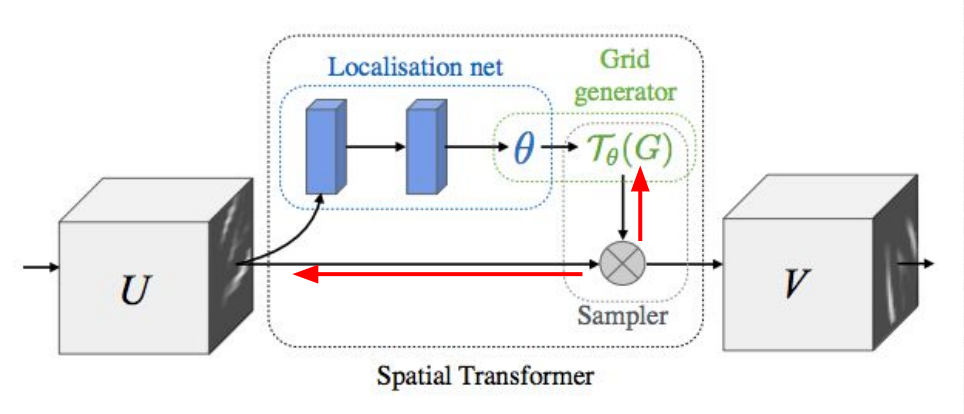
3. *Sampler:* each (x_i^s, y_i^s) in $\mathcal{T}_\theta(G_i)$ defines where a sampling kernel $k()$ is applied to U to get V

forward pass (bilinear interpolation):

$$V_i^c = \sum_n^H \sum_m^W U_{nm}^c max(0, 1 - |x_i^s - m|) max(0, 1 - |y_i^s - n|) \quad \forall i \in [1...H'W'], \forall c \in [1...C]$$

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015

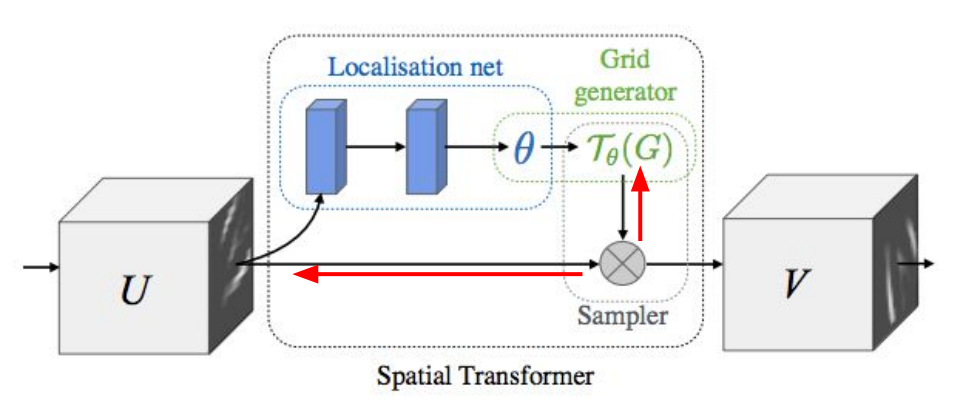


The backwards pass!
First, backprop the bilinear sampler to get: $\frac{\partial V_i^c}{\partial U_{mn}^c}$ $\frac{\partial V_i^c}{\partial x_i^s}$ $\frac{\partial V_i^c}{\partial y_i^s}$

$$\frac{\partial V_i^c}{\partial U_{mn}^c} = \sum_n^H \sum_m^W max(0, 1 - |x_i^s - m|) max(0, 1 - |y_i^s - n|)$$

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015



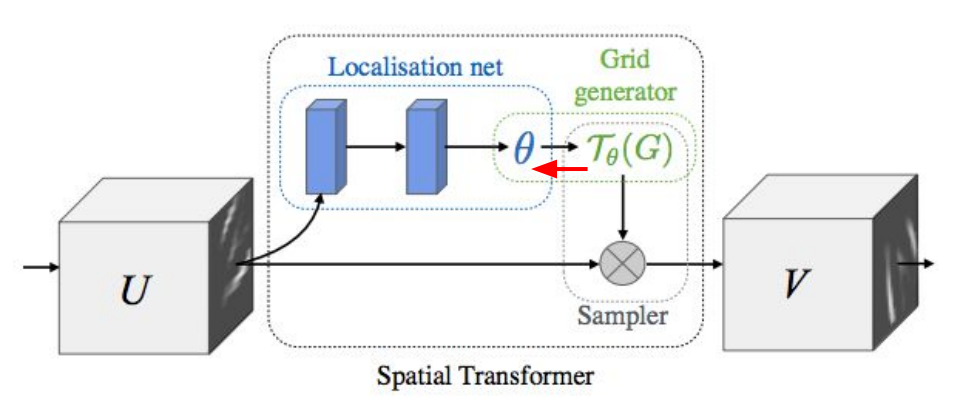
The backwards pass!
First, backprop the bilinear sampler to get: $\frac{\partial V_i^c}{\partial U_{mn}^c}$ $\frac{\partial V_i^c}{\partial x_i^s}$ $\frac{\partial V_i^c}{\partial y_i^s}$

$$\frac{\partial V_i^c}{\partial x_i^s} = \sum_n^H \sum_m^W U_{nm}^c max(0, 1 - |y_i^s - n|) \begin{cases} 0 & \text{if } |m - x_i^s| \geq 1 \\ 1 & \text{if } |m < x_i^s| \\ -1 & \text{if } |m \geq x_i^s| \end{cases}$$

note: this has discontinuities, so use subgradients

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015

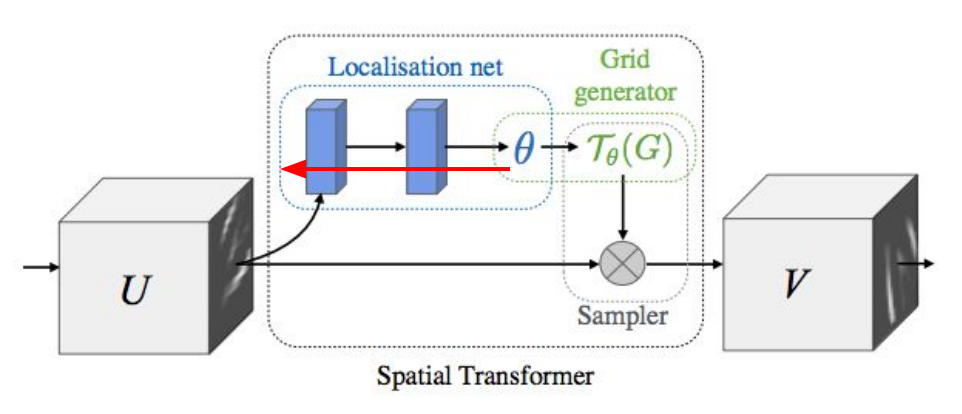


The backwards pass!
First, backprop the bilinear sampler to get: $\frac{\partial V_i^c}{\partial U_{mn}^c}$ $\frac{\partial V_i^c}{\partial x_i^s}$ $\frac{\partial V_i^c}{\partial y_i^s}$

Second, backprop $\mathcal{T}_\theta(G_i)$ using the affine transform equation to get $\frac{\partial x_i^s}{\partial \theta_i}$ $\frac{\partial y_i^s}{\partial \theta_i}$

End-to-end learning: the Spatial Transformer Net

Jaderberg et al, 2015

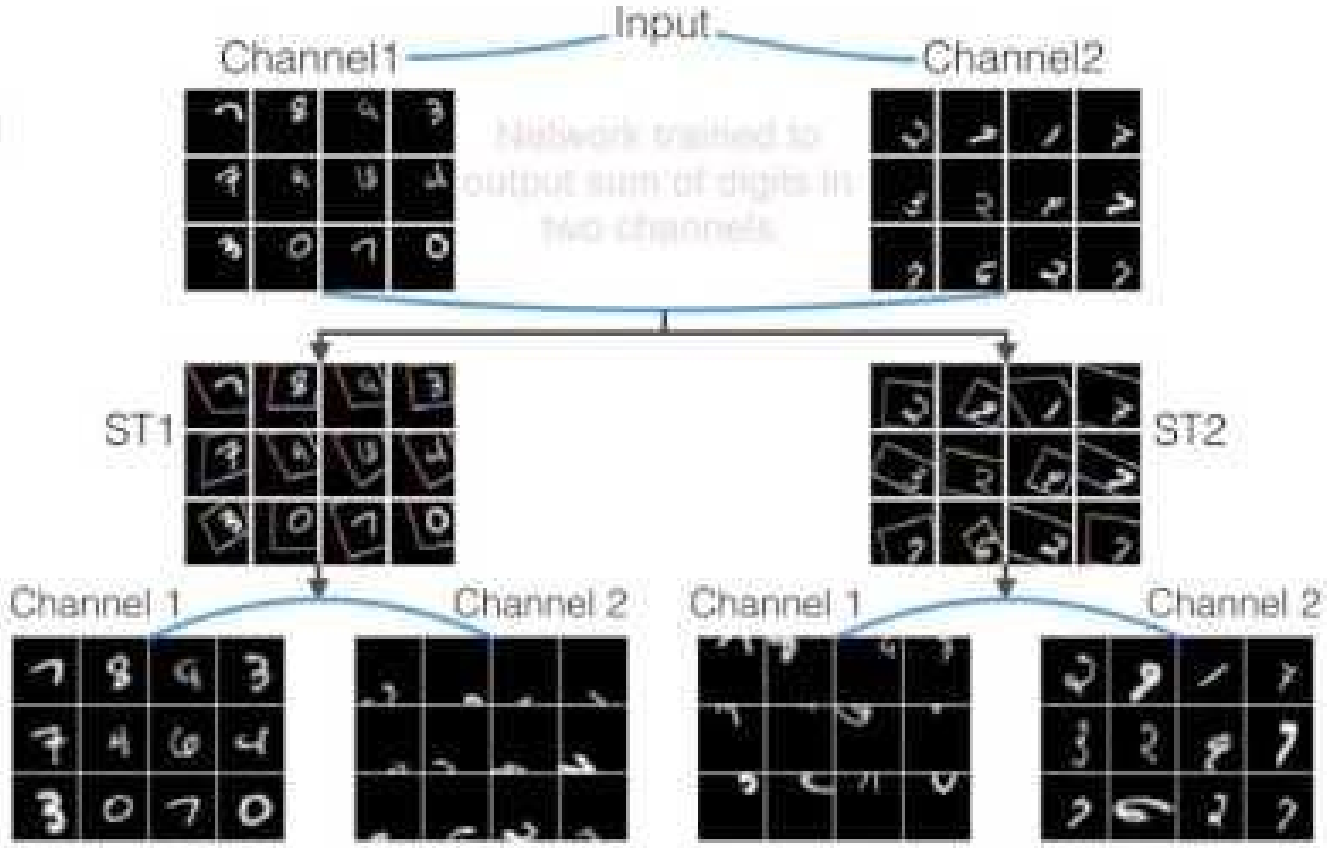


The backwards pass!
First, backprop the bilinear sampler to get: $\frac{\partial V_i^c}{\partial U_{mn}^c}$ $\frac{\partial V_i^c}{\partial x_i^s}$ $\frac{\partial V_i^c}{\partial y_i^s}$

Second, backprop $\mathcal{T}_\theta(G_i)$ using the affine transform equation to get $\frac{\partial x_i^s}{\partial \theta_i}$ $\frac{\partial y_i^s}{\partial \theta_i}$

Last, backprop the localisation net in the standard way to get $\frac{\partial \theta_i}{\partial U_{mn}^c}$

MNIST Addition



Learning relationships from data

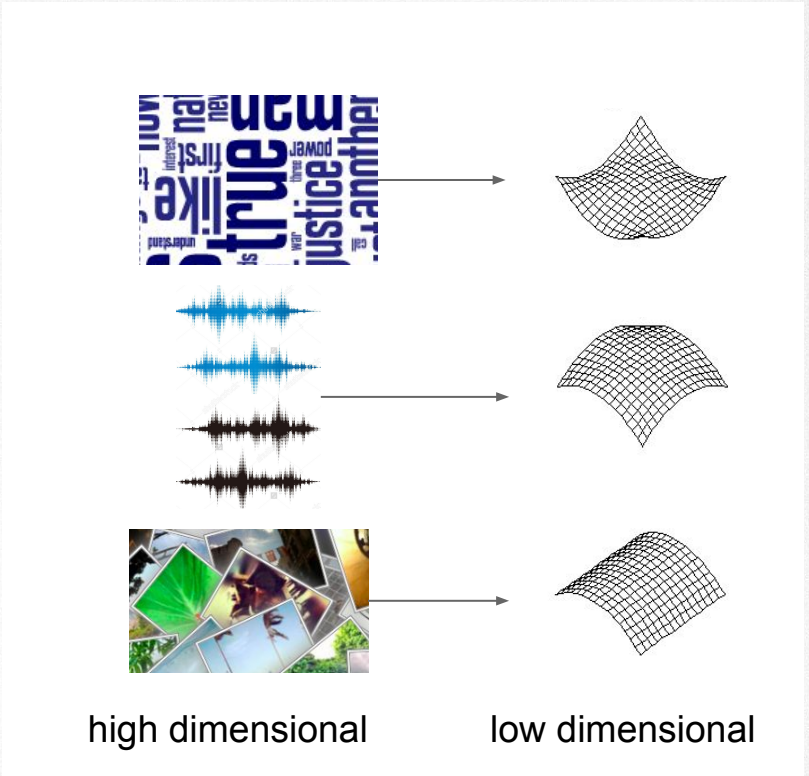
embeddings, manifolds, and clusters

Learning relationships from data

embeddings, manifolds, and clusters

Sometimes (often) the goal is not classification, nor regression.

- similarity / dissimilarity
- visualisation
- inferring relations
- clustering
- underlying structure

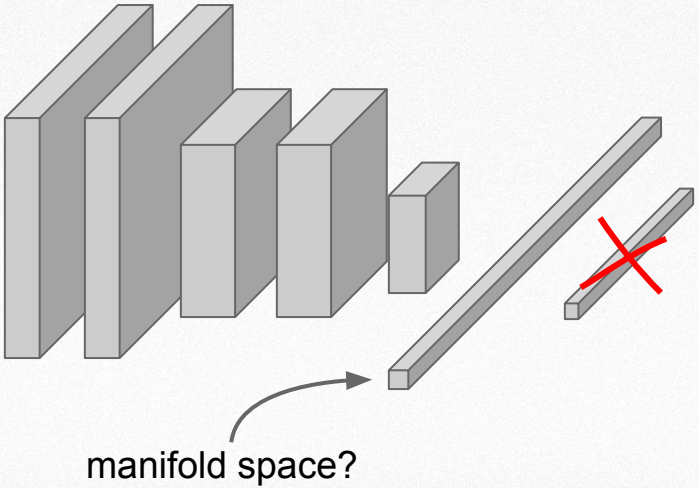


Learning relationships from data

embeddings, manifolds, and clusters

Sometimes (often) the goal is not classification, nor regression.

- similarity / dissimilarity
- visualisation
- inferring relations
- clustering
- underlying structure



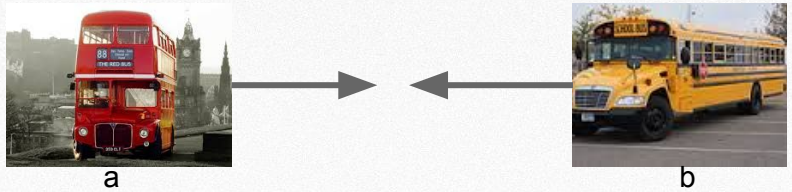
→ May **not** have training labels,
or may need to generalise to **new classes**

Learning relationships from data

embeddings, manifolds, and clusters

Classification loss associates each input with a label.

Embedding losses associate inputs with *each other*.



Learning relationships from data

embeddings, manifolds, and clusters

Designing losses for learning embeddings:

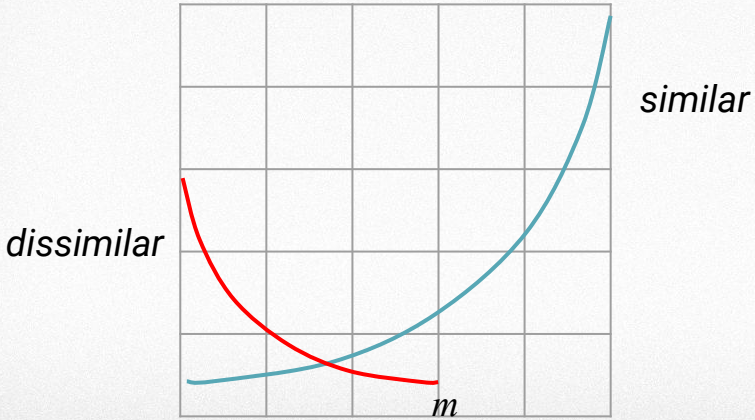
- involves distances in feature space, i.e., L2

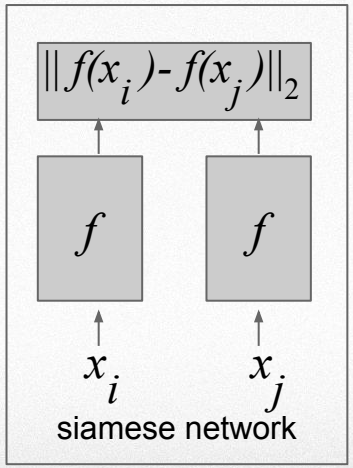
Learning relationships from data

embeddings, manifolds, and clusters

- Contrastive squared loss (per sample):

$$l(x_i, x_j, y_{ij}) = \begin{cases} \frac{1}{2}\|f(x_i) - f(x_j)\|_2^2 & \text{if } y_{ij} = 1 \\ \frac{1}{2}\max(0, m - \|f(x_i) - f(x_j)\|_2)^2 & \text{if } y_{ij} = -1 \end{cases}$$





hadsell et al, 2006

Learning relationships from data

embeddings, manifolds, and clusters

- Cosine similarity loss (per sample):

$$l(x_i, x_j, y_{ij}) = \frac{1}{2}(y_{ij} - \sigma(wd + b))^2$$

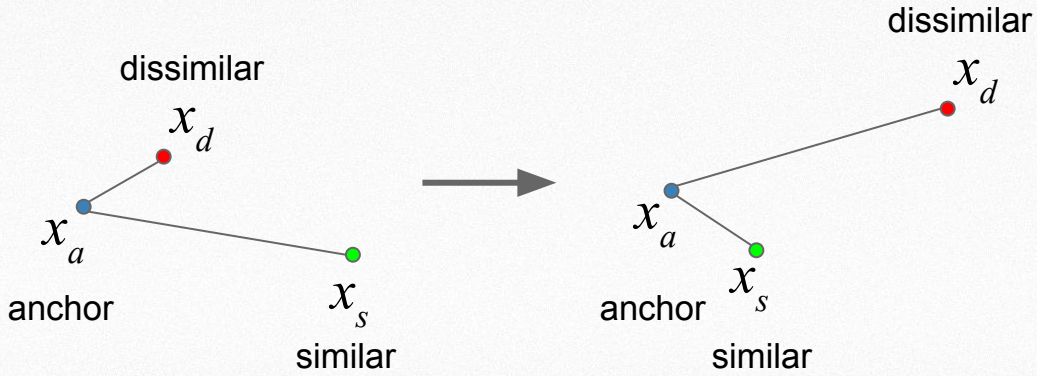
cosine similarity $$d = \frac{f(x_i) \cdot f(x_j)}{\|f(x_i)\|_2 \|f(x_j)\|_2}$$

Learning relationships from data

embeddings, manifolds, and clusters

- Triplet loss (per sample):

$$l(x_a, x_s, x_d) = [\|f(x_a) - f(x_d)\|_2^2 - \|f(x_a) - f(x_s)\|_2^2 + m]_+$$



schroff et al, 2015

FaceNet, DeepFace, and DeepID2!

all state of the art face recognition uses learned embeddings

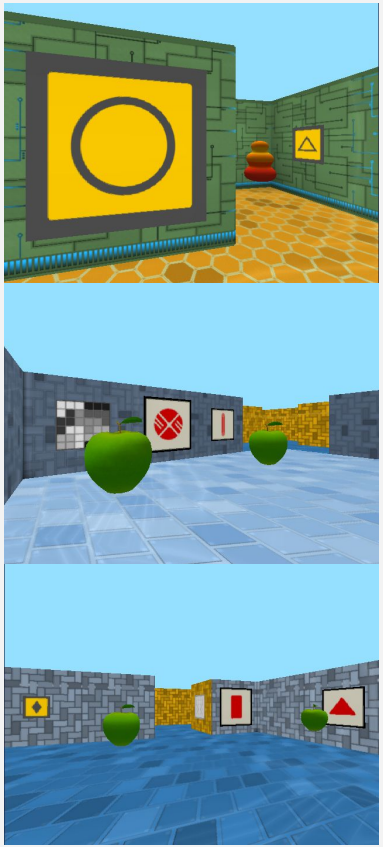
a single cluster (person) on the feature manifold learnt by FaceNet

false accepts (LFW dataset)

RL + DL: Learning to navigate in 3D mazes

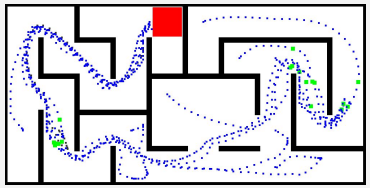
- Recent research that brings together RL plus self-supervised auxiliary losses.
- https://arxiv.org/abs/1611.03673 (Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew J. Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, Raia Hadsell)

Navigation mazes



+10

+1

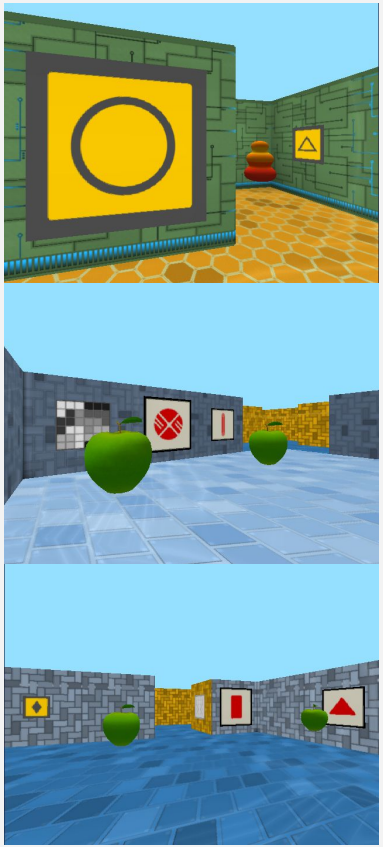


Game **episode:**

1. Random start
2. Find the goal (+10)
3. Teleport randomly
4. Re-find the goal (+10)
5. Repeat (limited time)

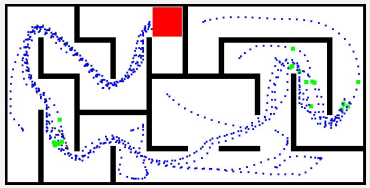
DeepMind Lab

Navigation mazes



+10

+1



Game **episode:**

1. Random start
2. Find the goal (+10)
3. Teleport randomly
4. Re-find the goal (+10)
5. Repeat (limited time)

Variants:

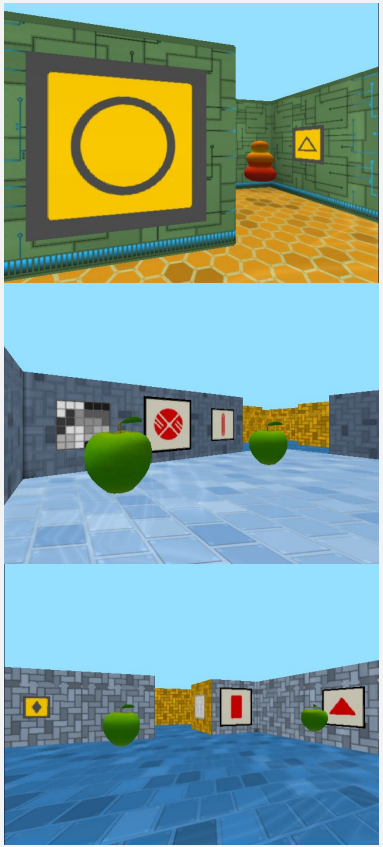
- Static maze, static goal
- Static maze, random goal
- Random maze

Observations: RGB, velocity

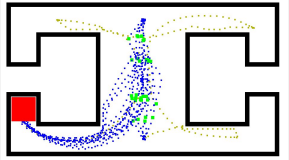
Actions: 8

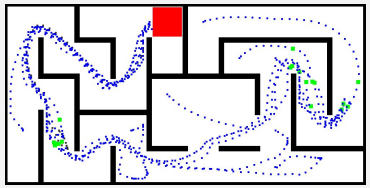
DeepMind Lab

Navigation mazes



3600 steps/episode





Game **episode:**

1. Random start
2. Find the goal (+10)
3. Teleport randomly
4. Re-find the goal (+10)
5. Repeat (limited time)

Variants:

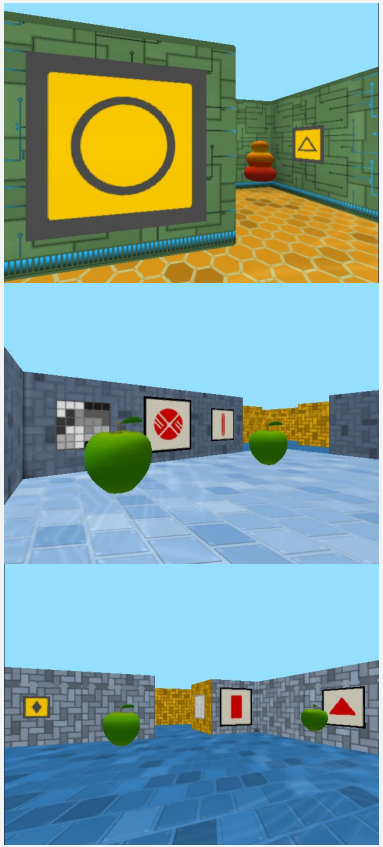
- Static maze, static goal
- Static maze, random goal
- Random maze

Observations: RGB, velocity

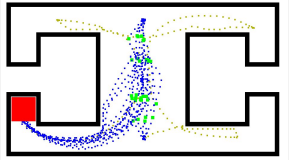
Actions: 8

DeepMind Lab

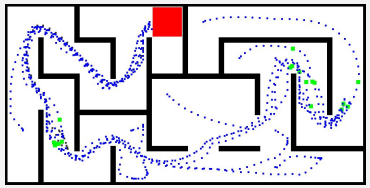
Navigation mazes



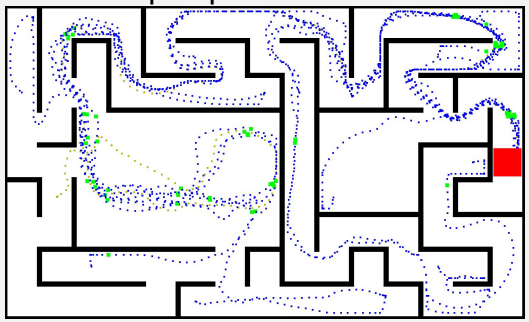
3600 steps/episode



3600 steps/episode



10800 steps/episode



Game **episode:**

1. Random start
2. Find the goal (+10)
3. Teleport randomly
4. Re-find the goal (+10)
5. Repeat (limited time)

Variants:

- Static maze, static goal
- Static maze, random goal
- Random maze

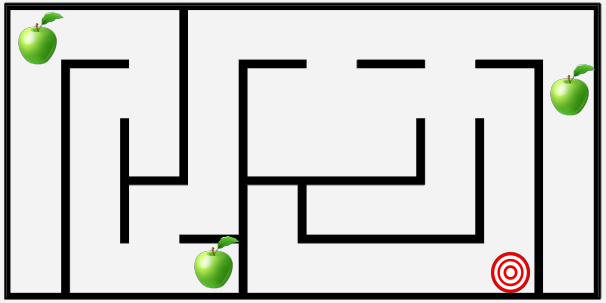
Observations: RGB, velocity

Actions: 8

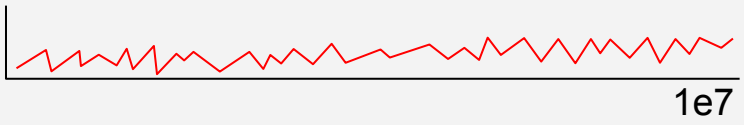
DeepMind Lab

Why is learning navigation via reinforcement learning hard?

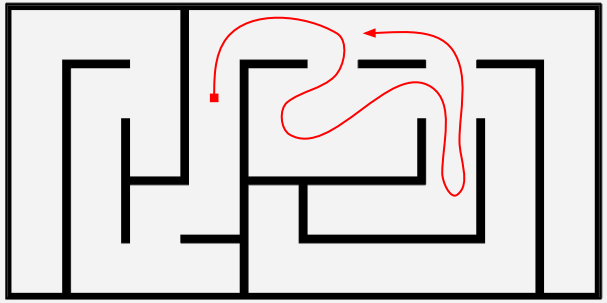
Given:
sparse rewards



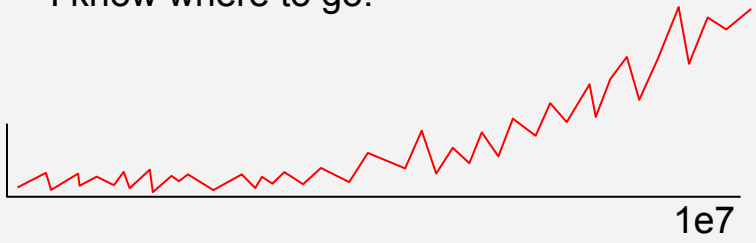
The vast and meaningless silence of an agent exploring...



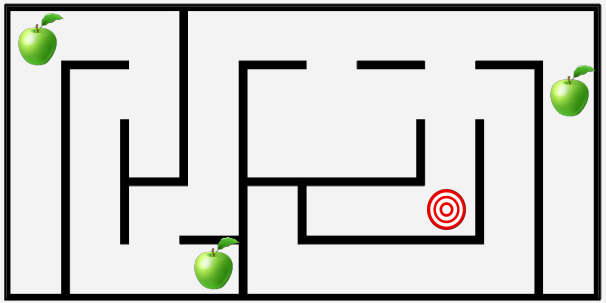
Wanted:
spatial knowledge



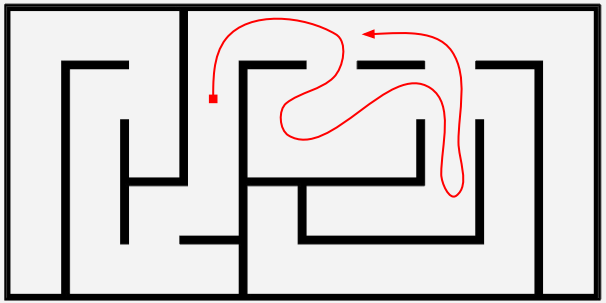
I have been here before!
I know where to go!



Given:
Sparse rewards



Wanted:
Spatial knowledge



Accelerate reinforcement learning through auxiliary losses

➔ Stable gradients help learning, even if unrelated to reward

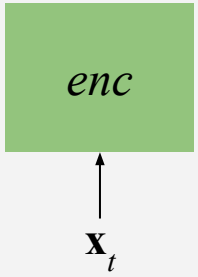
Drive spatial knowledge through choice of auxiliary tasks:

- Depth prediction
- Loop closure prediction

Assess navigation skills through position decoding

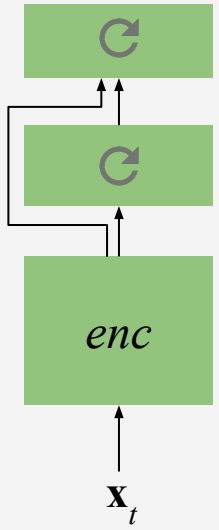
nav agent ingredients:

1. Convolutional encoder and RGB inputs



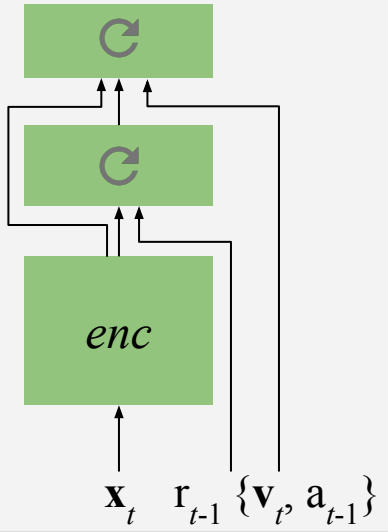
Nav agent ingredients:

1. Convolutional encoder and RGB inputs
2. Stacked LSTM



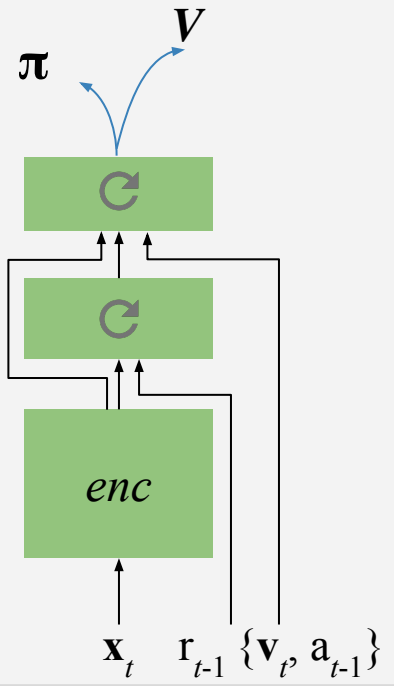
nav agent ingredients:

1. Convolutional encoder and RGB inputs
2. Stacked LSTM
3. Additional inputs (reward, action, and velocity)



Nav agent ingredients:

1. Convolutional encoder and RGB inputs
2. Stacked LSTM
3. Additional inputs (reward, action, and velocity)
4. RL: Asynchronous advantage actor critic (A3C)

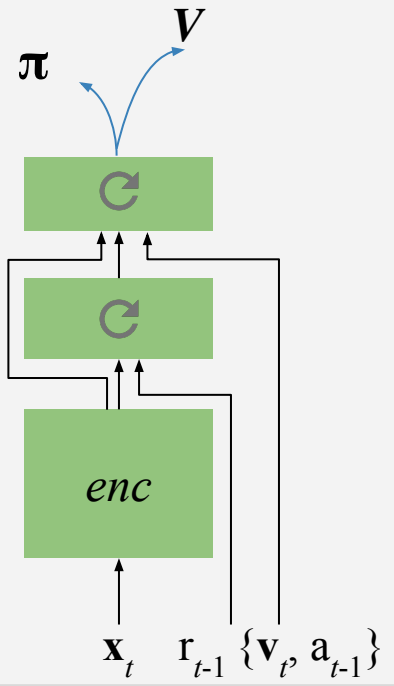


Nav agent ingredients:

1. Convolutional encoder and RGB inputs
2. Stacked LSTM
3. Additional inputs (reward, action, and velocity)
4. RL: Asynchronous advantage actor critic (A3C)

Value $V(s_t; \theta_V)$ and policy $\pi(a_t | s_t; \theta)$ are updated with estimate of policy gradient given by the k-step advantage function

$$A(s_t, a_t; \theta_V) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(s_{t+k}; \theta_V) - V(s_t; \theta_V)$$

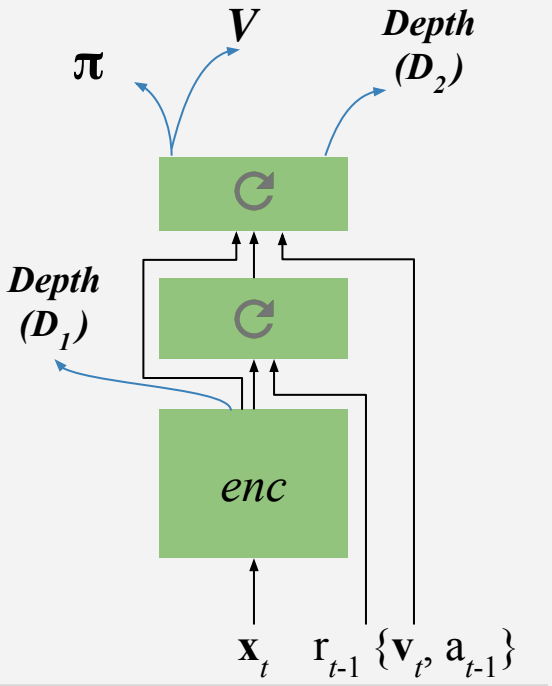


DeepMind

Mnih et al. (2016)

Nav agent ingredients:

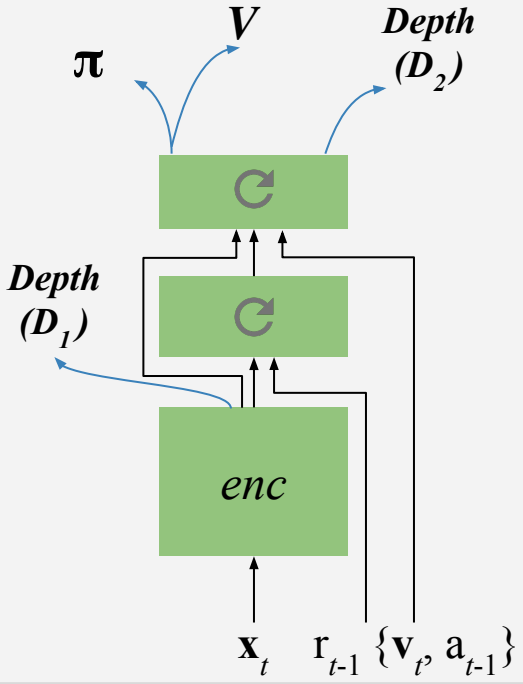
1. Convolutional encoder and RGB inputs
2. Stacked LSTM
3. Additional inputs (reward, action, and velocity)
4. RL: Asynchronous advantage actor critic (A3C)
5. Aux task 1: Depth predictors



Nav agent ingredients:

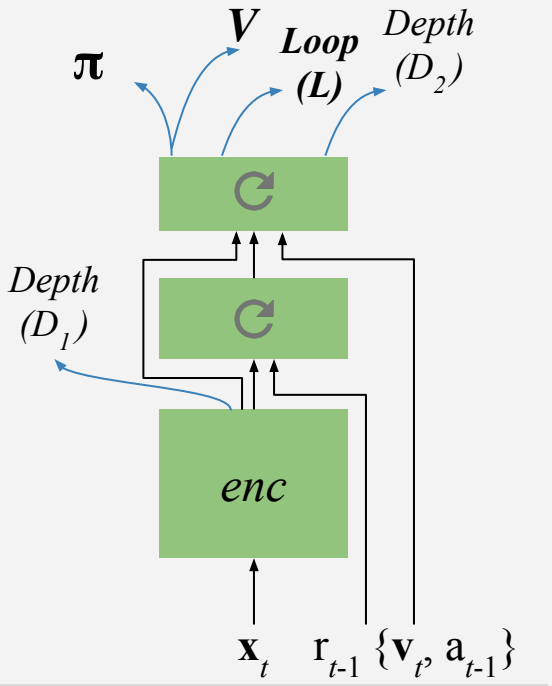
1. Convolutional encoder and RGB inputs
2. Stacked LSTM
3. Additional inputs (reward, action, and velocity)
4. RL: Asynchronous advantage actor critic (A3C)
5. Aux task 1: Depth predictors

Coarse prediction of depth (4x16). 
Single-layer MLP (128 hid) has independent softmax outputs that predict depth per position.



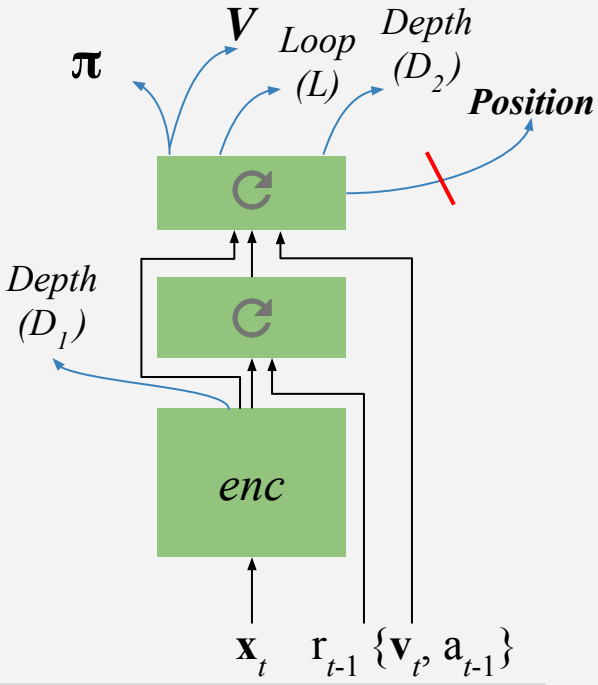
Nav agent ingredients:

1. Convolutional encoder and RGB inputs
2. Stacked LSTM
3. Additional inputs (reward, action, and velocity)
4. RL: Asynchronous advantage actor critic (A3C)
5. Aux task 1: Depth predictor
6. Aux task 2: Loop closure predictor

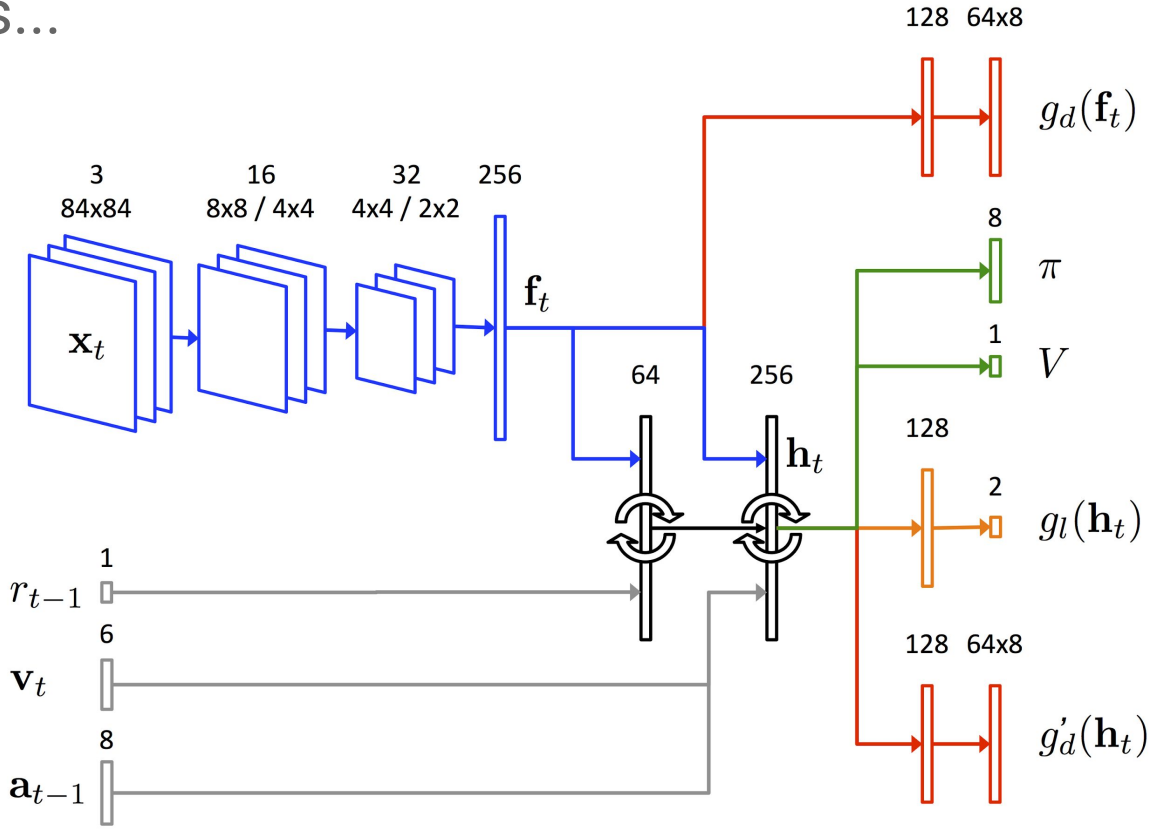


Nav agent ingredients:

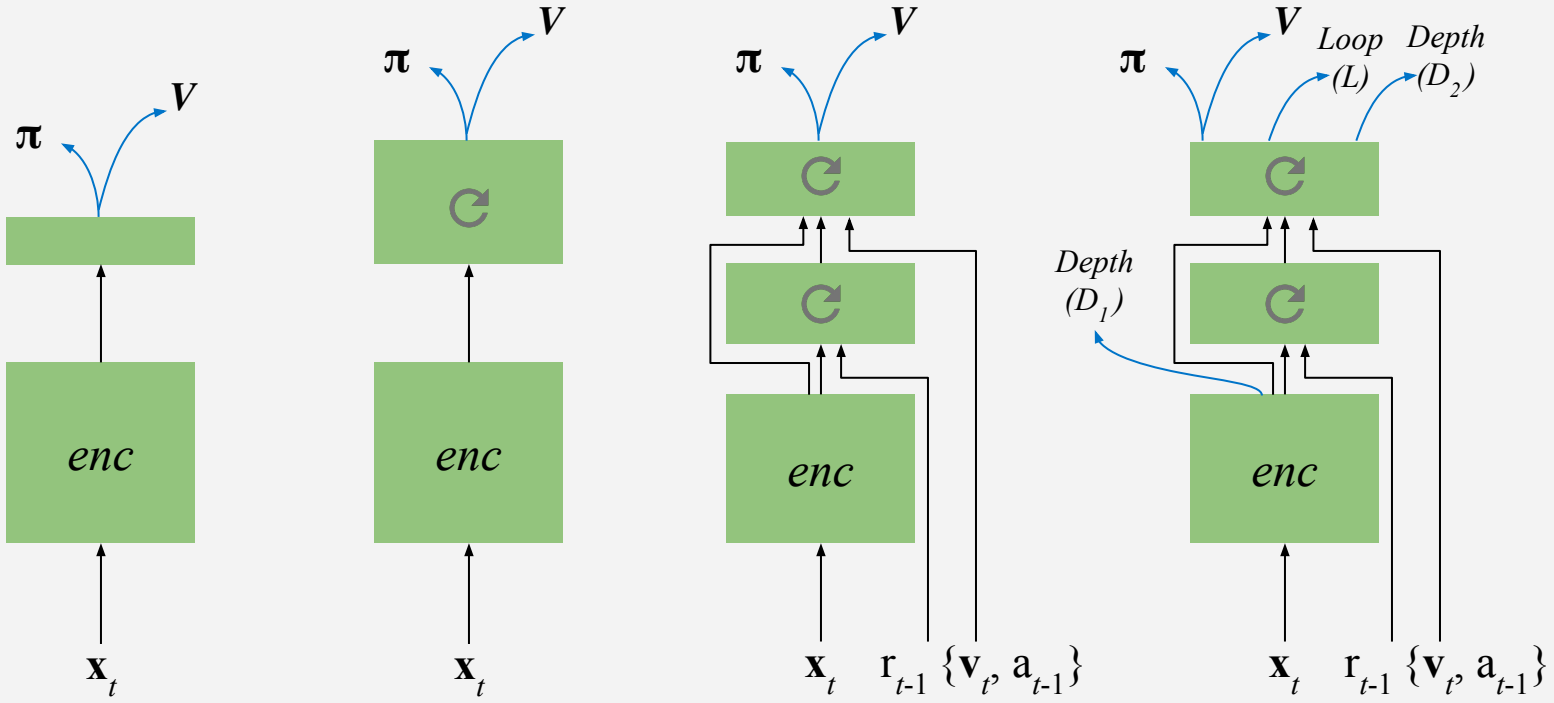
1. Convolutional encoder and RGB inputs
2. Stacked LSTM
3. Additional inputs (reward, action, and velocity)
4. RL: Asynchronous advantage actor critic (A3C)
5. Aux task 1: Depth predictor
6. Aux task 2: Loop closure predictor
7. For analysis: Position decoder



The details...


3
84x84
16
8x8 / 4x4
32
4x4 / 2x2
256
$\mathbf{x}_t$
$\mathbf{f}_t$
64
256
$\mathbf{h}_t$
128
64x8
$g_d(\mathbf{f}_t)$
8
π
1
V
128
2
$g_l(\mathbf{h}_t)$
128
64x8
$g'_d(\mathbf{h}_t)$
1
r_{t-1}
6
$\mathbf{v}_t$
8
$\mathbf{a}_{t-1}$

Variations in architecture

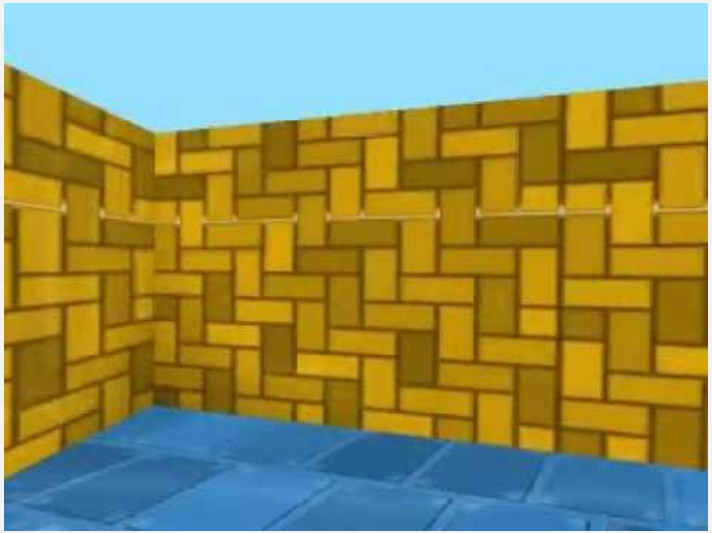


a. FF A3C b. LSTM A3C c. Nav A3C d. Nav A3C +D_1D_2L

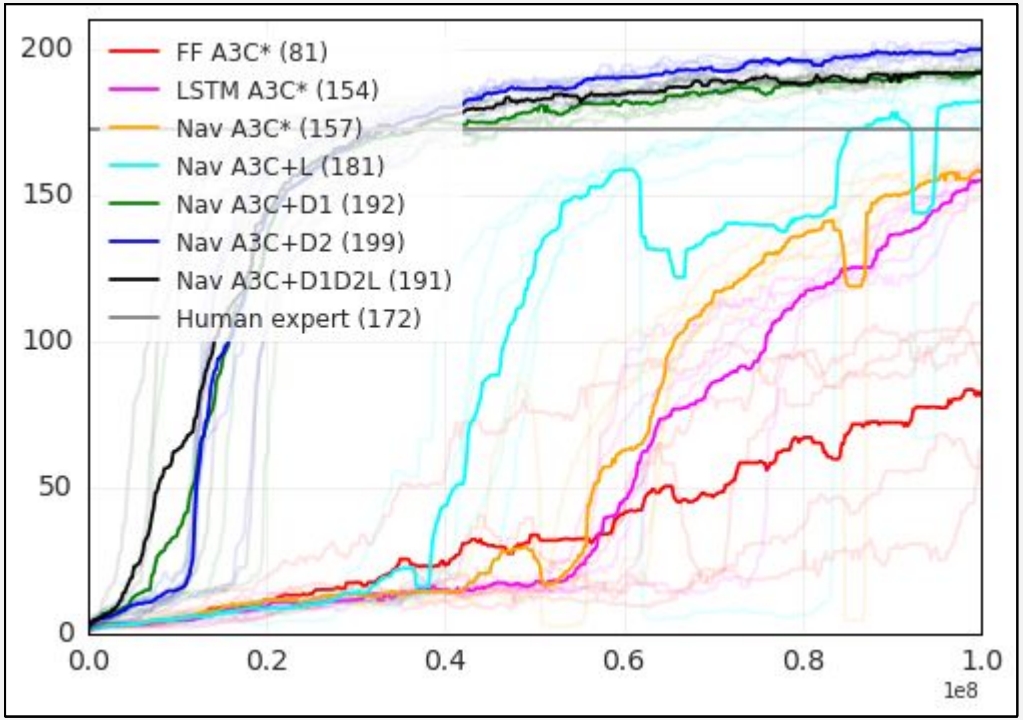
Results on large maze with static goal

+10

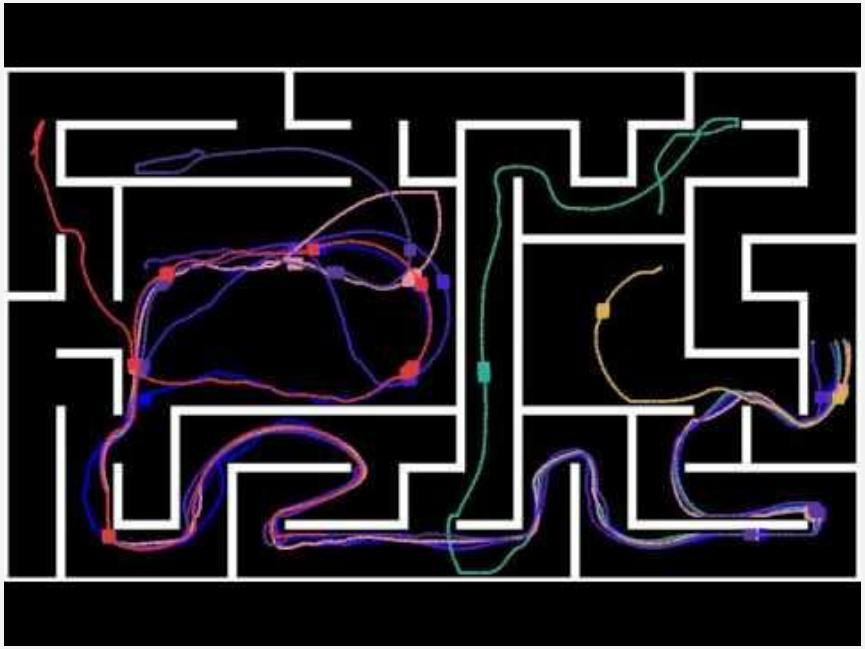
+1

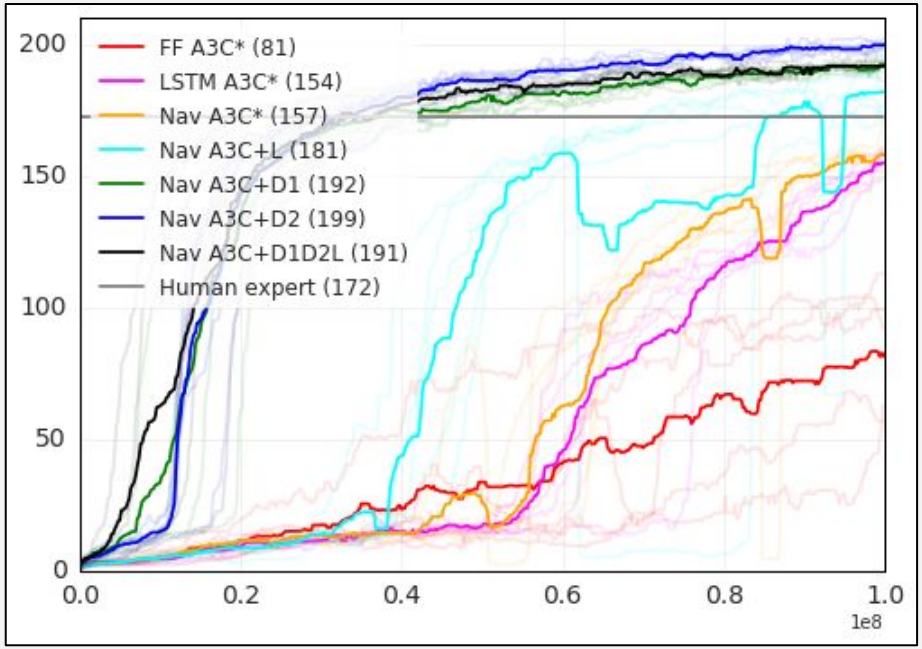


https://www.youtube.com/watch?v=zHhbypmKaj0



Results on large maze with static goal

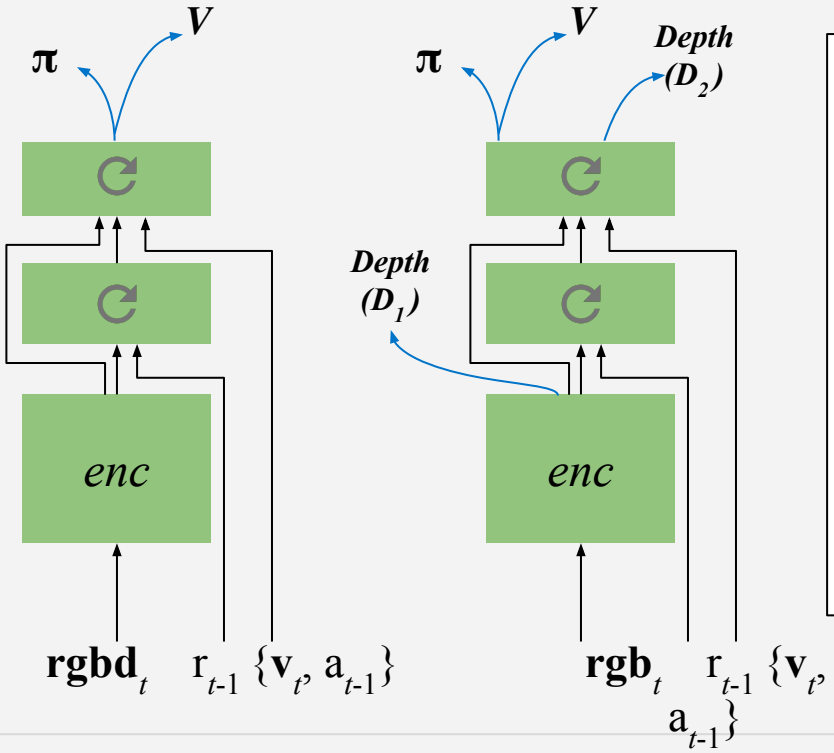


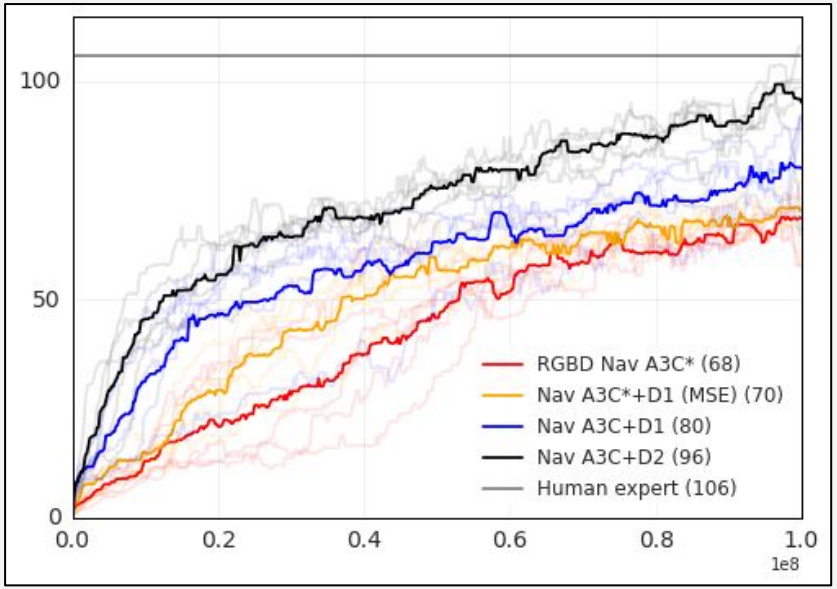


https://www.youtube.com/watch?v=Phpns44aeNo

Should depth be an input? Or a target?

Should depth be an input? Or a target?





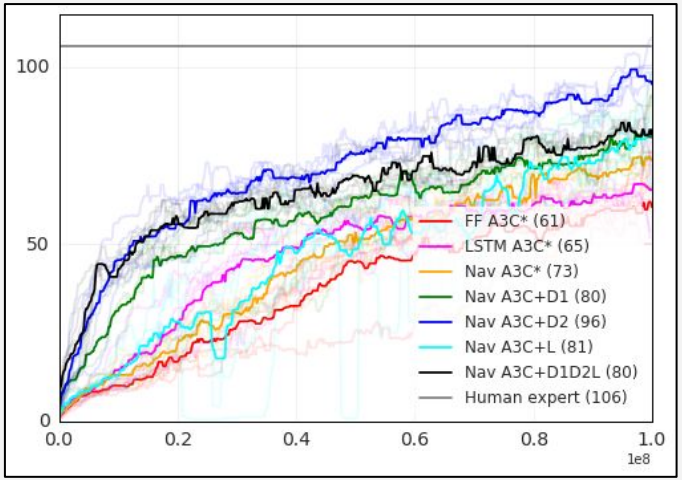
Results with random goal

- Nav++ agent achieves human level performance on smaller maze.

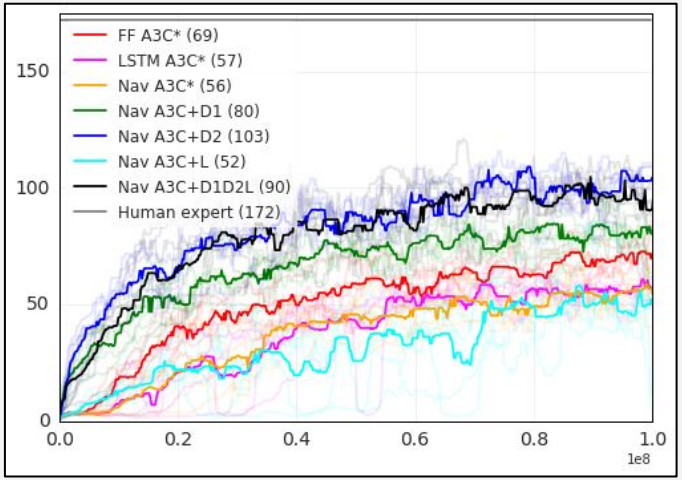
Is the agent remembering the goal location?

- Mean time to first goal find of episode: **14.0** sec
- Mean time to subsequent goal finds: **7.2** sec
- Not as impressive for large mazes: **15.4** sec vs **15.0** sec.

small

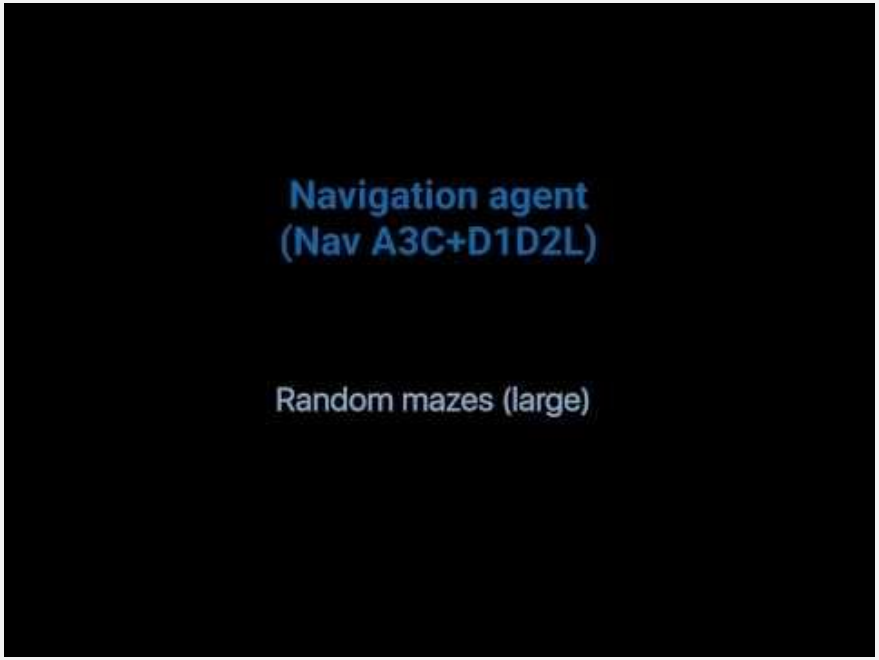


large



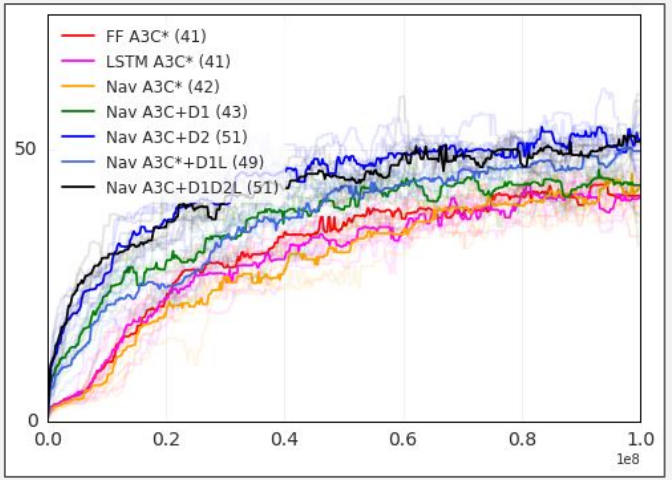
Results in random mazes

- Nav++ agent still outperforms Nav and baseline agents.

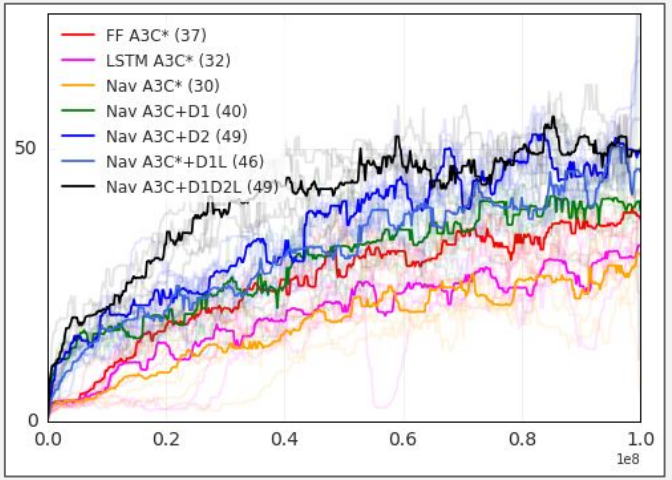


https://www.youtube.com/watch?v=EKXQAjoNdGM

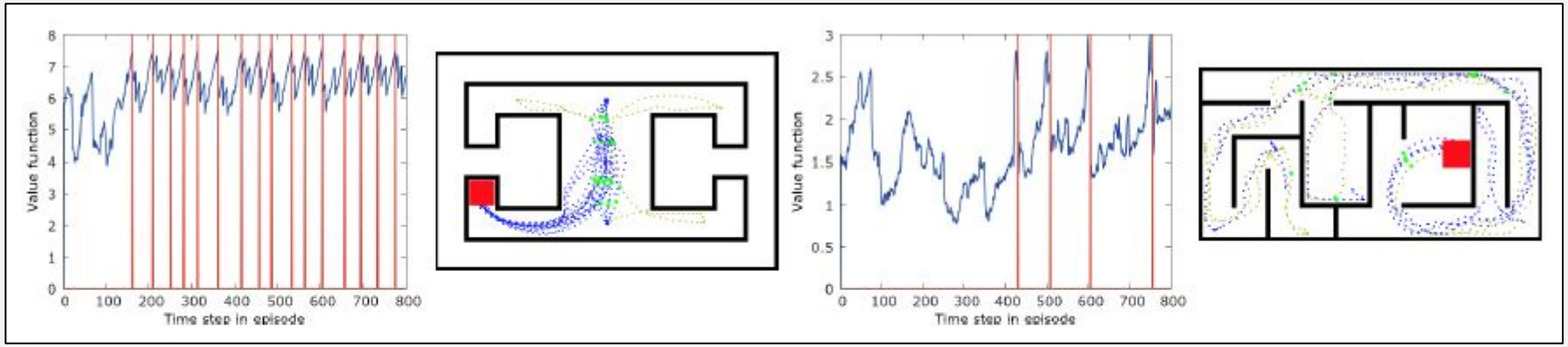
small



large

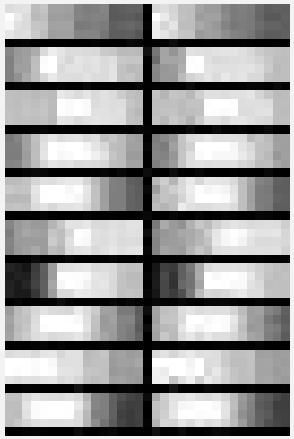


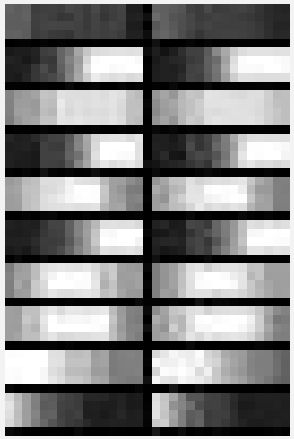
Latency to goal (as the agent returns)

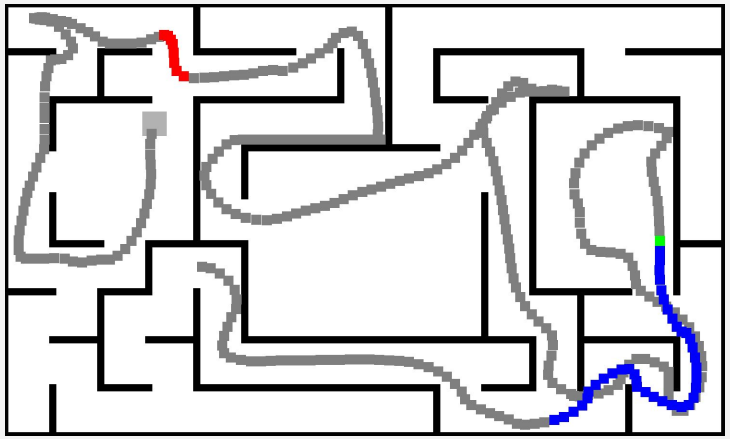


- Trajectories of the Nav A3C+D+L agent in the I-maze and random goal maze over the course of one episode
- Value function and goal finding (red lines) are shown

Performance on aux tasks

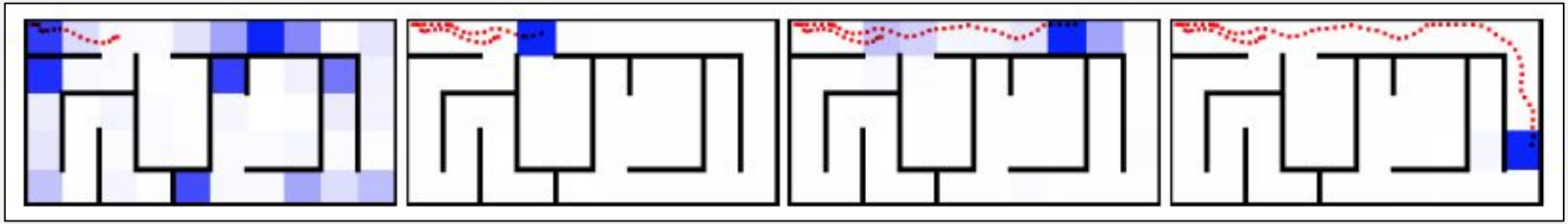


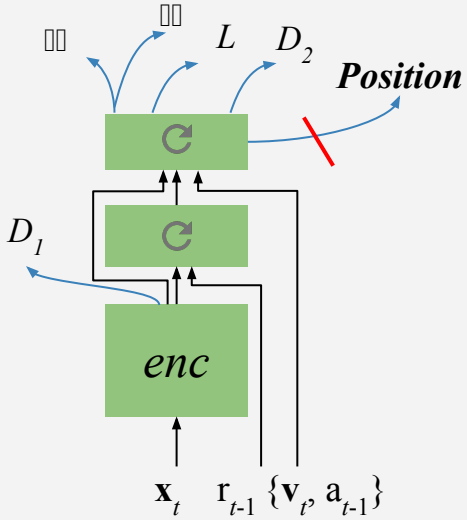




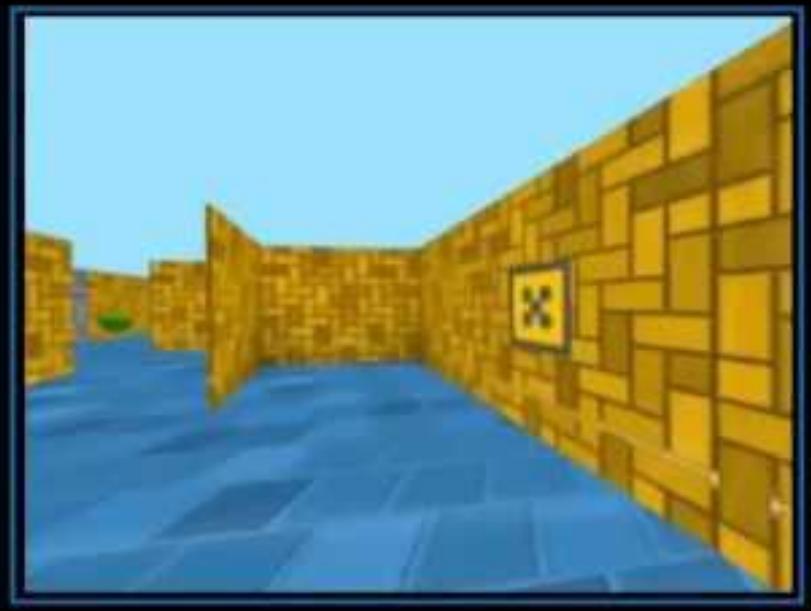
Both train to a high accuracy (not really the point though!)

Position decoding

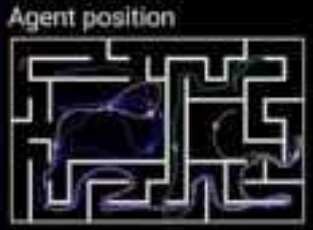


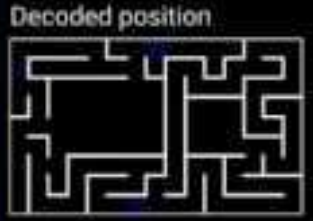


- Trajectories of the Nav A3C+D+L agent in the random goal maze
- Position likelihoods are overlaid (predicted from LSTM hiddens)
- Initial uncertainty gives way to accurate position estimation.



Large static maze





https://www.youtube.com/watch?v=INoaTyMZsWI

Overview

1. End-to-end learning with more complex models and tasks
    a. Beyond classification: detection, segmentation, videos
    b. End-to-end case study: spatial transformer networks
2. Learning without labels: embeddings and manifolds
3. Putting many things together - RL + sequence learning + auxiliary losses