Carnegie Mellon
School of Computer Science

Deep Reinforcement Learning and Control

Imitation Learning

Lecture 14, CMU 10703

Katerina Fragkiadaki

So far in the course

Reinforcement Learning: Learning policies guided by sparse rewards, e.g., win or not the game.

- Good: simplest, cheapest form of supervision
- Bad: High sample complexity

Where is it successful so far?

- in simulation, where we can afford a lot of trials, easy to parallelize
- not in robotic systems:
    1. action execution takes long
    2. we cannot afford to fail
    3. safety concerns

Crusher robot

Learning from Demonstration for Autonomous Navigation in Complex Unstructured

Reward shaping

Ideally we want dense in time rewards to closely guide the agent closely along the way.

Who will supply those shaped rewards?

1. We will manually design them: *"cost function design by hand remains one of the 'black arts' of mobile robotics, and has been applied to untold numbers of robotic systems"*
2. We will learn them from demonstrations: *"rather than having a human expert tune a system to achieve desired behavior, the expert can demonstrate desired behavior and the robot can tune itself to match the demonstration"*

Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain, Silver et al. 2010

Reward shaping

Ideally we want dense in time rewards to closely guide the agent closely along the way.

Who will supply those shaped rewards?

1. We will manually design them: *"cost function design by hand remains one of the 'black arts' of mobile robotics, and has been applied to untold numbers of robotic systems"*
2. We will learn them from demonstrations: *"rather than having a human expert tune a system to achieve desired behavior, the expert can demonstrate desired behavior and the robot can tune itself to match the demonstration"*

Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain, Silver et al. 2010

Learning from Demonstrations

Learning from demonstrations a.k.a. Imitation Learning:

Supervision through an expert (teacher) that provides a set of demonstration trajectories: sequences of states and actions.

Imitation learning is useful when is easier for the expert to demonstrate the desired behavior rather than:

a) coming up with a reward that would generate such behavior,

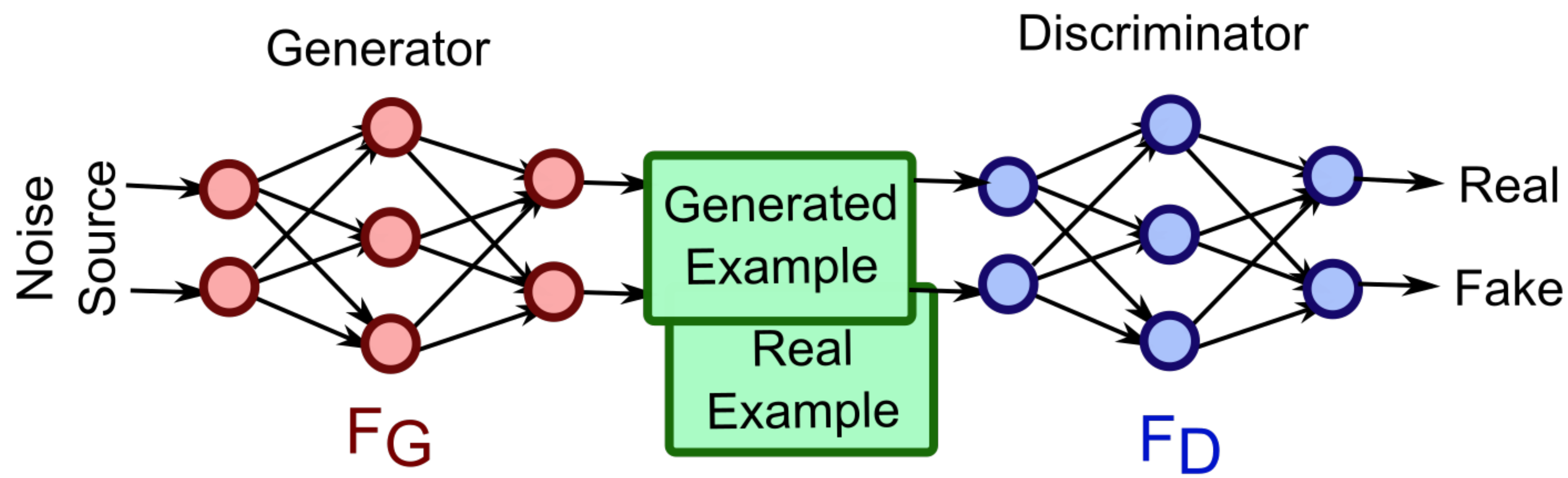
b) coding up the desired policy directly.

The Imitation Learning problem

The agent (learner) needs to come up with a policy whose resulting state, action trajectory distribution matches the expert trajectory distribution.

Does this remind us of something…?

GANs! Generative Adversarial Networks (on state-action trajectories)



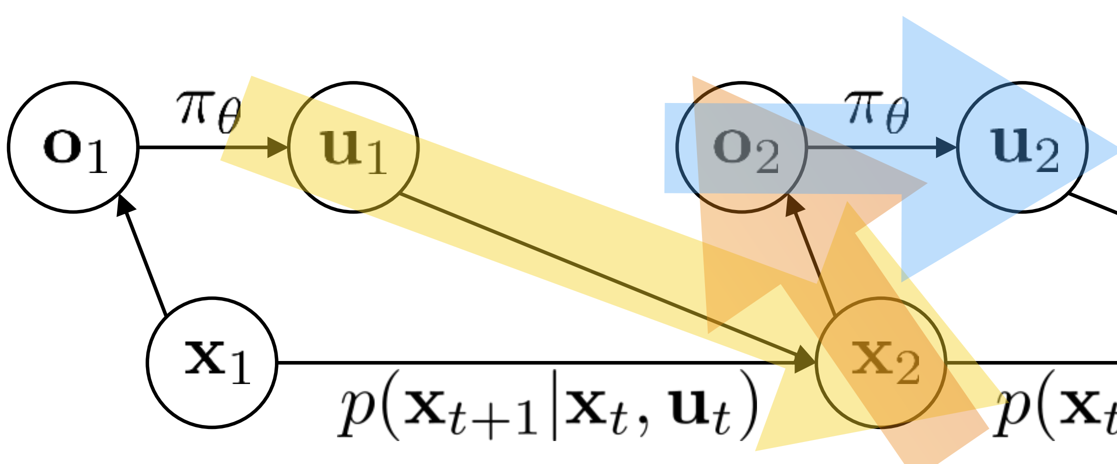
Generative Adversarial Networks, Goodfellow et al. 2014

The Imitation Learning problem: Challenge

Actions along the trajectories are interdependent, as actions determine state transitions and thus states and actions down the road.

interdependent labels -> structure prediction

Action interdependence in time:



Algorithms developed in Robotics for imitation learning found applications in structured predictions problems, such as, sequence generation/labelling e.g. parsing.

Imitation Learning

For taking this structure into account, numerous formulations have been proposed:

- Direct: Supervised learning for policy (mapping states to actions) using the demonstration trajectories as ground-truth(a.k.a. behavior cloning) + ways to handle the neglect of action interdependence.

- Indirect: Learning the latent rewards/goals of the teacher and planning under those rewards to get the policy, a.k.a. Inverse Reinforcement Learning (next lecture)

Experts can be:

- Humans

- Optimal or near Optimal Planners/Controllers

Outline

This lecture

- Behavior Cloning: Imitation learning as supervised learning
- Compounding errors
- Demonstration augmentation techniques
- DAGGER
- Structured prediction as Decision Making (learning to search)
- Imitating MCTS

Next lecture:

- Inverse reinforcement learning
- Feature matching
- Max margin planning
- Maximum entropy IRL
- Adversarial Imitation learning

Outline

This lecture

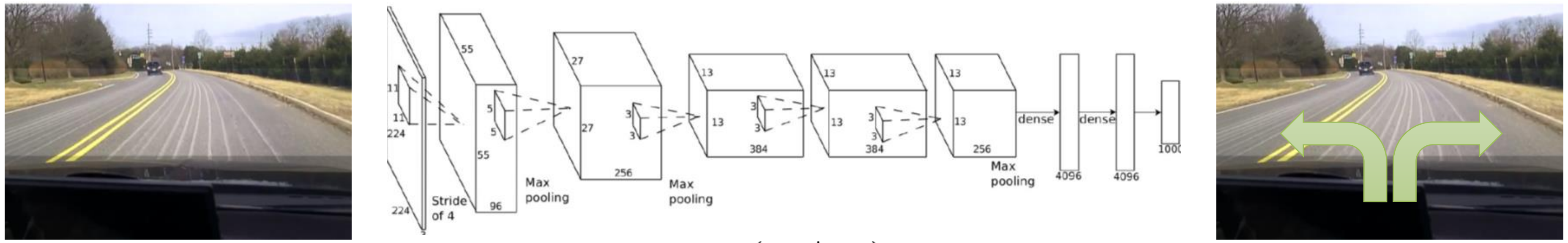
- Behavior Cloning: Imitation learning as supervised learning
- Compounding errors
- Demonstration augmentation techniques
- DAGGER
- Structured prediction as Decision Making (learning to search)
- Imitating MCTS

Next lecture:

- Inverse reinforcement learning
- Feature matching
- Max margin planning
- Maximum entropy IRL
- Adversarial Imitation learning

Imitation Learning for Driving

Driving policy: a mapping from (history of) observations to steering wheel angles

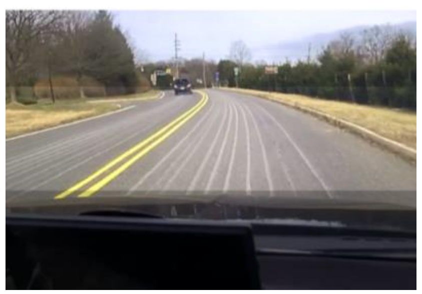


$\mathbf{o}_t$ $\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$ $\mathbf{u}_t$

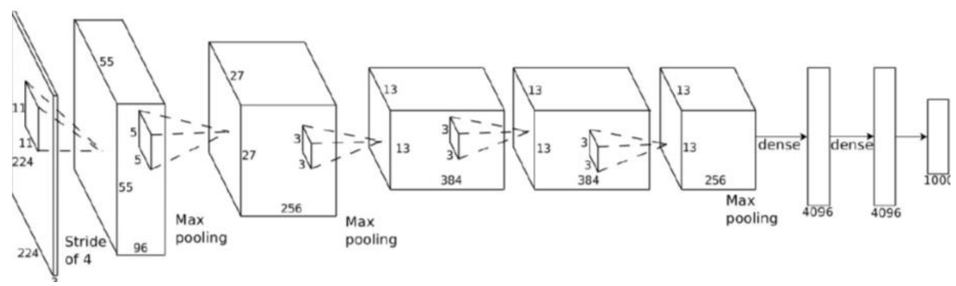
End to End Learning for Self-Driving Cars, Bojarski et al. 2016

Imitation Learning as Supervised Learning

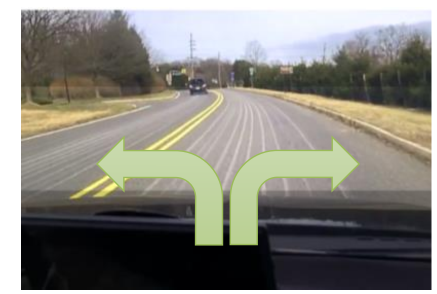
Driving policy: a mapping from (history of) observations to steering wheel angles



$\mathbf{o}_t$



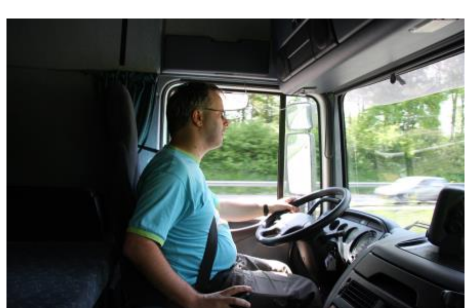
$\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$



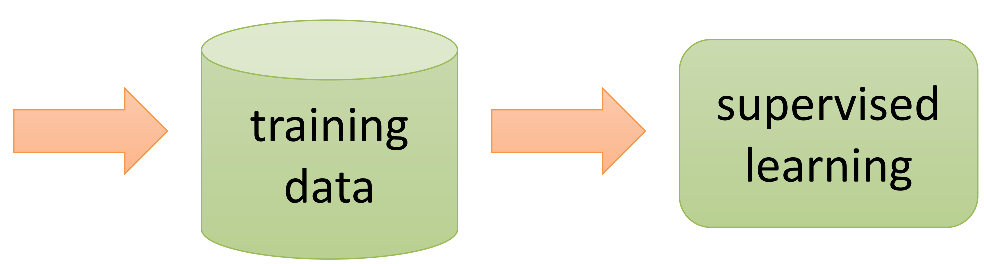
$\mathbf{u}_t$

Behavior Cloning=Imitation Learning as Supervised learning

- Assume actions in the expert trajectories are i.i.d.
- Train a classifier or regressor to map observations to actions at each time step of the trajectory.



$\mathbf{o}_t$
$\mathbf{u}_t$



$\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$

End to End Learning for Self-Driving Cars, Bojarski et al. 2016

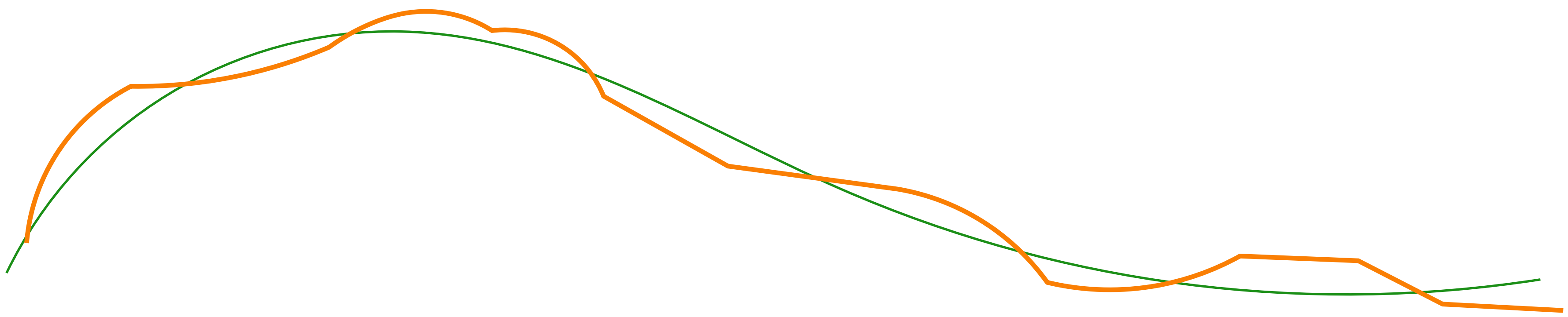
Classifier or regressor?

Because multiple actions u may be plausible at any given observation o, policy network $p_{\pi_\theta}(u_t|o_t)$ usually is not a regressor but rather:

- A classifier (e.g., softmax output and cross-entropy loss, after discretizing the action space)
- $J(\theta) = -\sum_{i=1}^{m}\sum_{k=1}^{K} 1_{y(i)=k} \log[P(y_{(i)} = k|x_{(i)};\theta)]$
- A GMM (mixture components weights, means and variances are parametrized at the output of a neural net, minimize GMM loss, (e.g., Hand writing generation Graves 2013)

- A stochastic network (previous lecture)

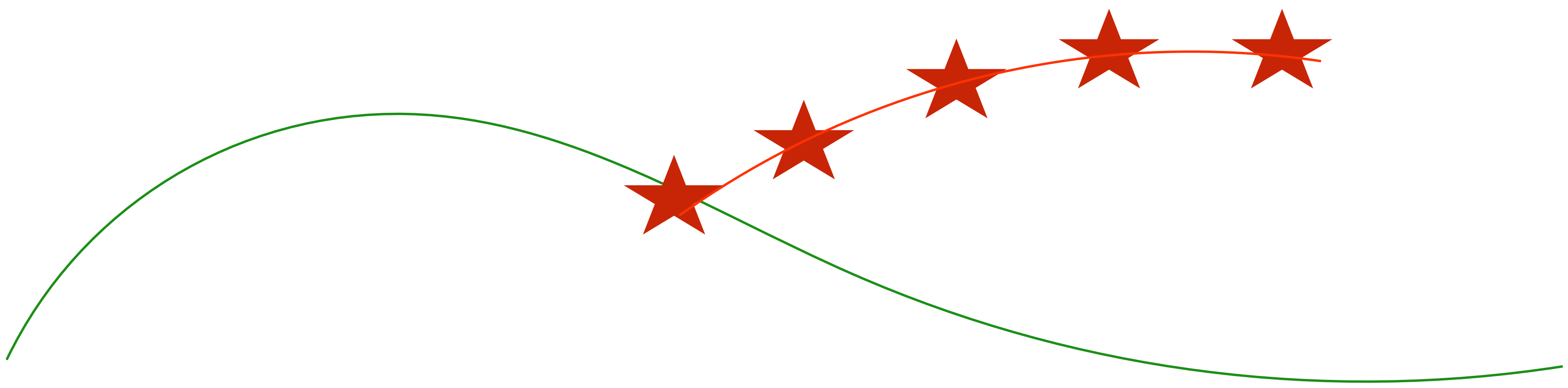
Independent in time errors



error at time t with probability ε

E[Total errors] ≲ εT

Compounding Errors



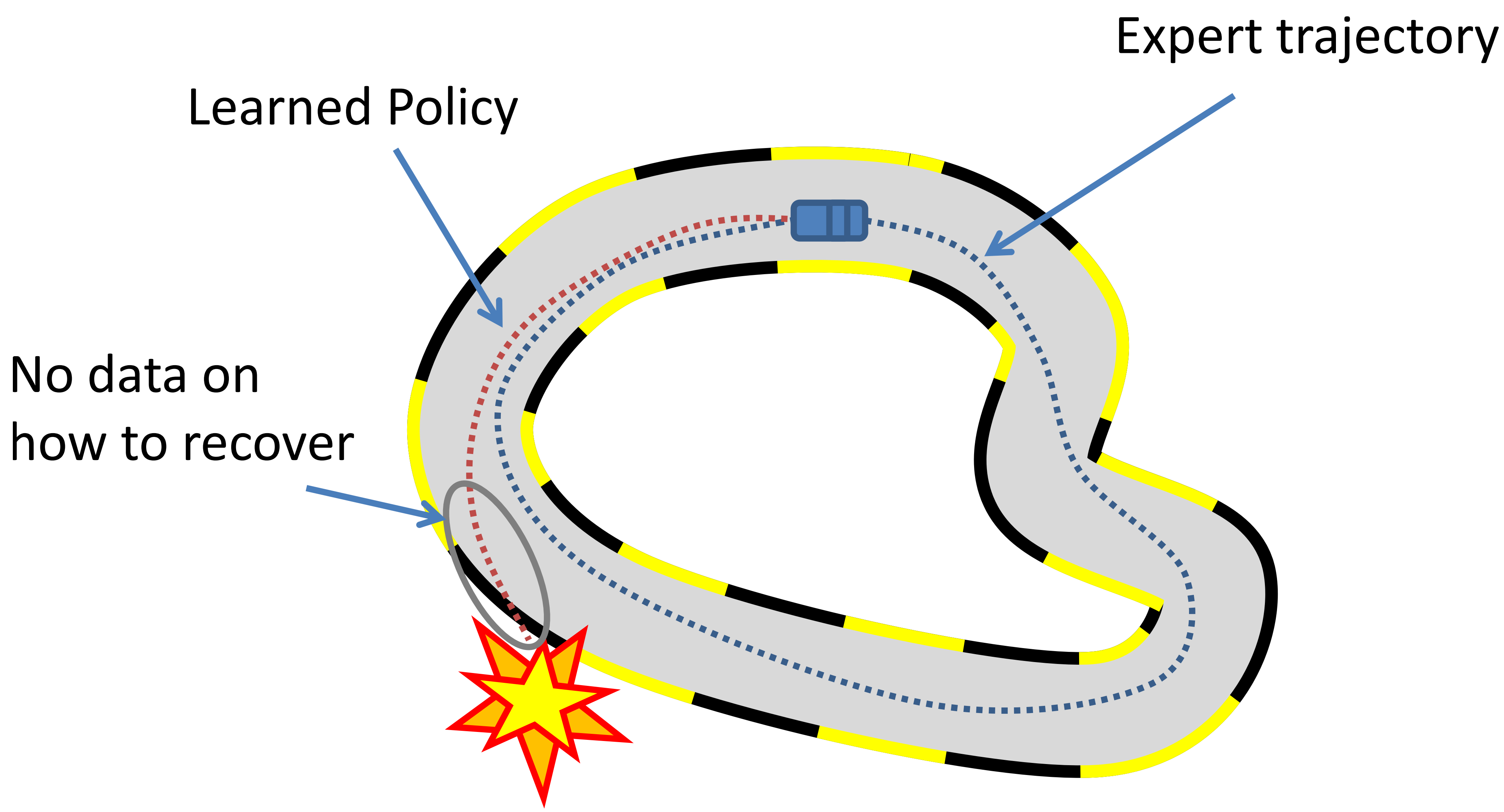
error at time t with probability ε

E[Total errors] ≲ ε(T + (T-1) + (T-2) + …+ 1) ∝ εT^2

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross et al. 2011

Data Distribution Mismatch!

$$p_{\pi^*}(o_t) \neq p_{\pi_\theta}(o_t)$$



Data Distribution Mismatch!

	supervised learning	supervised learning + control (NAIVE)
train	$(x,y) \sim D$	$s \sim d_{\pi^*}$
test	$(x,y) \sim D$	$s \sim d_{\pi}$

SL succeeds when training and test data distributions match, that is a fundamental assumption.

Solutions

Change $p_{\pi^*}(\mathrm{o}_t)$ using demonstration augmentation!

Add examples in expert demonstration trajectories to cover the states/observations points where the agent will land when trying out its own policy.

Outline

This lecture

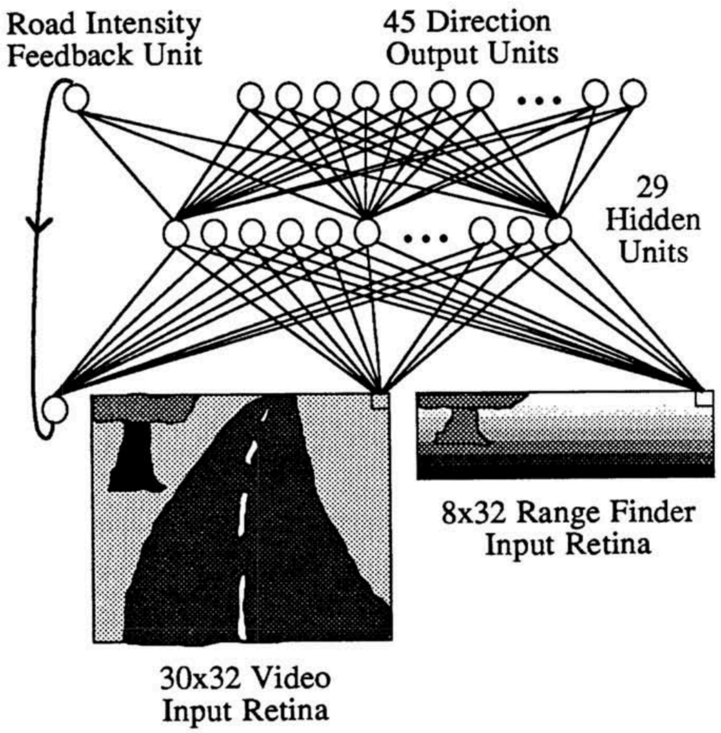
- Behavior Cloning: Imitation learning as supervised learning
- Compounding errors
- Demonstration augmentation techniques
- DAGGER
- Structured prediction as Decision Making (learning to search)
- Imitating MCTS

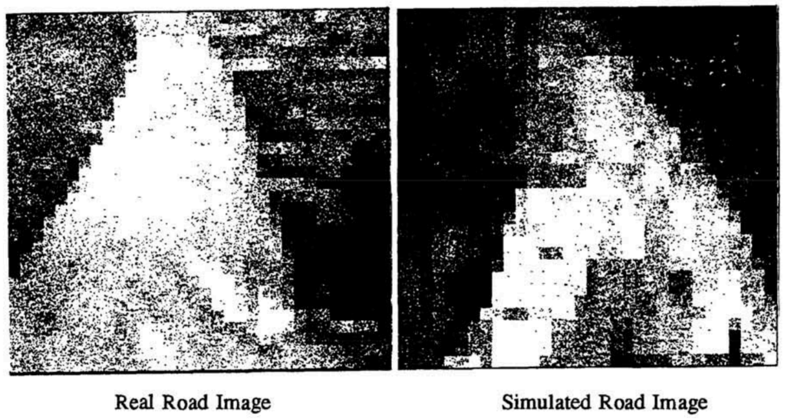
Next lecture:

- Inverse reinforcement learning
- Feature matching
- Max margin planning
- Maximum entropy IRL
- Adversarial Imitation learning

Demonstration Augmentation: ALVINN 1989

Road follower

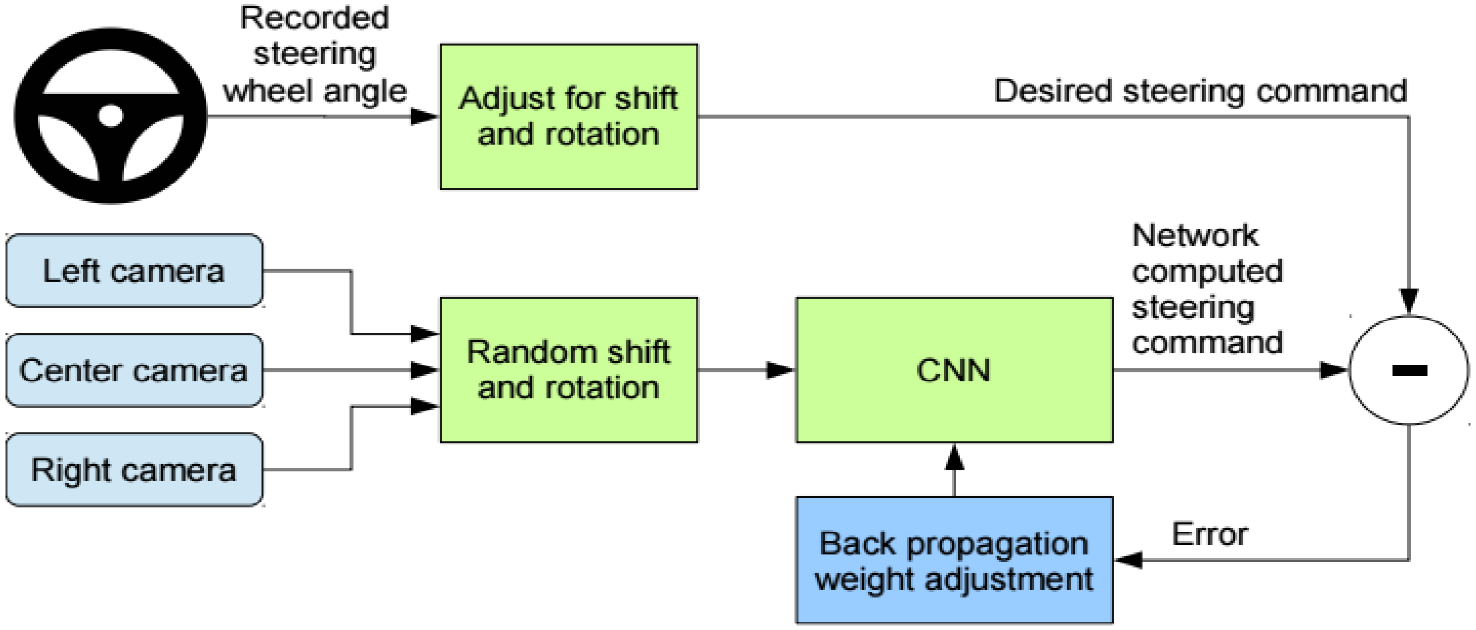


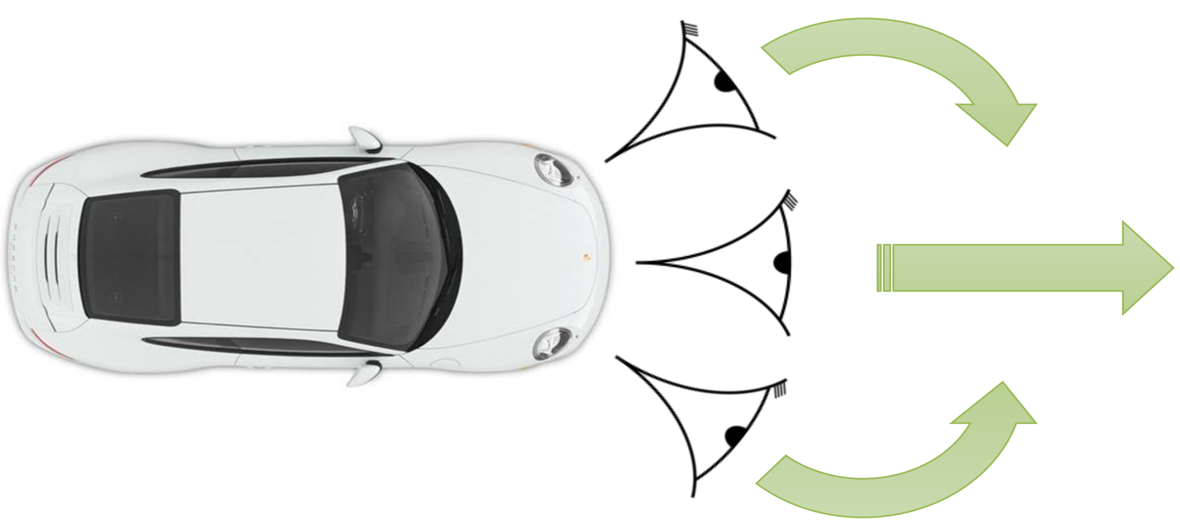


- Using graphics simulator for road images and corresponding steering angle ground-truth
- Online adaptation to human driver steering angle control
- 3 layers, fully connected layers, very low resolution input from camera and lidar..

"In addition, the network must not solely be shown examples of accurate driving, but also how to recover (i.e. return to the road center) once a mistake has been made. Partial initial training on a variety of simulated road images should help eliminate these difficulties and facilitate better performance. " ALVINN: An autonomous Land vehicle in a neural Network, Pomerleau 1989

Demonstration Augmentation: NVIDIA 2016



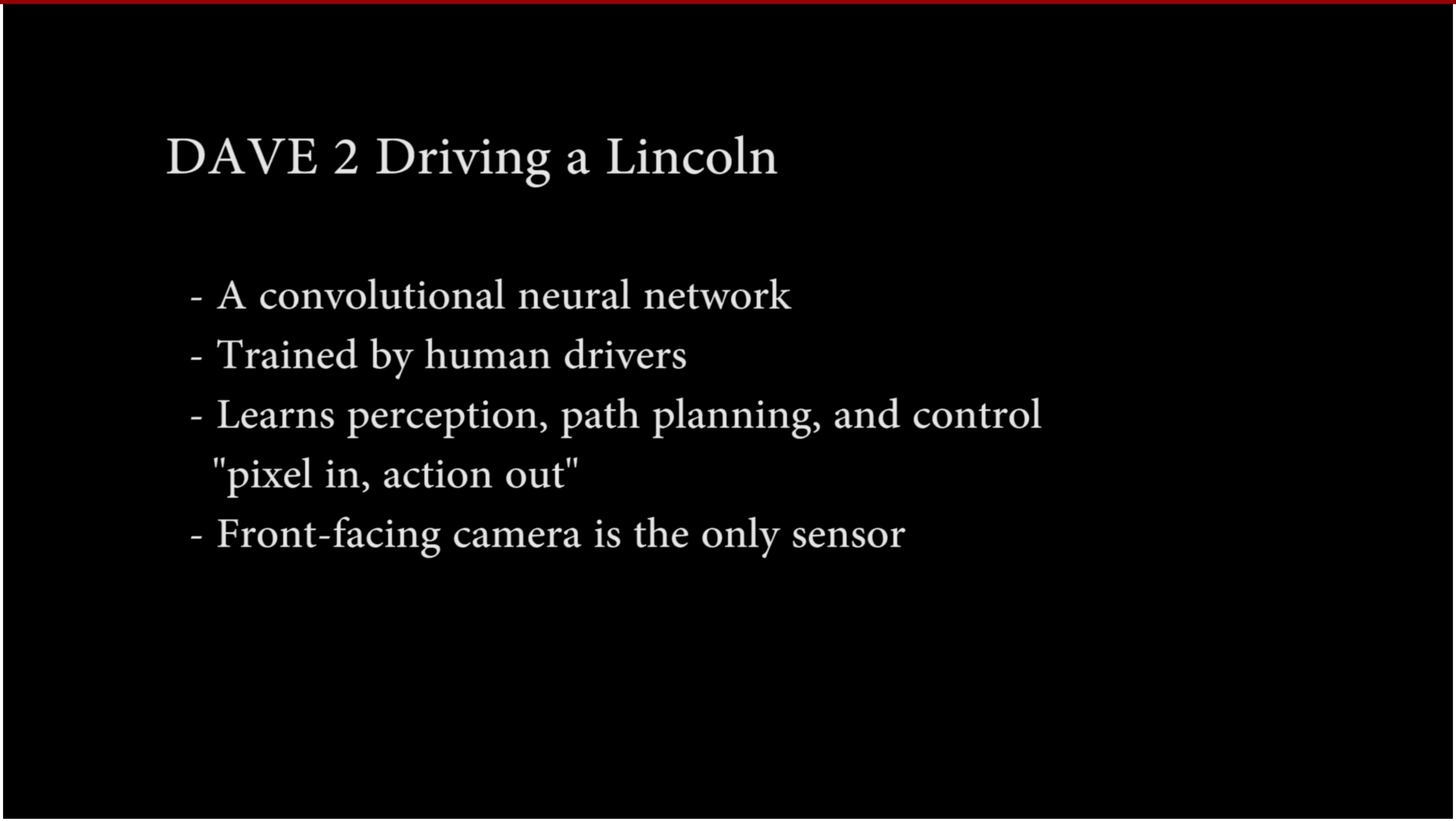


Additional, left and right cameras with automatic grant-truth labels to recover from mistakes

"*DAVE-2 was inspired by the pioneering work of Pomerleau [6] who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system*. Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. …",

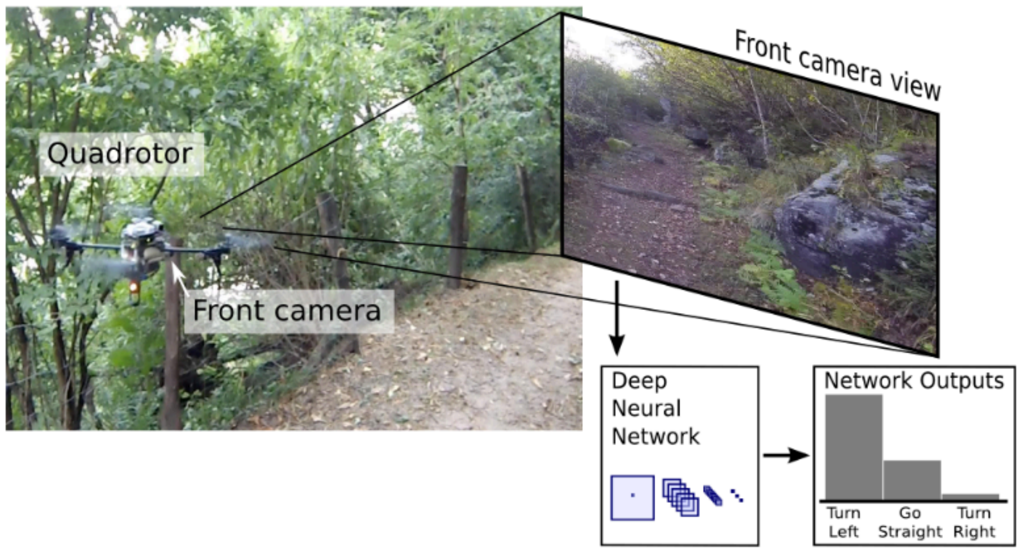
End to End Learning for Self-Driving Cars , Bojarski et al. 2016

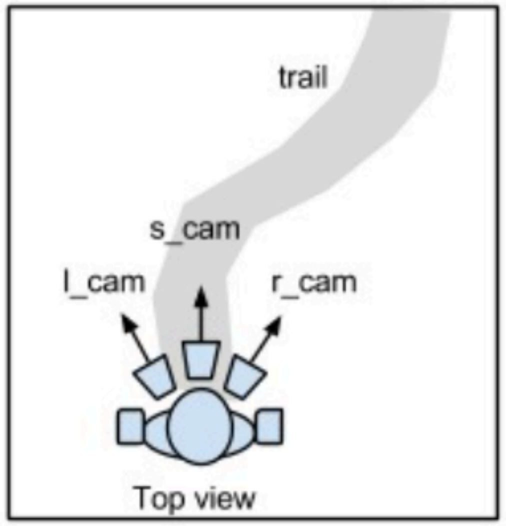
Data Augmentation (2): NVIDIA 2016

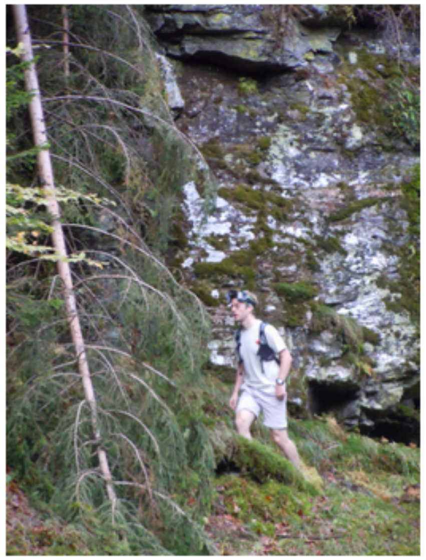


"*DAVE-2 was inspired by the pioneering work of Pomerleau [6] who in 1989 built the Autonomous Land Vehicle in a Neural Network (ALVINN) system*. Training with data from only the human driver is not sufficient. The network must learn how to recover from mistakes. …", End to End Learning for Self-Driving Cars , Bojarski et al. 2016

Data Augmentation (3): Trails 2015







A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots Giusti et al.

Data Augmentation (3): Trails 2015

A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots Giusti et al.

Outline

This lecture

- Behavior Cloning: Imitation learning as supervised learning
- Compounding errors
- Demonstration augmentation techniques
- DAGGER
- Structured prediction as Decision Making (learning to search)
- Imitating MCTS

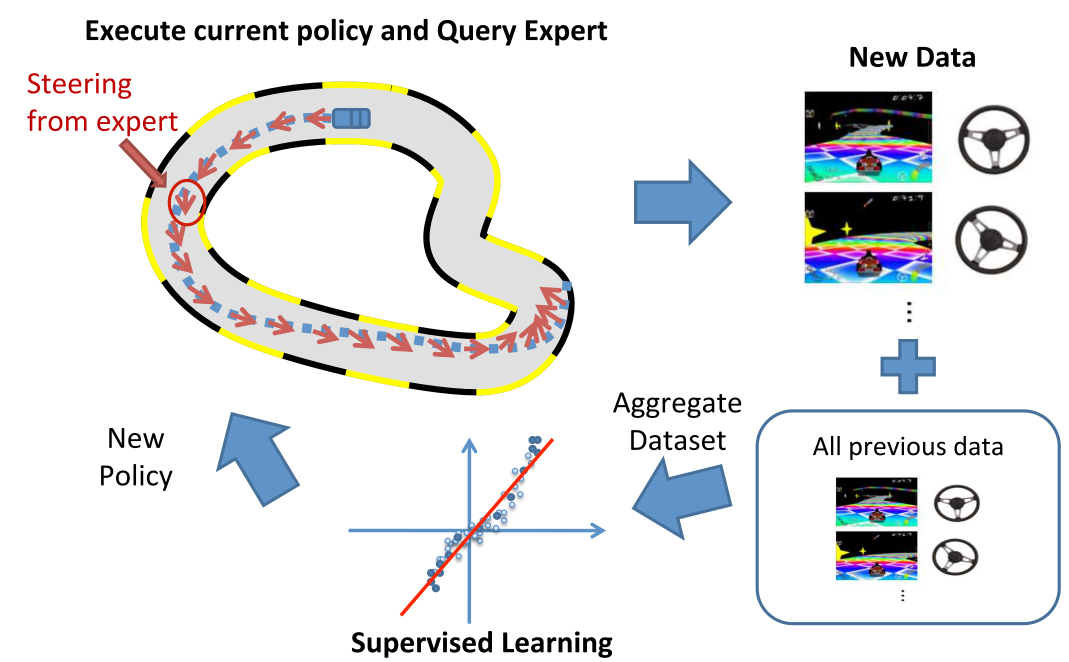
Next lecture:

- Inverse reinforcement learning
- Feature matching
- Max margin planning
- Maximum entropy IRL
- Adversarial Imitation learning

DAGGER (in simulation)

Dataset AGGregation: bring learner's and expert's trajectory distributions closer by labelling additional data points resulting from applying the current policy

1. train $\pi_\theta(u_t|o_t)$ from human data $\mathcal{D}_{\pi^*} = \{o_1, u_1, ..., o_N, u_N\}$
2. run $\pi_\theta(u_t|o_t)$ to get dataset $\mathcal{D}_\pi = \{o_1, ..., o_M\}$
3. Ask human to label $\mathcal{D}_\pi$ with actions u_t
4. Aggregate: $\mathcal{D}_{\pi^*} \leftarrow \mathcal{D}_{\pi^*} \cup \mathcal{D}_\pi$
5. GOTO step 1.



Problems:

- execute an unsafe/partially trained policy
- repeatedly query the expert

A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning, Ross et al. 2011

DAGGER (in a real platform)

Application on drones: given RGB from the drone camera predict steering angles

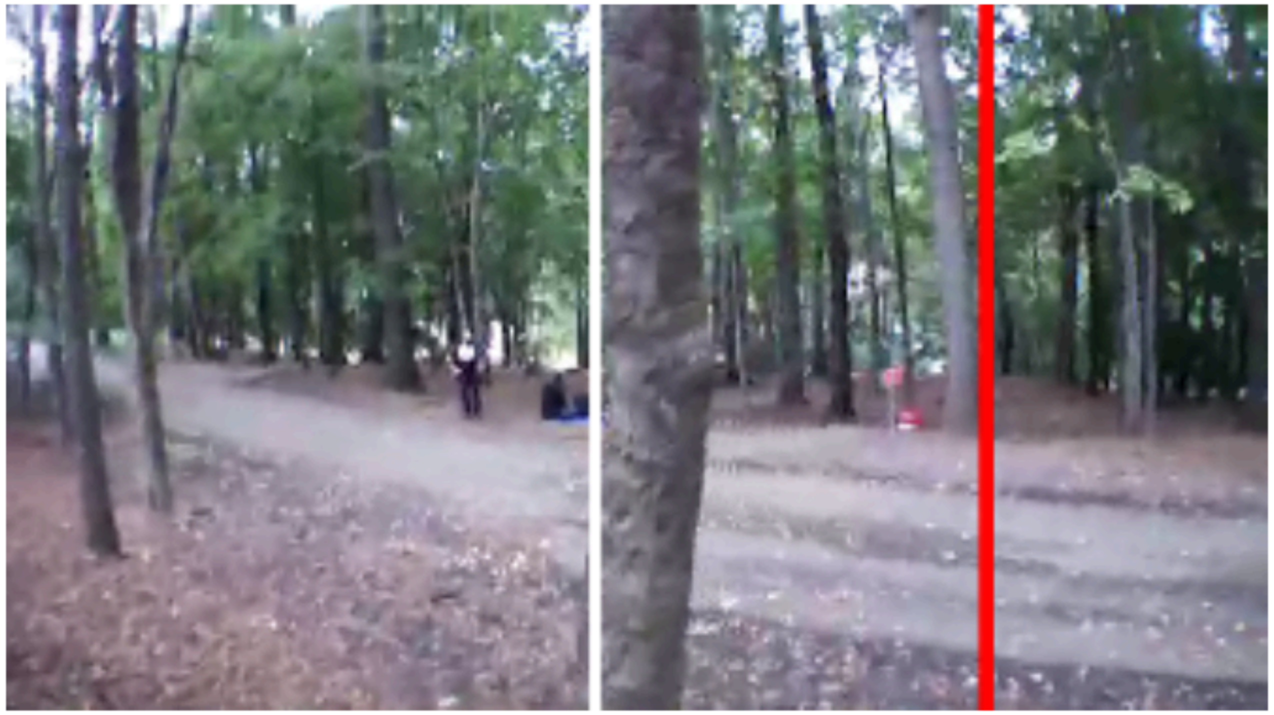
Learning monocular reactive UAV control in cluttered natural environments, Ross et al. 2013

DAGGER (in a real platform)

Application on drones : given RGB from the drone camera predict steering angle

Caveats:

1. Interaction with the expert is hard: Is hard for the expert to provide the right magnitude for the turn without feedback of his own actions! Solution: provide him his visual feedback



Learning monocular reactive uav control in cluttered natural environments, Ross et al. 2013

DAGGER (in a real platform)

Caveats:

1. Is hard for the expert to provide the right magnitude for the turn without feedback of his own actions! Solution: provide him his visual feedback
2. The expert's reaction time to the drone's behavior is large, this causes imperfect actions to be commanded. Solution: play-back in slow motion offline and record their actions.
3. Executing an imperfect policy causes accidents, crashes into obstacles. Solution: safety measures which make again the data distribution matching imperfect between train and test, but good enough..

Learning monocular reactive uav control in cluttered natural environments, Ross et al. 2013

Outline

This lecture

- Behavior Cloning: Imitation learning as supervised learning
- Compounding errors
- Demonstration augmentation techniques
- DAGGER
- Structured prediction as Decision Making (learning to search)
- Imitating MCTS

Next lecture:

- Inverse reinforcement learning
- Feature matching
- Max margin planning
- Maximum entropy IRL
- Adversarial Imitation learning

Structured prediction

Structured prediction: a learner makes predictions over a set of interdependent output variables and observes a joint loss.

Example: part of speech tagging

```
x = the monster ate the sandwich
y = Dt  Nn      Vb  Dt  Nn
```

A structured prediction problem consists of an *input space* $\mathcal{X}$, an *output space* $\mathcal{Y}$*,* a fixed but unknown distribution $\mathcal{D}$ over $\mathcal{X} \times \mathcal{Y}$, and a non-negative *loss function* $l(y^*, \hat{y}) \rightarrow \mathbb{R}^{\geq 0}$ which measures the distance between the true $y*$ and predicted $\hat{y}$ outputs. The goal of structured learning is to use N samples $(x_i, y_i)_{i=1}^N$ to learn a mapping $f : \mathcal{X} \rightarrow \mathcal{Y}$ that minimizes the expected structured loss under $\mathcal{D}$.

Structured prediction

Sequence labelling:

Part of speech tagging

```
x = the monster ate the sandwich
y = Dt  Nn      Vb  Dt  Nn
```

Structured prediction

Sequence labelling:

Part of speech tagging

NER (Name Entity Recognition)

```
x = Yesterday I traveled to Lille
y =       -    PER    -       -  LOC
```

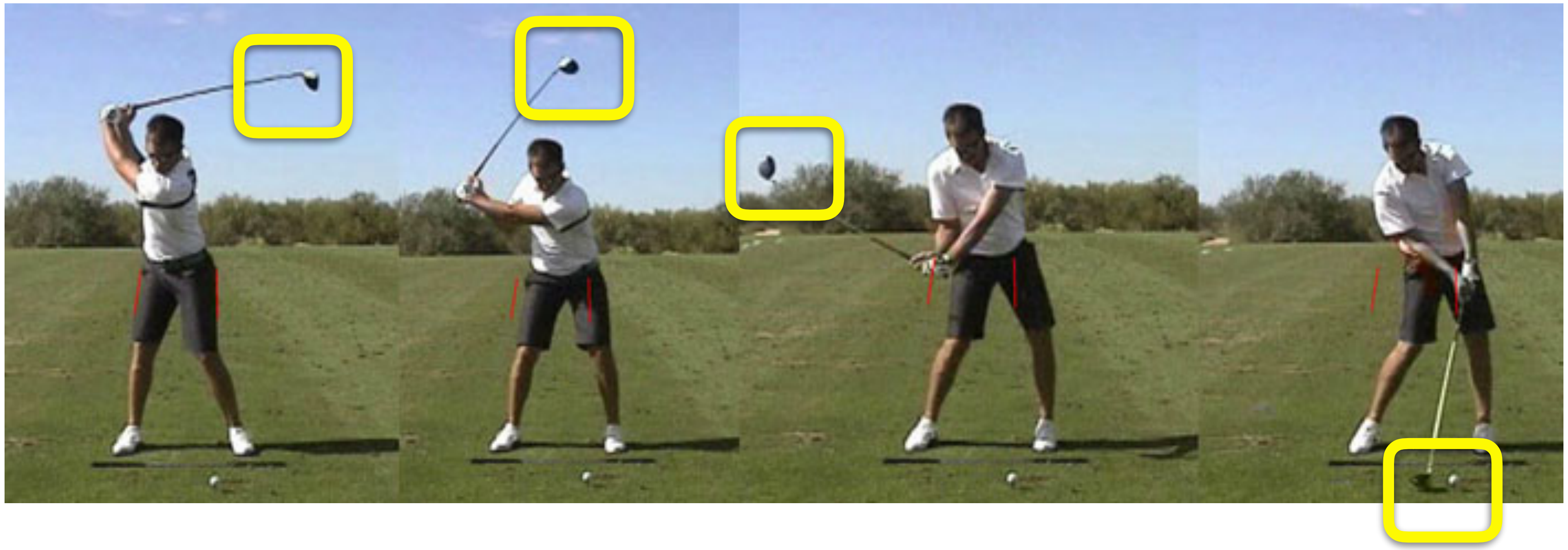
Structured prediction

Sequence labelling:

Part of speech tagging

NER

Attentive Tracking



Structured prediction

Sequence labelling:

- Part of speech tagging
- NER
- Tracking

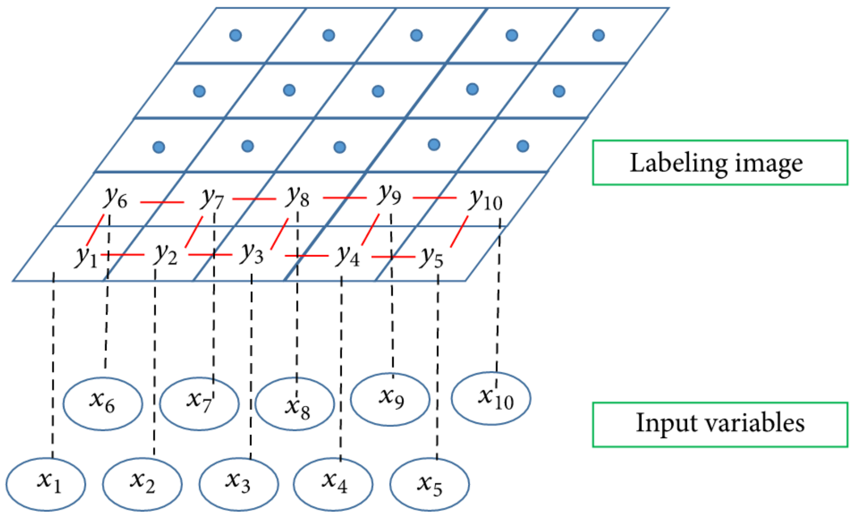
Sequence generation:

- Captioning
- Machine translation

Optimizing Graphical Models for Structured prediction

Graph labelling

- Encode output labels as a MRF
- Learn parameters of that model to:
  - maximize p(true labels I input)
  - minimize loss(true labels, predicted labels)

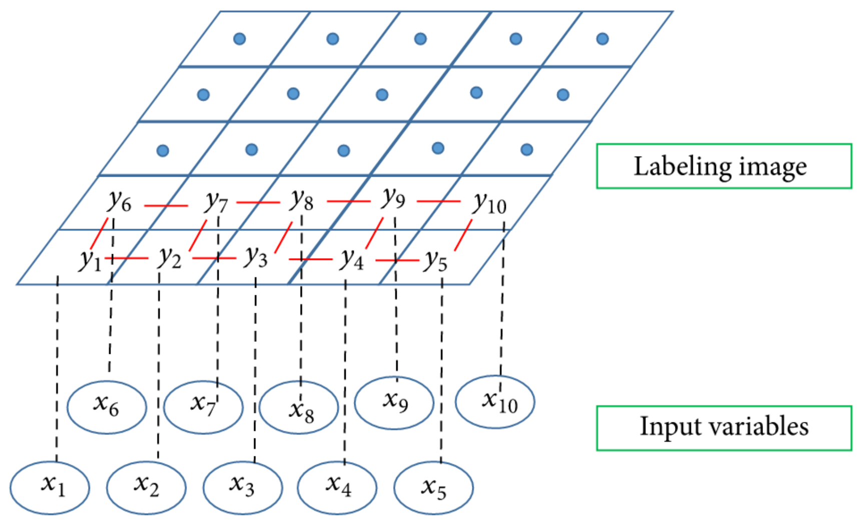


Let $G = (V, E)$ be a graph such that

$\boldsymbol{Y} = (\boldsymbol{Y}_v)_{v \in V}$, so that $\boldsymbol{Y}$ is indexed by the vertices of G. Then $(\boldsymbol{X}, \boldsymbol{Y})$ is a conditional random field when the random variables $\boldsymbol{Y}_v$, conditioned on $\boldsymbol{X}$, obey the Markov property with respect to the graph: $p(\boldsymbol{Y}_v | \boldsymbol{X}, \boldsymbol{Y}_w, w \neq v) = p(\boldsymbol{Y}_v | \boldsymbol{X}, \boldsymbol{Y}_w, w \sim v)$, where $w \sim v$ means that w and v are neighbors in G.

Optimizing Graphical Models for Structured prediction

- Encode output labels as a MRF
- Learn parameters of that model to:
    - maximize p(true labels | input)
    - minimize loss(true labels, predicted labels)



- Assumed Independence assumptions may not hold
- Computationally intractable with too many “edges” or non-decomposable loss functions (that involve many ys)

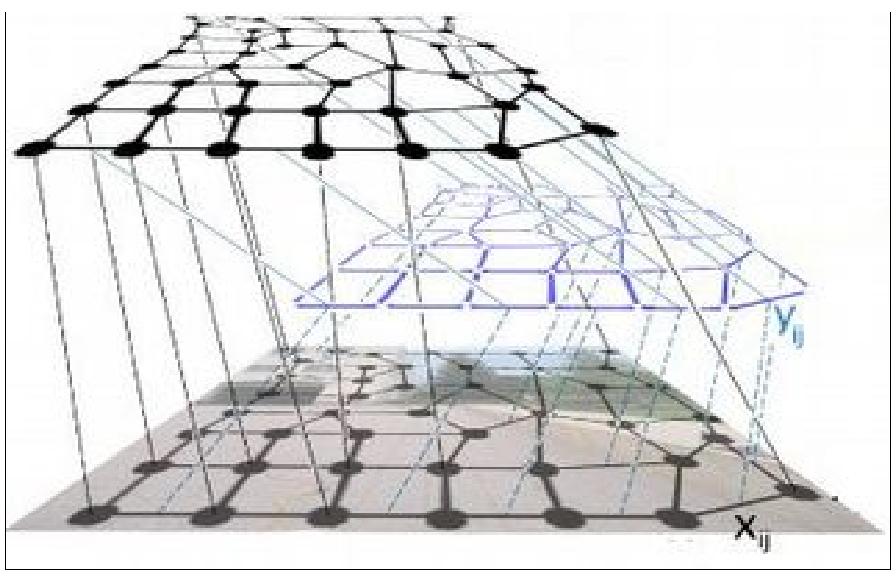
Instead: Decomposition of label

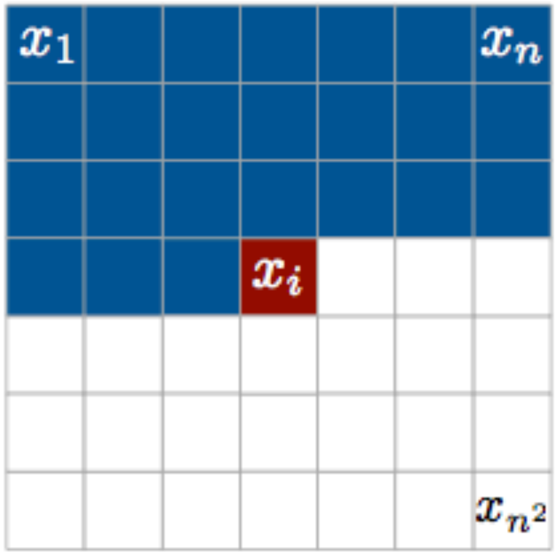
Sequence generation/labelling:

We can define an ordering and generate labels one at a time, where each output generated depends on all previous ones. E.g., sequential data admits the natural sequential ordering.

Image generation/labelling:

Here again we can define an ordering:

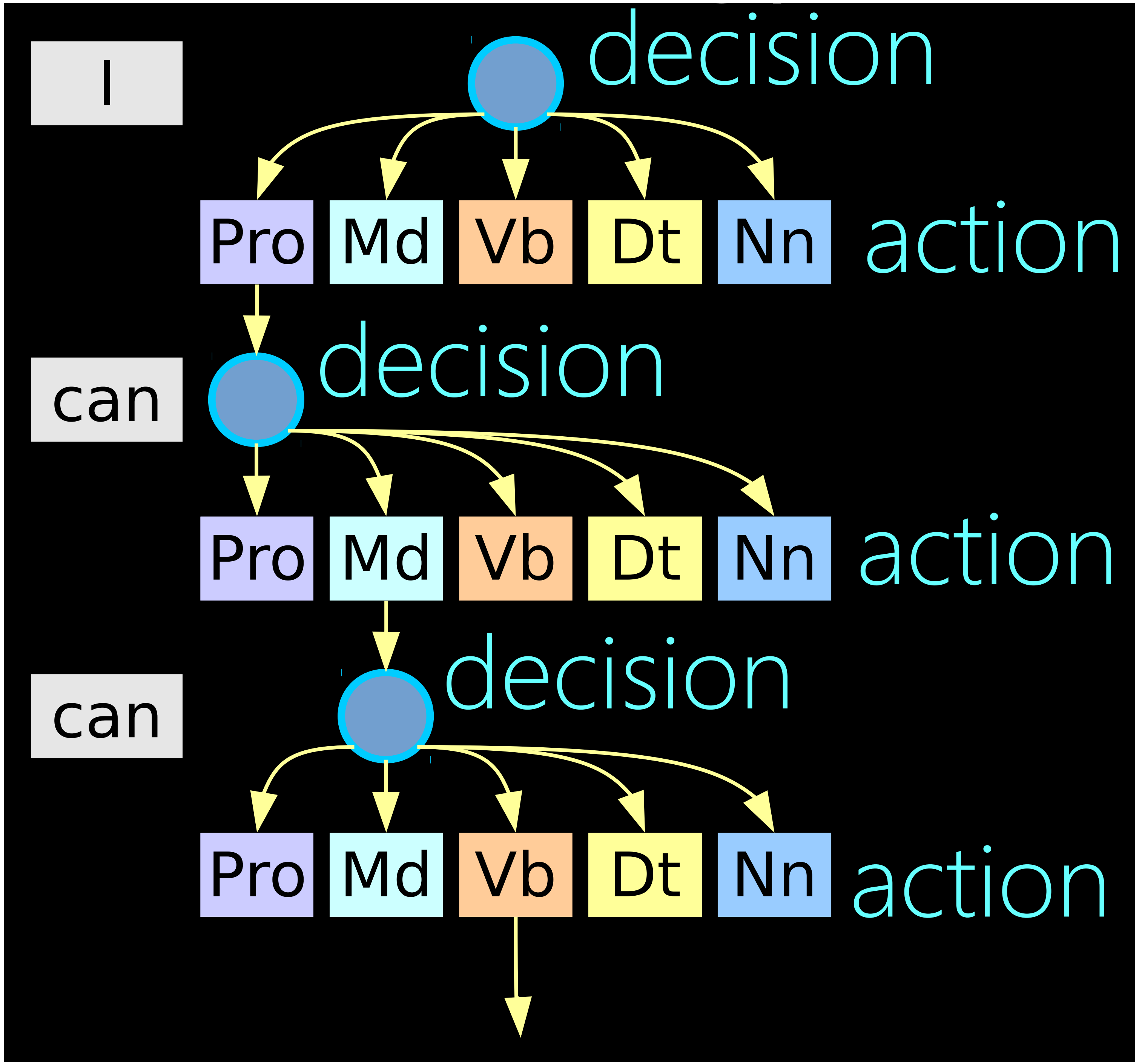




Pixel Recurrent Neural Networks, van den Oord et al

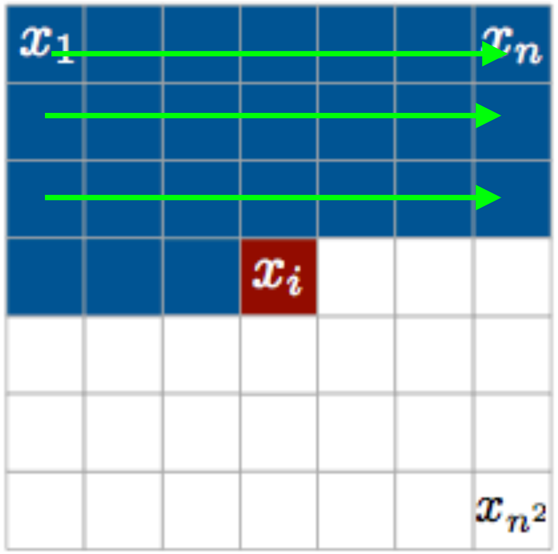
Structured prediction as sequential decision making

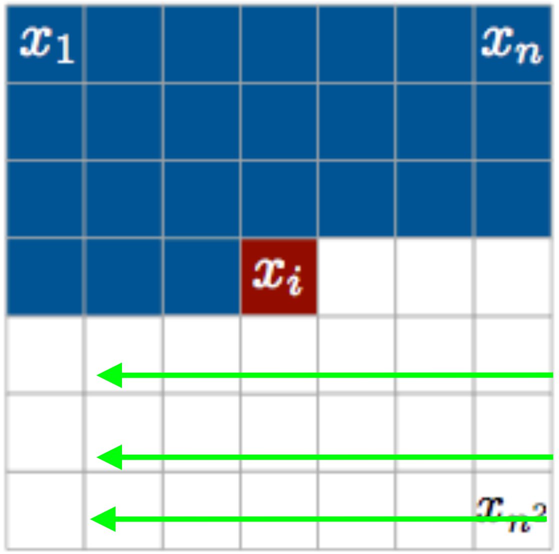
When y decomposes in an ordered manner, a sequential decision making process emerges



Structured prediction as sequential decision making

When y decomposes in an ordered manner, a sequential decision making process emerges





Structured prediction as sequential decision making

```
x = the monster ate the sandwich
y = Dt  Nn      Vb  Dt  Nn
```

- Example: Sequence labelling
- State: captures input sequence x and whatever labels (here part of speech tags) we have produced so far
- Actions: Next label to output
- Policy: a mapping of the input x and labels generated so far to the next label
- Reward: agreement of the predicted \hat{y} with ground-truth y*: $\ell(e) = \ell(\boldsymbol{y}^*, \boldsymbol{y}_e)$

Structured prediction as sequential decision making

Caption: A blue monster is eating a cookie

- Example: Image captioning
- State: captures the image and whatever words we have produced so far
- Actions: Next word to output
- Policy: a map of the state to the next word
- Reward: agreement of the predicted \hat{y} with ground-truth y*: $\ell(e) = \ell(\boldsymbol{y}^*, \boldsymbol{y}_e)$
- The loss here is not decomposable.

Structured prediction as sequential decision making

Sequence labelling:

Parsing

NER

Tracking

Sequence generation:

Captioning

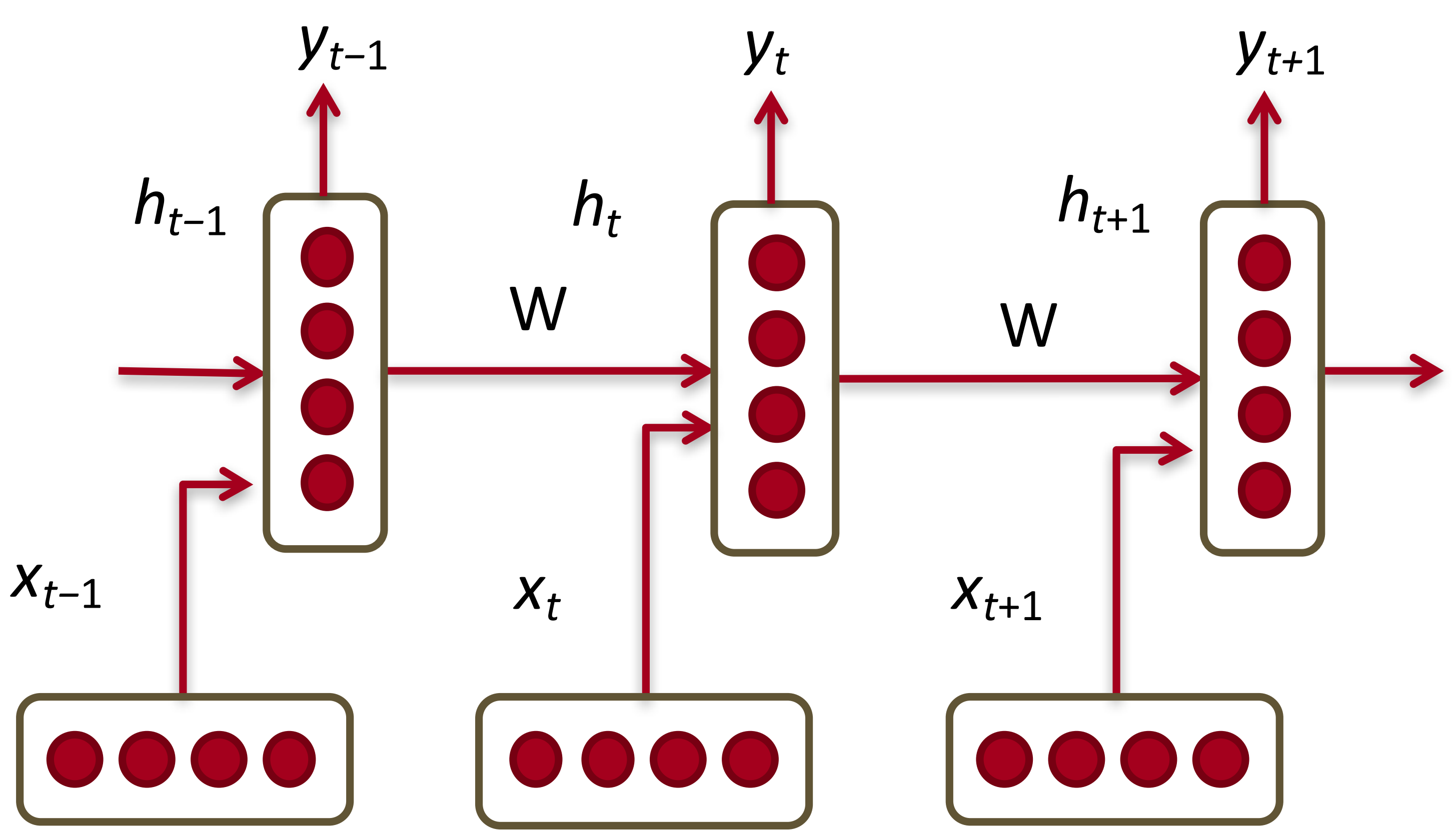
Machine translation

Etc..

What function approximation shall we use for our state representations in case of sequence/image labelling/generation?

Recurrent Neural Networks

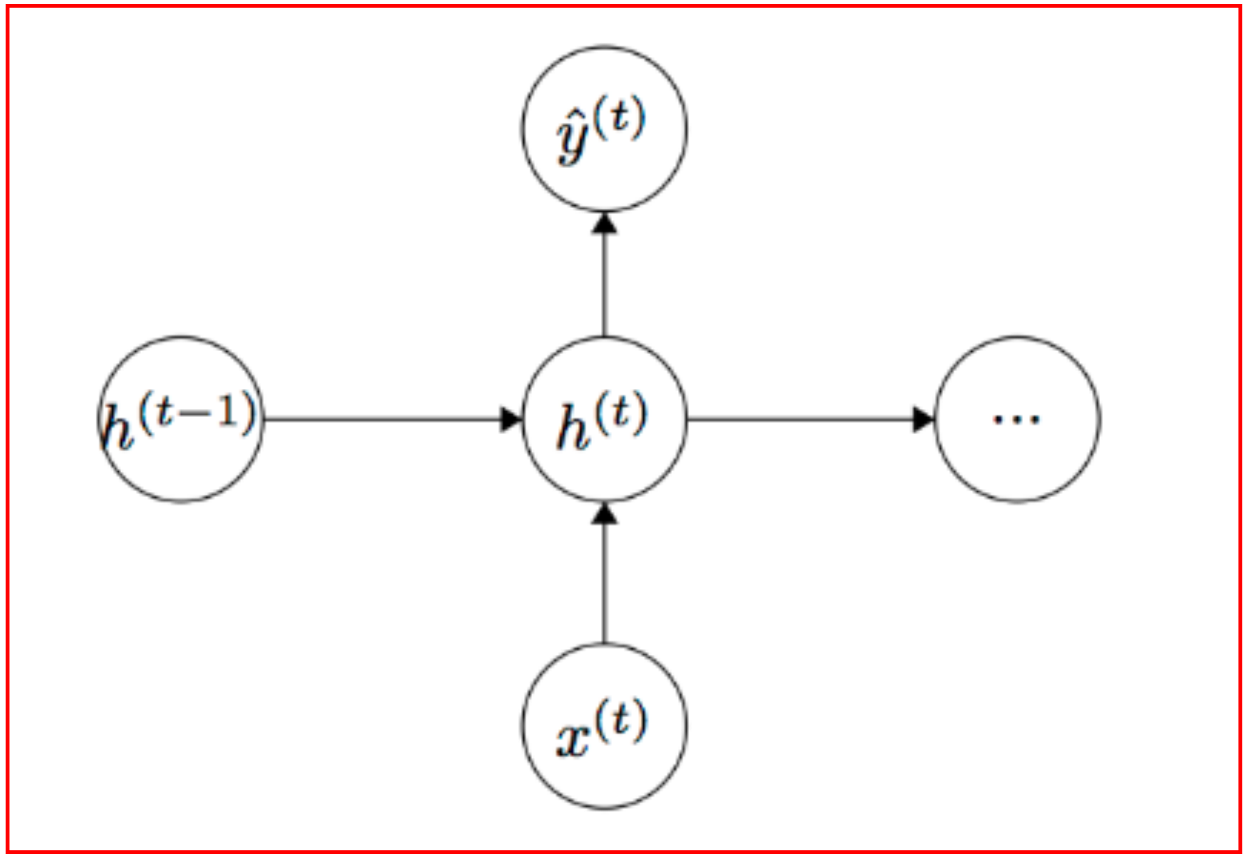
- RNNs tie the weights at each time step
- Condition the neural network on all previous inputs
- In principle, any interdependencies can be modeled between inputs and outputs, as well as between output labels.
- In practice, limitations from SGD training, capacity, initialization etc.



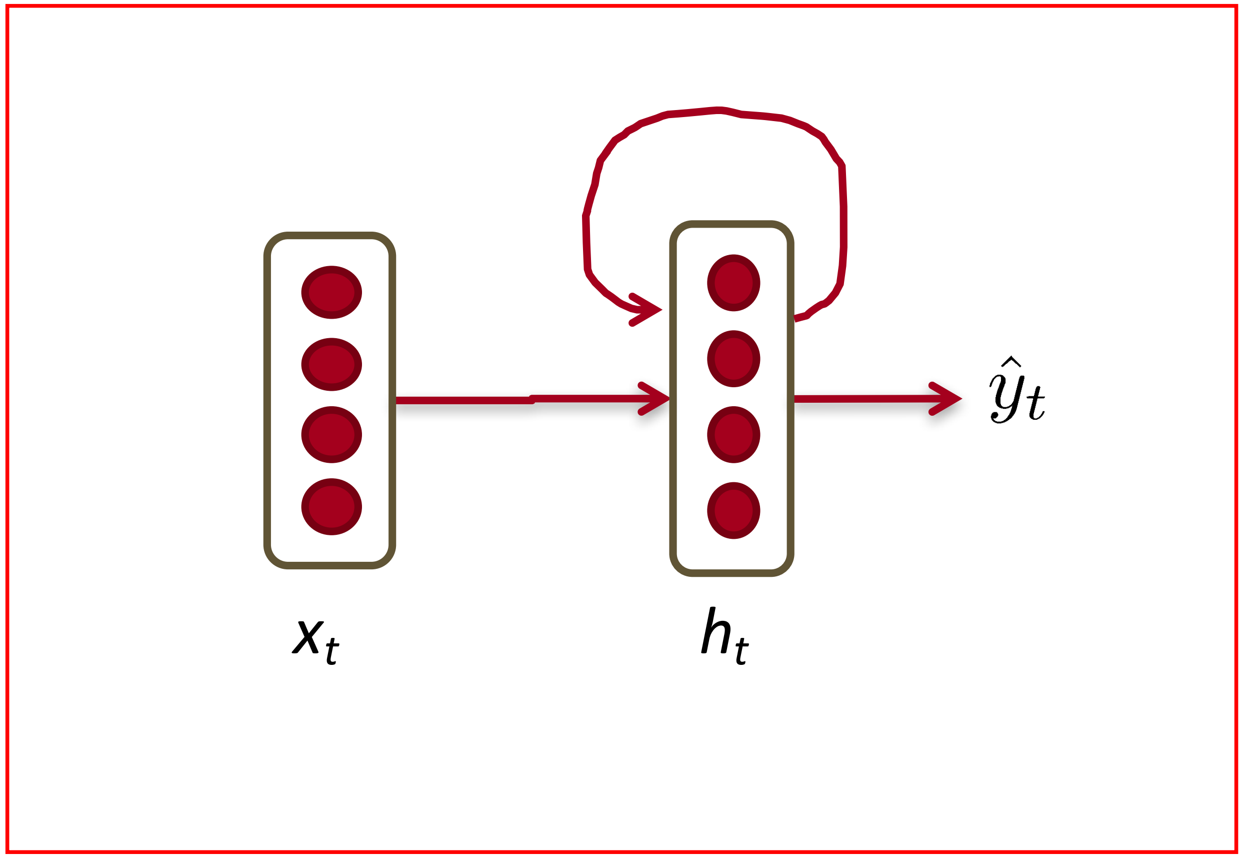
Recurrent Neural Network (single hidden layer)

- Given list of **vectors**: $x_1, \ldots, x_{t-1}, x_t, x_{t+1}, \ldots, x_T$
- At a single time step:

$$h_t = \sigma\left(W^{(hh)} h_{t-1} + W^{(hx)} x_{[t]}\right)$$
$$\hat{y}_t = \mathrm{softmax}\left(W^{(S)} h_t\right)$$

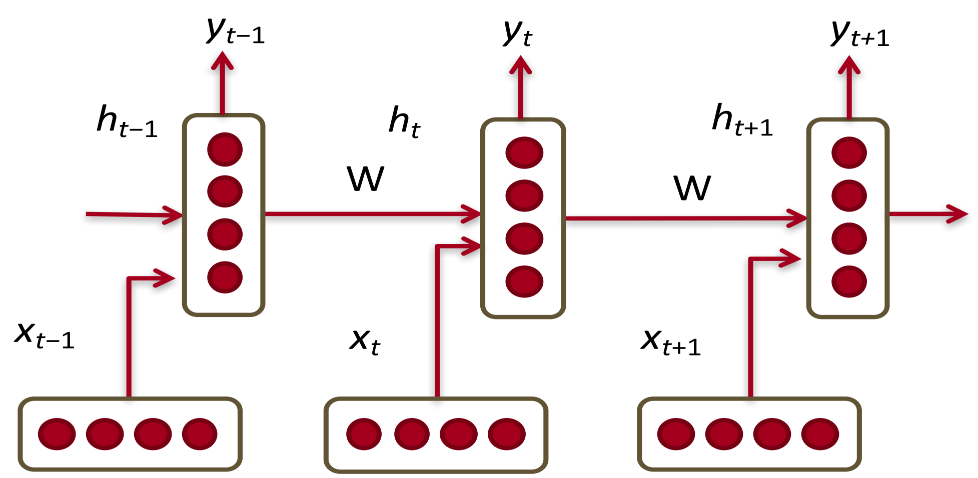


<->

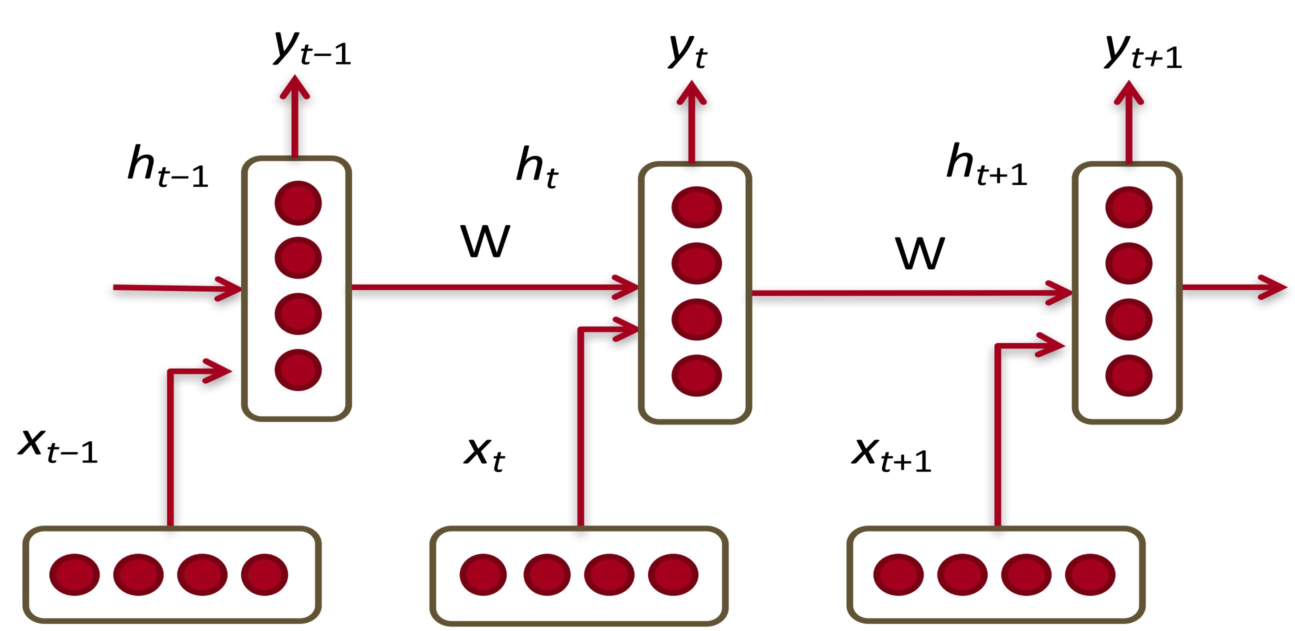


Recurrent Neural Networks

For sequence labelling problems, actions of the labelling policies are y_t, e.g., part of speech tags



For sequence generation, actions of the labelling policies are $y_t = x_{t+1}$, e.g., word in answer generation $\hat{P}(x_{t+1} = v_j | x_t, ..., x_1) = \hat{y}_{t,j}$



Recurrent Neural Networks

The network is typically trained to maximize the log-likelihood of the output sequences given the input sequences of a training set $\mathcal{D} = \{(x^{(i)}, y^{(i)})\}$:

$$\theta^* = \arg\max_\theta \log \sum_{(x^{(i)}, y^{(i)}) \in \mathcal{D}} P_\theta(y^{(i)}, x^{(i)})$$

If the likelihood of an example decomposes over individual time steps:

$$\log P_\theta(y|x) = \sum_t \log P_\theta(y_t|h_t)$$

Else loss is computed at the end of the sequence and is back propagated through time.

A learned policy is the inference function of the model:

$$\hat{\theta}(h_t) = \arg\max_y P(y_t = y|h_t; \theta)$$

The reference policy is the policy that always outputs the true labels:

$$\theta^*(h_t) = y_t$$

Recurrent Neural Networks

The regular training procedure of RNNs treat true labels y_t as actions while making forward passes. Hence, the learning agent follows trajectories generated by the reference policy rather than the learned policy. In other words, it learns:

$$\hat{\theta}^{sup} = \arg\min_{\theta} \mathbb{E}_{h \sim d_{\pi^*}} [l_\theta(h)]$$

However, our true goal is to learn a policy that minimizes error under its own induced state distribution:

$$\hat{\theta} = \arg\min_{\theta} \mathbb{E}_{h \sim d_\theta} [l_\theta(h)]$$

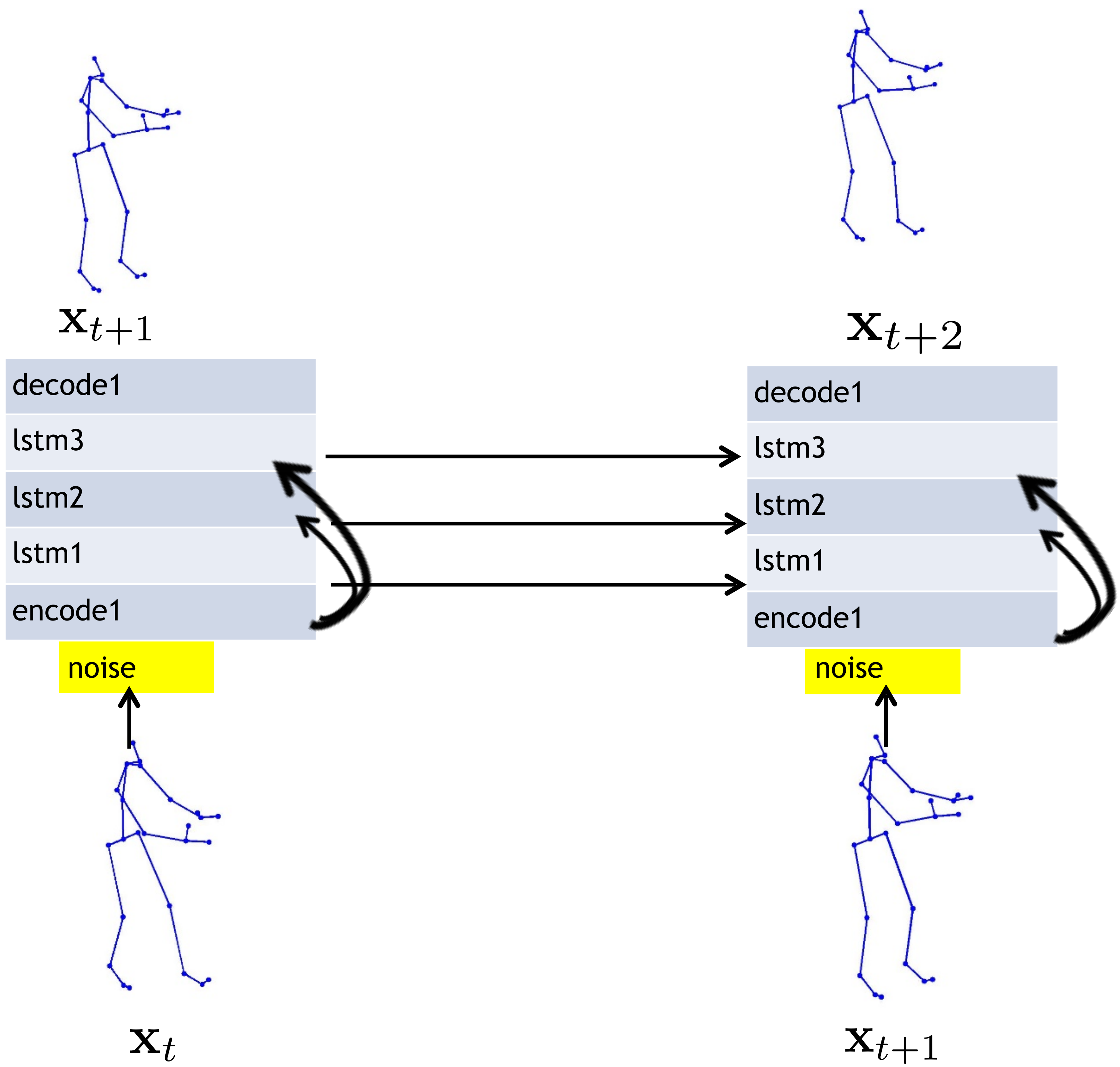
Imitation Learning with Recurrent Neural Networks, Nyuyen 2016

DAGGER for sequence labelling/generation with RNNs

function TRAIN(N, α)
  Intialize $\alpha = 1$.
  Initialize model parameters θ.
  for $i = 1..N$ **do**
    Set $\alpha = \alpha \cdot p$.
    Randomize a batch of labeled examples.
    for each example (x, y) in the batch **do**
      Initialize $h_0 = \Phi(X)$.
      Initialize $\mathcal{D} = \{(h_0, y_0)\}$.
      for $t = 1 \ldots |Y|$ **do**
        Uniformly randomize a floating-number $\beta \in [0, 1)$.
        if $\alpha < \beta$ **then**
          Use true label $\tilde{y}_{t-1} = y_{t-1}$
        else
          Use predicted label: $\tilde{y}_{t-1} = \arg\max_y P(y \mid h_{t-1}; \theta)$.
        end if
        Compute the next state: $h_t = f_\theta(h_{t-1}, \tilde{y}_{t-1})$.
        Add example: $\mathcal{D} = \mathcal{D} \cup \{(h_t, y_t)\}$.
      end for
    end for
    Online update θ by $\mathcal{D}$ (mini-batch back-propagation).
  end for
end function

Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks, Bengio(Samy) et al.
Imitation Learning with Recurrent Neural Networks, Nyuyen 2016

Data augmentation(4):Mocap generation

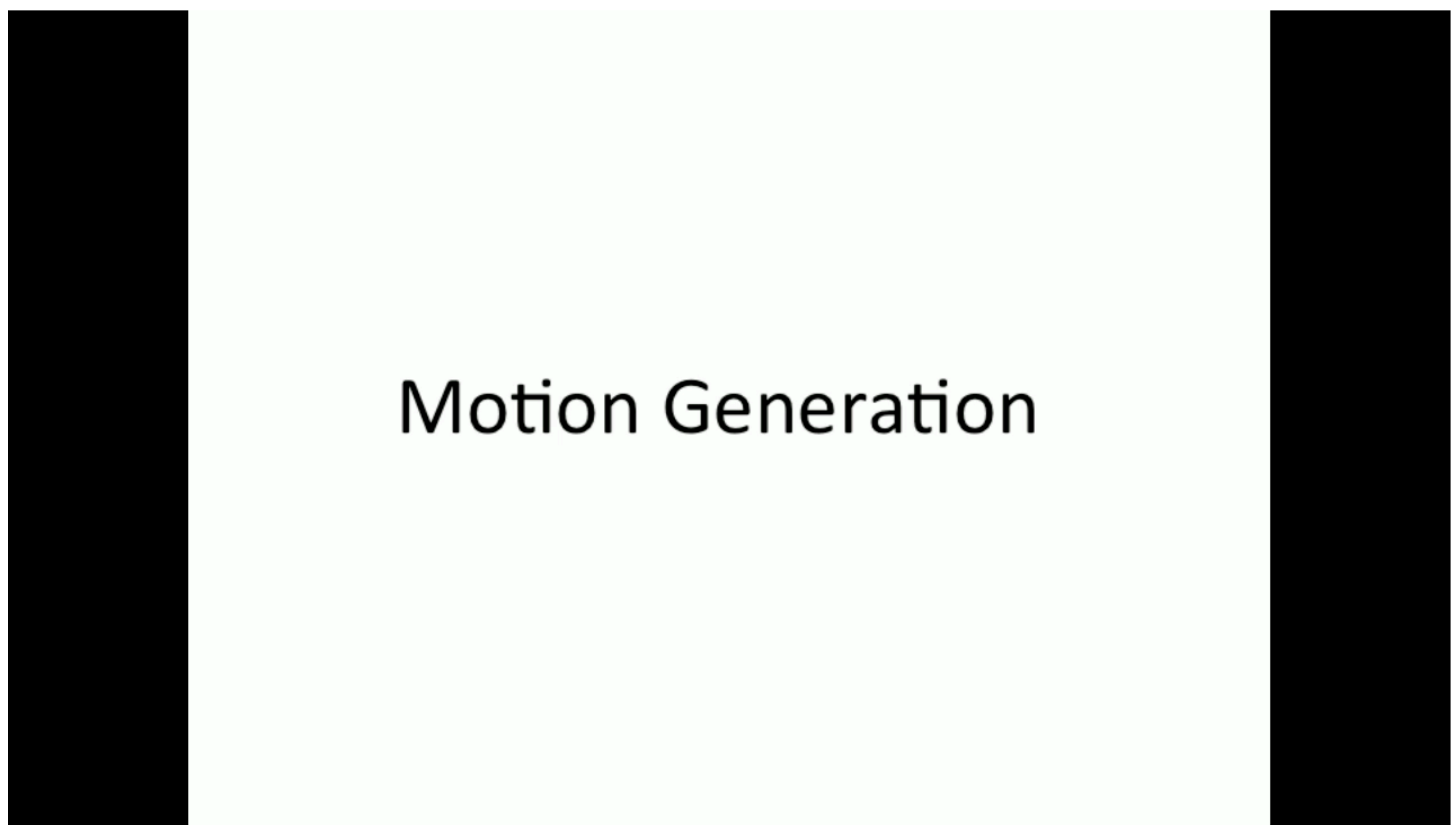


Graves et al.

Mocap generation

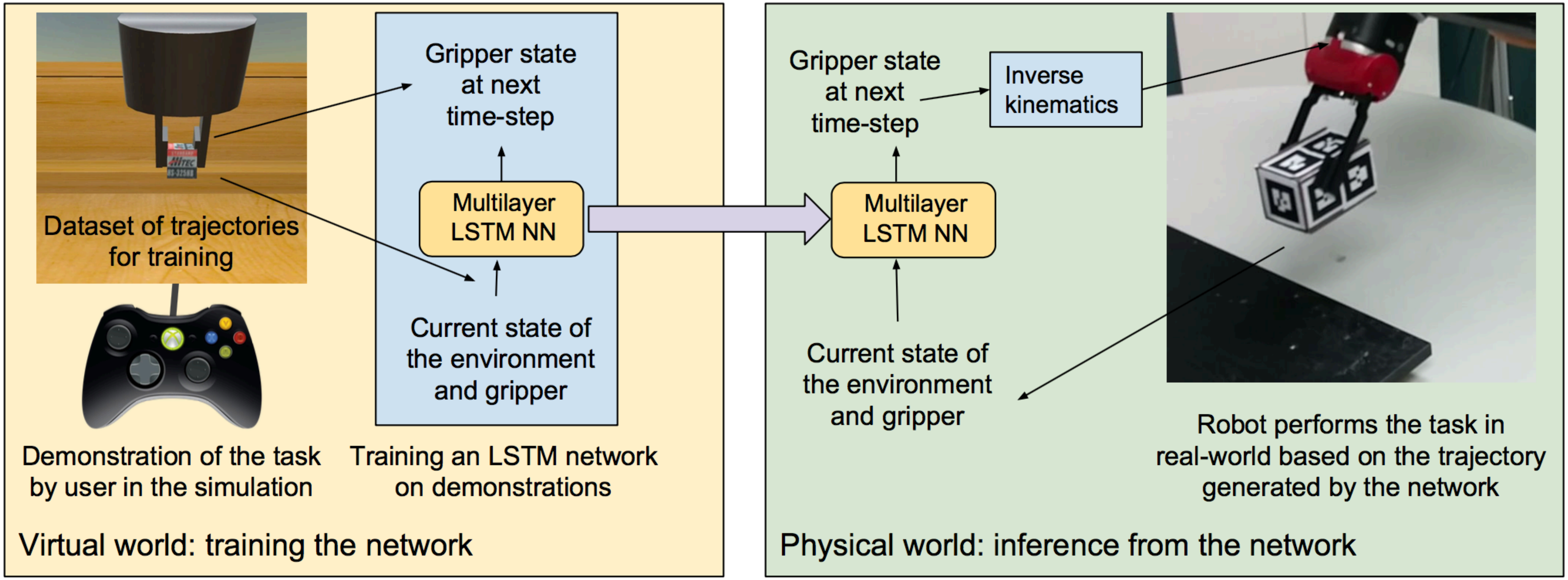
Right: no augmentation, using only ground-truth input

Left: augmentation



When you add noise to the input, despite the instantaneous error being larger, the long term error is lower.

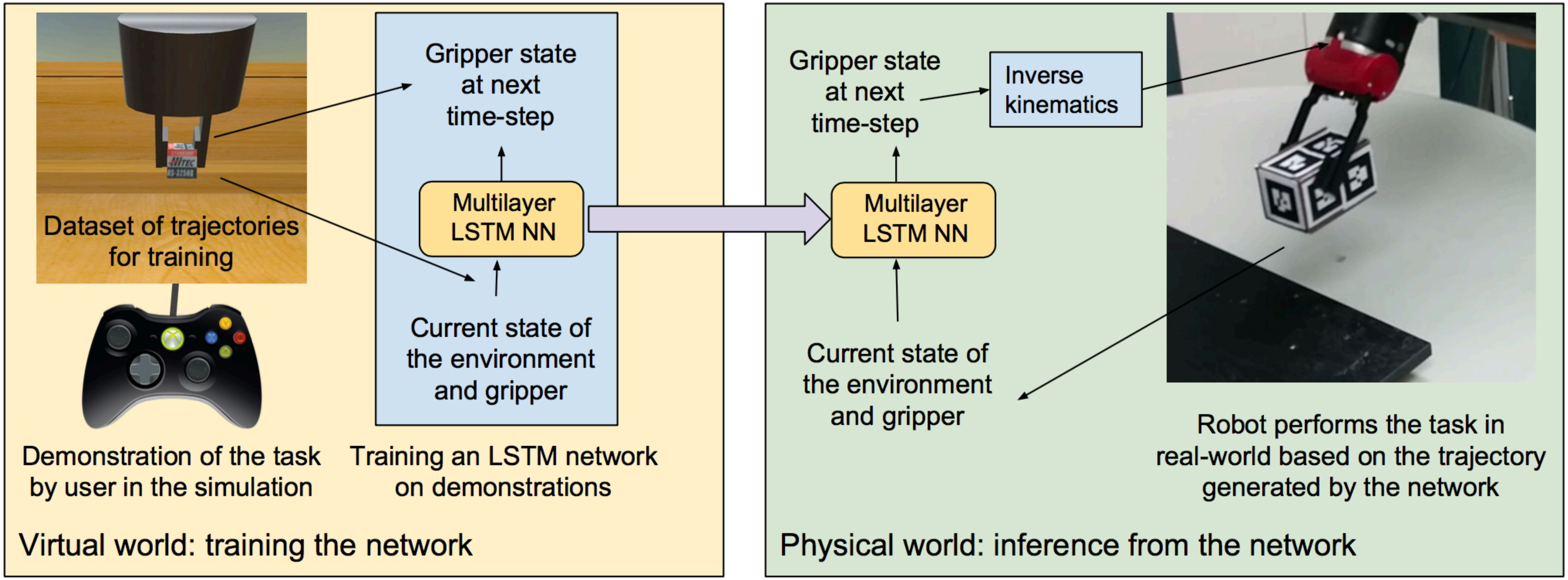
Demonstration Augmentation: Temporal subsampling



- Two tasks considered: pick and place, move to desired pose
- Input x: the poses of all the objects in the seen (rotations, translations) and the pose of the end effector
- Output y: the desired next pose of the end effector

Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016

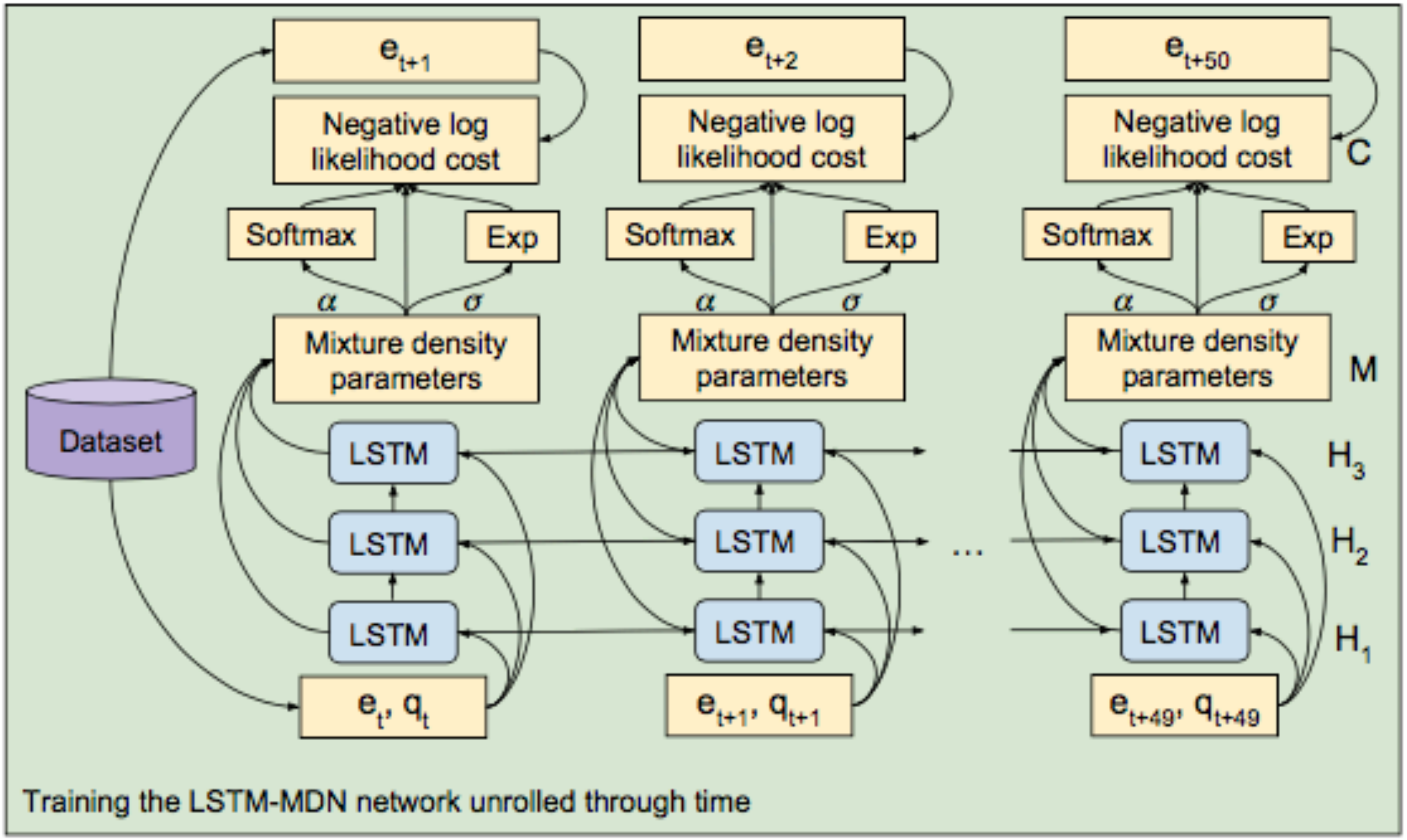
Demonstration Augmentation: Temporal subsampling



- Supervision: expert trajectories in the simulator
- Data augmentation: consider multiple trajectories by subsampling in time the expert ones, and by translating in space the end effector

Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016

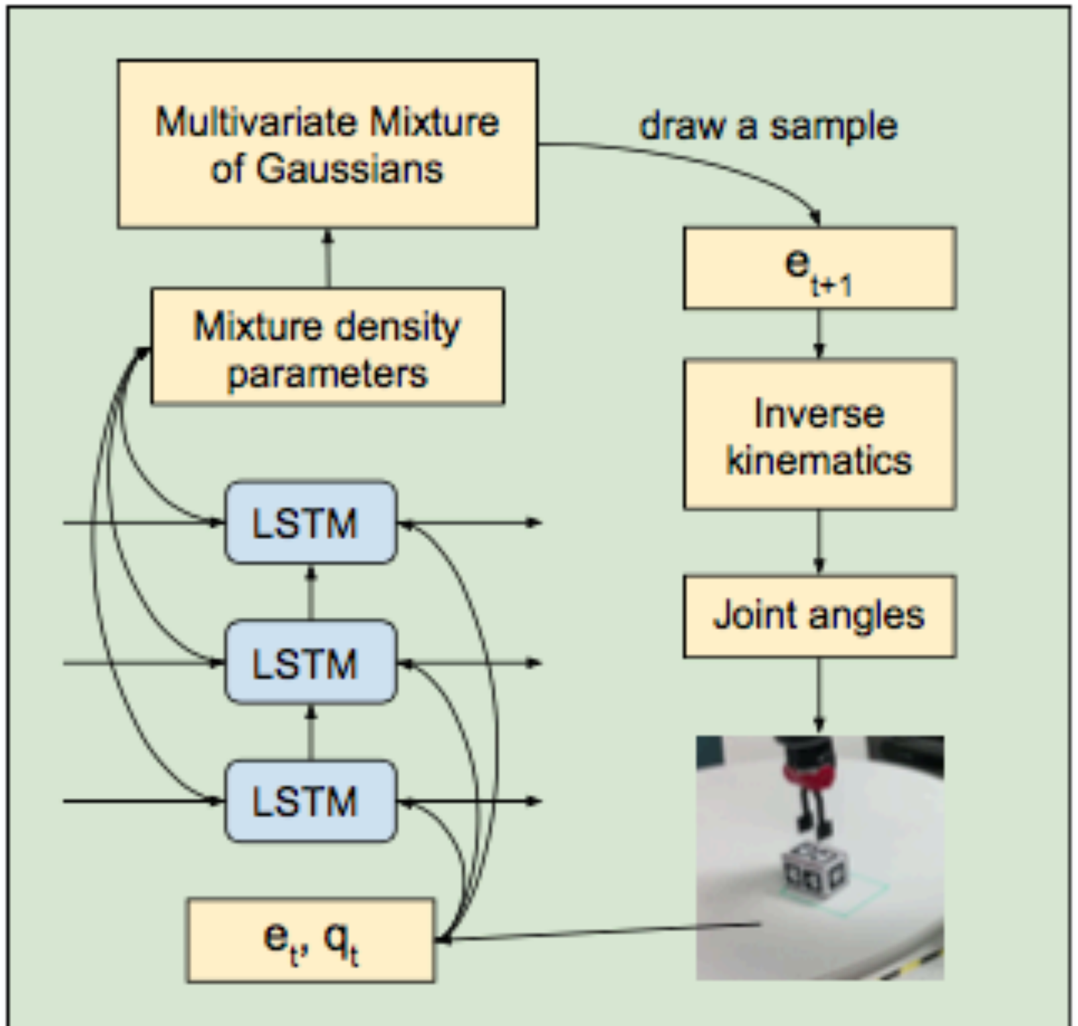
RNNs for Imitation(1)



- Multimodality of actions-> GMM loss!
- Predict mixture weights over a Gaussian Mixture Model at the output (alphas) and mean and variances for the mixture components.

Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016

Recurrent Neural Networks for Imitation(1)

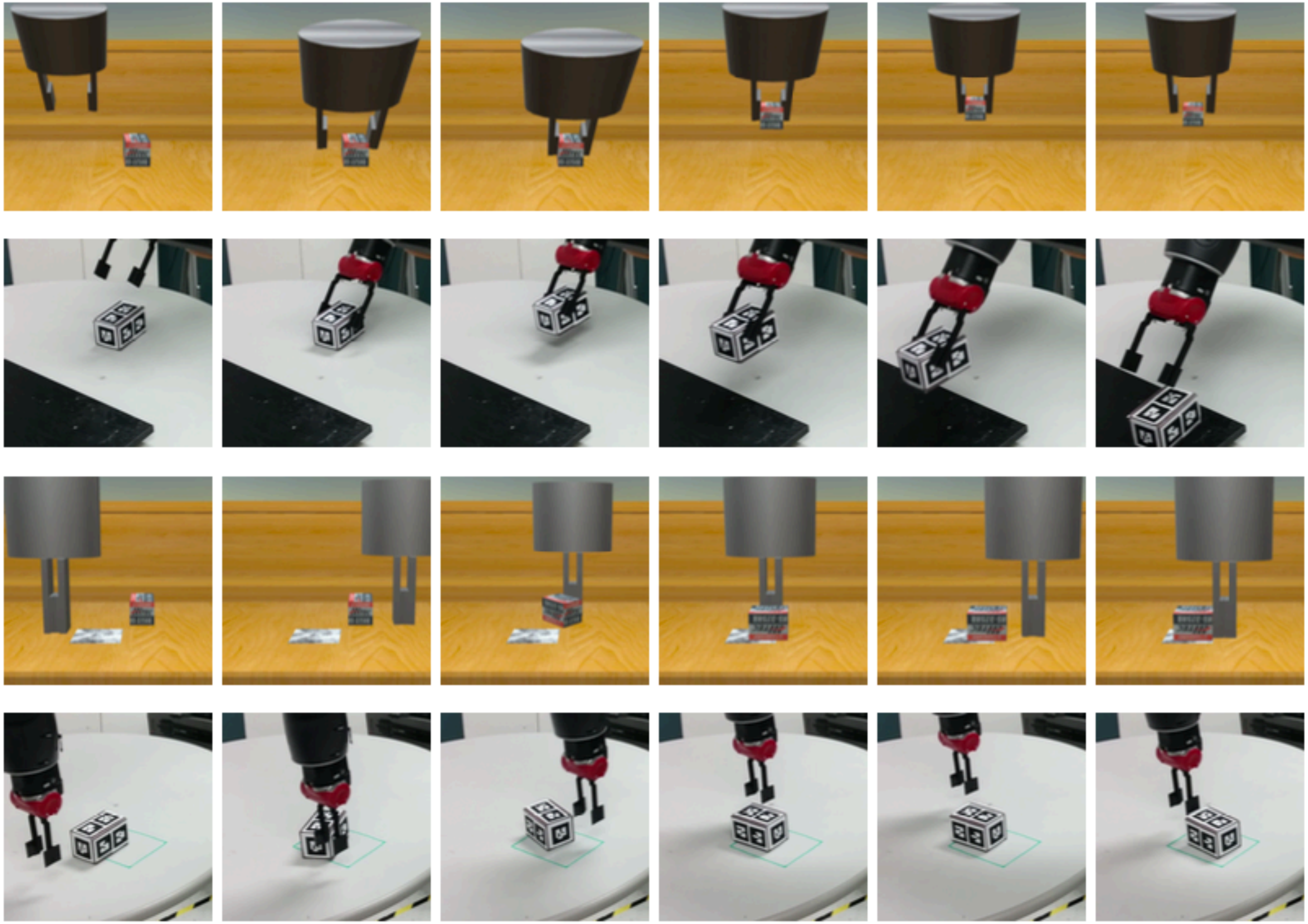


LSTM-MDN network performing the task in a closed loop

- Multimodality: predict mixture weights over a Gaussian Mixture Model at the output (alphas) and mean and variances for the mixture components. Minimize a GMM loss.

Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016

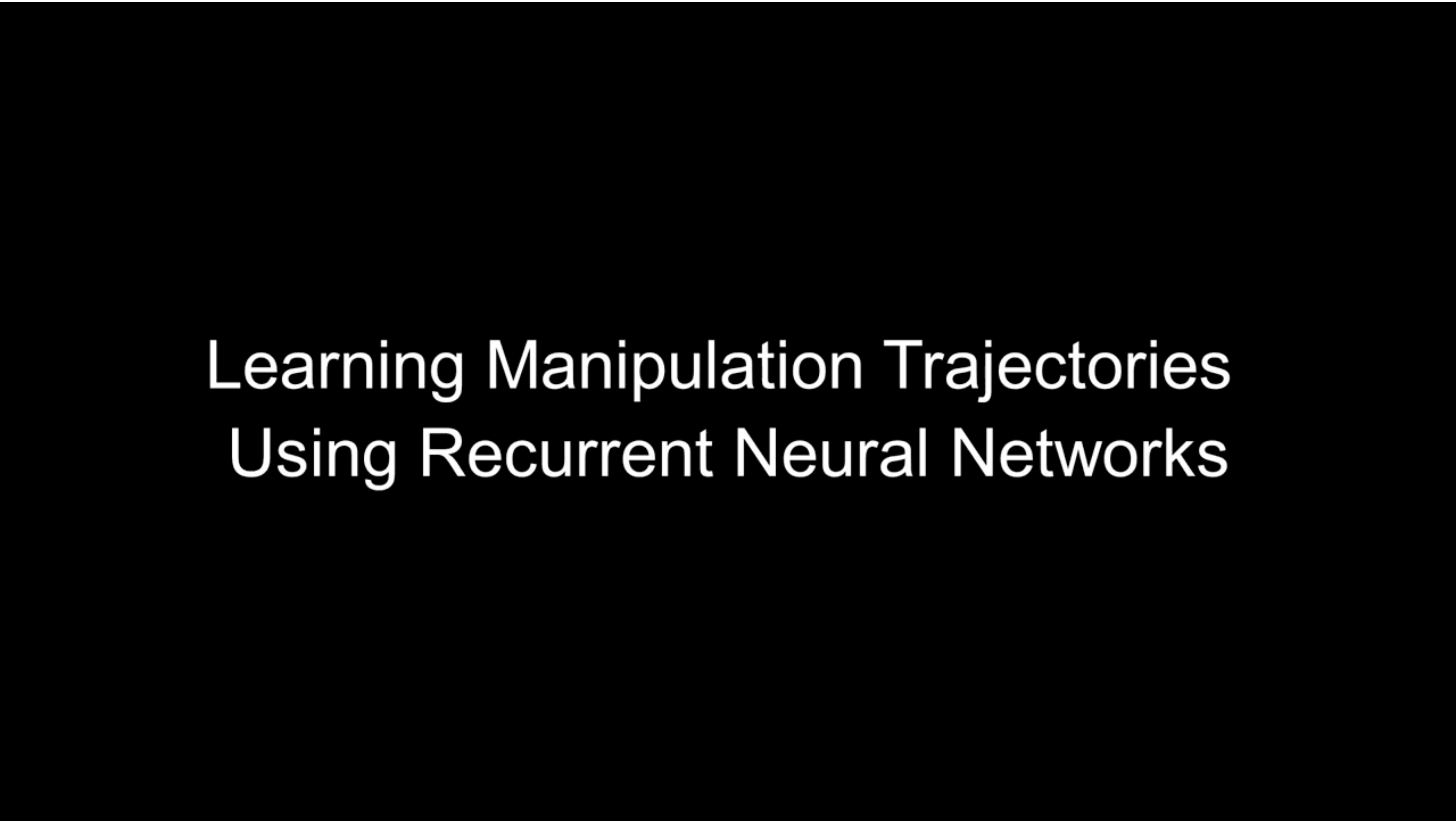
Recurrent Neural Networks for Imitation(1)



https

Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016

Recurrent Neural Networks for Imitation(1)



Learning real manipulation tasks from virtual demonstrations using LSTM, Rahmatizadeh et al 2016

Learning to imitate Search

Task: playing Atari games

1. DQN :model free, knows nothing about the game dynamics
2. MCTS: performs better than DQN but!
   a. takes too long per step to choose the action (too many trees to search)
   b. assume access to the game simulator to ``look ahead”

Idea: instead of learning from trial and error learn to imitate MCTS!

Let MCTS run for long enough to provide the ground-truth actions

Dealing with compounding errors: MCTS uses the current learnt policy to unfold the tree

Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning, Guo et al

Learning to imitate MCTS (2)

Agent	*B.Rider*	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S.Invaders*
DQN	4092	168	470	20	1952	1705	581
-best	5184	225	661	21	4500	1740	1075
UCC	5342 (20)	175 (5.63)	558 (14)	19 (0.3)	11574(44)	2273 (23)	672 (5.3)
-best	10514	351	942	21	29725	5100	1200
-greedy	5676	269	692	21	19890	2760	680
UCC-I	5388 (4.6)	215 (6.69)	601 (11)	19 (0.14)	13189 (35.3)	2701 (6.09)	670 (4.24)
-best	10732	413	1026	21	29900	6100	910
-greedy	5702	380	741	21	20025	2995	692
UCR	2405 (12)	143 (6.7)	566 (10.2)	19 (0.3)	12755 (40.7)	1024 (13.8)	441 (8.1)

Table 2: Performance (game scores) of the off-line UCT game playing agent.

Agent	*B.Rider*	*Breakout*	*Enduro*	*Pong*	*Q*bert*	*Seaquest*	*S.Invaders*
UCT	7233	406	788	21	18850	3257	2354

but… 800 games * 1000 actions/game * 10000 rollouts/action * 300 steps/rollout = 2.4e12 steps

Deep Learning for Real-Time Atari Game Play Using Offline Monte-Carlo Tree Search Planning, Guo et al

Outline

This lecture

- Behavior Cloning: Imitation learning as supervised learning
- Compounding errors
- Demonstration augmentation techniques
- DAGGER
- Structured prediction as Decision Making (learning to search)
- Imitating MCTS

Next lecture:

- Inverse reinforcement learning
- Feature matching
- Max margin planning
- Maximum entropy IRL
- Adversarial Imitation learning