Neural network for reinforcement learning

Tom F. Buchholz
Takayuki Osa

Advanced Topics on Machine Learning, July 8, 2015

Introduction

- We will pick up

Learning Neural Network Policies
with Guided Policy Search under Unknown Dynamics

NIPS 2014, Sergery Levine and Pieter Abbeel

Agenda

- Related works on Neural Network for reinforcement learning

- Method of learning neural network policies with guided policy search

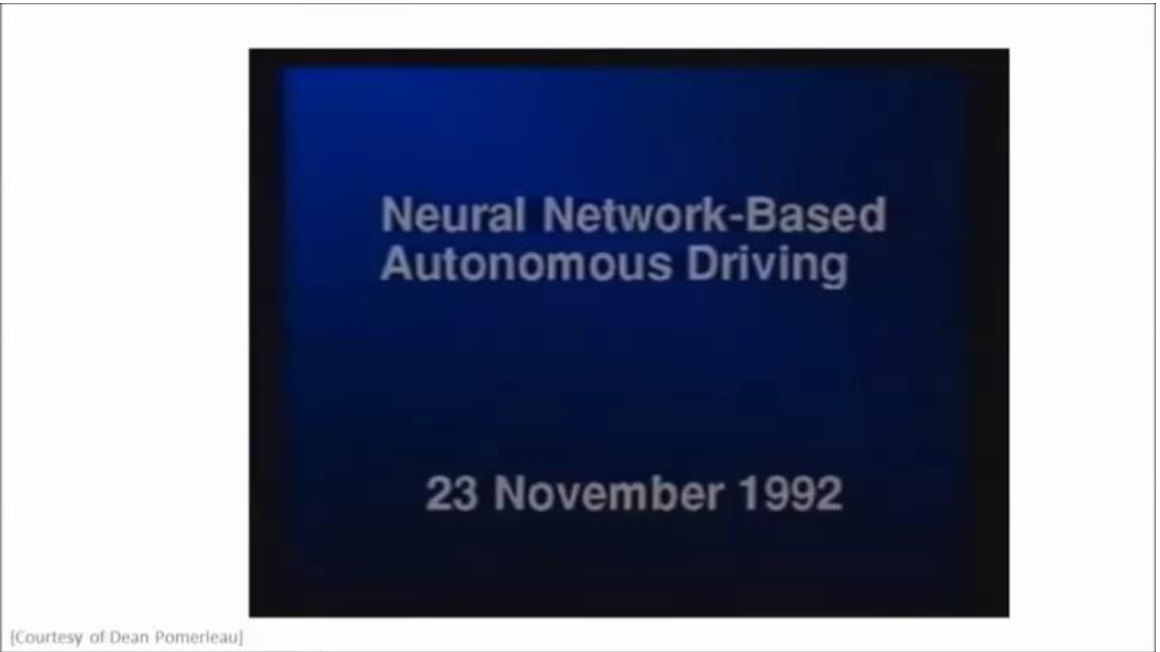
- Some recent results

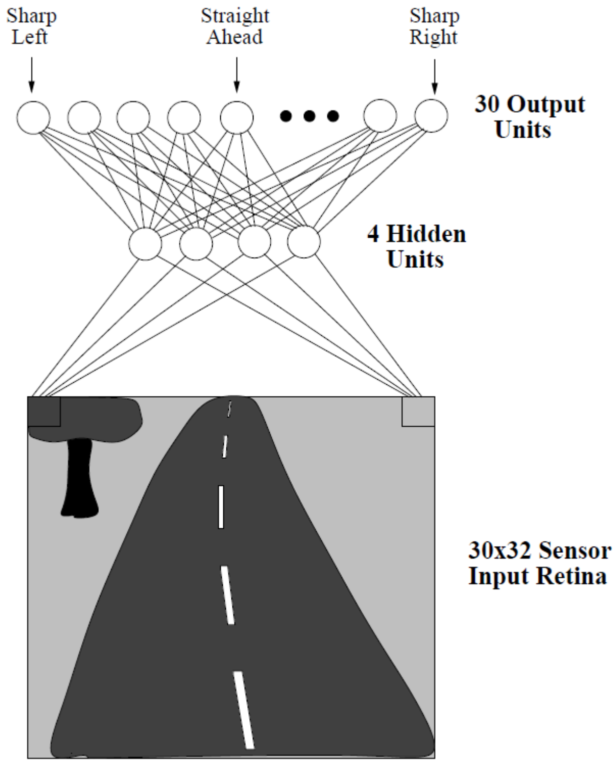
Neural Network for control

- Neural network has been employed for control since 1980s

ALVINN (**A**utonomous **L**and **V**ehicle **I**n a **N**eural **N**etwork) Project

- Neural network was employed to recognize the road in the image input



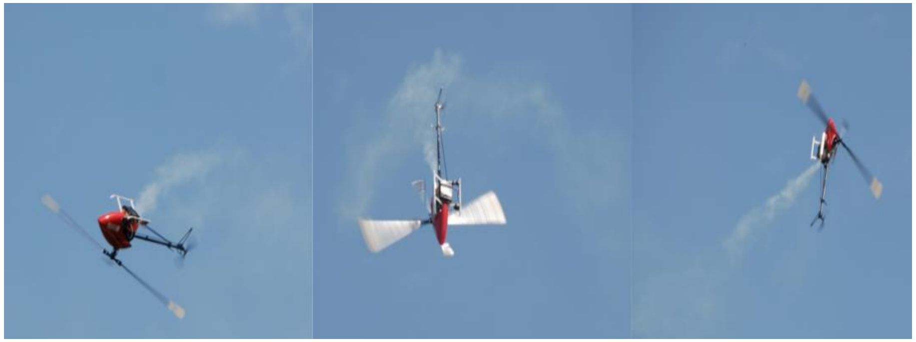


[Pomerleu et al., 1995]

Recent works on Neural Network for control

Deep Learning Helicopter Dynamics Models
(Punjani and Abbeel, 2015, ICRA)

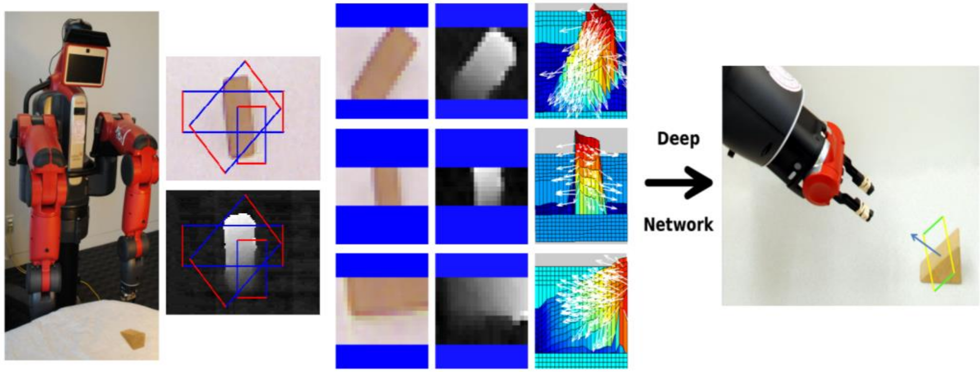
- Neural network with rectified linear unit for learning helicopter dynamics



[Punjani and Abbeel, 2015]

Deep Learning for Detecting Robotic Grasps
(Lenz, Lee, and Saxena, 2014, IJRR)

- Neural network for detecting appropriate features for grasping.

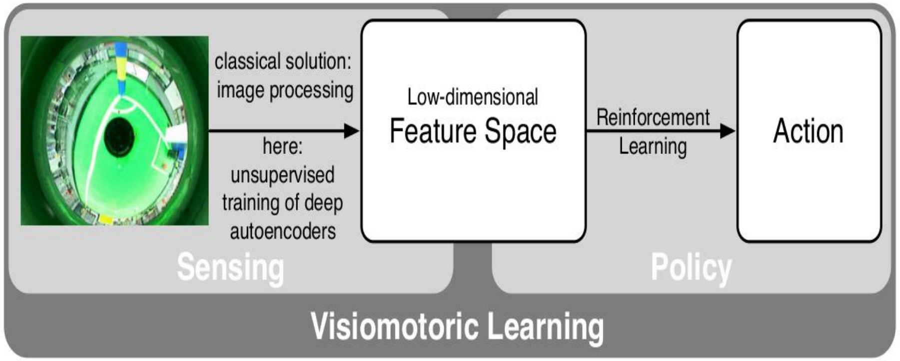


[Lenz, Lee and Saxena, 2014]

Recent works on Neural Network for control

<u>Neural Fitted Q Iteration</u>
(Riedmiller et al. ECML2005, IJCNN2010)

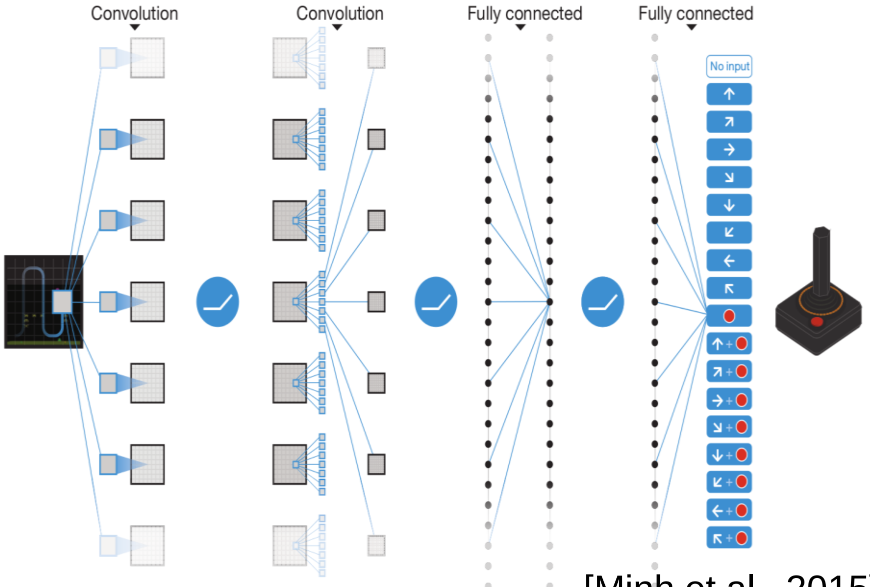
- Auto-encoders for extracting features
- Neural Network for learning Q function



[Lange and Riedmiller, 2010]

<u>Deep-Q-Learning</u>
<u>(Minh et al, 2015, Nature)</u>

- Estimate Q value directly from image input
- Convolutional Neural Network for learning Q function



[Minh et al., 2015]

Neural Network for Reinforcement Learning in 1990s

G. Tesauro “TD-Gammon” Communication of ACM, 1995

- First good example of NN for RL
- Autonomous player of backgammon
- Temporal-difference learning
- Neural network was used to represent nonlinear policy

Pollack and Blair “Why td-gammon work” NIPS 1996

Tsitsiklis and Roy “An analysis of temporal-difference learning with function approximation” Automatic Control,1997

- It was clarified that TD with nonlinear function approximation does not converge

Problem of reinforcement learning with neural network in 1990s

Q learning (Watkins, 1989)

- Learn action-value function

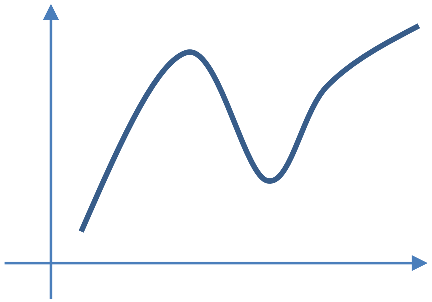
$$Q^*(s,a) = \max_\pi E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s, a_t = a, \pi\right]$$

- Update Q function every time step

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \alpha\left[r_{t+1} + \gamma \max_a Q(s_{t+1},a_t) - Q(s_t,a_t)\right]$$

Problem of studies in 1990s

- when the Q function is approximated with a nonlinear function, updating with sequential data may cause divergence of the optimization.



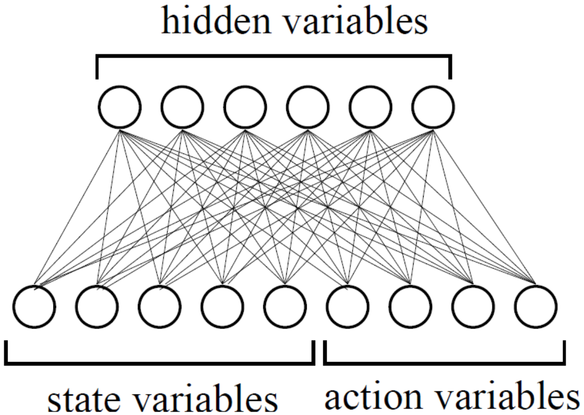
- When the Q function is nonlinear, policy may change rapidly with slight changes to Q values

-> Policy may oscillate

Neural network for reinforcement learning

Sallans and Hinton “Reinforcement Learning with Factored States and Actions” JMLR 2004

- Restricted Boltzmann Machines was used to learn action-value function



[Sallans and Hinton, 2004]

- Action-value function was approximated with negative free energy

$$Q(s,a) \approx -F^{\theta}(s,a)$$

- Action is selected by holding the state variables and sampling the action variables

- But, the problem of convergence was not addressed in this work

“Neural fitted-Q iteration”

Riedmiller et al. ECML2005, IJCNN2010

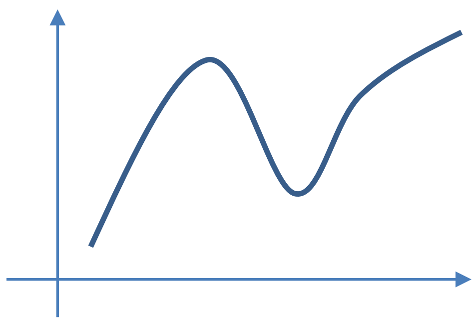
- Learn Action-value function with neural network

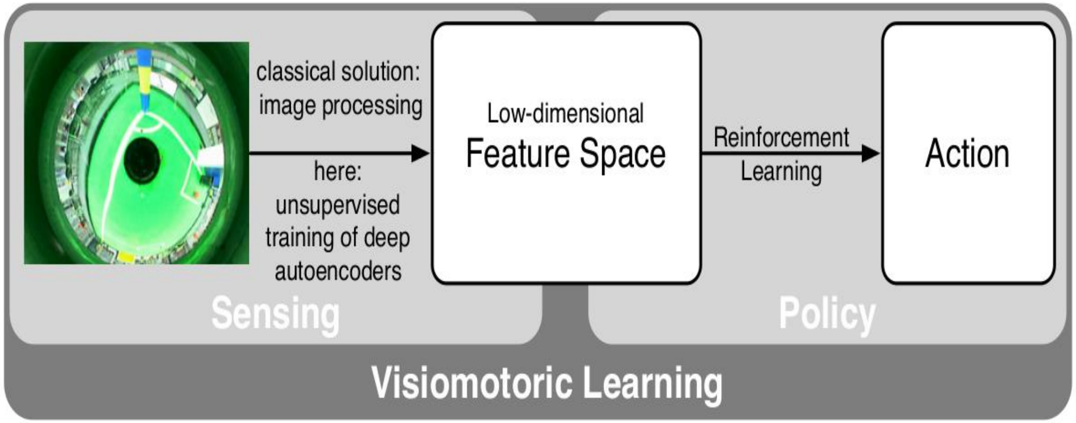
$$Q^*(s,a) = \max_{\pi} E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s, a_t = a, \pi\right]$$

- Update value function at all transition concurrently

- Use whole data to train the Q function, i.e. batch learning

- Computational cost is proportional to the size of data-sets



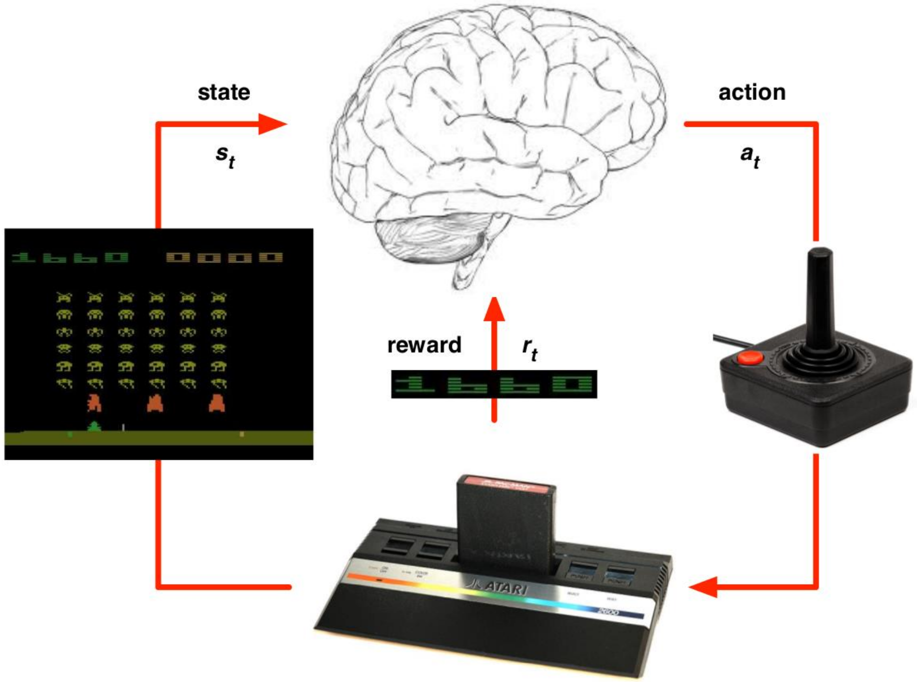


“Human-level control through deep reinforcement learning”

Mnih et al. nature, 2015

- State: (preprocessed) image
  Action: joystick/button position
  Reward: score of the game

- Neural network with convolutional layers and rectified linear units



[Minh et al., 2015]

- Learn Action-value function

$$Q^*(s,a) = \max_\pi E\left[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \mid s_t = s, a_t = a, \pi\right]$$

- Optimize the objective using stochastic gradient descent

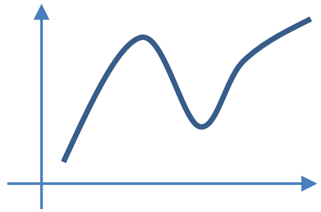
$$L(w) = E_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s',a',w) - Q(s,a,w)\right)^2\right]$$

“Human-level control through deep reinforcement learning”

“Experience replay” - avoids the correlation of sequential data

$$e_t = (s_t, a_t, r_t, s_{t+1})$$

$$D_t = \{e_1, \ldots e_t\}$$ Randomly sample mini-batch → Update Q function



“Fixed target Q-network” - avoids the oscillation of policy

Compute Q-learning targets w.r.t. old fixed parameters w^-

$$r + \gamma \max_{a'} Q(s', a', w^-)$$

Optimize MSE between neural net and Q-learning target

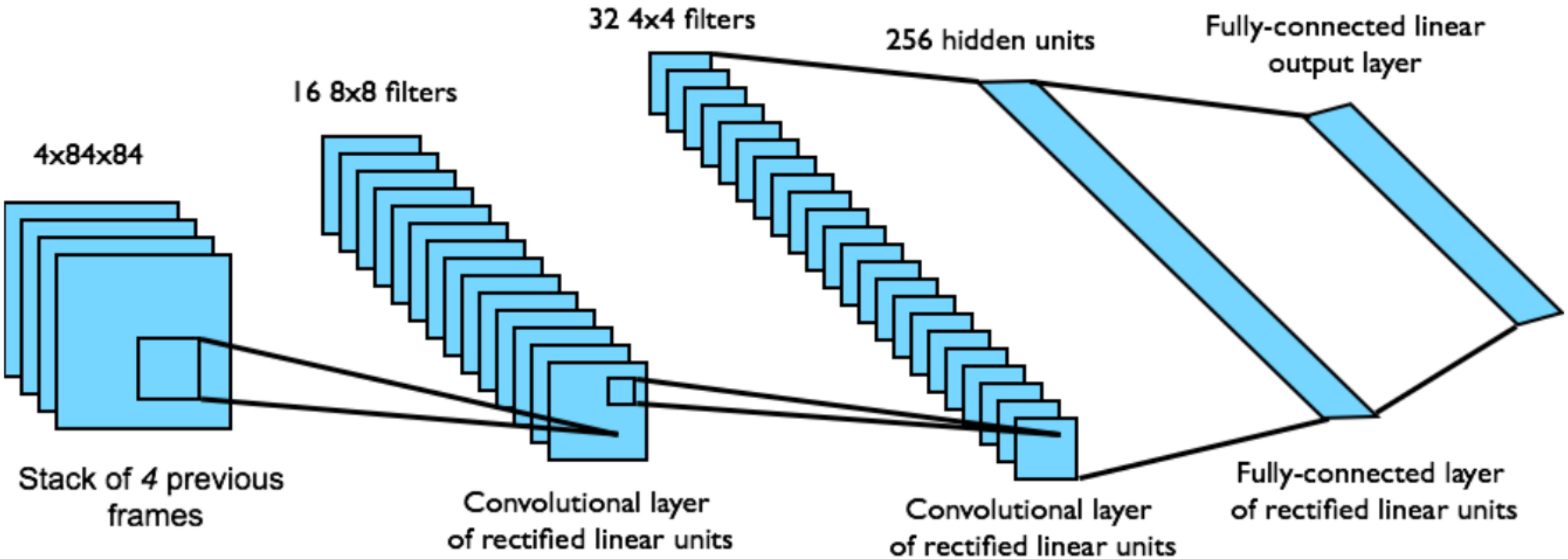
$$L(w) = E_{s,a,r,s' \sim D}\left[\left(r + \gamma \max_{a'} Q(s', a', w^-) - Q(s, a, w)\right)^2\right]$$

Periodically update fixed parameters $w^- \leftarrow w$

Fix this term for a while

“Human-level control through deep reinforcement learning”

Mnih et al. nature, 2015



[Minh et al., 2015]

- Input state is stack of pixel from last 4 frames
- Output is Q value for 18 joystick/button positions

“Human-level control through deep reinforcement learning”

https://www.youtube.com/watch?v=iqXKQf2BOSE [Minh et al., 2015]

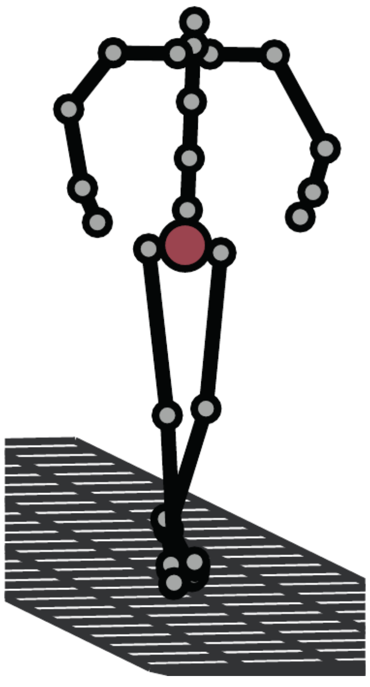
“Human-level control through deep reinforcement learning”

- “Deep Q learning” works well

- Some heuristic techniques are underlying.

- What will happen if we put a constraint of KL divergence in the optimization?

Complex problems



*Graphics from Sergey Levine

Model-free methods:
require low dimensional parameterizations

Model-based methods:
require ability to learn an accurate dynamics model

Recent publications of Sergey Levine

- **“Guided Policy Search”**
- ICML2013

- **“Learning Complex Neural Network Policies with Trajectory Optimization”**
- ICML 2014

- **“Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics”**
- NIPS 2014

- **“Learning Contact-Rich Manipulation Skills with Guided Policy Search”**
- ICRA2015

- **“End-to-End Training of Deep Visuomotor Policies”**
- arXiv 2015

Learning Neural Network Policies with Guided Policy Search under Unknown Dynamics

Sergey Levine and Pieter Abbeel
Department of Electrical Engineering and Computer Science
University of California, Berkeley
Berkeley, CA 94709
{svlevine, pabbeel}@eecs.berkeley.edu

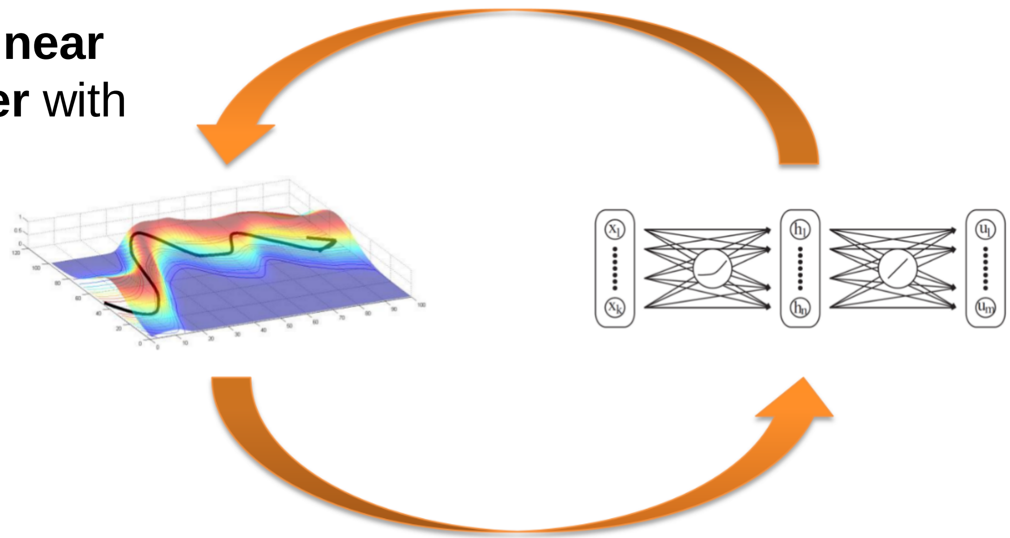
NIPS 2014

Idea

1. Learn **linear dynamics**

5. **Sample** new trajectories from system with linear controller

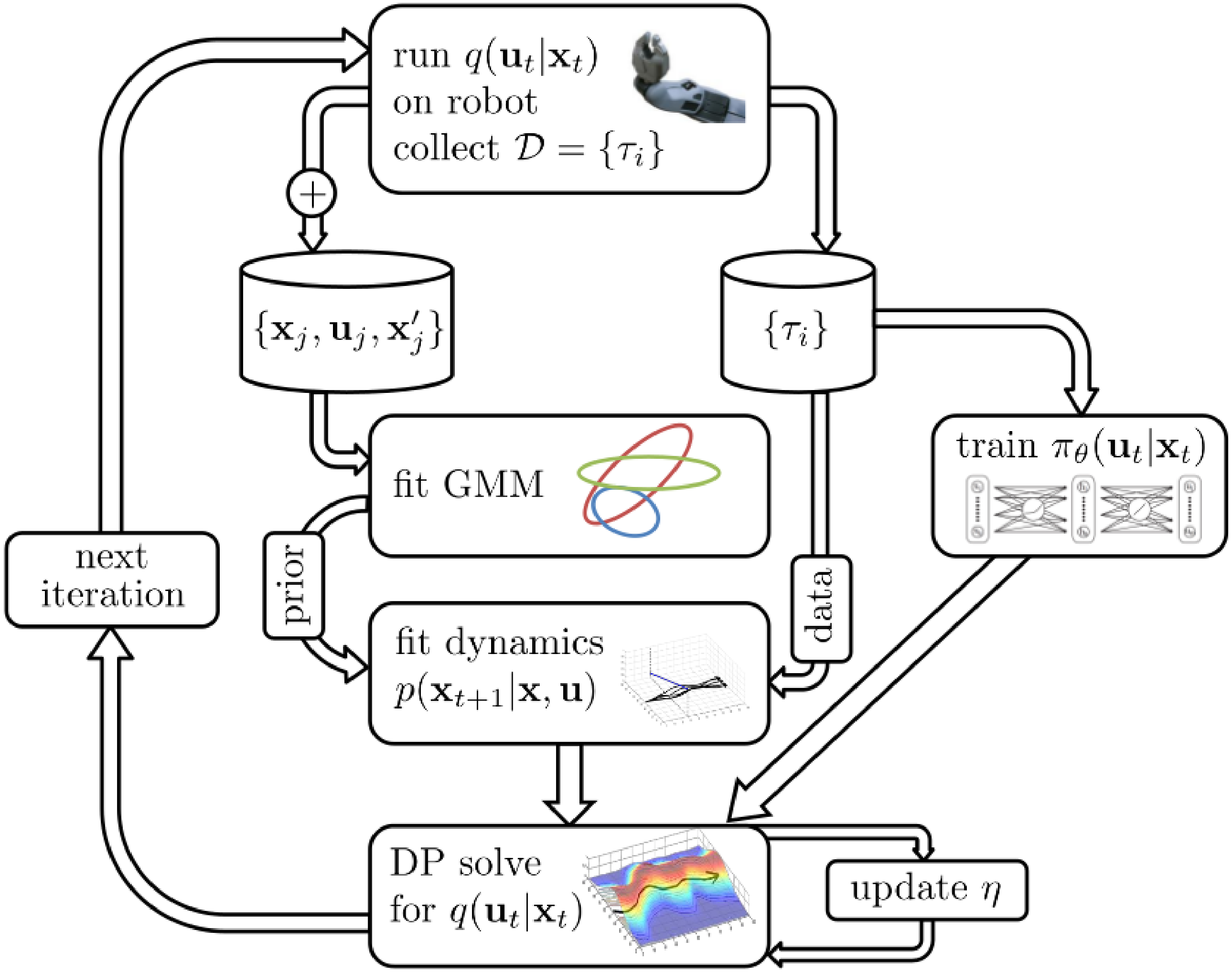
2. Obtain optimal **linear** feedback **controller** with **DDP**



3. **KL-Divergence** constraint to fulfill linearity assumption

4. Use samples to **guide** highly complex and nonlinear policy

Full picture



Fitting linear dynamic model

run $q(\mathbf{u}_t|\mathbf{x}_t)$
on robot
collect $\mathcal{D} = \{\tau_i\}$

$\Rrightarrow \{\tau_i^j\} \Rrightarrow p_i(\mathbf{x_{t+1}}|\mathbf{x_t}, \mathbf{u_t})$

$$\mathcal{N}(f_{\mathbf{x}t}\mathbf{x}_t + f_{\mathbf{u}t}\mathbf{u}_t, \mathbf{F}_t)$$

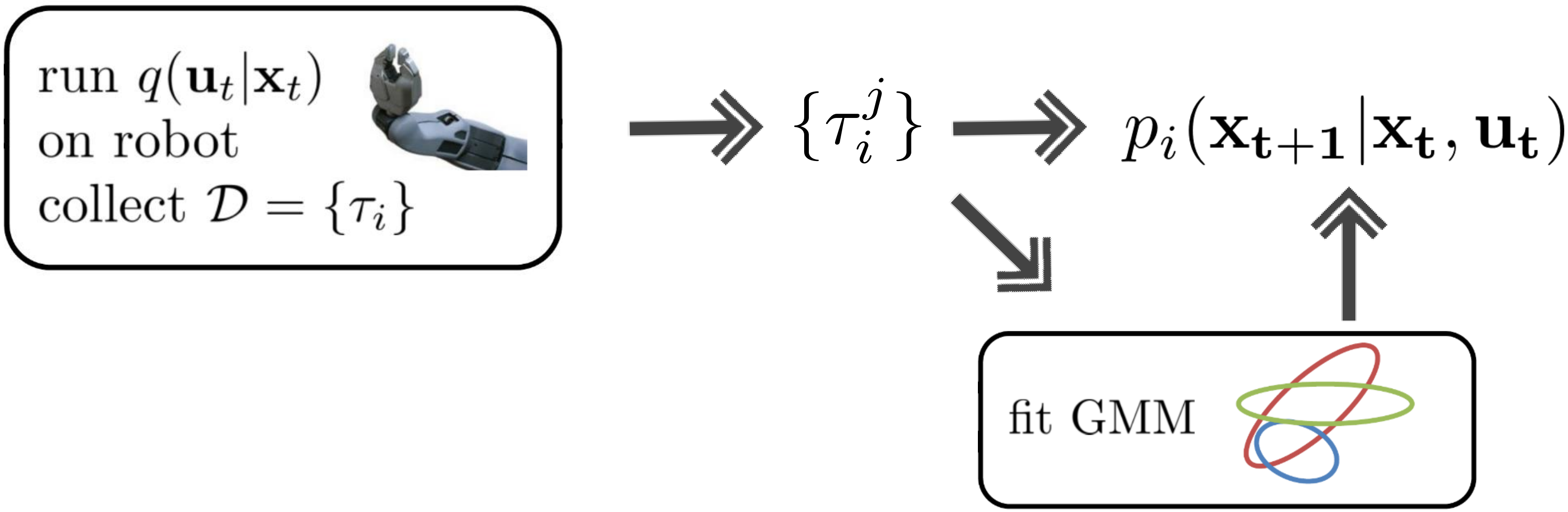
iteratively refitted time-varying local linear model
(Gaussian)

Better dynamics model
→ faster convergence

$\Rrightarrow$

exploit the correlation between
→ nearby time steps and
→ successive iterations

Adding a prior



Gaussian mixture model as a prior to reduce sample count

Soft piecewise linear dynamics

Guided policy search
- Trajectory optimization -

Tassa et al. "Synthesis and Stabilization of Complex Behaviors through Online Trajectory Optimization" IROS 2012

Quadratic cost function

$$\hat{x}_{t+1} \approx f_{xt}\hat{x}_t + f_{ut}\hat{u}_t$$

$$r(u_t, x_t) = \hat{x}_t^T r_{xt} + u_t^T r_{ut} + \frac{1}{2}\hat{x}_t^T r_{xxt}\hat{x}_t + \frac{1}{2}\hat{u}_t^T r_{uut}\hat{u}_t + \hat{u}_t^T r_{uxt}\hat{x}_t + r(\bar{u}_t, \bar{x}_t)$$

Q-function is estimated recursively as:

$$Q_{xxt} = r_{xxt} + f_{xt}^T V_{xxt} f_{xt} \qquad Q_{xt} = r_{xt} + f_{xt}^T V_{xt+1}$$

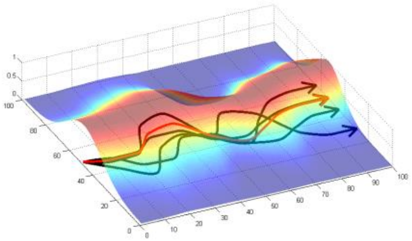
$$Q_{uut} = r_{uut} + f_{ut}^T V_{xxt} f_{ut} \qquad Q_{ut} = r_{ut} + f_{ut}^T V_{xt+1}$$

$$Q_{uxt} = r_{uxt} + f_{ut}^T V_{xxt} f_{xt}$$

Optimal policy is given as:

$$g(x_t) = \bar{u} + k_t + K_t(x_t - \bar{x}_t) \qquad \text{where} \quad k_t = -Q_{uut}^{-1}Q_{ut} \quad K_t = -Q_{uut}^{-1}Q_{uxt}$$

Controller update



Time-varying linear-Gaussian controller

$$p(\mathbf{u_t}|\mathbf{x_t}) = \mathcal{N}(\hat{\mathbf{u}}_\mathbf{t} + \mathbf{k_t} + \mathbf{K_t}(\mathbf{x_t} - \hat{\mathbf{x}}_\mathbf{t}), Q_{\mathbf{u},\mathbf{u}t}^{-1})$$

Restricting the change of the new controller from the old one

$$\min_{p(\tau)\in\mathcal{N}(\tau)} E_p[\ell(\tau)] s.t. D_{KL}(p(\tau)||\hat{p}(\tau)) \leq \epsilon$$

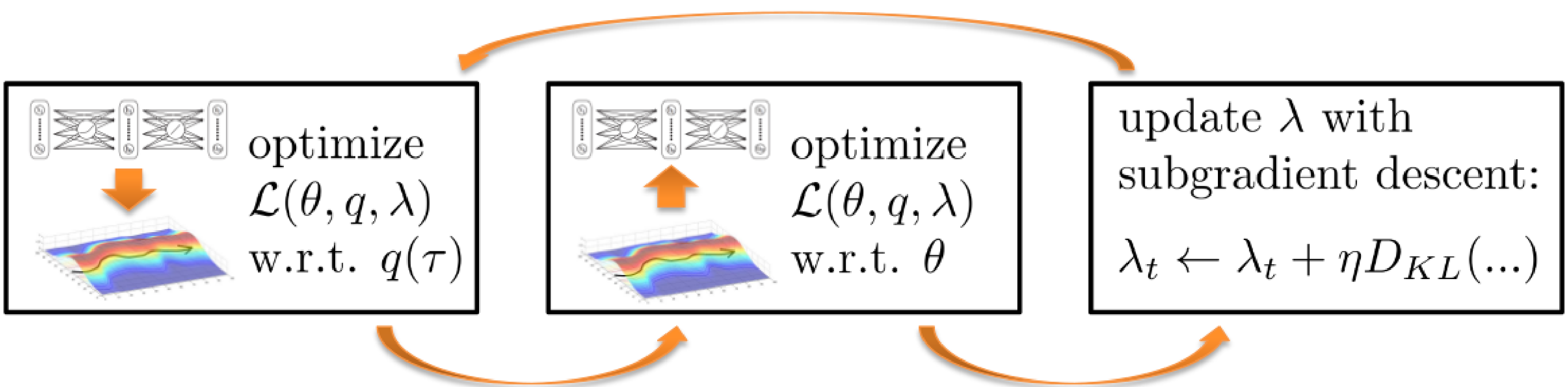
$$\mathcal{L}_{\text{traj}}(p(\tau), \eta) = E_p[\ell(\tau)] + \eta[D_{\text{KL}}(p(\tau)\|\hat{p}(\tau)) - \epsilon]$$

Guided Policy Search

Enforce agreement between policy and linear controller

$$\min_{\theta, p(\tau)} E_{p(\tau)}[\ell(\tau)] \;\; s.t \;\; D_{KL}(\pi_\theta(\mathbf{u_t}|\mathbf{x_t})||p_i(\mathbf{u_t}|\mathbf{x_t})) = 0 \quad \forall t$$

$$\mathcal{L}_{GPS}(\theta, p, \lambda) = E_{p(\tau)}[\ell(\tau)] + \sum_{i,t} \lambda_{i,t} D_{KL}(\pi_\theta(\mathbf{u_t}|\mathbf{x_t})||p_i(\mathbf{u_t}|\mathbf{x_t}))$$



Guided policy search
- Trajectory optimization -

Cost with <u>policy</u> KL divergence

$$L_{\mathrm{GPS}}(p) = E_{p(\tau)}[l(\tau)] + \sum_{t=1}^{T} \lambda_t D_{\mathrm{KL}}\left(p(\mathrm{x}_t)\pi_\theta\left(u_t \mid x_t\right) \| \, p(\mathrm{x}_t, \mathrm{u}_t)\right)$$

Add KL divergence constrain of <u>trajectory</u>

$$L_{\mathrm{GPS}}(p) = E_{p(\tau)}[l(\tau) - \eta \log \hat{p}(\tau)] - \eta H(p) + \sum_{t=1}^{T} \lambda_t D_{\mathrm{KL}}\left(p(\mathrm{x}_t)\pi_\theta\left(u_t \mid x_t\right) \| \, p(\mathrm{x}_t, \mathrm{u}_t)\right)$$

Divide by η and introduce $\tilde{l}(\mathrm{x}_t, \mathrm{u}_t) = \frac{1}{\eta} l(\mathrm{x}_t, \mathrm{u}_t) - \log \hat{p}(\mathrm{u}_t \mid \mathrm{x}_t)$

$$\frac{1}{\eta} L_{\mathrm{GPS}}(p) = E_{p(\tau)}[\tilde{l}(\tau)] - H(p) + \sum_{t=1}^{T} \frac{\lambda_t}{\eta} D_{\mathrm{KL}}\left(p(\mathrm{x}_t)\pi_\theta\left(u_t \mid x_t\right) \| \, p(\mathrm{x}_t, \mathrm{u}_t)\right)$$

Guided policy search
- Trajectory optimization -

$$\frac{1}{\eta} L_{\text{GPS}}(p) = E_{p(\tau)}[\tilde{l}(\tau)] - H(p) + \sum_{t=1}^{T} \frac{\lambda_t}{\eta} D_{\text{KL}}\big(p(\mathrm{x}_t)\pi_\theta(u_t \mid x_t) \| \, p(\mathrm{x}_t, \mathrm{u}_t)\big)$$

$$p(\mathrm{u}_t \mid \mathrm{x}_t) = \mathcal{N}(K_t \mathrm{x}_t + k_t, C_t)$$

second Taylor expansion

linear-Gaussian approximation to the policy

$$\pi_\theta(\mathrm{u}_t \mid \mathrm{x}_t) = \mathcal{N}(\mathrm{u}_t; \mu_{xt}^{\pi}(\hat{\mathrm{x}}_t)\mathrm{x}_t + \mu_{xt}^{\pi}(\hat{\mathrm{x}}_t), \Sigma_t^{\pi})$$

Gaussian mixture model is used to model the prior distribution

Guided policy search
- Trajectory optimization -

$$\frac{1}{\eta} L_{\mathrm{GPS}}(p) = E_{p(\tau)}[\tilde{l}(\tau)] - H(p) + \sum_{t=1}^{T} \frac{\lambda_t}{\eta} D_{\mathrm{KL}}\left(p(\mathrm{x}_t)\pi_\theta\left(u_t \mid x_t\right) \| \, p(\mathrm{x}_t, \mathrm{u}_t)\right)$$

$$p(\mathrm{u}_t \mid \mathrm{x}_t) = \mathcal{N}(K_t \mathrm{x}_t + k_t, C_t)$$

second Taylor expansion

linear-Gaussian approximation to the policy

$$\pi_\theta(\mathrm{u}_t \mid \mathrm{x}_t) = \mathcal{N}(\mathrm{u}_t; \mu_{xt}^{\pi}(\hat{\mathrm{x}}_t)\mathrm{x}_t + \mu_{xt}^{\pi}(\hat{\mathrm{x}}_t), \Sigma_t^{\pi})$$

$(\hat{\mathrm{x}}_t, \hat{\mathrm{u}}_t)^T, \Sigma_t$:Mean and covariance of $p(\mathrm{x}_t, \mathrm{u}_t)$

$$L_{\mathrm{GPS}}(p) \approx \sum_{t=1}^{T} \frac{1}{2} \begin{bmatrix} \hat{\mathrm{x}}_t \\ \hat{\mathrm{u}}_t \end{bmatrix}^T \tilde{l}_{\mathrm{xu,\,xu}t} \begin{bmatrix} \hat{\mathrm{x}}_t \\ \hat{\mathrm{u}}_t \end{bmatrix} + \begin{bmatrix} \hat{\mathrm{x}}_t \\ \hat{\mathrm{u}}_t \end{bmatrix}^T \tilde{l}_{\mathrm{xu}t} + \frac{1}{2} \mathrm{tr}\left(\Sigma_t \tilde{l}_{\mathrm{xu,\,xu}t}\right) - \frac{1}{2} \log\left|C_t\right| +$$

$$\frac{\lambda_t}{2\eta} \log\left|C_t\right| + \frac{\lambda_t}{2\eta} \left(\hat{\mathrm{u}}_t - \mu_t^{\pi}(\hat{\mathrm{x}}_t)\right)^T C_t^{-1} \left(\hat{\mathrm{u}}_t - \mu_t^{\pi}(\hat{\mathrm{x}}_t)\right) + \frac{\lambda_t}{2\eta} \mathrm{tr}\left(C_t^{-1} \Sigma_t^{\pi}\right) +$$

$$\frac{\lambda_t}{2\eta} \mathrm{tr}\left(S_t \left(\mathrm{K}_t - \mu_t^{\pi}(\hat{\mathrm{x}}_t)\right)^T C_t^{-1} \left(\mathrm{K}_t - \mu_t^{\pi}(\hat{\mathrm{x}}_t)\right)\right)$$

S_t : covariance of $p(\mathrm{x}_t)$

Updating variables

How to update k_t, K_t, C_t

$$Q_{\mathrm{xu}t} = l_{\mathrm{xu}t} + f_{\mathrm{xu}t}^T L_{\mathrm{x}t+1}$$

$$Q_{\mathrm{xx}t} = l_{\mathrm{xu,xu}t} + f_{\mathrm{xu}}^T L_{\mathrm{x,x}t+1} f_{\mathrm{xu}}$$

Assuming locally linear dynamics and ignore higher order term

$$L_{\mathrm{u}t} = Q_{\mathrm{u,u}t}\hat{u}_t + Q_{\mathrm{u,x}t}\hat{x}_t + Q_{\mathrm{u}t} + \lambda_t C_t^{-1}\left(\hat{u}_t - \mu_t^{\pi}(\hat{x}_t)\right)$$

$$L_{\mathrm{u,u}t} = Q_{\mathrm{u,u}t} + \lambda_t C_t^{-1}$$

… yield to the following equations.

$$C_t Q_{\mathrm{u,u}t} C_t + (\lambda_t - 1)C_t - \lambda_t M = 0$$

$$k_t = -\left(Q_{\mathrm{u,u}t} + \lambda_t C_t^{-1}\right)^{-1}\left(Q_{\mathrm{u}t} + \lambda_t C_t^{-1}\mu_t^{\pi}(\hat{x}_t)\right)$$

$$K_t = -\left(Q_{\mathrm{u,u}t} + \lambda_t C_t^{-1}\right)^{-1}\left(Q_{\mathrm{u,x}t} + \lambda_t C_t^{-1}\mu_{\mathrm{x}t}^{\pi}(\hat{x}_t)\right)$$

Guided policy search
- Policy optimization -

$$L_{\mathrm{GPS}}(\theta) = \sum_{t=1}^{T} \lambda_t \sum_{i=1}^{N} D_{\mathrm{KL}}\left(\pi_\theta\left(u_t \mid x_{ti}\right) \| \, p(\mathrm{u}_t \mid \mathrm{x}_{ti})\right)$$

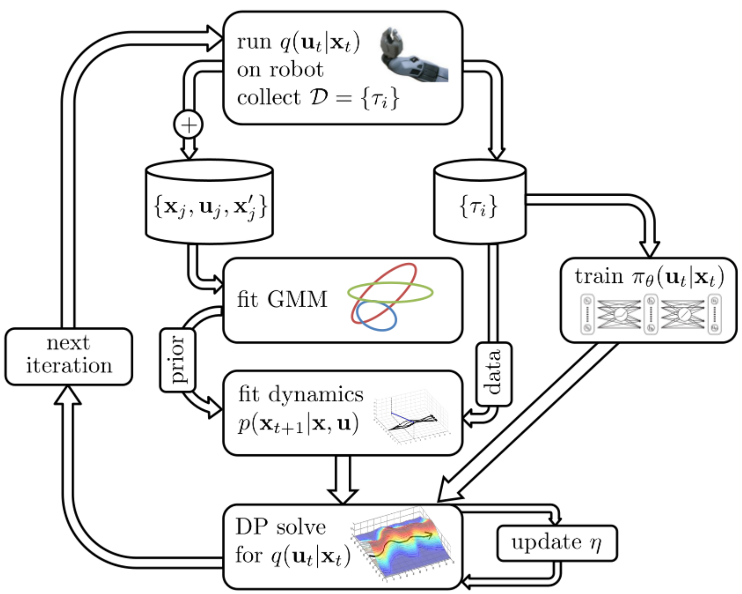
$$= \sum_{t=1}^{T} \lambda_t \sum_{i=1}^{N} \frac{1}{2} \left\{ \mathrm{tr}\left(\Sigma_{\mathrm{t}}^{\pi}(x_{ti}) C_t^{-1}\right) - \log\left|\Sigma^{\pi}(\mathrm{x}_{ti})\right| + \left(K_t x_{ti} + k_t - \mu^{\pi}(x_{ti})\right)^T C_t^{-1} \left(K_t x_{ti} + k_t - \mu^{\pi}(x_{ti})\right) \right\}$$

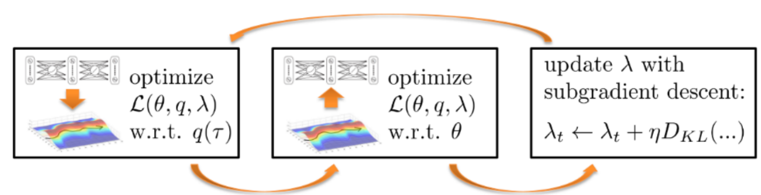
This is equivalent to train neural network in a manner of **supervised learning**.

Recap

Algorithm 1 Guided policy search with unknown dynamics

for iteration $k = 1$ to K **do**
  Generate samples $\{\tau_i^j\}$ from each linear-Gaussian controller $p_i(\tau)$ by performing rollouts
  Fit the dynamics $p_i(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ to the samples $\{\tau_i^j\}$
  Minimize $\sum_{i,t} \lambda_{i,t} D_{\text{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| p_i(\mathbf{x}_t, \mathbf{u}_t))$ with respect to θ using samples $\{\tau_i^j\}$
  Update $p_i(\mathbf{u}_t|\mathbf{x}_t)$ using the algorithm in Section 3 and the supplementary appendix
  Increment dual variables $\lambda_{i,t}$ by $\alpha D_{\text{KL}}(p_i(\mathbf{x}_t)\pi_\theta(\mathbf{u}_t|\mathbf{x}_t) \| p_i(\mathbf{x}_t, \mathbf{u}_t))$
end for
return optimized policy parameters θ

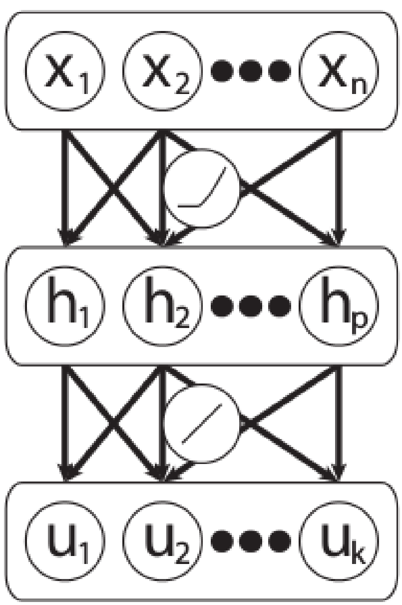


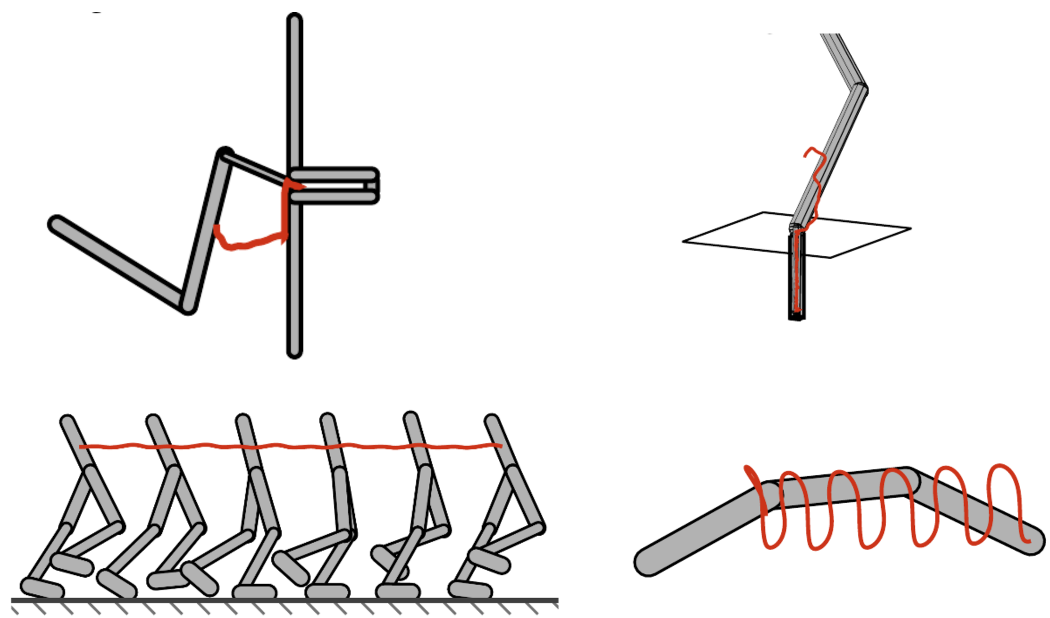


Evaluation

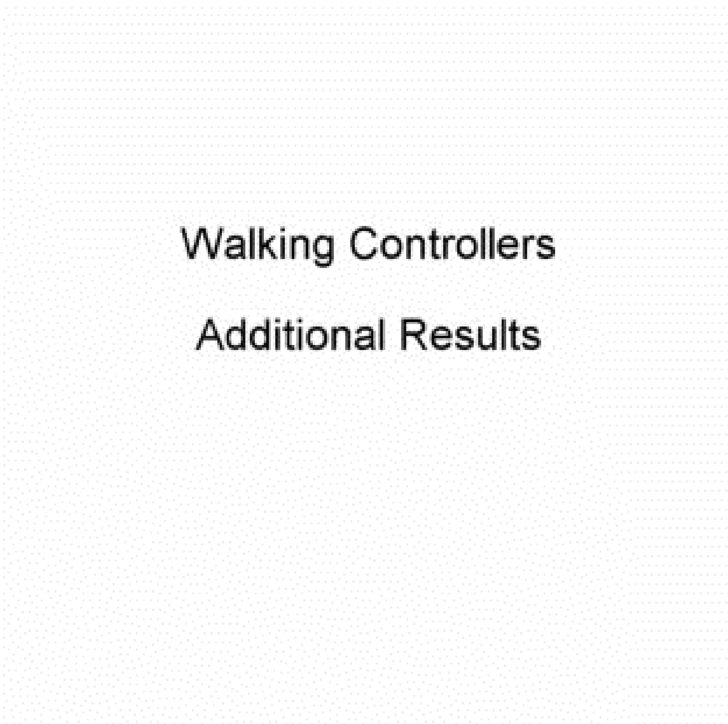
State consists of joint **angles** and **velocities**, action correspond to joint **torques**

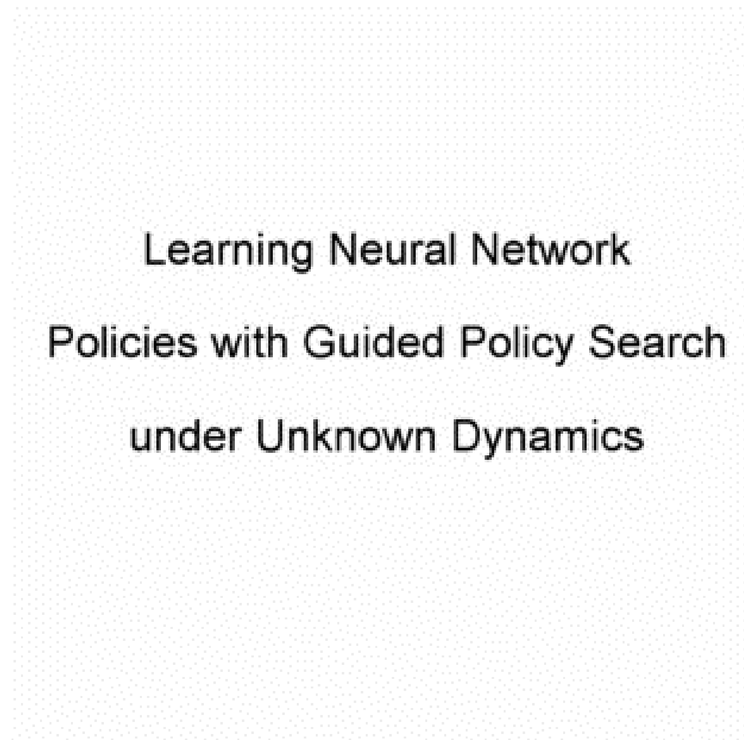
<u>Policy representation:</u>
Neural network with one hidden layer and a soft rectifier nonlinearity of the form $a = \log(1 + \exp(z))$





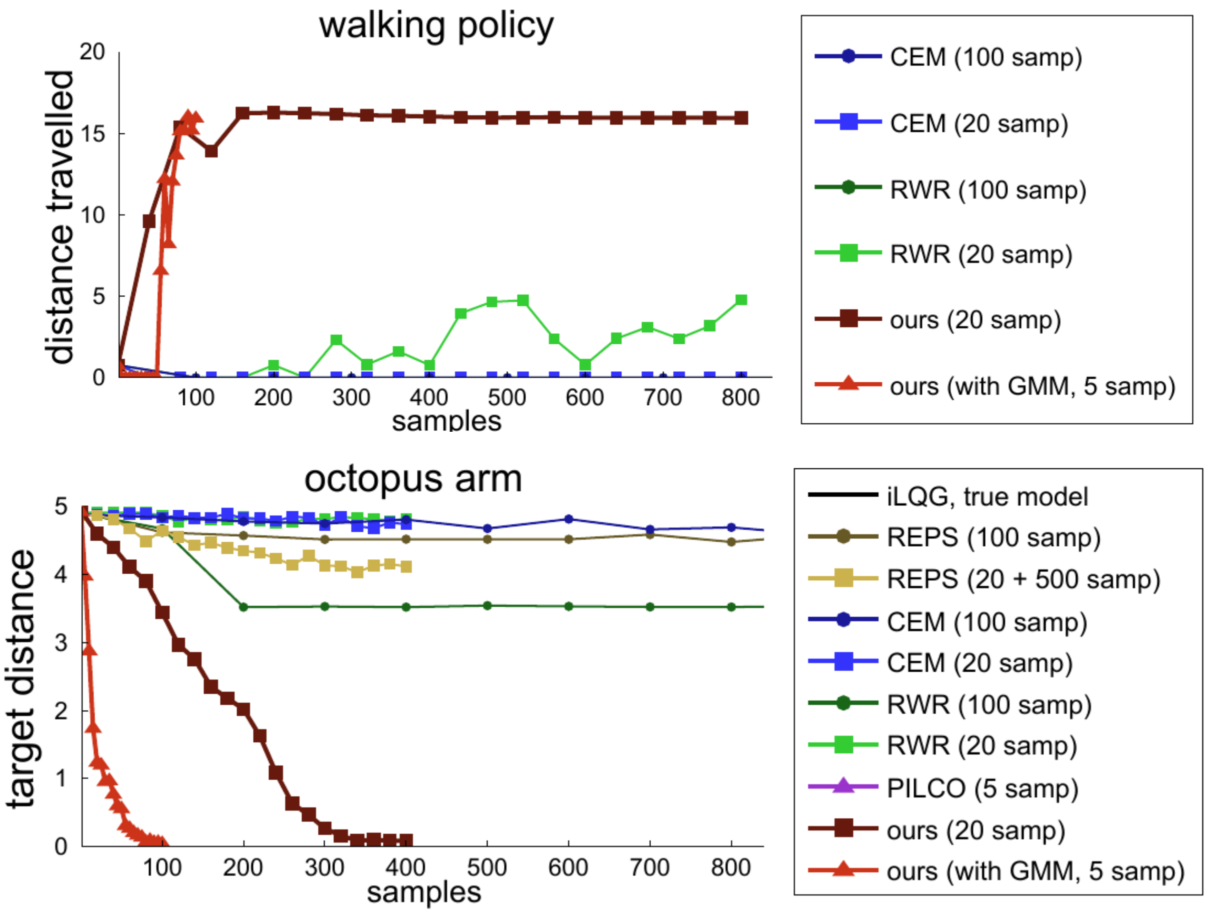
Results





http://rll.berkeley.edu/nips2014gps/

Results



walking policy
distance travelled
samples
0
5
10
15
20
100
200
300
400
500
600
700
800
CEM (100 samp)
CEM (20 samp)
RWR (100 samp)
RWR (20 samp)
ours (20 samp)
ours (with GMM, 5 samp)
octopus arm
target distance
samples
0
1
2
3
4
5
100
200
300
400
500
600
700
800
iLQG, true model
REPS (100 samp)
REPS (20 + 500 samp)
CEM (100 samp)
CEM (20 samp)
RWR (100 samp)
RWR (20 samp)
PILCO (5 samp)
ours (20 samp)
ours (with GMM, 5 samp)

Results from ICRA paper

http://rll.berkeley.edu/icra2015gps/index.htm

End-to-End Training of Deep Visuomotor Policies

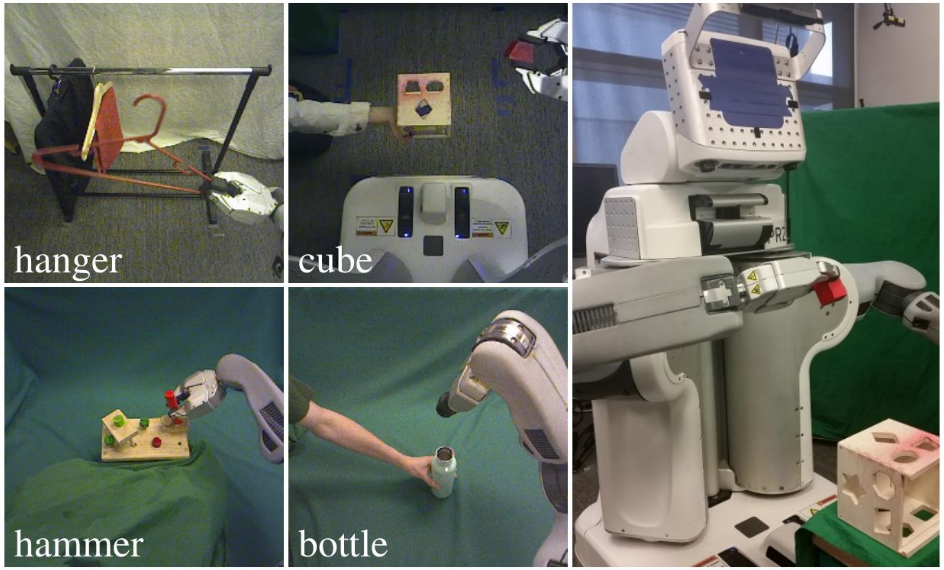
Sergey Levine*, Chelsea Finn*, Trevor Darrell, Pieter Abbeel
Department of Electrical Engineering and Computer Sciences, UC Berkeley
{svlevine,cbfinn,trevor,pabbeel}@eecs.berkeley.edu

ArXiv 2 Apr 2015

More recent result
- "End-to-End Training of Deep Visuomotor Policies" -

Practical applications often require **hand-engineered** components for **perception, state estimation** and **low-level control**

Learning policies that map raw, low-level observations like **camera images directly to joint torques**

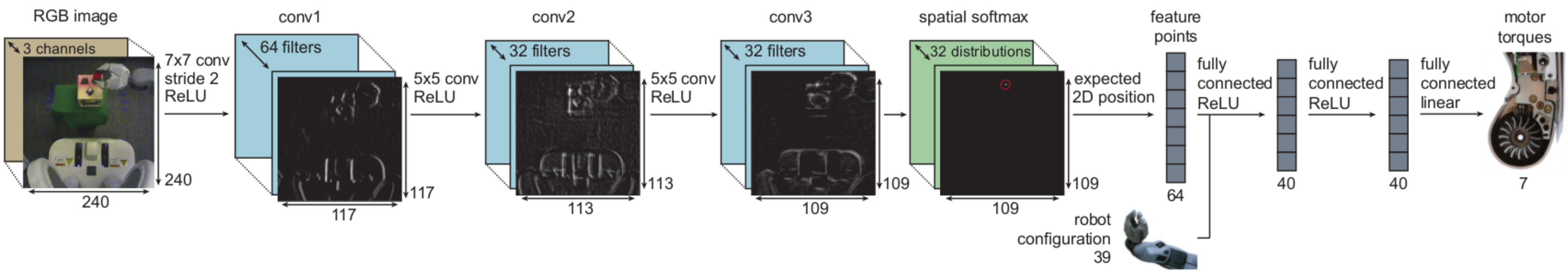


$$\pi_\theta(\mathbf{u}_t|\mathbf{o}_t)$$

Observations instead of states

End-to-End Training of Deep Visuomotor Policies

Results from arXiv paper “End-to-End Training of Deep Visuomotor Policies”

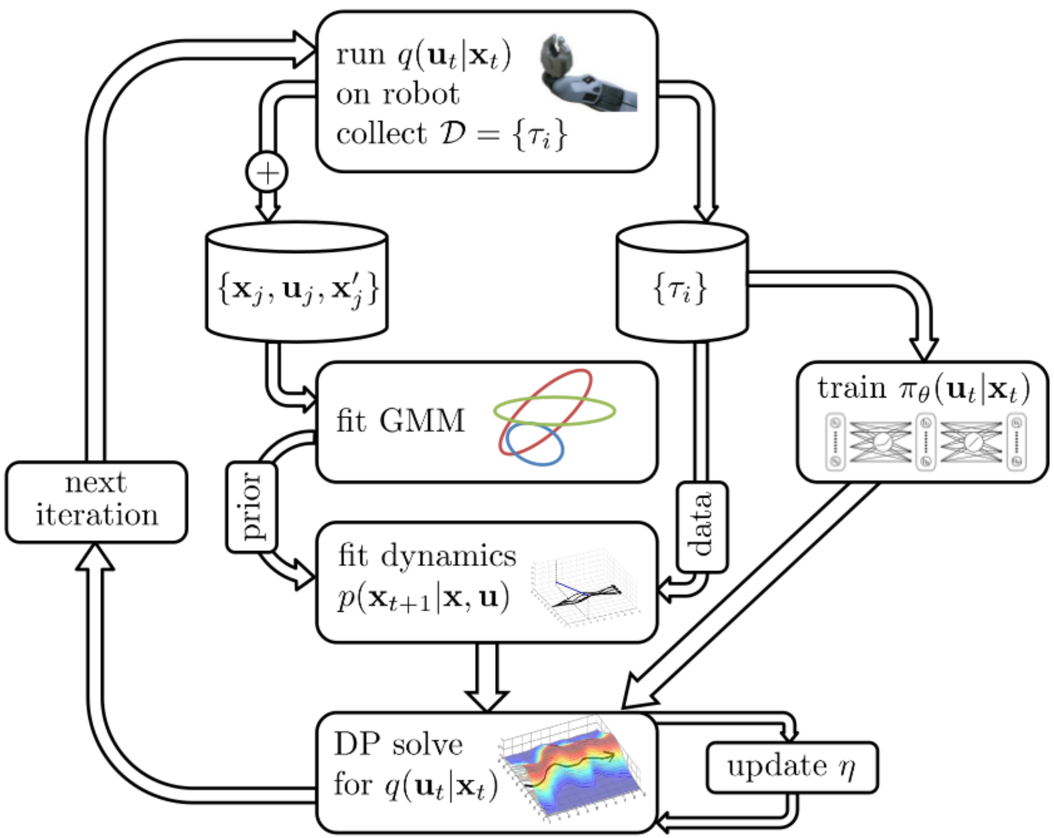


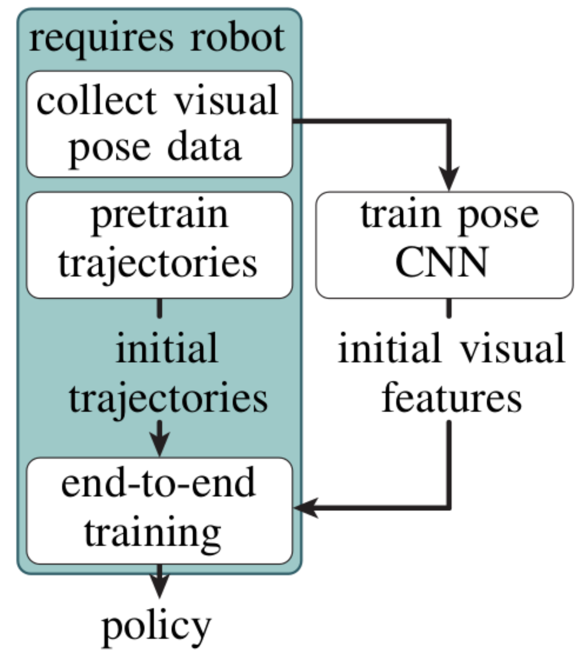
Deep Convolutional neural network with 92,000 parameters and 7 layers for **extracting features and determining control input**

The convolutional layer was **pre-trained with ImageNet** dataset.

More recent result
- “End-to-End Training of Deep Visuomotor Policies” -

Input to the neural network policy: image input





http://sites.google.com/site/visuomotorpolicy

End-to-End Training of Deep Visuomotor Policies

Learned Visual Representations

http://sites.google.com/site/visuomotorpolicy

Summary

Learning nonlinear policy in reinforcement learning was not successful until recently

Sergey achieved learning nonlinear policy with neural network by using KL divergence constraint on trajectory and policy optimization

Neural network is playing an importance role to enable high dimensional regression in reinforcement learning