

Deep Reinforcement Learning and Control

# Optimal Control, Trajectory Optimization, Learning Dynamics

Katerina Fragkiadaki



# So far..

- Most Reinforcement Learning literature: Learning policies without knowing how the world works (model-free)
- Model based RL: build a model from data and use it to sample experience as opposed to just use experience by interacting with the world
- Imitation learning: learn from a teacher -> most recent and successful formulations suggest to train a classifier/regressor with data augmentation/scheduling to handle compounding errors
- Planners (e.g. Monte Carlo Tree Search)-> Search for actions while knowing how the world works (model-based)->Search in *Discrete* space.
- Imitation learning: learn from a planner: imitate MCTS to learn to play Atari games better than what you would get from model-free RL
- Sidd assumed he knew some simplified models of Physics and that allowed him to do formidable look-aheads using MCTS and particle trajectories.

# Today's lecture

- Most Reinforcement Learning literature: Learning policies without knowing how the world works (model-free)
- Model based RL: build a model from data and use it to sample experience as opposed to just use experience by interacting with the world
- Imitation learning: learn from a teacher -> most recent and successful formulations suggest to train a classifier/regressor with data augmentation/scheduling to handle compounding errors
- Planners (e.g. Monte Carlo Tree Search)-> Search for actions while knowing how the world works (model-based)->Search in *Discrete* space.
- Imitation learning: learn from a planner: imitate MCTS to learn to play Atari games better than what you would get from model-free RL
- Sidd assumed he knew some simplified models of Physics and that allowed him to do formidable look-aheads using MCTS and particle trajectories.
- Optimal Control: Search for actions while knowing how the world works (model-based) -> Search in Continuous space. (We will use derivates).

# Today's lecture

- Optimal Control, trajectory optimization formulation
- Special but important case: Linear dynamics, quadratic costs (LQR)
- iterative-LQR / Differential Dynamic programming for Non-linear dynamical systems
- Examples of when it works and when it does not
- Learning Dynamics: Global Models

# Next two lectures

- Learning Dynamics: Local Models
- Learning local or global controllers with a trust region based on KL divergence of their trajectory distributions (i-LQR, TRPO)
- Imitating Optimal Controllers to find **general** policies: execute desired task from any initial state
- Successful examples from the literature

# Optimal Control (Open Loop)

- The optimal control problem:

$$\min_{x,u} \sum_{t=0}^T c_t(x_t, u_t)$$

s.t.  $x_0 = \bar{x}_0$

$$x_{t+1} = f(x_t, u_t) \quad t = 0, \dots, T - 1$$

# Optimal Control (Open Loop)

- The optimal control problem:

$$\min_{x,u} \sum_{t=0}^T c_t(x_t, u_t)$$

s.t.  $x_0 = \bar{x}_0$

$$x_{t+1} = f(x_t, u_t) \quad t = 0, \dots, T - 1$$

- Solution:
  - Sequence of controls  $u$  and resulting state sequence  $x$
  - In general non-convex optimization problem, can be solved with sequential convex programming (SCP): [https://stanford.edu/class/ee364b/lectures/seq\\_slides.pdf](https://stanford.edu/class/ee364b/lectures/seq_slides.pdf)

# Optimal Control (Closed Loop a.k.a. MPC)

Given:  $\bar{x}_0$

For  $t = 0, 1, 2, \dots, T$

- Solve

$$\min_{x,u} \sum_{k=t}^T c_k(x_k, u_k)$$

$$\text{s.t. } x_{k+1} = f(x_k, u_k), \quad \forall k \in \{t, t+1, \dots, T-1\}$$

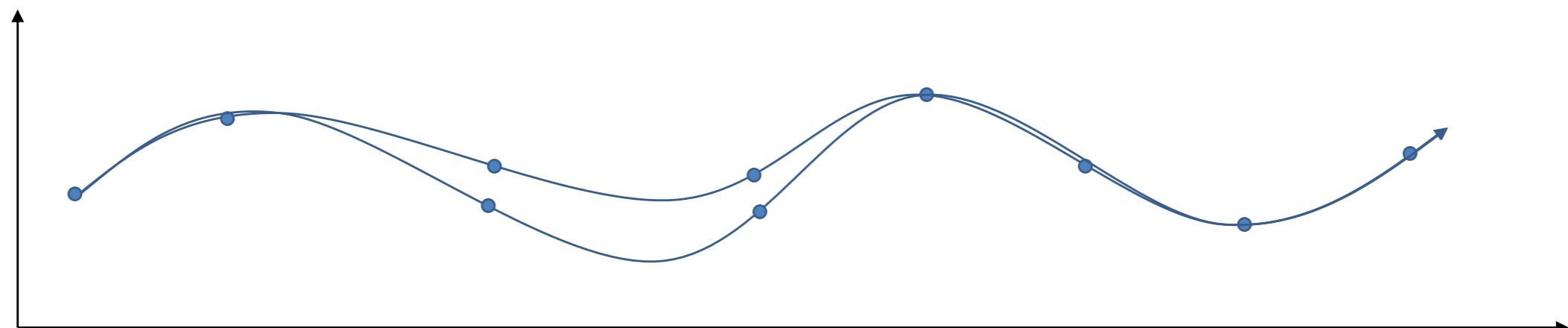
$$x_t = \bar{x}_t$$

- Execute  $u_t$
- Observe resulting state,  $\bar{x}_{t+1}$
- Initialize with solution from  $t - 1$  to solve fast at time  $t$

# Shooting methods vs collocation methods

Collocation Method: optimize over actions and state, with constraints

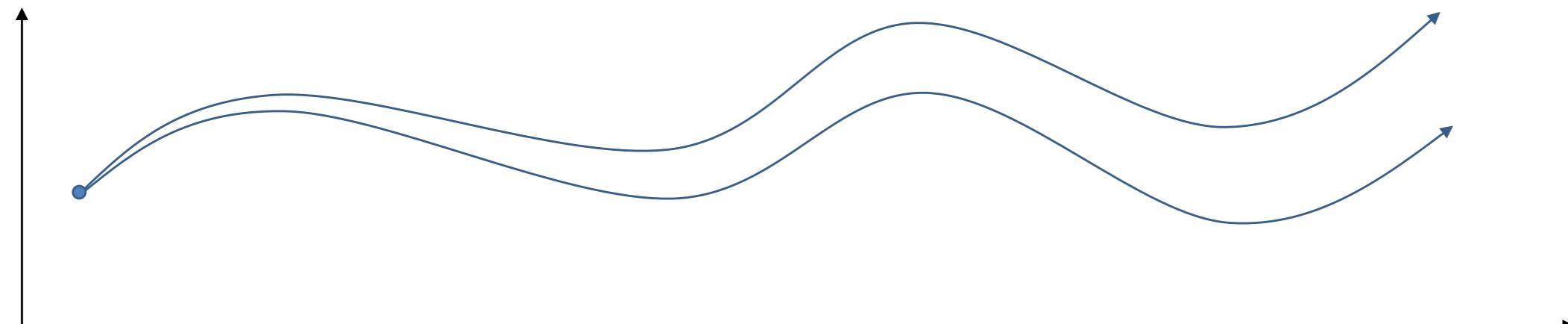
$$\min_{u_1, \dots, u_T, x_1, \dots, x_T} \sum_{t=1}^T c(x_t, u_t) \text{ s.t. } x_t = f(x_{t-1}, u_{t-1})$$



# Shooting methods vs collocation methods

Shooting Method: optimize over actions only

$$\min_{u_1, \dots, u_T} c(x_1, u_1) + c(f(x_1, u_1), u_2) + \dots + c(f(f(\dots)), u_T)$$



Indeed,  $x$  are not necessary since every  $u$  results (following the dynamics) in a state sequence  $x$ , for which in turn the cost can be computed

- Not clear how to initialize in a way that nudges towards a goal state

# Bellman's Curse of Dimensionality

- n-dimensional state space
- Number of states grows exponentially in n (for fixed number of discretization levels per coordinate)
- In practice
  - Discretization is considered only computationally feasible up to 5 or 6 dimensional state spaces even when using
    - Variable resolution discretization
    - Highly optimized implementations

# Linear case: LQR

- Very special case: Optimal Control for Linear Dynamic Systems and Quadratic Cost (a.k.a. LQ setting)
- Can solve continuous state-space optimal control problem exactly
- Running time:  $O(Tn^3)$

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots)), \mathbf{u}_T)$$

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

---

linear

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

---

quadratic

# Linear dynamics: Newtonian Dynamics



- $x_{t+1} = x_t + \Delta t \dot{x}_t + \Delta t^2 F_x$
- $y_{t+1} = y_t + \Delta t \dot{y}_t + \Delta t^2 F_y$
- $\dot{x}_{t+1} = \dot{x}_t + \Delta t F_x$
- $\dot{y}_{t+1} = \dot{y}_t + \Delta t F_y$

# What is the state $x$ ?

In most robotic tasks, state is hand engineered and includes:

- position and velocities of the robotic joints
- position and velocity of the object being manipulated

Those are both known: the robot knows its state and we perceive the state of the objects in the world. In tasks where we do not even want to bother with object state, we just concatenate the robotic state across multiple time steps to implicitly infer the interaction (collision with the object)

Later lecture: learning the state from raw pixels, Embed to Control

# What is the cost $c(x_t, u_t)$

- $c(x_t, u_t) = \|x_t - x^*\| + \beta \|u_t\|$

$x^*$  is a final goal configuration I want to achieve

- In the final time step, you can add a term with higher weight:

Final cost       $c(x_T, u_T) = 2(\|x_T - x^*\| + \beta \|u_T\|)$

- For object manipulation,  $x^*$  includes not only desired pose of the end effector but also desired pose of the objects

# Linear Quadratic Regulator (LQR)

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots)), \mathbf{u}_T)$$

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

Definitions:

$Q(x_t, u_t)$ : optimal action value function, cost-to-go at state  $x_t$  executing control  $u_t$

$V(x_t)$ : optimal state value function, cost-to-go from state  $x_t$

$$V(x_t) = \min_{u_t} Q(x_t, u_t)$$

# Linear Quadratic Regulator (LQR)

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots) \dots), \mathbf{u}_T)$$

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

Value iteration: backward propagation!

Start from  $u_T$  and work backwards

# Linear Quadratic Regulator (LQR)

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + \underbrace{c(f(f(\dots) \dots), \mathbf{u}_T)}_{\text{only term that depends on } \mathbf{u}_T}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

Value iteration: backward propagation!

Start from  $\mathbf{u}_T$  and work backwards

$$Q(\mathbf{x}_T, \mathbf{u}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{c}_T$$

Cost matrices

for the last time step:

$$\mathbf{C}_T = \begin{bmatrix} \mathbf{C}_{\mathbf{x}_T, \mathbf{x}_T} & \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \\ \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} & \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \end{bmatrix}$$

$$\mathbf{c}_T = \begin{bmatrix} \mathbf{c}_{\mathbf{x}_T} \\ \mathbf{c}_{\mathbf{u}_T} \end{bmatrix}$$

# Linear Quadratic Regulator (LQR)

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + \underbrace{c(f(f(\dots) \dots), \mathbf{u}_T)}_{\text{only term that depends on } \mathbf{u}_T}$$

$$c(\mathbf{x}_t, \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{C}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{c}_t$$

$$f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{F}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \mathbf{f}_t$$

Value iteration: backward propagation!

Start from  $\mathbf{u}_T$  and work backwards

$$Q(\mathbf{x}_T, \mathbf{u}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{u}_T \end{bmatrix}^T \mathbf{c}_T$$

$$\nabla_{\mathbf{u}_T} Q(\mathbf{x}_T, \mathbf{u}_T) = \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} \mathbf{x}_T + \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{u}_T + \mathbf{c}_{\mathbf{u}_T}^T = 0$$

$$\mathbf{u}_T = -\mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T}^{-1} (\mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} \mathbf{x}_T + \mathbf{c}_{\mathbf{u}_T})$$

$$\mathbf{u}_T = \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T$$

$$\mathbf{K}_T = -\mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T}^{-1} \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T}$$

$$\mathbf{k}_T = -\mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T}^{-1} \mathbf{c}_{\mathbf{u}_T}$$

only term that depends on  $\mathbf{u}_T$

Cost matrices

for the last time step:

$$\mathbf{C}_T = \begin{bmatrix} \mathbf{C}_{\mathbf{x}_T, \mathbf{x}_T} & \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \\ \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} & \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \end{bmatrix}$$

$$\mathbf{c}_T = \begin{bmatrix} \mathbf{c}_{\mathbf{x}_T} \\ \mathbf{c}_{\mathbf{u}_T} \end{bmatrix}$$

# Linear Quadratic Regulator (LQR)

Remember:  $V(x_t) = \min_{u_t} Q(x_t, u_t)$

Substituting the minimizer  $u_T$  into  $Q(x_T, u_T)$  gives us  $V(x_T)$ !

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{C}_T \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix} + \begin{bmatrix} \mathbf{x}_T \\ \mathbf{K}_T \mathbf{x}_T + \mathbf{k}_T \end{bmatrix}^T \mathbf{c}_T$$

$$\begin{aligned} V(\mathbf{x}_T) = & \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T, \mathbf{x}_T} \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{K}_T \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} \mathbf{x}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{K}_T \mathbf{x}_T + \\ & \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{k}_T + \frac{1}{2} \mathbf{x}_T^T \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{k}_T + \mathbf{x}_T^T \mathbf{c}_{\mathbf{x}_T} + \mathbf{x}_T^T \mathbf{K}_T^T \mathbf{c}_{\mathbf{u}_T} + \text{const} \end{aligned}$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \mathbf{x}_T^T \mathbf{V}_T \mathbf{x}_T + \mathbf{x}_T^T \mathbf{v}_T$$

cost-to-go as a function of  
the final state

$$\mathbf{V}_T = \mathbf{C}_{\mathbf{x}_T, \mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{K}_T + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{x}_T} + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{K}_T$$

$$\mathbf{v}_T = \mathbf{c}_{\mathbf{x}_T} + \mathbf{C}_{\mathbf{x}_T, \mathbf{u}_T} \mathbf{k}_T + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T} + \mathbf{K}_T^T \mathbf{C}_{\mathbf{u}_T, \mathbf{u}_T} \mathbf{k}_T$$

# Linear Quadratic Regulator (LQR)

We propagate the optimal value function backwards!!

Immediate cost

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{C}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{c}_{T-1} + V(f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}))$$

best cost-to-go

$$q_*(s, a) = r(s, a) + \gamma \sum_{s' \in S} T(s'|s, a) v_*(s')$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \mathbf{x}_T^T \mathbf{V}_T \mathbf{x}_T + \mathbf{x}_T^T \mathbf{v}_T$$

We can eliminate  $\mathbf{x}_T$  by writing only in terms of quantities of  $T-1$ !

$$f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \mathbf{x}_T = \mathbf{F}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \mathbf{f}_{T-1}$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}}_{\text{quadratic}} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \underbrace{\begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}}_{\text{linear}} + \underbrace{\begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{F}_{T-1}^T \mathbf{v}_T}_{\text{linear}}$$

We have written  $V(x_T)$  only in terms of  $x_{T-1}, u_{T-1}$  !

# Linear Quadratic Regulator (LQR)

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{C}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{c}_{T-1} + V(f(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}))$$

$$V(\mathbf{x}_T) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}}_{\text{quadratic}} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1}}_{\text{linear}} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \underbrace{\mathbf{F}_{T-1}^T \mathbf{v}_T}_{\text{linear}}$$

$$Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{Q}_{T-1} \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix} + \begin{bmatrix} \mathbf{x}_{T-1} \\ \mathbf{u}_{T-1} \end{bmatrix}^T \mathbf{q}_{T-1}$$

$$\mathbf{Q}_{T-1} = \mathbf{C}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{F}_{T-1}$$

$$\mathbf{q}_{T-1} = \mathbf{c}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{V}_T \mathbf{f}_{T-1} + \mathbf{F}_{T-1}^T \mathbf{v}_T$$

We have written optimal action value function  $Q(x_{T-1}, u_{T-1})$  only in terms of  $x_{T-1}, u_{T-1}$ !

$$\nabla_{\mathbf{u}_{T-1}} Q(\mathbf{x}_{T-1}, \mathbf{u}_{T-1}) = \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{x}_{T-1}} \mathbf{x}_{T-1} + \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}} \mathbf{u}_{T-1} + \mathbf{q}_{\mathbf{u}_{T-1}}^T = 0$$

$$\mathbf{u}_{T-1} = \mathbf{K}_{T-1} \mathbf{x}_{T-1} + \mathbf{k}_{T-1} \quad \mathbf{K}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}}^{-1} \mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{x}_{T-1}} \quad \mathbf{k}_{T-1} = -\mathbf{Q}_{\mathbf{u}_{T-1}, \mathbf{u}_{T-1}}^{-1} \mathbf{q}_{\mathbf{u}_{T-1}}$$

# Linear case: LQR

Backward recursion

for  $t = T$  to 1:

$$\mathbf{Q}_t = \mathbf{C}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{F}_t$$

$$\mathbf{q}_t = \mathbf{c}_t + \mathbf{F}_t^T \mathbf{V}_{t+1} \mathbf{f}_t + \mathbf{F}_t^T \mathbf{v}_{t+1}$$

$$Q(\mathbf{x}_t, \mathbf{u}_t) = \text{const} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{Q}_t \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix} + \begin{bmatrix} \mathbf{x}_t \\ \mathbf{u}_t \end{bmatrix}^T \mathbf{q}_t$$

$$\mathbf{u}_t \leftarrow \arg \min_{\mathbf{u}_t} Q(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

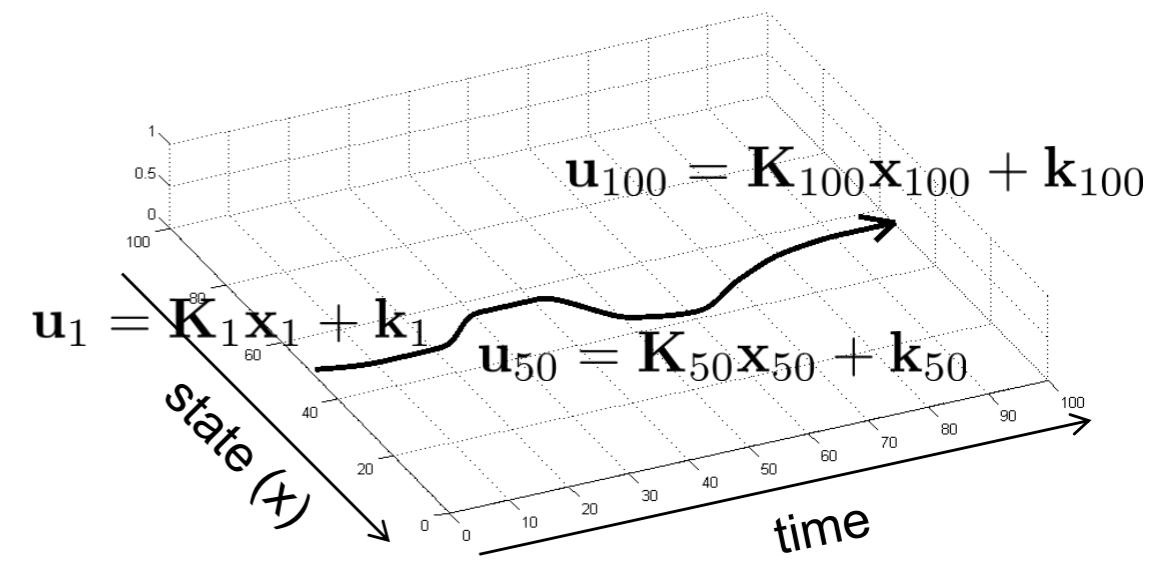
$$\mathbf{K}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t}$$

$$\mathbf{k}_t = -\mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t}^{-1} \mathbf{q}_{\mathbf{u}_t}$$

$$\mathbf{V}_t = \mathbf{Q}_{\mathbf{x}_t, \mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{K}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{x}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{K}_t$$

$$\mathbf{v}_t = \mathbf{q}_{\mathbf{x}_t} + \mathbf{Q}_{\mathbf{x}_t, \mathbf{u}_t} \mathbf{k}_t + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t} + \mathbf{K}_t^T \mathbf{Q}_{\mathbf{u}_t, \mathbf{u}_t} \mathbf{k}_t$$

$$V(\mathbf{x}_t) = \text{const} + \frac{1}{2} \mathbf{x}_t^T \mathbf{V}_t \mathbf{x}_t + \mathbf{x}_t^T \mathbf{v}_t$$



we know  $x_0$

Forward recursion

for  $t = 1$  to  $T$ :

$$\mathbf{u}_t = \mathbf{K}_t \mathbf{x}_t + \mathbf{k}_t$$

$$\mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$$

# Stochastic dynamics

Same control solution  $u_t = K_t x_t + k_t$  for stochastic but Gaussian dynamics:

$$f(x_t, u_t) = F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t$$

$$x_{t+1} \sim p(x_{t+1} | x_t, u_t)$$

$$p(x_{t+1} | x_t, u_t) = \mathcal{N}\left(F_t \begin{bmatrix} x_t \\ u_t \end{bmatrix} + f_t, \Sigma_t\right)$$

# Non-linear case: Use iterative approximations!

First order Taylor expansion for the dynamics around a trajectory  $\hat{x}_t, \hat{u}_t, t = 1 \dots T$ :

$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

Second order Taylor expansion for the cost around a trajectory  $\hat{x}_t, \hat{u}_t, t = 1 \dots T$ :

$$c(\mathbf{x}_t, \mathbf{u}_t) \approx c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix} + \frac{1}{2} \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}^T \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \mathbf{u}_t - \hat{\mathbf{u}}_t \end{bmatrix}$$

$$\bar{f}(\delta \mathbf{x}_t, \delta \mathbf{u}_t) = \underbrace{\mathbf{F}_t}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}$$

$$\bar{c}(\delta \mathbf{x}_t, \delta \mathbf{u}_t) = \frac{1}{2} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}^T \underbrace{\mathbf{C}_t}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix} + \underbrace{\begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}^T}_{\nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)} \mathbf{c}_t$$

$$\begin{aligned} \delta \mathbf{x}_t &= \mathbf{x}_t - \hat{\mathbf{x}}_t \\ \delta \mathbf{u}_t &= \mathbf{u}_t - \hat{\mathbf{u}}_t \end{aligned}$$

Now we can run LQR with dynamics  $\bar{f}$ , cost  $\bar{c}$ , state  $\delta \mathbf{x}_t$ , and action  $\delta \mathbf{u}_t$

# Iterative LQR (i-LQR)

Initialization: Given  $\hat{x}_0$ , pick a random control sequence  $\hat{u}_0 \dots \hat{u}_T$  and obtain corresponding state sequence  $\hat{x}_0 \dots \hat{x}_T$

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \quad \forall t$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \quad \forall t$$

$$\mathbf{C}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \quad \forall t$$

Run LQR backward pass on state  $\delta \mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$  and action  $\delta \mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t \quad \forall t$

Run forward pass with real nonlinear dynamics and  $u_t = \hat{u}_t + K_t(x_t - \hat{x}_T) + k_t \quad \forall t$

Update  $\hat{\mathbf{x}}_t$  and  $\hat{\mathbf{u}}_t$  based on states and actions in forward pass  $\forall t$

# Iterative LQR (i-LQR)

Initialization: Given  $\hat{x}_0$ , pick a random control sequence  $\hat{u}_0 \dots \hat{u}_T$  and obtain corresponding state sequence  $\hat{x}_0 \dots \hat{x}_T$

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \quad \forall t$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \quad \forall t$$

$$\mathbf{C}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \quad \forall t$$

Linear approximation around  $\hat{x}, \hat{u}$

Find  $\Delta u_t, t = 1 \dots T$  so

that  $\hat{u}_t + \Delta u_t$  minimizes the linear approximation

Run LQR backward pass on state  $\delta \mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$  and action  $\delta \mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t \quad \forall t$

Run forward pass with real nonlinear dynamics and  $u_t = \hat{u}_t + K_t(x_t - \hat{x}_T) + k_t \quad \forall t$

Update  $\hat{\mathbf{x}}_t$  and  $\hat{\mathbf{u}}_t$  based on states and actions in forward pass  $\forall t$

Go to the  $\hat{x}' = \hat{x} + \Delta x_t$  and  $\hat{u}' = \hat{u} + \Delta u_t$

# Iterative LQR (i-LQR)

i-LQR approximates Newton's method for solving:

$$\min_{\mathbf{u}_1, \dots, \mathbf{u}_T} c(\mathbf{x}_1, \mathbf{u}_1) + c(f(\mathbf{x}_1, \mathbf{u}_1), \mathbf{u}_2) + \dots + c(f(f(\dots)), \mathbf{u}_T)$$

Compare to Newton's method for computing  $\min_{\mathbf{x}} g(\mathbf{x})$ :

until convergence:

$$\mathbf{g} = \nabla_{\mathbf{x}} g(\hat{\mathbf{x}})$$

$$\mathbf{H} = \nabla_{\mathbf{x}}^2 g(\hat{\mathbf{x}})$$

$$\hat{\mathbf{x}} \leftarrow \arg \min_{\mathbf{x}} \frac{1}{2} (\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{H} (\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{g}^T (\mathbf{x} - \hat{\mathbf{x}})$$

Same as with Newton method, since the original objective is non-convex, the optimization can get stuck in local minima.

What are we missing to be exact Newton's method?

# Differential Dynamic Programming

Second order approximation for the dynamics:

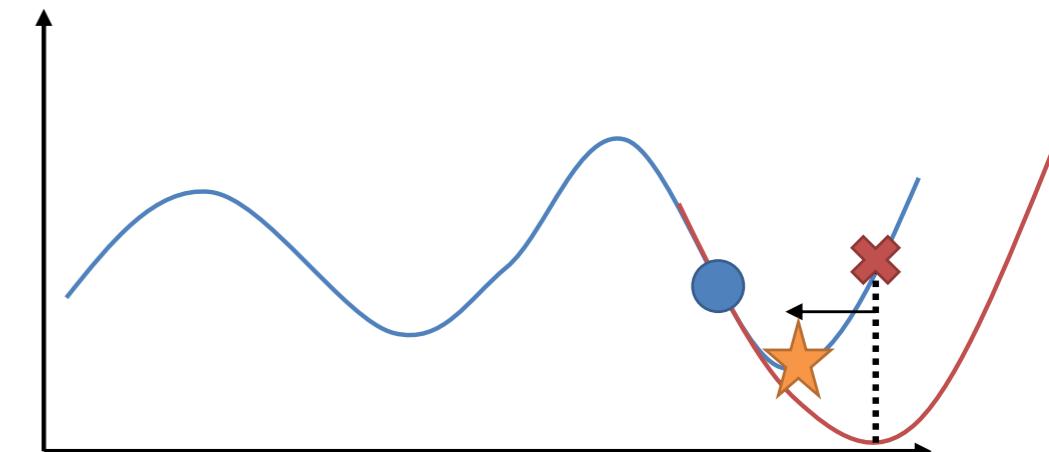
$$f(\mathbf{x}_t, \mathbf{u}_t) \approx f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) + \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix} + \frac{1}{2} \left( \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t) \cdot \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix} \right) \begin{bmatrix} \delta \mathbf{x}_t \\ \delta \mathbf{u}_t \end{bmatrix}$$

The resulting method is called differential dynamic programming.

# Nonlinear case: DDP/iterative LQR

The quadratic approximation is invalid too far away from the reference trajectory

$$\hat{\mathbf{x}} \leftarrow \arg \min_{\mathbf{x}} \frac{1}{2} (\mathbf{x} - \hat{\mathbf{x}})^T \mathbf{H} (\mathbf{x} - \hat{\mathbf{x}}) + \mathbf{g}^T (\mathbf{x} - \hat{\mathbf{x}})$$



Instead of finding the argmin i do a line search

until convergence:

$$\mathbf{F}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} f(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{c}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t} c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

$$\mathbf{C}_t = \nabla_{\mathbf{x}_t, \mathbf{u}_t}^2 c(\hat{\mathbf{x}}_t, \hat{\mathbf{u}}_t)$$

line search for \alpha

Run LQR backward pass on state  $\delta \mathbf{x}_t = \mathbf{x}_t - \hat{\mathbf{x}}_t$  and action  $\delta \mathbf{u}_t = \mathbf{u}_t - \hat{\mathbf{u}}_t$

Run forward pass with real nonlinear dynamics and  $u_t = \hat{u}_t + K_t(x_t - \hat{x}_T) + \alpha k_t$

Update  $\hat{\mathbf{x}}_t$  and  $\hat{\mathbf{u}}_t$  based on states and actions in forward pass

More principled ways of doing this for stochastic policies in the next lecture

# Nonlinear case: DDP/iterative LQR

- So far we have been planning (e.g. 100 steps) and then we close our eyes and hope our modeling was accurate enough..
- At convergence of iLQR and DDP, we end up with linearization around the (state, input) trajectory the algorithm converged to.
- In practice: the system could not be on this trajectory due to perturbations / initial state being off / dynamics model being off / ...
- Can we handle such noise better?

# Model Predictive Control

- Yes! If we close the loop! Model predictive control!
- Solution: at time  $t$  when asked to generate control input  $u_t$ , we could re-solve the control problem using iLQR or DDP over the time steps  $t$  through  $T$

every time step:

observe the state  $\mathbf{x}_t$

use iLQR to plan  $\mathbf{u}_t, \dots, \mathbf{u}_T$  to minimize  $\sum_{t'=t}^{t+T} c(\mathbf{x}_{t'}, \mathbf{u}_{t'})$

execute action  $\mathbf{u}_t$ , discard  $\mathbf{u}_{t+1}, \dots, \mathbf{u}_{t+T}$

- Re-planning entire trajectory is often impractical -> in practice: replay over horizon  $H$  (receding horizon control)

# i-LQR: When it works

Synthesis and stabilization of complex behaviors with online trajectory optimization

Yuval Tassa, Tom Erez and Emo Todorov

Movement Control Laboratory  
University of Washington

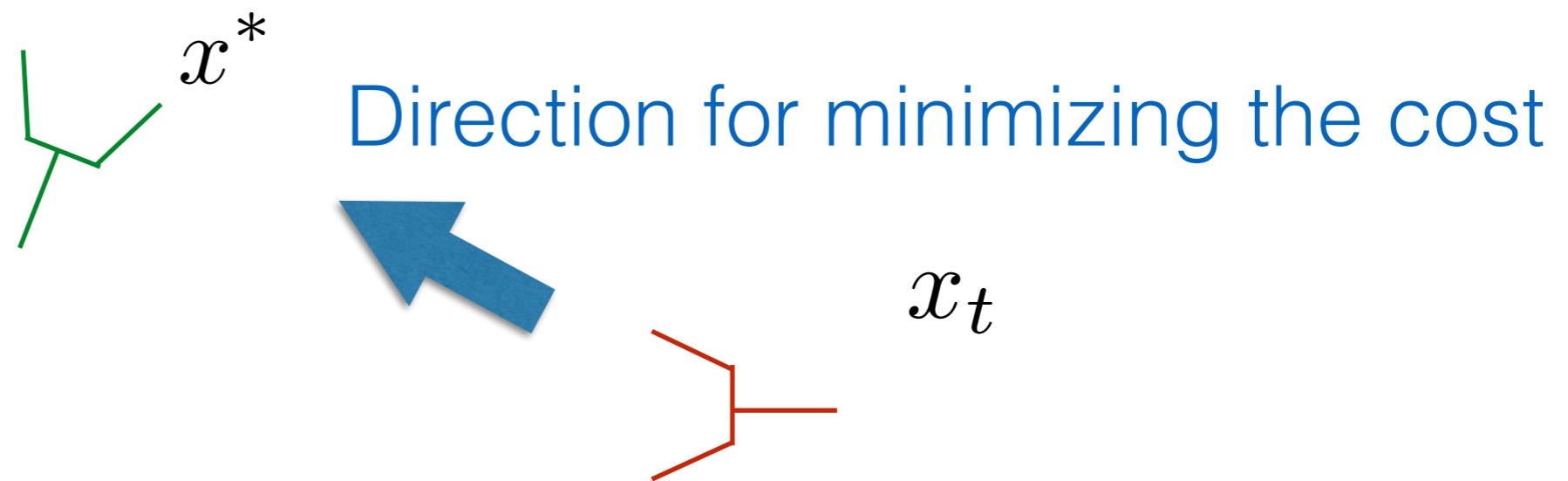
IROS 2012

---

Synthesis and stabilization of complex behaviors with online trajectory optimization, Tassa, Erez, Todorov, IROS 2012

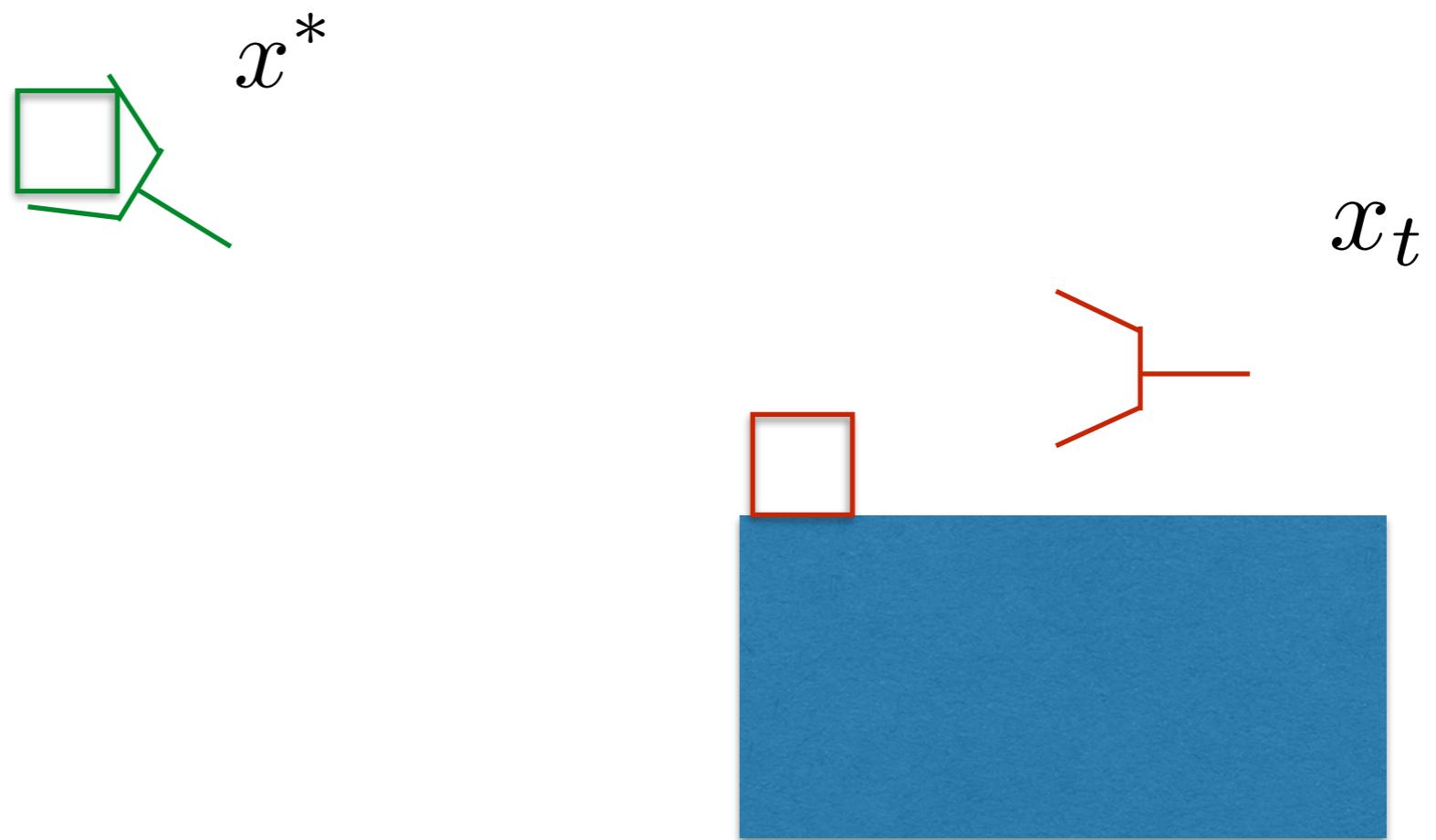
# i-LQR: When it works

Cost:  $\|x_t - x^*\|$



# i-LQR: When it doesn't work

Cost:  $\|x_t - x^*\|$



Due to discontinuities of contact, the local search fails! Solution?

Initialize using a human demonstration instead of random!





# Learning Dynamics

- So far we assumed we knew how the world works:  $f(\mathbf{x}_t, \mathbf{u}_t) = \mathbf{x}_{t+1}$
- We used discrete or continuous searches (e.g., MCTS(samples) and i-LQR(derivatives)) to **look-ahead**. We got good results, better than model-free RL. Dynamics help.
- However knowing the transition model (other than for rigid objects) is unrealistic! In fact, we do not have good physics simulators easily accessible for even basic things like turning a wheel. A huge part (deformable objects, object interactions, etc ) we do not know how to model well.
- Even for rigid objects, where we know the equations of motion, we do not know the coefficients, e.g., friction, mass etc.
- Thus: instead of assuming them known, **we need to learn dynamics.**

# Learning Dynamics

Newtonian Physics equations

VS

general parametric form (no priors from Physics knowledge)



**System identification:** when we assume the dynamics equations given and only have *few* unknown parameters



Much easier to learn but suffers from under-modeling, bad models



**Neural networks:** *tons* of unknown parameters



Very flexible, very hard to get it to generalize

# Learning Dynamics: Regression

Search with learnt dynamics:

1. Run base policy  $\pi_0(u_t|x_t)$ (e.g., random policy) to collect  $\mathcal{D} = \{(x, u, x')_i\}$
2. Learn dynamics model  $f(x, u)$ to minimize  $\sum_i \|f(x_i, u_i) - x'_i\|^2$
3. Plan under the learnt dynamics to choose actions (e.g., MCTS or i-LQR)

# Learning Dynamics: Regression

Search with learnt dynamics:

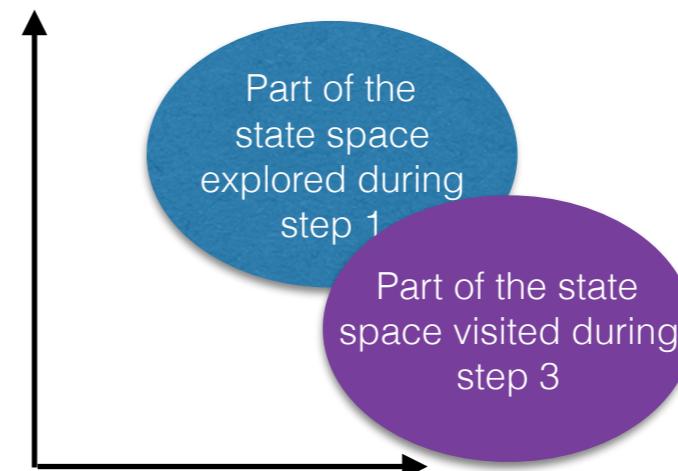
1. Run base policy  $\pi_0(u_t|x_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(x, u, x')_i\}$
2. Learn dynamics model  $f(x, u)$  to minimize  $\sum_i \|f(x_i, u_i) - x'_i\|^2$
3. Plan under the learnt dynamics to choose actions (e.g., MCTS or i-LQR)

Let's apply it:

1. A robot randomly interacts (pokes objects) and collects data ( $x, u, x'$ )
2. Fit dynamics model
3. Plan actions to accomplish a particular goal, e.g., push an object as far away as possible



What goes wrong:



Part of the state space explored during step 1

Part of the state space visited during step 3

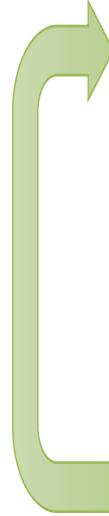
state distribution mismatch between dynamics learning and policy execution

$$p_{\pi_f}(\mathbf{x}_t) \neq p_{\pi_0}(\mathbf{x}_t)$$

# Learning Dynamics: DAGGER

1. Run base policy  $\pi_0(u_t|x_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(x, u, x')_i\}$
2. Learn dynamics model  $f(x, u)$  to minimize  $\sum_i \|f(x_i, u_i) - x'_i\|^2$
3. Plan under the learnt dynamics to choose actions (e.g., MCTS or i-LQR)
4. Execute those actions and add the resulting data  $\{(x, u, x')_j\}$  to  $\mathcal{D}$

# Learning Dynamics: DAGGER

- 
1. Run base policy  $\pi_0(u_t|x_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(x, u, x')_i\}$
  2. Learn dynamics model  $f(x, u)$  to minimize  $\sum_i \|f(x_i, u_i) - x'_i\|^2$
  3. Plan under the learnt dynamics to choose actions (e.g., MCTS or i-LQR)
  4. Execute those actions and add the resulting data  $\{(x, u, x')_j\}$  to  $\mathcal{D}$

If the action sequence is long the model errors accumulate in time.

It is impossible our dynamic model to be perfect and tolerate long chaining.

It is also not biologically plausible, e.g., humans are very bad at predicting ball collisions accurately.

# Learning Dynamics: MPC

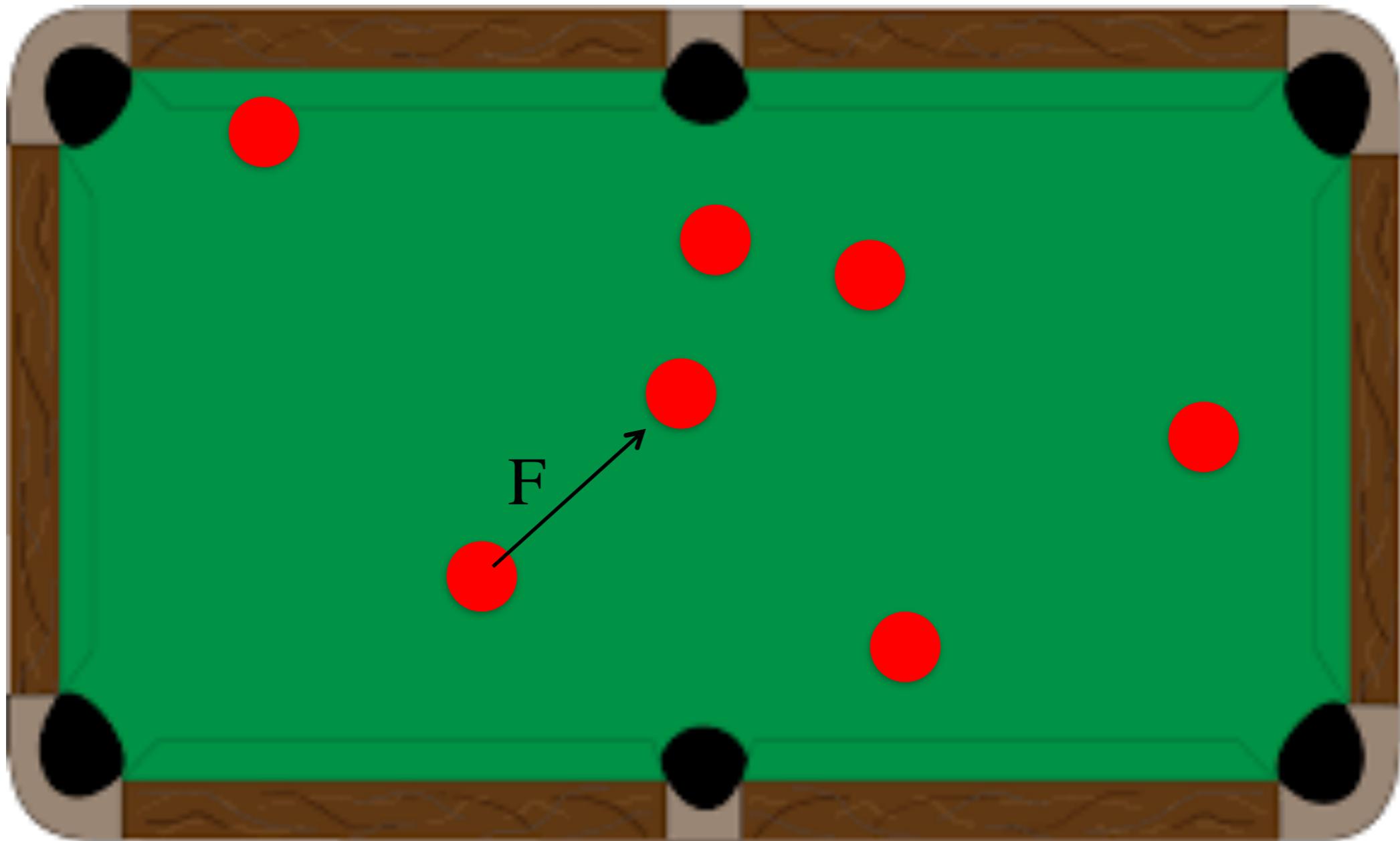
1. Run base policy  $\pi_0(u_t|x_t)$  (e.g., random policy) to collect  $\mathcal{D} = \{(x, u, x')_i\}$
2. Learn dynamics model  $f(x, u)$  to minimize  $\sum_i \|f(x_i, u_i) - x'_i\|^2$
3. Plan under the learnt dynamics to choose actions (e.g., MCTS or i-LQR)
4. Execute the first planned action, observe resulting state  $x'$  (MPC)
5. Append  $(x, u, x')$  to dataset  $\mathcal{D}$

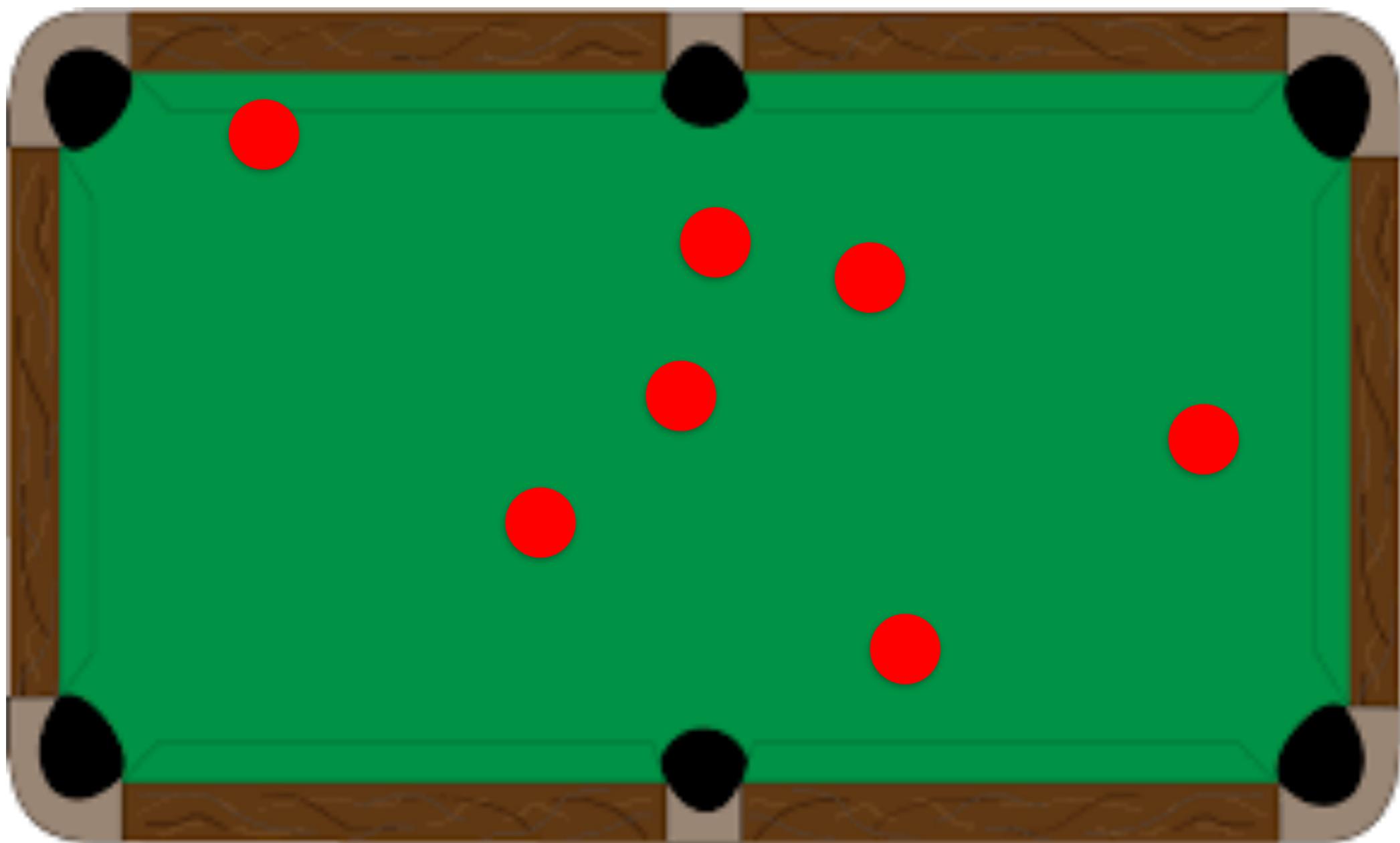
every N steps

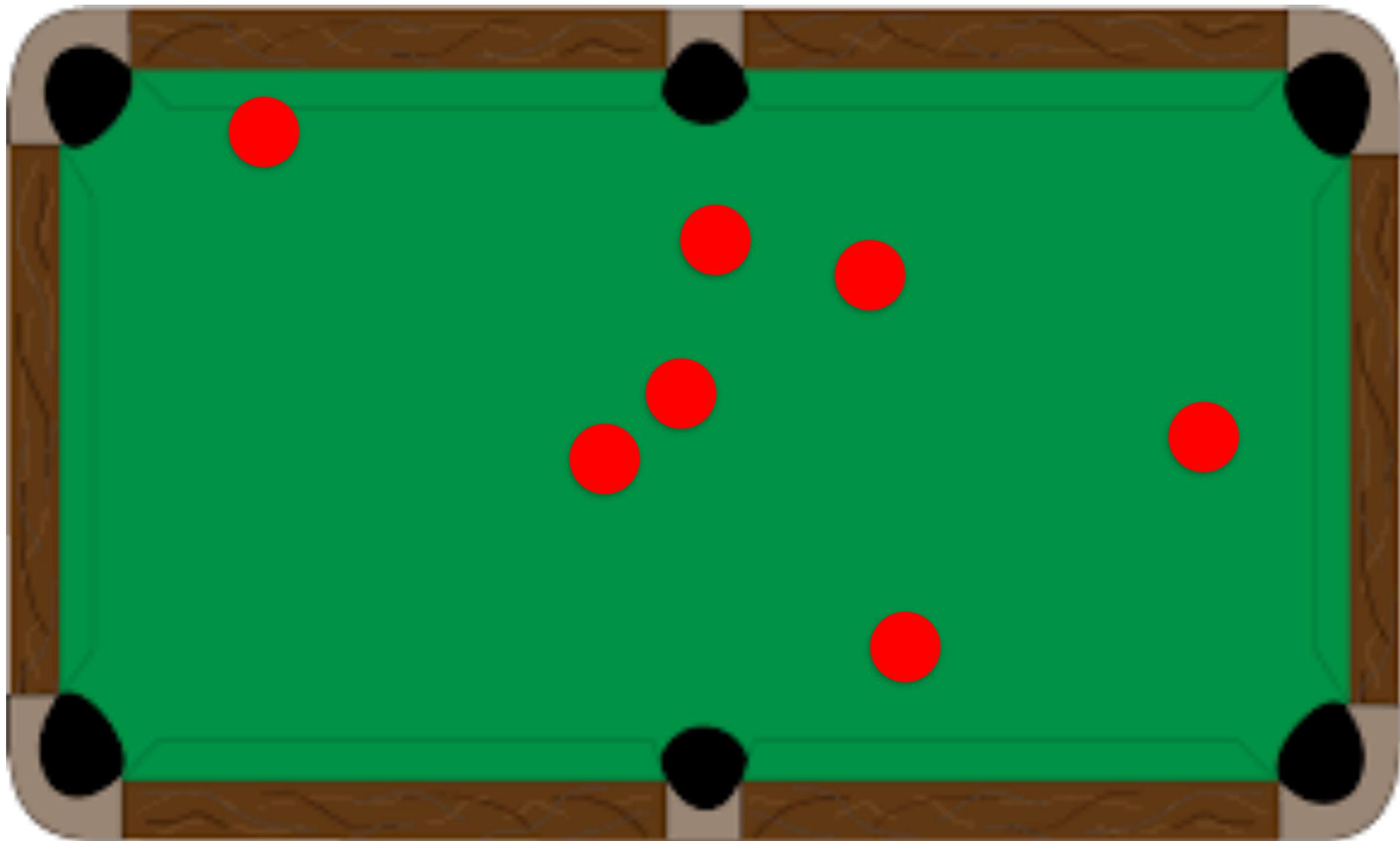
# Learning Dynamics (Summary)

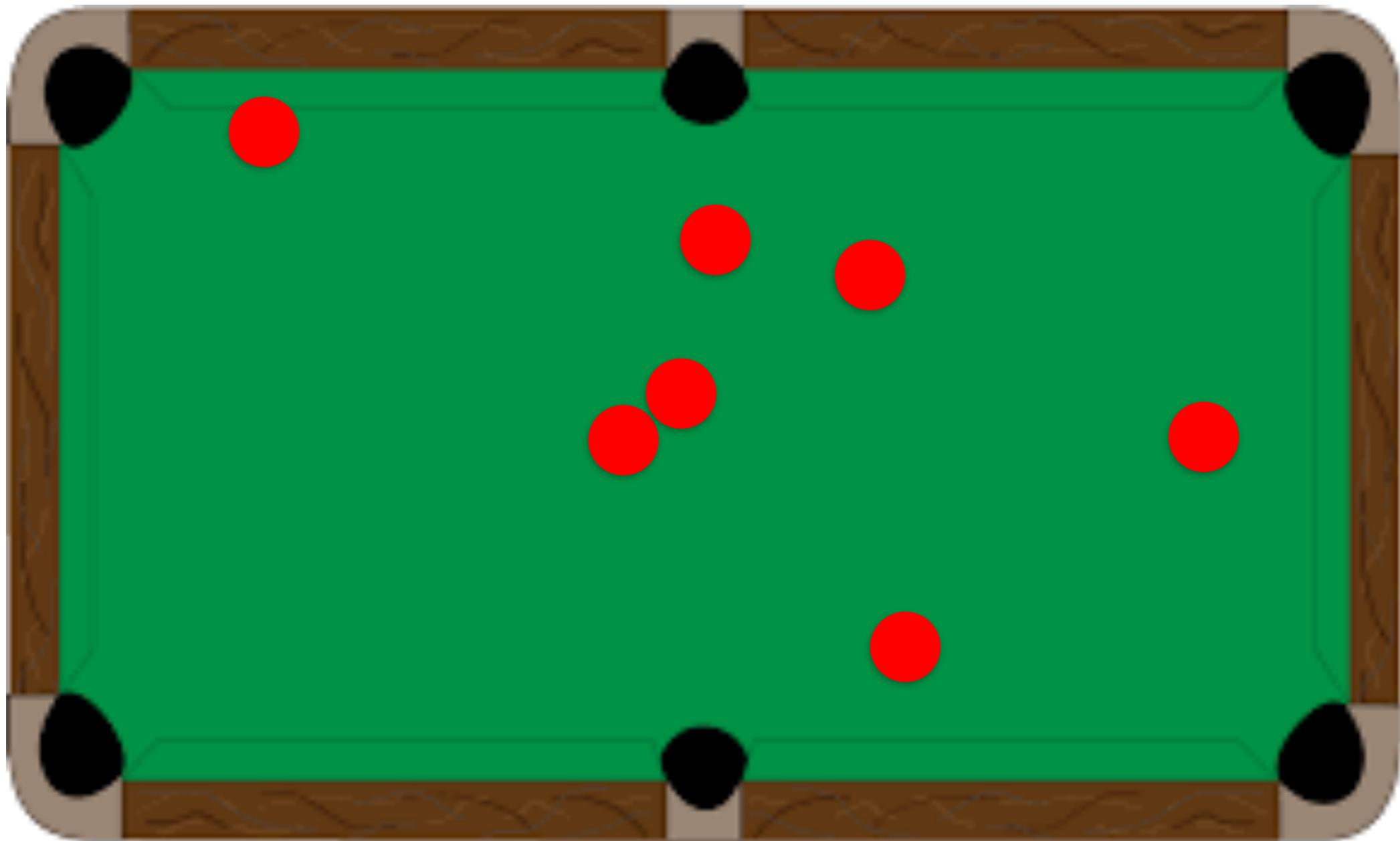
- Regression problem: collect random samples, train dynamics, plan
  - Pro: simple, no iterative procedure
  - Con: distribution mismatch problem
- DAGGER: iteratively collect data, replan, collect data
  - Pro: simple, solves distribution mismatch
  - Con: open loop plan might perform poorly, esp. in stochastic domains
- MPC: iteratively collect data using MPC (replan at each step)
  - Pro: robust to small model errors
  - Con: computationally expensive, but have a planning algorithm available

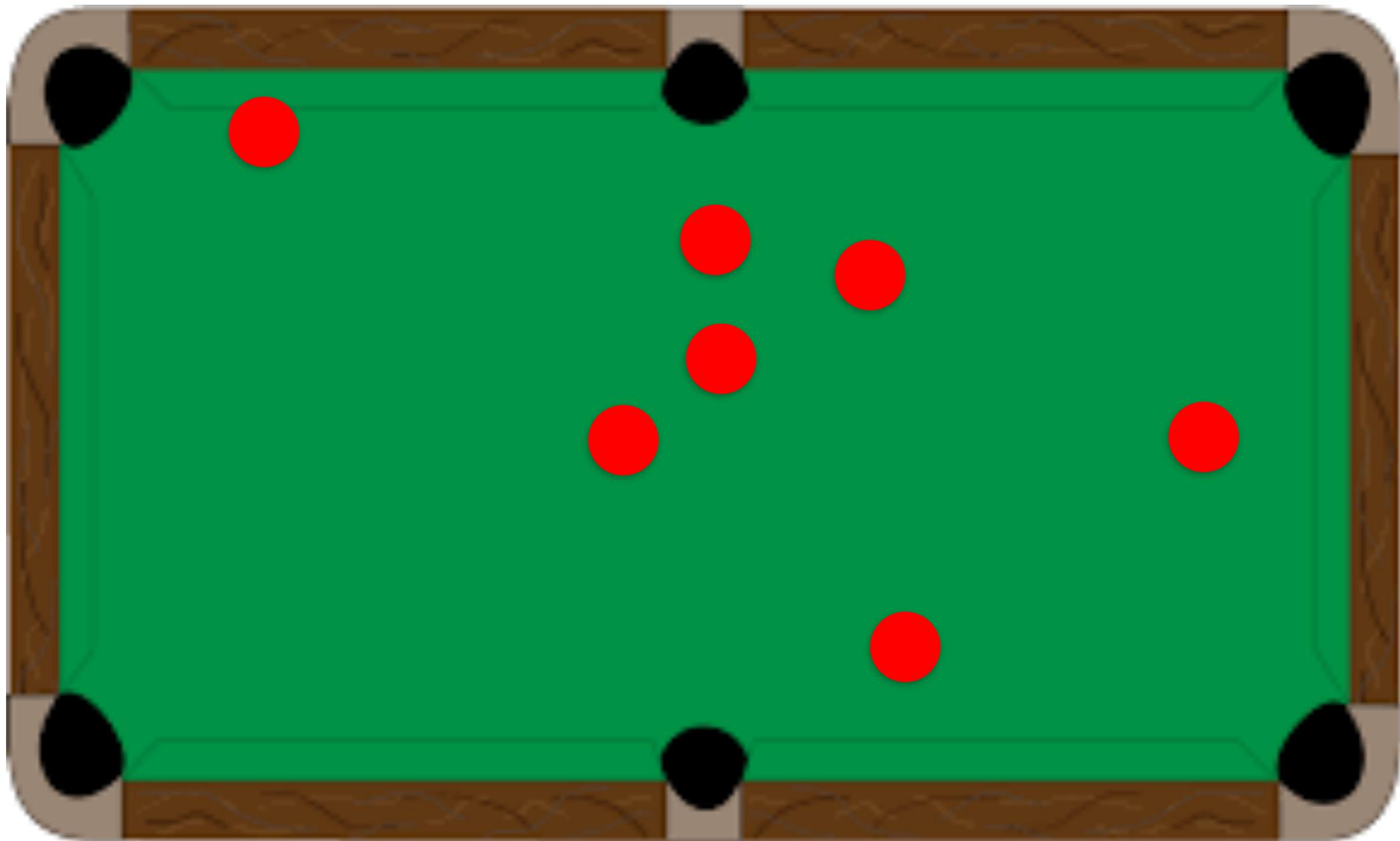
# Examples of Learning Dynamics

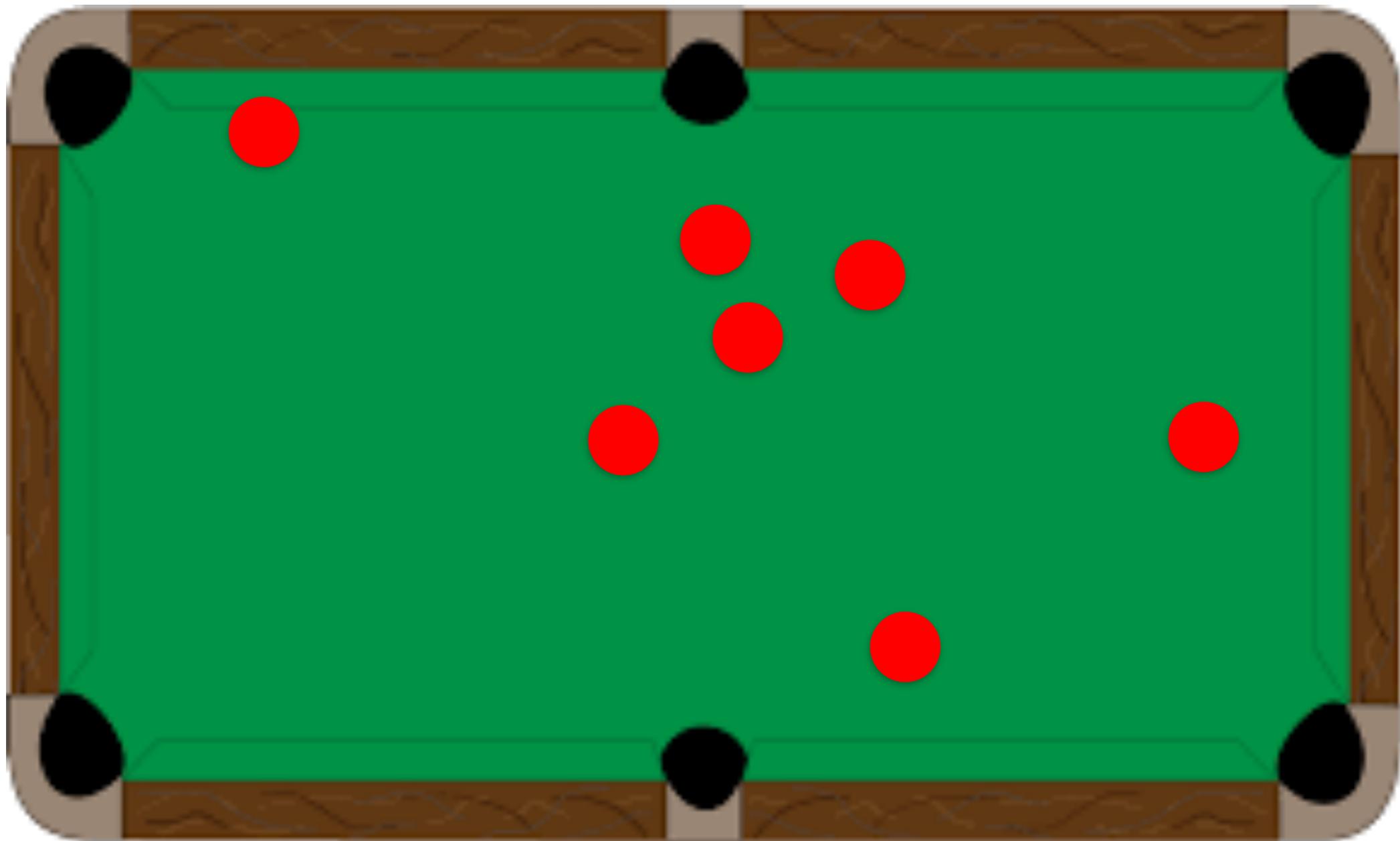












# Learning Action-Conditioned Dynamics

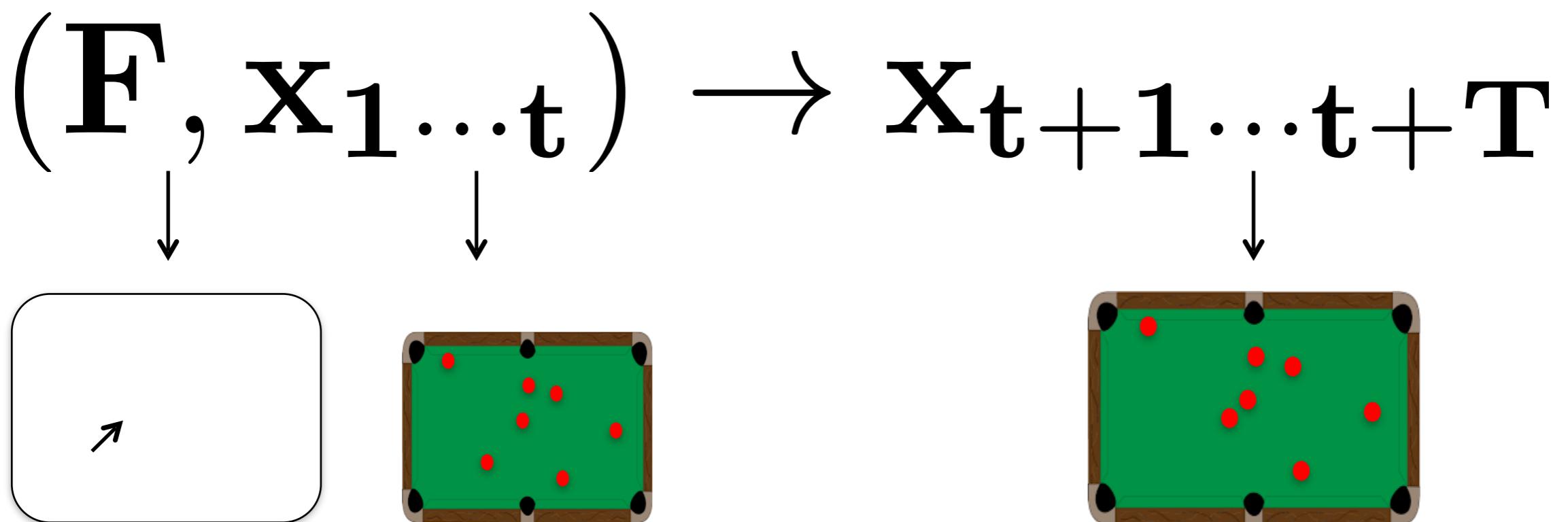
➤ Newtonian Physics  $F, I, \alpha, m, k$

Galileo: Perceiving Physical Object Properties by Integrating a Physics Engine with Deep Learning, Wu et al.

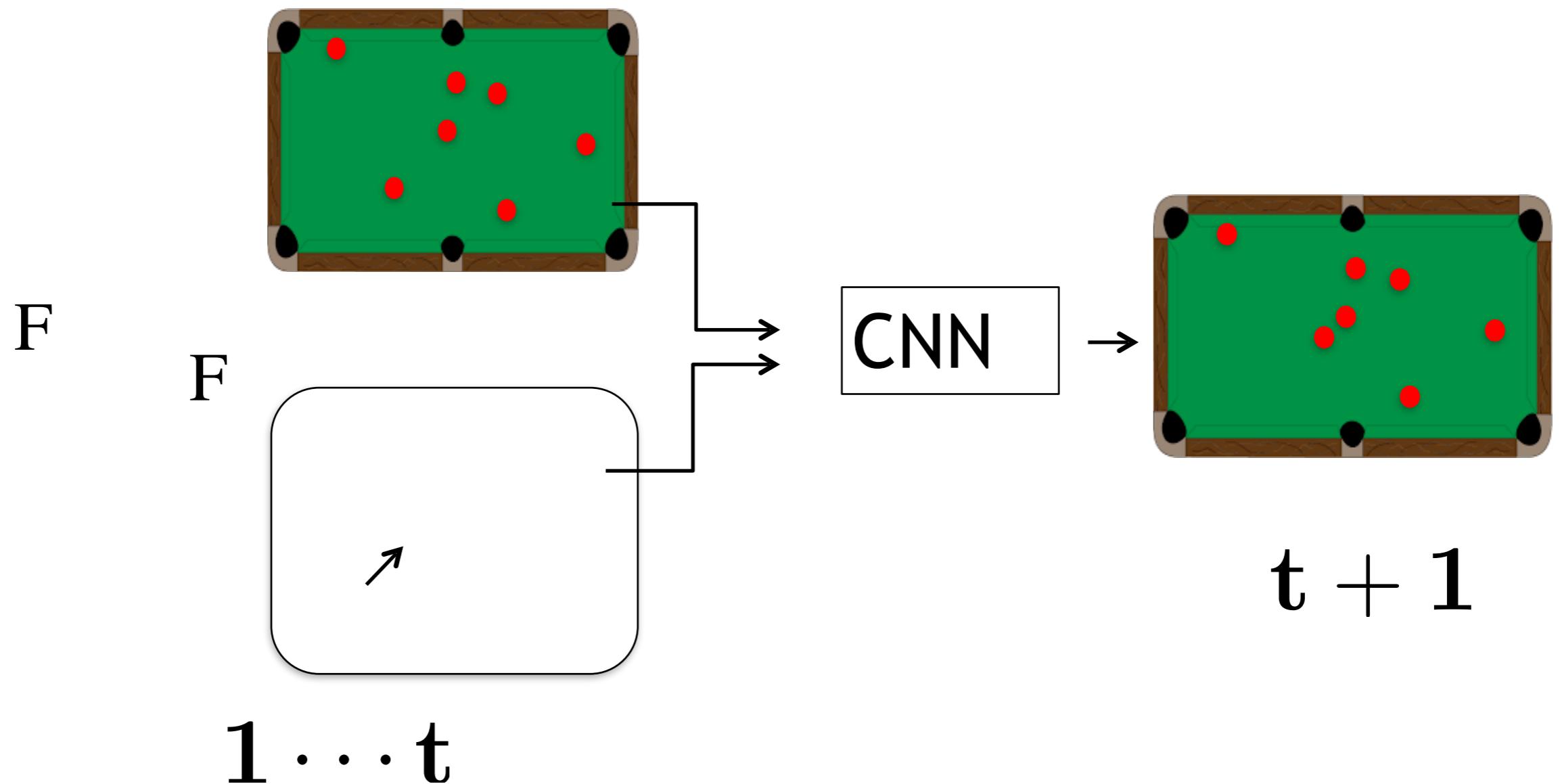
Newtonian Image Understanding: Unfolding the Dynamics of Objects in Static Images, Mottaghi et al.

➤ Predictive Physics  $(F, x_1 \dots t) \rightarrow y_{t+1} \dots t+T$

# Learning Action-Conditioned Dynamics



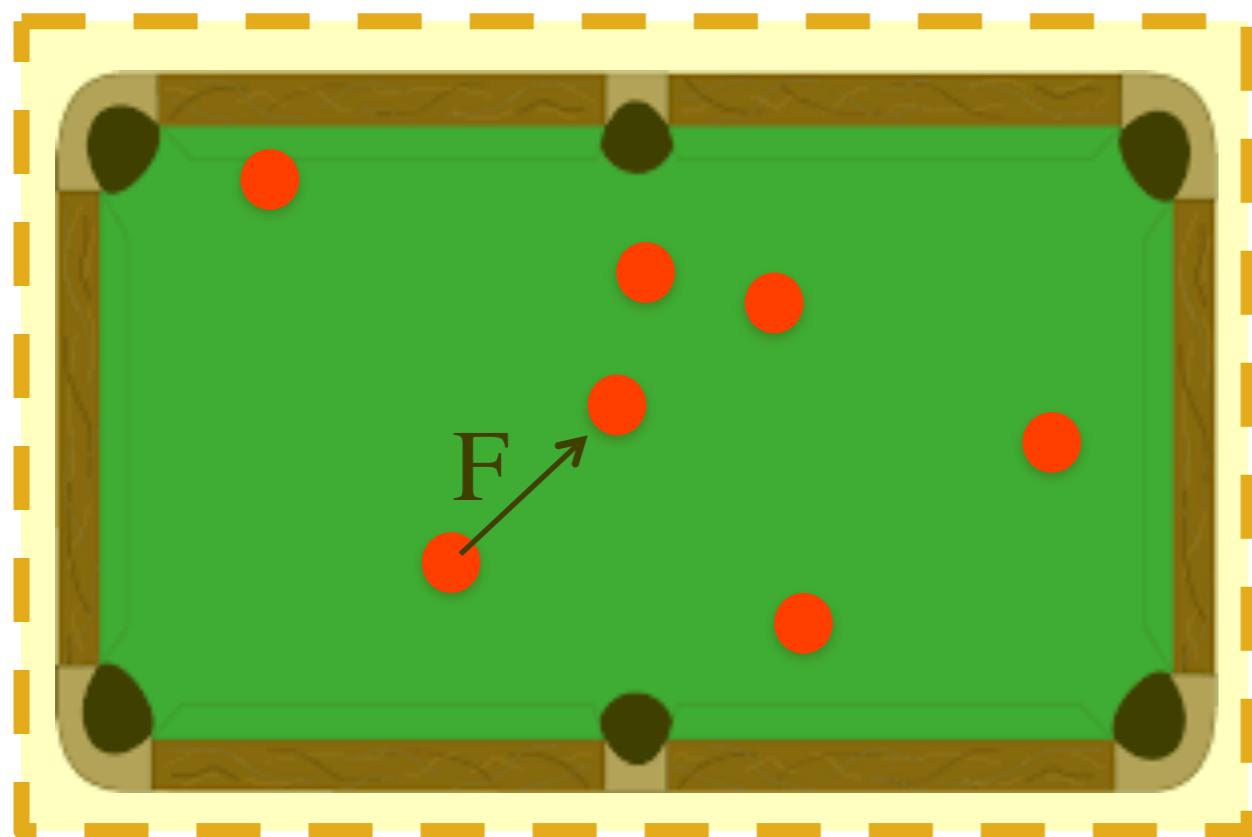
# Learning Action-Conditioned Dynamics



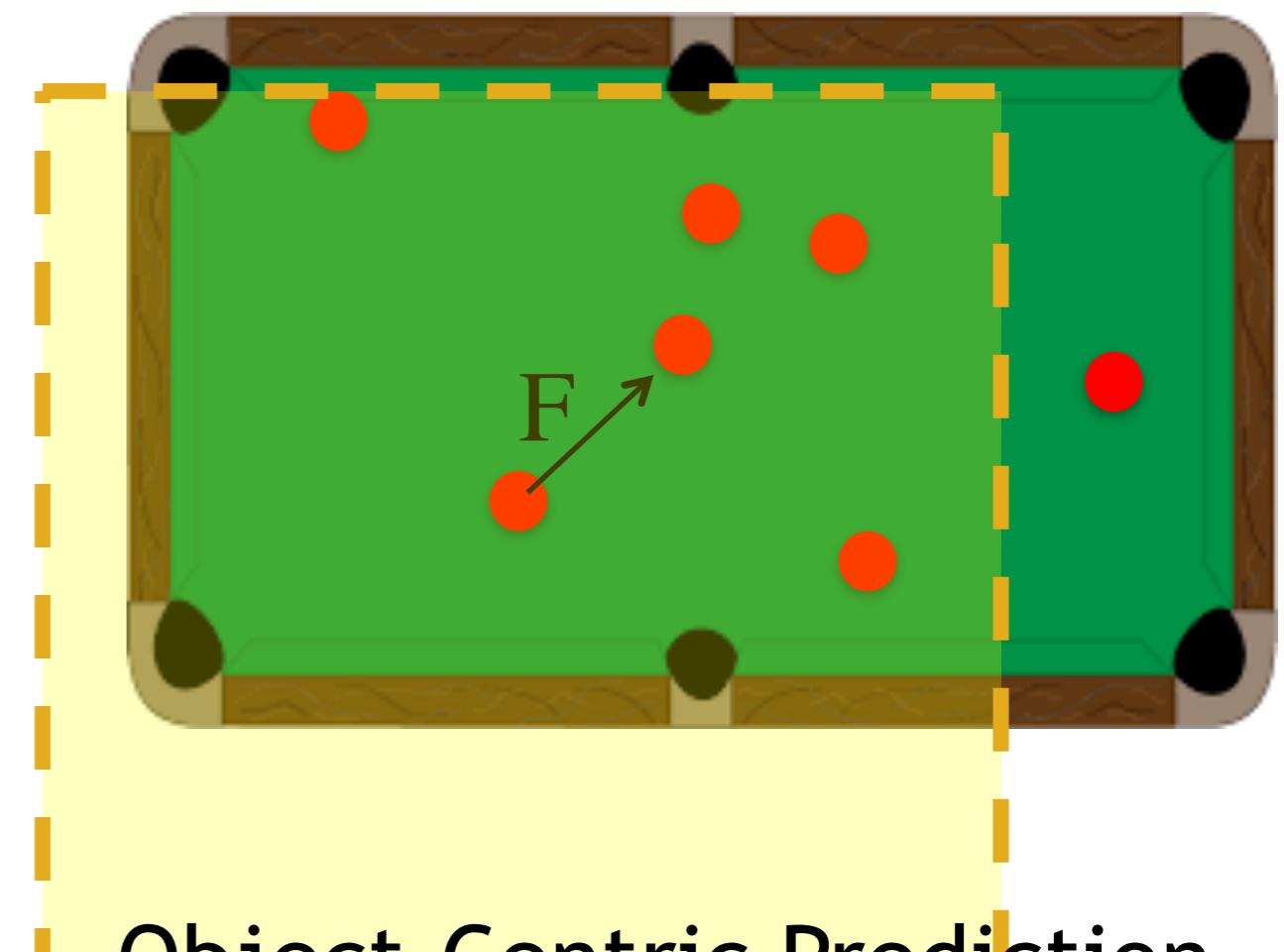
Problems:

- Forces are translation invariant!

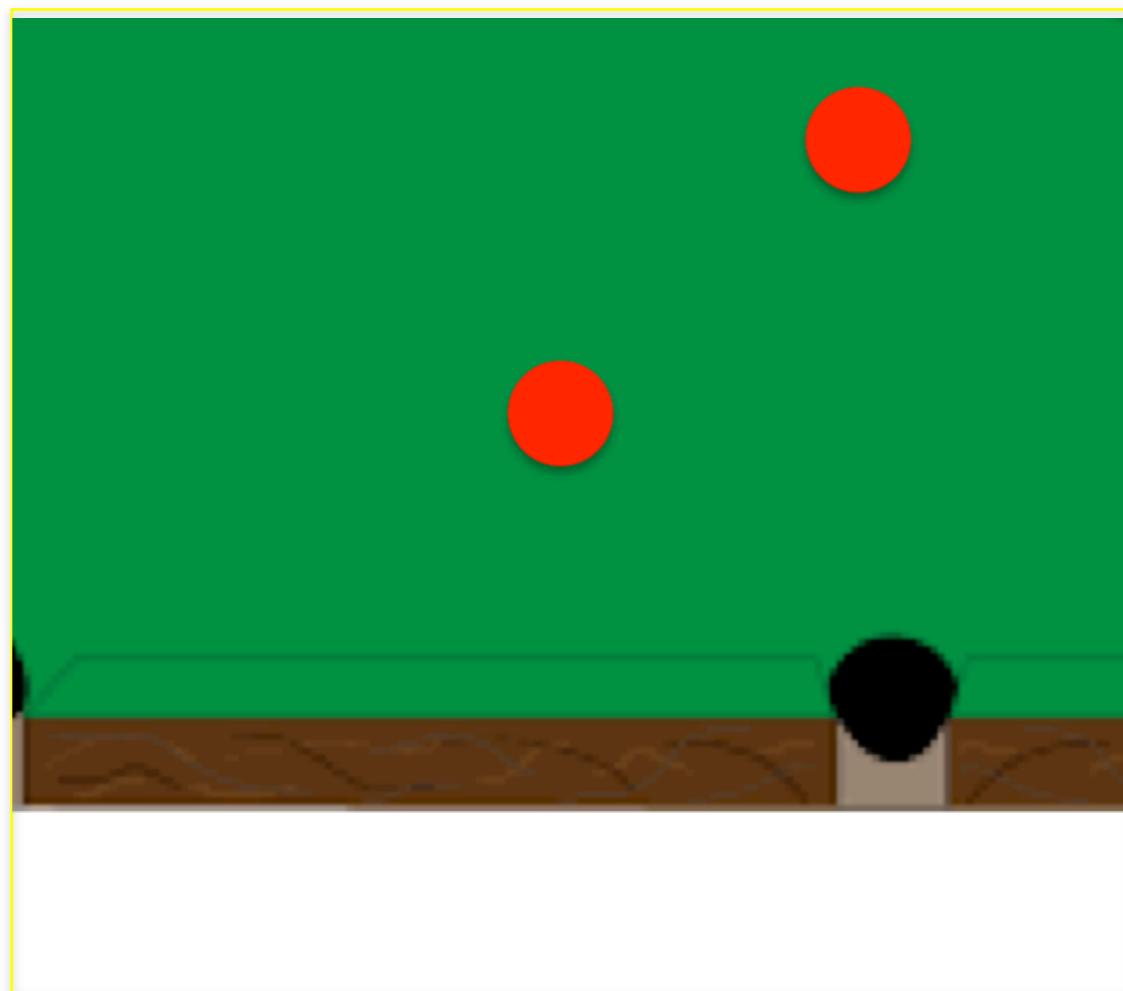
# Object-centric prediction

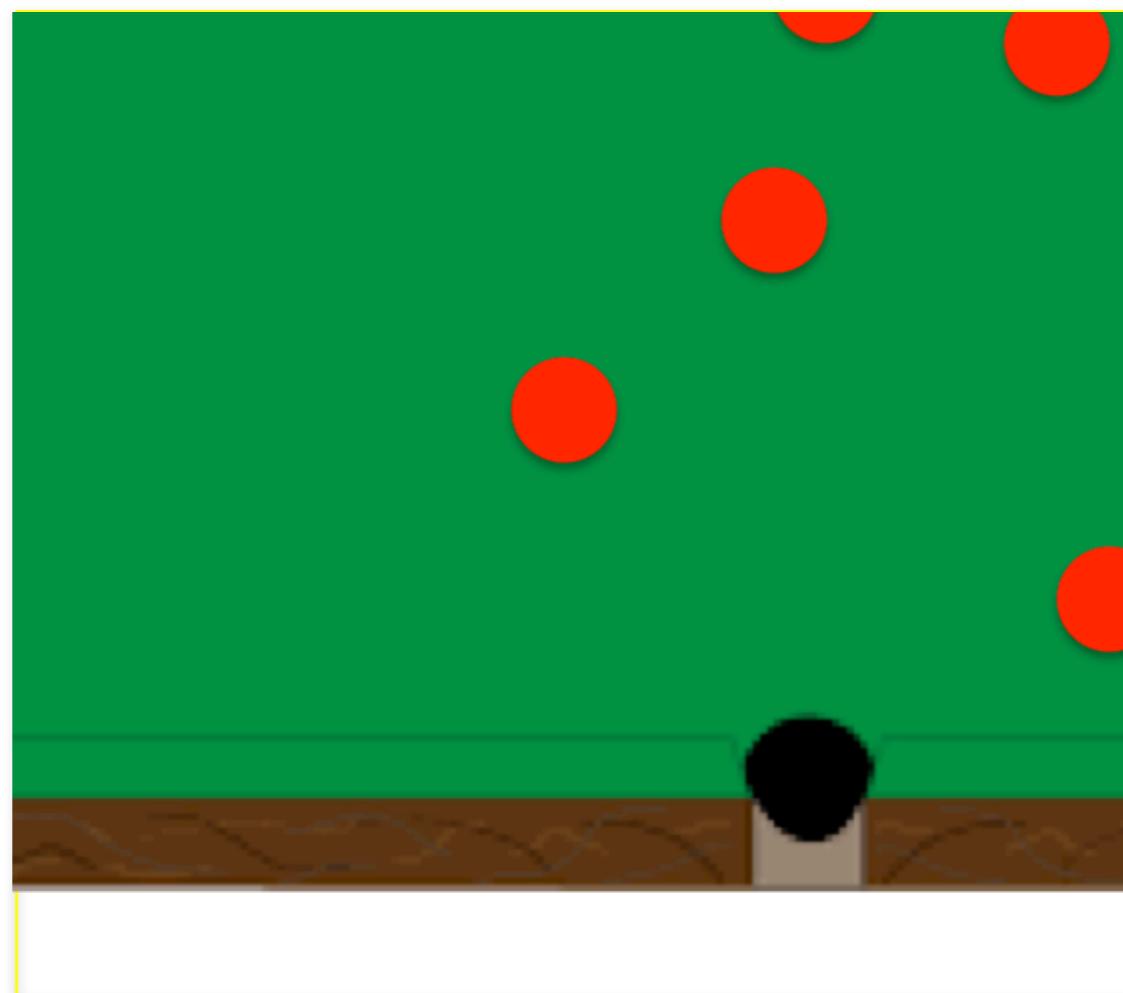


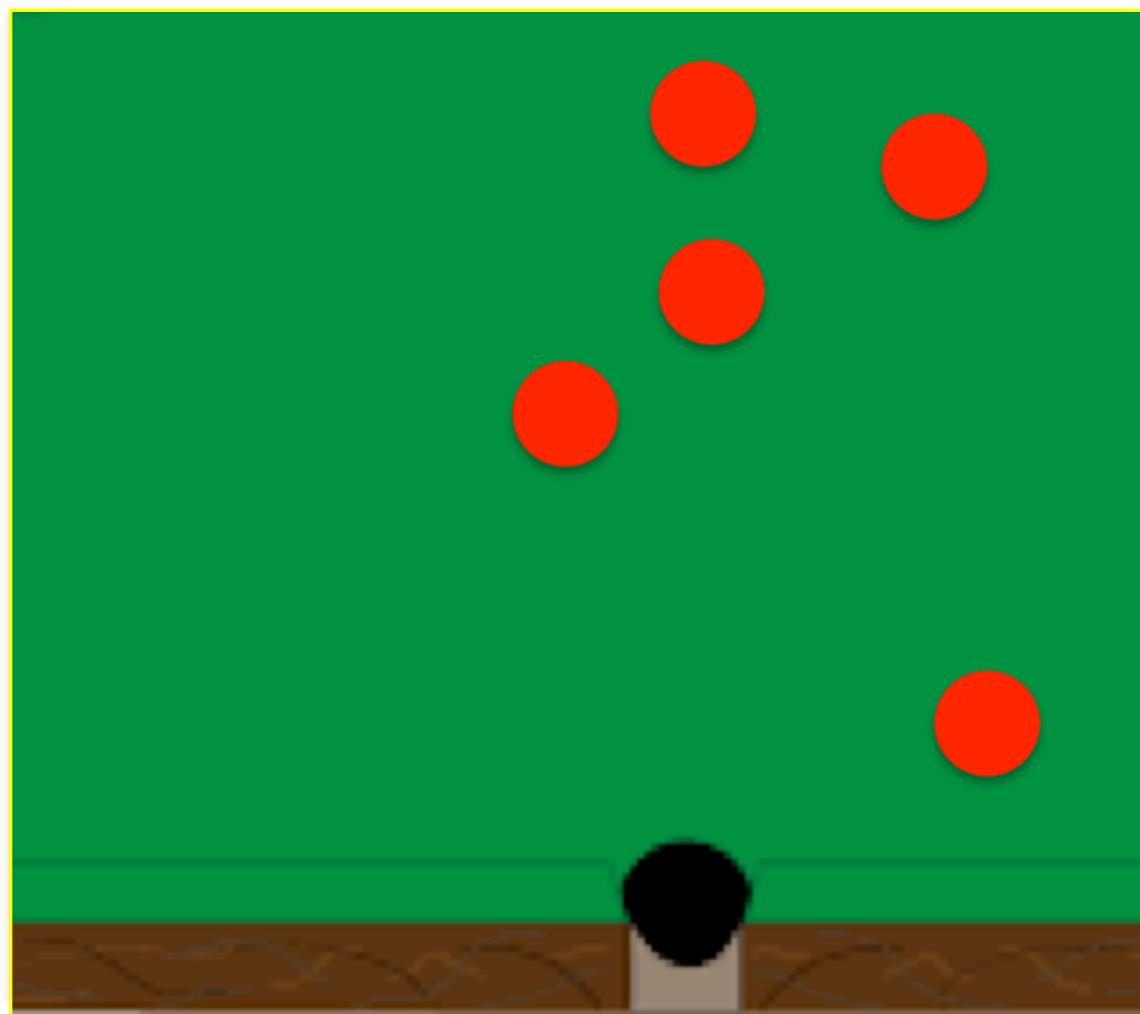
World-Centric Prediction

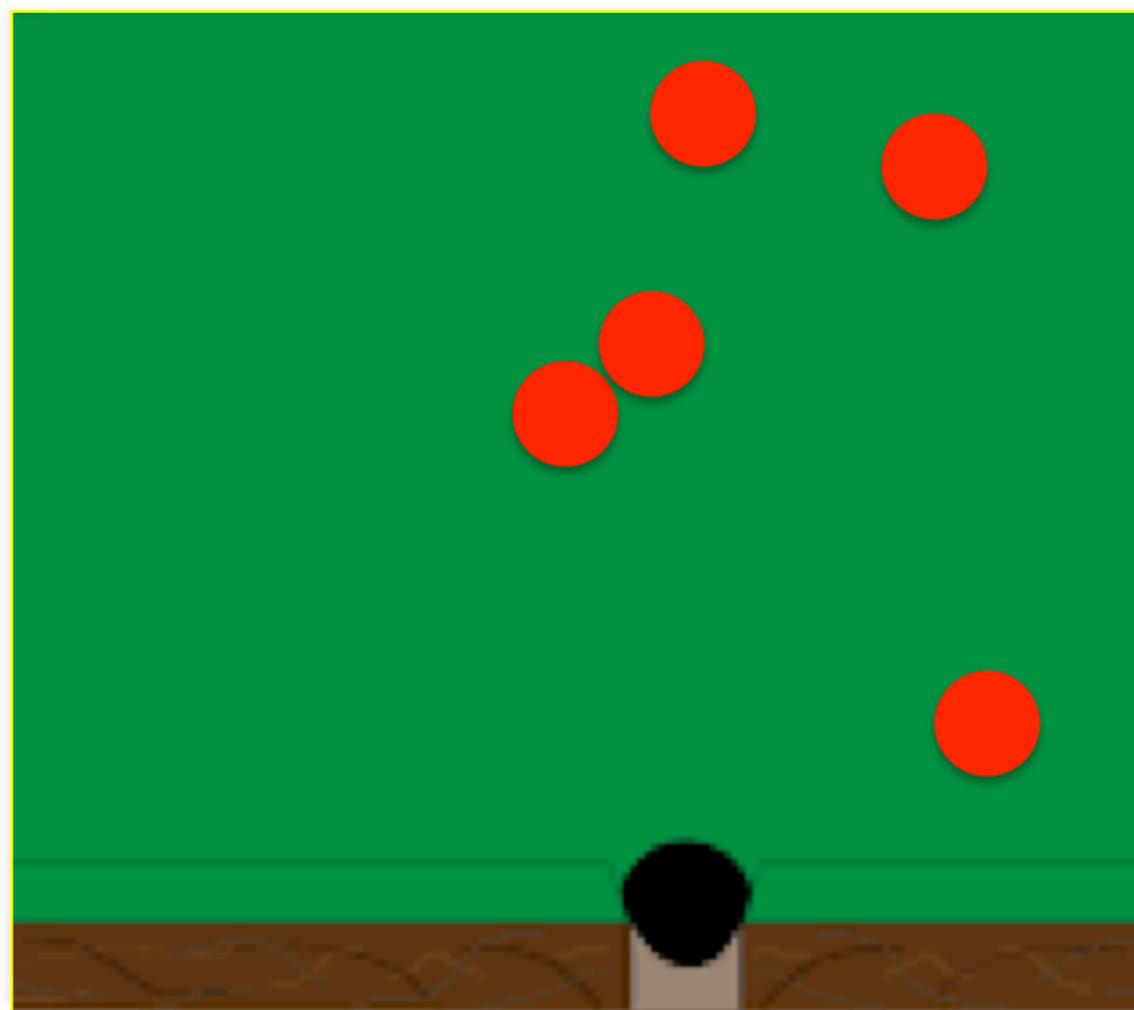


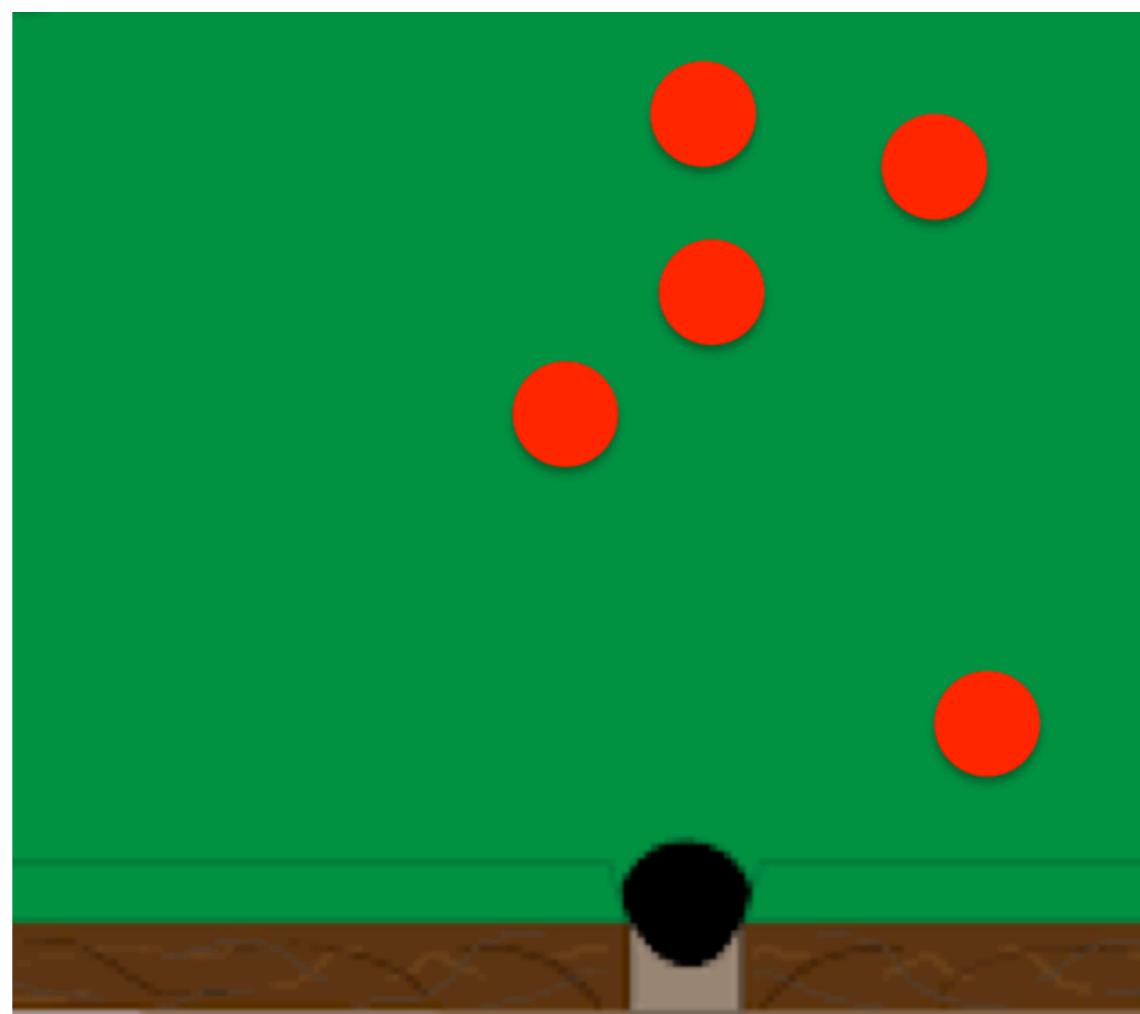
Object-Centric Prediction

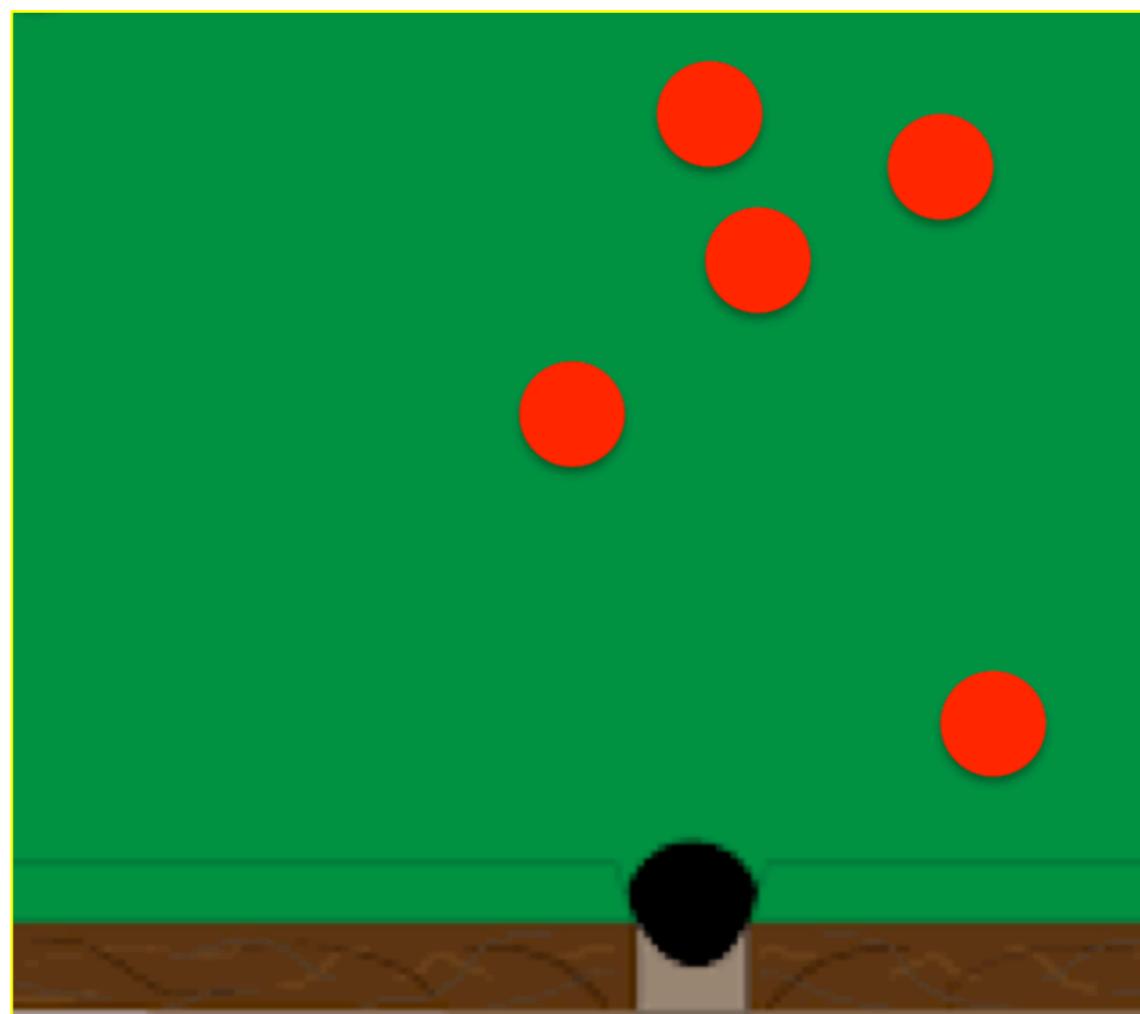


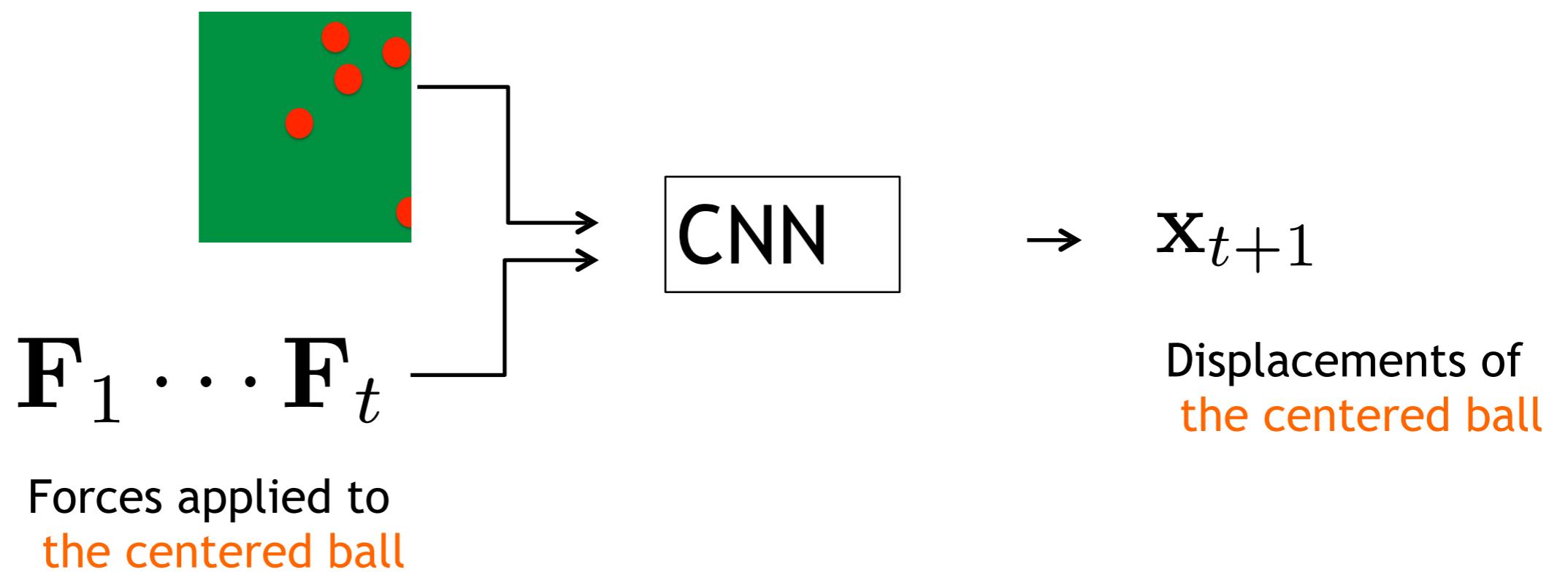




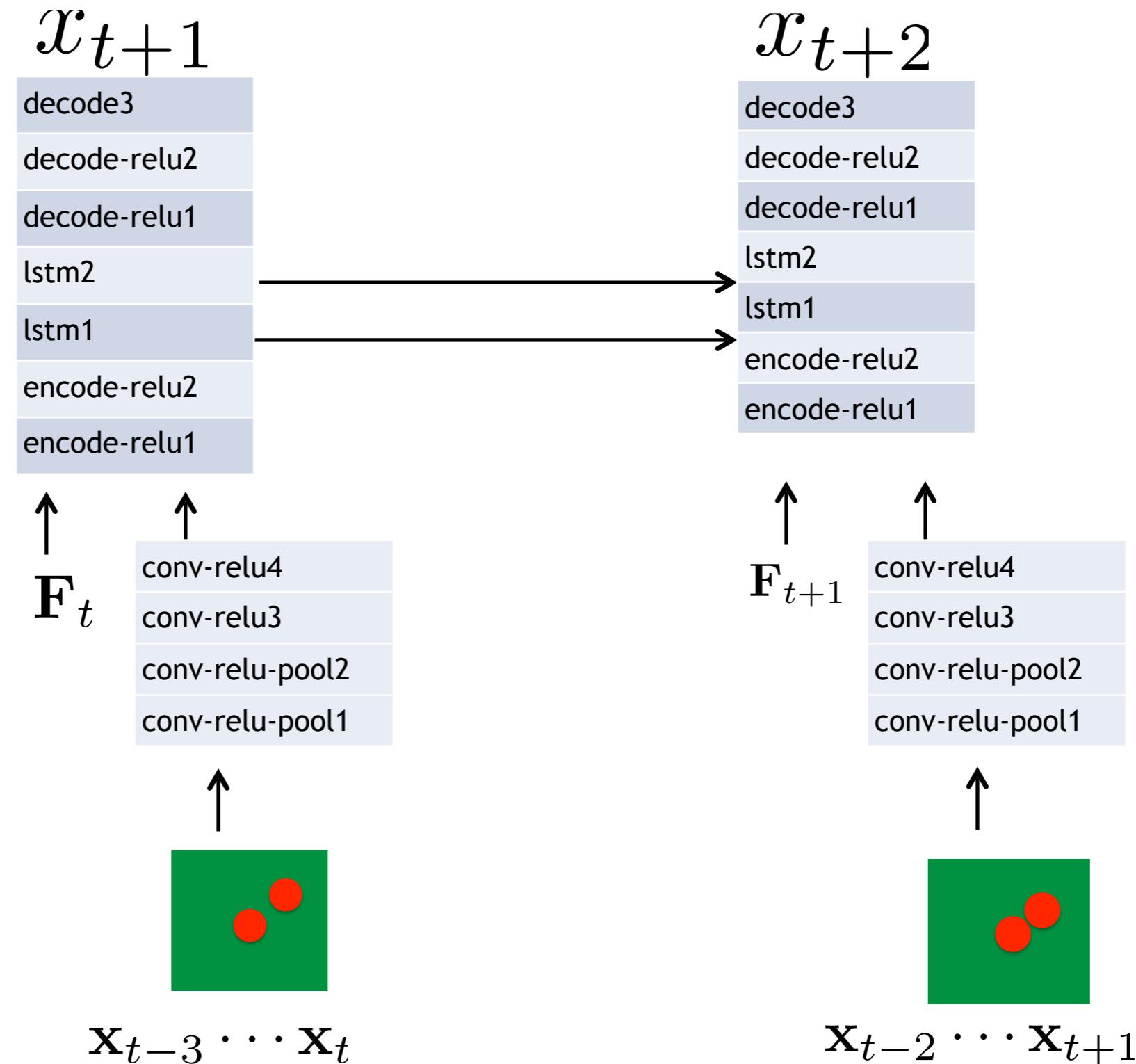




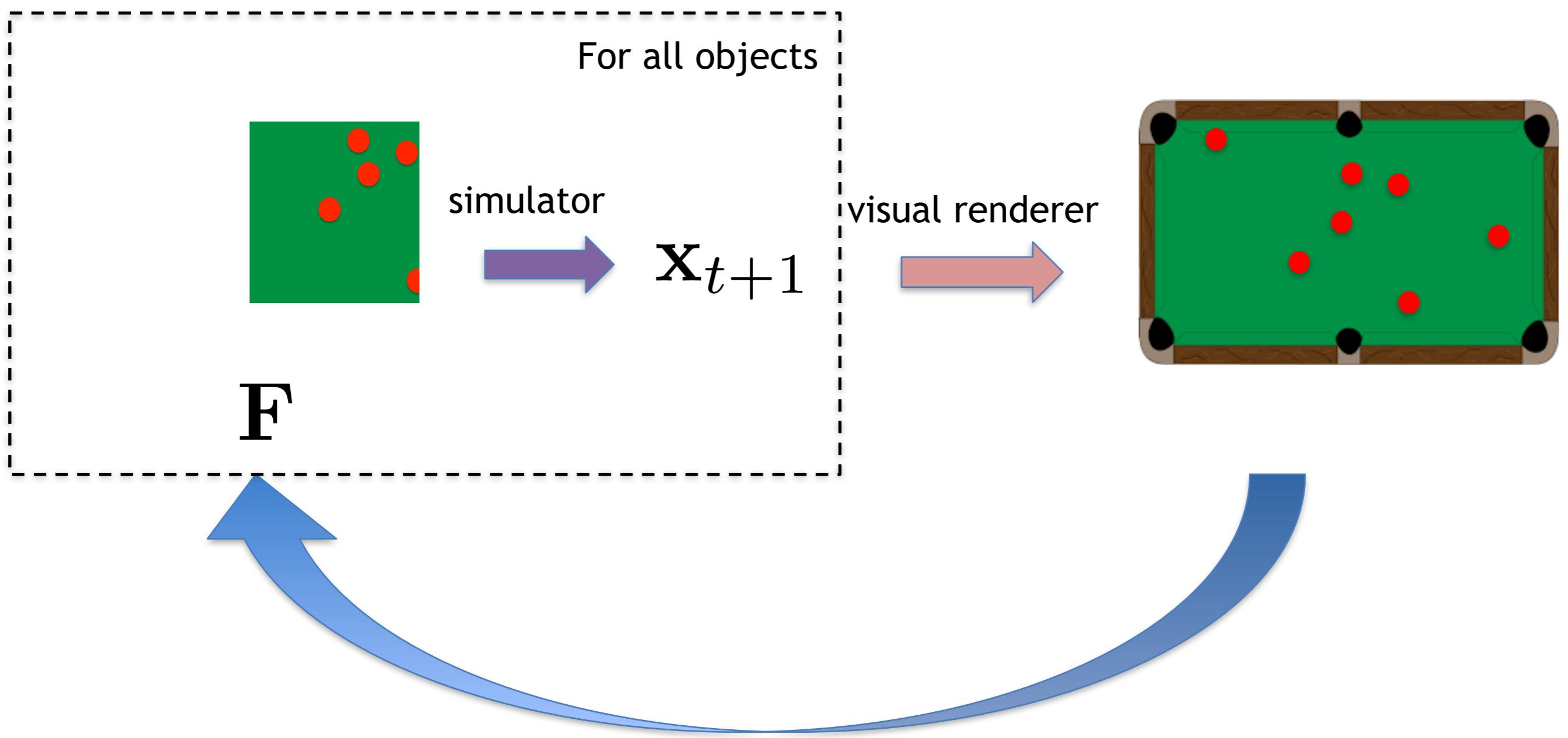


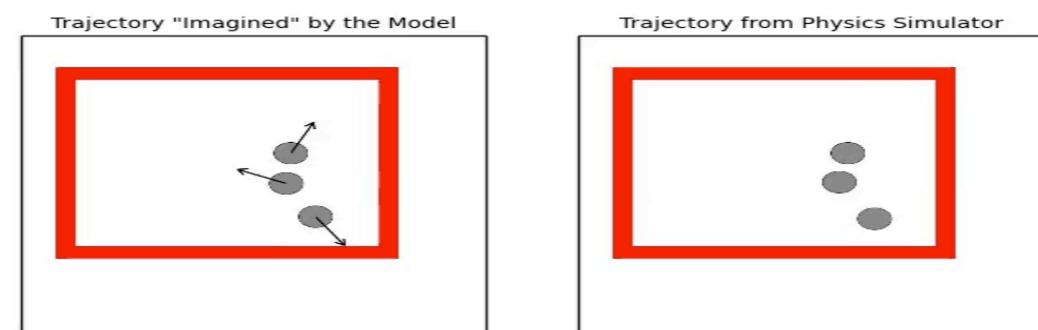
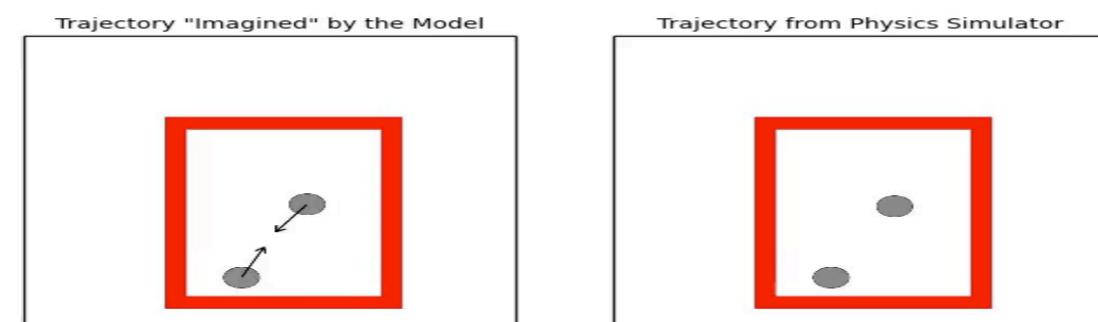
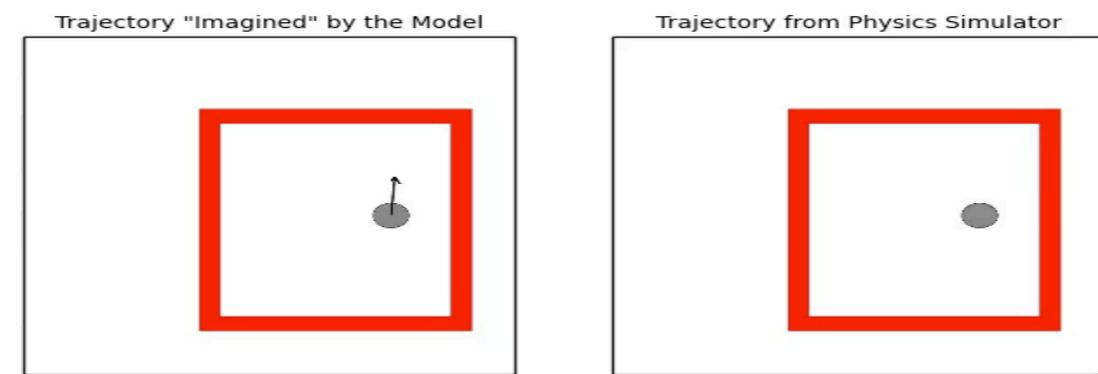


# Network Architecture



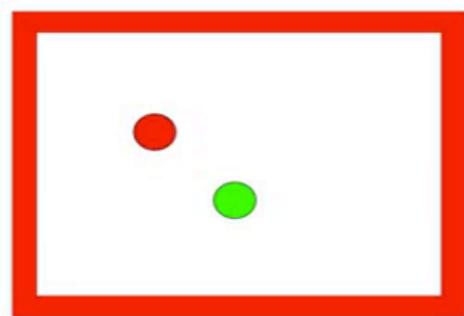
# Test Time: Visual Imaginations



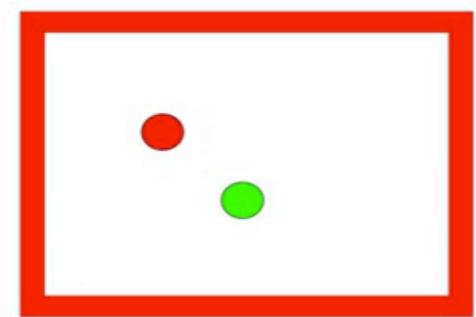


final

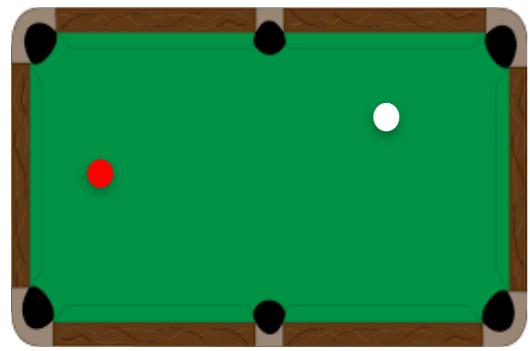
# Playing Billiards



# Playing Billiards

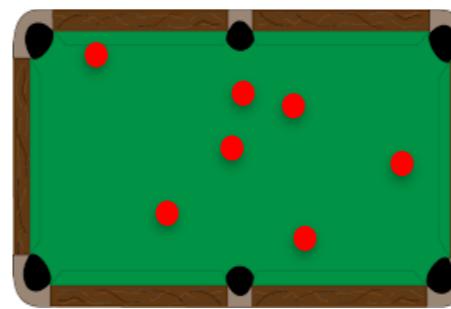


# Policy Learning

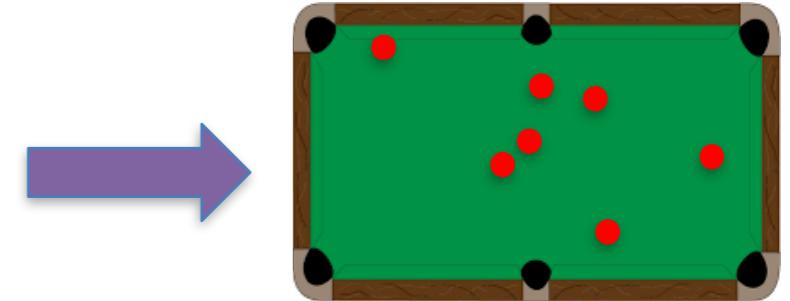


→ F

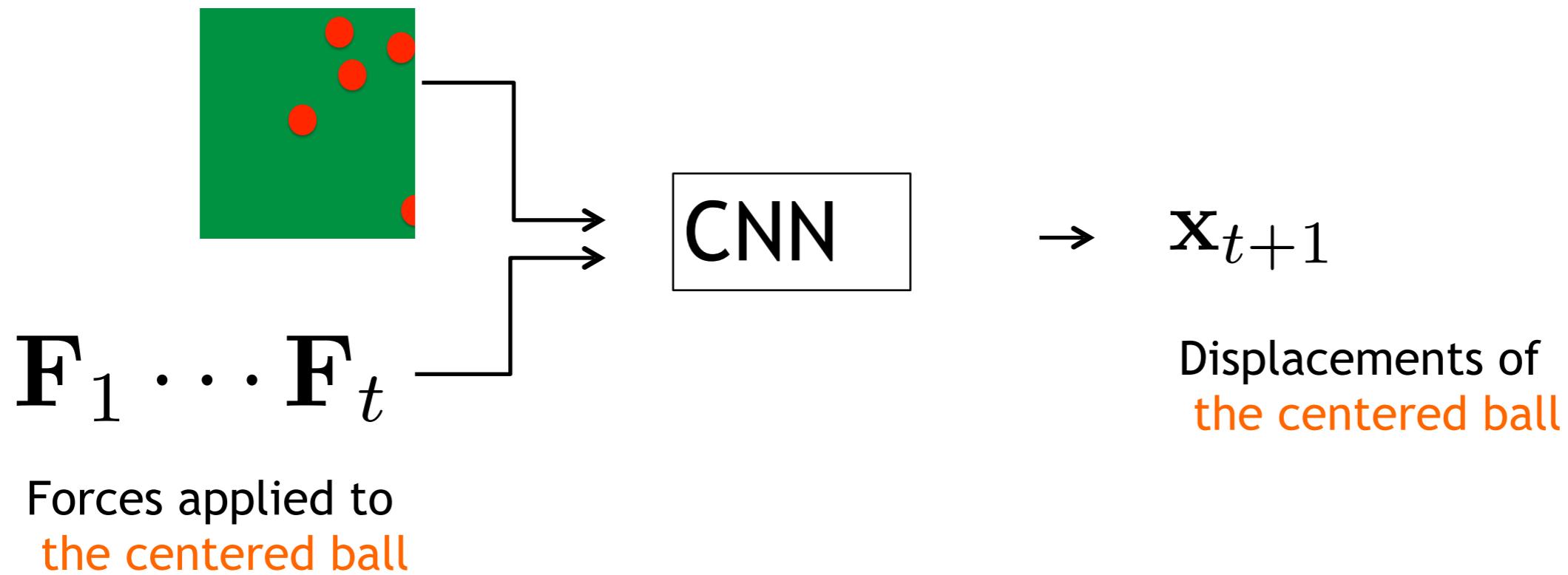
Learning Action  
Conditioned Dynamics



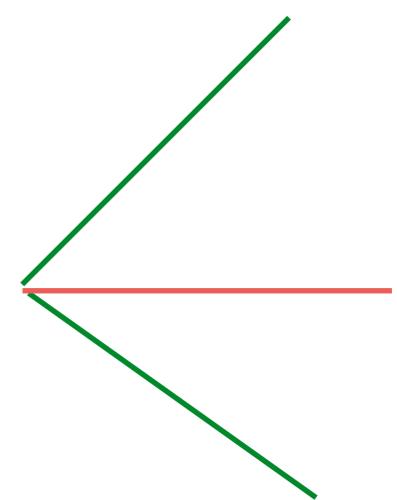
F



# Handling uncertainty in prediction



The model presented so far was deterministic: regression  
It cannot handle uncertainty! Regression to the mean:



Solutions:

- Gaussian Mixture Model
- Stochastic networks

# Text/Handwriting Generation using RNNs

Machine generated sentence:

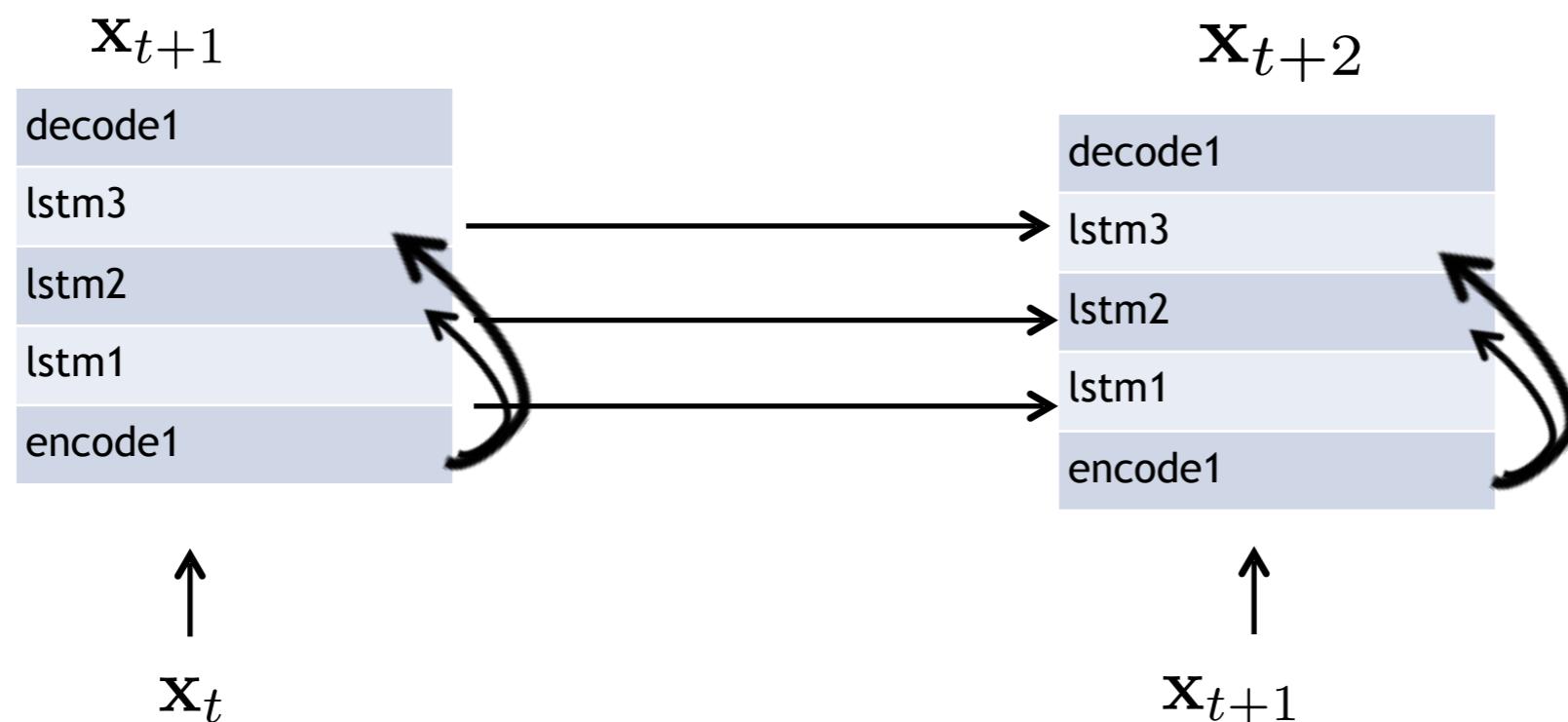
Besides these markets (notably a son of humor).

Machine generated handwriting:

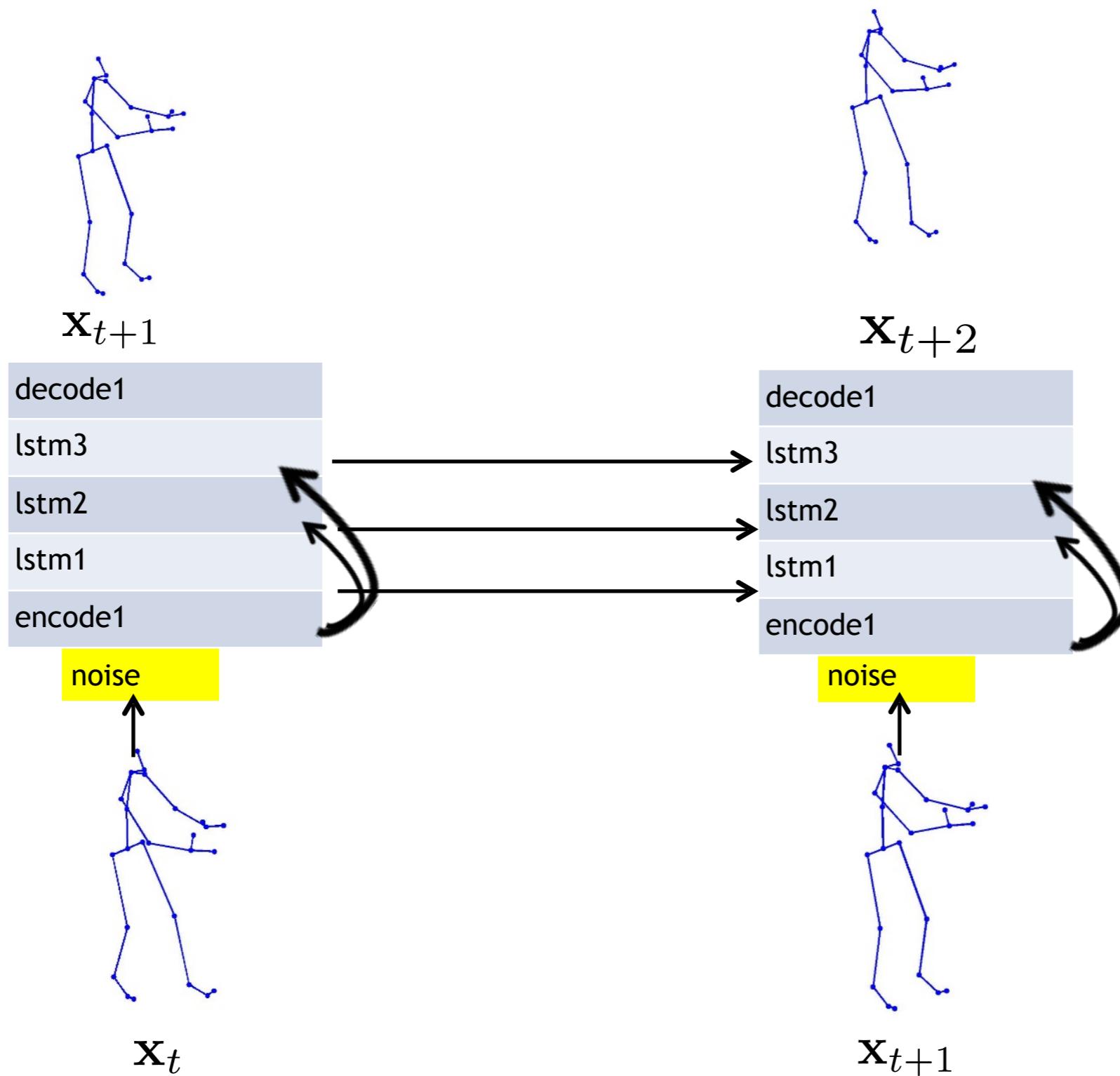
heist. Y Coecls the gaoker w.

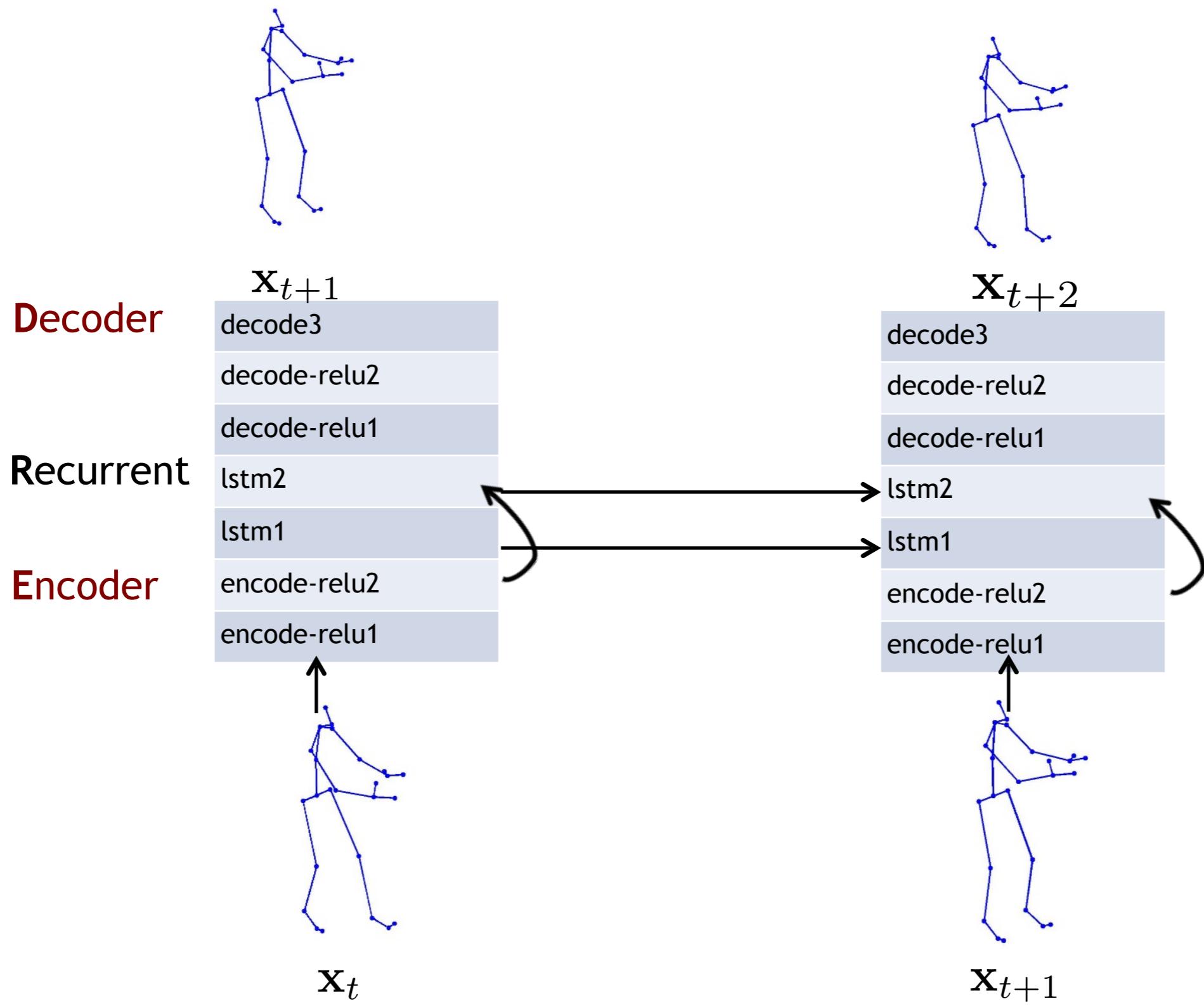
Graves et al., Sutskever et al.

# Text/Handwriting Generation using RNNs

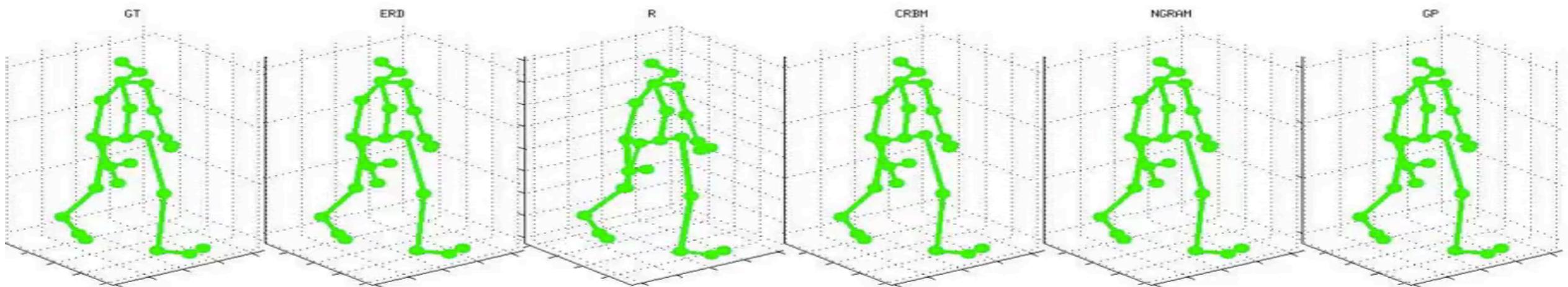


# Learning Dynamics from Motion Capture





Ground-truth      ERD      LSTM-3LR      CRBM      NGRAM      GP

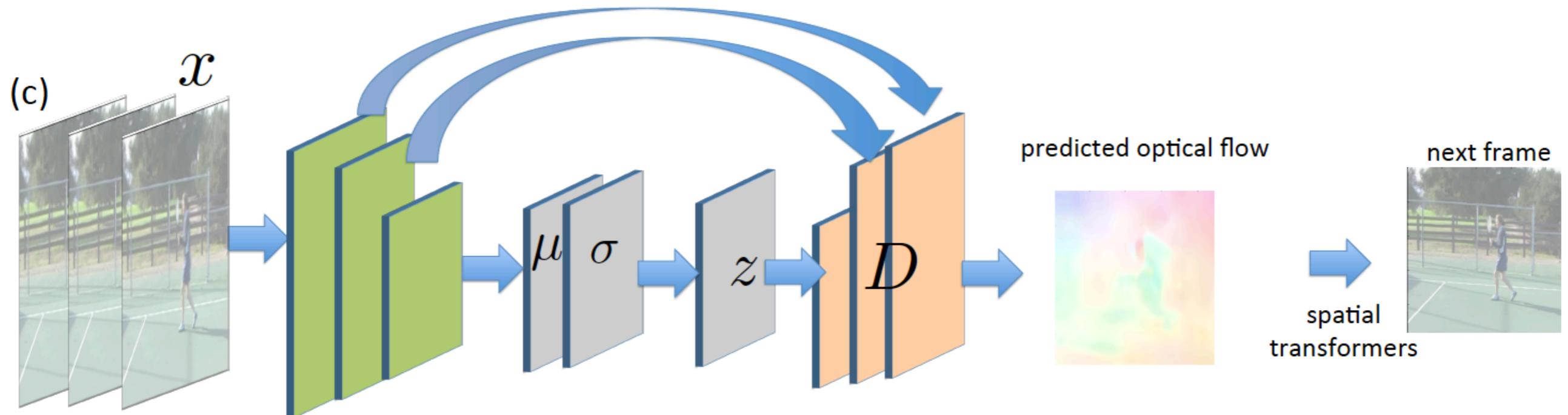


green: conditioning

blue: generation

# Video prediction under multimodal future distributions

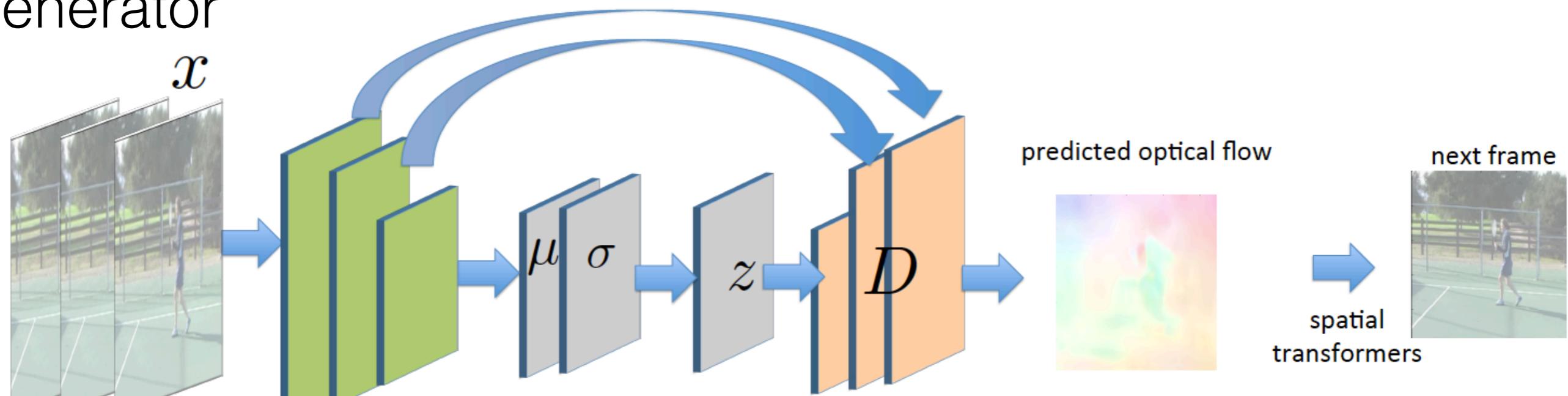
- Stochastic networks



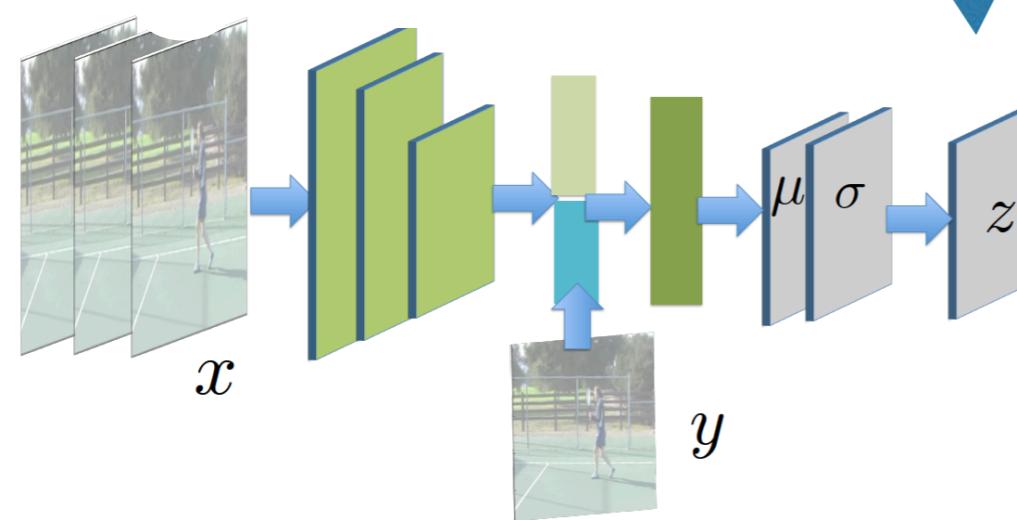
# Video prediction under multimodal future distributions

Train it with (conditional) variational autoencoder

Generator



Recognition network



KL

During training the loss is a combination of KL divergence and reconstruction loss of the recognition network

# Video prediction under multimodal future distributions



# Video prediction under multimodal future distributions

**green: input, red: sampled future motion field and corresponding frame completion**



# Video prediction under multimodal future distributions

green: input  
red: sampled  
future  
completion



more  
future  
completions



green: input  
red: sampled  
future  
completion



more  
future  
completions



# Global models

- Despite the abundance of research papers on the forecasting problem, such approaches so far have not been successful in aiding model predictive control
- Naturally, not all part of the state space will have accurate dynamics, and such inaccuracies will be exploited by the planner
- For MPC, local models currently dominate

# Today's lecture

- Optimal Control, trajectory optimization formulation
- Special but important case: Linear dynamics, quadratic costs (LQR)
- iterative-LQR / Differential Dynamic programming for Non-linear dynamical systems
- Examples of when it works and when it does not
- Learning Dynamics: Global Models

# Next two lectures

- Learning Dynamics: Local Models
- Learning local or global controllers with a trust region based on KL divergence of their trajectory distributions (i-LQR, TRPO)
- Imitating Optimal Controllers to find **general** policies that do not depend on initial state  $x_0$
- Successful examples from the literature