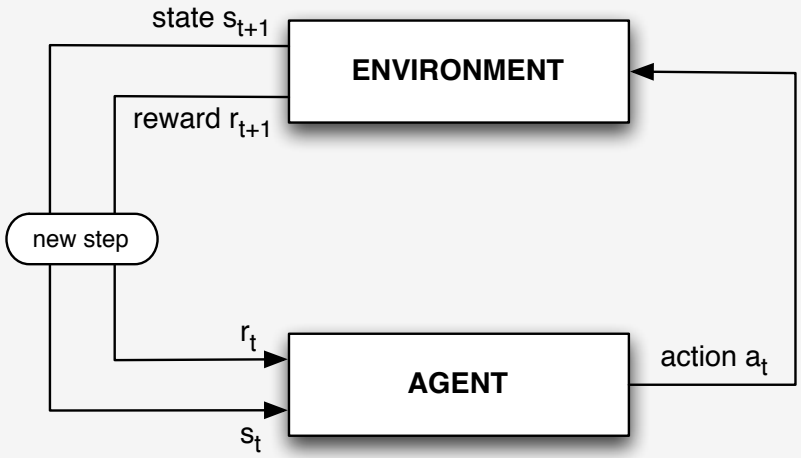
Machine Learning I
Continuous Reinforcement Learning

Thomas Rückstieß

Technische Universität München

January 7/8, 2010

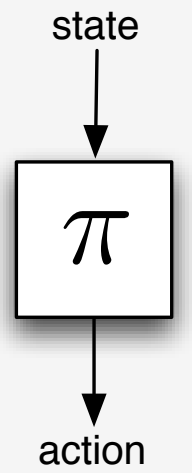
RL Problem Statement (reminder)



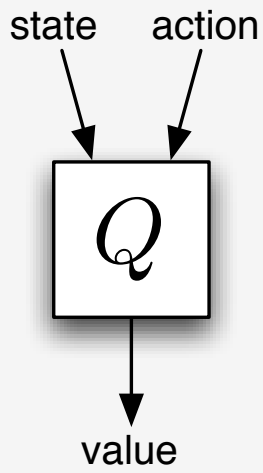
Definition (agent, environment, state, action, reward)

An agent interacts with an environment at discrete time steps $t = 0, 1, 2, \ldots$ At each time step t, the agent receives state $s_t \in \mathcal{S}$ from the environment. It then chooses to execute action $a_t \in \mathcal{A}(s_t)$ where $\mathcal{A}(s_t)$ is the set of available actions in s_t. At the next time step, it receives the immediate reward $r_{t+1} \in \mathbb{R}$ and finds itself in state s_{t+1}.

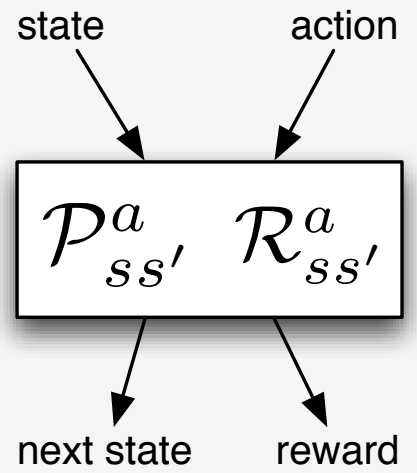
Different Types of RL (reminder)



Direct RL

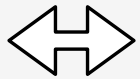


Value-based RL



Model-based RL

data is cheap
computation is limited
e.g. embedded systems

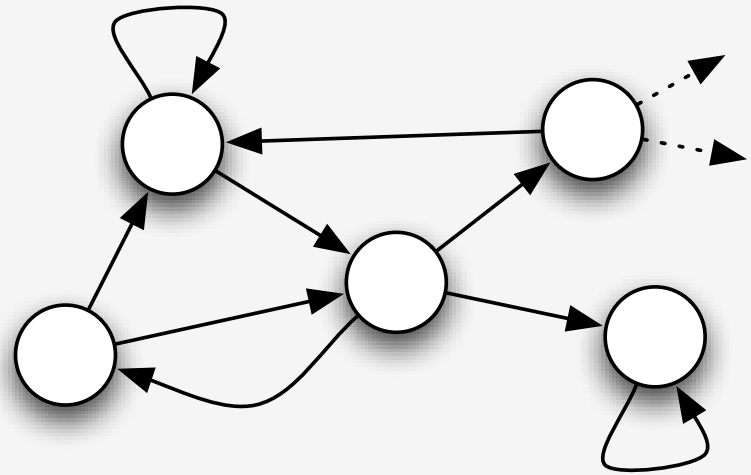


data is expensive
computation doesn't matter
e.g. medical trials

General Assumptions (reminder)

For now, we assume the following:

- Both states and actions are discrete and finite.
- Our problem fulfills the Markov property (MDP)
  - the current state information summarizes all relevant information from the past (e.g. chess, cannonball)
  - the next state is only determined by the last state and the last action, not the entire history
  - the environment has a stationary $\mathcal{P}^a_{ss'}$ and $\mathcal{R}^a_{ss'}$.



Continuous Reinforcement Learning

Why continuous reinforcement learning?

- Problems with too many states/actions
- Generalization for similar states/actions
- Continuous domains, like robotics, computer vision, ...

Let's loosen the restrictive assumptions from last week:

- Both states and actions are discrete and finite.

What changes when we allow $s, a \in \mathbb{R}^n$?

- No transition graphs anymore
- No Q-table anymore
- Q-function? $Q(s, a) \mapsto \mathbb{R}$ is possible, but $\max_a Q(s, a)$ difficult

Continuous RL – Overview

- Value-based Reinforcement Learning
  - Continuous states, discrete actions – NFQ
  - Continuous states and actions – NFQCA
- Direct Reinforcement Learning (Policy Gradients)
  - Finite Difference methods
  - Likelihood Ratio methods
  - REINFORCE
  - 1D controller example
  - Application to Neural Networks

NFQ – Neural Fitted Q-iteration

- We want to apply Q-Learning to continuous states (but discrete actions for now).
- Instead of a Q-table, we have a Q-function (or function approximator, e.g. neural network), that maps $Q(s_t, a_t) \mapsto \mathbb{R}$.
- We sample from the environment and collect (s_t, a_t, r_t)-tuples

Q-Learning Update Rule

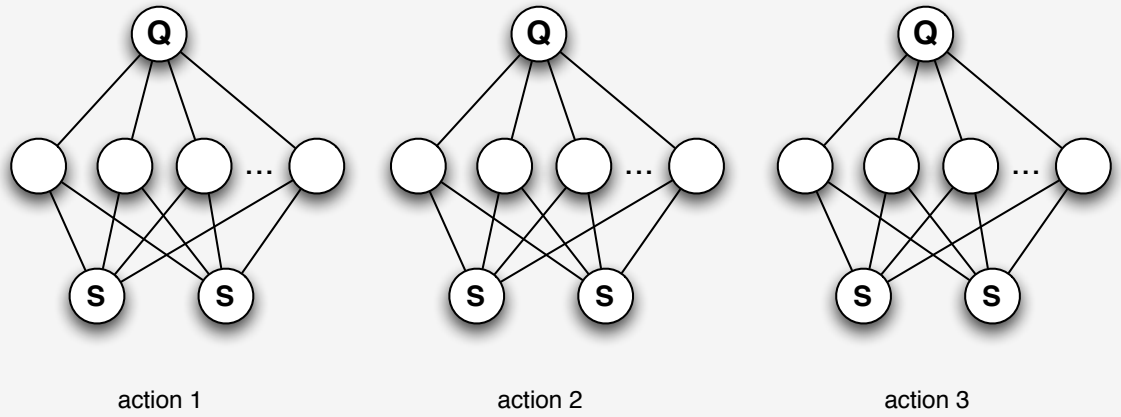
$$Q^\pi(s_t, a_t) \leftarrow Q^\pi(s_t, a_t) + \alpha \left(r_{t+1} + \gamma \max_a Q^\pi(s_{t+1}, a) - Q^\pi(s_t, a_t) \right)$$

- How do we get the maximum over all actions in a certain state s?

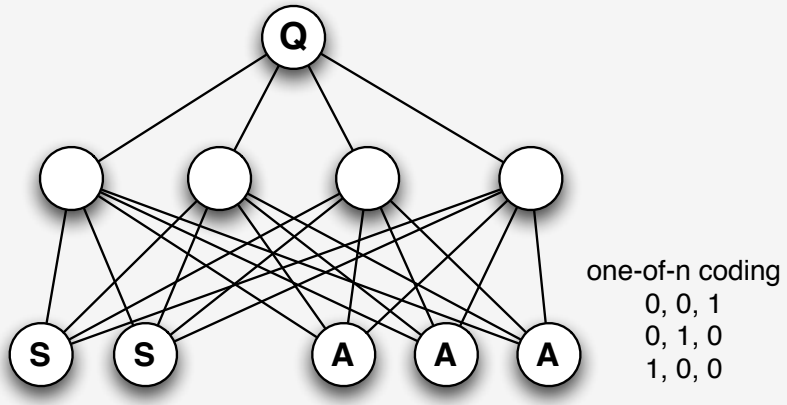
NFQ – Neural fitted Q-iteration

Maximum over discrete actions:

1. Use several neural networks, one for each action



2. or encode the action as additional input to the network



NFQ – Neural fitted Q-iteration

- a forward pass in the network returns $Q^\pi(s_t, a_t)$
- to train the net, convert the (s_t, a_t, r_t)-tuples to a dataset with

  input (s_t, a_t)

  target $Q^\pi(s_t, a_t) + \alpha\left(r_{t+1} + \gamma \max_a Q^\pi(s_{t+1}, a) - Q^\pi(s_t, a_t)\right)$

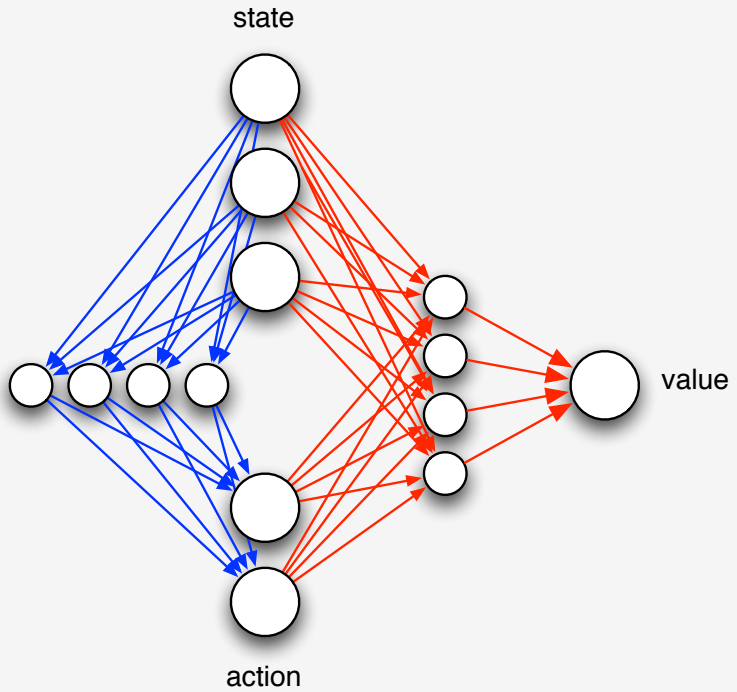
- train network with dataset (until convergence)
- collect new samples by experience and start over

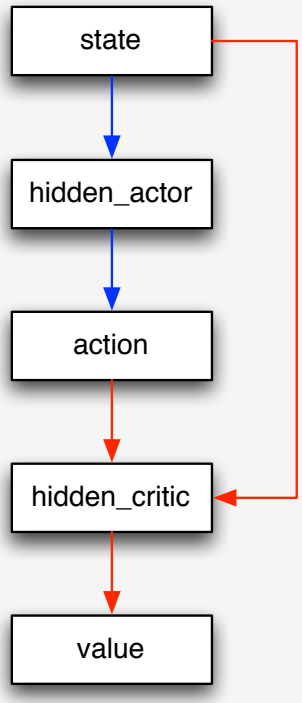
Unfortunately, there is no guarantee of convergence, because the Q-values change during training. But in many cases, it works anyway.

NFQCA – NFQ with continuous actions

With continuous actions, getting the maximum value of a state over all actions is infeasable. Instead, we can use an actor / critic architecture:

- One network (the actor) predicts actions from states
- The second network (the critic), predicts values from states and actions

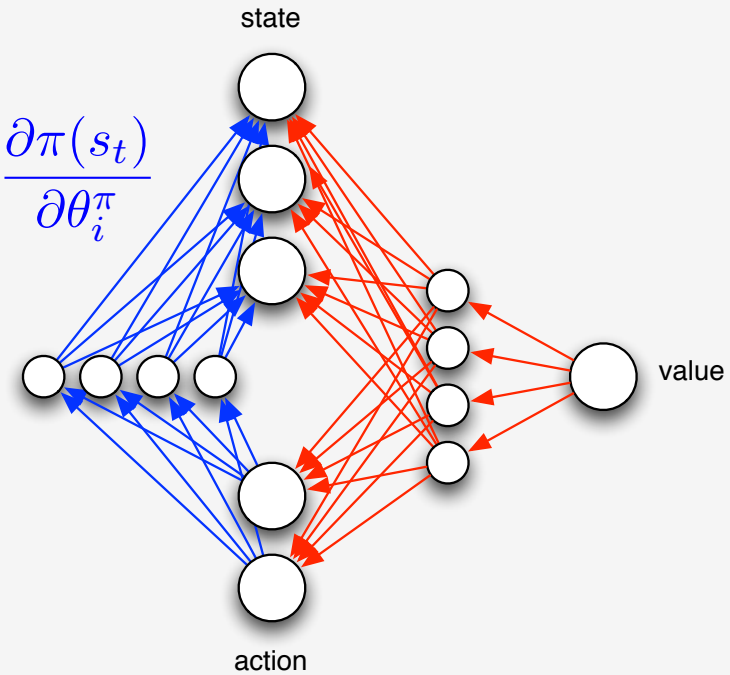




NFQCA Training

1. Backprop TD error through critic network
2. Backprop resulting error further through actor network

$$\theta_i^\pi \leftarrow \theta_i^\pi + \alpha \frac{\partial Q_t(s_t, a_t)}{\partial \pi(s_t)} \frac{\partial \pi(s_t)}{\partial \theta_i^\pi}$$



$$\theta_i^Q \leftarrow \theta_i^Q + \alpha(r_t + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t)) \frac{\partial Q_t(s_t, a_t)}{\partial \theta_i^Q}$$

More value-based continuous RL

There are other methods of using function approximation with value-based RL ($\rightarrow$ Sutton&Barto, Chapter 8).

Continuous RL – Overview

- Value-based Reinforcement Learning
  - Continuous states, discrete actions – NFQ
  - Continuous states and actions – NFQCA

- Direct Reinforcement Learning (Policy Gradients)
  - Finite Difference methods
  - Likelihood Ratio methods
  - REINFORCE
  - 1D controller example
  - Application to Neural Networks

Direct Reinforcement Learning

Key aspects of direct reinforcement learning:

- skip value functions (change policy directly)
- sample from experience (like MC methods)
- calculate gradient of parameterized policy
- follow gradient to local optimum

$\Rightarrow$ Methods that follow the above description are called Policy Gradient Methods or short Policy Gradients.

Policy Gradients – Notation

For now, we will even loosen our second assumption:

- Our problem fulfills the Markov property.

The next state can now depend on the whole history h, not just the last state-action pair (s, a).

Policy $\pi(a|h, \theta)$ probability of taking action a when encountering history h. The policy is parameterized with θ.

History h^π history of all states, actions, rewards following policy π.
$h_0^\pi = \{s_0\}$
$h_t^\pi = \{s_0, a_0, r_0, s_1, \ldots, a_{t-1}, r_{t-1}, s_t\}$

Return $R(h^\pi) = \sum_{t=0}^{T} \gamma^t r_t$

Performance Measure $J(\pi)$

We need a way to measure the performance for the whole policy π. We define the overall performance of a policy as:

$$J(\pi) = E_\pi\{R(h^\pi)\} = \int_{h^\pi} p(h^\pi)R(h^\pi)\, dh^\pi \tag{1}$$

Optimize the parameters θ of the policy to improve J:

$$\begin{aligned} \nabla_\theta J(\pi) &= \nabla_\theta \int_{h^\pi} p(h^\pi)R(h^\pi)\, dh^\pi \\ &= \int_{h^\pi} \nabla_\theta p(h^\pi)R(h^\pi)\, dh^\pi. \end{aligned} \tag{2}$$

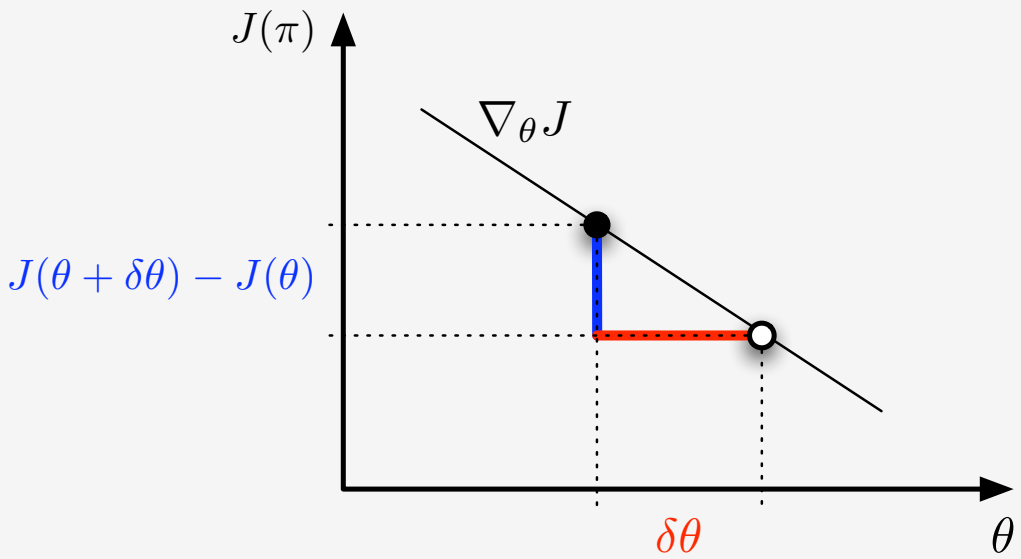
Knowing the gradient, we can update θ as

$$\theta_{t+1} = \theta_t + \alpha \nabla_\theta J(\pi) \tag{3}$$

Finite Differences

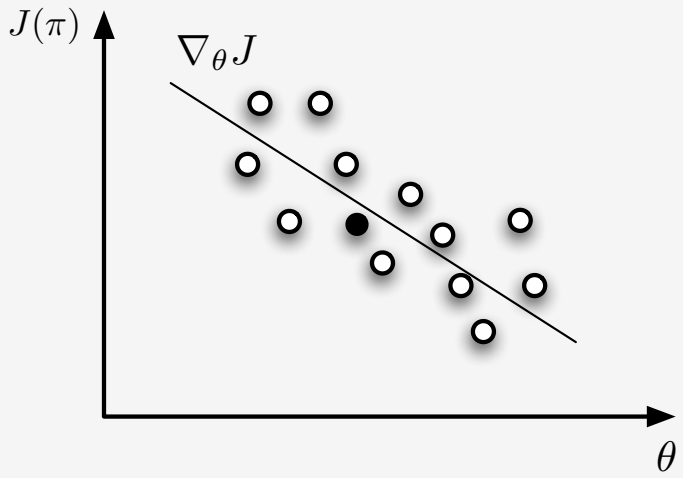
One method to approximate the gradient of the performance is Finite Differences:

$$\frac{\partial J(\theta)}{\partial \theta_i} \approx \frac{J(\theta + \delta\theta) - J(\theta)}{\delta\theta_i}$$



Finite Differences

Or even better: take many samples with different $\delta\theta$'s and run a linear regression ($\Rightarrow$ pseudo inverse)



$$\text{matrix } \Theta_i = \begin{bmatrix} \delta\theta_i & 1 \end{bmatrix}, \quad \text{vector } J_i = \begin{bmatrix} J(\theta + \delta\theta_i) \end{bmatrix}$$

$$\beta = (\Theta^T \Theta)^{-1} \Theta^T J$$

Finite Differences

Problems with Finite Differences

- For Finite Differences, the chosen action can be written as

$$a = f(h, \theta + \epsilon),$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is some exploratory noise.

- We change the policy parameters θ directly $\Rightarrow$ the resulting controller is not predictable.
Example robot control: changing the parameters randomly can damage the robot or cause a risk for nearby humans

- In some recent publications, finite differences perform badly in probabilistic settings $\Rightarrow$ most real problems are probabilistic.

Likelihood Ratio

The safer (and currently more popular) method is to estimate the gradient with likelihood ratio methods.

- Policy Gradients explore by perturbing the resulting action instead of the parameters

  $$a = f(h, \theta) + \epsilon,$$

  again with some exploratory noise $\epsilon \sim \mathcal{N}(0, \sigma^2)$.

- The policy, that causes this behavior is unknown (and might not even exist).

- $J(\theta + \delta\theta)$ cannot be measured.

- Another method of estimating $\nabla_\theta J$ is needed.

Likelihood Ratio

We start from the performance gradient equation:

$$\nabla_\theta J(\pi) = \int_{h^\pi} \nabla_\theta p(h^\pi) R(h^\pi)\ dh^\pi$$

where the probability of encountering history h under policy π is:

$$\begin{aligned} p(h^\pi) &= p(s_0)\pi(a_0|h_0^\pi)p(s_1|h_0^\pi, a_0)\pi(a_1|h_1^\pi)p(s_2|h_1^\pi, a_1)\ldots \\ &= p(s_0)\prod_{t=1}^{T}\pi(a_{t-1}|h_{t-1}^\pi)\, p(s_t|h_{t-1}^\pi, a_{t-1}) \end{aligned}$$

Multiplying with $1 = \frac{p(h^\pi)}{p(h^\pi)}$ gives

$$\nabla_\theta J(\pi) = \int_{h^\pi} \frac{p(h^\pi)}{p(h^\pi)} \nabla_\theta p(h^\pi) R(h^\pi)\ dh^\pi$$

Likelihood Ratio

$$\nabla_\theta J(\pi) = \int_{h^\pi} \frac{p(h^\pi)}{p(h^\pi)} \nabla_\theta p(h^\pi) R(h^\pi)\ dh^\pi$$

can be simplified by applying $\frac{1}{x} \cdot \nabla x = \nabla \log(x)$:

$$\nabla_\theta J(\pi) = \int_{h^\pi} p(h^\pi)\ \nabla_\theta \log p(h^\pi)\ R(h^\pi)\ dh^\pi$$

where – after a few more steps – we get

$$\nabla_\theta \log p(h^\pi) = \sum_{t=1}^{T} \nabla_\theta \log \pi(a_{t-1}|h_{t-1}^\pi)$$

which we will insert into above equation.

Likelihood Ratio – REINFORCE

This leads to the likelihood ratio gradient estimate

$$\begin{aligned}\nabla_\theta J(\pi) &= \int p(h^\pi) \cdot \sum_{t=1}^{T} \nabla_\theta \log \pi(a_{t-1}|h_{t-1}^\pi) \cdot R(h^\pi)\ dh^\pi \\ &= E_\pi \left\{ \sum_{t=1}^{T} \nabla_\theta \log \pi(a_{t-1}|h_{t-1}^\pi) \cdot R(h^\pi) \right\}\end{aligned}$$

Just like in the classical case, the expectation cannot be calculated directly. We use Monte-Carlo Sampling of episodes to approximate and get Williams' REINFORCE algorithm (1992):

$$\nabla_\theta J(\pi) \approx \frac{1}{N} \sum_{h^\pi} \sum_{t=1}^{T} \nabla_\theta \log \pi(a_{t-1}|h_{t-1}^\pi) \cdot R(h^\pi)$$

Example: Linear Controller (1D)

After this general derivation, we now go back to an MDP

$$\pi(a_t|h_t,\theta) = \pi(a_t|s_t,\theta)$$

Here with a linear controller:

$$a = f(s,\theta) + \epsilon = \theta s + \epsilon, \quad \epsilon \sim \mathcal{N}(0,\sigma^2)$$

The actions are distributed like

$$a \sim \mathcal{N}(\theta s, \sigma^2)$$

and the policy is thus

$$\pi(a|s) = p(a|s,\theta,\sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(a-\theta s)^2}{2\sigma^2}\right)$$

Example: Linear Controller (1D)

Policy from last slide:

$$\pi(a|s) = p(a|s, \theta, \sigma) = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{(a-\theta s)^2}{2\sigma^2}\right)$$

Deriving the policy with respect to the free parameters θ and σ results in

$$\begin{aligned} \nabla_\theta \log \pi(a|s) &= \frac{(a-\theta s)s}{\sigma^2} \\ \nabla_\sigma \log \pi(a|s) &= \frac{(a-\theta s)^2 - \sigma^2}{\sigma^3} \end{aligned}$$

Example: Linear Controller (1D)

REINFORCE Algorithm

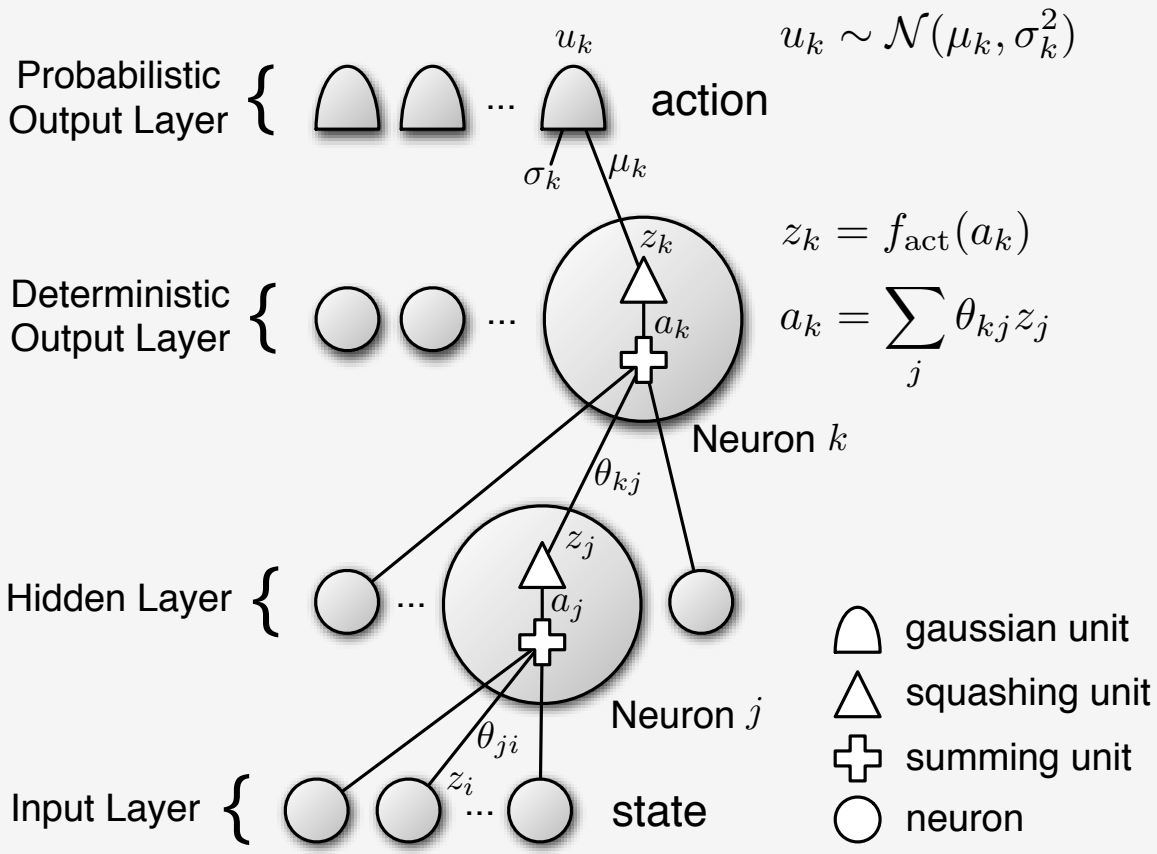
1. initialize θ randomly
2. run N episodes, draw actions $a \sim \pi(a|s,\theta)$, remember all s_t^n, a_t^n, r_t^n
3. approximate gradient with REINFORCE

$$\nabla_\theta J(\pi) \approx \frac{1}{N} \sum_{n=0}^{N-1} \sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t^n|s_t^n) \cdot R_t^n$$

4. update the parameter $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi)$
5. goto 2

Application to Neural Network Controllers

How does the policy for a NN controller look like?



Application to Neural Network Controllers

Again we need to derive the log of the policy with respect to the parameters, which here are the weights θ_{ij} of the network

$$\frac{\partial \log \pi(a|s)}{\partial \theta_{ji}} = \sum_{k \in \mathbb{O}} \frac{\partial \log \pi_k(a_k|s)}{\partial \mu_k} \frac{\partial \mu_k}{\partial \theta_{ji}}$$

The factor $\frac{\partial \mu_k}{\partial \theta_{ji}}$ describes the back-propagation through the network.

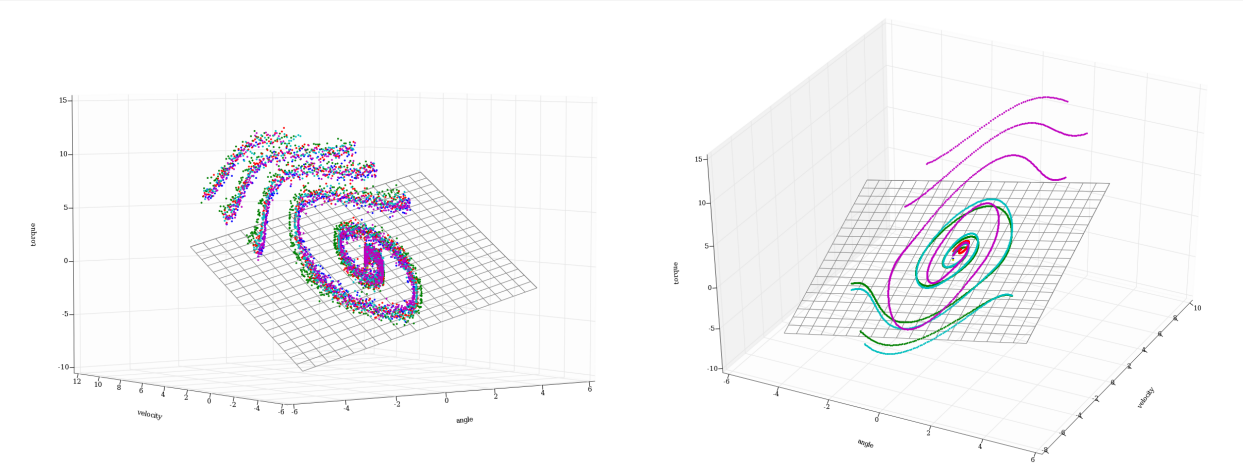
$\Rightarrow$ use existing NN implementation, but back-propagate the log likelihood derivatives $\frac{\partial \log \pi_k(a_k|s)}{\partial \mu_k}$ instead of the error from supervised learning.

$\Rightarrow$ use REINFORCE to approximate $\nabla_\theta J(\pi)$ which results in the weight update $\theta \leftarrow \theta + \alpha \nabla_\theta J(\pi)$.

Where did the exploration go?

- no explicit exploration
- probabilistic policy $\pi(s, a) = p(a \mid s)$
- covers two "random" concepts: non-deterministic policies and exploration
- this is actually not very efficient $\Rightarrow$ State-Dependent Exploration

$$a = f(s, \theta + \epsilon) \qquad a = f(s, \theta) + \epsilon \qquad a = f(s, \theta) + \epsilon(s)$$



Conclusion

Does it work?

- Yes, for few parameters and many episodes
- Policy Gradients converge to a local optimum
- There are ways to improve REINFORCE: baselines, pegasus, state-dependent exploration, ...
- New algorithms use data more efficiently: ENAC