Topics in Reinforcement Learning: AlphaZero, ChatGPT, Neuro-Dynamic Programming, Model Predictive Control, Discrete Optimization, Applications

Arizona State University

Course CSE 691, Spring 2025

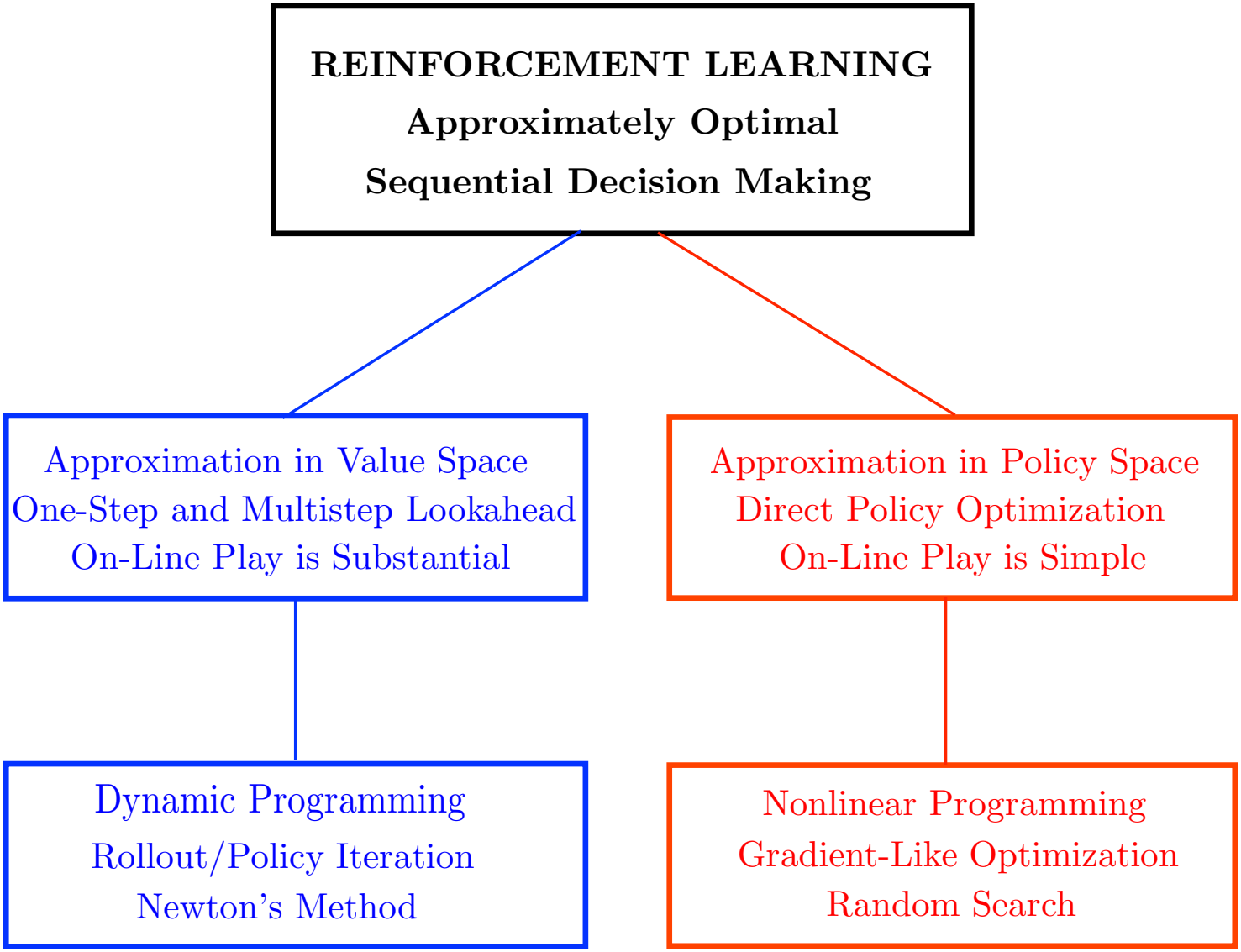
Links to Class Notes, Videolectures, and Slides at
http://web.mit.edu/dimitrib/www/RLbook.html

Prof. Dimitri P. Bertsekas (dimitrib@mit.edu)
and
Dr Yuchao Li (yuchaoli@asu.edu)

Lecture 12
Off-Line Training of Cost Functions and Policies, Approximation in Policy Space, Policy Gradient and Random Search Methods

Outline

Reinforcement Learning - An Overview Figure



RL deals with exactly the same mathematical problem as DP

Off-Line Training: Types of Approximations

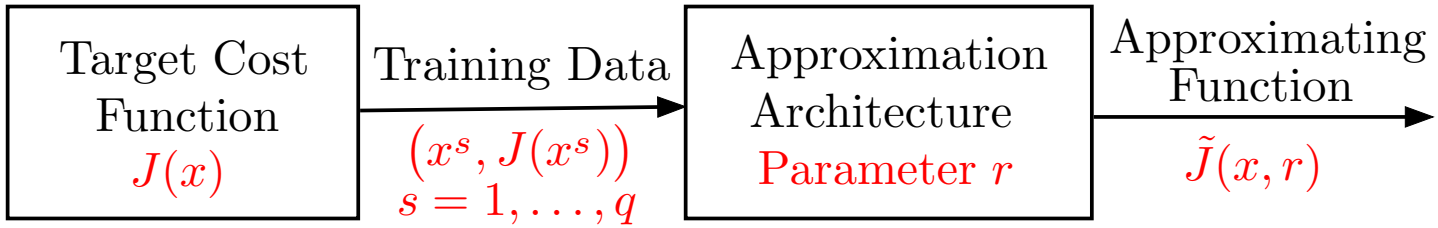
There are two types of off-line approximations in RL:

- Cost approximation in finite and infinite horizon problems
  - Optimal cost function $J_k^*(x_k)$ or $J^*(x)$ [also optimal Q-function $Q_k^*(x_k, u_k)$ or $Q^*(x, u)$]
  - Cost function of a given policy $J_{\pi,k}(x_k)$ [also $J_\mu(x)$, Q-function of a policy $Q_{\pi,k}(x_k, u_k)$ or $Q_\mu(x, u)$]
- Policy approximation in finite and infinite horizon problems
  - Approximation of an optimal policy $\mu_k^*(x_k)$ or $\mu^*(x)$
  - Approximation of a given policy $\mu_k(x_k)$ or $\mu(x)$

We will focus on infinite horizon parametric approximations $\tilde{J}(x, r)$ and $\tilde{\mu}(x, r)$

- These are functions of x that depend on a parameter vector r
- An example is neural networks (r is the set of weights)

First Context: Approximation of a Target Cost Function



TRAINING CAN BE DONE WITH SPECIALIZED OPTIMIZATION SOFTWARE SUCH AS

GRADIENT-LIKE METHODS OR OTHER LEAST SQUARES METHODS

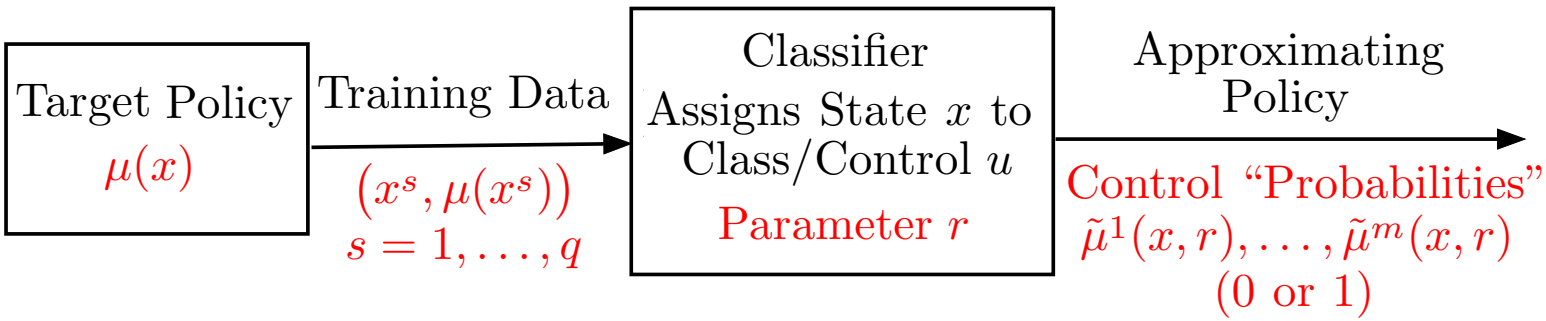
Second Context: Approximation of a Target Policy

Similarly, we introduce a parametric approximation architecture $\tilde{\mu}(x, r)$

If the control has continuous/real-valued components, the training is similar to the cost function case

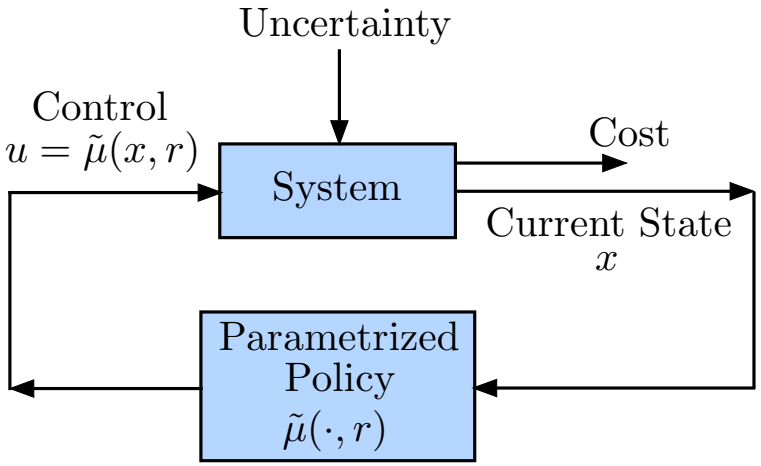
If the control comes from a finite control space $\{u^1, \ldots, u^m\}$, an alternative approach is possible

View a policy μ as a classifier: A function that maps x into a "category" $\mu(x)$ (i.e., a "black box" that classifies objects into one of finite number of categories)



TRAINING CAN BE DONE WITH CLASSIFICATION SOFTWARE

Third Context: Direct Policy Optimization



Select a policy by parametric cost optimization

- Each parameter value r defines a stationary (possibly randomized) policy $\tilde{\mu}(r)$ with cost $J_{\tilde{\mu}(r)}(x_0)$ starting from initial state x_0
- Determine r through the minimization

$$\min_r J_{\tilde{\mu}(r)}(x_0)$$

- More generally, determine r through the minimization

$$\min_r E\left\{J_{\tilde{\mu}(r)}(x_0)\right\}$$

where the $E\{\cdot\}$ is with respect to a suitable probability distribution of x_0

Parametric Cost Function Approximation Generalities

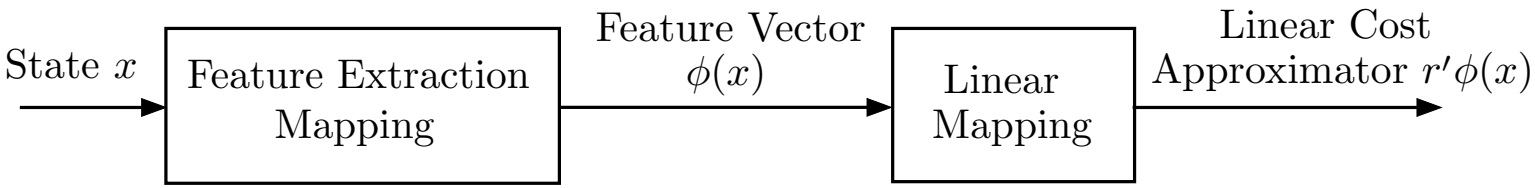
- We start with a parametric architecture, i.e., a class of functions $\tilde{J}(x, r)$ that depend on x and on a vector $r = (r_1, \ldots, r_m)$ of m "tunable" scalar parameters.
- The training problem: Adjust r so that $\tilde{J}(x, r)$ "matches" the training data from the target function.

Linear, nonlinear, and feature-based architectures

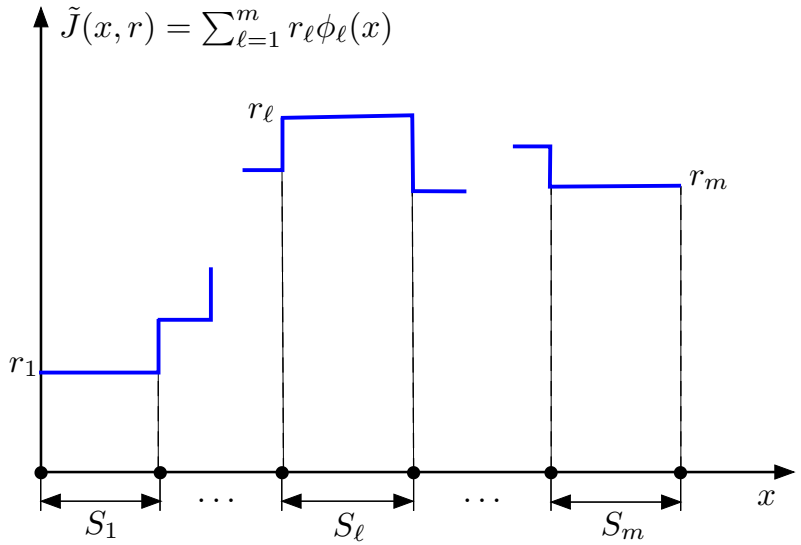
- Architectures are called linear or nonlinear, if $\tilde{J}(x, r)$ is linear or nonlinear in r.
- Architectures are feature-based if they depend on x via a feature vector $\phi(x)$ that captures "major characteristics" of x,

$$\tilde{J}(x, r) = \hat{J}\big(\phi(x), r\big),$$

where $\hat{J}$ is some function. Intuitive idea: Features capture dominant nonlinearities.
- A linear feature-based architecture: $\tilde{J}(x, r) = \sum_{\ell=1}^{m} r_\ell \phi_\ell(x) = r'\phi(x)$, where r_ℓ and $\phi_\ell(x)$ are the ℓth components of r and $\phi(x)$.



A Simple Example of a Linear Feature-Based Architecture



Piecewise constant approximation

- Partition the state space into subsets $S_1, \ldots, S_m$. The ℓth feature is defined by membership in the set S_ℓ, i.e., the indicator function of S_ℓ,

$$\phi_\ell(x) = \begin{cases} 1 \text{ if } x \in S_\ell \\ 0 \text{ if } x \notin S_\ell \end{cases}$$

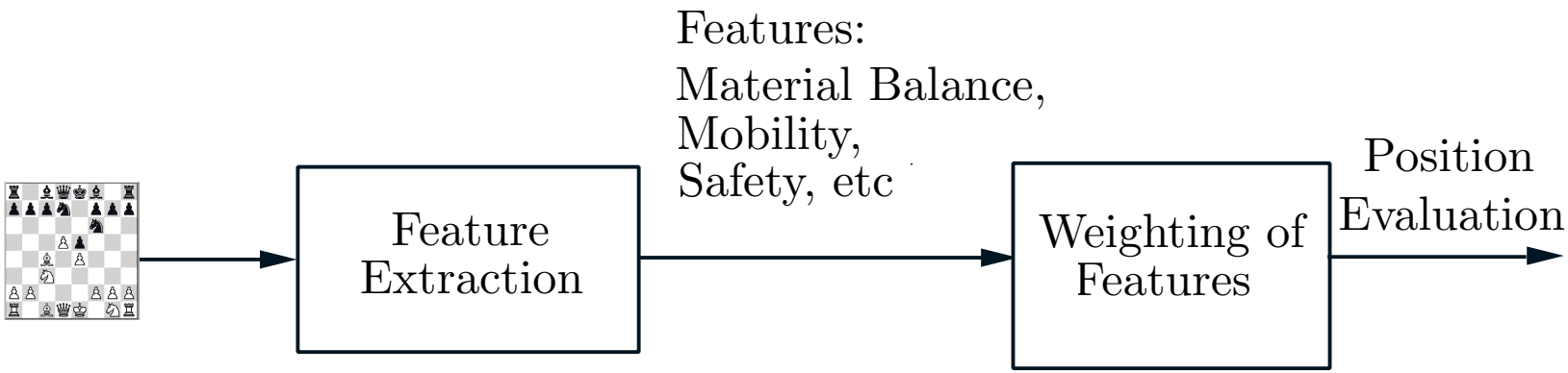
- The architecture

$$\tilde{J}(x,r) = \sum_{\ell=1}^{m} r_\ell \phi_\ell(x),$$

is piecewise constant with value r_ℓ for all x within the set S_ℓ.

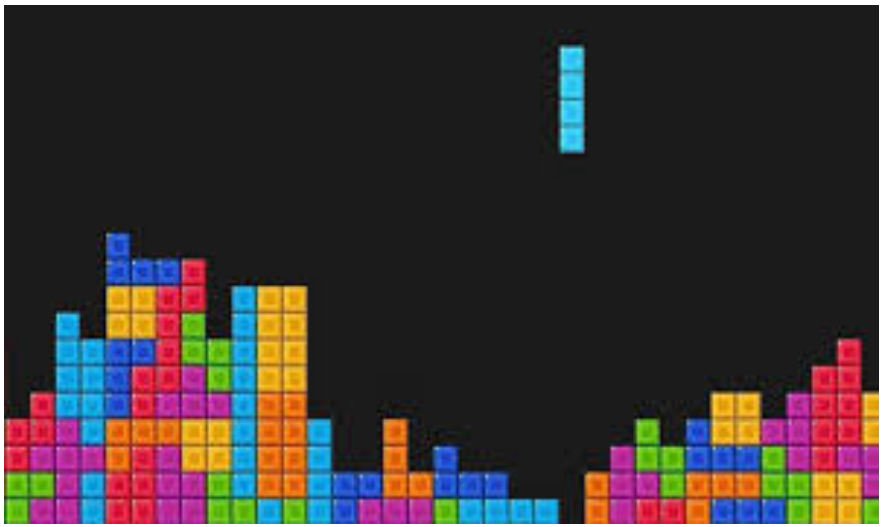
Examples of Problem-Specific Feature-Based Architectures

Chess

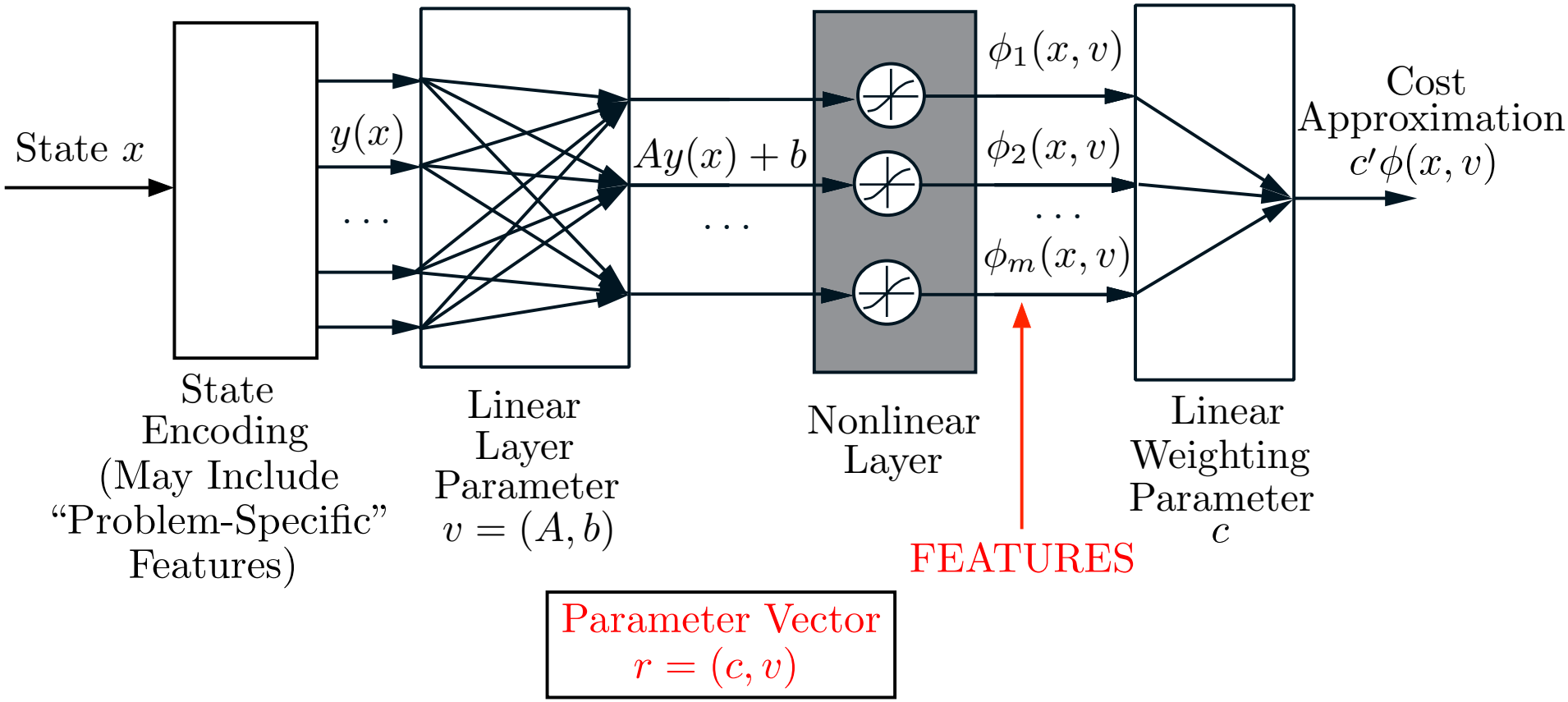


****

Tetris



Neural Networks: An Architecture that Works with No Knowledge of Features



A SINGLE LAYER NEURAL NETWORK

Training by Sampling and Least Squares

Least squares regression

- Collect a set of state-cost training pairs (x^s, β^s), $s = 1, \ldots, q$, where β^s is equal to the target cost $J(x^s)$ plus some "noise".
- r is determined by solving the problem

$$\min_r \sum_{s=1}^{q} \left(\tilde{J}(x^s, r) - \beta^s\right)^2$$

- Sometimes a quadratic regularization term $\gamma\|r\|^2$ is added to the least squares objective, to facilitate the minimization (among other reasons - issue of overfitting).

Training of linear feature-based architectures can be done exactly

- If $\tilde{J}(x, r) = r'\phi(x)$, the exact solution of the problem is

$$\hat{r} = \left(\sum_{s=1}^{q} \phi(x^s)\phi(x^s)'\right)^{-1} \sum_{s=1}^{q} \phi(x^s)\beta^s$$

- This is expensive for a large number of features and data set; may not be best.
- Alternatives include Temporal Difference (TD) methods for the case where $r'\phi(x)$ approximates the cost function of a given policy in a finite spaces MDP

Incremental Gradient Methods (The Workhorse of Machine Learning)

Sum of terms optimization problem

$$\min_y f(y) = \sum_{i=1}^{m} f_i(y) \quad \text{(e.g., } f_i(y) \text{ is the Error Between } i\text{th Data Point and a Model } y\text{)}$$

Each f_i is a differentiable scalar function of the vector y

The ordinary gradient method generates y^{k+1} from y^k according to

$$y^{k+1} = y^k - \gamma^k \nabla f(y^k) = y^k - \gamma^k \sum_{i=1}^{m} \nabla f_i(y^k)$$

where $\gamma^k > 0$ is a stepsize parameter

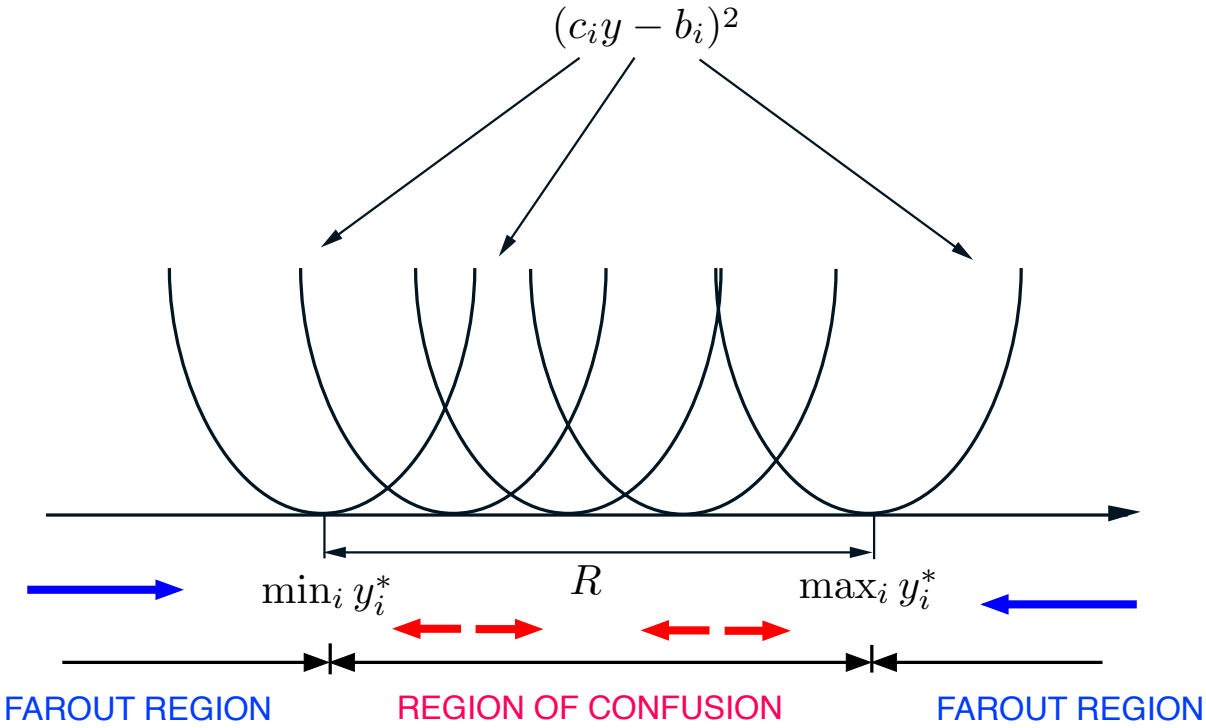
The incremental gradient method (aka "stochastic gradient descent")

Choose an index i_k and iterate according to

$$y^{k+1} = y^k - \gamma^k \nabla f_{i_k}(y^k)$$

where $\gamma^k > 0$ (in practice, diagonal scaling is commonly used - ADAM software)

The Advantage of Incrementalism: An Interpretation from my 1996 Neurodynamic Programming Book



Minimize $f(y) = \frac{1}{2}\sum_{i=1}^{m}(c_i y - b_i)^2$

Two cases of interest:

- When far from convergence: Incremental gradient is as fast as ordinary gradient with $1/m$ amount of work.
- When close to convergence: Incremental gradient gets confused and requires a diminishing stepsize for convergence.

Learning from Human or Software-Based Experts

- Suppose we have a software or human expert that can choose a “good" or “near-optimal" control $u = \hat{\mu}(x)$ at any state x. We view $\hat{\mu}$ as the target policy.
- We form a sample set of state-control pairs (x^s, u^s), $s = 1, \ldots, q$, where $u^s = \hat{\mu}(x^s)$.
- We introduce a parametric family of policies $\tilde{\mu}(x, r)$. Then obtain r by

$$\min_r \sum_{s=1}^{q} \left\| u^s - \tilde{\mu}(x^s, r) \right\|^2$$

  For this the control space should be continuous [or else $\tilde{\mu}(x, r)$ should be randomized].
- This approach is known as supervised learning from an expert.
- It has been used (in various forms) in games (backgammon, chess), robotics, etc.
- It can also be used, among others, for initialization of other methods.

Policy Approximation on Top of Value Approximation

- First compute approximate cost-to-go function $\tilde{J}$ using an approximation in value space scheme.
- This defines the corresponding suboptimal policy $\hat{\mu}$ through one-step lookahead,

$$\hat{\mu}(x) \in \arg \min_{u \in U(x)} E\Big\{ \big(g(x,u,w) + \tilde{J}\big(f(x,u,w)\big)\big) \Big\}$$

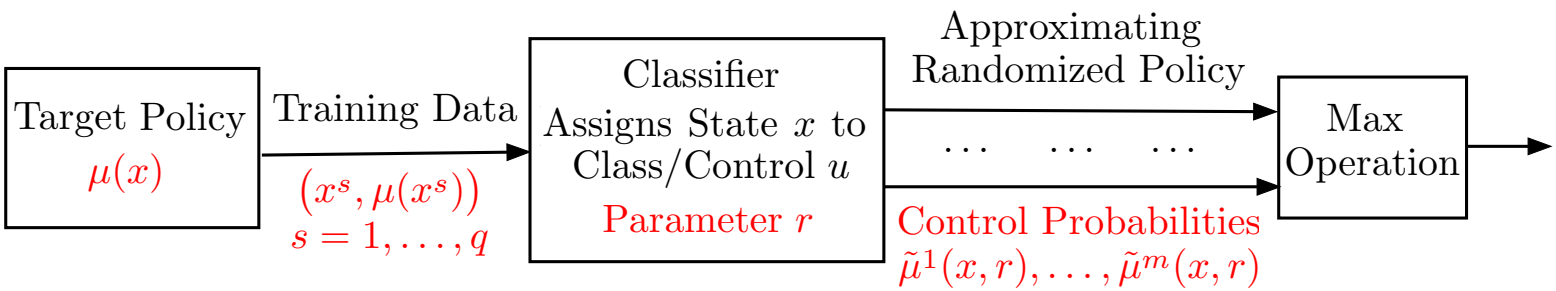
  or a multistep lookahead version. We view $\hat{\mu}$ as the target policy.
- Approximate $\hat{\mu}$ using a training set consisting of a large number q of sample pairs (x^s, u^s), $s = 1, \ldots, q$, where $u^s = \hat{\mu}(x^s)$.
- In particular, introduce a parametric family of policies $\tilde{\mu}(x, r)$. Then obtain r by

$$\min_r \sum_{s=1}^{q} \big\| u^s - \tilde{\mu}(x^s, r) \big\|^2$$

- The advantage: On-line play is greatly expedited
- Disadvantage: It does not deal well with adaptive control contexts/on-line replanning

Approximation of a Target Policy by Classification

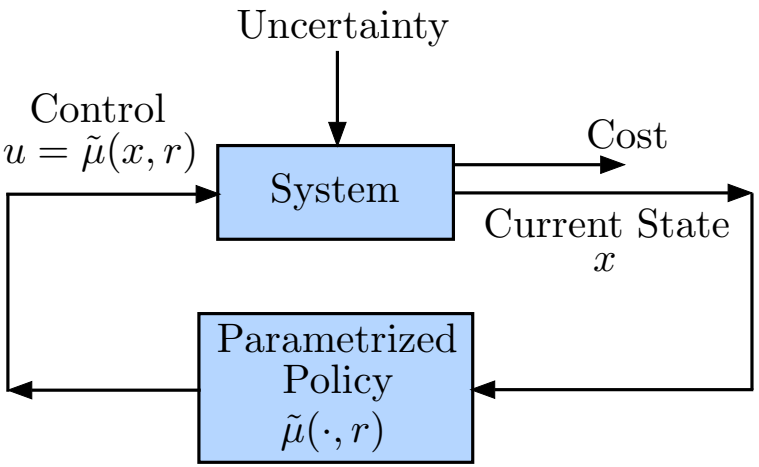
- If the control space is finite, an alternative approach is possible
- View a policy μ as a classifier: A function that maps x into a "category" $\mu(x)$
- Some classifiers introduce randomized policies
- Then the output of the classifier are the "control probabilities" where $\tilde{\mu}^u(x, r)$, $u = 1, \ldots, m$, gives the "probability" of using control u at state x
- Idea: Randomized policies have continuous components. This helps algorithmically
- The training can be done with classification software



A Fifteen-Minute Break

Catch our breath and think about issues relating to the first half of the lecture.

Direct Policy Optimization Framework



Select a policy by parametric cost optimization

- Each parameter value r defines a stationary (possibly randomized) policy $\tilde{\mu}(r)$ with cost $J_{\tilde{\mu}(r)}(x_0)$ starting from a fixed initial state x_0
- Determine r through the minimization

$$\min_r J_{\tilde{\mu}(r)}(x_0)$$

- More generally, determine r through the minimization

$$\min_r E\{J_{\tilde{\mu}(r)}(x_0)\}$$

where the $E\{\cdot\}$ is with respect to a suitable probability distribution of x_0

- Commonly solved by forms of incremental gradient and random search methods

Training by Random Search

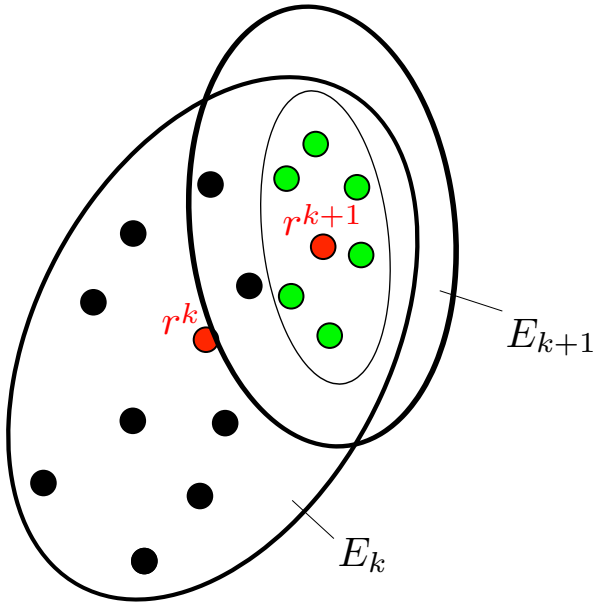
Random search methods apply to the general minimization $\min_r F(r)$

- They generate a parameter sequence $\{r^k\}$, aiming for cost reduction.
- Given r^k, points are chosen in some random fashion in a neighborhood of r^k, and some new point r^{k+1} is chosen within this neighborhood.
- In theory they have good convergence properties. In practice they can be slow, but this may be OK since the training is done off-line.
- They are not affected as much by local minima (e.g., like gradient-type methods).
- They don't require a differentiable cost function, and they apply to discrete as well as continuous minimization.
- There are many methods and variations.

Some examples

- Evolutionary programming.
- Tabu search.
- Simulated annealing.
- Cross entropy method.

Cross-Entropy Method - A Sketch



- At the current iterate r^k, construct an ellipsoid E_k centered at r^k.
- Generate a number of random samples within E_k. “Accept" a subset of the samples that have “low" cost.
- Let r^{k+1} be the sample “mean" of the accepted samples.
- Construct a sample “covariance" matrix of the accepted samples, form the new ellipsoid E_{k+1} using this matrix, add some more samples within E_{k+1} and continue.
- Limited convergence rate guarantees. Success depends on domain-specific insight and the skilled use of implementation heuristics.
- Simple and well-suited for parallel computation. Resembles a “gradient method".

They apply to the generic optimization problem $\min_{z \in Z} F(z)$

We take a bold and unusual step: Convert this problem to the stochastic optimization problem

$$\min_{p \in \mathcal{P}_Z} E_p\{F(z)\}$$

where

- z is viewed as a random variable.
- $\mathcal{P}_Z$ is the set of probability distributions over Z.
- p denotes the generic distribution in $\mathcal{P}_Z$.
- $E_p\{\cdot\}$ denotes expected value with respect to p.

The approximation approach

- We restrict attention to a parametrized subset of probability distributions $p(z; r)$, where r is a continuous parameter.
- We optimize over r.

Gradient Method for Approximate Solution of $\min_{z \in Z} F(z)$

Parametrization of the probability distributions

- We approximate the problem $\min_{z \in Z} F(z)$ with the parametrized problem

$$\min_r \; E_{p(z;r)}\{F(z)\}$$

- Often the parametrization is done using a soft max parametrization (to remove the prob. distribution simplex constraints)

- We use a gradient method for solving this problem:

$$r^{k+1} = r^k - \gamma^k \nabla \Big(E_{p(z;r^k)}\{F(z)\} \Big)$$

- Key fact: There is a useful formula for the gradient, which involves the gradient with respect to r of the natural logarithm $\log\big(p(z; r^k)\big)$.

The Gradient Formula (Reverses the Order of $E\{\cdot\}$ and ∇)

Assuming that $p(z; r^k)$ is a discrete distribution, we have

$$\begin{aligned}\nabla\Big(E_{p(z;r^k)}\big\{F(z)\big\}\Big) &= \nabla\left(\sum_{z\in Z} p(z; r^k)F(z)\right)\\ &= \sum_{z\in Z} \nabla p(z; r^k)F(z)\\ &= \sum_{z\in Z} p(z; r^k)\frac{\nabla p(z; r^k)}{p(z; r^k)}F(z)\\ &= E_{p(z;r^k)}\left\{\nabla\Big(\log\big(p(z; r^k)\big)\Big)F(z)\right\}\end{aligned}$$

Sample-based/incremental gradient method for solving $\min_r E_{p(z;r)}\big\{F(z)\big\}$

- At r^k obtain a sample z^k according to the distribution $p(z; r^k)$.
- Compute the sample gradient $\nabla\Big(\log\big(p(z^k; r^k)\big)\Big)F(z^k)$.
- Use it to iterate according to

$$r^{k+1} = r^k - \gamma^k\nabla\Big(\log\big(p(z^k; r^k)\big)\Big)F(z^k)$$

Policy Gradient Method - Discounted Problem

- Denote by z the infinite horizon state-control trajectory:

$$z = \{i_0, u_0, i_1, u_1, \ldots\}.$$

- We consider a parametrization of randomized policies $p(u \mid i; r)$ with parameter r, i.e., the control at state i is generated according to a distribution $p(u \mid i; r)$ over $U(i)$.
- Then for a given r, the state-control trajectory z is a random trajectory with probability distribution denoted $p(z; r)$.
- The cost corresponding to the trajectory z is

$$F(z) = \sum_{m=0}^{\infty} \alpha^m g(i_m, u_m, i_{m+1}),$$

and the problem is to minimize $E_{p(z;r)}\{F(z)\}$, over r.

- The gradient needed in the gradient iteration

$$r^{k+1} = r^k - \gamma^k \nabla\Big(\log\big(p(z^k; r^k)\big)\Big) F(z^k)$$

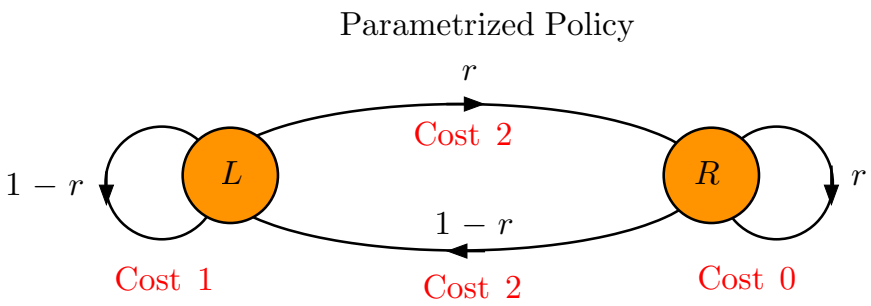
is given by

$$\nabla\Big(\log\big(p(z^k; r^k)\big)\Big) = \sum_{m=0}^{\infty} \log\big(p_{i_m i_{m+1}}(u_m)\big) + \sum_{m=0}^{\infty} \nabla\Big(\log\big(p(u_m \mid i_m; r^k)\big)\Big)$$

Unusual Aspects of the Policy Gradient Method

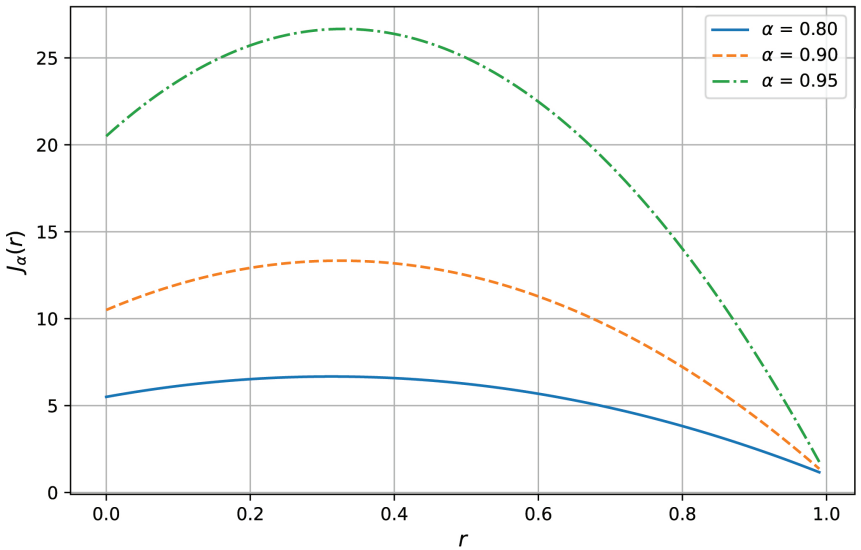
- It involves the cost function of the discounted problem, but not its gradient ... In fact the cost per stage g may be nondifferentiable!
- The problem solved is continuous (to allow the use of gradient-like methods), but the original can be discrete ...
- The randomized policy version of the problem typically has an extremely difficult structure (may have a concave cost function and many local minima)

Pitfalls of Policy Optimization - An α-Discounted Two-State Example Single Parameter r



Optimal DP policy: $L \to R$ and $R \to R$ (assuming $\alpha > 1/2$)

$r = 1$ yields the optimal DP policy



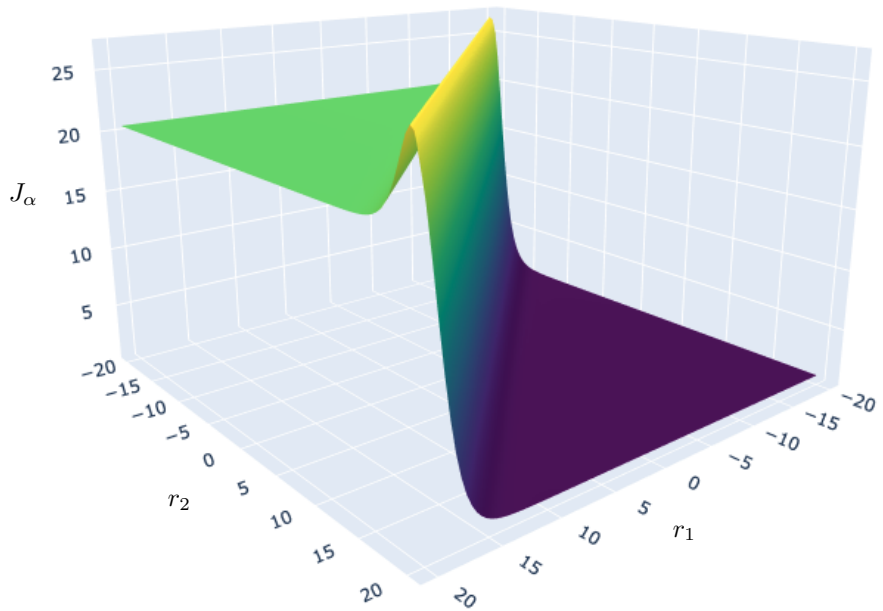
Parametrized cost function (concave!! with two local minima!!)

Reparametrization Using the Soft-Max Operation

- Idea: Eliminate the probabilistic constraints $0 \leq r \leq 1$ by reparametrization, so we can use an unconstrained gradient-like method
- In place of r and $1 - r$, we use two parameters r_1, r_2 that are unconstrained

$$r \mapsto \frac{e^{r_2}}{e^{r_1} + e^{r_2}}, \qquad 1 - r \mapsto \frac{e^{r_1}}{e^{r_1} + e^{r_2}}$$

- Then the optimization over (r_1, r_2) becomes unconstrained
- But $J(r_1, r_2)$ has no local minima! The optimal policy is $r = 1$, or $r_1 = -\infty$, $r_2 = \infty$



A Final Word on the Practical Uses of Direct Policy Optimization

Relations between policy gradient optimization and random search

- Both methods aim at cost improvement of a parametric policy
- Both methods are slow but that is OK because they are off-line training methods
- Policy gradient in practice involves randomization of the starting parameter, among others ... so it can be viewed as a form of random search

There is a generic difficulty with using a fixed policy on-line:

- It is all-training no on-line play. (Hence, fast on-line computation.)
- It does not adapt to changes in the problem's parameters.
- So approximation in policy space may not work well in adaptive control contexts.
- Also it does not yield the benefit of on-line lookahead minimization/Newton step.
- Approximation in value space, and rollout may work much better (e.g., in AlphaZero).

We Reached the End of the Last Lecture for this Course

Some words of caution

- There are challenging implementation issues in all approaches that involve off-line training, and there are no fool-proof methods.
- Training algorithms are not as reliable as you might think by reading the literature.
- Recognizing success or failure can be a challenge!
- The RL successes in game contexts are spectacular, but they have benefited from perfectly known and stable models and small number of controls (per state).
- Problems with partial state observation remain a big computational challenge.
- Approximation in value space may require long on-line computation.
- Approximation in policy space does not deal well with on-line replanning (approximation in value space does, and involves a Newton step).

On the positive side

- Approximation in value space and rollout are relatively simple and reliable.
- Massive computational power and distributed computation are a source of hope.
- Silver lining: We can address practical problems of unimaginable difficulty!
- There is an exciting long journey ahead!

Some Words of Relevance

Some old quotations ...

- The book of the universe is written in the language of mathematics. Gallileo
- Learning without thought is labor lost; thought without learning is perilous. Confucius (In the language of Confucius' day: learning $\approx$ knowledge; thought $\approx$ reflection)
- White cat or black cat it is a good cat if it catches mice. Deng Xiaoping

... and some more recent ones

- Machine learning is the new electricity. Andrew Ng
  (Electricity changed how the world operated. It upended transportation, manufacturing, agriculture and health care. AI will have a similar impact.)
- Machine learning is the new alchemy. Ali Rahimi and Ben Recht
  (We do not know why some algorithms work and others don't, nor do we have rigorous criteria for choosing one architecture over another ...)

Thank you for your attendance!

Good luck with your term papers and projects!