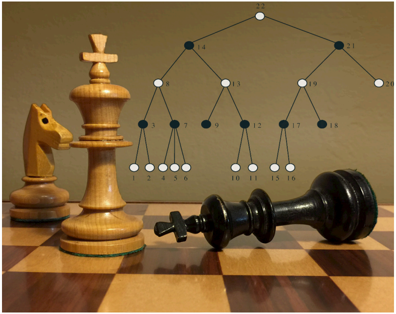
Feature-Based Aggregation and Deep Reinforcement Learning

Dimitri P. Bertsekas

Laboratory for Information and Decision Systems
Massachusetts Institute of Technology

Arizona State University

April 2018



AlphaZero

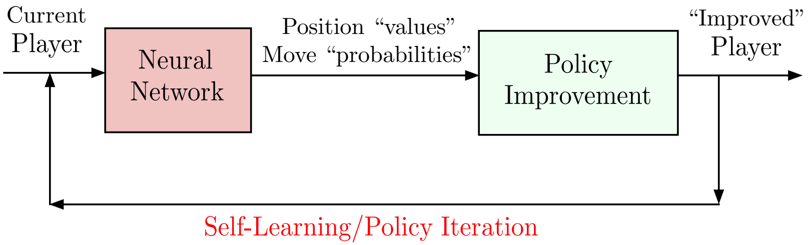
Plays much better than all chess programs

Plays different!

Learned from scratch ... with 4 hours of training!

Same algorithm learned multiple games (Go, Shogi)

AlphaZero was Trained Using Self-Generated Data



AlphaZero implements a form of policy iteration/approximate DP method

- Generates a sequence of players/policies, each implemented by a deep neural net
- A player's games are used to train an "improved" player (self-learning)
- The neural net of a player/policy provides at any position: the "value" of the position, and a "probabilistic ranking" of the possible moves
- The games of a player are generated by Monte-Carlo Tree Search (MCTS, a form of randomized multistep lookahead)
- Training uses a form of regression
- AlphaZero bears similarity to earlier works, e.g., TD-Gammon (Tesauro,1992), but is more complicated because of the MCTS and the deep NN

DP/RL: A CLASSICAL AND UNIVERSAL METHODOLOGY

Exact DP applies (in principle) to a very broad range of optimization problems

- Deterministic <—-> Stochastic
- Combinatorial optimization <—-> Optimal control w/ infinite state/control spaces
- One decision maker <—-> Two player games
- ... BUT is plagued by the curse of dimensionality and need for a math model

Approximate DP/Reinforcement Learning

- Overcomes the difficulties of exact DP by using:
  - Approximation (to reduce dimension)
  - Simulation (in place of a math model)
- Can be used in a very broad range of challenging/large scale problems
- Has proved itself in many fields ...
- ... BUT implementation is a challenge/art and success is not guaranteed
- Still there is theory that guides the art

A Summary

Some History

- 1950s-60s: Bellman (DP), Shannon (chess), Samuel (checkers)
- 80s-early90s: Approximation and simulation-based methods: Barto/Sutton [TD(λ), AI-DP connection], Watkins (Q-learning), Tesauro (backgammon, self-learning)
- 1990s: Rigorous analysis, mathematical understanding, first books
- Late 90s-Present: Rollout, Monte-Carlo Tree Search, Deep Neural Nets, Model Predictive Control

Methodology

- Math framework is DP (plus function approximation, training by simulation)
- Approximations in value space and in policy space (compact/low-dimensional, feature-based parametric architectures)
- Supervised vs unsupervised learning (using external vs self-generated data)
- No dominant method. Some ideas are solid and some are heuristic
- Success depends on finding the right mix of implementation ideas, and using massive computational power
- The AlphaZero program combines in a skillful way ideas that have been known since around 2005

Purpose of this Talk

Selectively survey the state of the art with focus on:

- Approximate policy iteration
- Neural network implementations
- Aggregation

Describe the relevant contributions of neural networks:

- Provide an approximation architecture for the cost function of a policy
- Automatically construct the features of the architecture using self-generated data
- Use in neural network-based policy iteration

Describe the feature-based aggregation methodology, and how it can be used in combination with neural nets

References

Survey paper

Bertsekas, “Feature-Based Aggregation and Deep Reinforcement Learning: A Survey and Some New Implementations," Lab. for Information and Decision Systems Report, MIT, April 2018; http://arxiv.org/abs/1804.04577

DP/RL Book references

- Bertsekas and Tsitsiklis, Neuro-Dynamic Programming, 1996
- Sutton and Barto, Reinforcement Learning, 1998 (2nd ed. on-line, 2018)
- Bertsekas, Dynamic Programming and Optimal Control: 4th edition, 2017

My latest theoretical monograph on DP

Bertsekas, Abstract Dynamic Programming: 2nd edition, 2018

Relations and Terminology in RL/AI and DP/Control

RL uses Max/Value, DP uses Min/Cost

- Reward of a stage = (Opposite of) Cost of a stage.
- State value = (Opposite of) State cost.
- Value (or state-value) function = (Opposite of) Cost function.

Controlled Markov chain terminology

- Agent = Controller or decision maker.
- Action = Control.
- Environment = System.

Methods terminology

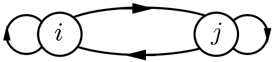
- Learning = Solving a DP-related problem using simulation.
- Self-learning (or self-play in the context of games) = Solving a DP problem using simulation-based policy iteration.
- Planning vs Learning distinction = Solving a DP problem with math model-based vs model-free simulation.
- Prediction = Policy evaluation.

Outline

Discounted Infinite Horizon Problem

Transition probabilities $p_{ij}(u)$

Cost $\alpha^k g(i, u, j)$



Controlled Markov Chain

A Markov chain with states $1, \ldots, n$, and control u

- $p_{ij}(u)$: Transition probability from i to j under u
- $\alpha^k g(i, u, j)$: Cost of the kth transition; $\alpha \in (0, 1)$: discount factor

Policy (or feedback controller) μ: Maps each state i to a control $\mu(i)$

- Total cost of μ starting at i_0: $J_\mu(i_0) = E\left\{\sum_{k=0}^{\infty} \alpha^k g\big(i_k, \mu(i_k), i_{k+1}\big)\right\}$
- Optimal cost starting at i_0: $J^*(i_0) = \min_\mu J_\mu(i_0)$
- Optimal policy μ^*: Satisfies $J_{\mu^*}(i) = J^*(i)$ for all i

Basic Theory

Bellman's equation for J^*

$$J^*(i) = \min_u \sum_{i=1}^{n} p_{ij}(u)\big\{g(i,u,j) + \alpha J^*(j)\big\}, \qquad \text{for all } i$$

Optimal cost at i = $\min_u$ E{1st stage exp. cost + optimal cost of remaining stages}

Policy evaluation (Bellman) equation for the cost function J_μ of a given policy μ

$$J_\mu(i) = \sum_{i=1}^{n} p_{ij}(\mu(i))\big\{g(i,\mu(i),j) + \alpha J_\mu(j)\big\}, \qquad \text{for all } i$$

Policy improvement principle

Given a policy μ and its evaluation J_μ, we can obtain an improved policy $\hat{\mu}$ through

$$\hat{\mu}(i) = \arg\min_u \sum_{i=1}^{n} p_{ij}(u)\big\{g(i,u,j) + \alpha J_\mu(j)\big\}, \qquad \text{for all } i$$

We have $J_{\hat{\mu}}(i) \leq J_\mu(i)$ for all i

Exact and Approximate Policy Iteration (PI)

Exact policy iteration is successive policy improvement:

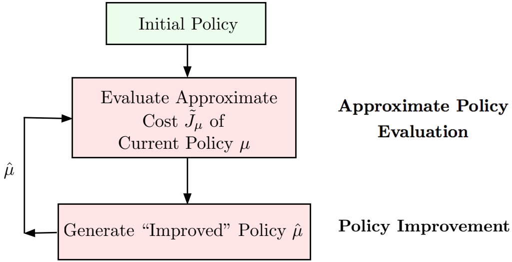
$$\mu_0 \;\Rightarrow\; \mu_1 : \text{improved policy over } \mu_0 \;\Rightarrow\; \mu_2 : \text{improved policy over } \mu_1 \;\Rightarrow\; \cdots$$

We have $J_{\mu_k} \to J^*$.

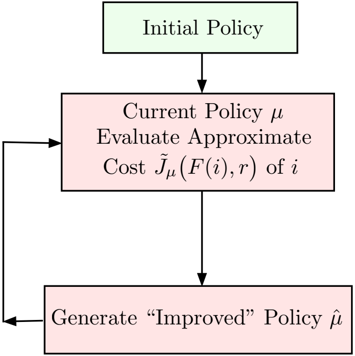
Approximate policy iteration is policy improvement w/ approximate evaluation:

$$\mu_0 \;\Rightarrow\; \mu_1 : \text{“improved" policy over } \mu_0 \;\Rightarrow\; \mu_2 : \text{“improved" policy over } \mu_1 \;\Rightarrow\; \cdots$$

“Converges" to optimum within an error bound [of order $O((1-\alpha)^2)$ or $O((1-\alpha))$].



Feature-Based Policy Evaluation



Approximation in a space of basis functions

$\tilde{J}_\mu\big(F(i), r\big)$: Feature-based parametric architecture

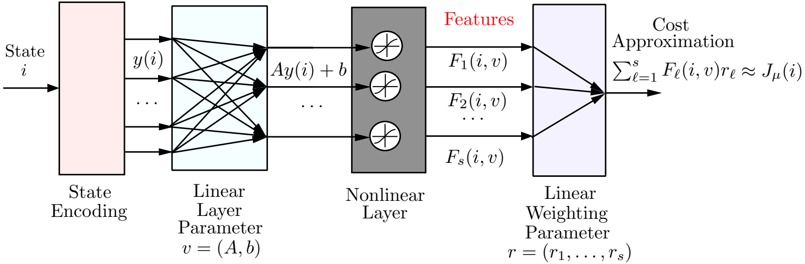
$F(i) = \big(F_1(i), \ldots, F_s(i)\big)$: Vector of Features of i

r: Vector of weights

Features F and weights r provide a lower-dimensional representation of J_μ

- The features can be viewed as basis functions
- The weights depend on μ (sometimes the features also)
- Critical question: How to find good features?
  - Handcrafted, based on a priori knowledge/intuition
  - Constructed from data, e.g., using a neural network (this is the BIG contribution of NNs)

NN-Based Evaluation of $\tilde{J}_\mu$ for a Given Policy μ

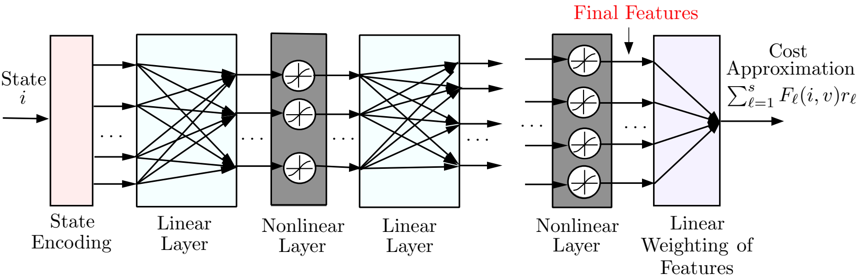


Generate state-cost samples (i_m, β_m), $m=1,\dots,M$, $\beta_m = J_\mu(i_m)$+"noise"

- Use nonlinear optimization/regression: Find (v, r) that minimizes

$$\sum_{m=1}^{M} \big(\tilde{J}_\mu(i_m, v, r) - \beta_m\big)^2$$

- Use of an incremental gradient method (also called SGD, backpropagation)
- Making the method work is an art (regularization, hot start, stepsize, etc)
- Universal approximation theorem
- To generate the cost samples: We simulate the Markov chain under μ
- We can use alternative regressions (e.g., based on temporal differences, etc)



A deep NN just has many layers

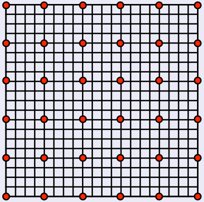
- Can be viewed as providing a “hierarchy of features"
- The last set of features is the one used in the cost approximation
- More “sophisticated" features with each stage, fewer weights needed (?)
- Sampling and training is the same as in single layer nets
- Is deeper better? Tesauro’s and subsequent backgammon implementations used one nonlinear layer!
- For our purposes, deeper is better. There are fewer final features in deep NNs

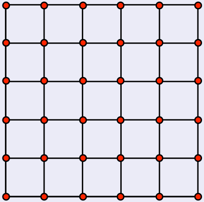
Basic Principles of Aggregation

An old idea: Problem approximation (rather than algorithm approximation)

- Group "similar" states together and represent them as a single state
- Approximate the original DP problem with a fewer-state DP problem, called aggregate problem
- Solve the aggregate problem and "extend" its cost function to the original
- The aggregate problem can be solved by exact DP and simulation-based methods

A simple example: Approximate a fine grid with a coarse grid

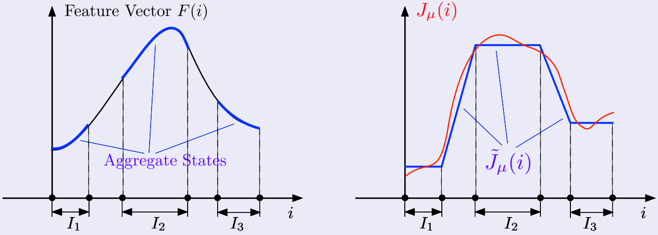




Another example (hard aggregation): Partition the state space into disjoint subsets, each viewed as a single "aggregate state"

Use a Feature Map $F(i)$ to Form the Aggregate DP Problem

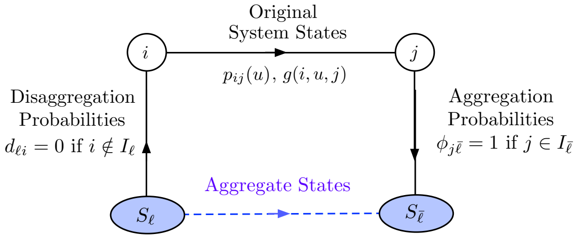
Idea: Group together states with “similar" features (i.e., small variation of F)



Aggregate states: Disjoint subsets $S_1, \ldots, S_q$ of state-feature pairs $(i, F(i))$

- System states j relate to the aggregate states according to “membership/interpolation weights" $\phi_{1\ell}, \ldots, \phi_{n\ell}$ (called aggregation probabilities)
- Each aggregate state S_ℓ relates to its “footprint", the set $I_\ell = \{i \mid (i, F(i)) \in S_\ell\}$, according to “importance weights" $d_{\ell 1}, \ldots, d_{\ell n}$ (called disaggregation probabilities)
- Constraints:
  - If $j \in S_\ell$ then $\phi_{j\ell} = 1$ (membership weight 1 for states in the footprint)
  - If $i \notin I_\ell$ then $d_{\ell i} = 0$ (importance weight 0 for states outside the footprint)

Aggregate DP Problem: Approximation through Features



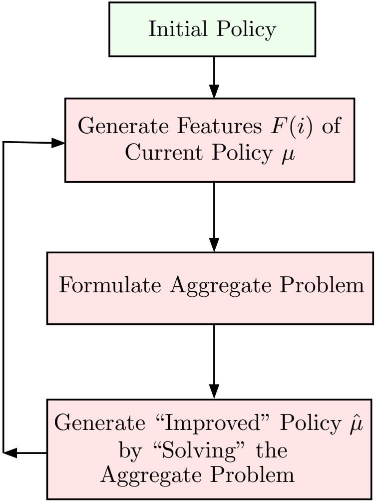
- States: Aggregate states plus two copies of the original system states
- Costs and transition probabilities: As shown
- Optimal costs: r_ℓ^* for aggregate state S_ℓ, $\tilde J_0(i)$ for left state i, $\tilde J_1(j)$ for right state j
- By Bellman's equation for the aggregate problem we have

$$\tilde J_1(j) = \sum_{\ell=1}^{q} \phi_{j\ell} r_\ell^*, \quad j = 1, \ldots, n \qquad \text{(piecewise linear)}$$

- Once we compute r_ℓ^*, we can obtain an "improved" policy

$$\hat\mu(i) = \min_{u \in U(i)} \sum_{j=1}^{n} p_{ij}(u) \left(g(i,u,j) + \alpha \sum_{\ell=1}^{q} \phi_{j\ell} r_\ell^* \right), \qquad i = 1, \ldots, n$$

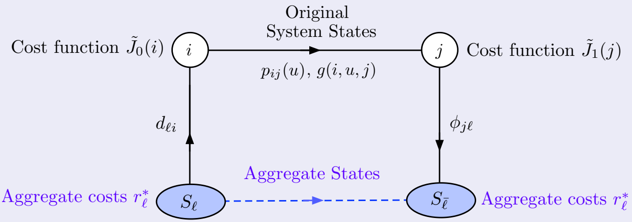
Aggregation-Based Approximate Policy Iteration



Use a Neural Network or Other Scheme
Possibly Include "Handcrafted" Features

Form the Aggregate States
Choose the Aggregation and Disaggregation Probabilities

Properties of the Aggregate Problem



- Aggregate problem lends itself to simulation if the original problem does
- r_ℓ^* is computable with exact/tabular methods, e.g., TD(λ), LSTD, LSPE, Q-learning

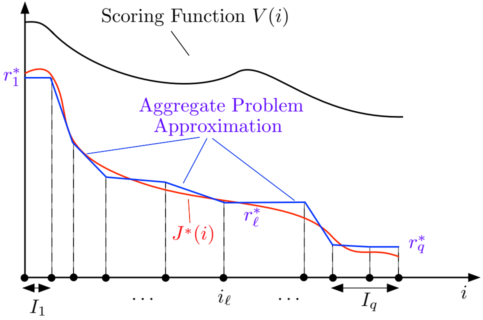
Intuition and analysis/error bounds suggest the following general strategy:

Use features that conform to J^*, i.e.,

$$J^*(i) \approx J^*(i') \implies F(i) \approx F(i')$$

Form aggregate states so that F varies little within their footprint

Using “Scoring" Functions



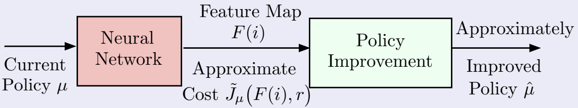
Suppose we have a function V with “similar form" to J^* (up to a constant shift)

- We can use V as a feature map and group states with similar values of V
- Each interval may contain one or multiple states
- Many intervals lead to more accurate but more time-consuming solution

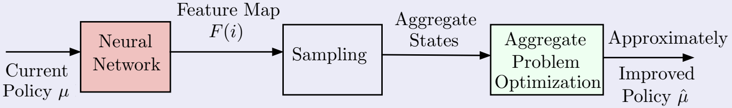
Extend this idea to a vector of scoring functions $V(i) = \big(V_1(i), \dots, V_s(i)\big)$

Approximate PI with Aggregation and Neural Nets

“Standard" NN-based PI



NN-based PI with aggregation



- Start with a training set of state-cost pairs generated using the current policy μ
- Evaluate μ using the NN; obtain a feature map F, and a sample of $\big(i, F(i)\big)$ pairs
- Construct aggregate states and a feature-based aggregate problem (essentially use F as a vector scoring function, possibly with some handcrafted features)
- Use as “improved" policy $\hat{\mu}$ the optimal policy of the aggregate problem
- More work for policy improvement, but may yield better “improved" policy

Concluding Remarks

- NNs resolve a major difficulty of approximate PI: Automatically extract features of the cost function of a policy
- Good features, once extracted can be used for other purposes, including aggregation. Deep NNs provide fewer final features, which favors aggregation
- Aggregation benefits from the solidity of exact DP algorithms

Some words of caution on approximate PI

- There are challenging implementation issues
  - Approximation architecture design using features
  - Sample design/explore well the state space
  - Training algorithms
  - Oscillations
  - Recognizing success or failure!
- The RL game successes are spectacular, but they have benefited from perfectly known and stable models and relatively small number of controls (per state)
- On the positive side, massive computational power together with distributed computation are a source of hope
- There is an exciting journey ahead ...

Thank you!