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!
Can be used to learn other games (Go, Shogi)
AlphaZero implements a form of approximate DP/policy iteration method

- Generates a sequence of players/policies
- A player/policy consists of: a value function/position evaluator, and a probabilistic move generator at any position
- Each player is trained with data generated by the preceding player (self-learning)
- Training uses forms of regression
- Examples are generated by Monte-Carlo tree search (a form of randomized multistep lookahead)
- AlphaZero bears similarity to earlier works, e.g., TD-Gammon (Tesauro, 1992)
Exact DP applies to a very broad range of optimization problems

- Deterministic/one decision maker $\leftrightarrow$ Stochastic/two player games
- Combinatorial optimization $\leftrightarrow$ Optimal control w/ infinite state/control spaces
- ... BUT is plagued by the curse of dimensionality and need for a math model

Approximate DP (Reinforcement Learning, Neuro-Dynamic Programming)

- Overcomes the difficulties of exact DP by using:
  - **Approximation** (to reduce dimension)
  - **Simulation** (in place of a math model)
- Can be applied to very 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-early 90s**: Approximation and simulation-based methods: Barto/Sutton [TD(\(\lambda\)), AI-DP connection], Watkins (Q-learning), Tesauro (backgammon, self-learning)
- **1990s**: Analysis, theoretical understanding, first books and research monographs
- **Late 90s-Present**: Rollout, Monte-Carlo tree search, deep neural nets

Methodology

- **Math framework is DP** (plus function approximation, training by simulation)
- Approximations in value space and in policy space (compact/low-dimensional, feature-based approximation architectures)
- Supervised learning vs unsupervised learning
- **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

DP/RL Book references
- Bertsekas and Tsitsiklis, Neuro-Dynamic Programming, 1996
- Sutton and Barto, Reinforcement Learning, 1998 (2nd ed. on-line, 2018)

My latest theoretical monograph on DP
Bertsekas, Abstract Dynamic Programming: 2nd edition, 2018
RL uses Max/Value, DP uses Min/Cost

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

Markov chain terminology

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

PI terminology

- **Learning** = Solving a DP 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

1. Exact and Approximate Policy Iteration
2. Approximate PI with Neural Nets
3. Feature-Based Aggregation
4. Feature-Based Aggregation with Neural Networks
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 $k$th transition; $\alpha \in (0, 1)$: discount factor

Policy $\mu$: A mapping of each state $i$ to a control $\mu(i)$

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

### Bellman’s equation for $J^*$

$$J^*(i) = \min_u \sum_{i=1}^{n} p_{ij}(u) \{ g(i, u, j) + \alpha J^*(j) \}, \quad \forall i$$

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

### Bellman’s equation to evaluate the cost function $J_\mu$ of a given policy $\mu$

$$J_\mu(i) = \sum_{i=1}^{n} p_{ij}(\mu(i)) \{ g(i, \mu(i), j) + \alpha J_\mu(j) \}, \quad \forall i$$

### Policy improvement principle

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

$$\hat{\mu}(i) = \arg\min_u \sum_{i=1}^{n} p_{ij}(u) \{ g(i, u, j) + \alpha J_\mu(j) \}, \quad \forall i$$

We have $J_{\hat{\mu}}(i) \leq J_\mu(i)$ for all $i$
Exact policy iteration is successive policy improvement:

\[ \mu_0 \Rightarrow \mu_1 : \text{improved policy over } \mu_0 \Rightarrow \mu_2 : \text{improved policy over } \mu_1 \Rightarrow \cdots \]

Converges to an optimal policy.

Approximate policy iteration is policy improvement w/ approx. 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 \( O((1 - \alpha)) \)]. Possibility of multistep lookahead and Monte Carlo tree search (not discussed here).
Feature-Based Policy Evaluation

\[ J_\mu(F(i)) : \text{Feature-based architecture} \]
\[ F(i) = (F_1(i), \ldots, F_s(i)) : \text{Vector of Features of } i \]

If \( J_\mu(F(i)) = \sum_{\ell=1}^s F_\ell(i)r_\ell \) \( (r_1, \ldots, r_s : \text{Scalar weights}) \)

it is a linear feature-based architecture

Approximation in a space of basis functions

Policy Improvement

Features provide a lower-dimensional representation/approximation of \( J_\mu \)

- The features can be viewed as basis functions
- The weights depend on \( \mu \) (sometimes the features also)
- How to find good features?
  - Handcrafted, tailored to the problem at hand
  - Automatically, e.g., using a neural network (this is the BIG advantage of NNs)
NN-Based Policy Evaluation for a Given Policy $\mu$

![Diagram of NN-Based Policy Evaluation](image)

Generate many state-cost samples $(i_m, \beta_m), m = 1, \ldots, M, \beta_m \approx J_{\mu}(i_m)$

- Use **nonlinear optimization/regression**: Find $(\nu, r)$ that minimizes
  \[ \sum_{m=1}^{M} \left( \tilde{J}_{\mu}(i_m, \nu, r) - \beta_m \right)^2 \]

- For this we may use an **incremental gradient method** (also called SGD)
- Making the method work is an art (regularization, hot start, stepsize etc)
- Simulation is used to generate the cost samples
- **Universal approximation** theorem
- Alternative regressions may be used (e.g., based on temporal differences, etc)
Use of Deep NNs

A deep NN just has many layers

- Simulation and training is the same as in single layer nets (the incremental gradient method is called backpropagation)
- Can be viewed as providing a “hierarchy of features"
- More “sophisticated” features with each stage, fewer weights needed (?)
- The last (and most sophisticated) set of features is the one used in the cost approximation
- **Is deeper better?** Tesauro’s and subsequent backgammon implementations used one hidden layer!
- **For our purposes, deeper is better.** There are fewer final features in deep NNs
Basic Principles of Aggregation

An old idea: Problem approximation

- Group “similar” states together and represent them as a single state
- Approximate the original DP problem with a lower-dimensional 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"
Aggregate DP Problem Based on a Feature Map $F(i)$

**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 with “membership/interpolation weights” $\phi_{1\ell}, \ldots, \phi_{n\ell}$ (called aggregation probabilities)
- Each aggregate state $S_\ell$ relates to its subset of states $I_\ell = \{i \mid (i, F(i)) \in S_\ell\}$ with “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 an aggregate state)
  - If $i \not\in I_\ell$ then $d_{\ell i} = 0$ (importance weight 0 for states outside an aggregate state)
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**: $r_\ell^*$ for aggregate state $S_\ell$, $\tilde{J}_0(i)$ for left state $i$, $\tilde{J}_1(j)$ for right state $j$

By Bellman's equation for the aggregate problem we have

$$\tilde{J}_1(j) = \sum_{\ell=1}^{q} \phi_{j\ell} r_\ell^*, \quad j = 1, \ldots, n \quad \text{(piecewise linear)}$$

Once we compute $r_\ell^*$, we can obtain an “improved” policy

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

- Aggregate problem lends itself to simulation if the original problem does.
- $r^*_\ell$ is computable with exact/tabular methods, e.g., TD($\lambda$), LSTD, LSPE, Q-learning.

- The approximation $\tilde{J}_1(j) = \sum_{\ell=1}^{q} \phi_{j\ell} r^*_\ell$ is piecewise constant if $I_1, \ldots, I_\ell$ cover the entire space (hard aggregation).
- Key fact for hard aggregation: If $J^*$ does not vary within each set $I_\ell$, $\tilde{J}_1 = J^*$.
- More generally: If $J^*$ varies little within each set $I_\ell$, $\tilde{J}_1 \approx J^*$.

General strategy:

Find features that conform to $J^*$, i.e., $F(i) \approx F(i') \implies J^*(i) \approx J^*(i')$ so we can find aggregate states where $J^*$ varies little.
Suppose we have a function $V$ with “similar form" to $J^*$ (up to a constant shift)

- We can use $V$ as a feature mapping and group states with similar values of $V$
- Sample the state space, compute the corresponding scores $V(i)$, partition the real line into intervals of small variation of the scoring function
- 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 $V(i) = (V_1(i), \ldots, V_s(i))$
Approximate PI with Aggregation and Neural Nets

“Standard” NN-based PI

![Diagram showing the process of "Standard" NN-based PI]

NN-based PI with aggregation

![Diagram showing the process of NN-based PI with aggregation]

- Start with a training set of state-cost pairs generated using the current policy \( \mu \)
- Evaluate \( \mu \) using the NN; obtain a feature map \( F \), and a sample of \((i, F(i))\) 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
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.

Approximate PI implementation is challenging. Issues with:
- Approximation architecture design using features
- Exploration/sample design
- Training algorithms
- Oscillations
- Recognizing success or failure!

Spectacular as the RL successes have been, they have benefited from the perfectly known and stable models of games.

A lot of data can be externally provided or self-generated, but many of the current algorithms are not data efficient and are often flaky.

Aggregation benefits from the solidity of exact DP algorithms, but must deal with quantization errors.

On the positive side, massive computational power together with distributed computation are a source of hope.

There is an exciting journey ahead ...
Thank you!