Evolution Strategies as a Scalable Alternative to Reinforcement Learning

強化学習勉強会
2017/04/05
関根 嵩之

概要

- 進化戦略:Evolution Strategy(ES)の適用

Q学習や方策勾配法などの強化学習の方法に新たに代わるものとして, MuJoCoやAtariなどの問題に対してESを適用

- 他の手法にはないESの様々な利点を示した

  - CPUの増加に伴い実現できる高い並列性
  - 報酬の分布, 行動の頻度や報酬遅延の影響を受けない
  - 一時的な割引を行わない, 価値関数を利用しない

概要

■ 顕著な実験成果

・[MuJoCoでの実験]
1440個のCPUコアを用いた並列化により、ESが3Dヒューマノイドロボットの学習を10分以内で実現

・[Atariでの実験]
A3Cよりも3倍, 10倍のデータを用いたが, 高い並列化やBack Propagationや価値関数を使用しないことより1時間で各ゲームで他の手法と十分競争できる結果を出した(A3Cの1日分の計算量)

Evolution Strategy (ES) [Rechenberg & Eigen 73]

- アルゴリズムの中で世代の良い生き残りの集団から次の集団の確率分布を決める、ということから“進化”戦略と言われている
- ブラックボックス関数の最適化手法の一つ
- 様々な種類が存在する
  - CMA-ES (Coveriance Matrix Adaptation Evolution Strategy)
    - ⟹ 最も広く使われている手法
  - NES (Natural Evolution Strategy)
    - ⟹ 今回使用するのはコレ

Natural Evolution Strategy (NES)

[Wierstra 08]

- NESでの用語と最適化問題としての表記

$F(\theta)$ ： 適応度 (fitness) ⇒ パラメータ θ の目的関数

θ ： 遺伝子(genotype)

$p_\psi(\theta)$ ： θ の確率分布

ψ ： $p_\psi(\theta)$ のハイパーパラメータ

$\eta(\psi)$ ： 目的関数(適応度)の期待値（ $\mathbb{E}_{\theta\sim p_\psi} F(\theta)$ ）

$\nabla_\psi \eta$ ： 目的関数の期待値の勾配

Natural Evolution Strategy (NES)

目的関数 F をパラメータ θ で数値最適化する代わりに

パラメータ θ の確率分布 $p_\psi(\theta)$ として

目的関数: $$\eta(\psi) = \mathbb{E}_{\theta \sim p_\psi} F(\theta)$$

を ψ についてSGD(Stochastic gradient decent)を用いて

最適化. その時の勾配は

勾配: $$\nabla_\psi \eta(\psi) = \mathbb{E}_{\theta \sim p_\psi} \{F(\theta) \nabla_\psi \log p_\psi(\theta)\}$$

Natural Evolution Strategy (NES)

- 強化学習の問題とした場合

$F(\cdot)$ ： 適応度 (fitness) ⇒ 確率的な累積報酬

$\theta + \sigma\epsilon$ ： 遺伝子(genotype)

π_θ ： $\theta + \sigma\epsilon$ の確率分布（ $\epsilon \sim N(0, I)$ ）

θ ： π_θ のハイパーパラメータ (方策のパラメータ)

$\eta(\theta)$ ： 目的関数(適応度)の期待値 （ $\mathbb{E}_{\epsilon \sim N(0,I)} F(\theta + \sigma\epsilon)$ ）

$\nabla_\theta \eta(\theta)$ ： 目的関数の期待値の勾配（ $\nabla_\theta \eta(\theta) = \frac{1}{\sigma} \mathbb{E}_{\epsilon \sim N(0,I)} \{F(\theta + \sigma\epsilon)\, \epsilon\}$ ）

Natural Evolution Strategy (NES)

- アルゴリズム

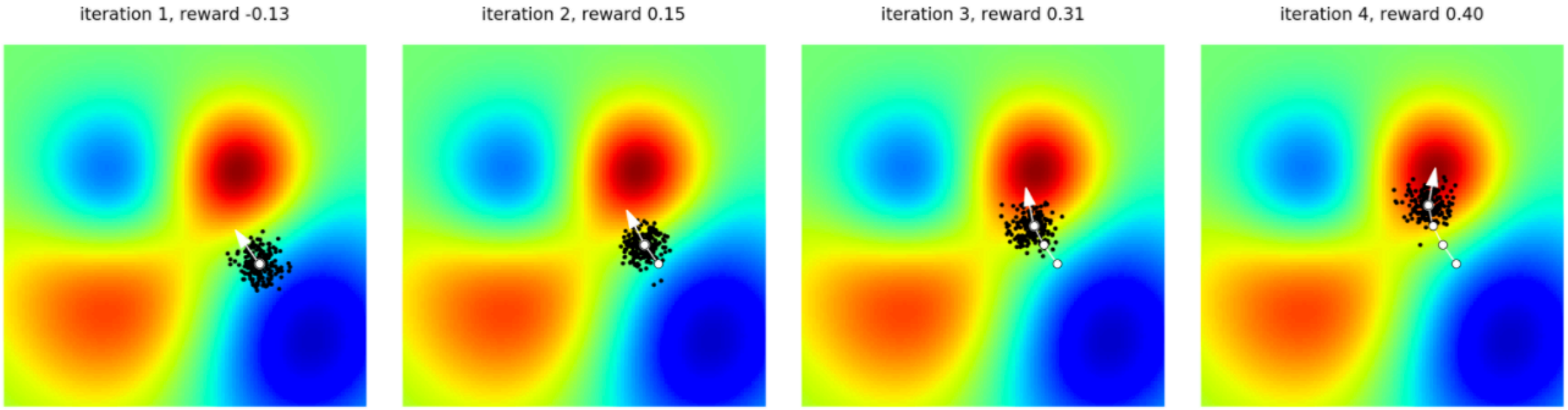
Algorithm 1 Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  Sample $\epsilon_1, \ldots \epsilon_n \sim \mathcal{N}(0, I)$
  Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$ for $i = 1, \ldots, n$
  Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{i=1}^{n} F_i \epsilon_i$
end for

・ガウスノイズを使ってn個のランダムな点を利用している
・有限差分法に近い

NES

- NESの最適化の様子



(参考: OpenAI blog https://blog.openai.com/evolution-strategies/)

NESの並列化

■ ESが並列化/スケールしやすい3つの理由

1. 一つのエピソードが一つのworkerで動作する
   ▶ worker同士のコニュニケーションの頻度が少ない

2. それぞれのworkerから得られる情報は1つのエピソードからの返り値であるスカラ値のみである
   ▶ 通信する情報が少ない

3. 価値関数近似が必要ではない
   ▶価値関数近似はシーケンシャルな処理を前提にしているため、1イテレーションで情報交換できるESとは違い、意味のある単位の情報を手に入れるために複数イテレーション必要としてしまう

NESの並列化

- 並列化した場合のアルゴリズム

Algorithm 2 Parallelized Evolution Strategies

Input: Learning rate α, noise standard deviation σ, initial policy parameters θ_0
Initialize: n workers with known random seeds, and initial parameters θ_0
for $t = 0, 1, 2, \ldots$ **do**
  for each worker $i = 1, \ldots, n$ **do**
    Sample $\epsilon_i \sim \mathcal{N}(0, I)$
    Compute returns $F_i = F(\theta_t + \sigma\epsilon_i)$
  end for
  Send all scalar returns F_i from each worker to every other worker
  for each worker $i = 1, \ldots, n$ **do**
    Reconstruct all perturbations ϵ_j for $j = 1, \ldots, n$
    Set $\theta_{t+1} \leftarrow \theta_t + \alpha \frac{1}{n\sigma} \sum_{j=1}^{n} F_j \epsilon_j$
  end for
end for

PGとESの違い

■ 方策勾配法 (Policy Gradient Method : PG)

・状態:s, 行動:a, 方策:π, 方策のパラメータ:θ, 累積報酬:R
・ここで、**行動が離散値, 方策πが決定的**とした場合

$$F(\theta) = R(\mathbf{a}(\theta))$$

の勾配はθについて連続ではなくなる

$$a_t = \pi(s;\theta)$$
$$\mathbf{a} = \{a_1, \ldots, a_T\}$$

PGは方策をθについて連続にして、方策勾配を推定できるようにしたいので、方策を**確率的**として**適切な分布から行動のサンプリングをすること**により**行動空間に摂動を与えている**といえる

PGとESの違い

■ PG
⇨行動空間に摂動を加えている

目的関数

$$F_{PG}(\theta) = \mathbb{E}_{\epsilon}\, R(\mathbf{a}(\epsilon, \theta))$$

方策

$\pi(s;\theta)$　▶決定的方策は不可

■ ES
⇨パラメータ空間に摂動を加えている

目的関数

$$F_{ES}(\theta) = \mathbb{E}_{\xi}\, R(\mathbf{a}(\xi, \theta))$$

方策

$\pi(s;\tilde{\theta})$　▶ 決定的方策も可

($\tilde{\theta} = \theta + \xi$)

非常に似ているが、PGは行動空間へ
ESはパラメータ空間へ摂動を加えていると違いがある

ESの利点

■[ステップの長さの影響を受けない]
報酬と行動の相関がなく,シンプルなモンテカルロ法(REINFORCE)でPGとESを勾配のバリアンスを比較

$$\mathrm{Var}[\nabla_\theta F_{PG}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})]\,\mathrm{Var}[\nabla_\theta \log p(\mathbf{a};\theta)]$$

$\sum_{t=1}^{T} \nabla_\theta \log p(a_t;\theta)$
(ステップの長さTに依存)

$$\mathrm{Var}[\nabla_\theta F_{ES}(\theta)] \approx \mathrm{Var}[R(\mathbf{a})]\,\mathrm{Var}[\nabla_\theta \log p(\tilde{\theta};\theta)]$$

(ステップの長さTに**依存しない**)

▶ 実用上PGではステップの長さを小さくするために工夫が必要だったが、
ESは行動が非常に先のステップまで影響する場合やが非常に長い場合にも
報酬の割引が必要ない

■[Back Propagationが不要]
エピソードごとの計算量がBPと比較して2/3, メモリ消費はそれ以下

実験1 [MuJoCo]の実験条件

■ [問題設定]
古典的な倒立振り子から2D hoppingやwalking gaitsなどの
近年のRLに用いられる問題に適用

■ [比較対象]
チューニング済みの方策勾配法のTRPO

■ [シミュレーションのエンジン]
MuJoCo

■ [方策のNNの構造]
非線形関数 tanh を用いた64ユニット隠れ層2つという多層パーセプトロンの同じ構造の方策をES, TRPOで学習

実験1 [MuJoCo]の実験結果

Table 1. MuJoCo tasks: Ratio of ES timesteps to TRPO timesteps needed to reach various percentages of TRPO's learning progress at 5 million timesteps.

ENVIRONMENT	25%	50%	75%	100%
HALFCHEETAH	0.15	0.49	0.42	0.58
HOPPER	0.53	3.64	6.05	6.94
INVERTEDDOUBLEPENDULUM	0.46	0.48	0.49	1.23
INVERTEDPENDULUM	0.28	0.52	0.78	0.88
SWIMMER	0.56	0.47	0.53	0.30
WALKER2D	0.41	5.69	8.02	7.88

- 環境とのTRPOの500万タイプステップ後の出す最終的な精度
- Table1には学習中でのサンプル複雑性のトレードオフをリストアップしたもの
- 概して, TRPOと比べて困難な状況下(HopperとWalker2d)のサンプル複雑性が10倍以下のペナルティーでの問題をとくことができた
- シンプルな環境下では, TRPOよりも3倍以上良いサンプル複雑性まで達成できた

実験1 [MuJoCo]の補足

■[行動の離散化の実験]
HoppingやSwimming taskでは, 各行動の次元をその領域で10分割をしてESに用いる行動を分割させた

▶ このような環境下では, 方策は他と比べて十分な探索ができなかった

▶[理由]
行動がパラメータの摂動に対して非常になめらかであるから

実験2[Atari]の実験条件

■ [問題設定]
ESの並列実装でOpenAI GymのAtari 2600

■ [比較対象]
A3C

■ [方策のNNの構造]
A3C(Mnih et al. 2016)で利用されていたのと同じ前処理と
フィードフォワードの同じCNNアーキテクチャを利用

実験2[Atari]の実験結果

■ [計算量]
A3Cでは1日で320millioin framesを使っていた発表されたものと同様な結果を出すのに,ESでは1billion frames計算できた。(同じNNの計算量)
▶これはESが
・Back Propagation を使用しない
・価値関数を使用しない
ことによる

■ [並列化]
Amazon EC2で720CPUを使って摂動パラメータの評価を並列化することにより学習に必要な時間をゲームあたり1時間程度に抑えることができた

■[最終結果比較]

A3Cの最終的な性能と比較したところ
ESが
・23ゲームでより良い精度
・28ゲームでは悪い結果
となった

結果一部

Game	DQN	A3C FF, 1 day	ES FF, 1 hour
Alien	570.2	182.1	994.0
Amidar	133.4	283.9	112.0
Assault	3332.3	3746.1	1673.9
Asterix	124.5	6723.0	1440.0
Asteroids	697.1	3009.4	1562.0
Atlantis	76108.0	772392.0	1267410.0

実験3[並列性]

■ 3Dヒューマノイドのスコア6000を達成するために使用したCPUのコア数と時間の関係

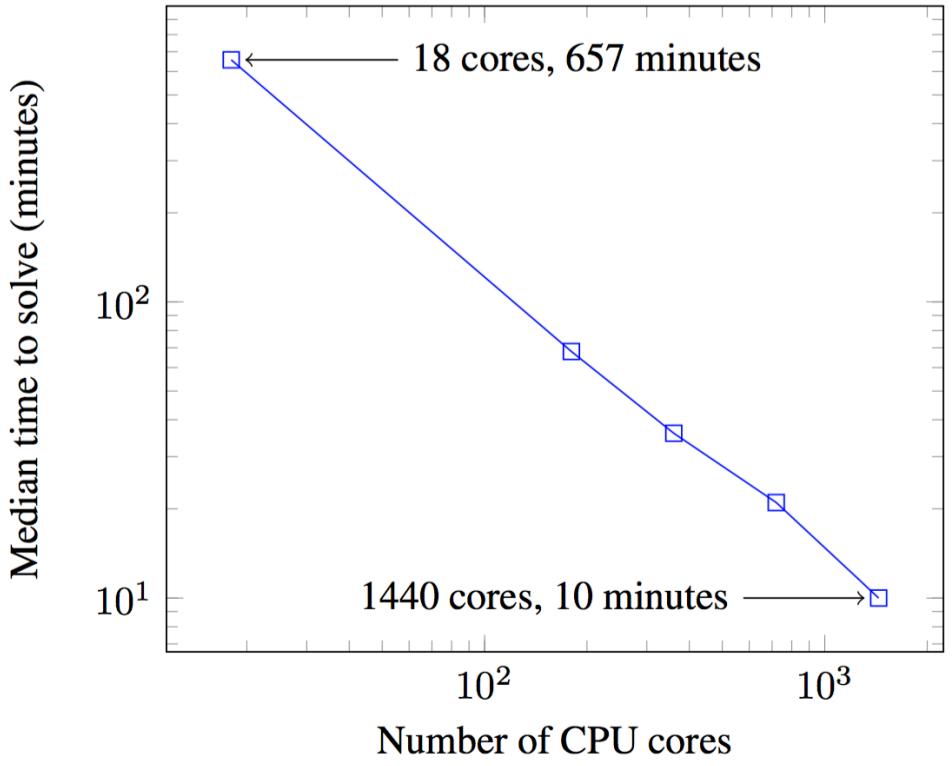


Figure 1. Time to reach a score of 6000 on 3D Humanoid with different number of CPU cores. Experiments are repeated 7 times and median time is reported.

実験4[フレームスキップ]

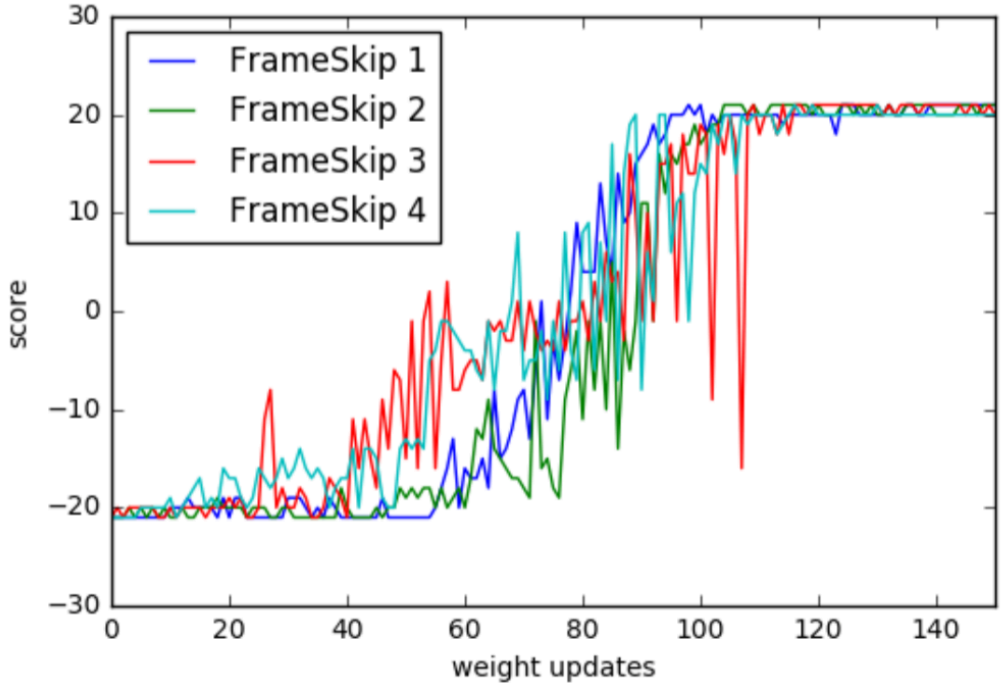


Figure 2. Learning curves for Pong using varying frame-skip parameters. Although performance is stochastic, each setting leads to about equally fast learning, with each run converging in around 100 weight updates.

■ AtariのPongで複数のフレームスキップで実験

■ フレームスキップパラメータの影響を受けにくいという結果が得られた
▶勾配の推定がエピソードの長さの影響を受けないため

※ フレームスキップ
シミュレーターが環境を実行するのに用いられる回数に対してエージェントは少ない頻度で行動を選択することが多く、その程度を表すハイパーパラメータ

実験5 [TRPOのバリアンスリダクション]

Table 2. Ratio of timesteps needed by TRPO without variance reduction to reach various fractions of the learning progress of TRPO with variance reduction at 5 million timesteps. (∞ means TRPO performance with variance reduction was never reached.)

ENVIRONMENT	25%	50%	75%	100%
HALFCHEETAH	3.76	4.19	3.12	2.85
HOPPER	0.66	1.03	2.58	4.25
INVERTEDDOUBLEPENDULUM	1.18	1.26	1.97	∞
INVERTEDPENDULUM	0.96	0.99	0.99	1.92
SWIMMER	0.20	0.20	0.22	0.14
WALKER2D	1.86	3.81	5.28	7.75

- バリアンスリダクション使わない場合のTRPOのパフォーマンス調査
- 評価方法は実験1と同様
- バリアンスリダクションを使わないTRPOはESと同様な性能をとなったが、ESよりも並列が困難であるため

まとめ

- ESがQ学習や方策勾配法のような普及した強化学習手法と比較しても複数の**魅力的な特徴**を持つことを実験を通して示した

- 魅力的な特徴は以下である
    1. 並列化に非常にすぐれている.
       これにより低下したデータ効率を補うことができる
    2. 行動の頻度や遅延報酬の影響を受けない
    3. 時間割引や価値関数推定も必要ではない
    4. 実装が用意

今後の研究

- 長時間報酬が得られない問題、複雑な報酬構造などの問題などのRLに適していないとされる問題に対してESを適用する
- meta-learningとかlearning-to-learnへの適用
- 勾配を使わないという性質を十分に活用するために, ESと高速で低性能のNN実装を組み合わせる

Appendix

実験2[Atari]の補足

Game	DQN	A3C FF, 1 day	ES FF, 1 hour
Alien	570.2	182.1	994.0
Amidar	133.4	283.9	112.0
Assault	3332.3	3746.1	1673.9
Asterix	124.5	6723.0	1440.0
Asteroids	697.1	3009.4	1562.0
Atlantis	76108.0	772392.0	1267410.0
Bank Heist	176.3	946.0	225.0
Battle Zone	17560.0	11340.0	16600.0
Beam Rider	8672.4	13235.9	744.0
Berzerk	NaN	1433.4	686.0
Bowling	41.2	36.2	30.0
Boxing	25.8	33.7	49.8
Breakout	303.9	551.6	9.5
Centipede	3773.1	3306.5	7783.9
Chopper Command	3046.0	4669.0	3710.0
Crazy Climber	50992.0	101624.0	26430.0
Demon Attack	12835.2	84997.5	1166.5
Double Dunk	-21.6	0.1	0.2
Enduro	475.6	-82.2	95.0
Fishing Derby	-2.3	13.6	-49.0
Freeway	25.8	0.1	31.0
Frostbite	157.4	180.1	370.0
Gopher	2731.8	8442.8	582.0
Gravitar	216.5	269.5	805.0
Ice Hockey	-3.8	-4.7	-4.1
Kangaroo	2696.0	106.0	11200.0
Krull	3864.0	8066.6	8647.2
Montezuma's Revenge	50.0	53.0	0.0
Name This Game	5439.9	5614.0	4503.0
Phoenix	NaN	28181.8	4041.0
Pit Fall	NaN	-123.0	0.0
Pong	16.2	11.4	21.0
Private Eye	298.2	194.4	100.0
Q*Bert	4589.8	13752.3	147.5
River Raid	4065.3	10001.2	5009.0
Road Runner	9264.0	31769.0	16590.0
Robotank	58.5	2.3	11.9
Seaquest	2793.9	2300.2	1390.0
Skiing	NaN	-13700.0	-15442.5
Solaris	NaN	1884.8	2090.0
Space Invaders	1449.7	2214.7	678.5
Star Gunner	34081.0	64393.0	1470.0
Tennis	-2.3	-10.2	-4.5
Time Pilot	5640.0	5825.0	4970.0
Tutankham	32.4	26.1	130.3
Up and Down	3311.3	54525.4	67974.0
Venture	54.0	19.0	760.0
Video Pinball	20228.1	185852.6	22834.8
Wizard of Wor	246.0	5278.0	3480.0
Yars Revenge	NaN	7270.8	16401.7
Zaxxon	831.0	2659.0	6380.0