CSE-599-W Reinforcement Learning — Due: 11:29 am on May 28, 2020

Homework 3: Tetris Player

Instructor: Byron Boots — *Submitted by Colin Summers and Liyiming Ke*

1 The Tetris Game

We are playing the Tetris game on a 21x10 board. The game is one shot, i.e. the agent sees only the current piece to place. The game is simplified such that the action space contains only the x-axis and the orientation of the piece to place, therefore excluding techniques like the T-Spins. The order of Tetriminos to fall is randomized. It's worth noticing that the game we are playing is different from what humans usually play. The later usually uses a 7-bag Random Generator to come up with the next 7 Tetriminos, which seems to make the game easier.

2 Our Approach

Following [2], we implemented a CMA-ES (Covariance Matrix Adaptation - Evolution Strategy) algorithm to learn to play Tetris. We assume that the "preference" of a state is a linear function of its features and attempts to optimize the agent's scores by learning the linear weights.

2.1 The Cost Function

If the game is not finished, it yields **1** point as reward for each line cleared in that turn.
When the game ends, it always returns **0**.

We chose our cost function such that the sum of the reward is the true objective that we want to maximize. We debated whether to give a small "living" award, i.e. yield 0.1 for each turn survived. Eventually we ruled out this idea because evolution strategies exploring in parameter space is usually less sensitive to local gradients and incentives than certain deep reinforcement learning algorithms.

2.2 Representation of state

Feature extractors are all in `features.py`.

Our state contains the board configuration *and* the placement of current piece. Following [2], we extract 8 features given the board and the current action: number of holes in the board, the landing height of the current piece, the row transitions, the column transitions, the well depth, the eroded piece, the hole depth and the row holes. This set of features is denoted as "BCTS" features [1]. We chose this set because it is verified in [2] to be able to clear 35M lines.

We experimented with alternative featurizer: the simple 4 features (aggregate height of columns, number of holes, bumpiness of walls and number of completed lines in one move) as suggested by [6] who cleared more than 1M lines with these features albeit on Tetris with 7-bag Random Generator.

2.3 CMA-ES

CMA-ES is a black box optimization algorithm. It maintains a centroid for the parameters to optimize. At each iteration, it samples λ offsprings of parameters, combine the best μ to update the centroid. The name of CMA-ES comes from the fact that it uses an evolution path to adapt its covariance matrix, i.e. the update takes into account not only the current generation but also successful offsprings in past generations.

We strictly follow the CMA-ES tutorial [3] in our implementation and setting of all parameters. Therefore, our implementation differs from [2] in 1) ours updates $\boldsymbol{p}_{\mathrm{c}} \leftarrow (1 - c_{\mathrm{c}})\, \boldsymbol{p}_{\mathrm{c}} + h_\sigma \sqrt{c_{\mathrm{c}}\,(2 - c_{\mathrm{c}})\, \mu_{\mathrm{eff}}} \langle \boldsymbol{y} \rangle_{\mathrm{w}}$, note the inclusion of h_σ and 2) $\boldsymbol{C} \leftarrow (1 + c_1 \delta\,(h_\sigma) - c_1 - c_\mu \sum w_j)\boldsymbol{C} + c_1 \boldsymbol{p}_c \boldsymbol{p}_{\mathrm{c}}^\top + c_\mu \sum_{i=1}^{\lambda} w_i^\circ \boldsymbol{y}_{i:\lambda} \boldsymbol{y}_{i:\lambda}^\top$ note the weight of last covariance, the rank-1 update and rank-μ update. The effect shall be minor though, as h_σ frequently evaluates to 1, making 1) equivalent to [3] and weights in covariance update are tunable parameters.

We adopt the suggestion in [2] to apply weight normalization on all sampled offsprings. Note this requires also changing the update rules of the original CMA-ES to use $\frac{\mathbf{m}^{t+1} - \mathbf{m}^t}{\sigma^t}$ in place of $\langle \boldsymbol{y} \rangle_{\mathrm{w}}$ where $\langle \boldsymbol{y} \rangle_{\mathrm{w}} = \sum_{i=1}^{\mu} w_i \boldsymbol{y}_{i:\lambda}$.

To learn to be a good Tetris player, we let the CMA-ES starts with a centroid at origin point with sigma set to 0.5. For each iteration we sampled 10 offsprings. For each offspring agent we evaluate its performance on 16×6 games using parallel programming and use the average score as its fitness value. Note that the order of Tetriminos in these games are the same across agents. We observed when running with 40 evaluations per agent (as opposed to 100), CMA-ES couldn't produce agents to clear more than 10k lines for us in 100 iterations.

2.4 DQN

We also implemented Deep Q Learning with Experience Replay [5] and Double Q Learning [4] . We used a 2-layer neural network with ReLU activations to predict the Q value of a state action pair. In our case, the extracted features ϕ encodes the state and the action already and our network predict $Q(s, a) = Q(\phi(s, a))$. At each training episode we rollout one game, either use the optimal action under Q or use the epsilon greedy exploration strategy. All steps in the rollout are stored in a replay buffer. At each episode we sample 1024 steps from the buffer to update the neural network, minimizing the mean squared error between $Q(\phi(s, a))$ and $r + \gamma \mathbb{E}_{s'}[\max_{a'} Q'\big(\phi(s', a')\big)]$ where Q' is the "target" neural network introduced by Double Q Learning, its parameters are copied from Q every fixed episodes.

3 Experiments

Approach	Features	Avg lines cleared in 100 games	Highest lines cleared
CMA-ES	BCTS	14194.0	55086
CMA-ES	Simple4	1138.42	5069
DQN	BCTS	33.14	214
DQN	Simple4	152.17	464

Note that we stopped all algorithms after 24 hours. At termination, CMA-ES with Simple4 stopped improving its performance. CMA-ES with BCTS ran only 13 iterations and its performance was still improving. DQN with BCTS features took longer to run than DQN with Simple4 features. DQN with Simple4 features converged but DQN with BCTS features.

References

[1] Simón Algorta and Özgür Şimşek. The game of tetris in machine learning. *arXiv preprint arXiv:1905.01652*, 2019.

[2] Amine Boumaza. How to design good tetris players. 2013.

[3] Nikolaus Hansen. The cma evolution strategy: A tutorial. *arXiv preprint arXiv:1604.00772*, 2016.

[4] Hado V Hasselt. Double q-learning. In *Advances in neural information processing systems*, pages 2613–2621, 2010.

[5] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8(3-4):293–321, 1992.

[6] Lee Yiyuan. Tetris ai-the (near) perfect bot. 2013.