BSc Project

Learning to Play Tetris using the Covariance Matrix Adaptation Evolution Strategy

January 12, 2016

Contents

1 Abstract

ABSTRACT

2 Nomenclature

2.1 Abbriviations

CMA-ES, Covariance Matrix Adaption Evolution Strategy

CE, Cross-Entropy

2.2 Symbols

m, the mean of a distribution.

M, the matix used as second argument to the gaussian distribution, which describes the shape of the distribution. Usually, when a sample from a gaussian distribution is drawn, it's written as: $x \sim \mathcal{N}\left(\mu, \sigma^2\right)$, however in (Hansen, 2015), μ and σ is used for other purposes, and therefore these symbols are reserved for this.

N, number of agents generated per generation/iteration.

γ, number of games played by each agent for evaluation.

μ, number of the population samples included in the next generation.

O, the function that estimates the performance of an agent/sample point.

Z, noise term added to the variance, specific to Cross-Entropy.

t, counter indicating the current generation/iteration.

w, the weight configuration of a Tetris playing agent(n-dimensional vector).

F, a feature of the Tetris game state. This is a function that maps a feature of the Tetris board to a real value.

V, a linear combination of feature functions and weight. The function used for evaluating a Tetris game state. The function is composed of a linear combination of feature sets and their associated weights, accepts a game state, and yields a single real-value.

s, a state of the Tetris game board.

x, an individual vector from a generation/iteration. Represented with an n-dimensional vector.

3 Introduction

On the topic of reinforcement learning, a widely used benchmark for learning algorithms are designing agents for playing the classical game of Tetris. Tetris is an appealing benchmarking problem due to it's complexity. The standard games plays on a board made from a grid that is 10 cells wide and 20 cells tall. As the game progress, differently shaped pieces fall from the top of the board. When a row on the board is fully occupied by pieces, the line is removed, all lines above it moved one line down and a score point is given to the player. If a cell above the 20 rows first is occupied, the game ends. The task of the player is to move and rotate the falling pieces in a way that yields the highest score before the game ends.

Tetris is indeed a hard task to computationally optimize, as the game has a very high number of board configurations estimated to be 10^{59} (Scherrer, 2009b). Because of this complexity, a common approach in the literature is to use *one-piece controllers*[1], such as described in Scherrer (2009a). These controllers are aware of only the current board state and the currently falling piece. Using these controllers, the search space is reduced to only looking at the current board, and the possible places to drop the piece. The game used for the benchmark is a simplified version of Tetris, in which controllers need only to decide in what column to drop the current piece, and what orientation the piece should have when dropped. Thus, the simplified version of Tetris differs from the original game mainly in two significant aspects. First, the controller is disallowed to move the piece horizontally while the piece is falling. Second, the controller has 'infinite' time to make its decision on where to drop the current piece. Thus, the controller cannot take advantage of moving the piece during the fall, but is not restricted by the time limitations. This is however a common way of benchmarking Tetris controllers (Scherrer, 2009b)

When the controllers decide which action to take, it will simulate each of the possible actions and choose the one that leads to the most favourable board state. To evaluate the board state, the controller uses a set of features that defines various qualities of the board, and associate a weight to each feature. This means that the efficiency of the controller is determined by the features the controller is aware of and how heavily they are weighted. This allows the controller with n features to be expressed as an n dimensional real-valued vector, with one dimension per feature, and the value in that dimension the weight. An often referred to controller is the Dellacherie's controller, as described in Scherrer (2009b). This controller takes six features of the board into account, seen in table 1 on page 5.

3.1 Goals of the thesis

Both the Cross-Entropy and the CMA-ES methods has been used in learning Tetris with *one piece controllers*, but as mentioned, to our knowledge, only little effort has been put into comparing the two methods. This thesis will explore how the two methods compare against each other under similar conditions. Therefore, we will use a set of features among those commonly used, and compare how the two optimizing algorithms differ. The goal however is not to find a controller that outperforms existing controllers, but only to investigate how the Cross-Entropy and CMA-ES differs when learning Tetris with similar controllers.

[1]Agents and controllers both refer to artificial players.

In this paper, we will explore how the two state-of-the-art optimization algorithms, Cross-Entropy and CMA-ES, differ when applied to the task of playing Tetris.

The SHARK library (Igel et al., 2008) contains a working implementation of the CMA-ES algorithm. However, the Cross-Entropy method is not present in the SHARK library and thus, a part of the thesis is to implement it ourselves according to the other researchers work. To document the soundness of the implemented CE method, we will replicate the experiment in (Thiery and Scherrer, 2009) and verify that we obtain the similar results.
Then, we will benchmark CMA-ES and CE against each other to determine if one yields better optimization results than the other.

4 Previous work

Over the time, numerous researchers has tried different feature sets and applied various optimizers to find the best possible Tetris controllers. The features used are typically ones that attempt to mimic the board conditions that would normally catch the attention of a human player, such as how high the overall pile of pieces is and how many holes the board has. In (Scherrer, 2009a) table 1, a table presents some feature sets used throughout various publications on the subject. In later works, many authors have had success with applying evolutionary stochastic search methods for tuning the weights of the feature sets towards efficient controllers. For the purpose of this thesis, we are in particular addressing the Cross-Entropy method described in detail in (Pieter-Tjerk De Boer and Rubenstein, 2014) and the Covariance Matrix Adaption Evolution Strategy (CMA-ES), described in (Hansen, 2015). The particular Cross-Entropy method applied is the one described in (Szita and Lörincz, 2006) as the "Noisy Cross Entropy Method".

Feature	**Description**
Landing height	The height of the piece when it lands
Eroded piece cells	Number of rows cleared in the last move times the number of bricks cleared from the last move
Row transitions	Number of horizontal cell transitions
Column transitions	Number of vertical cell transitions
Holes	Number of empty cells covered by a full cell
Board wells	Cumulative sum of cells to the depth of the board wells.

Figure 1: features of the Dellacherie controller

Currently, many researchers have proposed numerous feature sets and multiple optimization methods have been explored. A controller often referred to is the Dellacherie controller (Fahey, 2003). This controller was hand-tuned by trial and error, and did originally, on a regular non-simplified Tetris game, achieve an average of 660 000 lines. The same feature set (see figure 1) is often incoorporated in later works when optimizing controllers. An earlier feature set is the set proposed by (Bertsekas and Tsitsiklis, 1996) referred to as Bertsekas and Tsitsiklis features. In 2006, Szita and Lőrincz (Szita and Lörincz, 2006) applied the Cross-

Entropy method using the Bertsekas and Tsitsiklis features. They report that using no noise, their controller converged at 300 000 lines on average. The best result reported in (Szita and Lörincz, 2006) is when decreasing noise is applied, in which the controller's score exceeded 800 000 lines. However, in a later paper, using Dellacherie, Bertsekas and two selfdefined features achieved 35.000.000 lines ±20% (Scherrer, 2009b).

Creating a Tetris-controller is a NP-complete problem, where we want to find a strategy which maximizes the average score. Most researchers utilize three general approaches to create policies.

- Handwritten controllers
- Reinforcement learning approaches
- Optimization algorithms

But as seen in (Scherrer, 2009b), a combination of the methods can yield good results, where Thiery and Scherrer employed Dellacherie's handwritten policy and used CE to optimize it.

5 Scope and limitations

The experiments will be carried out on the simplified version of Tetris using the MDPTetris software found at (MDPTetris, 2009), which is the same simulator used by Scherrer et al. in (Scherrer, 2009a) among other authors in various papers. This software already have the well known feature sets implemented, so we will not ourselves extend any of the features. The source code of the Tetris simulator is used as-is, and is therefore not altered prior to running the experiments. For comparing the optimizers, the SHARK (Igel et al., 2008) library will be used. This library already contains an implementation of the CMA-ES optimizer, but lacks the Cross-Entropy. Therefore, a part of this thesis will be to implement and document the Cross-Entropy method in Shark.

6 Objective function

The optimizing algorithms used in this thesis both attempts to optimize a certain function. In this case, the optimization function aims to develop the best possible Tetris playing agent. Thus, the value of the objective function describes an estimated performance of the input agent. The objective function serves as an abstraction to the Tetris emulator that will evaluate the agent by letting the agent play a pre-configured number of games. The objective function that estimates the performance of an agent is later referred to as O.

When the Tetris simulator plays Tetris, the internal decision process of the Tetris controller is configured with a set of parameters which remain fixed across a single game. These parameters are the weights associated with each feature function that is considered by the controller. The features F_i each map a state s of the game to a real value. An overview of the exact mappings can be seen in table 1 in (Scherrer, 2009a). How much each of these mappings should affect the final evaluation of the board is determined by w_i denoting the weight of the

i-th feature. Finally, the function to assess the value of current board state $V(s)$, with n features functions present, can be expressed as:

$$V(s) = \sum_{i=1}^{n} w_i F_i(s)$$

Let S be the set of all states that each possible action can lead to from the current state s. The controller will then for each reachable state $gameState_i \in S$ evaluate the state by $O(s_i)$. The chosen action is the one that yields the state of the highest value. The performance of the controller is hence directly tied to the features and weights in the evaluation function. To adjust these controllers, one can either change the set of applied feature functions, or as the optimization algorithms will do, change the weighting of the features. For the experiments, the feature sets remain fixed, and the task of the optimization algorithm applied is to tune the set of weights in order to maximize the performance of the controller.

In summary, the objective function accepts a vector of values for each weight and configures an agent with the weights corresponding to the entries in the input vector. It then plays γ games with the agent and reports the mean score.

7 Optimizers

In this thesis, the Cross-Entropy method and the Covariance Matrix Evolution Strategy are compared. Both of the methods fall into the category of *stochastic optimization* methods. These methods are useful for optimization problems that have no gradient. The optimization functions aim to optimize the parameter set $\mathbf{x}$ for the objective function O.

$$\hat{\mathbf{x}} = arg\ \underset{\mathbf{x}}{max}\ O(\mathbf{x})\ O : \mathbb{R}^n \to \mathbb{R}$$

In these methods, the optimizing algorithm uses a family of parametric distributions, and maintain a mean m along with other parameters to search the best possible solution for the objective function. In the case studied in this thesis both the CMA-ES and Cross-entropy methods use a Gaussian distribution to sample solutions to the objective function. Hence, both of the functions aim to find a mean m and an $n \times n$ matrix M[2], such that when a vector $\mathbf{x}$ is sampled by $\mathbf{x} \sim \mathcal{N}(m, M)$, then $O(\mathbf{x})$ is likely to yield preferable results.

The algorithms work iteratively, such that the mean and variance of the distribution is altered for each iteration t. The algorithms start by initializing the parameters either at random or some fixed point. A common configuration is setting the mean to all zeros and the standard deviation to the identity matrix. Thus, for the first iteration $t = 0$, a configuration could be:

$$m_0 = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \quad M_0 = \begin{bmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \end{bmatrix}$$

Where the subscript notes that the values occur in iteration 0.

[2]In (Hansen, 2015), σ is used for step-size in CMA-ES, so M is instead introduced as an arbitrary $n \times n$ matrix in its place.

In each iteration, the algorithms sample λ vectors and evaluate their fitness against the objective function. When each of the solutions are evaluated, they are ordered according to their fitness:

$$\{\mathbf{x}_1, \ldots, \mathbf{x}_N\} \text{ Such that } f(\mathbf{x}_1) \geq f(\mathbf{x}_2), \ldots, f(\mathbf{x}_{N-1}) \geq f(\mathbf{x}_N)$$

The mean and standard deviation for the next iteration, that is m_{t+1} and M_{t+1} is then updated usually by considering the best of the ordered solutions. How exactly these parameters are updated is individual for each method and can be seen in the following sections.

8 CE (Cross Entropy)

CE is described through many papers in slightly different ways. Using similar format as in the paper (Thiery and Scherrer, 2009).

This method uses a Gaussian distribution and attempts to find distribution parameters that yields good candidates for the objective function O.

The Cross-Entropy method starts with an initial mean m and standard deviation M. The mean is usually an n dimensional vector set to:

$$m = \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix}$$

The variance is kept individual for each dimension, and is usually initialized as follows:

$$M = \begin{bmatrix} \sigma_1 & \ldots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \ldots & \sigma_n \end{bmatrix}$$

Where in this context, σ represents *standard deviation.*

The algorithm then works iteratively on generations of individual search points acting as candidate inputs for the objective function. In each generation, N vectors are sampled by $x_i \sim \mathcal{N}\left(m, M^2\right),\ i \in \{1, \ldots, N\}$. The vectors are all evaluated against the fitness function and ordered such that $O\left(x_1\right) \geq, \ldots, \geq O\left(x_N\right)$, and the μ best are chosen for updating the distribution parameters. The mean is updated as the centroid of the chosen vectors, and the variance is updated as the variance of the chosen vector in each dimension.

The pseudo code and details of the algorithm can be seen in figure 3 on page 11.

Noisy cross-entropy method

```
input
$O$ : The function that estimates the performance of a vector $x$
$(m_0, M_0^2)$: The mean and variance of the initial distribution
$N$ : The number of vectors sampled per generation/iteration
$\mu$: The number of offspring selected for the new mean
$Z_t$: The noise added to each generation/iteration

loop
    Generate $N$ vectors $x_1, x_2, \ldots, x_N$ from $\mathcal{N}(m_t, M_t^2)$
    Evaluate each vector using $O$
    Select the $\mu$ vectors with the highest evaluation
    Update $m_{t+1}$ of the $\mu$ best vectors
    Update $M_{t+1}^2$ of the $\mu$ best vectors + $Z_t$
end loop
```

Figure 2: The pseudo code for the Cross-Entropy algorithm

8.1 Input

The objective function
The function used to assess the value of a sampled vector. As described in the 'Optimizers' section, CE is a general stochastic iterative algorithm that tries to solve an optimization problem of the form (Thiery and Scherrer, 2009):

$$\hat{\mathbf{x}} = arg \max_{\mathbf{x}} O(\mathbf{x})$$

Where x corresponds to a given vector, and O is our actual objective function.

The mean and variance of the gaussian distribution
Here m_t is the mean and M_t^2 is the variance of the gaussian distribution $(m_t,\ M_t^2)$. More specifically this gaussian distribution is defined as

$$\mathcal{N}(m_t, M_t^2)$$

Where t denotes the current iteration.

The number of vectors
N is the number of vectors sampled in each generation.

The number of offspring
μ is the number of vectors which are used to compute the new mean, m_{t+1}, and variance, M_{t+1}^2, for next generation/iteration. These offspring vectors gets selected directly by taking μ vectors which got the best evaluation.

The noise factor
The noise factor, Z_t, is the amount of noise which is applied to the variance M^2 in iteration/-generation t. In general, noise is used to avoid the risk of a local optimum. There are different

kinds of noise settings, such as: no noise, constant noise and linear decreasing noise (Szita and Lörincz, 2006). When using no noise, Z_t is simply set to zero. When using constant noise, the same value is added to the variance M^2 in each iteration/generation. When using linear decreasing noise, Z_t is defined as $Z_t = max(5 - t/10, 0)$.

8.2 Loop

Sampling the population
The first step of the loop is to create the new generation consisting of N vectors. These vectors are sampled randomly within the distribution $x_i \sim \mathcal{N}(m_t, M_t^2)$.

Evaluating the population
After sampling the population, the algorithm needs to order the vectors to find the μ best vectors, each vector x_i , $i \in \{1, \dots, N\}$ is evaluated using O. The value from the objective function then yields the estimated performance of each individual.

Selecting the offspring
As each x_i has an assigned evaluation value, and the μ best vectors gets selected by taking the x_i vectors with the highest evaluation value.

Updating the distribution parameters
When updating the distribution parameters for the next iteration (m_{t+1},M_{t+1}^2), the mean is updated by computing the centroid of the μ best vectors. This is formally defined as:

$$m_{t+1} := \frac{\sum_i^{\mu} x_i}{\mu}$$

The variance M_{t+1}^2 is updated to match the variance of the μ best vectors, such that the variance in dimension i matches the variance of the μ in dimension i. This is formally defined as:

$$M_{t+1}^2 := \frac{\sum_i^{\mu} (x_i - m_{t+1})^T (x_i - m_{t+1})}{\mu} + Z_{t+1}$$

9 CMA-ES

This description is based on the tutorial written by Nikolaus Hansen (Hansen, 2015). The section will not in detail cover how the CMA-ES is derived, but rather how it deviates from Cross Entropy.

The CMA-ES operates on a general level much like the Cross Entropy method, but includes some features that increases the adaptability of the algorithm.

The CMA-ES, like the Cross Entropy, uses a Gaussian distribution to search for good solutions to O. Yet, in the second argument in the Gaussian distribution, the CMA-ES provides a covariance matrix. As the Cross Entropy method only provides a diagonal matrix of scalers it's restricted to only scaling the ellipsoid of equal density along the coordinate axes. The CMA-ES however, with a full covariance matrix, allows the ellipsoid to rotate arbitrarily in the search space.

While the Cross Entropy only considers the next population when updating the distribution parameters, the CMA-ES keeps some information from earlier generations. This allows the CMA-ES somewhat keep track of the evolution of the sampled vectors.

The CMA-ES also differs from Cross Entropy in how it evaluates the influence of the offspring vectors. As Cross Entropy weights all vectors equally when moving the mean. The CMA-ES, at least from the implementation in SHARK, has the option of taking a weighted combination of the offspring in order to bias towards the better vectors.

CMA-ES

```
input
$O$ : The function that estimates the performance of a vector $x$
$(m, C)$: The mean and variance of the initial distribution, where $C$ is the covariance matrix usually set to $C = I$
$N$ : The number of vectors sampled per generation/iteration
$\mu$: The number of offspring selected for the new mean

initialization
Set initial internal parameters

loop
    Sample new generation
    Evaluate each vector using $O$ and recombine
    Step-size control
    Covariance matrix adaption
end loop
```

Figure 3: The pseudo code for the Cross-Entropy algorithm

ALL CMA SPECIFIC BEOW THIS IS NOT DONE!

9.1 Input

The objective function
This serves the same purpose as in Cross Entropy (see page 9).

The mean and variance of the gaussian distribution (Adapt to CMA)
Here m_t is the mean and C^t is the variance of the gaussian distribution $(m_t,\ M_t^2)$. More specifically this gaussian distribution is defined as

$$\mathcal{N}(m_t, M_t^2)$$

Where t denotes the current iteration.

The number of vectors
N is the number of vectors sampled in each generation.

The number of offspring
μ is the number of vectors which are used to compute the new mean, m_{t+1}, and variance, M_{t+1}^2, for next generation/iteration. These offspring vectors gets selected directly by taking μ vectors which got the best evaluation.

9.2 Initialization

Set parameters

$$N, \mu, w_{i \dots \mu}, c_\sigma, d_\sigma, c_c, c_1, c_\mu$$

To their default values according to table 1 in (Hansen, 2015).
Set evolution path $p_\sigma = 0$, $p_c = 0$, covariance matrix $C = I$ and $t = 0$

9.3 Loop

Sample new generation
Sample new population of search points, for $k = 1, \dots, N$

$$\begin{aligned} z_k &\sim \mathcal{N}(0, I) \\ y_k &= BDz_k \sim \mathcal{N}(o, C) \\ x_k &= m + \sigma y_k \sim \mathcal{N}(m, \sigma^2 C) \end{aligned}$$

Evaluate each vector using O and recombine
Selection and recombination

$$\langle y \rangle_w = \sum_{i=1}^{\mu} w_i y_{i:N}, \text{where} \sum_{i=1}^{\mu} w_i = 1, w_i > 0$$
$$m \leftarrow m + \sigma \langle y \rangle_w = \sum_{i=1}^{\mu} w_i x_{i:N}$$

Step-size control

$$p_\sigma \leftarrow (1 - c_\sigma) p_\sigma + \sqrt{c_\sigma (2 - c_\sigma) \mu_{\text{eff}}} C^{-\frac{1}{2}} \langle y \rangle_w$$
$$\sigma \leftarrow \sigma \times exp \left(\frac{c_\sigma}{d_\sigma} \left(\frac{||p_\sigma||}{E||\mathcal{N}(o, I)||} - 1 \right) \right)$$

Covariance matrix adaption

$$p_c \leftarrow (1 - c_c)\, p_c + h_\sigma \sqrt{c_\sigma\, (2 - c_c)\, \mu_{\text{eff}}} \langle y \rangle_w$$

$$C \leftarrow (1 - c_c - c_\mu)\, C + c_1 \left(p_c p_c^T + \delta\,(h_\sigma)\, C \right) + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\mu} y_{i:\mu}^T$$

10 Hypothesis

CMA-ES IS BETTER
ROTATEABILITY
KNOWLAGDE OF OLDER GENERATIONS
RECOMBINATION TYPES

11 Experiment settings

This section will walk through our experimental setup, our verification of CE and and a analysis/discussion of the results.

11.1 MDP-Tetris

When running the experiments, the source code of the MDP-Tetris (MDPTetris, 2009) was used to emulate the Tetris games. The source code is accompanied with files that describe the various existing features. These files contains the identifiers of each feature to use, as well as two numbers respectively describing the agents reward function and how to evaluate a game over state. The number for the reward function has remained unchanged at 0 during all experiments. The "game-over" evaluation was for the Bertsekas feature set initially set to 0. Setting the "game-over" evaluation to 0 means that the agent will not distinguish between regular moves and moves that results in losing the game. When running the experiments with this setting, a large portion of the agents never exceeded a zero mean score. However, setting the value to -1, meaning that a "game-over" move yields $-\infty$ reward, none of the experiments got stuck on only zero scores. An example of the layout of the feature file can be seen in figure 4.

```
0      <- Describes the reward function
-1     <- Actions leading to game over is avoided at all cost
22     <- The policy contains 22 features
8 0    <- The feature with id 8 initially has weight 0
...    <- The remaining 21 features
```

Figure 4: Example of a file that describes a feature set.

11.2 Normalization of samples

As mentioned by some authors (Boumaza, 2009), the vector that describes the agent can very well be normalized such that the vector is a point that lies on the n-dimensional hypersphere.

The reason for this lies in the nature of the evaluation function. When the controller chooses an action, it will evaluate all the possible actions possible with the current piece. It will use the value function V of each state s_i and choose the state with the highest value from the value function. Thus, if the states are ordered such that:

$$V(s_1) > \cdots > V(s_N)$$

The agent then chooses the action that transitions from the current state to state s_1.
Since the value function assess the state by the following:

$$V(s) = \sum_{i=1}^{n} w_i F_i(s)$$

Then scaling the input of the agent, the weight vector w, by a number $a \in \mathbb{R},\ a > 0$ the assessment is changed by:

$$\begin{aligned}\sum_{i=1}^{n} a w_i F_i(s) &= a \sum_{i=1}^{n} w_i F_i(s) \\ &= aV(s)\end{aligned}$$

And the ordering remains:

$$aV(s_1) > \cdots > aV(s_N)$$

Thus the order of the value functions of each state does not change, and the same s_1 is still chosen for any $a \in \mathbb{R},\ a > 0$.

To verify this, the Tetris objective function was executed with the same vector and the same seed for the random generator with a scale $a \in \{0.1, 0.2, 0.3, \ldots, 9.8, 9.9, 10.0\}$, and the agent scored exactly the same for each scale.

This can be used in experiments for various reasons. As reported in (Boumaza, 2009), normalizing the samples will prevent CMA-ES from diverging in step size, and it can prevent loosing precision if the magnitude of weight vector becomes larger than feasible for the used floating point number and avoids size limitations.

11.3 Setup

When executing the experiments, various parameters each have an impact on the final result of the learning curve. Thus, the parameters are adjusted, first to match the experiments run by other researchers, and later to conduct as fair as possible comparisons between Cross-Entropy and CMA-ES.

The amount of vectors sampled in each generation N has a high impact on the algorithm performance. By setting N high, more policies are evaluated per iteration, and leads to a more thorough exploration of the search space. Thus the higher N increases the chances of finding a better mean for the next iteration. However, higher N also results in the need for

more evaluations per iteration. The goal for tuning this parameter is then to set N high enough to ensure exploration of good solutions, and yet low enough to avoid unnecessary evaluations.
In the implementation of CMA-ES from SHARK (Igel et al., 2008) , the algorithm itself determines the value of N according to the size of the search space. Cross-Entropy however, does not seem to have a general rule for this parameter, so this value is manually adjusted to fit the problem as well as possible.

As both of the optimizing algorithm uses a subset of the sampled vectors from a generation to update the distribution parameters, the number of offspring μ influences how the next generation is sampled. By setting the value too high, the algorithm risks ceasing to progress any further since the updated mean would be too close to the previous one to significantly make a difference. By setting the value too low, the risk of reaching a local optimum increases since the high-scoring agents might have reached their high performance by chance.
The CMA-ES itself manages setting μ and Cross-Entropy is set according to the problem. Most authors that uses Cross-Entropy for Tetris sets the offspring size to 10% of population size, that is $\mu = \lfloor 0.1 \cdot N \rfloor$.

The number of games, γ, is the number of games which each agent plays in each iteration. An agent's score is defined as the mean of the score of these γ games. We want this value low as possible, because as with the number of agents, N, The number of games, γ, is another major factor in the run-time of the algorithm. As Tetris is stochastic by nature, the score deviates a lot, even when the same agent with the same policy plays multiple games. Hence, when assessing the true performance of a policy it's rarely enough to play just few games. Thus, setting γ high increases the likelihood of correctly choosing the best agents, yet, it also causes longer run times of the experiments.

Specific to the Cross-Entropy method, most authors report that the performance of the algorithm increases dramatically when the sampling distribution is associated with a noise term. The different types of noise are described in section 8. The noise term is adjusted in order to prevent the algorithm from reaching a local optimum. The current research shows that noise terms of $Z_t = 4$ and $Z_t = max\,(5 - t/10)$ (Thiery and Scherrer, 2009) produces the best results. The constant noise (such as $Z_t = 4$) ensures that the algorithm never settles in a too small area from which it samples, and forces it to explore solutions that are further away from the mean. The further the algorithm progresses, the less noise is assumed needed, as the mean should approach a global optimum. to address this, the linear decreasing noise is applied as it will lower the noise term as the algorithm progresses.

For the various experiments, these parameters will be tuned for the specific purpose at hand. In the verification of the Cross-Entropy, the parameters are set to match those reported in similar papers (Thiery and Scherrer (2009), Szita and Lörincz (2006)). In the comparison of the two algorithms, the parameters will be set such that the Cross-Entropy operates under as similar conditions as CMA-ES, to ensure an unbiased comparison.

11.4 Assesement of controller performance

The performance of a one-piece controller has a very high variation, and is in other research verified to be exponentially distributed. As a result of the high variance of the controllers, the performance of single controllers are often presented along with a confidence interval for the estimated mean score of the controller. The estimated mean of the controllers score is calculated by:

$$\hat{m} = \sum_{i=1}^{N} x_i$$

Thus, the maximum likelihood estimation of the rate parameter[3] of the distribution is given by

$$\hat{\lambda} = \frac{1}{\hat{m}}$$

A confidence interval is found by the following:

$$\frac{2N}{\hat{\lambda}\chi^2_{1-\frac{\alpha}{2},2N}} < \frac{1}{\lambda} < \frac{2N}{\hat{\lambda}\chi^2_{\frac{\alpha}{2},2N}}$$

However, for these experiments, an approximation for a 95% confidence on lower and upper bound of the rate parameter λ is used:

$$\lambda_{low} = \hat{\lambda}\left(1 - \frac{1.96}{\sqrt{N}}\right)$$
$$\lambda_{upp} = \hat{\lambda}\left(1 + \frac{1.96}{\sqrt{N}}\right)$$

By this, the 95% confidence interval for the mean m is:

$$\frac{1}{\lambda_{low}} < m < \frac{1}{\lambda_{upp}}$$

When a controllers score is presented as "$s \pm p$" this means has an empirical mean score of s and a real mean that is with 95% likelihood within $s \pm p$.
ADD REFERENCE TO EXPONENTIAL DISTRIBTUTIONS

[3]Note that λ is in this context not, the population size but instead the rate parameter for the exponential distribution.

11.5 Verification of CE

Because the SHARK (Igel et al., 2008) library already contains an implementation of CMA-ES, but not an implementation of CE, we extended the library with our own implementation of the algorithm. This ensures as many similar conditions as possible for the two optimization algorithms as possible.
In order to verify the correctness of the implementation, we used the same experiments as used by Christophe Thiery and Bruno Scherrer (Thiery and Scherrer, 2009). These experiments were used be Thiery and Scherrer to verify their own CE implementation with various types of noise correction. Therefore, we will perform the same experiments to verify our own contribution to the SHARK (Igel et al., 2008) library, by trying to achieve the same results.

The setup is mirrored from the paper (Thiery and Scherrer, 2009), with 100 agents ($N = 100$) per iteration. The 10 best agents ($\mu = 10$) will be used to update gaussian distribution. After each iteration, an agent with the updated mean plays 30 games and the mean of these scores are recorded for the learning curve.
During evaluation each agent plays one game, that is $\gamma = 1$.

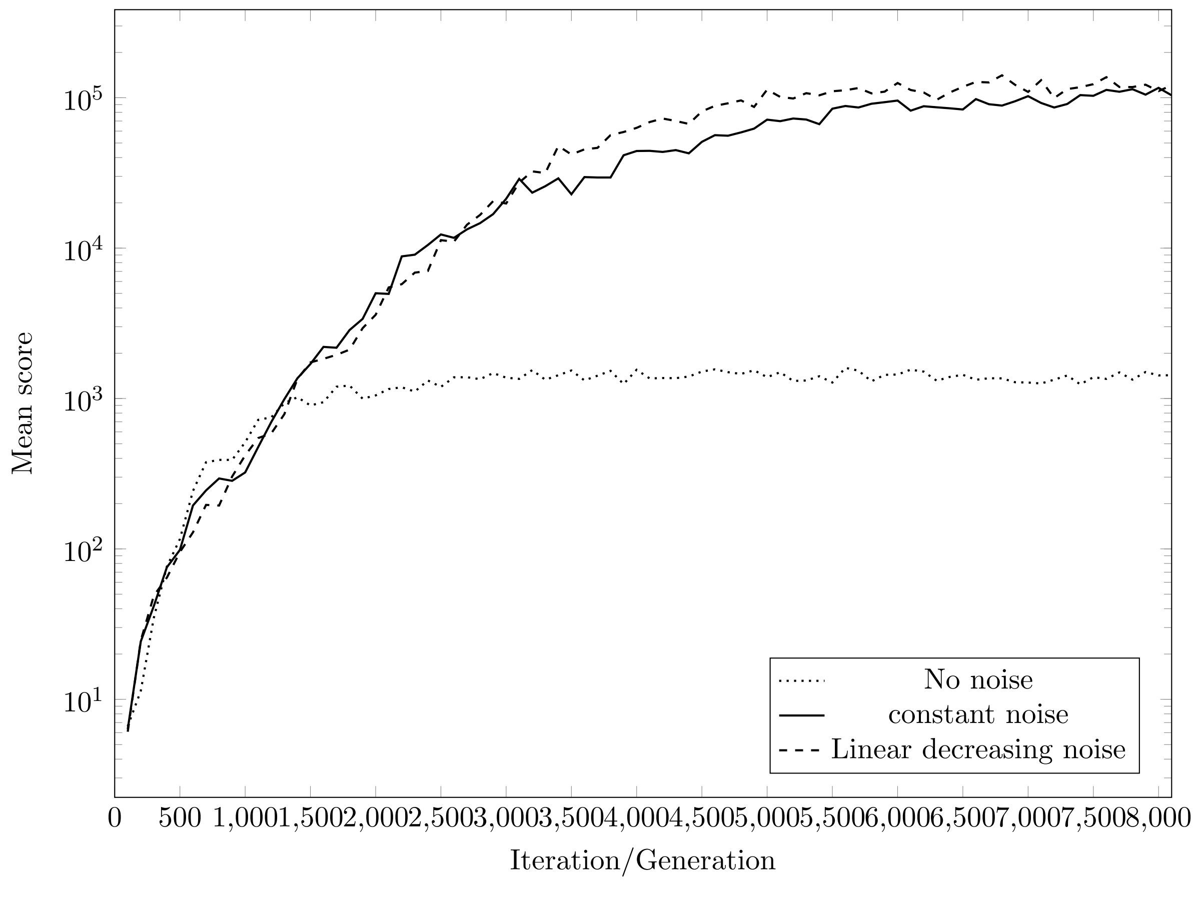

Figure 5: Cross-Entropy mean performance

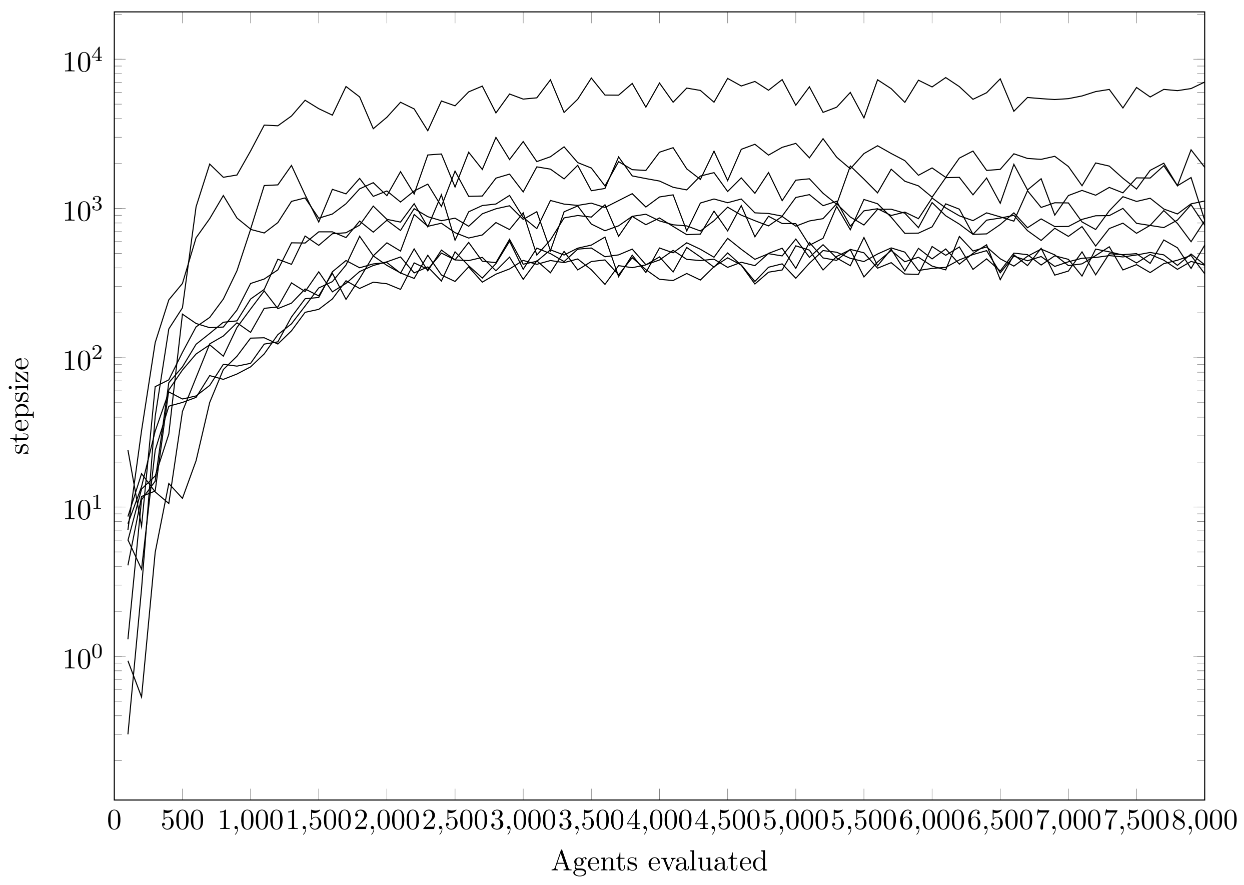

Figure 6: No noise

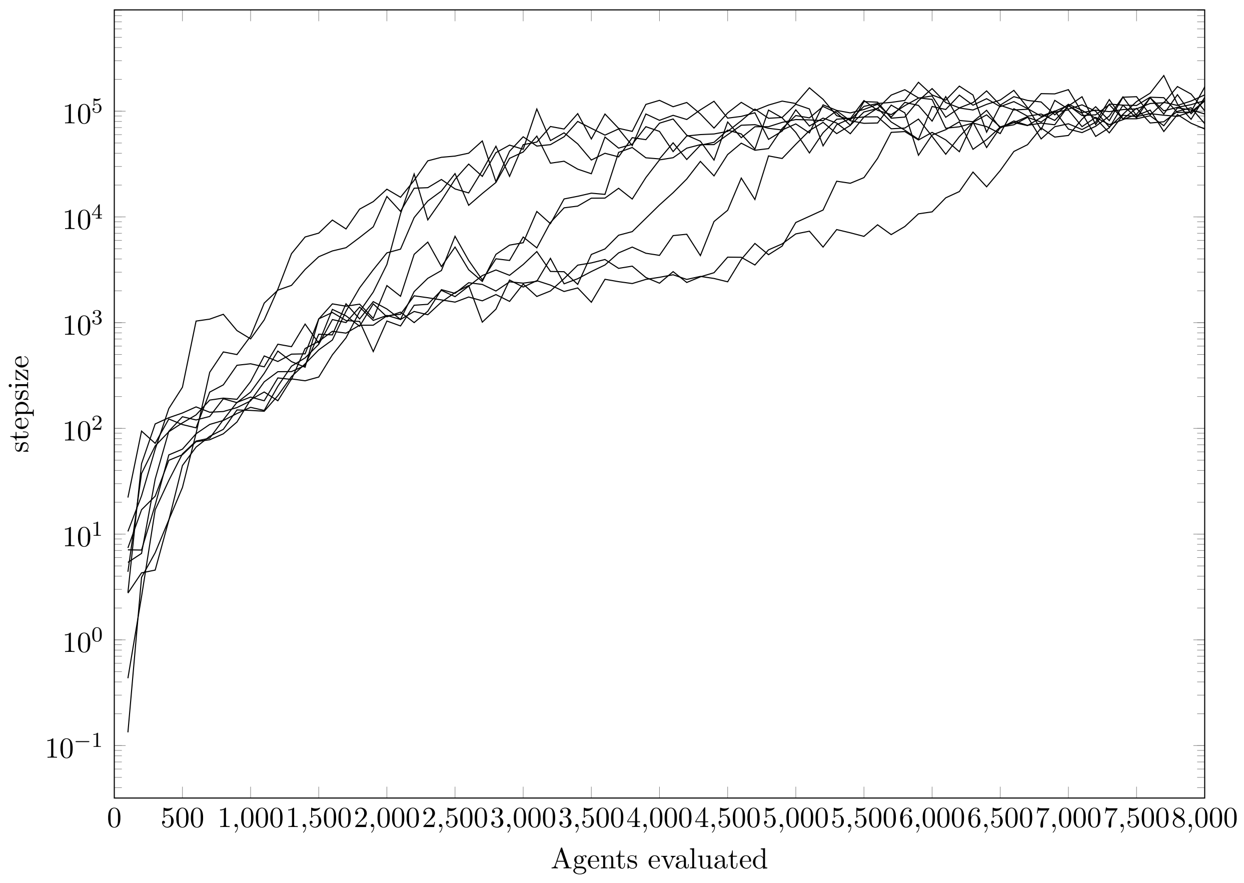

Figure 7: Constant noise

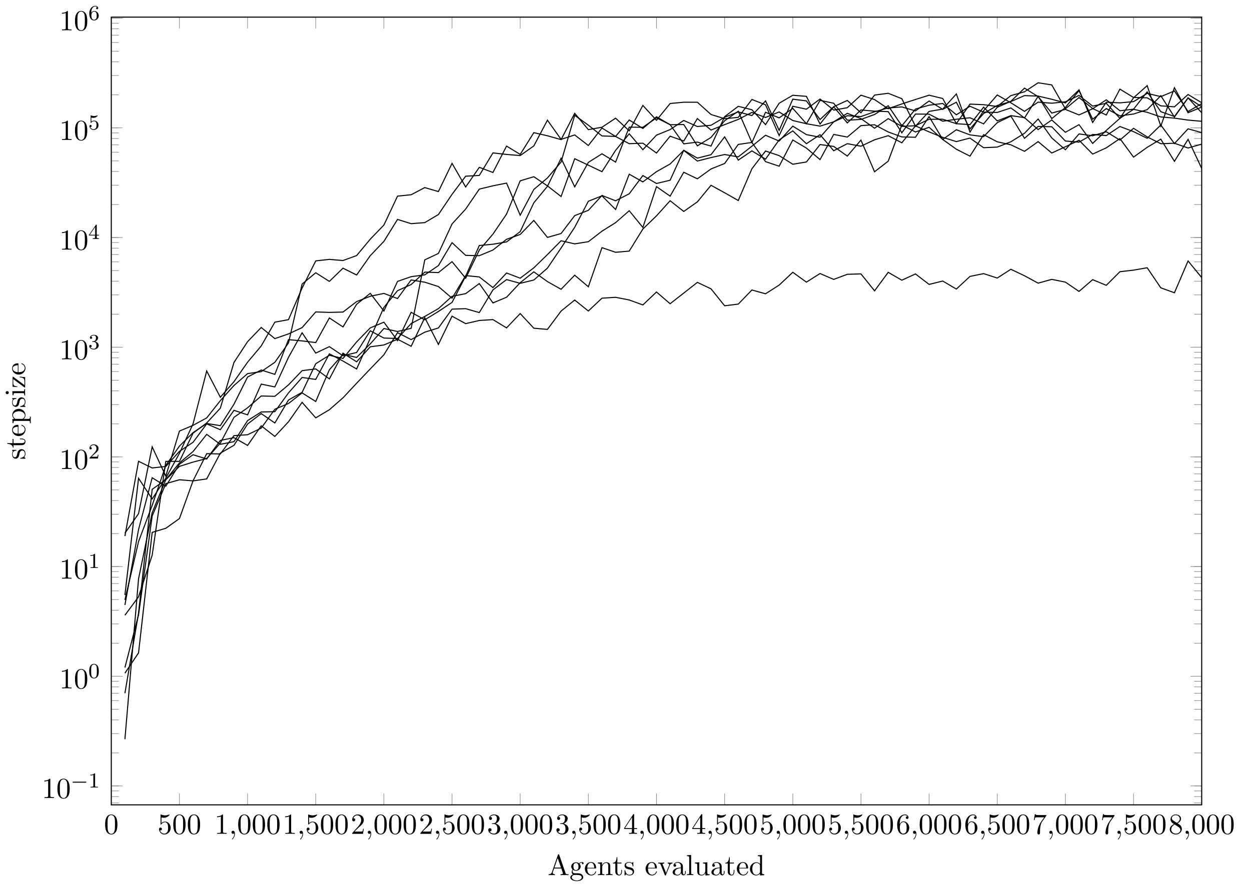

Figure 8: Linear decreasing noise

Figure 6, 7 and 8 shows 10 runs of each noise type. Figure 5 shows the mean graph for each of the noise types. The goal of these experiments were to replicate the experiments reported in (Thiery and Scherrer, 2009). As the results seen from our experiments to a high degree resemble those reported by Thiery et. al, we conclude that our Cross-Entropy implementation works similar to theirs.

When evaluating the score of the agent we also want to compute the confidence interval in verifying the implementation of CE. Since the population size is static for the mean agent, the confidence also becomes static for all experiments with 30 games, which is $\pm 36\%$. Compared to other papers, this confidence interval is similar (Scherrer, 2009b).
By looking at the individual graphs for the different noise types (Figure 6, 7 and 8), we get the following average scores.

Without noise (figure 6): The learning curve stabilizes after 1,500 agents evaluated. And as it can be observed the score variates much for the different executions between a score of 300 and 6,000 rows. This results in an average score of 1,400$\pm$36% rows.
Constant noise (figure 7): The 10 executions reaches equivalent performances at some point, with a score between 54,000 and 154,000. This results in an average score of 105,000$\pm$36% rows.

Linear decreasing noise (figure 8): Most of the executions of this noise type settles around 200,000. However, a single execution settled at a score of only around 5,000. The mean performance of this noise type yielded a score of 120,000±36%

Based on the mean graphs and confidence interval compared to other papers, we can hereby verify that our implementation of CE works as intended. Even though the experiments with linear decreasing noise in this case seems to outperform the constant noise, other runs with linear decreasing noise ended in a mean performance of only 90,000±36%. Yet, the constant noise is both from our own experiments, and described in other research, to be the most reliable noise type for reaching high scoring controllers (Scherrer, 2009b). Due to this, the constant noise is used in the benchmarking against CMA-ES.

11.6 Optimal settings for Cross Entropy

Other researchers run the Cross Entropy algorithm with population size of $N = 100$ and an offspring corresponding to 10% of the population size, resulting in $\mu = 10$. As it's not discussed why this exact setting is applied, various settings of the Cross-Entropy was run to asses the performance of other configurations. The experiments includes different population sizes $N \in \{10, 22, 50, 100, 200\}$ and offspring sizes of either 10% and 50%. A summary of the experiments can be seen in figure 10 on page 22.

N	μ	mean	Q1	Q2	Q3
10	10%	704.6	7.2	48.3	430.3
10	50%	9,272.5	149.6	7626.5	16,919.9
22	10%	35,841.6	20,391.9	42,045.5	48,464.6
22	50%	52,887.4	23,531.9	42,161.0	83,144.1
50	10%	95,623.1	82,738.9	93,388.9	111,351.5
50	50%	69,130.7	52,511.0	64,351.6	91,488.6
100	10%	115,868.7	84,368.5	122,238.5	146,457.0
100	50%	22,910.4	4,037.7	14,353.7	47,215.9
200	10%	85,181.7	45,201.5	96,803.1	117,578.0
200	50%	946.4	585.0	802.5	1,267.7

Figure 9: Cross Entropy configuration test

GRAPHS TO APPENDIX + COMMENT ABOUT THE QUANTILES?
The experiments with different population and offspring sizes does not seem to support a choice for any other configuration than the mostly commonly used $N = 100$ and $\mu = 10$.
However, with a configuration of $N = 50$ and $\mu = 5$ convergence is achieved faster. This means that the score limit is reached faster, which results in longer computation time, than the $N = 100$ and $\mu = 10$ configuration. In other words, the $N = 100$ and $\mu = 10$ configuration is therefore preferred since it takes shorter computation time and the end-result is similar compared to the $N = 50$ and $\mu = 5$ configuration, even though the latter configuration technically performs better.
ADD GRAPHS OF THE TWO DIFFERENT CONFIGURATIONS?

11.7 Step-size and lower bound for CMA

- WRITE ABOUT INTITIAL SIGMA

σ_0	mean	Q1	Q2	Q3
0.1	50769.3	21301.1	54588.7	73972.4
0.2	42290.6	32180.2	42290.6	49337.4
0.5	53893.7	14211.1	66773.0	85816.7
0.8	37557.7	1422.8	15450.8	93719.4
1.0	49537.9	31369.8	49537.4	58454.6

Figure 10: Results of CMA-ES with adjusted initial step-size

For the initial experiments using CMA-ES, the only adjusted parameter is the initial step-size σ_0. The configurations of step-sizes were $\sigma_0 \in \{0.1, 0.2, 0.5, 0.8, 1.0\}$. As the table shows, the final mean score does not seem to change with the initial step-size. From this, it's assumed that the initial step-size, at least in this range, does not have a significant impact, and the best of these runs, σ_0 is chosen for first comparison.

- WRITE STUFF ABOUT OUR PRACTICAL EXPERIENCES WITH THE STEP SIZE AND LOWER BOUND
- MAKE SURE TO FULLY CONCLUDE THAT THE INITIAL STEP-SIZE DOESN'T MATTER

12 Comparison between CMA-ES and CE

12.1 Global comparison settings

In all papers used for reference we haven't seen any experiments with different population and offspring sizes presented side-by-side. However, in our upcoming experimental settings CMA and CE are most likely to have different population and offspring sizes, which means that we can't compare them based on iterations/generations. Instead, using the *number of agents* as comparison reference, equal terms are secured for both algorithms in regards to learning potential. As one of the adjustments to the algorithms are tuning the population size, it's required that the frame of reference is invariant.
Regarding the graphs themselves, the comparison graphs' x-axis shows the total number of Tetris games evaluated, $\sum_{i=1}^{t} N$. Meanwhile the y-axis still represents the mean agent's score of the iteration t.

12.2 Initial comparison - Bertsekas

For the initial comparison we use the Bertsekas featureset, since the same featureset was used for verifying the Cross Entropy implementation. Furthermore, others researchers has used the Bertsekas featureset as a benchmarking standpoint (Thiery and Scherrer, 2009) & (Szita and Lörincz, 2006).
The goal of this comparison is to get an initial idea of how the Shark implementation of CMA compares to Cross Entropy.

Results
- MORE STATISTICAL RESULTS, MULTILEVEL MODEL, ETC.

Using Cross Entropy with the constant noise setting and CMA with an initial step-size of 0.5, we get the following results, seen in figure 11.

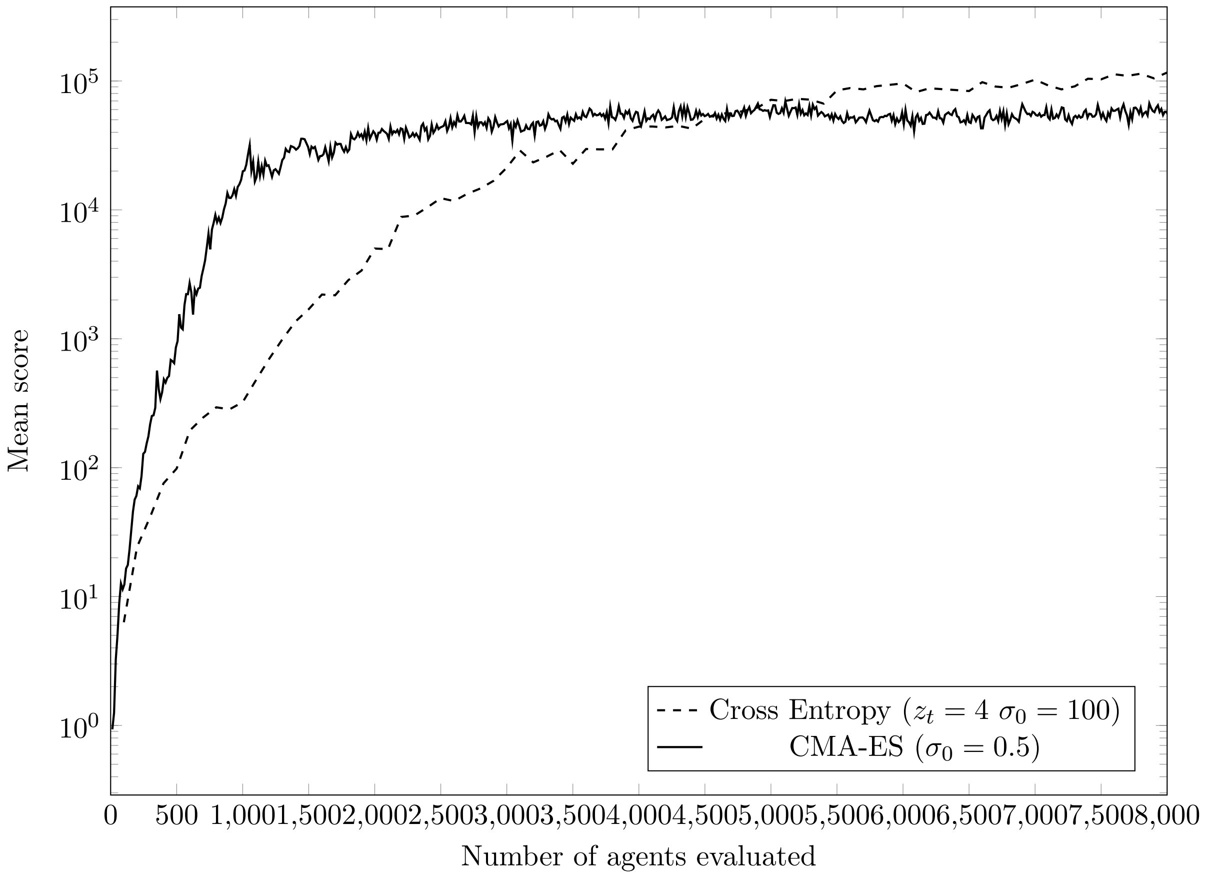

Figure 11: Initial comparison between CMA-ES and Cross Entropy

As it can be observed in figure 11 CMA converges faster, but seems to reach a local optimum around 2,000 agents evaluated. Meanwhile CE has a slower convergence but reaches a better local optimum compared to CMA, around 5,500 agents evaluated. In detail CMA on average reaches a score 50,000 rows, and CE reaches a score of 100,000.

Analysis and discussion
These results clearly defy our initial hypothesis, namely we estimated that CMA would clearly outperform CE, due to CMA's more sophisticated method of searching. One reason for this outcome could possible be that CMA has a very little population size compared to Cross Entropy, which could be a decisive lack as the evaluation function is non-deterministic with a very high variance. These results could indicate that we need to tweak the CMA configuration, as the CMA local optimum is reached quickly compared to Cross Entropy. More specifically we might need to adjust the number games played gamer per agent, since the low

population size and increasing variance makes the CMA converge on a local optimum. But as it can be seen in figure 11, the fast convergence suggests that one game per agent is sufficient until the score begins to stall at around 1,000-2,000 agents evaluated. It would seem that as the CMA-ES reaches a score of around 50,000, variance of the objective function appears to render the algorithm incapable of effectively distinguish performance of agents. Therefore, to increase the perception of the agents performs, more evaluations per agent could be necessary at higher scores.

As no experiment so far has yielded consistent scores of more than around 200,000 on average, one explanation could be that the feature set itself poses a limit for the optimization algorithms. To address this issue, further experiments will include other feature sets. Also, the specific task is not to find a good controller for Tetris, but to compare how well each algorithm performs. To reduce runtimes, the Tetris games played can be configured to be harder to play. This includes possible reduction of board size and posing higher chances of occurrence of pieces that are difficult to place.

Therefore, based on the current results, conducting more experiments to discover optimal settings for CMA-ES seems appropriate.

12.3 Configuration of CMA

- WRITE ABOUT ADJUSTMENTS MADE TO CMA BECAUSE OF THE PREVIOUS EXPERIMENT

12.4 Dellacherie comparison

INDLEDENEDE TEKST

Results

- WHAT RESULTS DID THE COMPARISON YIELD
- SHOW RESULTS FROM COMPARISON EXPERIMENTS

Analysis and discussion

- WHY DID WE GET THESE RESULTS
- DISCUSS THE RESULTS OF THE COMPARISON EXPERIMENTS

12.5 Dynamic games played per agent

INDLEDENEDE TEKST

Results

- WHAT RESULTS DID THE COMPARISON YIELD
- SHOW RESULTS FROM COMPARISON EXPERIMENTS

Analysis and discussion

- WHY DID WE GET THESE RESULTS
- DISCUSS THE RESULTS OF THE COMPARISON EXPERIMENTS

13 Conclusion

CAN WE CONCLUDE IF EITHER ALGORITHM PERFORMS BEST?

References

Bertsekas, D. and Tsitsiklis, J. (1996). Neuro-dynamic programming. *Athena Scientific.*

Boumaza, A. (2009). On the evolution of artificial tetris players. *HAL*, 14:1–14.

Fahey, C. (2003). Tetris ai. `http://www.colinfahey.com/tetris/`. Accessed: 2015-11-21.

Hansen, N. (2015). The CMA evolution strategy: A tutorial.

Igel, C., Heidrich-Meisner, V., and Glasmachers, T. (2008). Shark. *Journal of Machine Learning Research*, 9:993–996.

MDPTetris (2009). mdptetris. *None.*

Pieter-Tjerk De Boer, Dirk P. Kroese, S. M. and Rubenstein, R. Y. (2014). A tutorial on the cross-entropy method. *Springer Science*, 521:20–67.

Scherrer, C. T. B. (2009a). Building controllers for tetris. *International Computer Games Association Journal*, 32:3–11.

Scherrer, C. T. B. (2009b). Improvements on learning tetris with cross entropy. *International Computer Games Association Journal,*, 32:23–33.

Szita, I. and Lörincz, A. (2006). Learning tetris using the noisy cross-entropy method. *Neural Computation*, 18(12):2936–2941.

Thiery, C. and Scherrer, B. (2009). Improvements on learning tetris with cross entropy. *International Computer Games Association Journal*, 32(1):23–33.
[]