On the Role of Tracking in Stationary Environments

Richard S. Sutton SUTTON@CS.UALBERTA.CA
Anna Koop ANNA@CS.UALBERTA.CA
David Silver SILVER@CS.UALBERTA.CA
Department of Computing Science, University of Alberta, Edmonton, T6G 2E8, Canada

Abstract

It is often thought that learning algorithms that track the best solution, as opposed to converging to it, are important only on non-stationary problems. We present three results suggesting that this is not so. First we illustrate in a simple concrete example, the Black and White problem, that tracking can perform better than any converging algorithm on a stationary problem. Second, we show the same point on a larger, more realistic problem, an application of temporal-difference learning to computer Go. Our third result suggests that tracking in stationary problems could be important for meta-learning research (e.g., learning to learn, feature selection, transfer). We apply a meta-learning algorithm for step-size adaptation, IDBD (Sutton, 1992a), to the Black and White problem, showing that meta-learning has a dramatic long-term effect on performance whereas, on an analogous converging problem, meta-learning has only a small second-order effect. This small result suggests a way of eventually overcoming a major obstacle to meta-learning research: the lack of an independent methodology for task selection.

1. Introduction

Much of machine learning can be characterized as the search for a solution that, once found, no longer need be changed. Throughout conventional supervised and unsupervised learning the common presumption is that learning will be done in a separate phase and only after it is complete will the system be used. No learning is expected during the normal operation of the learned system. In supervised learning, of course, it is often not possible for learning to continue during normal operation because appropriate training data are no longer be available. Reinforcement learning and unsupervised learning, on the other hand, are not strongly limited in this way. They could continue to learn during their normal operation; the training information they require would still be available at this time. Nonetheless, the focus in the overwhelming majority of current work in these areas is on finding a single, stationary solution. When reinforcement learning is applied in a game-playing context, as in the world's best backgammon player TD-Gammon (Tesauro, 1995), the objective is to find a single, static, high-quality evaluation function. When reinforcement learning is applied to learning to fly a helicopter (Ng et al., 2004), the search for a good policy is done in simulation and no learning is done while the helicopter is actually flying. When reinforcement learning is applied to learning a good gait for a robot dog (Kohl & Stone, 2004), the learning occurs during an extensive self-training period and does not continue after the dog is playing soccer. In many of these cases, even if the learning could continue during normal operation, the prior experience is so extensive and the adaptation so slow that no significant learning would occur during normal operation. The standard paradigm of machine learning, with a few notable exceptions, is about learning systems that converge to an optimal or good solution. In this sense, machine learning has been more concerned with the results of learning than with the ongoing process of learning.[1]

Focusing only on the results of learning is of course not adequate when dealing with a non-stationary environment. If the world may change, then no prior learning

Appearing in *Proceedings of the 24th International Conference on Machine Learning*, Corvallis, OR, 2007.

[1] There are important exceptions to this general characterization, most notably work on meta-learning and continual learning, some of which we will discuss later in this paper. There is also extensive work on tracking algorithms in literatures more commonly associated with engineering.

could ever be sufficient on its own. When the world changes, mistakes will be made and, if learning does not continue, they will be made over and over again. The need for tracking in nonstationary environments, although widely acknowledged, has not been extensively pursued. For the most part researchers have chosen to focus on the stationary case, in part because it is clearer, and in part perhaps because it is seen as a step best completed before moving on to the nonstationary case.

In this paper we suggest that focusing on convergence to a single solution rather than continual tracking of a solution may not be good enough even for stationary problems. We assume that our data are coming from a system with evolving states and thus are not identically and independently distributed. For example, the world may be a Markov decision process. The need for tracking may arise in such cases just because the world is large. The learning agent encounters different parts of it at different times. In this case, rather than finding a single global solution, it may be advantageous for the agent to adapt to the local environment—the specific part of the state space it finds itself encountering at the particular time. By local here we mean temporally local. The right answers in nearby times may tend to be similar: in other words, the world may be *temporally coherent*. Temporal coherence occurs in nonstationary problems, of course, but it can also occur in a stationary problem.

In this paper we present two examples of stationary environments with temporal coherence, in which tracking algorithms perform better than the best converged solutions. The first is a very simple abstract problem that we call the Black and White world. This is an extreme example with strong representational limitations, so strong that its 20 states is a big space. This example helps refine our intuitions and terminology because it is absolutely concrete and in that sense clear. The second example is larger and more representative of how tracking might be important in applications. We show in a experiment with computer Go that a tracking algorithm can perform better than a converging algorithm in learning a static evaluation function in a stationary problem. This example suggests that making a distinction between tracking and convergence matters for the ultimate applications of machine learning.

Another role that tracking in stationary environments may play in machine learning is in helping make sense of the utility of meta-learning methods such as learning-to-learn, transfer between tasks, feature selection and relevance, and step-size adaptation. In studying these topics one is explicitly concerned with a sequence of learning problems. One way this can be explored is in a tracking context with a nonstationary world. Another way, and the most common way, is to use a experimenter-provided sequence of tasks. This approach has always been methodologically suspect (Ben-David & Schuller, 2003; Konidaris, 2006) because many meta-learning methods will work well on some set of tasks but not on others, and it is not clear where the tasks would come from in real applications. If it were possible to use a single stationary task to produce a sequence of tasks inherently related by being parts of an overall task, then this could provide a well-grounded methodology for studying demonstrating the merits of meta-learning.

Studies of meta-learning end up focusing on sequences of tasks because, ultimately, meta-learning is a second-order effect (Caruana, 2005). On any single task it will have a small effect, one that is easily overwhelmed by the first-order effect of the performance of the base learning system. Interest in meta-learning comes from the belief that even though it is a second-order effect on the first problem, over many problems the repeated influence of this second-order effect will become dominant. For example, on a single problem the time spent discovering new features may slightly improve performance, but the problem may be solved by the time the correct features are found. The big advantage of discovering new features become visible only on subsequent problems, on which the found features will enable rapid learning. Thus, in any single problem meta-learning's effect may be small and obscured, whereas in a tracking task these second-order effects may be dominant. To the extent that this is true, tracking may be a better paradigm for studying meta-learning.

In this paper we show first a very simple example called the Black and White world. We consider two versions of this world, one is in which there is temporal coherence and tracking is better than converging, and one in which there is no temporal coherence and there is no advantage to tracking. We show that in the former case a meta-learning technique can provide dramatic performance improvements, whereas in the case without temporal coherence there is little advantage to the meta-learning technique.

2. Tracking in the Black and White world

To illustrate the idea that tracking can be better than converging, we created a simple 20-state world (Figure 1). The agent follows a random walk, occasionally looking up and seeing either black or white. The goal

Figure 1. The Black and White world. The agent follows a random walk right and left, occasionally observing the color above it. The states wrap.

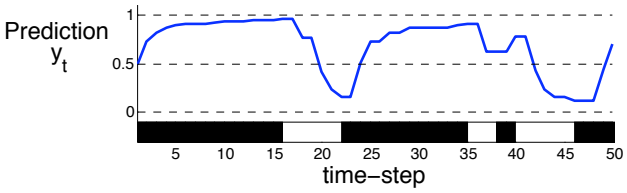

Figure 2. A sample trajectory in the Black and White world, showing the prediction on each time-step and the actual color above the agent. The prediction is modified only on time steps on which the color is observed. Here $\alpha = 2$.

is to predict the probability of observing black using a single scalar parameter. The environment is stationary, but cannot be represented accurately with only one parameter.

The prediction $y_t \in (0, 1)$ is computed from a logistic sigmoid over the learned parameter w:

$$y_t = \frac{1}{1 + e^{-w_t x_t}}, \tag{1}$$

where $w_t \in \Re^n$ denotes the learned parameter at time step t, and $x_t \in \Re^n$ denotes a feature vector at time step t, where here in the Black and White world we have the simplest case in which $n = 1$ and $x_t = 1$ for all t. We describe it here in the general, multi-dimensional form because we will use that form in the computer Go application presented in the next section. We will refer to the parameter w_t as the *weight vector* or, in the scalar case, simply as the *weight*.

The target value for the prediction is the actual observation when the agent looks up, which we denote as z_t, where $z_t = 0$ if the agent looks up and sees white, $z_t = 1$ if the agent looks up and sees black, and z_t is undefined if the agent does not look up on time step t. On time steps on which the agent looks up it incurs a loss, the cross entropy between the target z_t and the current prediction y_t:

$$L_t = -z_t \log(y_t) - (1 - z_t) \log(1 - y_t). \tag{2}$$

On these time steps the weight is updated by gradient descent:

$$w_{t+1} = w_t + \alpha \delta_t x_t, \tag{3}$$

where δ_t is the difference between the target and the prediction: $\delta_t = z_t - y_t$. The learning rate is determined by the step-size parameter $\alpha > 0$.

In the Black and White world, the single best parameter value for minimizing loss is $w_t = 0$, because in the long run the frequency of seeing black is 0.5. With a sufficiently small step-size parameter, w_t approaches 0. With a larger step-size parameter, the weight update is more influenced by the current error than by the long term average, and w_t will vary by a larger amount. In the Black and White world, observations made soon after each other are more likely to be the same color than observations separated by longer time frames. This temporal coherence suggests that tracking may be beneficial. Figure 2 illustrates this, charting which part of the world (black or white) the agent is in together with the prediction at that time for a typical sequence of 50 time steps.

When the agent remains in a consistent region, the prediction approaches the correct value. After the first observation in a new region, the prediction is adjusted accordingly. With a small α, as in this example, it may take several observations before the prediction catches up with the target.

To empirically illustrate the benefits of tracking in a stationary environment, we tested several settings of α in the Black and White world. For each setting, we ran 30 episodes each with 200,000 observation steps. Results are reported for the second 100,000 steps only, to remove any effect of the initial conditions. The `look`, `left`, and `right` actions were chosen randomly with probabilities 0.5, 0.25, and 0.25 respectively. The boundaries of the world wrap: taking the left action in the leftmost state moves the agent to the rightmost state, and similarly for the right action in the rightmost state.

The mean loss and standard errors are displayed in Figure 3. The dotted line is the loss of the best converged solution. The solid line shows the tracking results. For small values of α the solution was arbitrarily close to the converged solution, with corresponding loss. For very high values of α, the loss of the tracking solution was worse than that of the converged solution. For intermediate values, the loss of the tracking solution was significantly better than the converged solution. In this world, an α value of 4 resulted in the lowest loss among the values tested. Across all $\alpha = 4$ runs, the average loss was 0.24. The loss of the best converged solution was almost three times this, at 0.69.

The best choice of α depends on the degree of temporal

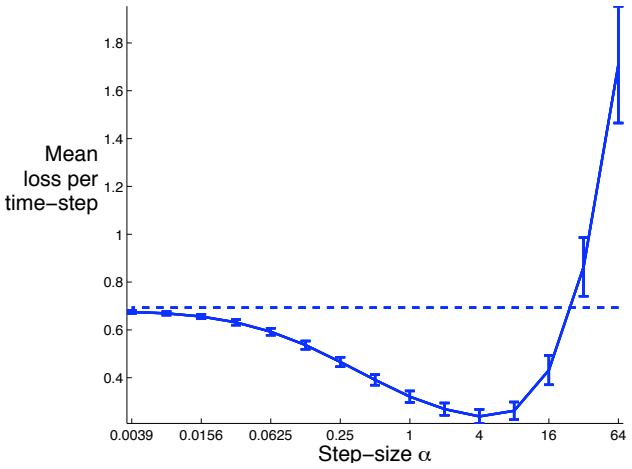

Figure 3. Comparison of the mean log loss per time-step for fixed step-sizes in the Black and White world. The dotted line marks the loss of the converged solution. Standard error bars are given.

coherence of the environment. If the probability of looking up is increased, the lowest loss occurs with larger values of α. When the probability of looking is very small, temporal coherence is completely lost and the best values for α are those that allow approximate convergence. In a later section we will see how α can be set by a meta-learning algorithm.

3. Tracking versus converging in Go

To compare tracking and converging algorithms in a more complex domain, we used the game of 5×5 Go. Even with a small board size, this domain poses a considerable challenge. There are more than 5×10^{10} unique states, and the game contains sufficient strategic depth to merit a regular column in professional Go periodicals (Davies, 1994).

In a complex domain such as Go, it is usual to seek the best approximation to the optimal policy that can be achieved by a particular representation, for example a linear combination of binary features (Silver, Sutton & Müller, 2007), or a multi-layer perceptron (Schraudolph, Dayan & Sejnowski, 1994; Enzenberger, 2003). However, it may be possible to do better than any fixed policy, given the same representation. At each time step, the agent seeks the best policy for the distribution of states encountered when starting from the current state. Thus, the agent devotes its learning resources to the current situation, rather than spreading them across the complete distribution of states.

To demonstrate this idea, we chose the representation used by Silver et al. (2007). The value function $V(s)$ is approximated by a linear combination of binary features $x(s)$, squashed by a sigmoid function (see Equation 1 and Figure 4). The reward function is $r = 1$ for winning, and $r = 0$ otherwise, so that the value function estimates the probability of winning the game.

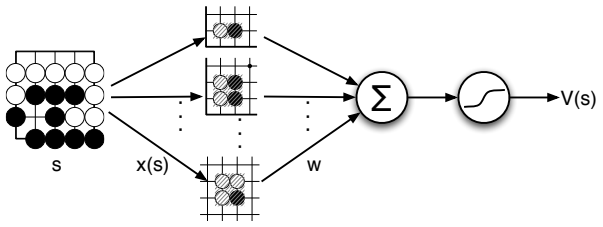

Figure 4. Value function approximation for 5×5 Go

Each binary feature recognizes a particular pattern of stones within some rectangle on the board. Binary features are used for all possible configurations from 1×1 up to 3×3; some example features are shown in the left sides of Figures 6 and 7. Weights are shared between sets of symmetric shapes, to take account of any rotational, reflectional and translational symmetries that may exist (Silver el al., 2007). The weights for these features can be interpreted as the expected contribution that each shape makes to winning the game, over the on-policy distribution of states.

As in the Black and White world, we adjust weights so as to minimize the cross entropy between the current prediction and the subsequent prediction. Thus, we use equations 2 and 3, where the target at time t is set according to the $TD(0)$ algorithm (Sutton, 1988):

$$z_t = r_{t+1} + V(s_{t+1}). \tag{4}$$

We considered two versions of the learning algorithm. For the converging agent, we initialized all weights to small random values and trained offline for 250,000 complete episodes of self-play. For the tracking agent, we also initialized the weights randomly. At every time-step t, we trained the agent online for 10,000 episodes of self-play, starting from the current position s_t.[2] The result of 5×5 Go is usually determined within the first 25 moves, thus the tracking

[2]This tracking approach to computer Go is surprisingly practical. Because we use a linear evaluation function and binary features, learning is very fast. In this setting the learning algorithm is fast enough to simulate and process 10,000 complete games in just a few seconds (see table 2). In fact, a fully functional 9x9 Computer Go program currently competes online on the Computer Go Online Server, using precisely this tracking algorithm. Not only does this demonstrate that the tracking algorithm is practical, but also that it can be used under strict time constraints (5 minutes per complete game on CGOS).

Features	Tracking beats converging		
	Black	White	Total
1×1	82%	43%	62.5%
2×2	90%	71%	80.5%
3×3	93%	80%	86.5%

Table 1. Percentage of 5×5 Go games won by the tracking agent playing against the converging agent when playing as Black (first to move) and as White.

agent received slightly less experience than the converging agent. We played the tracking and converging agents against each other to compare their performance. Both agents used an ϵ-greedy policy during self-play training, but a greedy policy to select their actual moves. The step-size was set to $\alpha_t = 0.1/||x(s_t)||$ for both agents.

The first experiment used only the 1×1 features. Each subsequent experiment included additional features of increasing complexity, up to 3×3. Every experiment consisted of 200 games, retraining both agents from scratch for each game, and alternating colours between games. In all experiments, the tracking agent won a substantial majority of the games (Table 1 and Figure 5) with the advantage being largest for the more expressive representations.

The simplest representation, using just the 1×1 features, demonstrates a clear advantage for tracking over converging. For example, it is usually bad for Black to play on the corner intersection, and so the converging agent learns a negative weight for this feature. However, Figure 6 shows a position in which the corner intersection is the most important point on the board for Black: it makes two eyes and allows the Black stones to live. By learning about the particular distribution of states arising from this position, the tracking agent learns a large positive weight for the corner feature. When playing Black in this position, the converging agent plays in the central intersection and loses; whereas the tracking agent plays in the corner and wins.

As the representation becomes more expressive, the agent is able to learn more complex patterns and the performance of both tracking and converging increases. However, the tracking agent is able to exploit the additional features better than the converging agent (see Figure 5). For example, the converging agent now learns that the corner intersection is bad in general, but good when it occurs in a 3×3 pattern providing two eyes. However, there are still special cases where this does not hold. Figure 7 shows a similar position in which this same corner pattern is

Features	Total features	CPU (minutes)	
		Tracking	Converging
1×1	75	3.5	10.1
2×2	1371	5.7	13.8
3×3	178518	9.1	22.2

Table 2. Memory and CPU requirements for tracking and converging agents. The total number of binary features indicates the memory consumption. The CPU time is the average training time required to play a complete game: 250,000 episodes of training for the converging agent; 10,000 episodes of training per move for the tracking agent.

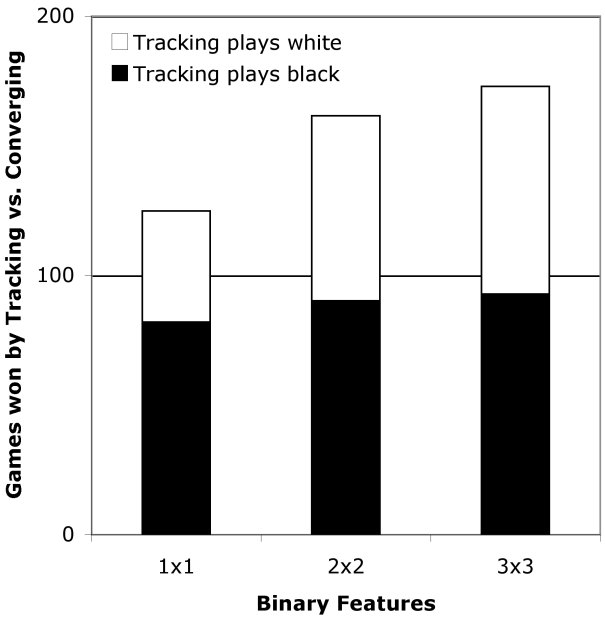

Figure 5. Games won by tracking agent against converging agent, playing 100 games as Black and 100 games as White.

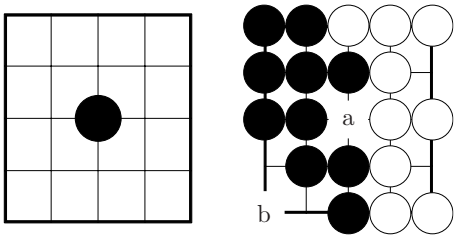

Figure 6. (Left) A 1×1 feature with a central black stone. (Right) With Black to play, move b is the winning move. Using 1×1 features, the converging agent plays centrally at a, having learned that this is a good feature in general. However, the tracking agent learns that Black must play at b in this particular situation, to make two eyes.

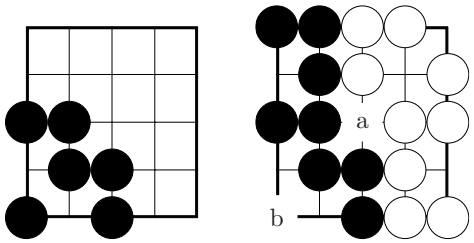

Figure 7. (Left) A 3×3 feature making two eyes in the corner. (Right) Black to play, move a is now the winning move. Using 3×3 features, the converging agent makes two eyes at b, believing this to be a good shape in general. However, the tracking agent realizes that move b is redundant (black already has two eyes) and learns to play the winning move at a.

now bad: Black already has two eyes and should play in the center to maximize his territory. The converging agent is unable to understand the global context and plays the wrong move in the corner. The tracking agent learns that the corner pattern is not as important as the central territory in this context, and plays the correct move in the center. Thus, the tracking agent customizes its policy to the current situation and outperforms the converging agent, even when the representation is expressive and rich with features.

4. Step-size adaptation in the Black and White world

As we saw in the Black and White world, the best step-size parameter α generally depends on the degree of temporal coherence of the world, which may not be known a priori. This is an area in which meta-learning might play a role. We present an adaptation of the incremental delta-bar-delta (IDBD) algorithm, an online meta-learning algorithm that uses gradient descent to learn step-size parameters (Sutton, 1992a, 1992b). Here we use a version of IDBD customized for the log loss we use in this paper. Our derivation of the IDBD algorithm for log loss directly parallels that presented by Sutton (1992a) for squared error.

The IDBD algorithm allows for a different step-size α^i for each component w^i of the parameter vector w. The weight update rule is similar to that for the scalar case shown in Section 2:

$$w^i_{t+1} = w^i_t + \alpha^i_{t+1}\delta_t x^i_t. \tag{5}$$

The step-size α^i_t is a function of a new parameter β^i_t:

$$\alpha^i_t = e^{\beta^i_t}. \tag{6}$$

The parameter β^i is updated according to the gradient descent rule with meta-learning rate μ. The derivative is with respect to β^i, which can be thought of as the derivative of the loss with respect to an infinitesimal change in β^i at all time steps. Let $h^i_t = \frac{\partial w^i_t}{\partial \beta^i}$. Then:

$$\begin{aligned}
\beta^i_{t+1} &= \beta^i_t - \mu \frac{\partial L_t}{\partial \beta^i} \\
&= \beta^i_t - \mu \frac{\partial}{\partial \beta^i}[-z_t \log(y_t) - (1 - z_t)\log(1 - y_t)] \\
&= \beta^i_t + \mu z_t (1 - y_t) \sum_{j=1}^{n} \frac{\partial w^j_t x^j_t}{\partial \beta^i} \\
&\quad - \mu(1 - z_t) y_t \sum_{j=1}^{n} \frac{\partial w^j_t x^j_t}{\partial \beta^i} \\
&\approx \beta^i_t + \mu z_t (1 - y_t) x^i_t \frac{\partial w^i_t}{\partial \beta^i} + \mu(z_t - 1) y_t x^i_t \frac{\partial w^i_t}{\partial \beta^i} \\
&= \beta^i_t + \mu \delta_t x^i_t h^i_t.
\end{aligned}$$

Note the derivative is exact in the scalar case.

We calculate the derivative of w^i_t with an accumulating trace:

$$\begin{aligned}
h^i_{t+1} &= \frac{\partial w^i_{t+1}}{\partial \beta^i} \\
&= \frac{\partial w^i_t}{\partial \beta^i} + \frac{\partial \alpha^i_{t+1} \delta_t}{\partial \beta^i} x^i_t \\
&= h^i_t + \frac{\partial e^{\beta^i_{t+1}}}{\partial \beta^i} \delta_t x^i_t + e^{\beta^i_{t+1}} x^i_t \frac{\partial (z_t - y_t)}{\partial \beta^i} \\
&= h^i_t + e^{\beta^i_{t+1}} \delta_t x^i_t - e^{\beta^i_{t+1}} x^i_t y_t (1 - y_t) \sum_{j=1}^{n} \frac{\partial w^j_t x^j_t}{\partial \beta^i} \\
&\approx h^i_t + e^{\beta^i_{t+1}} \delta_t x^i_t - e^{\beta^i_{t+1}} (x^i_t)^2 y_t (1 - y_t) \frac{\partial w^i_t}{\partial \beta^i} \\
&= h^i_t [1 - \alpha^i_{t+1} (x^i_t)^2 y_t (1 - y_t)] + \alpha^i_{t+1} \delta_t x^i_t
\end{aligned}$$

The full algorithm for semi-linear IDBD is given in Figure 1.

Algorithm 1 Semi-linear IDBD

Initialize h^i_0 to 0, w^i_0 and β^i_0 as desired.
for each time step t **do**
  $y \leftarrow \frac{1}{1 + e^{\sum_{i=1}^{n} - w^i x^i}}$
  $\delta \leftarrow z - y$
  for each weight i **do**
    $\beta^i \leftarrow \beta^i + \mu \delta x^i h^i$
    $\alpha^i \leftarrow e^{\beta^i}$
    $w^i \leftarrow w^i + \alpha^i \delta x^i$
    $h^i \leftarrow h^i [1 - \alpha^i (x^i)^2 y (1 - y)] + \alpha^i \delta x^i$
  end for
end for

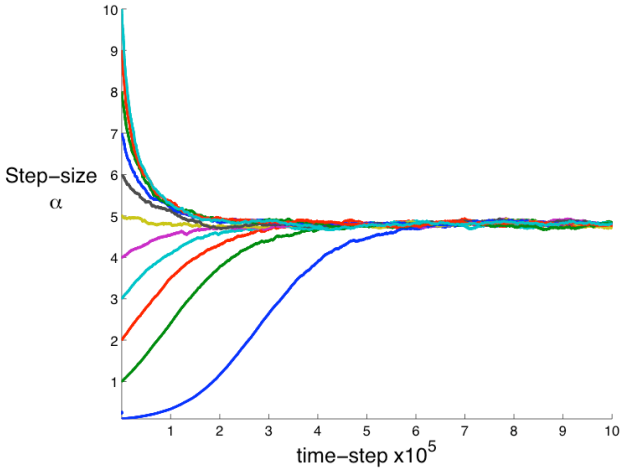

Figure 8. The step-size parameters found by several runs of IDBD in the Black and White world with different initial step-size values.

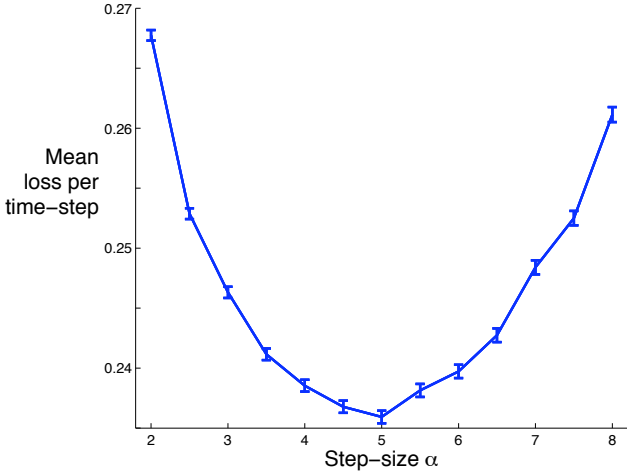

Figure 9. A more detailed look at performance in the Black and White world as a function of step size (cf. Figure 3).

In our earlier experiments with fixed values of α in the Black and White world we found that the best step size was approximately 4 (see Figure 3). Can IDBD find this value? Several sample runs of one million steps with $\mu = 2^{-13}$ are illustrated in Figure 8, with different initial values for β_0. In all cases, the α values in the last ten thousand steps were between 4.69 and 4.88. These values are certainly closer to 4 than to any of the other α values tried earlier, but is this range near the optimum? To determine this, we repeated the fixed-α experiment of Figure 3 at a finer grain. As before, we measured the mean loss for the last 100,000 steps of thirty 200,000-step runs. We can see from Figure 9 that the best α is between 4.5 and 5. The α found by IDBD is consistently within this range.

5. Tracking as a sensitive assay for meta-learning

In the Black and White world the advantages of step-size meta-learning were apparent—finding the best α parameter reduced the loss by a factor of three. This benefit arose because the problem is temporally coherent and is thus best approached as a tracking problem. If the problem was best approached as a converging problem it would be much harder to show a benefit for meta-learning.

To illustrate this, we created a temporally-incoherent version of the Black and White world. In it there is only one state and the task is to predict an observation that is randomly black or white with equal probability. The best prediction is always 0.5, achieving the minimal per-time-step loss of 1.0.

As before, only one weight is learned. We applied our learning algorithm with and without IDBD to this world. Without IDBD, a fixed step size was used for the duration of a run. With IDBD, the step size was initialized identically and then changed by the IDBD rule with a meta step-size of $\mu = 2^{-10}$. Each run consisted of 1,000 steps, and results were averaged over thirty runs. The weight w_0 was initialized to -5.

Results are shown in Figure 10. For intermediate values of α, the mean per time-step loss approached that of the optimal, converged solution (1.0). For smaller values of α, 1,000 steps was not long enough to learn a good weight, and the loss was higher. For large values of α, the prediction was always chasing the last observation, resulting in high loss. Only in the high loss case is there any significant advantage to using IDBD. Although the choice of step-size is important in this problem, there is not time to find it in the single, small, stationary task. Performance is dominated by the choice of the initial step size, swamping the effect of meta-learning.

6. Conclusion

We have shown two examples in which tracking algorithms perform better than converging ones even though the underlying problem is stationary. The Black and White world is an extremely small illustration where the issues can be fully examined and understood. Our computer Go example shows that the advantages of tracking methods over converging methods can arise in larger and more realistic problems. One of our conclusions is that tracking algorithms deserve more attention than they have previously been

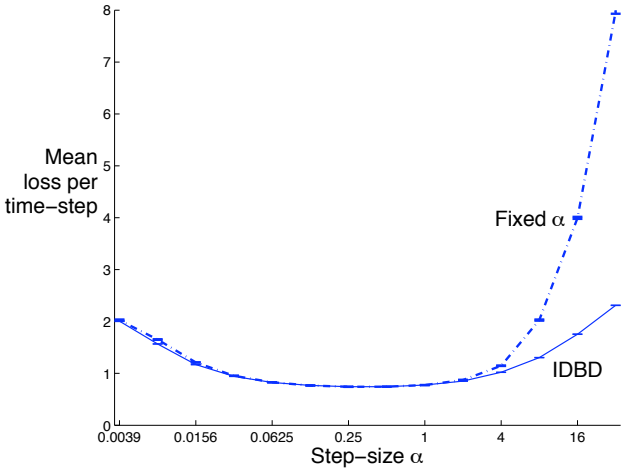

Figure 10. Performance on the temporally-incoherent version of the Black and White world as a function of a fixed step-size or the initial step-size of the IDBD meta-learning rule. On a single problem where tracking is not important the advantages of meta-learning may be negligible.

afforded. Machine learning's near exclusive focus on convergence may be causing opportunities to be overlooked.

Tracking becomes important in stationary domains which have temporal coherence and are too large to be solved exactly. We have emphasized the implications of tracking and temporal coherence for the study of meta-learning, using as an example the IDBD step-size learning algorithm. In the Black and White world, we showed that IDBD improved performance substantially on a temporally coherent version of the problem while having negligible effect on a version without temporal coherence. These observations may provide a route to resolving a nagging methodological problem for meta-learning research: how to justify the choice of a sequence of tasks. If the sequence can arise from different parts of a single task, then this choice can become non-arbitrary.

References

Ben-David, S., Schuller, R. (2003). Exploiting Task Relatedness for Multiple Task Learning. *Proceedings of the Conference on Learning Theory*, (pp. 567–580).

Caruana, R. (2005). Inductive Transfer Retrospective and Review. *NIPS 2005 Workshop on Inductive Transfer: 10 Years Later.*

Davies, J. (1994). 5x5 Go. *American Go Journal 28*(2) 9–12.

Enzenberger, M. (2003). Evaluation in Go by a neural network using soft segmentation. *10th Advances in Computer Games Conference* (pp. 97–108).

Kohl, N., Stone, P. (2004). Policy Gradient Reinforcement Learning for Fast Quadrupedal Locomotion. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA 2004)* (pp. 2619–2624).

Konidaris, G. D. (2006). A Framework for Transfer in Reinforcement Learning. *ICML-06 Workshop on Structural Knowledge Transfer for Machine Learning.*

Ng, A. Y., Coates, A., Diel, M., Ganapathi, V., Schulte, J., Tse, B., Berger, E., Liang, E. (2004). Inverted autonomous helicopter flight via reinforcement learning. *International Symposium on Experimental Robotics.*

Schraudolph, N. N. (1999). Local Gain Adaptation in Stochastic Gradient Descent. *Proceedings of the 9th International Conference on Artificial Neural Networks* (pp. 569–574).

Schraudolph, N., Dayan, P., & Sejnowski, T. (1994). Temporal difference learning of position evaluation in the game of Go. *Advances in Neural Information Processing 6*, (pp. 817-824).

Silver, D., Sutton, R., & Müller, M. (2007). Reinforcement learning of local shape in the game of Go. *20th International Conference on Artificial Intelligence* (pp. 1053–1058).

Sutton, R. (1988). Learning to predict by the method of temporal differences. *Machine Learning 3*(1) 9–44.

Sutton, R. S. (1992a). Adapting bias by gradient descent: An incremental version of delta-bar-delta. *Proceedings of the Tenth National Conference on Artificial Intelligence*, (pp. 171–176).

Sutton, R. S. (1992b). Gain adaptation beats least squares? *Proceedings of the Seventh Yale Workshop on Adaptive and Learning Systems*, (pp. 161–166). Yale University, New Haven, CT.

Tesauro, G. (1995). Temporal difference learning and TD-Gammon. *Communications of the ACM 38*(3).