Empirical Comparison of Gradient Descent and Exponentiated Gradient Descent in Supervised and Reinforcement Learning
Technical Report 96-70

Doina Precup, Rich Sutton

September 19, 1996

Abstract

This report describes a series of results using the exponentiated gradient descent (EG) method recently proposed by Kivinen and Warmuth. Prior work is extended by comparing speed of learning on a nonstationary problem and on an extension to backpropagation networks. Most significantly, we present an extension of the EG method to temporal-difference and reinforcement learning. This extension is compared to conventional reinforcement learning methods on two test problems using CMAC function approximators and replace traces. On the larger of the two problems, the average loss was approximately 25% smaller for the EG method. The relative computational complexity and parameter sensitivity of the two methods is also discussed.

1. Introduction

This report presents the results of several experiments designed to compare two different methods for learning the weights of neural networks: gradient descent (GD) and exponentiated gradient descent (EG), in the framework of supervised and reinforcement learning tasks.

Exponentiated gradient descent was proposed as an alternative on-line learning rule for linear predictors by Kivinen and Warmuth (1994). EG replaces the usual additive updates of the weights by multiplicative updates.

Kivinen and Warmuth (1994) have established theoretical worst case bounds for the cumulative error of both weight update methods, in the context of function approximation problems with linear predictors. The theoretical and experimental evidence presented suggest that EG has a lower bound on the cumulative error for problems in which only few of the input features are relevant. Thus, EG converges faster for this class of problems.

EG has been applied successfully to train linear classifiers for information routing tasks, in which every word is considered a feature (Lewis, Schapire, Callan and Papka, 1996). The experiments confirm the hypothesis that exponentiated gradient is faster than the classical Widrow-Hoff update rule in identifying irrelevant features, for sparse target problems.

Exponentiated gradient descent has several features that make it appealing for reinforcement learning. It enables on-line updates, required in RL tasks. The results from supervised learning experiments suggest that EG works best for training linear predictors with a big number of input features, few of them being relevant. Many reinforcement learning tasks in continuous state spaces have been tackled using linear function approximators with many input features (e.g. CMACs). Moreover, in reinforcement learning, getting good behaviour quickly is paramount for the performance of the system. EG updates can potentially bring a learning speed gain to RL methods.

The research presented in this report has the following goals:

- confirm and extend experiments with exponentiated gradient methods for linear predictors, in particular for non-stationary problems
- experiment EG updates in feed-forward multi-layered nets
- adapt exponentiated gradient descent for reinforcement learning
- test EG in RL tasks, in order to asess its speed and parameter tuning difficulty, with respect to additive updates

2. Exponentiated gradient method

The exponentiated gradient descent (EG) update rule is similar to gradient descent (GD), in the sense that it applies at each step a correction $\delta_i, i = 0..N$ to the existing weight vector $w_i, i = 0..N$. δ_i is computed based on the error between the predicted and the desired value for the current training example, estimated through a specified error function.

The form of the updates performed by the two methods is different. GD makes additive steps in weight space:

$$w_i := w_i + \delta_i$$

EG makes additive steps in the space of the logarithm of the weights:

$$\log(w_i) := \log(w_i) + \delta_i,$$

which leads to multiplicative updates in weight space:

$$w_i := w_i \exp(\delta_i)$$

The GD and EG update rules are obtained through similar derivations. The starting idea is that the goal of any update rule is to correct the weights using the error on the training examples, while avoiding big updates, which can lead to instability. Thus, the update rules for linear units can be viewed as trying to minimize a function of the form:

$$d(\bar{w}_{t+1}, \bar{w}_t) + \alpha E(y, \bar{w}_t \bar{x}^T)$$

where $\bar{x}$ is the input vector, y is the desired output value for the current input, $\bar{w}_t$ and $\bar{w}_{t+1}$ are the weight vectors at two succesive time steps, d is a distance metric and E is an error function. The value of the learning rate α determines if the emphasis is on the corrective or on the conservative part of the equation.

If d is the Euclidian distance ($d(u,v) = \sum_{i=1}^{N}(u_i - v_i)^2$), then minimizing the previous formula yields the GD update rule:

$$w_{t+1,i} := w_{t,i} + \alpha \frac{\partial E(y, \bar{w}_t \bar{x}^T)}{\partial w_{t,i}}$$

If d is an unnormalized relative entropy measure ($d(u,v) = \sum_{i=1}^{N}(v_i - u_i + u_i \log(\frac{u_i}{v_i}))$), assuming that the weights are restricted to positive values, the same derivation yields an EG type of update:

$$w_{t+1,i} := w_{t,i} \exp(\alpha \frac{\partial E(y, \bar{w}_t \bar{x}^T)}{\partial w_{t,i}})$$

Let the target function have the form:

$$k_0 + \sum_{i=1}^{N} k_i x_i$$

where $k_i, i = 0..N$ are fixed real coefficients, and the $x_i, i = 1..N$ are real valued input features. Let the error function E be the sum squared error on the training examples.

The gradient descent (GD) or Widrow-Hoff update rule is:

$$w_i := w_i + \alpha(y - o)x_i, \forall i = 0..N,$$

where α is the learning rate,

$$o = w_0 + \sum_{i=1}^{N} w_i x_i$$

is the real output of the predictor, and

$$y = k_0 + \sum_{i=1}^{N} k_i x_i$$

is the desired output for the training instance $(1, x_1, ..x_N)$.

Exponentiated gradient descent takes steps of the same magnitude, but in the log weight space:

$$\log w_i := \log w_i + \alpha(y - o)x_i, \forall i = 0..N$$

One implementation problem that arises for EG updates is the need to allow both positive and negative weights. The solution adopted in Kivinen and Warmuth (1994) is to duplicate the input vector, replacing it by $(1, x_1, ...x_N, -1, -x_1, ..., -x_N)$. Thus, although each weight is strictly positive, the difference of the weights corresponding to each feature can have any sign.

EG and GD updates both have linear complexity in the number of weights ($O(N)$). However, EG needs twice as many weights, and is more expensive in floating point operations (instead of an addition, for each weight we perform an exponentiation and a multiplication).

EG also has an additional parameter that needs to be tuned: the initial magnitude of the weights. If this magnitude is small, the algorithm will converge very slowly. Big initial magnitudes can lead to instability and overflow errors.

In order to keep the weight magnitudes bounded, Kivinen and Warmuth proposed a variation of the EG algorithm, which applies a supplementary normalization step, linear in the weights:

$$\sum_{i=0}^{2*N} w_i = W$$

Normalization introduces a supplementary parameter that needs to be tuned, i.e. the upper bound on the weights W.

3. Experiments with linear predictors

The first experiment evaluated EG and GD updates on a sparse target. The target function has 100 real-valued input features that can take values between -1 and 1. The target vector is $(1, -1, 1, 0, ...0)$. Thus, 97 of the 100 input features are irrelevant in this problem.

The experiment tested four learners: GD (the usual Widrow-Hoff update rule), EG (which performs additive updates in log weight space), EG-T (which also includes normalization with a tight weight bound) and EG-L (which performs normalization with a loose weight bound).

All the learners were used to train a linear unit with $N = 100$ input features. The training instances were drawn by sampling uniformly from $[-1..1]^N$.

The learning rates for the algorithms were set according to the recommended values from (Kivinen and Warmuth, 1994):

$$\alpha_{GD} = \frac{1}{4X^2},$$

where X is an estimated upper bound for $\max_x \parallel \bar{x} \parallel_2$.

$$\alpha_{EG} = \frac{1}{3XY},$$

where X is an upper bound on the feature values $(\mid x_i \mid \leq X, \forall i = 0..N)$ and Y is an upper bound on the output value $(\mid k_0 + \sum_{i=1}^{N} k_i x_i \mid \leq Y, \forall (1, x_1, ...x_N))$

$$\alpha_{EGnorm} = \frac{1}{3W^2X^2}$$

where X is an upper bound for $\max_{i=0}^{N} \mid x_i \mid$

The starting weights were set to 0 for GD. For the exponentiated gradient algorithms, the initial weight magnitude was 10, which actually started the algorithms at 0 weights for each feature (the same starting point as for GD).

The algorithms were trained on the same pool of instances, and then tested on the same independent test set. Two different measures were used to asess performance:

- the cumulative squared error on the training instances
- the root mean squared error on the test set

Figure 1 shows the RMSE on the test set and cumulative error on the training set averaged over 30 runs.

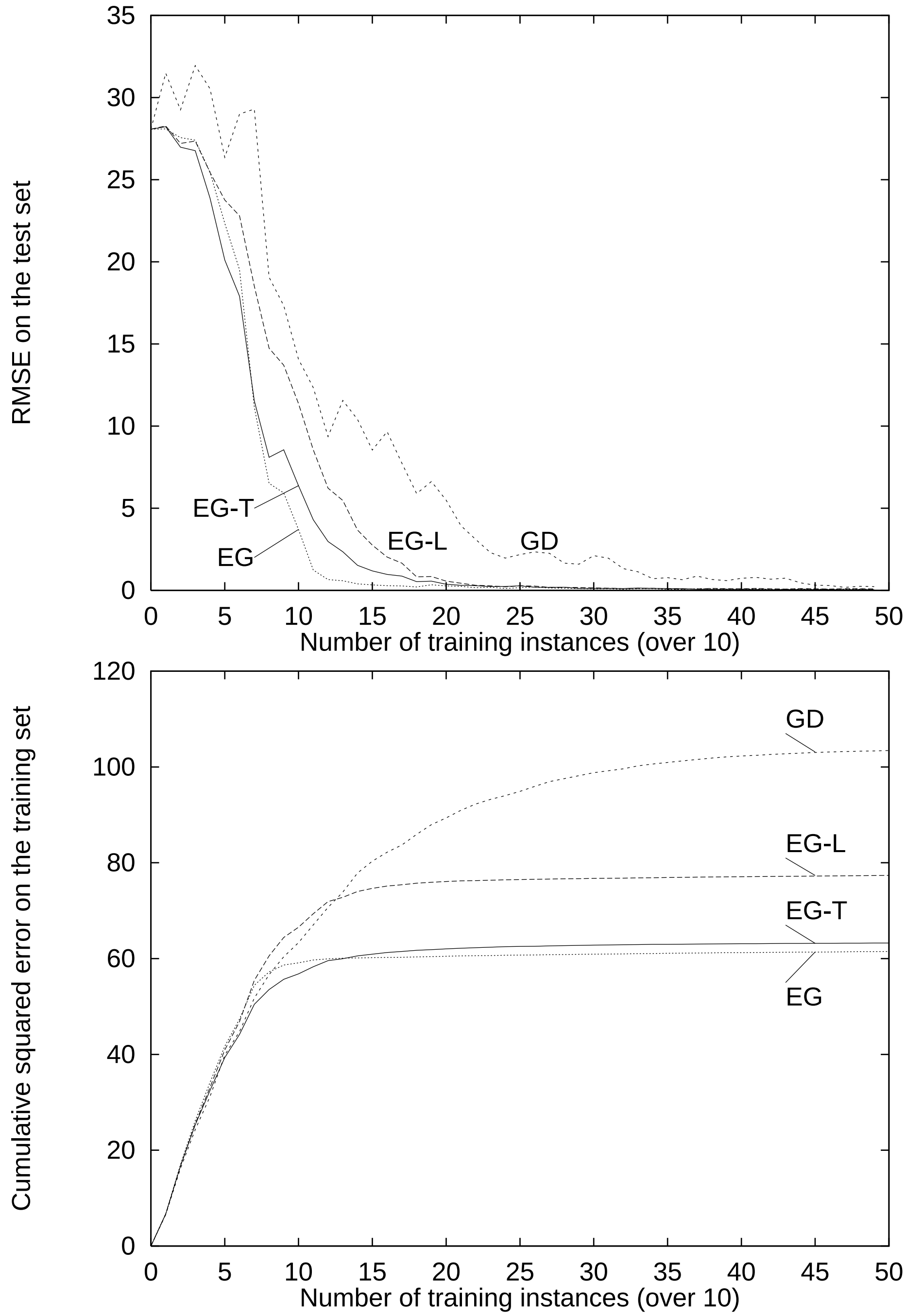

Figure 1. Supervised learning, linear unit, sparse target

All EG methods converge to the target twice as fast as GD. Their cumulative loss on the training set is also smaller. The unnormalized version (EG) and the tight bound algorithm (EG-T) have very similar performance, and they both perform better than the loose bound algorithm (EG-L).

The second task consisted of a dense target, in which all the features are relevant and they have equal weights: $(0.01, ...0.01)$. This is a worst case target for EG algorithms, which have bigger theoretical loss bounds than gradient descent for this type of problems.

The experimental methodology is similar to the previous experiment.

In this case, GD is the fastest algorithm to converge, as expected, according to the results presented in (Kivinen and Warmuth, 1994). It is worth noting that the unnormalized version of EG is faster and has a smaller cumulated loss than any of the normalized versions.

The third experiment aimed to evaluate the algorithms in terms of robustness to noise in the target value. The target was the same as for the first experiment, but for each training instance, random noise was added to the target value. The amount of noise was at most 10% of the maximum possible value for the target function.

The experimental setting and performance measures were the same as for the previous experiments.

The results of the experiment are presented in figure 3. All EG methods converge faster than GD. The unnormalized version appears to be the most robust. The tight bound algorithm has a very similar behaviour with the unnormalized version, while the loose bound algorithm has a poorer performance.

The forth experiment was set up to test the algorithms in the case of a non-stationary problem. The initial target was the same as for the first experiment. After 500 instances, the target was modified by switching the four non-zero weights with four null weights chosen randomly, and adding other two non-zero weights.

The performance measures for this experiment are presented in figure 4. The EG algoirithm converges twice as fast to the new target, compared to the performance of GD. This experiment also indicates that normalization can be a problem in non-stationary environements. The normalized algorithm that has a tight weight bound doesn't converge after the target is modified, because the new target is outside its weight space.

These experiments support the hypothesis that EG methods outperform GD for linear targets with many irrelevant features. EG also seems to be more robust to noise and faster to adapt to new conditions in non-stationary environments. This latter result is important particularly from the perspective of using EG updates in reinforcement learning tasks. Normalization hasn't been

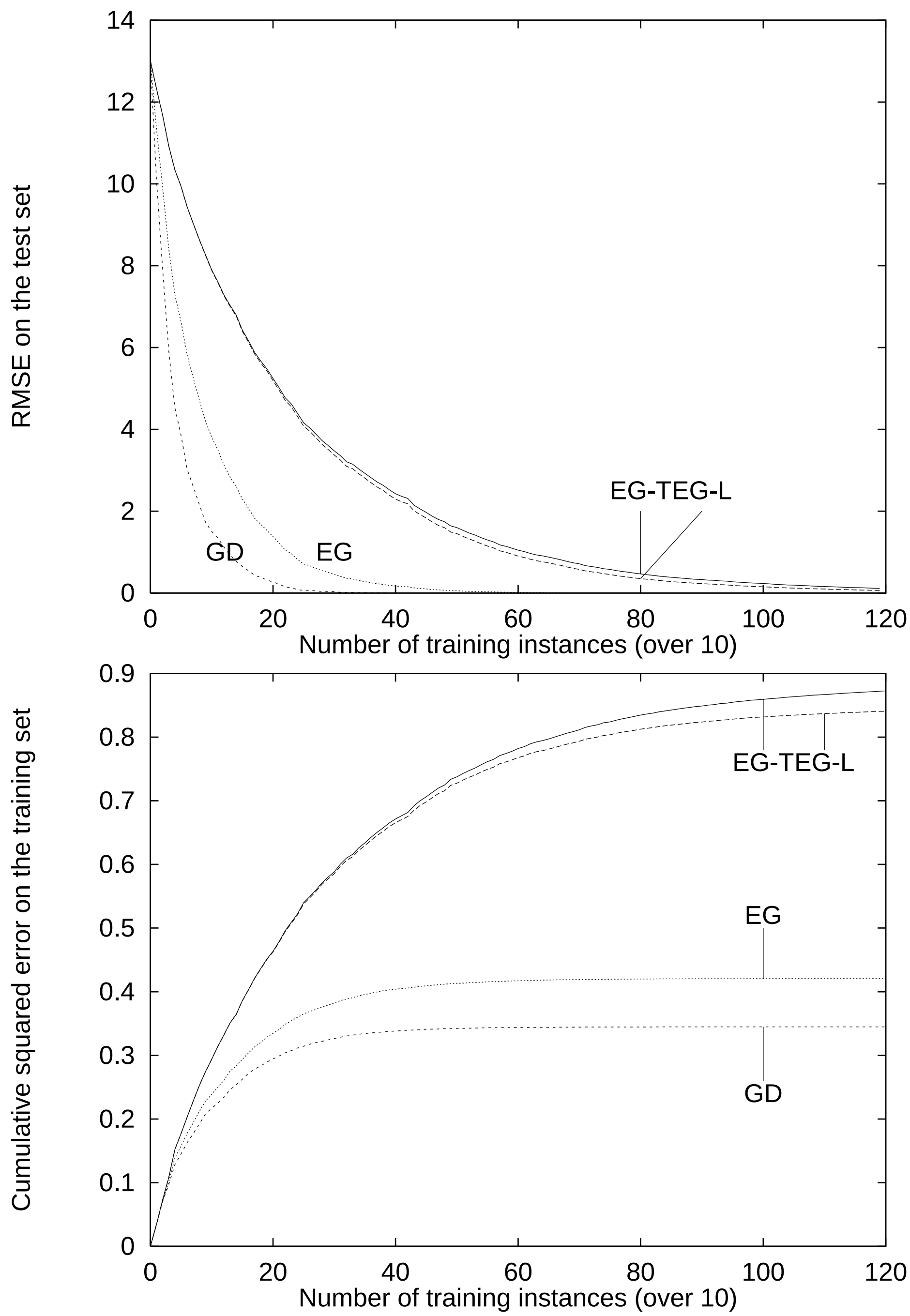

Figure 2. Supervised learning, linear unit, dense target

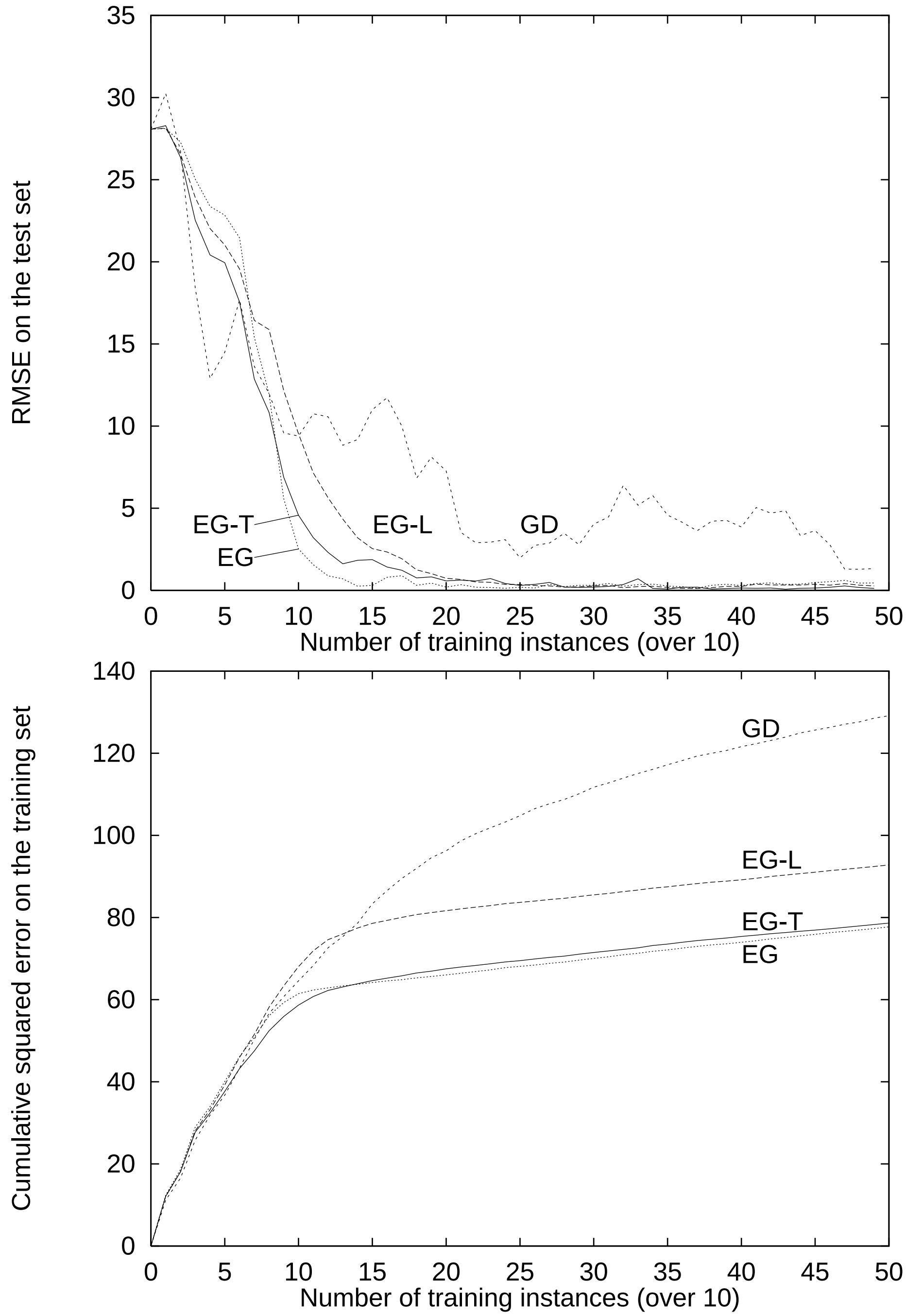

Figure 3. Supervised learning, linear unit, sparse target, 10% noise

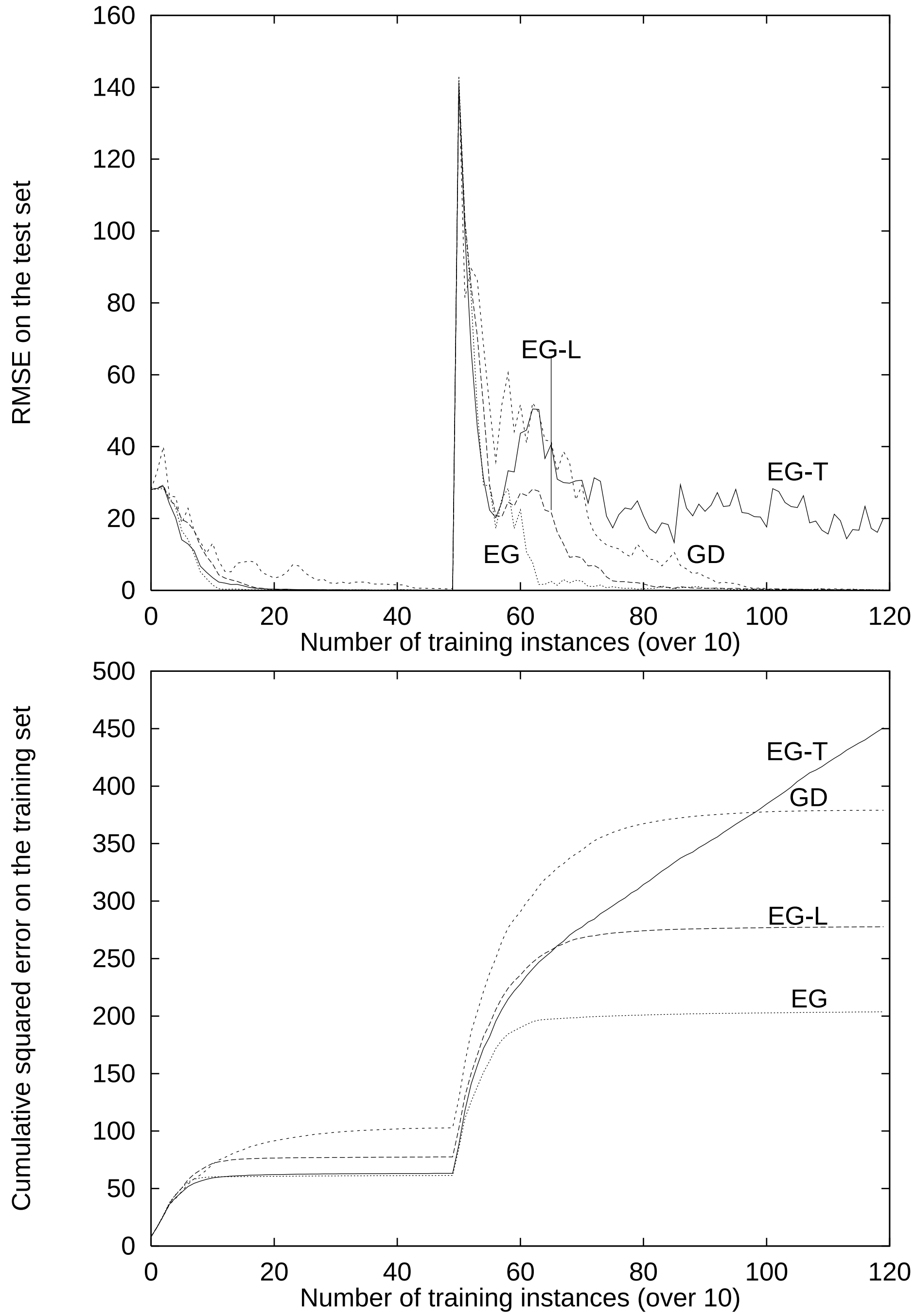

Figure 4. Supervised learning, linear unit, non-stationary target

found to improve performance; moreover, it can significantly degrade it in the non-stationary case.

4. Experiments with multilayer feed-forward neural nets

The extension of the EG update algorithm from a single neuron to feed-forward multilayered nets is straightforward. Assuming a sigmoid squashing function, the algorithm performs the following update

$$\log w_{i,j} := \log w_{i,j} + \delta_j$$

where

$$\delta_j := o_j(1-o_j)(y_j - o_j), \forall j \in outputs$$

$$\delta_j := o_j(1-o_j) \sum_{k \in outputs} w_{j,k}\delta_k, \forall j \in hidden$$

Some theoretical properties of EG updates used with sigmoid neurons are given in (Helmbold, Kivinen and Warmuth, 1995).

The learning task was a randomly generated function with weights in the interval [-1, 1], that can be represented by a 3-layered net with 5 hidden units. The function has 10 real-valued input variables that can take values between -1 and 1. The output is also a real value between -1 and 1.

The algorithms involved in the experiment were EG and the usual backpropagation algorithm (denoted GD). For EG, the same scheme of using duplicate negative inputs was used in order to allow for negative weights.

The experiment was performed on a 3-layered net with 10 input units, 5 hidden units and one output. All the units used a sigmoid squashing function. The training instances were generated by random sampling from the interval $[-1,1]^{10}$.

The learning rate was the same for all the units. The initial weights were 0 for GD and 1 for EG (thus effectively starting both algorithms at null weights).

In order to provide a fair comparaison, each algorithm was tested at 5 different learning rates.

Figure 5 presents the cumulative training error and the root mean squared error on an independent test set, for a typical learning curve obtained for a "best" learning rate.

EG has a smaller cumulated loss on the training set, but stabilizes at a higher error on the test set. The same pattern of behavior has been observed in several test problems. However, more experiments would be needed in order to form a conclusion for the multilayered net case.

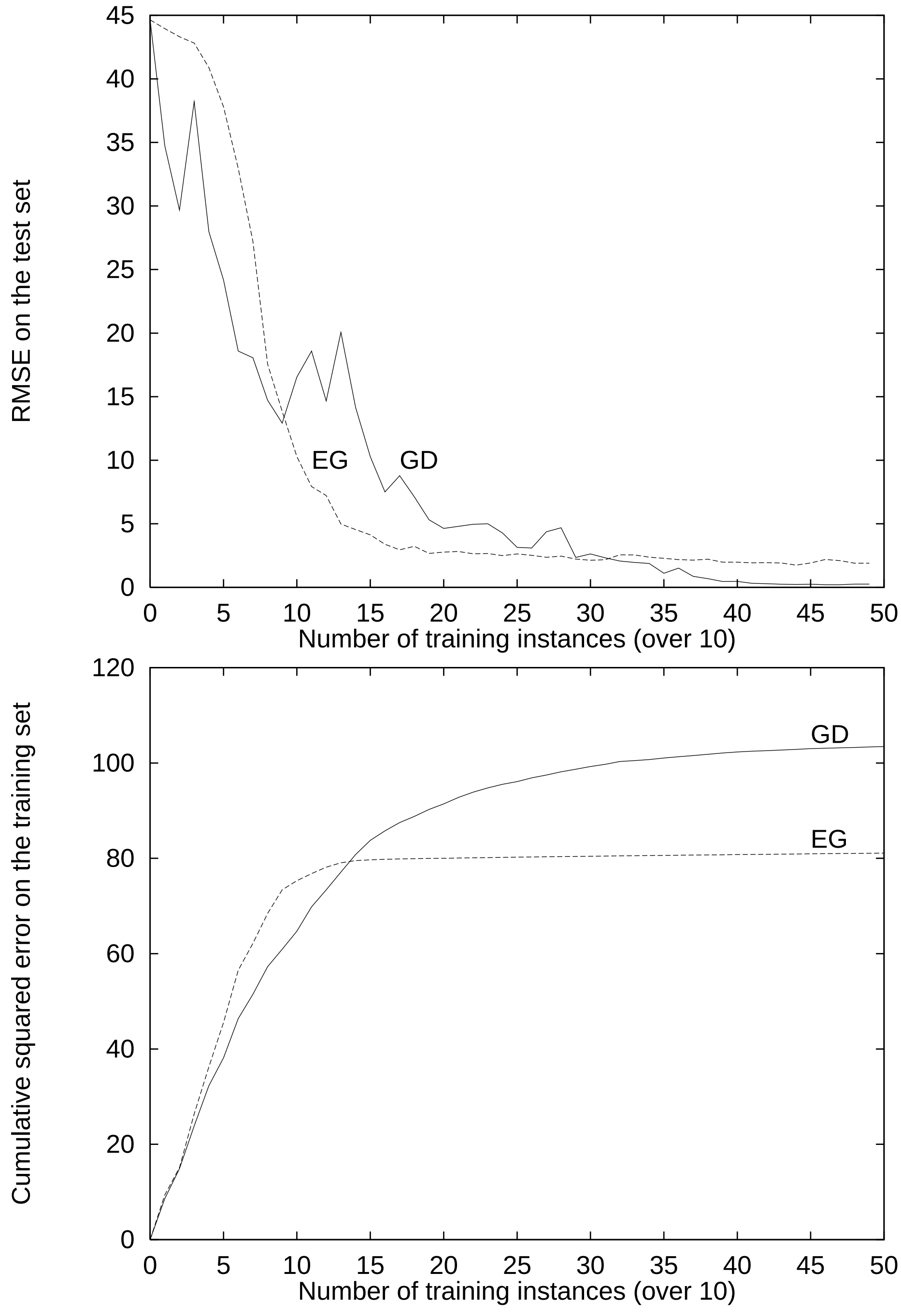

Figure 5. Supervised learning, multilayered net

5. Exponentiated gradient descent for reinforcement learning

EG updates can be adapted to work with reinforcement learning algorithms, by using the same type of update as in the linear supervised learning algorithm, but by computing a TD error instead of the training error. Thus, the weight update rule becomes:

$$\log w_i := \log w_i + \alpha \delta_t e_t$$

For state value computation,

$$\delta_t := r + \gamma V_t(s_{t+1}) - V_t(s_t)$$

In order to prevent the weights from getting too big, replace traces are used instead of accumulating traces (Singh and Sutton, 1996):

$$e_t(s) = \begin{cases} 1 & \text{if } s = s_t \\ \gamma \lambda e_t(s_t) & otherwise \end{cases}$$

The first task taken into consideration is a prediction task in a bounded continuous random walk problem. The walking interval is [0,1]. The starting point is always in the middle of the interval. The steps can have random lengths between -0.2 and 0.2. The outcome is 0 if exiting the interval through the left, and 1 if exiting on the right. There is no reward for any intermediate step. The discount factor is $\gamma = 1$.

Based on the theoretical properties of the supervised learning linear case, this problem is one in which EG should have a disadvantage with respect to GD, because all the features in this case are relevant and the weights have a linear variation.

Because the problem has a continuous state space, we used an approximate value function, implemented through a CMAC. The position, which is the only state variable in this case, was divided in into 10 evenly spaced intervals. An additional division was added, in order to allow for an initial offset of each tile. We used 10 tilings, with different randomly selected offsets. Thus, the algorithms have $11 * 11 * 10 = 1210$ input features. At each step, exactly 10 features are active (one for each tiling), representing the current state.

The detailed learning algorithm with EG updates is presented in figure 6. In order to allow for negative weights, we used the input duplication scheme described in section 2. We denote by w_i^+ and w_i^- the weights associated respectively to x_i and $-x_i$.

Both algorithms start at null actual weights. In order to account for the effect of the learning rate and step-size, there were several instances of the learning algorithms, generated for different values of λ and α. Each of them was treated as a separate algorithm, which ran for 10 trials. The performance measure for a

1. Initialisation:

$$w_i^+ := 1.0, w_i^- := 1.0, e_i := 0, \forall i \in CMAC - tiles$$

2. Start of trial:

$$s = 0.5$$

$$F := features(s)$$

3. Eligibility trace updating:

$$e_i = \lambda e_i, \forall i \in CMAC - tiles$$

$$e_i = 1, \forall i \in F$$

4. Make a random step $ds \in [-0.2, 0.2]$ and get the reward r.

$$s' := s + ds$$

$$F' := \begin{cases} features(s') & \text{if } s' \in (0,1) \\ \emptyset & \text{otherwise} \end{cases}$$

5. Update weights:

$$\log(w_i^+) := \log(w_i^+) + \frac{\alpha}{10}[r + \sum_{i \in F'}(w_i^+ - w_i^-) - \sum_{i \in F}(w_i^+ - w_i^-)]e_i$$

$$\log(w_i^-) := \log(w_i^-) - \frac{\alpha}{10}[r + \sum_{i \in F'}(w_i^+ - w_i^-) - \sum_{i \in F}(w_i^+ - w_i^-)]e_i$$

6. If $s' \in (0,1), F := F', s := s'$. Go to 3. Otherwise, go to 2.

Figure 6. TD prediction task learning algorithm

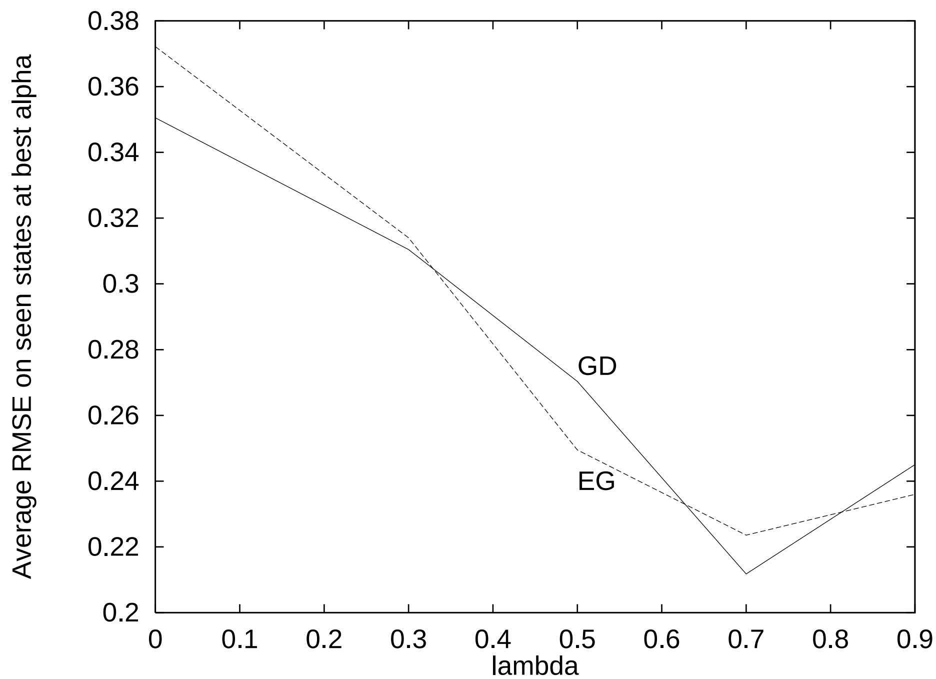

Figure 7. Summary of the performance of EG and GD updates on a continuous random task

trial was the RMSE between the correct predictions and the predictions made at the end of the trial for the states previously seen in that or a previous trial. These errors were then averaged over 10 trials and over 100 runs, to provide a measure of the performance of each algorithm.

Figure 7 shows the performance as a function of λ, at the best α value.

The EG and GD update rules yield similar errors at all the values of the learning rate. The differences are not statistically significant. As this type of target is preferred by GD, the result can be considered favourable to EG.

The second task was the mountain-car control problem, proposed by Moore (1991) (see figure 8). The objective of the car is to pass the top of the mountain. Because the gravity is stronger than the engine, the solution is to accelerate backwards first and then thrust forwards towards the goal. The detailed physics of the problem are given in (Singh and Sutton, 1996).

This is a minimum time control problem, so the reward is -1 for all time steps until the goal is reached, without any discounting. The three possible actions at each step are: accelerate forward, accelerate backward and no acceleration.

The algorithm applied to the problem is Sarsa (Rummery and Niranjan,

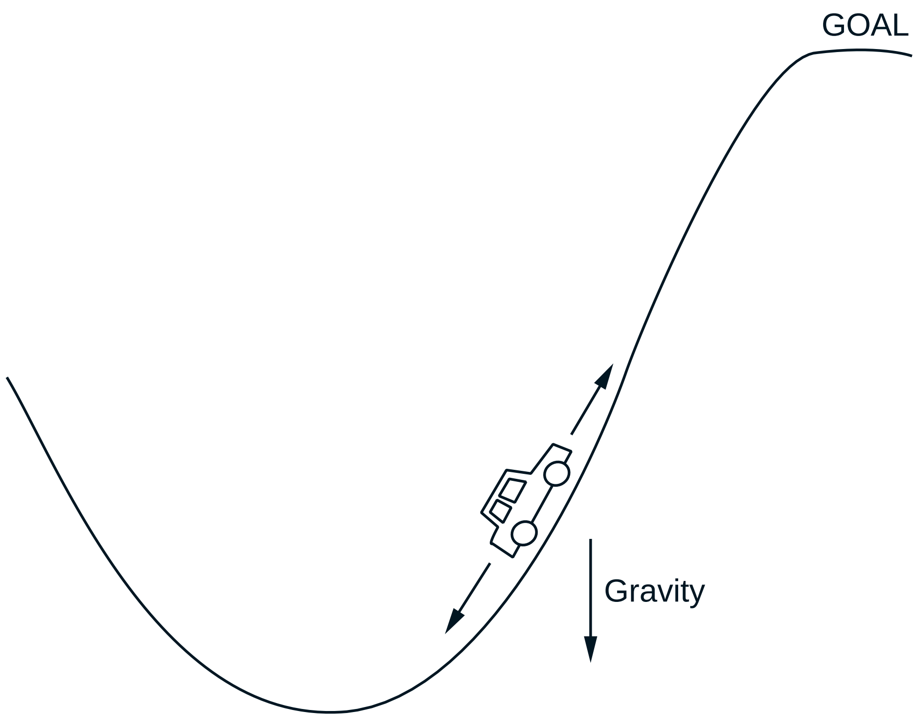

Figure 8. Mountain-car task

1994) with replacing eligibility traces. The algorithm with gradient descent additive updates is described in detail in (Singh and Sutton, 1996). The EG update algorithm is analogous, but uses the additive update in log weight space. For the implementation, we used Mahadevan's code as a strating point.

The mountain-car task has a continuous state space, with two state variables: position and velocity. We used a set of three CMACs, one for each action. Each CMAC has 5 tilings with random offsets. Each variable was evenly divided in 8 intervals, adding one additional division in order to be able to generate the offsets. Thus, there are $9 * 9 * 5 = 405$ input features, from which 5 are active simultaneously.

The initial values were 0 for all the states, which gives an optimistic estimate of the true values. This ensures exploration in the beginning, even when using a greedy policy for picking actions. For EG, the initial weight magnitude was 1.

The performance measure for a trial was the number of steps until reaching the goal, averaged over the first 20 trials, and then over 30 runs. A range of different λ and α values were used, in order to provide a fair comparaison. All

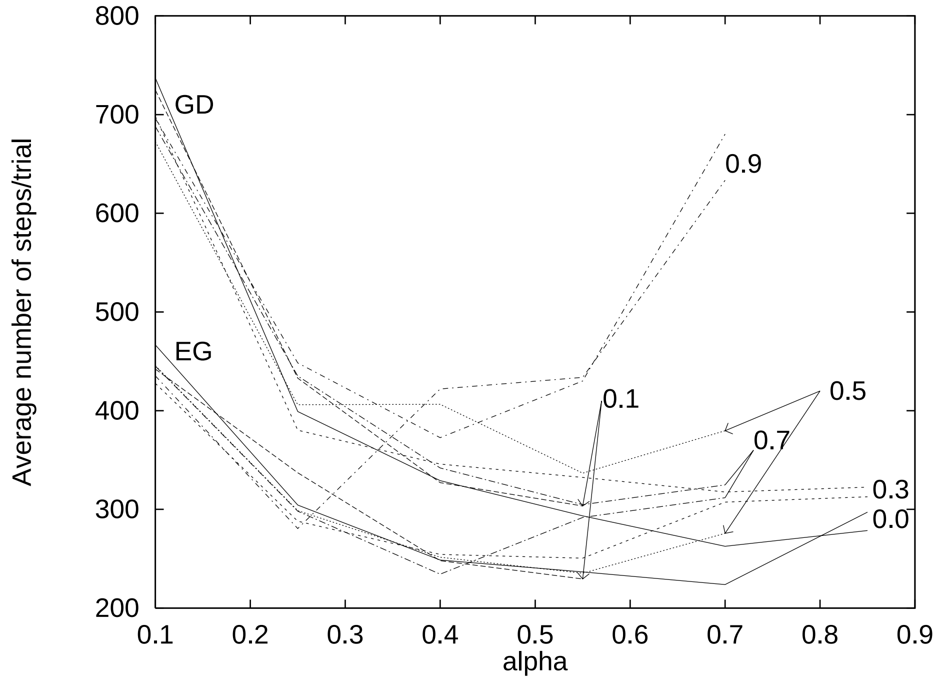

Figure 9. Results for the mountain-car problem, at several values of λ and α

the algorithm instances used the same random starting positions for the trials.

Figure 9 shows the detailed results for different values of λ and α. Figure 10 is a summary of the results, showing only the best performance of each algorithm at each λ value.

EG generates shorter trials than GD at almost all combinations of parameter settings. Moreover, for all λ values, EG is about 25% faster than GD at the best α, and the speed differences are statistically significant.

5. Conclusions

The experiments confirmed that exponentiated gradient methods yield faster convergence for supervised learning problems with linear targets that have lots of irrelevant input features. Moreover, EG is more robust to noise in the target value and adapts faster to target alterations in non-stationary problems. Normalization does not appear to improve performance.

The report also presents a straightforward extension of EG updates to reinforcement learning algorithms. In this case, replace traces are used instead of accumulating traces, in order to keep the weigths bounded. The experiments

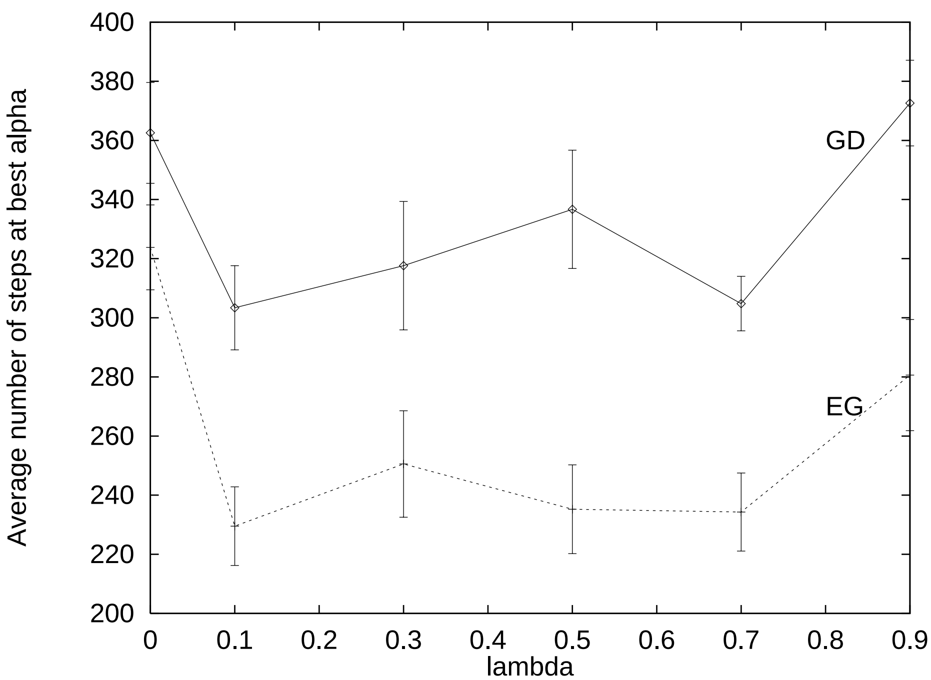

Figure 10. Summary results of the mountain-car problem, using the best α for each λ

suggest that EG updates yield faster reinforcement learning algorithms when used with linear function approximators. In the bigger task (mountain-car) EG has a 25% smaller average loss than the classic additive update rule. Parameter tuning is not significantly harder than in the case of multiplicative updates, although the initial magnitude of the weights impacts on the speed of convergence.

References

Kivinen J., Manfred K.W. (1994). Exponentiated gradient versus gradient descent for linear predictors. Technical Report UCSC-CRL-94-16, University of California, Santa Cruz, June 1994. (Short version appeared as "Additive versus exponentiated gradient updates for linear prediction" in Proceedings of the 27th Annual ACM Symposium on Theory of Computing, pages 209-218, The Association for Computing Machinery, New York, May 1995.)

Helmbold D.P., Kivinen J., and Warmuth M.K. (1995) Worst-case loss bounds for sigmoided neurons. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, Advances in Neural Information Precesses 8 (NIPS*95). MIT Press, Cambridge, Massachusetts: 309-315

Lewis, D.D., Schapire, R.E., Callan, J.P. and Papka, R. (1996). Training algorithms for linear text classifiers. In Proceedings of the Nineteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Zurich, Switzerland: 298-315.

Moore A.W. (1991). Variable Resolution Dynamic Programming: Efficiently learning action mapsin multivariate real-valued state-spaces. In Machine Learning: Proceedings of the Eighth International Workshop, San Mateo, CA. Morgan Kaufmann: 333-337.

Rummery G.A., Niranjan M. (1994). On-line Q-learning using connectionist systems. Technical report CUED/F-INFENG/TR 166, Cambridge University Engineering Dept.

Singh, S.P., Sutton, R.S. (1996). Reinforcement Learning with Replacing Eligibility Traces. Machine Learning 22: 123-158.