Adaptive Learning Rate Selection for Temporal Difference Learning

Harsh Gupta [1] **R. Srikant** [1] **Lei Ying** [2]

Abstract

Temporal Difference (TD) learning is a widely used class of algorithms in reinforcement learning. The success of TD learning algorithms relies heavily on the choice of the learning rate used. In this paper, we use recent results on finite-time performance of TD learning with linear function approximation to design an adaptive learning rate selection rule. The rule comprises a diagnostic test which determines whether the algorithm has reached the steady-state, i.e., the algorithm has stopped learning. Once such a determination is made, the rule adapts the learning rate (reducing the step size) so that the algorithm continues to learn. We implement our proposed rule on a variety of popular reinforcement learning applications and show that in all these scenarios, our rule outperforms the best fixed learning rate as well as other commonly used adaptive learning rate selection rules.

1. Introduction

Reinforcement learning (RL) refers to a collection of learning algorithms to compute near optimal policies in dynamic programming and Markov Decision Processes (MDP) when the underying model is unknown or too complicated to describe. Here, we consider a critical component of RL, which is the estimation of a value function of a given feedback policy in an MDP. Temporal difference (TD) learning is commonly used for this purpose, along with some form of function approximation to approximate the value function. Common function approximation schemes include linear function and neural network approximations. In this paper, we study TD learning with linear function approximation, given Markovian samples from an MDP operating under a fixed feedback/control policy.

The convergence rate of TD learning can vary significantly depending on the learning rate rule that is used. The three commonly studied learning rate rules are the following:

- *Diminishing learning rates:* The most widely studied learning rate rule in the theoretical literature is the diminishing learning rate rule motivated by the stochastic approximation theory. Under this rule, if ϵ_k denotes the learning rate at time-step k, then $\{\epsilon_k\}$ is chosen to satisfy $\sum_k \epsilon_k = \infty$ and $\sum_k \epsilon_k^2 < \infty$ (popularly known as a Robbins-Monro sequence). The reason for these conditions on the learning rates is that, under these conditions, it can be shown that the TD learning algorithm converges asymptotically with probability 1. However, this learning rate rule is typically not used in practice due to the fact that the learning rate quickly becomes very small and thus, parameter updates under TD learning change very slowly. In this family of learning rates, the most well-known is the *simple back-off strategy*, which sets $\epsilon_k = \frac{\alpha}{k}$ for some constant $\alpha > 0$. Note that we can replace k in the above strategy by k^β, where $\beta \in (\frac{1}{2}, 1]$, and still satisfy the Robbins-Monro sequence conditions (see (George & Powell, 2006) for a detailed discussion).

- *Constant learning rate:* The other extreme from diminishing learning rates is a constant learning rate, i.e., $\epsilon_k = \epsilon \quad \forall k$. The rationale here is that, since the learning rate is fixed, it cannot become arbitrarily small and hence, learning will never slow down. However, given a certain number of samples from which one has to learn the value function, it may be difficult to determine the optimal or appropriate fixed ϵ to use. If ϵ is "large," the learning proceeds quickly initially but the error at the end will be large, and on the other hand, if ϵ is "small," the transient phase of the learning algorithm will last a long time and one may not be able to get a desirable accuracy before the samples are exhausted.

- *Learning rate schedules:* In practice, most RL software developers use a learning rate schedule, i.e., a fixed ϵ is used for a certain amount of time and then it is reduced, and this is repeated a few times. The choice of when the learning rate ϵ should be reduced is determined by extensive experimentation and is chosen carefully for each RL environment studied. Typically, these are published by researchers on platforms such as github and then used by others.

[1]ECE and CSL, University of Illinois at Urbana-Champaign, USA [2]ECEE, Arizona State University, USA. Correspondence to: Harsh Gupta <hgupta10@illinois.edu>.

Real-world Sequential Decision Making workshop at ICML 2019.

Our goal in this paper is to adaptively choose the learning rate for TD learning with linear function approximation by observing the evolution of the function parameters as the algorithm progresses. For this purpose, we will leverage the recent results on finite-time performance bounds for TD-learning in (Srikant & Ying, 2019).

2. Background

We consider a discounted cost MDP, and let $\{X_k\}$ denote the Markov chain associated with the MDP operating under a fixed policy. Here it is assumed that X_k takes values in a finite state space. We let $V(i)$ denote the value function in state i and thus, it satisfies the Bellman equation

$$V(i) = c(i) + \alpha \sum_j p_{ij} V(j),$$

where c is the instantaneous cost, α is the discount factor, and p_{ij} is the probability of transitioning from state i to j.

The tabular version of the TD learning algorithm estimates V from samples X_k as follows:

$$V_{k+1}(X_k) = V_k(X_k) - \epsilon(V(X_k) - c(X_k) - \alpha V(X_{k+1})),$$

where ϵ is a fixed learning rate. Since the theory from which we derive our adaptive learning rate rule later is based on a fixed learning rate, we have presented the TD learning equation above in terms of a fixed learning rate, but other learning rate rules can be used as well. However, this version of TD learning requires one to store the value function for all states and when the state space is large, such a storage requirement makes the algorithm infeasible to implement. Instead, a common approach is to approximate the value function $V(i)$ by $\theta^T \phi(i)$, i.e., $V(i) \approx \theta^T \phi(i)$, where $\phi(i)$ is a known feature vector associated with state i and θ is the unknown parameter to be estimated.

The TD learning algorithm associated with linear function approximation is as follows:

$$\begin{aligned} \theta_{k+1} &= \theta_k \\ &\quad -\epsilon \left(\theta_k^T \phi(X_k) - c(X_k) - \alpha \theta_k^T \phi(X_{k+1})\right) \phi(X_k) \end{aligned} \tag{1}$$

Recently, a number of papers have studied the convergence properties of the above algorithms, under additional assumptions such as the i.i.d. samples assumption and the addition of a projection step (Lakshminarayanan & Szepesvari, 2018; Bhandari et al., 2018). Some of these papers also allow for averaging of the iterates.

The starting point for our work is the result in (Srikant & Ying, 2019) which studies the basic TD algorithm (1), with no significant additional assumptions, and provides convergence rate bounds. We first note that, as in (Srikant & Ying, 2019), we can rewrite (1) in a form that is more convenient for our subsequent discussion.

$$\theta_{k+1} - \theta^* = \theta_k - \theta^* + \epsilon \left(A(X_k)(\theta_k - \theta^*) + b(X_k)\right), \tag{2}$$

where $A(X_K)$ is a matrix whose entries are functions of the state X_k, $b(X_k)$ is a zero-mean random vector, and θ^* is the centering offset from 0, the equilibrium point of the ODE

$$\frac{d\theta}{dt} = E(A(X_\infty))\theta,$$

where X_∞ denotes a random variable with a distribution equal to the stationary distribution (which is assumed to exist) of the Markov chain X. In (Srikant & Ying, 2019), the linear stochastic approximation algorithm (2) was studied and the following result was established.

Theorem 1

$$\mathbb{E}(\|\theta_k - \theta^*\|^2) \leq K_1(1 - K_2\epsilon)^k + K_3\epsilon,$$

for all sufficiently small ϵ and large k, where K_1, K_2 and K_3 are problem-dependent constants. $\diamond$

3. Adaptive Learning Rate Rule

Suppose that our goal is to choose the learning rate as a function of the time step in order to minimize the mean-squared error for a given number of samples. In principle, one can assume time-varying learning rates, use the mean-squared error expressions for time-varying learning rates in (Srikant & Ying, 2019) (which are generalizations of Theorem 1), and try to optimize the learning rates to minimize the error for the given number of samples. However, this optimization problem is computationally intractable. We note that even if we assume that we are only going to change the learning rate a finite number of times, the resulting optimization problem of finding the times at which such changes are performed and finding the learning rate at these change points is an equally intractable optimization problem. Therefore, we have to devise simpler adaptive learning rate rules.

To motivate our learning rate rule, we first consider a time T such that error due to the transient and steady-state parts in Theorem 1 are equal, i.e.,

$$K_1(1 - K_2\epsilon)^T = K_3\epsilon \tag{3}$$

From this time onwards, running the TD algorithm any further with ϵ as the learning rate is not going to significantly improve the mean-squared error. In particular, the mean-squared error beyond this time is upper bounded by twice the steady-state error $K_3\epsilon$. Thus, at time T, it makes sense to reset ϵ as $\epsilon \leftarrow \epsilon/\xi$, where $\xi > 1$ is a hyperparameter. Roughly speaking, T is a time at which one is close to steady-state for a given learning rate, and therefore, it is the time to reduce the learning rate to get to a new "steady-state" with a smaller error.

The key difficulty with implementing the above idea is that it is difficult to determine T. If we were able to plot $\|\theta_k - \theta^*\|$ as a function of k, then one can devise some algorithm to determine whether this expression has reached close to its steady-state. However, θ^* is unknown and hence, it is difficult to use this approach.

Our second key idea is to estimate whether the algorithm is close to its steady-state by observing $\|\theta_k - \theta_0\|$ where θ_0 is our initial guess for the unknown parameter vector and is thus known to us. Note that $\|\theta_k - \theta_0\|$ is zero at $k = 0$ and will increase to $\|\theta^* - \theta_0\|$ in steady-state in the absence of any randomness, see Figure 1 for an illustration.

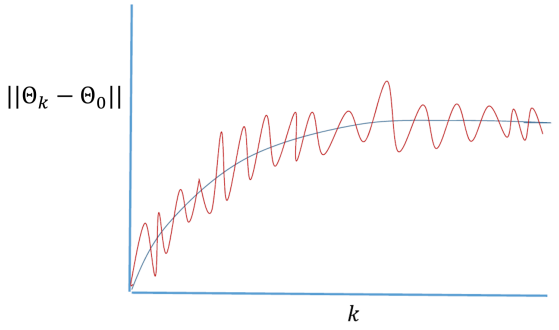

Figure 1. The evolution of $\|\theta_k - \theta_0\|$.

To derive a test to estimate whether $\|\theta_k - \theta_0\|$ has reached steady-state, we first note the following inequality for $k \geq T$ (i.e., after entering the steady-state time defined in (3)):

$$\begin{aligned} &\mathbb{E}[\|\theta_0 - \theta^*\|] - \mathbb{E}[\|\theta_k - \theta^*\|] \\ &\quad \leq \mathbb{E}[\|\theta_k - \theta_0\|] \leq \mathbb{E}[\|\theta_k - \theta^*\|] + \mathbb{E}[\|\theta_0 - \theta^*\|] \\ \Rightarrow\ & c - \sqrt{2K_3\epsilon} \leq \mathbb{E}[\|\theta_k - \theta_0\|] \leq c + \sqrt{2K_3\epsilon} \end{aligned} \tag{4}$$

where the first pair of inequalities follow from the triangle inequality and the second pair of inequalities follow from Theorem 1, Jensen's inequality and letting $c = \mathbb{E}[\|\theta_0 - \theta^*\|]$. Now, for $k \geq T$, consider the following N points: $\{x_i = i, y_i = \|\theta_{k+i} - \theta_0\|\}_{i=1}^N$. Since these points are all obtained after the steady-state is reached, if we draw the best-fit line through these points, its slope should be small. More precisely, let ψ_N denote the slope of the best-fit line passing through these N points. Using (4) along with formulas for the slope in linear regression, and after some algebraic manipulations, one can show that:

$$|\mathbb{E}[\psi_N]| = O\left(\frac{\sqrt{\epsilon}}{N}\right), \quad \text{Var}(\psi_N) = O\left(\frac{1}{N^2}\right) \tag{5}$$

Therefore, if $N \geq \frac{\kappa}{\sqrt{\epsilon}}$, then the slope of the best-fit line connecting $\{x_i, y_i\}$ will be $O\left(\frac{\sqrt{\epsilon}}{N}\right)$ with high probability (for a sufficiently large constant $\kappa > 0$). On the other hand, when the algorithm is in the transient state, the difference between $\|\theta_{k+m} - \theta_0\|$ and $\|\theta_k - \theta_0\|$ will be $O(m\epsilon)$ since θ_k in (2) changes by $O(\epsilon)$ from one time slot to the next. Using this fact, the slope of the best-fit line through N consecutive points in the transient state can be shown to be $O(\epsilon)$, similar to (5). Since we choose $N \geq \frac{\kappa}{\sqrt{\epsilon}}$, the slope of the best-fit line in steady state, i.e., $O\left(\frac{\sqrt{\epsilon}}{N}\right)$ will be lower than the slope of the best-fit line in the transient phase, i.e., $O(\epsilon)$ (for a sufficiently large κ). We use this fact as a diagnostic test to determine whether or not the algorithm has entered steady-state. If the diagnostic test returns true, we update the learning rate (see Algorithm 1).

Algorithm 1 Adaptive Learning Rate Rule

Hyperparameters: α, σ, ξ, N
Initialize $\epsilon = \alpha$, $\psi_N = 2\sigma\sqrt{\epsilon}$, θ_0, $\theta_{\text{ini}} = \theta_0$.
for $i = 1, 2, \ldots$ **do**
  Do TD learning update.
  Compute $\psi_N = \text{Slope}\left(\{k, \|\theta_{i-k} - \theta_{\text{ini}}\|\}_{k=0}^{N-1}\right)$.
  if $|\psi_N| < \frac{\sigma\sqrt{\epsilon}}{N}$ **then**
    $\epsilon = \frac{\epsilon}{\xi}$.
    $\theta_{\text{ini}} = \theta_i$.
  end if
end for

4. Experiments

4.1. Setup

We implemented our adaptive learning rate selection rule on a variety of popular reinforcement learning domains and compared its performance with that of the best fixed learning rate and the best diminishing/back-off learning rate schedule (as discussed in Section 1). We consider the following reinforcement learning problems/domains[1]:

1. **Mountain Car:** In the basic mountain car problem, an underpowered car is positioned in a valley between two mountains on a one-dimensional track. The aim of the problem is to drive the car to the top of the mountain on the right-hand side, but the engine power available is insufficient to simply accelerate and power through to the top. Therefore, a player has to build up momentum by going back and forth between the two mountains until the car has sufficient momentum to reach its goal. The state space, action space, cost structure and initialization details for the mountain car problem are as follows:
   - *State Space:* (Car Position, Car Velocity) $\in [-1.2, 0.6] \times [-0.07, 0.07]$.

[1]We used the OpenAI Gym implementation of these environments, available at https://gym.openai.com/.

- *Action Space:* 0, 1 and 2 (denoting left, no and right acceleration respectively).
- *Cost Structure:* $+1$ cost incurred for every time step the car has not achieved its goal. 0 cost incurred upon reaching the goal.
- *Initialization/Starting State:* The car's position is initialized to a random value in $[-0.6, 0.4]$. Its velocity is initialized to 0.

2. **Inverted Pendulum:** In the classic inverted pendulum swing-up problem, a frictionless pendulum is hinged/pivoted at one end and the aim of the problem is to keep the pendulum in an upright position (with respect to the pivot) for as long as possible by applying a torque at the pivot point (sometimes referred to as the joint effort). The state space, action space, cost structure and initialization details for the inverted pendulum problem are as follows:
   - *State Space:* $(\cos(\theta), \sin(\theta), \dot{\theta}) \in [-1.0, 1.0] \times [-1.0, 1.0] \times [-8.0, 8.0]$. Here, $\theta \in [-\pi, \pi]$ denotes the angular position of the pendulum with respect to the pivot.
   - *Action Space:* Torque $\in [-2.0, 2.0]$.
   - *Cost Structure:* The equation associated with the cost function is the following:
   $$\theta^2 + 0.1\dot{\theta} + 0.001 \times \text{torque}^2.$$
   - *Initialization/Starting State:* The pendulum's angular position is initialized to a random value in $[-\pi, \pi]$. Its angular velocity is initialized to a random value $\in [-1, 1]$.
3. **Frozen Lake:** The frozen lake game is a single player game in which the player has to navigate a 4×4 grid with the goal of arriving at the finish point on the grid from the starting position without falling in any of the death zones (also known as holes). At each point on the grid, the player has the option to move around to any of the neighboring points. If the player moves to a hole, the player is killed and re-spawned at the starting point. The state space, action space, cost structure and initialization details for the frozen lake game are as follows:
   - *State Space:* A set with a cardinality of 16, i.e., the total number of points on the grid.
   - *Action Space:* Up, Down, Left or Right. Note that if the point on the grid is such that it does not have a neighbor in a particular direction, moving in that direction will keep the player at his/her current position.
   - *Cost Structure:* 0 cost associated with every time step until the player reaches his/her goal. -1 cost incurred upon reaching the goal.
   - *Initialization/Starting State:* The player is always initialized to state 0 which is the starting position on the grid.

We evaluate the following policies using TD learning for each aforementioned problem/domain:

- Mountain Car - At each time step, choose a random action $\in \{0, 2\}$, i.e., accelerate randomly to the left or right.
- Inverted Pendulum - At each time step, choose a random action in the entire action space, i.e., apply a random torque $\in [-2.0, 2.0]$ at the pivot point.
- Frozen Lake - At each time step, pick a random action in the entire action space, i.e., go in a random direction from the current position.

Since the true value of θ^* is not known in the problem domains we consider, to quantify the performance of the TD learning algorithm, we use the error metric known as the *norm of the expected TD update* (NEU, see (Sutton et al., 2009) for more details). For linear function approximation, in the Mountain Car and the Inverted Pendulum problems, we use a $O(3)$ Fourier basis (see (Konidaris et al., 2011) for more details) and for the Frozen Lake game, we use one-hot encoding for the different states. Also, for all the experiments, we use $\gamma = 0.95$ as the discount factor.

4.2. Hyperparameter Tuning

For each learning rate rule, we did the following tuning to choose the best hyperparameters:

- For finding the best fixed learning rate in each problem domain, we did a grid search over a wide variety of fixed learning rates and found the following to be the best: *Mountain Car* - 0.0075, *Inverted Pendulum* - 0.001, *Frozen Lake* - 0.01.
- For finding the best back-off strategy ($\epsilon_k = \max\{\epsilon_{\min}, \frac{\alpha}{k^\beta}\}$), we first fixed α and conducted a grid search to find the best β. We considered the following values for β: $(0.6, 0.7, 0.8, 0.9, 1.0)$. In all the problem domains, the best performance was obtained for $\beta = 0.6$. We subsequently conducted a grid search for α over a wide variety of values and found the following to be the best: *Mountain Car* - 0.5, *Inverted Pendulum* - 0.2, *Frozen Lake* - 1.
- For the proposed adaptive learning rate rule, we fixed $\xi = 1.2, N = 200$ in all the domains since we didn't want the decay in the learning rate to be too aggressive and the resource consumption for slope computation to be high. We then fixed α and conducted a grid search over a wide variety of values to find the best σ. Subsequently, we conducted a grid search over

α. Interestingly, the adaptive learning rate rule was reasonably robust to the value of α used and hence we use $\alpha = 0.05$ in all the domains. Effectively, the only hyperparameter that really required tuning was σ. The following values for σ were found to be the best: *Mountain Car* - $\sigma = 0.0075$, *Inverted Pendulum* - $\sigma = 10$, *Frozen Lake* - $\sigma = 0.01$.

Note that for each learning rate rule we ensure that the learning rate never goes below $\epsilon_{\min} = 10^{-5}$. This is done to ensure that the TD learning algorithm does not stop learning altogether.

4.3. Results

For each experiment, one run involved the following: $10,000$ episodes with the number of iterations in each episode varying from $50 - 200$ depending on the problem/domain. After every $1,000$ episodes, training/learning was paused and the NEU was computed, averaged over $1,000$ test episodes. Depending on the problem/domain, $50 - 250$ of such runs were conducted and the results averaged over these runs. Note that the learning rate for each adaptive strategy was adapted at the episodic level due to the episodic nature of the problems. The results are reported in Figures 2 - 4. We report the performance of the different rules from the $5,000^{\text{th}}$ episode onward since the large error values in the initial episodes make it hard to visually interpret the graph for the eventual performance of the different rules.

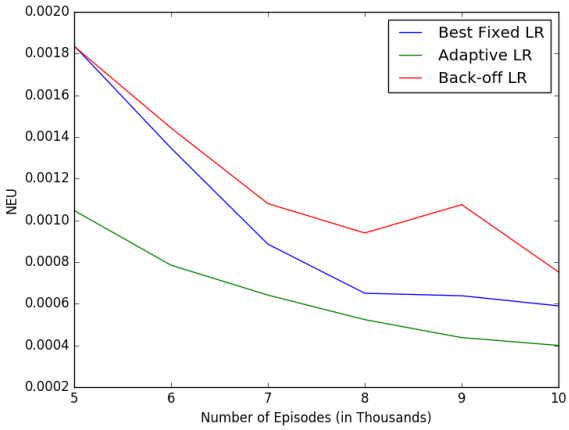

Figure 2. Performance of different learning rate rules in the Mountain Car problem.

From the figures, we can observe that our rule performs much better than the best fixed learning rate as well as the best back-off learning rate strategy. Note that since we tuned each strategy to have the lowest error at the end of $10,000$ episodes, the relative difference between the performance of our strategy and others is significant.

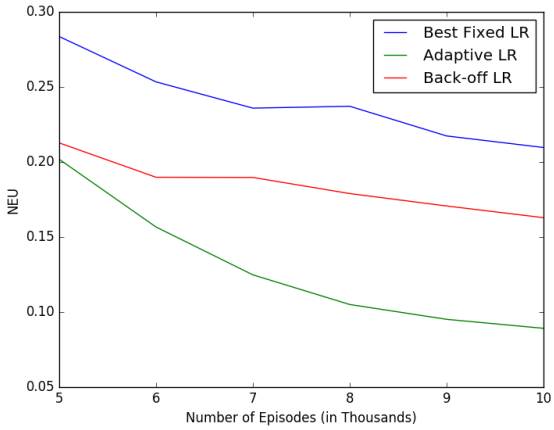

Figure 3. Performance of different learning rate rules in the Inverted Pendulum problem.

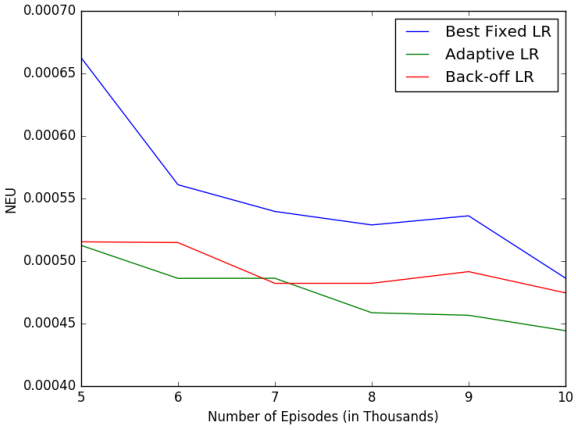

Figure 4. Performance of different learning rate rules in the Frozen Lake problem.

Remark 1 *We also implemented a procedure similar to SGD$^{1/2}$ in (Chee & Toulis, 2018), which was proposed recently and has a similar motivation as our idea but a different diagnostic test. We do not report the results of that procedure here as in all the domains that we considered, their diagnostic test was too conservative and failed to detect the steady state in a reasonable amount of time resulting in a much worse performance as compared to our method.*

5. Conclusion

In this paper, we presented an adaptive learning rate selection rule for TD learning with linear function approximation. The rule comprises a diagnostic test to determine whether the algorithm has entered steady state and adapts the learning rate accordingly. We implemented the rule on various popular reinforcement learning applications and demonstrated its significant utility in practice.

References

Bhandari, J., Russo, D., and Singal, R. A finite time analysis of temporal difference learning with linear function approximation. In *Conference On Learning Theory*, pp. 1691–1692, 2018.

Chee, J. and Toulis, P. Convergence diagnostics for stochastic gradient descent with constant learning rate. In *International Conference on Artificial Intelligence and Statistics*, pp. 1476–1485, 2018.

George, A. P. and Powell, W. B. Adaptive stepsizes for recursive estimation with applications in approximate dynamic programming. *Machine learning*, 65(1):167–198, 2006.

Konidaris, G., Osentoski, S., and Thomas, P. Value function approximation in reinforcement learning using the fourier basis. In *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

Lakshminarayanan, C. and Szepesvari, C. Linear stochastic approximation: How far does constant step-size and iterate averaging go? In *International Conference on Artificial Intelligence and Statistics*, pp. 1347–1355, 2018.

Srikant, R. and Ying, L. Finite-time error bounds for linear stochastic approximation and TD learning. *Conference on Learning Theory (COLT)*, 2019. ArXiv preprint arXiv:1902.00923,.

Sutton, R. S., Maei, H. R., Precup, D., Bhatnagar, S., Silver, D., Szepesvári, C., and Wiewiora, E. Fast gradient-descent methods for temporal-difference learning with linear function approximation. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pp. 993–1000. ACM, 2009.