LEARNING TO FIGHT

Thore Graepel, Ralf Herbrich, Julian Gold
Microsoft Research Ltd.
7 J J Thomson Avenue, CB3 0FB
Cambridge, UK
Email: {thoreg|rherb|jgold}@microsoft.com

KEYWORDS

Reinforcement Learning, Fighting Games, SARSA, Q-Learning, Markov Decision Process

ABSTRACT

We apply reinforcement learning to the problem of finding good policies for a fighting agent in a commercial computer game. The learning agent is trained using the SARSA algorithm for on-policy learning of an action-value function represented by linear and neural network function approximators. We discuss the selection and construction of features, actions, and rewards as well as other design choices necessary to integrate the learning process into the game. The learning agent is trained against the built-in AI of the game with different rewards encouraging aggressive or defensive behaviour. We show that the learning agent finds interesting (and partly near optimal) policies in accordance with the reward functions provided. We also discuss the particular challenges arising in the application of reinforcement learning to the domain of computer games.

INTRODUCTION

Computer games constitute a very interesting domain for the application of machine learning techniques. Games can often be considered simulations of (aspects of) reality (Salen and Zimmerman, 2004). As a consequence, modelling the behaviour of agents in computer games may capture aspects of behaviour in the real world. Also, the competitive and interactive nature of games allows the exploration of policies in a rich dynamic environment. In contrast to modelling behaviour in the real world, there are (at least theoretically) two great advantages enjoyed by a simulation/game approach: i.) full control of the game universe including full observability of the state ii.) reproducibility of experimental settings and results.

Computer games provide one of the rather few domains in which artificial intelligence (game AI) is currently applied in practice. That said, it is a common complaint of *gamers* that the game AI behaves either in boring ways or is too strong or too weak to provide interesting and entertaining game play. Hence, adaptive game AI has the potential of making games more interesting and ultimately more fun to play. This is particularly true since the sophistication of other areas of computer games such as sound, graphics, and physics have leapt ahead of AI in recent years and it is anticipated that advances in game AI will be a considerable driving force for the games market in the future. In fact, games such as *Creatures*, *Black and White* and *Virtua Fighter 4* were designed around the notion of adaptation and learning.

Machine learning can be applied in different computer game related scenarios (Rabin, 2002). *Supervised learning* can be used to perform *be-*

havioural cloning of the player, e.g., to represent him as an avatar at times when he is not available in person for a multi-player game (this includes mitigating latency in network-based games). However, the most appealing application of learning may be a non-player character (NPC) that adapts by *reinforcement learning* (Sutton and Barto, 1998) in response to the opponent's behaviour and the environment during game play (see Stone and Sutton, 2001 for a RoboCup application), or even an agent who learns by playing against a clone of himself (see Tesauro, 1995 for a Backgammon application). Alternatively, such an adapting NPC may be useful at development time to create built-in AI adapted to varying conditions, or even to systematically test built-in AI for exploitable weaknesses.

In this paper we apply the SARSA reinforcement learning algorithm (Rummery and Niranjan, 1994; Sutton and Barto, 1998) together with a linear action-value function approximator to *Tao Feng*, a state-of-the-art commercial fighting game released for the Xbox game platform in 2003. Fighting games constitute a classical genre of computer games in which two opponents carry out a martial arts fight. Tao Feng provides about 12 different fighting characters with varying styles, combat taking place in 10 different arenas. There are over 100 different actions (moves and combo attacks) available to the player. The game comprises 350 000 lines of code of which 64 000 constitute the built-in AI, which is implemented as a non-deterministic finite-state machine.

A particular focus of the paper will be a discussion of the challenges that had to be met in order to adapt standard reinforcement learning to a real-time computer game and integrate the learning process into such a complex code base. The paper is structured as follows: We give a brief introduction to reinforcement learning with an emphasis on learning the action-value function. Then, we describe and discuss the specific choices made for applying reinforcement learning to Tao Feng. Finally, we present and discuss experimental results.

REINFORCEMENT LEARNING AND THE ACTION-VALUE FUNCTION

Markov Decision Processes

We model the agent's decision and learning process in the framework of reinforcement learning (Sutton and Barto, 1998) which aims at finding an (near) optimal policy for an agent acting in a Markov decision process (MDP). An MDP is characterised by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ with

1. A *state space* $\mathcal{S}$ with states $s \in \mathcal{S}$. In the most straight-forward case $\mathcal{S}$ is a finite set. In Tao Feng the state can be represented by nominal features such as physical situation of a player (*on the ground*, *in the air*, *knocked*) or spatial features (wall behind, wall to the right etc.) However, depending on the representation chosen, real-valued state features such as distance between players or state of the health bar are conceivable as well.

2. An *action space* $\mathcal{A}$ with actions $a \in \mathcal{A}$. We will only consider the case of $\mathcal{A}$ being a finite set. More precisely, we are dealing with action spaces $\mathcal{A}(s)$ that depend on the current state s. Typical actions in Tao Feng include punches, kicks, throws, blocks and combo moves.

3. An unknown stochastic *transition dynamics* $\mathcal{T}^a_{s,s'} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$ which gives the probability of a transition from state s to state s' if action $a \in \mathcal{A}(s)$ is taken,

$$\mathcal{T}^a_{s,s'} := \mathbf{P}_{s_{t+1}|s_t=s,a_t=a}\left(s'\right) . \tag{1}$$

In Tao Feng, the dynamics is given by the (partially stochastic) state machine that drives the game. In particular, the dynamics derives from the combination of the game engine and the built-in AI of the game.

4. An *average reward function* $\mathcal{R}^{a}_{s,s'} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ which assigns a reward to the agent if it carries out action $a \in \mathcal{A}(s)$ in state s and ends up in state s',
$$\mathcal{R}^{a}_{s,s'} := \mathbf{E}_{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'} [r] \ . \tag{2}$$
The reward is a key element for learning, because it provides the feedback signal on which the learning agent can improve. In Tao Feng the reward is typically tied to the health-bar, the traditional goal of the game being to decrease the opponent's health-bar while maintaining one's own.

As seen above, a Markov decision process models some relevant aspects of the fighting agent's situation. However, one must keep in mind that two important aspects are neglected in this model:

1. The state space used in practice provides only an approximation to the true and complete state, parts of which remain unobservable (see, e.g., Kaelbing et al., 1998 for a discussion of partially observable MDPs). The missing state information includes details of the environment as well as hidden states in the opponent's built-in AI. However, the problem is partly overcome by the assumption of a stochastic transition dynamics which allows us to model the resulting uncertainty as a noise process.

2. The MDP model is ignorant of the adversarial aspects of fighting in that the behaviour of the opponent is simply captured by the transition dynamics $\mathcal{T}$ given in (1). Depending on the nature of the game AI it might be more appropriate to consider game-theoretic models that take into account that there is more than one decision-making agent involved (see, e.g., Filar and Vrieze, 1996 on competitive MDPs)

Learning in Markov Decision Processes

The goal of the agent is to devise a sequence of actions $(a_t)_{t=0}^{\infty}$ so as to maximise his average expected reward. In order to make the agent autonomous in a given environment it must be equipped with a (stochastic) *policy* $\pi : \mathcal{S} \times \mathcal{A} \to [0, 1]$ which prescribes the probability of taking action $a \in \mathcal{A}(s)$ in state $s \in \mathcal{S}$, satisfying $\sum_{a \in \mathcal{A}(s)} \pi(s, a) = 1$ for all states $s \in \mathcal{S}$. A typical goal of reinforcement learning is to find a policy π that maximises the *discounted return*
$$R_t := r_{t+1} + \gamma r_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \ , \tag{3}$$
where $0 \leq \gamma < 1$ is called the discount rate. The infinite sum R_t takes finite values for $\gamma < 1$ (convergence of geometric series), $\gamma = 0$ corresponding to a "myopic" agent and larger values of γ increasing the planning horizon.

We will focus on methods involving the state-action value function $Q^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ for a given policy π defined as
$$\begin{aligned} Q^{\pi}(s, a) &:= \mathbf{E}_{\pi | s_t = s, a_t = a} [R_t] \qquad (4) \\ &= \mathbf{E}_{\pi | s_t = s, a_t = a} \left[\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \right] . \end{aligned}$$
Its value indicates how beneficial (in terms of future expected discounted reward) it is for the agent to take action $a \in \mathcal{A}(s)$ when in state s. We prefer the state-action value function $Q^{\pi} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ to the state-value function $V^{\pi} : \mathcal{S} \to \mathbb{R}$, because Q^{π} immediately provides a policy without requiring a model of the dynamics $\mathcal{T}$.

The "optimal" way of using Q^{π} for generating a policy is by choosing the action a^* with the highest associated Q^{π} value in a given state s. However, this choice is optimal only with respect to exploiting current knowledge. In order to be successful in the long run we need to balance *exploration* and *exploitation*. We use the *soft-max or Gibbs policy*
$$\pi(s, a) := \frac{\exp(\beta Q(s, a))}{\sum_{a' \in \mathcal{A}(s)} \exp(\beta Q(s, a'))} \ , \tag{5}$$
where the "temperature" parameter $\beta \geq 0$ determines how peaked the probability distribution is around a^*.

SARSA and Q-Learning

The SARSA algorithm (Rummery and Niranjan, 1994) is an on-policy temporal difference learning algorithm for Markov decision processes (Sutton and Barto, 1998). It is based on the following update equation

$$Q(s,a) \leftarrow (1-\alpha)\, Q(s,a) + \alpha \left[r + \gamma Q\left(s', a'\right) \right],$$

where $0 \le \alpha \le 1$ is a learning rate parameter and $0 \le \gamma < 1$ is the discount factor for future rewards. The update equation states that for a given state-action pair $(s,a) \in \mathcal{S} \times A$ the new state-action value is obtained by adding a small (depending on α) correction to the old value. The correction is the difference between the immediate reward r increased by the discounted future state-action value $\gamma Q\left(s', a'\right)$ and the old state-action value $Q(s,a)$. SARSA (s,a,r,s',a') is an on-policy learning algorithm in the sense that it estimates the value of the same policy that it is using for control. Q-Learning (Watkins and Dayan, 1992) constitutes an off-policy alternative to SARSA and replaces the term $\gamma Q\left(s', a'\right)$ by $\gamma \max_{a' \in \mathcal{A}(s')} Q\left(s', a'\right)$ in the above equation. This allows for separating the policy being evaluated from the policy used for control.

The update equation as given assumes a tabular representation of the Q function. In practice, even for small problems such a representation is unsuitable for learning because an unrealistic amount of data would be necessary to estimate all the independent table entries reliably. As a consequence we decided to represent the state-action value function $Q(s,a)$ with different function approximators (see Sutton and Barto, 1998).

Algorithm 1 SARSA with linear function approximator and game embedding

```
Require: Learning rate $0 < \alpha$, SoftMax parameter $\beta > 0$ and Discount rate $\gamma < 1$
Require: functions getValidActions(), getStateVector(), getReward()
Require: functions submitAction($a$), getExecutedAction($a'$)
Require: function selectSoftMaxAction($\mathbf{s}', A, \{\mathbf{w}_a\}, \beta$) (see Equation (5))
  Initialise $\forall \tilde{a} : \mathbf{w}_{\tilde{a}} \leftarrow \mathbf{0}$ and set $a \leftarrow$ undefined,
  for every frame in turn do
    $\mathbf{s}' \leftarrow$ getStateVector()
    $A \leftarrow$ getValidActions()
    $a' \leftarrow$ selectSoftMaxAction($\mathbf{s}', A, \{\mathbf{w}_a\}, \beta$)
    submitAction($a'$)
    if getExecutedAction($a'$) $\neq a'$ then
      Continue
    end if
    if $a \neq$ undefined then
      $\mathbf{w}_a \leftarrow \mathbf{w}_a + \alpha \left(r + \gamma \mathbf{w}_{a'}^\top \mathbf{s}' - \mathbf{w}_a^\top \mathbf{s}\right) \mathbf{s}$
      $r \leftarrow$ getReward($\mathbf{s}, \mathbf{s}'$)
      $\mathbf{s} \leftarrow \mathbf{s}', a \leftarrow a'$
    end if
  end for
```

REINFORCEMENT LEARNING IN TAO FENG

The goal of our project was to develop a learning fighter that starts at a level of ignorance comparable to a human beginner, plays Tao Feng either against the built-in AI or against a human player, and develops fighting skills corresponding to the reward function provided.

Choice of Learning Algorithm

The choice of the learning algorithm was mostly determined by the design of Tao Feng. Although we were in possession of the full code base of the game, in practice, our ability to control the player as well as to observe the environment was severely limited by the structure of the code and the concurrency of processes. Hence, at any point in time we know neither the exact state nor the exact transition dynamics (1) and reward function (2) for Tao Feng. The application of methods based on the state value function would require us to learn a separate model of the Tao Feng dynamics, thus introducing a layer of complication.

Although we originally intended to apply Q-Learning (Watkins and Dayan, 1992) it turned out to be very difficult to reliably determine the set $A(s)$ of available actions for the evaluation of $\gamma \max_{a' \in \mathcal{A}(s')} Q(s', a')$. The reason for this complication lies in the graphical animation system of Tao Feng, which rejects certain actions depending on the animation state. As a consequence, it is only possible to submit a given action a (using submitAction(a)) and to check (using getExecutedAction(a)) which action has actually been performed. We chose the SARSA algorithm (Rummery and Niranjan, 1994) because it does not require knowledge of $A(s)$.

Choice of Features, Actions, and Rewards

Features There are essentially three groups of useful features, environment-related, opponent-related and agent-related features. Environment-related features such as "blocked behind" are designed to facilitate navigation of the arena. Opponent-related features such as "opponent's physical state" and "distance to opponent" are designed to make the agent react to the opponent. Finally, features related to the learning fighters themselves such as "my previous action" and "my physical state" may serve to give the agent's actions continuity and consistency.

Actions While the game provides over 100 different actions, we focused on a number of atomic actions such as simple punches, kicks and throws. In addition, we constructed meta-actions that are composed of repeated atomic actions such as `block`, `stepleft`, `stepright` and `lungeback`. This was necessary because some of these actions have a very short duration and their execution in isolation has almost no effect. As an example, we constructed a meta action `block25` that holds up the block for 25 frames corresponding to half a second.

Rewards In order to assess rewards it is necessary to define the end of a round. Since in Tao Feng the two opponents do not act in sync (multi-threading) there is no clear definition of a round. We assign reward to the agent only when the subsequent action has been successfully selected thus indicating completion of the previous one. As a consequence, rounds have different durations Δt (measured in seconds) over which the reward-per-action r is spread out. We take this into account by considering the rate of reward $r/\Delta t$ in the learning.

Implementation Issues

The integration of the learning algorithms into the code base was hindered by the complex multi-threaded and animation centred architecture of the system. Systematic monitoring of the learning process was only possible because we devised an on-line monitoring tool that continuously sent data such as rewards, actions and parameters from the Xbox via the network to a PC, where the data was analysed and visualised in Matlab. Although the implementation as such is not planned to be productised, it served as a test-bed for a library of reusable modules including function approximators (look-up tables, linear function approximation, and neural network) and learning algorithms (Q-Learning and SARSA), which are suitable for use in future Xbox games.

EXPERIMENTS

We performed experiments in order to see if we could learn a good policy for fighting against the built-in AI. We employed the SARSA learning algorithm with function approximation as detailed in Algorithm 1. Throughout we used the parameter settings $\alpha = 0.01$ and $\gamma = 0.8$. We used two types of reward functions depending on the change

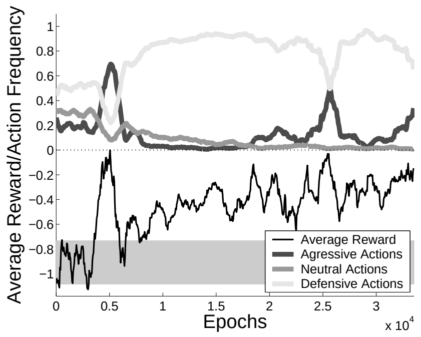

Figure 1: Average rewards and frequency of selected action classes for a Gibbs policy with $\beta = 2$ in the aggressive setting using *linear* function approximators. The gray band indicates performance of a uniformly random policy (mean $\pm$ 2 standard deviations)

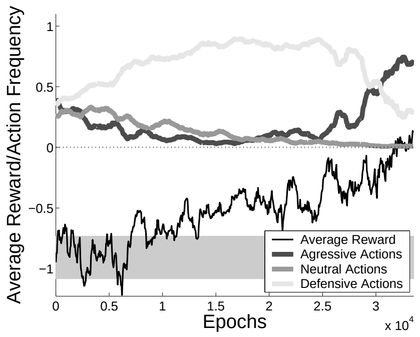

Figure 2: Average rewards and frequency of selected action classes for a Gibbs policy with $\beta = 1$ in the aggressive setting using *linear* function approximators. The gray band indicates performance of a uniformly random policy (mean $\pm$ 2 standard deviations)

in health of both combatants, ΔH_{self} and ΔH_{opp}. In order to encourage an aggressive fighting style we defined $r_{\text{aggressive}} := 0.7 \times \Delta H_{\text{opp}} - 0.3 \times \Delta H_{\text{self}}$. In contrast, we also defined a peace-encouraging reward function $r_{\text{aikido}} := -0.5 \times \left(\Delta H_{\text{opp}} + \Delta H_{\text{self}}\right)$.

In order to represent the game state s the learning agent used the following 15 features, which were grouped into a feature vector $\mathbf{s} \in \mathbb{R}^3 \times \{0, 1\}^{12}$ suitable for input to the function approximators used:

- Distance to opponent coded in terms of three real-valued features based on unit-variance Gaussians placed at 1, 3 and 5 meters,
- 4 binary features coding which of the four sides of the agents are blocked,
- 6 binary features coding the previous action of the opponent,
- 2 binary features coding the physical situation of the opponent player (`in air` and `knocked`).

The learning agent was equipped with three classes of actions:

- Aggressive: `throw`, `kicktrail`, `kicklead`, `punchlead`, `punchtrail`.
- Defensive: `block10`, `block25`, `block50`, `stepleft`, `stepright`, `lungeback`.
- Neutral: `getup`, `run10`, `run25`, `run50`, `crouch10`, `crouch25`, `crouch50`.

In a first set of experiments we consider the reward function $r_{\text{aggressive}}$ and use linear function approximators for the Q-function. The results for $\beta = 2$ and $\beta = 1$ are shown in Figure 1 and 2. In both cases, the reward rate increases with considerable fluctuations by 0.8 and 1.1, respectively. Starting from the random policy, which loses approximately one reward unit per second, the learning agent achieves a reduction of the loss to -0.2 sec^{-1} for $\beta = 2$ and even a net gain of reward of 0.1 sec^{-1} for $\beta = 1$. In the case $\beta = 2$ the average reward stagnates at a suboptimal level presumably being stuck in a local optimum. From the action frequencies it can be seen that despite the aggressive reward function, the agent prefers defensive actions and lacks the

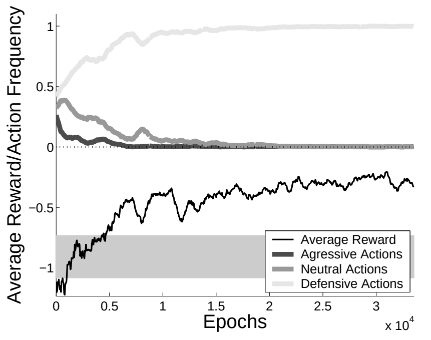

Figure 3: Average rewards and frequency of selected action classes for a Gibbs policy with $\beta = 2$ in the aggressive setting using a *neural network* function approximator with 3 hidden units

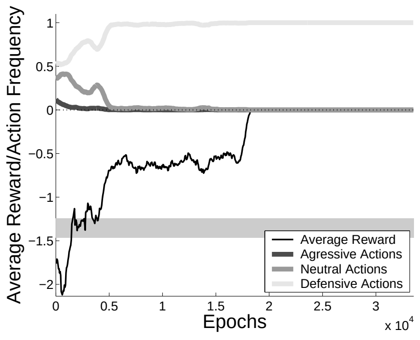

Figure 4: Average rewards and frequency of selected action classes for a Gibbs policy with $\beta = 2$ in the Aikido setting using a *linear* function approximator.

degree of exploration necessary for discovering the aggressive moves required for winning. In the more explorative setting of $\beta = 1$, the agent learns more slowly due to increased exploration but eventually discovers the usefulness of aggressive actions around episode 30 000 which results in a winning policy.

In a second set of experiments we replaced the linear function approximator with a feed-forward neural network with 3 hidden units. The results are shown in Figure 3. The learning agent finds a policy similar in performance to the linear function approximator at the same SoftMax parameter $\beta = 2$. Interestingly, the learning curve is smoother and the action selection appears to be less explorative. However, we did not fully explore the parameter space in order to avoid such local optimal.

In a third set of experiments, the reward function $r_{\text{aggressive}}$ was replaced with r_{aikido} and we returned to linear function approximation. As can be seen from Figure 4, the learning eventually results in an optimal policy (zero reward, i.e., no punishment). Note that we only depicted the *class* of actions selected by the learning agent. For example, in this particular case, a successful behaviour was achieved by an ingenious combination of side stepping and blocking.

In summary, our experimental results show that good policies can be learnt for varying reward functions. Some of the resulting fighting agents displayed interesting behaviour, others exposed weaknesses of game engine and the built-in AI: They found simple repetitive patterns of actions that exploit gaps in the rule-based built-in AI, such as the optimal policy found in Aikido mode (see Figure 4). More exploration of the parameter space and the feature set as well as the incorporation of an eligibility trace in the SARSA update (Sutton and Barto, 1998) may further improve the policies found.

CONCLUSIONS

This work demonstrates that reinforcement learning can be applied successfully to the task of learning behaviour of agents in fighting games with the caveat that the implementation requires considerable insight into the mechanics of the game engine.

As mentioned earlier, our current approach neglects hidden state information and adversarial aspects of the game. One idea to tackle this problem is to separately model the game engine and the opponent. Based on these two models, standard planning approaches (e.g., min-max search, beam search) can be employed that take into account that there is more than one decision maker in the game. Also an important aspect of human fighting game play involves timing which is hardly captured by the MDP model and requires explicit representation of time.

ACKNOWLEDGEMENTS

We would like to thank JCAB, Glen Doren and Shannon Loftis for providing us with the code base and Mark Hatton for his initial involvement in the project.

References

Filar, J. and K. Vrieze (1996). *Competitive Markov decision processes*. Berlin: Springer.

Kaelbing, L. P., M. L. Littman, and A. R. Cassandra (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence 101*, 99–134.

Rabin, S. (2002). *AI Game Programming Wisdom*. Hingham, Massachusetts: Charles River Media, Inc.

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

Salen, K. and E. Zimmerman (2004). *Rules of Play: Game Design Fundamentals*. Cambridge, Massachusetts: MIT Press.

Stone, P. and R. S. Sutton (2001). Scaling reinforcement learning toward RoboCup soccer. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pp. 537–544. Morgan Kaufmann, San Francisco, CA.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning: An Introduction*. MIT Press.

Tesauro, G. J. (1995). Temporal difference learning and td-gammon. *Communications of the ACM 38*(3), 58–68.

Watkins, C. J. C. H. and P. Dayan (1992). Q-learning. *Machine Learning 8*, 279–292.