Perceptron Q-learning Applied to Super Smash Bros Melee

Liam Brown * and Jeremy Crowley†
Stanford University, Stanford, CA, 94305

Reinforcement learning (RL) is able to produce effective policies in environments with small discrete state and action spaces but has significant limitations when the state-action space is continuous. Discretization techniques can solve the issue of continuous state-action spaces, however this often results in intractable state spaces or a poor approximation of the continuum.In this paper, we discuss a reinforcement learning strategy for t Super Smash Bros Melee, a video game with a continuous state and action space by applying global approximation in the Q-learning algorithm. We train the algorithm on progressively harder opponents and show improvement in the agents metrics over time.

I. Introduction

Reinforcement learning (RL) is an area of research focusing on finding an action that will maximize a reward function given the state of an agent. Finding the optimal state action value requires large amounts exploration of the environment. The ability to gather reliable data is crucial to the learning process because this provides the learning algorithm with an accurate representation of how the state evolves with a given action.

RL has recently become a topic of interest due to its ability to solve problems without knowledge of the underlying dynamics.This model free approach is used over a large set of applications including control theory, natural language processing, image recognition, and medical diagnoses. A very popular model-free reinforcement learning algorithm is Q-learning which operates by applying incremental estimation to the Bellman equation[1].

$$Q(s,a) \leftarrow Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) \tag{1}$$

The game we would like to investigate, Super Smash Bros Melee, presents an environment complex dynamics an a continuous state space that would be infeasible to represent as discrete values. A model-free approach eliminates the need for a state transition model and generalization allows the agent to approximate the optimal action at a given state. Given the nature of the problem, we chose to implement a perceptron Q-learning algorithm. In this algorithm, a set of weights for each action θ_a is trained on a basis function $\beta(s)$, such that the state-action value can be globally approximated as $Q(s,a) = \theta_a^T \beta(s)$. The action weights are trained in a batch learning process after each game.

$$\theta_a \leftarrow \theta_a + \alpha(r + \gamma \max_{a'} \theta_{a'}^T \beta(s',a') - \theta_a^T \beta(s,a))\beta(s,a) \tag{2}$$

*Graduate Student, Aeronautics and Astronautics.
†Graduate Student, Aeronautics and Astronautics.

Super Smash Bros Melee is a platform based fighting game in which the goal is to knock your opponent off of the stage. Damage dealt to the opponent increases the distance they fly when hit. A winning strategy in the game involves dealing damage to the opponent and subsequently knocking them off the stage while simultaneously avoiding the opponents attempts to do the same. While the game has many available characters and stages, we limited the project to a single stage (Final Destination) and character (Captain Falcon) for convenience of training, although the work done is general and allows for training of any character and stage combination.

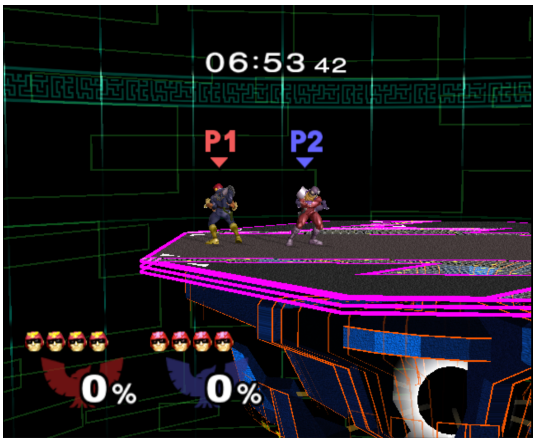

Fig. 1 Super Smash Brothers Melee game environment.

To interact with the game environment, we leverage the open-source game emulator Dolphin and an python library libmelee which provides an interface for reading state values and sending controller inputs to the agent.

II. Related Work

Q-learning is a popular reinforcement learning strategy for discrete state and actions, however the state space and action spaces are often continuous and cannot be represented in tabular form. In [2], Gasket et al implemented a novel interpolator to approximate the Q-function with state and action generalization. In addition to a continuous state space, applying reinforcement learning in an adversarial environment can prove to be difficult because the environment is working against the agent. In [3], a generalized reinforcement learning algorithm is applied to an agent in an adversarial environment. Both [2] and [3] show that applying generalization to reinforcement learning is viable solution to dealing with continuous state-action spaces and adversarial environments. In a recent advancement in generalization for reinforcement learning, Emigh et al applied a nearest neighbor local approximation reinforcement learning algorithm in [4] to Frogger, allowing generalization of the state space based on state proximity. This proves to be an effective strategy in environments with limited data and large similarity between optimality in nearby states.

Another difficulty in reinforcement learning is assigning credit to the action the led to the reward in cases of large delay. Take for example winning the lottery, there is a large delay between when we but the lottery ticket and when we get the reward. If we are drinking coffee right before we find out we win the lottery, the reward of winning the lottery

should still be assigned to buying the ticket and not to drinking coffee. Sutton et al developed a method in [5] to properly assign credit to the action or sequence of actions that lead to a reward. This can become increasingly difficult to address as the dimensionality of the problem increases.

III. Applications to Super Smash Bros Melee

A. Discretized actions

The controller for the Nintendo game cube can be thought of as a continuous action space for Super Smash Brothers Melee. There are two analog control sticks which can be placed at any value from -1 to 1 in both the x and y direction, two analog shoulder buttons which are functionally identical and range from 0 to 1, as well as four digital face buttons.

To reduce the number of potential actions, repetitive combinations of buttons were discarded and the analog inputs were discretized. We represented seven buttons (A,B,X,X_s,L,Z, and $\varnothing$) as binary variables, where a value of one maps to pressed and zero maps to not pressed, and the analog stick as a variable with three possible values for both the x direction (left, middle, and right) and the y direction (down, middle, and up). To create an action, a single button and a value for the main analog sick is selected. This results in an action space $\mathbb{A}$ with seven possible buttons and nine possible analog stick values, for a total of 63 total possible actions.

B. Basis Functions

We designed a set of basis functions to span the state space in Super Smash Brothers Melee and allow for perceptron based global approximation of state action values. Our beta function, β, contains the following elements.

- A set of normal distributions along the x and y axes to approximate the positions
- A set of normal distributions for the relative distance between the agent and the opponent
- A flag for the direction the agent is facing and a flag for the direction the opponent is facing
- A set of flags for the agent and a set of flags for the opponent to represent unique animations
- A set of normal distributions for the agent and opponents damages
- A set of flags for the number of jumps left

Additionally, the state space is discretized into three "super-states", being off the stage to the left, on the stage, and off the stage to the right. A zero padded vector $\beta_p = [\ 0_{|\beta|},\ 0_{|\beta|},\ 0_{|\beta|}]$, where $0_{|\beta|}$ is a zero vector of length $|\beta|$, is then created. The base β function replaces $0_{|\beta|}$ in the appropriate index. The need and justification for this technique is discussed in the novel approaches section.

C. Reward Functions

In order to apply perceptron Q-learning to to SSBM, a reward function was defined. As described in the introduction, the goal of the game is to ultimately knock your opponent off the stage, which becomes easier as their damage increases

since the damage causes them to fly further. This indicates a careful balance between accumulating damage to facilitate a knock out move and taking an action that knocks the opponent back far to knock them out (deferring accumulating more damage in favor of knocking the opponent out). The reward function is then designed to favor dealing damage to the opponent while they are at low damage, with the reward decaying exponentially for damage dealt to opponents at higher damage. Additionally, the agent receives a penalty for taking damage in the same fashion. To prevent the agent from jumping off the stage and prematurely dying, a reward was also given when the agent managed to move from being off of the side of the stage in state s, to on the stage in state s'. Rewards are also assigned for kills and penalties imposed for dying.

Denoting the opponents damage as d_o, the agents damage as d_a, and the on stage parameter of the agent as a true - false flag ON, the reward function becomes:

$$R = (d'_o - d_o)e^{-.01*d_o} - (d'_a - d_a)e^{-.01*d_a} + r_{jump}\delta_{0,ON}\delta_{1,ON'} + r_{kill} + r_{death} \tag{3}$$

The reward assignment was not straight forward due to the lag in state-evolution and delay between actions and resulting kills and deaths. The methods in which we handle these issues is described in the next section.

IV. Unique Approaches

To apply perceptron Q-learning to Super Smash Brothers Melee, we apply unique techniques to deal with discontinuities in optimal behavior, state-evolution, and large delays between an action and its result.

A. Discontinuity in Optimal Behavior

In Super Smash Brothers Melee, the optimal action largely depends on the current state. An issue arises in global-approximation techniques when states that produce similar basis function outputs should have dissimilar optimal actions. An example of this is the case of an agent standing on the edge of the stage, in which its goal is to deal damage to the opponent, and an agent being slightly off of the stage (about to fall to its death unless some preventative action is taken), in which it should make an effort to navigate back to the stage and avoid dying.

To prevent generalization from nearby but dissimilar states, we discretize the environment into an additional three "super-states": off the stage to the left, on the stage, and off the stage to the right. These super-states represent different situations in which the optimal behavior of the agent should be unique from a nearby position that is in a different super-state. As discussed in the applications section, the padded beta vector results in differently trained perceptron weights in different super-states for each map partition. This improves the agents ability to survive after being knocked off of the stage because the agent learns to jump back towards the stage in an effort to survive (and avoid penalties for dying).

B. State Evolution

Another issue we address was the impact of ”action-lockout”, in which an action taken may have no impact on the agents transition to the next state. An example of this is when the agent selects an action with a long wind-up animation that cannot be canceled by inputting other actions. To address this, actions that are taken during an "action-lockout" are ignored during the training process of the basis function weights.

While performing batch learning, this is taken into account. In the weight update process, $\theta_a \beta(s)$ of the last action take is used for each update, while the state s' is taken as usual. The rewards are computed at each state step and assigned back to the last action taken. The result is that the original state action pair is rewarded for the state evolution that occurs, rather than rewarding the action at intermediate states of the evolution.

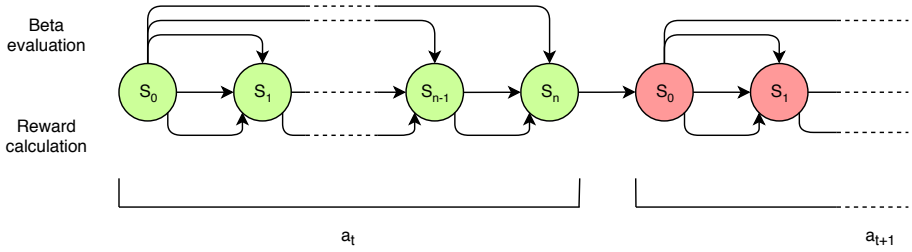

Fig. 2 State evolution diagram

Figure 2 is a representation a state evolution. All the green states indicate a single state evolution with action-lockout occurring at each new state S_t. The transition to the red states are when an action is taken that affect the transition to the next state, starting a new state evolution. We can see the how the reward is calculated from S_t to S_{t+1} while the beta function is evaluated from S_0 to S_{t+1}.

C. Action Impact Delay

Dealing with rewards for kills and deaths is not straight forward due to a large time delay between an action that results in one of these events and the event occurring. To assign rewards to killing the opponent, the last action the agent take that deals damage is recorded as a "last damaging action". When the opponent dies, a large reward is assigned to this "last damaging action". This is necessary to avoid assigning large rewards to actions that do not cause the opponent to die, and is equivalent to knowing immediately after the action is taken if it will result in the death of the opponent.

V. Results

The agent was initialized with no prior against a level 1 CPU and was allowed to train overnight, training the weights between each match. For training, we used a learning rate of $\alpha = 0.01$, a discount factor of $\gamma = 0.95$, and a non-fixed soft-max exploration parameter λ. The agent was trained against progressively harder opponents while using priors from the previous training, until we reached a level 4 CPU.

In Figure 3, we see that the uninformed agent with a high exploration bias quickly achieves an approximately 80 percent win-rate against the base level 1 opponent. When this agent is then given a lower exploration bias, that win-rate quickly converges to 100%. The first real challenge faced is against the level 3 opponent, were the agent undergoes a difficult period, having its win-rate fall to near 50% before recovering back to 100% over a period of approximately 100 games. A similar trend is observed when the agent faces off with a level 4 opponent. We can also see similar trends in Figures 4, and 5 where the stock differential and the damage differential slowly improve as the agent trains.

In addition to the quantitative performance metrics observed in the Figures, the agents behavior also improved qualitatively. Against the level 1 opponent, the agent learned basic behaviors that lead to a win. The agent would simply repeat the same attacking move that the level 1 opponent was unable to deal with. As the agent faced progressively more difficult opponents (level 3 and 4), this behavior would sometimes work but would often result in the agent taking damage. The agent was able to determine states in which this learned "spamming" behavior was optimal and which states it should avoid taking this action in, developing new strategies depending on the state.

An interesting trend in the data is the increase in games required for the agent to learn an effective policy against its opponents. Against the level 1 CPU, we see that the winning strategy can be learned in approximately 85 games. Against the level 3, this process takes 85 games, and against the level 4 CPU we were only able to achieve an 85% winrate after 245 games.

VI. Conclusion

Applying perceptron Q-learning for Super Smash Brothers Melee was successful. Given enough time, the agent learned to progressively beat higher level default CPUs, die less, and take less damage. One severely limiting factor of the project was the need to run the agent in real time against a CPU in order to train. This caused iteration on the basis functions, reward functions, hyper parameters as well as progression through CPU difficulty to be slow. On the other hand, the combination of our achieved performance and low iterations on parameters and functions shows the robustness of perceptron Q-learning in this environment.

We believe that with improvement to the reward functions and careful detail to to hyper parameters, as well as more time, the agent could learn to beat significantly higher level CPUs than demonstrated here. Additionally, while the agent has only ever played as captain falcon against captain falcon, the agent should be able to learn to play any matchup and on any stage. This could be done by storing different basis function weights for each situation.

Our group plans to continue the project by both continuing to play against higher level agents with the existing perceptron Q-learning approach, as well as extending the bots capabilities by implementing DQN. We believe DQN will provide a higher level of performance as it is able to represent non-linear basis functions, while perceptron Q-learning is limited to linear relationships. We are also looking to explore different characters and stages.

Appendix

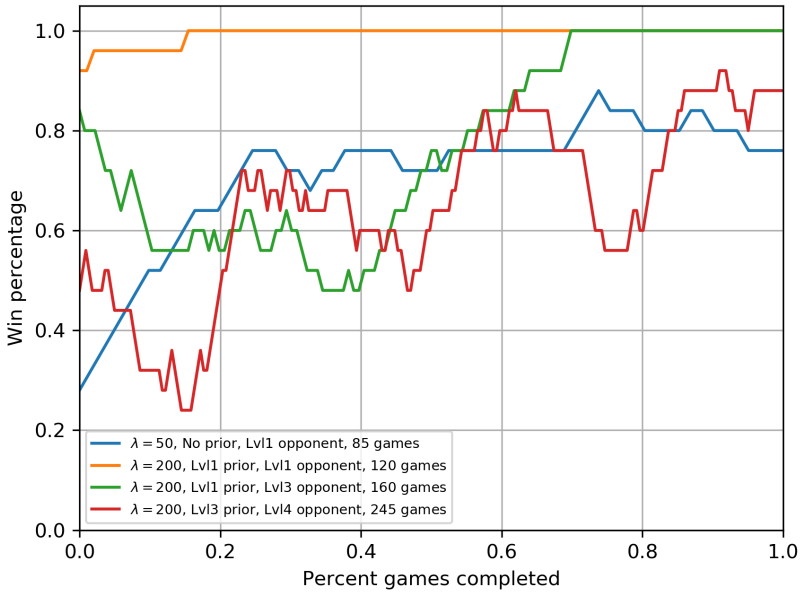

Fig. 3 25 game moving average of winning percentage.

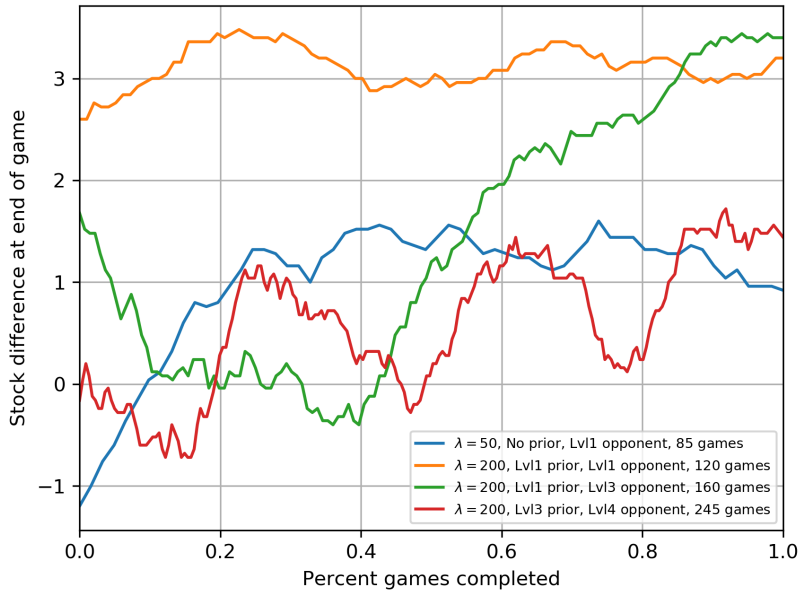

Fig. 4 25 game moving average of stock differential at end of game (higher is better).

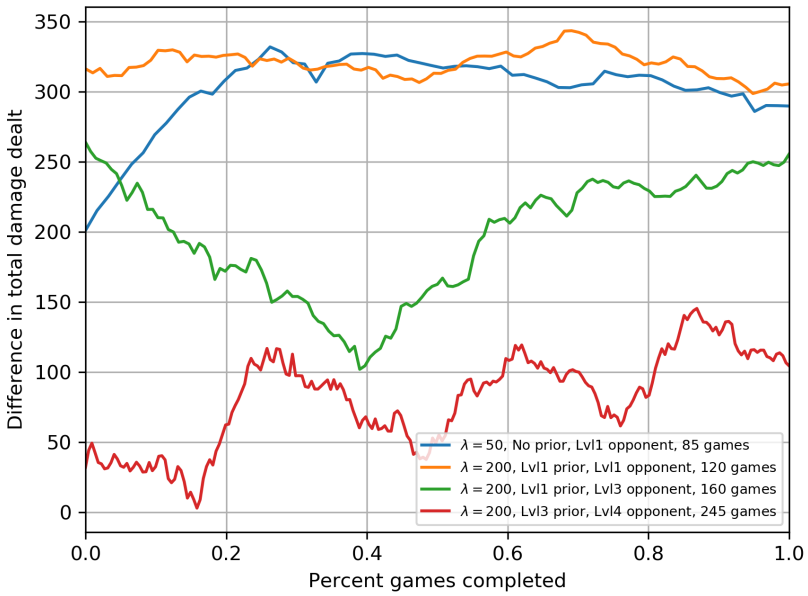

Fig. 5 25 game moving average of difference in total damage dealt in each game (higher is better).

Acknowledgments

We would like to acknowledge libmelee for providing an open source solution to obtaining information about the game while it is being played. We would also like to thank Professor Mykel Kochenderfer and the entire AA228 course staff for providing a fantastic learning experience.

References

[1] Kochenderfer, M. J., *Decision Making Under Uncertainty: Theory and Application*, 2nd ed., MIT Press, The address, 2015.

[2] Gasket, C., Wettergreen, D., and Zelinsky, A., "Q-Learning in Continuous State and Action Spaces," *Advanced Topics in Artificial Intelligence*, , No. 4, 1999, pp. 417–428.

[3] Uther, W. T., and Veloso, M. M., "Generalizing adversarial reinforcement learning," *AAAI Fall Symposium on Model Directed Autonomous Systems*, , No. 3, 1997.

[4] Emigh, M. S., Kriminger, E. G., Brockmeier, A. J., Principe, J. C., and Pardalos, P. M., "Reinforcement Learning in Video Games Using Nearest Neighbor Interpolation and Metric Learning," *IEEE Transactions on Computational Intelligence and AI in Games*, Vol. 8, No. 1, 2016, p. 56–66. doi:10.1109/tciaig.2014.2369345.

[5] Sutton, R. S., "Temporal Credit Assignment in Reinforcement Learning," , 1984.