Stanford University

Learning to Play Atari Games

David Hershey, Blake Wulfe

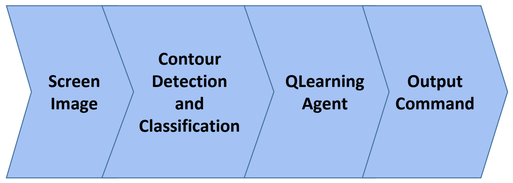
A Project for CS229 at Stanford University

Problem Statement

Our goal for this project was to enable a computer to learn to play a set of Atari 2600 video games based solely on the visual output from those games. Video games in general have very large state-spaces, which require either intense computational time to explore completely, or algorithms that reduce this state-space. Even after reducing the state space, it is important to use efficient learning techniques with the best possible function estimation algorithms. Previous papers have used methods such as convolutional neural networks and deep learning in order to solve these problems. We've elected to experiment with more simple approaches such as linear function approximation and feed-forward neural networks, supported by feature extraction techniques.

Our Solution

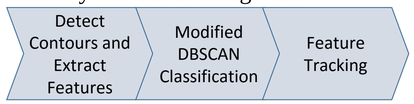
- Contour Detection, Classification, and Tracking
  - Using computer vision techniques, identify screen entities
  - Classify entities with clustering algorithm
  - Track entities between time frames
  - Output entities to Q-Learning algorithm
- Q-Learning
  - Linear Function Approximation
    - Most simple method of estimating value of a state
  - Feed-Forward Neural Network
    - Alternative option that can learn non-linear approximation functions



Methods

Feature Detection and Classification

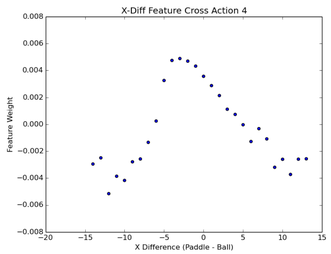
In order to reduce the dimensionality of the state-space, we use a feature detector that detects contours in the game screen, and then classifies them into an arbitrary number of categories.

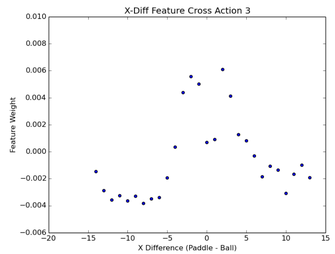


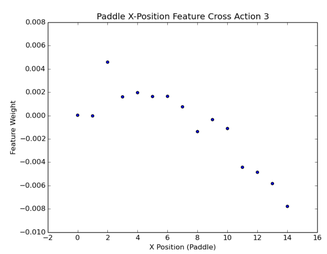
First image contours are extracted using computer vision methods from openCV. Defining features of these contours are then extracted and prepared for clustering. These features, along with features from recent frames, are clustered by the DBSCAN clustering algorithm. This algorithm automatically estimates the number of clusters, allowing the number of unique entities observed by the algorithm to grow over time. After classifying the features, tracking logic is used to calculate derivatives of features frame to frame. The classified features are then passed through to the Q-Learning agent as its input.

Q-Learning with linear function approximation

Q-learning is a model-free, off-policy, reinforcement learning algorithm that estimates the optimal values of states, represented by the extracted object features. Linear function approximation uses a linear function to map from features to values, which allows the algorithm to avoid exploring an intractable number of states. In order to account for nonlinear relationships between object features and their value, we used pairwise features between positions, their derivatives, and the action.







Q-Learning with Neural Network for Function Approximation

Hand-crafting features is often difficult and using feature templates can result in large, sparse feature vectors. A neural network function approximator circumvents these issues by automatically learning relationships between features; however, this model can experience difficulty learning in a reinforcement learning setting. We used Theano to implement a Q-learning agent that employs neural network function approximation and encountered these problems. In order to address these issues we applied more advanced learning methods, for example a semi-static optimization target and a replay memory, and the tuning of hyperparameters..

Replay Memory

The consecutive screens encountered by the learning agent are highly correlated, which can make learning difficult. A Replay memory stores past experiences and randomly samples from them during training to in order to reduce these correlations and improve learning. Biasing the sampling procedure to select for particularly informative or highly rewarded experiences can promote the agents exploration of high value states.

Stationary Target Function

Q-learning is a bootstrapping method, which means it uses its current estimate of the value function in determining its target value. This helps speed up learning, but can cause a function approximator to diverge. To avoid this, a semi-static target function is used. This allows for training the network with a higher learning rate which translates to faster learning.

Results

With linear function approximation, we were able to achieve above-random performance on the game Breakout. This was accomplished with a training time of only three hours. When using the neural network for function approximation, we have achieved only slightly above-random performance on Breakout. Neural networks require more training time and more carefully thought out input features. We believe future work with more time training the neural network with different feature permutations could result in the neural network achieving well above random performance and potentially even approach human-level performance.