Machine Learning Engineer Nanodegree
Capstone Proposal
Mike Allen
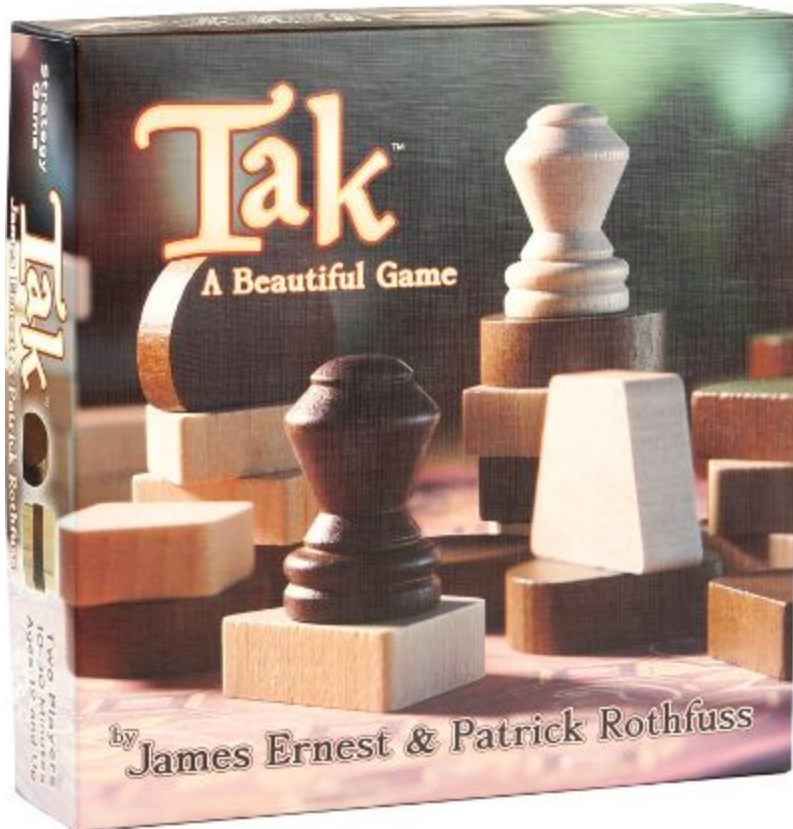December 31st, 2050

Domain Background

Tak is a new abstract strategy board game created by James Ernest and Patrick Rothfuss, based on the game in the book "The Wise Man's Fear" by Patrick Rothfuss. Through KickStarter, the game became funded and distributed in December 2016. Given the recent age of the game, currently no app exists that offers human vs computer play.

[Rules for TAK](#)
Tak board game rules used to create the simulated environment.

[How to Play Video](#)

Competitive human level play is achieved in similar games such as backgammon and go leveraging reinforcement learning and neural networks. Both Backgammon and Go have an enormous number of possible states which make table look-up and brute force strategies infeasible.

As described by Gerald Tesauro:

> Programming a computer to play high-level backgammon has been found to be a rather difficult undertaking. In certain simplified endgame situations, it is possible to design a program that plays perfectly via table look-up. However, such an approach is not feasible for the full game, due to the enormous number of possible states (estimated at over 10 to the power of 20). Furthermore, the brute-force methodology of deep searches, which has worked so well in games such as chess, checkers and Othello, is not feasible due to the high branching ratio resulting from the probabilistic dice rolls. ([Temporal Difference Learning and TD-Gammon](#))

Google researchers describe Go in similar contexts:

> The search space in Go is vast -- more than a googol times larger than chess (a number greater than there are atoms in the universe!). As a result, traditional "brute force" AI methods -- which construct a search tree over all possible sequences of moves -- don't have a chance in Go. ([AlphaGo: Mastering the ancient game of Go with Machine Learning](#))

Given the success of TD-Gammon and AlphaGo leveraging reinforcement learning in conjunction with deep neural networks, I believe the approach can be applied to Tak which has a similarly large search space.

Learning Resources

[Udacity - Reinforcement Learning (ud600)](#) with Michael Littman, Charles Isbell
Topics include game theory including Zero-Sum games and the MiniMax algorithm. In addition there is some introductory discussion on generalization of the value function using neural networks in place of the Q-Table.

[NPTEL Online Course - DQN and Fitted Q-Iteration](#) with Balaraman Ravindran
Video lecture discussing Fitted Q-Iteration and Experience Replay.

[Reinforcement Learning: An Introduction](#) by Richard S. Sutton and Andrew G. Barto

Problem Statement

The problem at hand is to program an agent that is able to play the boardgame Tak competitively as measured through average game score and win ratio.
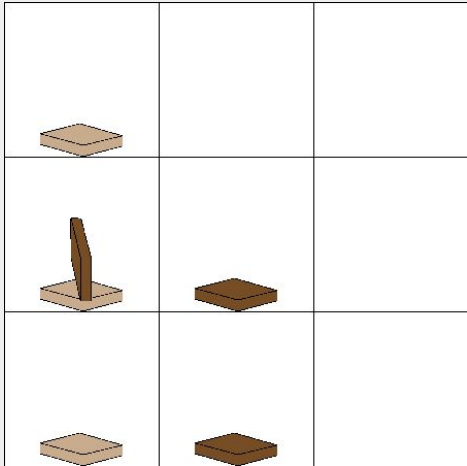
Rather than approaching as a supervised learning problem, relying on human expertise to craft features, a possible solution is to leverage reinforcement learning, allowing the agent to learn the game over time.

A challenge in leveraging reinforcement learning will be creating a model that is able to represent an enormous number of possible states, as well as account for an opponent that is actively working against the agent.

Datasets and Inputs

Data is generated through an agent interacting with a simulated environment. The simulated environment is written in python and leverages OpenAI Gym as a framework to create the agent-environment loop.

Reinforcement learning agents can then leverage these recorded experiences to learn the expected long term reward of any given state. Using these values, the learning agent chooses the best action to take given a state.

State	State of the simulated environment before taking an action. State is structured as a 3d-array of the shape: **board size** x **board size** x **board height**. This structure is conveniently suited for convolutional neural network layers.
	Board sizes range from 3x3 to 8x8. For a 3x3 board, each player starts with 10 pieces. Because pieces can stack on top of each other, the maximum height of the 3x3 board is 20. Below is an example 3x3 board's state representation.
	Note: CNN requires a fixed input shape, each state has the same size regardless of whether or not those spaces are filled. This 0 padding could be applied by the CNN pipeline to save on storage space and memory.
	Also note that white pieces are indicated by positive numbers and black

	pieces are indicated by negative numbers . In terms of the value function, this makes it easy to view the state from the other player's perspective, simply by multiplying the state by -1.
Action	Action taken by the agent at given state.
Reward	Reward agent received by taking action. Reward is crafted to match game score as described in evaluation metrics.
State Prime	State of the simulated environment after taking action.
Player	Active player.
Player Prime	Active player after taking action.

Solution Statement

Feedback received while exploring a simulated Tak environment is used to train a reinforcement learning agent to play Tak.

Typically, on-line updates are made to the Q-Table during exploration using the Bellman equation. However, to account for an enormous number of possible states, a deep convolutional neural network is substituted as the value function. Using Fitted Q-Iteration, and Experience Replay techniques the learning agent is trained from experiences recorded through agent-environment interaction.

In addition, to account for an opponent working against the agent, the MiniMax algorithm which seeks to optimize the agent's reward while minimizing the opponent's reward.

Once trained through reinforcement learning, an agent is able to play Tak competitively as measured through average game score and win ratio.

Benchmark Model

The learner agent is benchmarked against an agent taking random actions. Additionally a benchmark against human level play is achieved by providing a pyGame interface to allow user input. Human players are recruited from friends, and are novice players.

Evaluation Metrics

Win Ratio: Ratio of wins to losses.

Game Score: A winner is awarded "the board" (one point for every space on the board, for example 25 for a 5x5 board), and "the pieces," which is the number of the winning player's own pieces that remained out of play.

Project Design

Simulated Environment

The first step in the project is to create a simulated environment for Tak, written in python, leveraging OpenAI Gym. Once created, agents can explore the environment, receiving feedback by taking actions within the simulated environment.
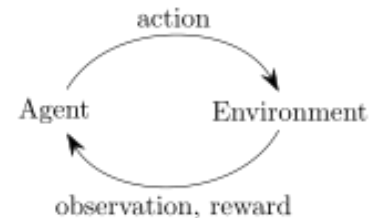
Additionally a visual interface is provided for human interaction.

Agent-Environment Loop and Value Functions

For reinforcement learning, In simplistic cases, the value of a state could be memorized within a table of all possible states. Due to the enormous number of possible states for Tak, we cannot search every possible state to determine the value. In this case the value function is replaced with a regressor such as a deep neural network.

One challenge, when replacing the value function with a function approximator, is that the value updates are now global. Because of this, on-Line updates to reduce error on a single experience leads to globally unstable results.

To account for this, updates to the model are performed off-line. During the agent-environment loop, the experiences are simply recorded for later.

Fitted Q-Iteration and Experience Replay

Fitted Q-Iteration leverages off-line updates to the model allowing more stable updates. Now instead of minimizing error on a single example, the model is trained through a batch of experiences. In addition, more efficient use of data is achieved through the use of Experience Replay.

Agents

Two different agents are created. The first agent is a learning agent using a deep convolutional neural network for the value function. Convolutional layers are used to take advantage of spatial information that might be lost with fully connected layers.

Additional refinement of the network structure is done upon experimentation.

The second agent is a random agent, which always takes random actions. This agent is used to benchmark the learning agent.

Zero Sum Game & MiniMax Algorithm

In adversarial environments, the actions of the opponent must be considered in maximizing long term reward. The minimax algorithm seeks to maximize reward while minimizing the opponents reward. In terms of the updates to the model through the Bellman equation, positive future rewards become negative if the next player is the opponent.

Multiple Board Sizes

Board sizes can range from 3x3 to 8x8. Along with the differences in board sizes, the number of pieces changes. Because of this, the strategy might differ from board size to board size. A separate convolutional network model and weights are to be created for each board size to account for this.

Refinement

Refinement to the learning agent model will come with experimentation and evaluation of the performance metrics.

Possible changes include introducing inception modules, residual connections, and LSTM, as well as adjustments to the filter parameters and dimensionality of convolutional layers

Additional refinement may come in the form of adjustments to how sampling is performed for Fitted Q-Iteration and Experience Replay. For example, should we sample at random or use all recorded data? How many iterations should be used during Experience Replay?

Generators could be leveraged to perform simple data augmentation such as rotating the board. This would allow the deep convolutional network to learn value independent of board orientation.

Other refinements include adjusting regularization and activation functions to reduce error and overfitting.