Applied Machine Learning Final Project Proposal

Josh Dawson, Cal Colbert-Pollack, Mario Stinson-Maas

Q1

There is substantial contemporary literature on learning algorithms and their attempts to 'play' various games. When examining how video games compare to the real world, this is not that surprising. Ideally, machine learning models and the algorithms that constrain them will be able to model and learn from the real world. While this is possible to a certain extent, these models are expensive to build and train. Video games, however, are much more accessible for training and testing. Beyond this, their worlds have a finite structured amount of rules, and actions always have the same outcome (ignoring RNG). This makes them a great place to experiment with new models and algorithms, and also allows for a cheap way of accessing domain expertise.

Some of these models, such as, AlphaGo, are incredibly specialized, with an architecture that includes substantial domain knowledge, while others—like Mnih et al.'s deep q-network—are explicitly generalized and perform well across many (Atari 2600) games. We are curious about the relative performance of several different models when varying available inputs, outputs, utility functions and hyperparameters. We hope to contribute to the chain of training agents to play increasingly complex games using Doom (1993), and within this, learn new things about how the parameters involved in the algorithm inform the resulting performance.

Q2

Evolving Neural Networks through Augmenting Topologies considers the state of the Neuroevolution literature, and proposes some solutions to existing problems. The authors propose that the co-evolution of weights and topologies can drastically improve performance. By minimizing the structure, they drastically reduce the size and dimensionality of the search space—hopefully speeding learning. However, this solution comes with difficulties: genotypes must be encoded such that meaningful breeding between distinct topologies is possible (and ideally efficient), novel topologies must be protected from more developed ones trapped at local optima, and network complexity must be minimized (as otherwise we defeat the whole point of topological evolution). The authors review many potential solutions to these complex constraints—comparing their tradeoffs—and propose their own, providing our own work with a strong grounding in the types of algorithms we may investigate.

Human-level control through deep reinforcement learning is a more recent advance in general game-playing. They provide a strong psychological, biological and neurological

grounding for their advances, proposing that their techniques are consistent with heuristics *real* evolution seems to have followed. In particular, the authors work to evolve lower dimensional encodings of sensory inputs by following the work of Kunihiko Fukushima, who was himself inspired by Hubel and Wiesel's seminal paper on the encoding of visual information in the striate cortex of cats. Not merely restricted to sensory analysis, they consider a biologically inspired method they term “experience replay,” hopefully allowing agents to learn from thinking about—and remembering—past performance.

Genetic Algorithms are a Competitive Algorithm for DNN and RL introduces genetic algorithms as a gradient-free method for solving deep reinforcement learning problems. The paper details the performance and implementation of other deep learning methods, such as: Q-learning methods (DQN), policy gradient methods (A3C), TRPO, and evolutionary strategies (ES). This article also details novelty search algorithms, by which the agent is rewarded when exploring an unseen path; this helps the agent avoid traps such as local extrema. In the article, the researchers show their results of these algorithms applied to multiple Atari games; the proposed genetic algorithms outperform all other models in certain games, with some of the quickest training times across the board. Other evolutionary strategies prove to be useful in certain tasks, along with A3C, although A3C takes much longer to train. We hope to utilize these deep genetic algorithms to competitively train deep neural networks for reinforcement learning tasks.

In this video, the creator restricted the action-space of the agent. It was this innovation that directly inspired this project. The creator is experimenting with different deep learning techniques such as Q-learning and A3C to train a deep neural network to play Doom. We also plan to experiment with deep learning techniques to solve this task, although we hope to incorporate different techniques such as evolutionary strategies and genetic algorithms to compare performance.

Q3

To train any models, we first need to collect the data they will be trained on. In this case, the data is the state of the game, represented by the pixels on the screen. We will use the BizHawk emulator to run Doom (1993), released for the Sony PlayStation. By having our program run this application and in a loop, we can read the live pixel data, process it, and send game inputs back to the emulator. The pixel data is the first part of data collection that we need to do; this will be fed into a neural network that should yield an output of one game input. We hope to utilize genetic algorithms to build and optimize such a network, potentially using neuroevolution of augmenting topologies (NEAT) to alter not only the weights, but the structure of this neural network. Any given iteration must then have its performance measured through a reward system structured in a utility function. This performance summary will then be used as data to train further generations of models.

Q4

We are interested in learning how evolutionary deep neural networks can be used to solve high-input-dimension optimization tasks, specifically the id Software video game Doom (1993). To do this, we will explore genetic algorithms and reinforcement learning systems relevant to this problem. Beyond this, we are interested in implementing several competing models with varying degrees of “bespokeness” in architecture and with variations in hyperparameters (that is, how tuned a model is to this particular game, by leveraging our intuitions). After they have run, we can compare their performances. Are the assumptions we make about successful models in this domain valid, or can a sufficiently advanced model outperform those constrained by our intuition?

To answer this, we will examine how varying utility functions, with different rewards for speed, score, kills and damage taken will change the performance of the model. We will attempt to compress the input-space in a way that speeds learning without decreasing performance. For example, blurring the input image, removing irrelevant pixels, or (perhaps if possible) directly inputting map information.

We may also limit performance by limiting the output space, this could be done by restricting certain button presses, or not allowing the agent to move backwards. To achieve this, we will choose from one of the algorithms outlined in the literature review, likely NEAT or DQN. We plan to track and visualize the skill of our artificial gamers over time. This data may suggest a particular hypothesis about the nature of learning in this context, though of course data from a single game and a single algorithm is unlikely to generalize.

Q5

In order to complete this project, we will need the following components:

- Program to run the emulator, capable of reading in live pixel data and sending game inputs to the client
    - Must also be able to execute application and control the client
- A parsing function that preprocesses this pixel data as it is read
    - Determine what data is useful, and what isn’t
    - Varying this as a hyperparameter leads to different performance among models
- A utility function capable of assigning rewards to given game events
    - This function will reward moving through the level, not taking damage, killing enemies, etc.
    - We may use novelty search to reward agents that explore the map, leading to fewer agents being trapped by local minima
- Initial neural network to serve as a starting point for our final model
    - This will initially be a guesser, sending pseudo random inputs when the game begins
- Ways to train this network using different methods, allowing us to compare results across models

    - Using NEAT would allow us to alter the structure of our deep neural network, which could lead to higher (and faster– meaning requiring less epochs) performance
    - Also want to try Q-learning
- Analysis of the outcomes of these performances
    - May require a range of statistical methods depending on what specific parameters are varied

Q6

From previous experience with video games, we have gained intuition for how to manipulate our utility function, since we know how the parameters are likely to influence gameplay. More importantly, we have learned how to consider the benefits and drawbacks of algorithms in different contexts throughout this semester. Here, instead of directly considering different architectures, we apply domain expertise to clean or preprocess inputs and outputs.

Q7

To ensure the timely completion of this project, we outline a speculative schedule below (to be adjusted as needed):

Completion Date (Week of)	Task To Complete
4/14: Testing/Planning	Ensure Emulator works and that data can be acquired. Identify a model to pick variants for, and brainstorm good ranges for the parameters of these variants.
4/21: Data Acquisition and initial training	Essentially, to acquire data: create a sandbox allowing our agents to interact with the game world. Here, it would also be helpful to choose a model and several different variants for it. These variants would be chosen by modifying different hyperparameters, levels of preprocessing, limits on outputs, and utility functions. Begin training.
4/28: Wrap up model/variant choice and adjustments; Continuation of training	Finish anything not completed in the previous section and continue training. Make any adjustments necessary to the variants and model so that Analysis can be conducted in the next week.
5/5 Analysis	Make a final analysis. Which models did better? Which did worse? Why? In what scenarios? Are there any larger trends we can identify and are they explainable? Start Report.
5/12 Completion	Complete Report and prepare for presentation.