Machine Learning & Game A.I.

Jamie Cho

With the recent victory of AlphaGo, A.I. has demonstrated stellar performance in tasks designed to be challenging for even humans. This project, in reproducing Deepmind's architecture from scratch, implements a wide breadth of relevant machine-learning concepts without relying on external frameworks.

I. So, What did You Do?

Fig1. 2048 is a single-player sliding block puzzle game by 19-year-old Italian web developer Gabriele Cirulli. [Wikipedia]

Have you ever played 2048?

If you haven't, download the app now and play it while listening! It's almost as fun as this presentation. Not quite, but close.

Anyways, I implemented 2048 A.I. with C++, without external framework other than Armadillo, which is a CPU-based library for Linear Algebra.

II. Why Should I Care?

Fig 2. JARVIS, a robotic butler in the Iron-Man Franchise.

Because this is the first step towards your robotic butler... maybe.

Games provide a simulated setting for the development of Artificial Intelligence, in which the relevant information for training are readily available in a repeatable and scalable manner; moreover, they often present tasks that are difficult for even humans to accomplish.
All of these traits makes games a very appropriate experimental ground for machine-learning, prior to more complex applications.

III. How Does It Work?

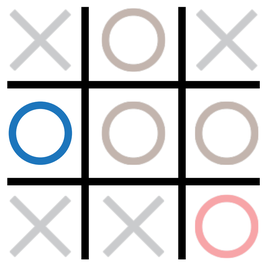

Fig 3. Q-Learning Algorithm finds the *value* of an action *a* in state *s*. Here, for instance, *blue* corresponds to a desirable action -- *red*, not so much.

$$Q_t(s,a) = (1-\alpha)Q_{t-1}(s,a) + \alpha(r_t(s,a) + \gamma max\,(Q(s',a')))$$

Q-Learning is a class of temporal difference learning that perform an iterative update based on interaction with the environment. The Q function correlates state-action pairs with its *value*, or utility.

By linearly interpolating between the previous value and the anticipated future reward, the true reward gets propagated throughout the system.

IV. How Can I Do It?

Deep Q-Learning

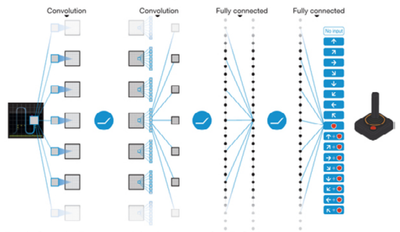

Fig 4. Google Deepmind Architecture for DQN.

Deep Q-Learning is a recent advancement that provides a *general framework* upon to develop a *universally adaptable agent.*

In a traditional Q-Learning setting, the value function was stored via a table that mapped transitions to its utility. Naturally, the algorithm suffered dearly from the curse of dimensionality.
What's more, the agent cannot infer from previous examples: it is completely vulnerable in unforeseen situations, regardless of the resemblance with earlier experiences.
Neural Network, in this way, serve as an excellent approximator of the value function: it is able to identify trends in the correlation between the input and the output.

Neural Network

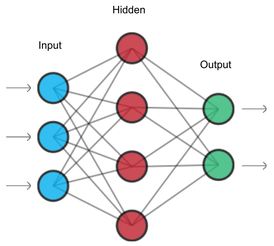

Fig 5. A simple illustration of Neural Network Architecture.

Neural Networks, in machine-learning, are biologically inspired computational counterparts to synaptic connections.
In a FeedForward Neural Network, each neuron interacts with a set of input and output neurons, as follows:

$$\vec{o_0} = \vec{x}$$

$$\vec{o_l} = f_l(W_l\vec{o}_{l-1} + \vec{b_l})$$

Neural Networks learn through a process called **Back Propagation**. This is essentially identifying how much of the previous input was responsible for the error, and propagating this error backwards. The canonical equation is illustrated below.

$$\vec{\delta_L} = \vec{t} - \vec{o_L}$$

$$\vec{\delta_l} = (W_{l+1}^{\intercal}\vec{\delta_{l+1}}) \cdot f_l'(\vec{i_l})$$

$$\Delta W_l = \vec{\delta_l}\vec{o_{l-1}^{\intercal}}$$

$$\Delta b_l = \vec{\delta_l}$$

Convolutional Neural Network

Convolutional Neural Network, ConvNet in short, is a specialized form of a neural network devised by Yann Lecun to specifically process highly structured image-type data. In fact, it is inspired by the visual cortex.

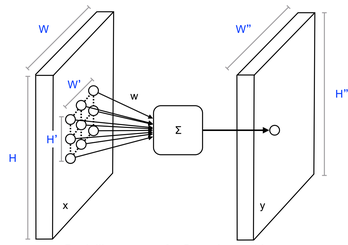

Fig 6. Illustration of a Convolutional Layer.

The most powerful trait of a ConvNet is ***feature extraction***; ConvNet learns the filters with which to extract higher-level features from the data. This enables a full-fledged automata without preprocessed input.

In 2048, there is a strong two-dimensional correlation in space, so ConvNet is a natural choice; however, due to heavy computational costs, I couldn't run the simulation enough to find the appropriate hyperparameters.

Quest To Convergence

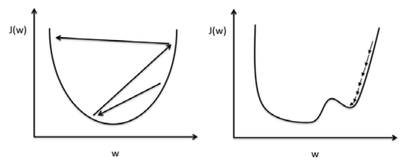

Fig 7. Issues of uncalibrated learning rates, in either direction.

Adaptive Learning

Neural networks as value-function approximators are known to be very unstable: prone to diverge; the dilemma with the learning rates is *neither small nor large are good* for fast and stable convergence.
Adaptive Learning optimizations provide a means to achieve this by adjusting the learning rates based on the response of the system to recent updates.

Experience Replay

Experience Replay roots itself in reminiscence -- namely, using the experience multiple times in the training process. Prior to its development, each experience in Q-Learning was used exactly once and discarded.

By randomly selecting a sample among a pool of memorized experiences, the Q-function learns to generalize the system due to the removal of the temporal correlation between neighboring updates, thus leading to better convergence.

V. How Well Does It Perform?

While the power of my A.I. was limited by time, storage, and implementation constraints, it produced some noteworthy results.

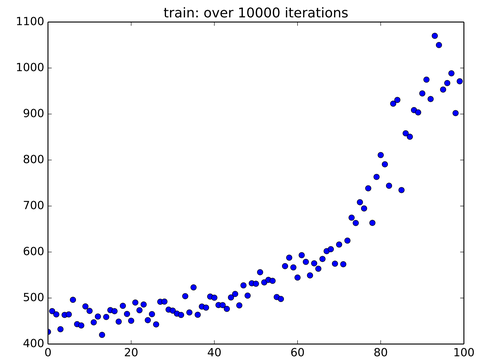

Fig 8. training score, averaged over 100 games, with the scoring scheme of the original game.
A clear trend of improvement is visible, though possibly due to epsilon-annealing; a completely random agent, in comparison, achieves an average score of approximately 400.

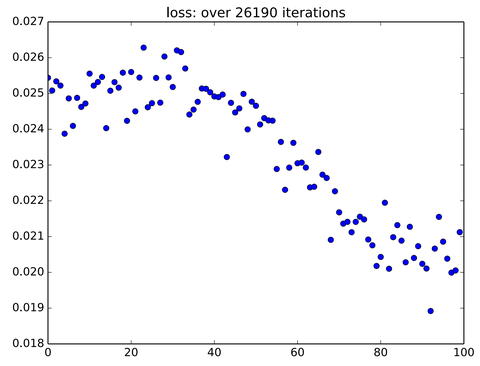

Fig 9. Q-Learning loss, averaged over approximately 262 queries. Despite the fact that the loss was recalibrated with alpha-annealing, a clear trend of decreasing loss is visible, plateauing out in the end.

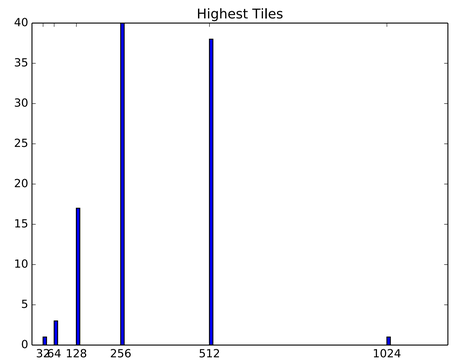

Fig 10. Highest Tiles on the board out of 100 testing games. One of the games were actually able to reach 1024, which is halfway towards beating the game. Generally, the game ended with 512 or 256 as the end-game values.

VI. What Next?

I believe that the A.I. definitely has potential to consistently beat the game.

To this end, I plan on implementing backpropagation with **minibatch gradients**. I've recently realized that minibatch and ER work very well together, both in concept and implementation.

Furthermore, I'm willing to try more adaptive learning rate algorithms. So far, I've only looked into AdaDelta and RMSProp. On that note, **Nesterov momentum** allegedly converges faster and in a more reliable way, which I haven't tried yet.

I have already implemented **ConvNet** with OpenCV, but I wasn't quite able to apply that to this project as it was slow, unreliable and inconvenient. I'm currently implementing a fully GPU-based ConvNet, in CUDA, with its own Linear Algebra Kernels to speed up the learning process.