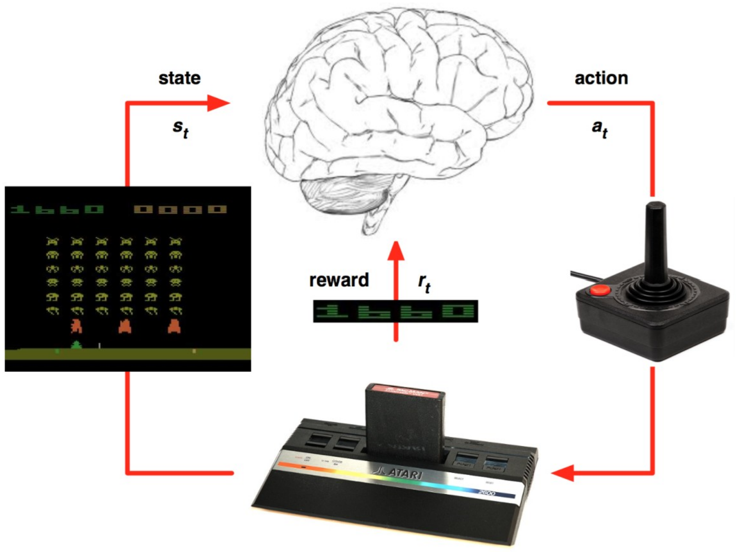
Deep Reinforcement Learning Hearthstone

Alexander Christner

Inspirations



AlphaGo

Overview

Use reinforcement learning to train an AI to play Hearthstone.

Utilize several neural networks, transfer learning, and a fan-made emulator.

Attempt to overcome the extra complications of this game with smart design.

Benchmark against agents that other people made using that emulator.

Q-Learning: Background

The goal is to learn a policy, which specifies what action to take given state in order to maximize expected reward.

Reward of taking action *a* from state *s*:

$$Q(s,a)=\sum_{s'} T(s,a,s')\left[R(s,a,s')+\lambda\, max_{a'} Q(s',a')\right]$$

Choosing which action to take:

$$\pi(s)=argmax_a Q(s,a)$$

Q-Learning: Training

Simulate games using the current policy to generate datapoints consisting of a state, action, reward, and successor state (s,a,r,s').

Update Q given a (s,a,r,s') datapoint:

$$Q(s,a) \Leftarrow (1-\alpha) Q(s,a) + \alpha(r + \lambda max_{a'} Q(s',a'))$$

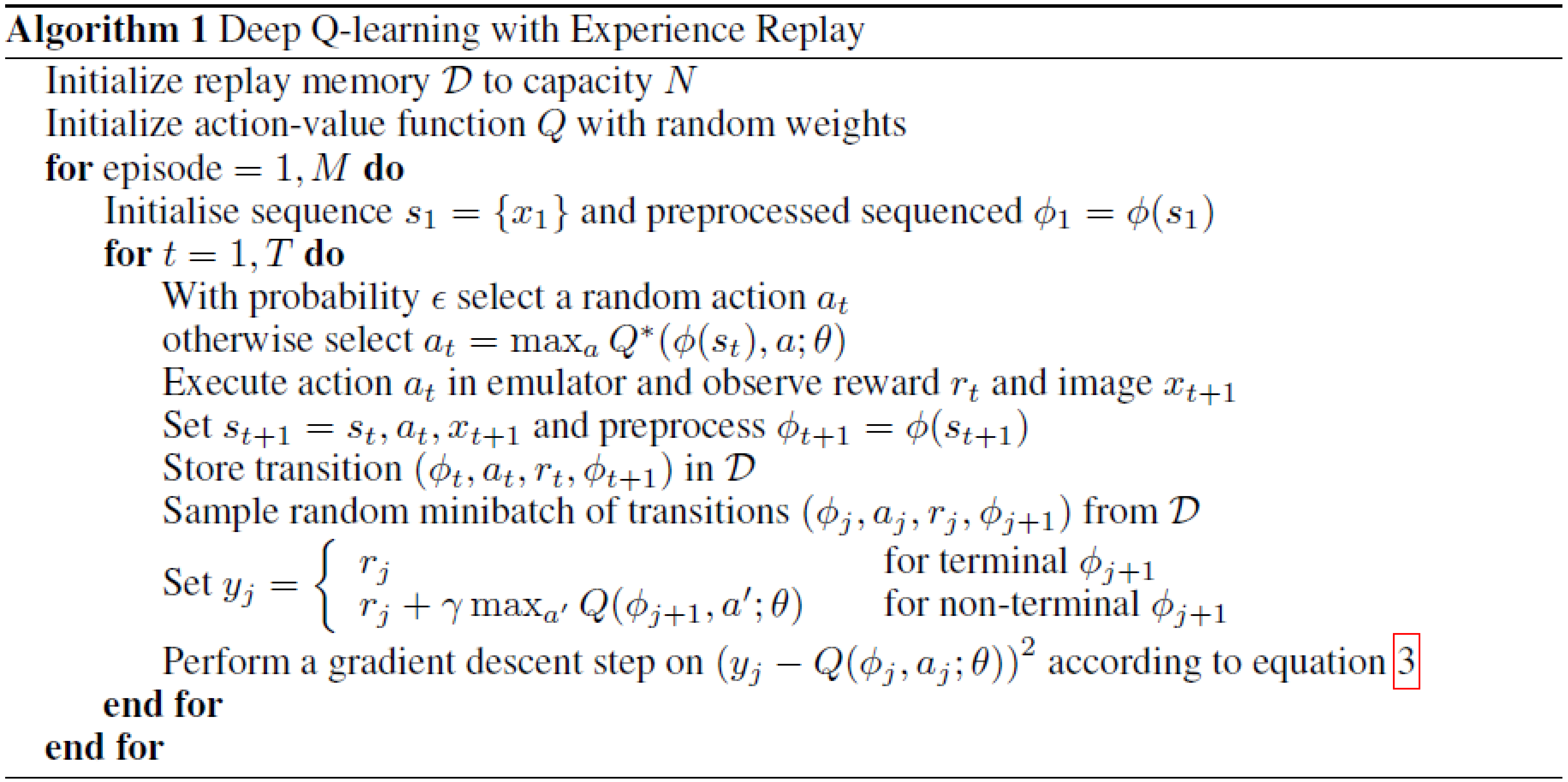
Deep Q-Learning

Instead of a table, use a neural network to approximate Q.

Update Q given a sample of (s,a,r,s’) datapoints by minimizing loss via SGD:

$$Loss = \frac{1}{2}\left(r + \lambda\, max_{a'} Q^N(s', a') - Q^N(s, a)\right)^2$$

Deep Q Network Basic Algorithm

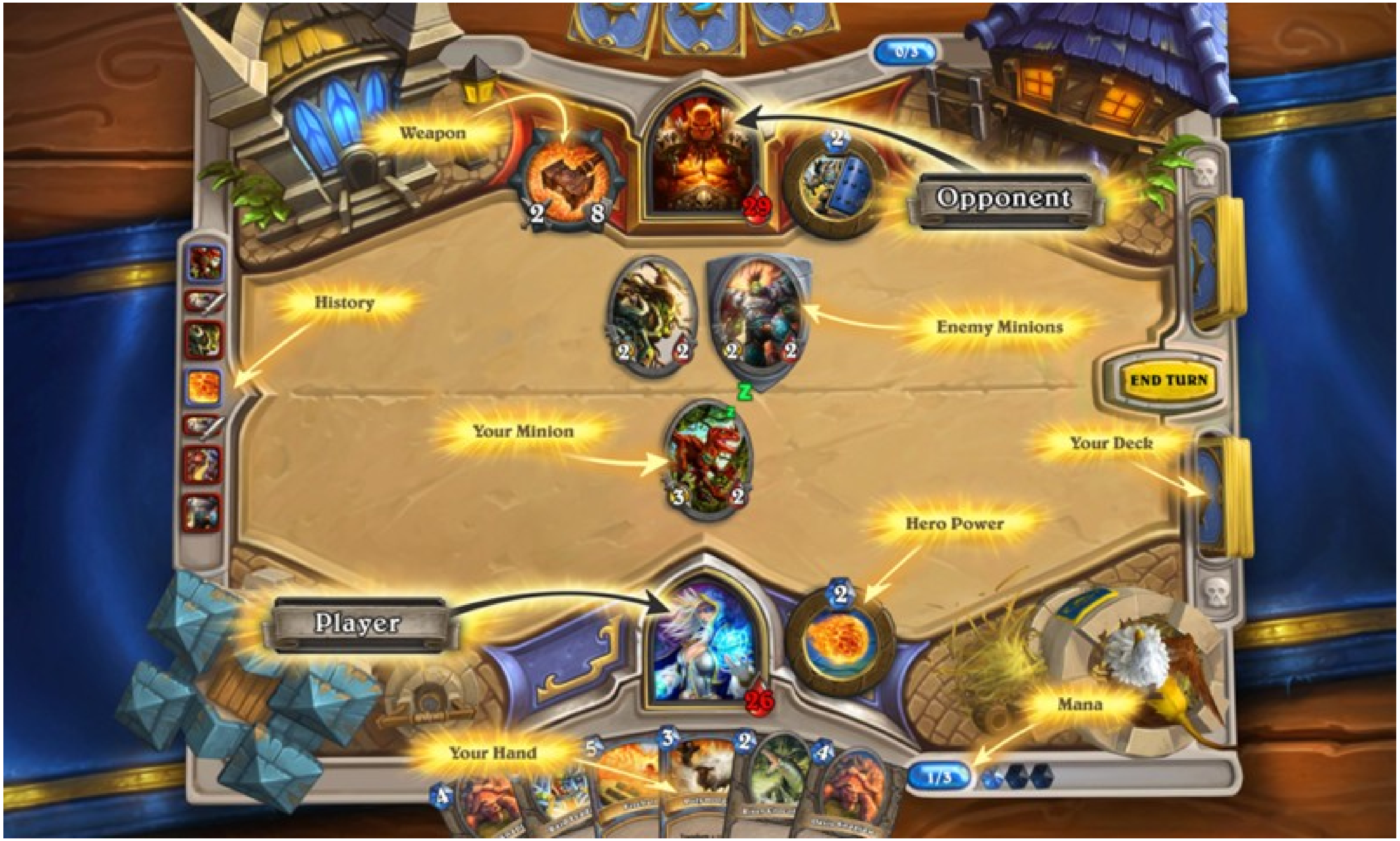


Algorithm 1 Deep Q-learning with Experience Replay

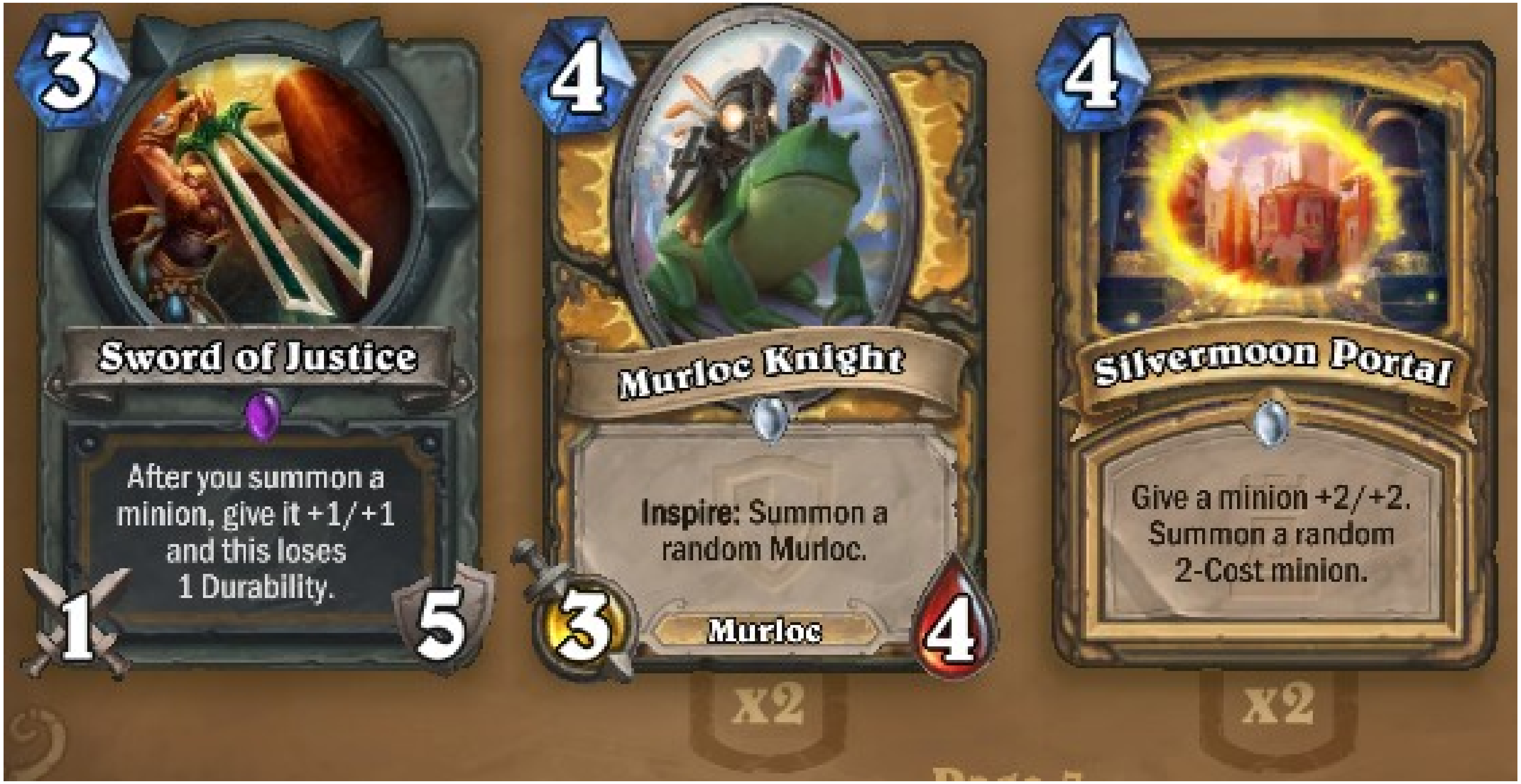
```
Initialize replay memory $\mathcal{D}$ to capacity $N$
Initialize action-value function $Q$ with random weights
for episode $= 1, M$ do
    Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
    for $t = 1, T$ do
        With probability $\epsilon$ select a random action $a_t$
        otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
        Execute action $a_t$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
        Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation $\boxed{3}$
    end for
end for
```

Mnih, Volodymyr et al. “Playing Atari with Deep Reinforcement Learning.” ArXiv abs/1312.5602 (2013): n. pag.

Hearthstone



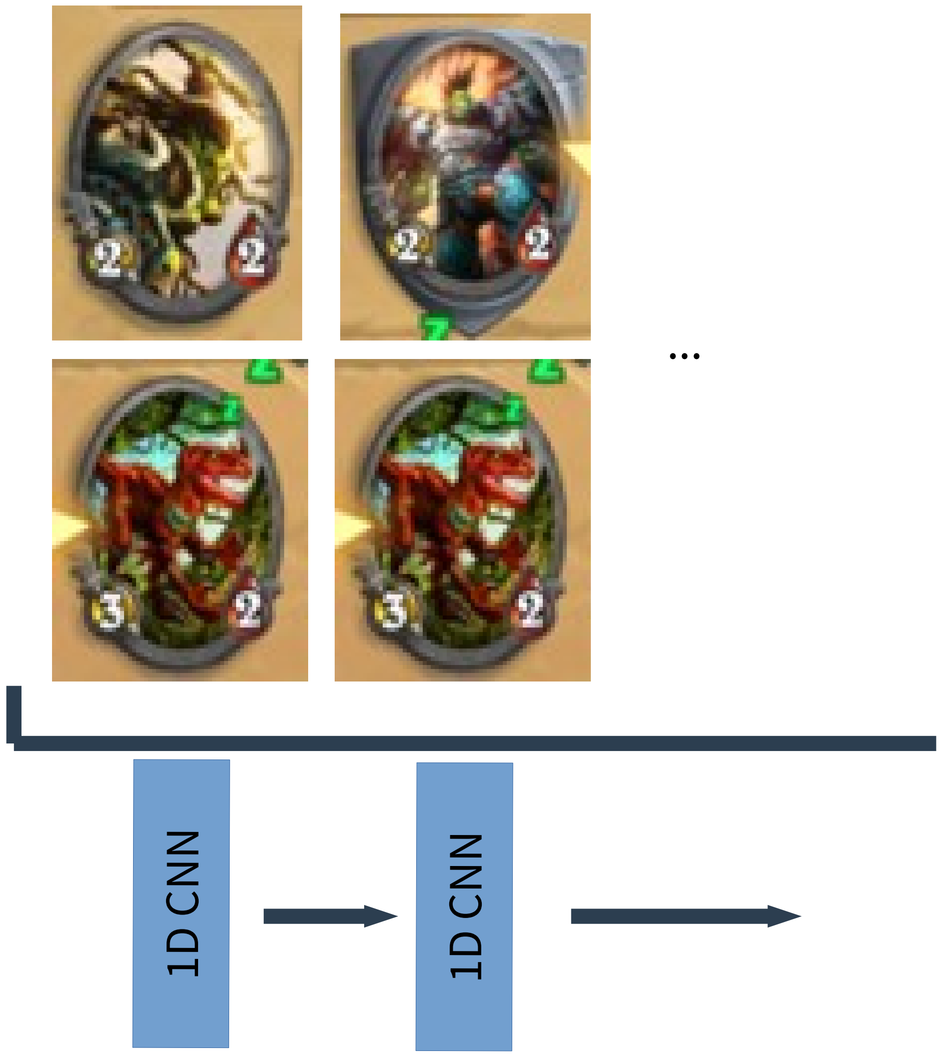
Hearthstone


3
Sword of Justice
After you summon a minion, give it +1/+1 and this loses 1 Durability.
1
5
4
Murloc Knight
Inspire: Summon a random Murloc.
3
Murloc
4
x2
4
Silvermoon Portal
Give a minion +2/+2. Summon a random 2-Cost minion.
x2

Challenges

Typical Past Applications	**Hearthstone**	**Solution**
Small, constant or near constant action space across states	Large, variable action space across states	Save possible successor actions along with (s,a,r,'s). i.e. (s,a,r,s,{a'}). Let *a* be in the neural network's input and output one value, rather than inputting just *s* and returning a value for each action.
Perfect information	Hidden information	Let the hidden information be a part of the randomness inherent in the environment. Let past states be part of the input so the neural network can infer the opponent's strategy/hand.
One move per turn	Multiple moves per turn, some of which are stochastic	Use tree search to get the possible move combinations. Define the state you end your turn on as the action *a*. Simulate multiple actions for stochastic actions and average their Q value.

Network Inputs: Minion Pairs



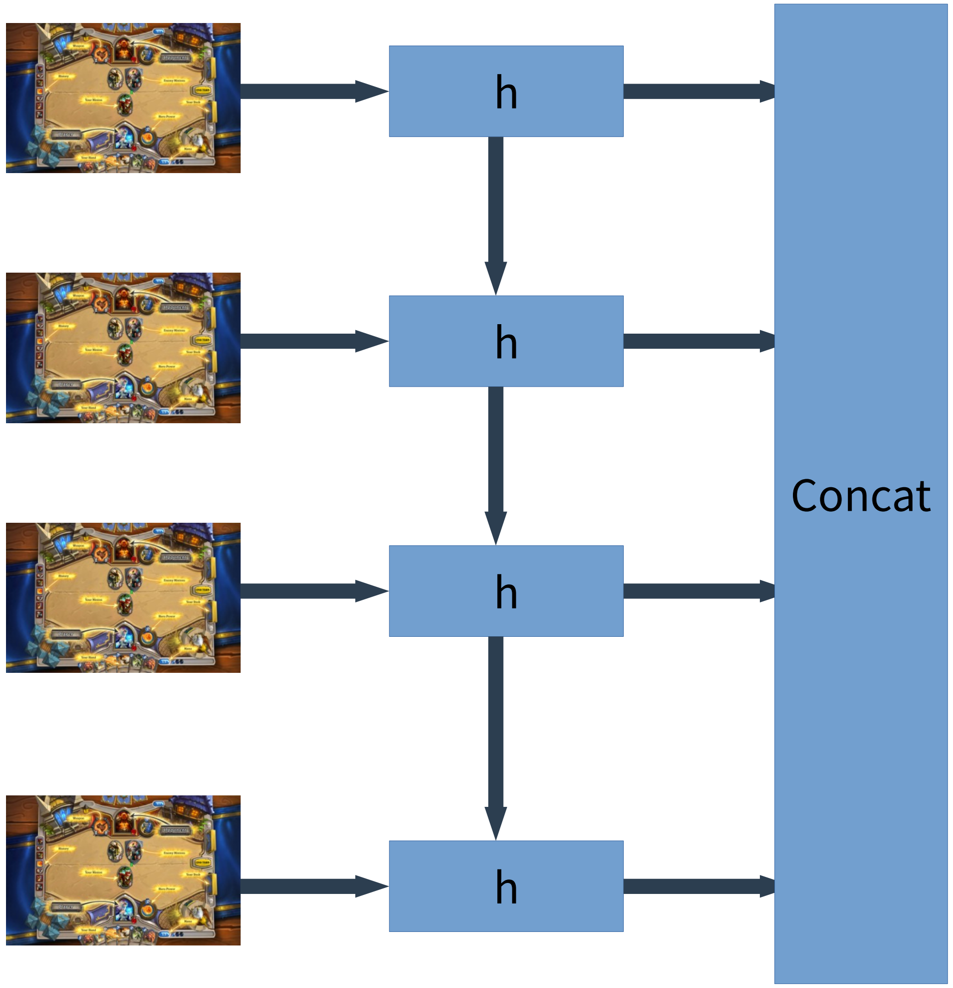
Input each possible pairing of friendly and enemy minion.

Captures data regarding minion interactions.

Use 1D convolution to analyze each pair in the same way, saving weights.

Sort the pairs so the result is order agnostic.

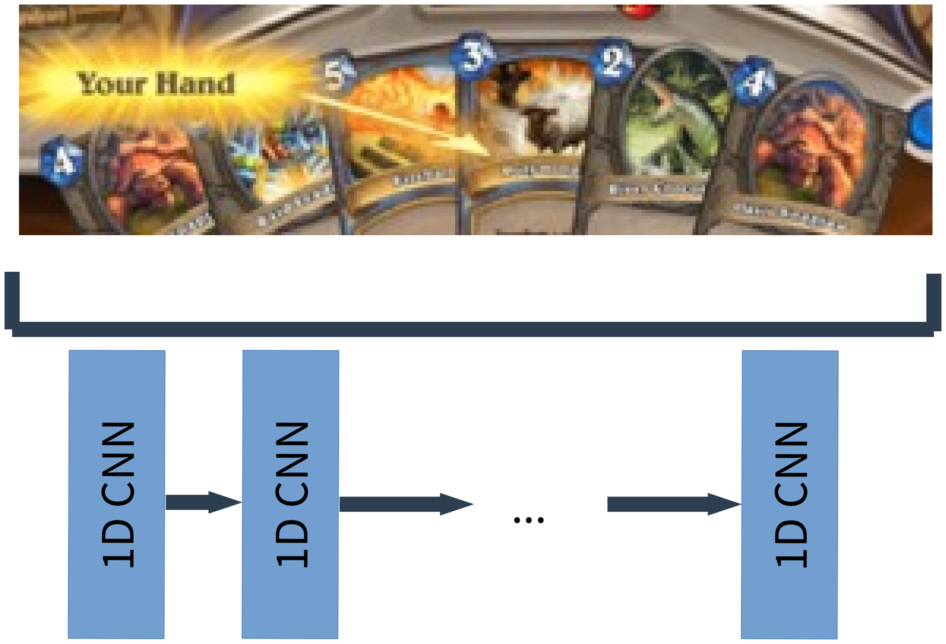
Network Inputs: Board History



Input the important board information of the last few states and feed them through a Recurrent NN.

This will allow the network to estimate hidden information based on recent action or inaction of the opponent.

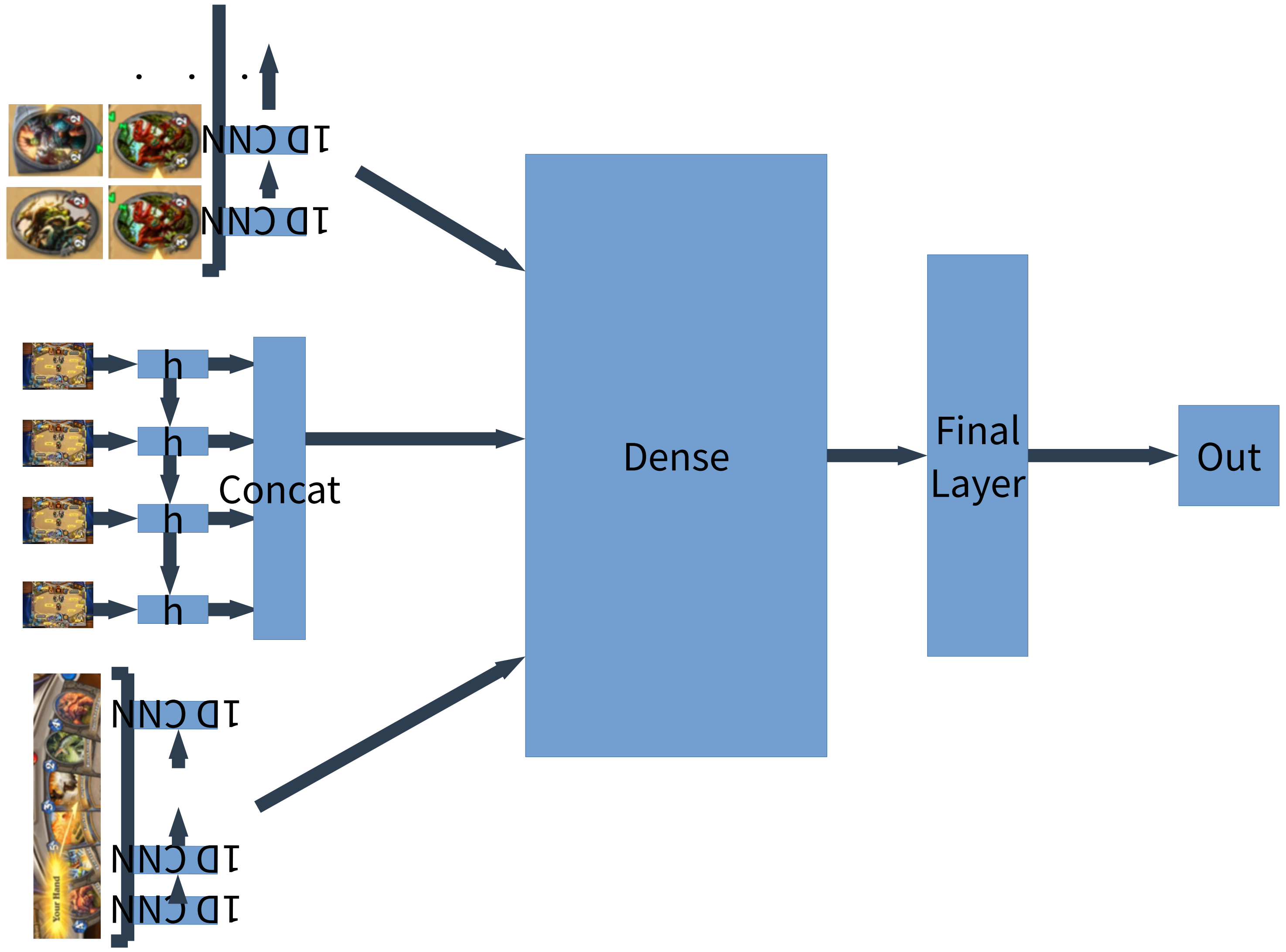
Network Inputs: Player's Hand



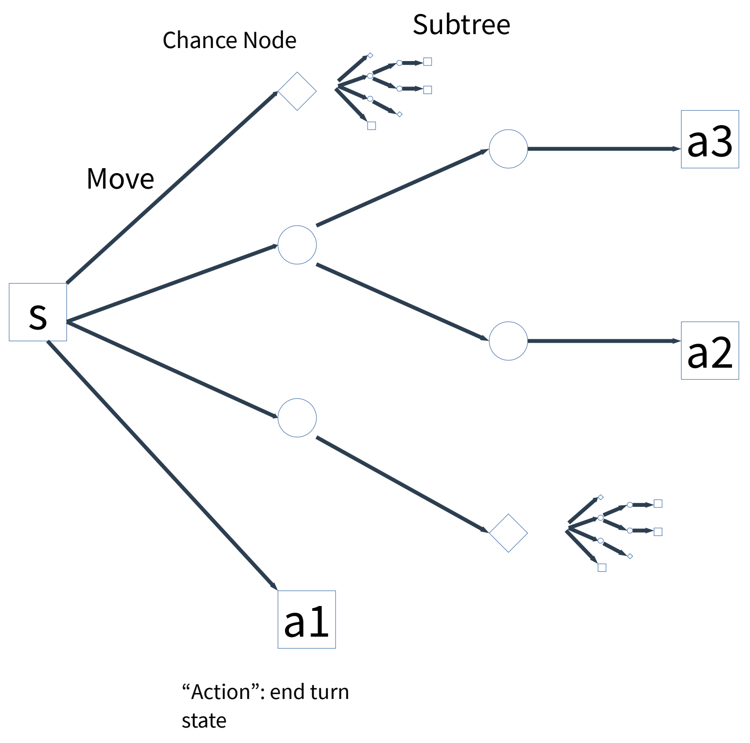
Put each card in your hand through a 1D Convolutional NN.

Sort the cards to ensure the input is order-agnostic.

Total Network



Tree Search for Identifying Actions



- Starting with the start of the turn *s* as the root, search to find all combinations of moves you can take during your turn, ending with the "end turn" move.
- If the result of a move is stochastic, make a *chance node* containing subtrees whose roots are a sample of possible resulting states.
- To choose an action, find the *end turn* or *chance* node with the highest score; the score of an *end turn node* is according to Q(s,a); the score of a *chance node* is the average maximum Q(s,a) score of its subtrees.
- If an *end turn node* is chosen, wait for your next turn before making more decisions. If a *chance node* is chosen, take the moves that lead to it, then take the best moves from the tree representing the resulting state you wound up in.
- Define the state you end your turn on as the "action" *a*, the start of your next turn as the "successor" *s'*, and the reward gained in between the two states as the observed reward *r*.

Decomposing Reward

Decompose immediate reward into two components:

Deterministic and known given state and action: $R^{det}(s,a)$

Stochastic, unknown, or also dependent on successor: $R^{sto}(s,a,s')$

$$R(s,a,s') = R^{det}(s,a) + R^{sto}(s,a,s')$$

Modify Q equation

$$Q(s,a) = R^{det}(s,a) + \sum_{s'} T(s,a,s')\left[R^{sto}(s,a,s') + \lambda \max_{a'} Q(s',a')\right]$$

Decomposing Reward

Change purpose of the neutral network N(s,a):

Estimates stochastic reward portion plus future reward

$$Q(s,a)=R^{det}(s,a)+N(s,a)$$

$$N(s,a)=\sum_{s'} T(s,a,s')\left[R^{sto}(s,a,s')+\lambda\, max_{a'}(R^{det}(s',a')+N(s',a'))\right]$$

Making Decisions:

$$\pi(s)=argmax_{a}(R^{det}(s,a)+N(s,a))$$

Training Network

Update N given a sample of (s,a,r,s’) datapoints by minimizing loss via SGD:

- Squared error between predicted network output and calculated target
- Note that *r* is a sample of the stochastic component of the whole immediate reward

$$Loss^N = \frac{1}{2}\left(r + \lambda\, max_{a'}\left(R^{det}(s', a') + N(s', a')\right) - N(s, a)\right)^2$$

Effects

Gives learning a "head start."

Less exploration is needed at the start to find sensible actions.

Doesn't make the network learn what is "already known."

The greater the deterministic portion is, the more pronounced these effects.

Quicker convergence?

Applying to Hearthstone

The deterministic component consists of reward gained/lost by the action taken during your turn.

According to a hand-crafted reward function.

The stochastic component consists of rewards gained/lost by the action the opponent takes during their turn.

Using Transfer Learning

In Hearthstone, each player starts with a deck of 30 out of the hundreds of cards available. The player knows the contents of their own deck, but not their opponent's.

First, train a network on games with random decks.

This lets the network learn how to evaluate actions in general.

Then, train a network on games using one deck and additional features based on the deck.

This lets the network learn to play one deck starting using the previously learned game knowledge.

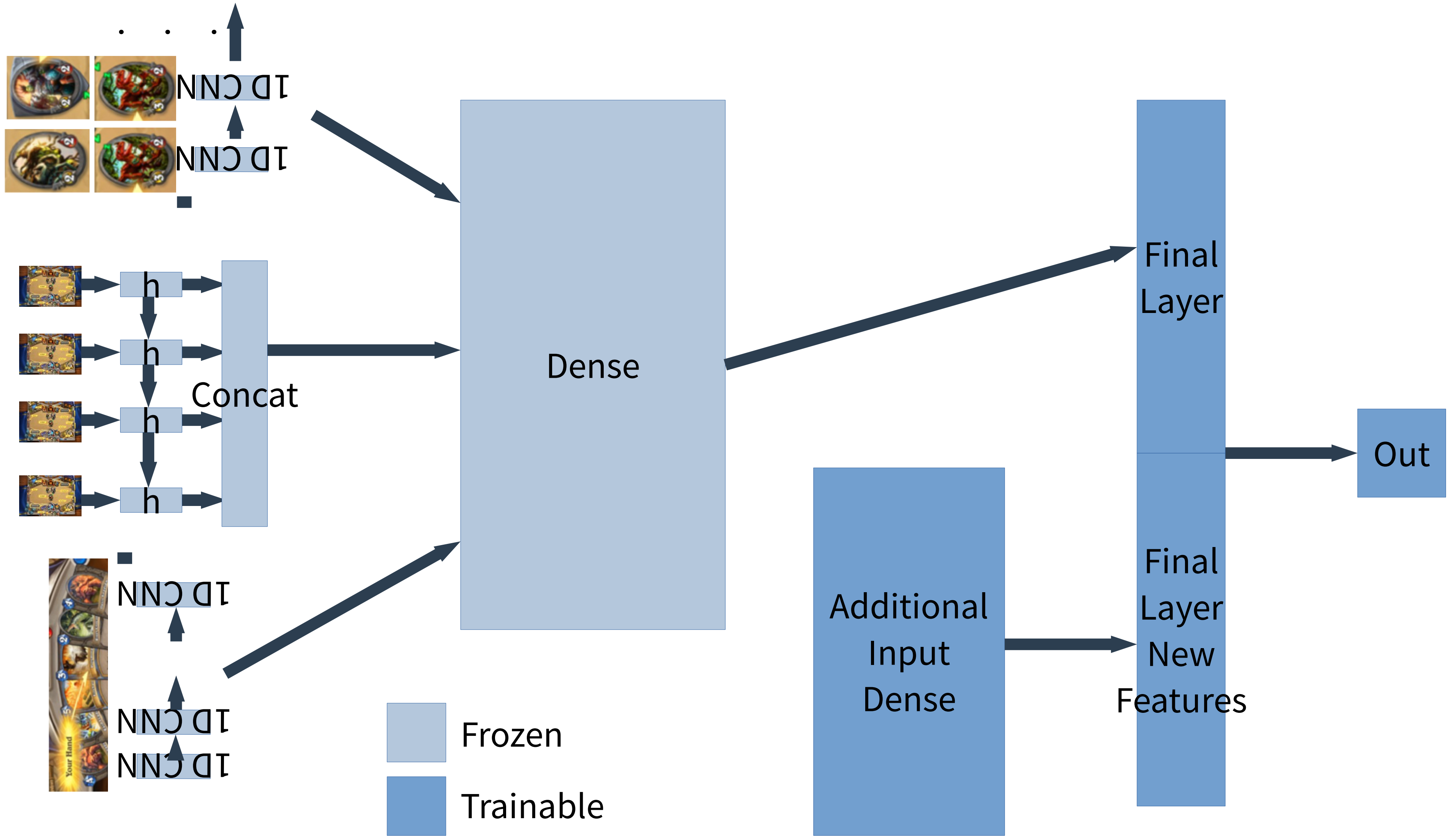
Using Transfer Learning: Additional Inputs

A deck contains 30 cards, each of which can be in 1 of 4 locations (deck, hand, field, or graveyard).

Represent this as 30, 4-length one-hot vectors, and input it through a dense NN.

When training new network, freeze all the old weights except the ones directly connected to the output.

Transfer Learning Network



Progress

Finished*:

Research and Design

Tree search

Scorer

To do:

Neural Network

Training