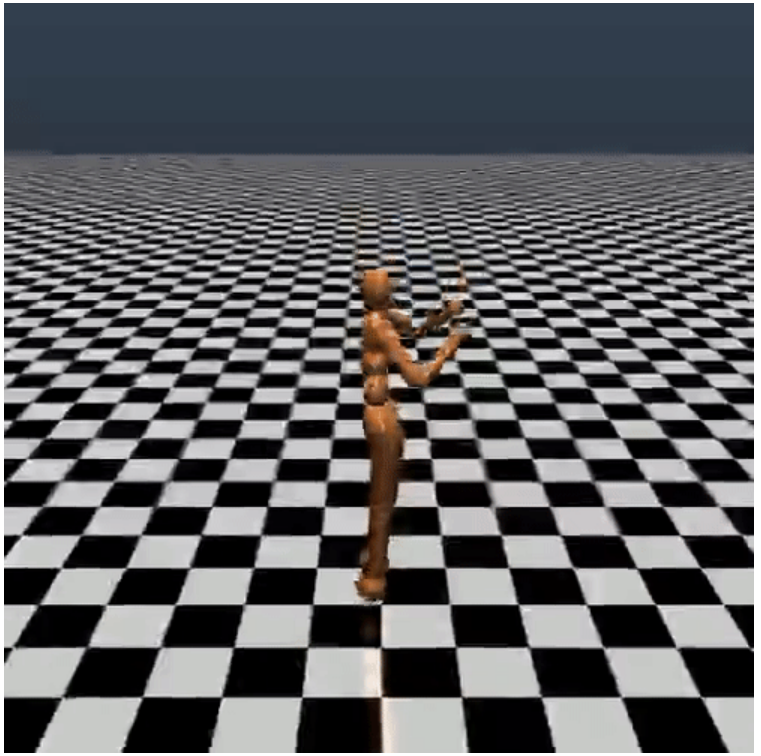
# Deep Q-Learning

# Goals

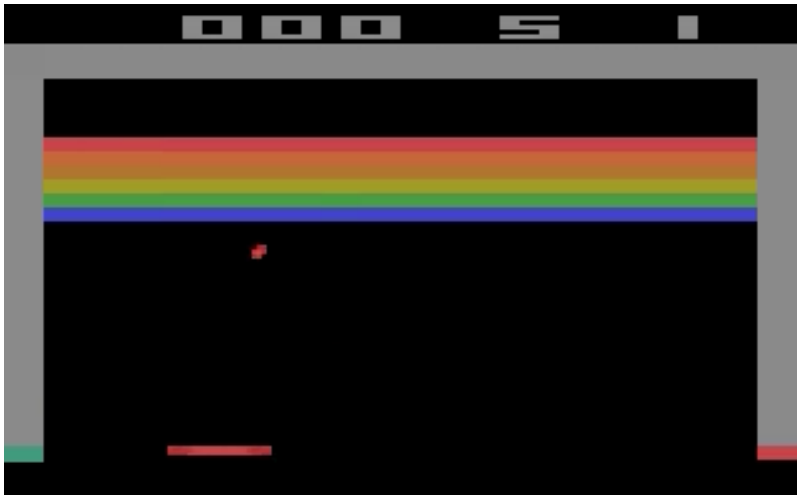
- Overview Reinforcement Learning
- Q-Learning
- Deep Q-Learning
- Playing Games
- Improvements
- Do it Yourself

# Reinforcement Learning

Continuous Control



Atari Games

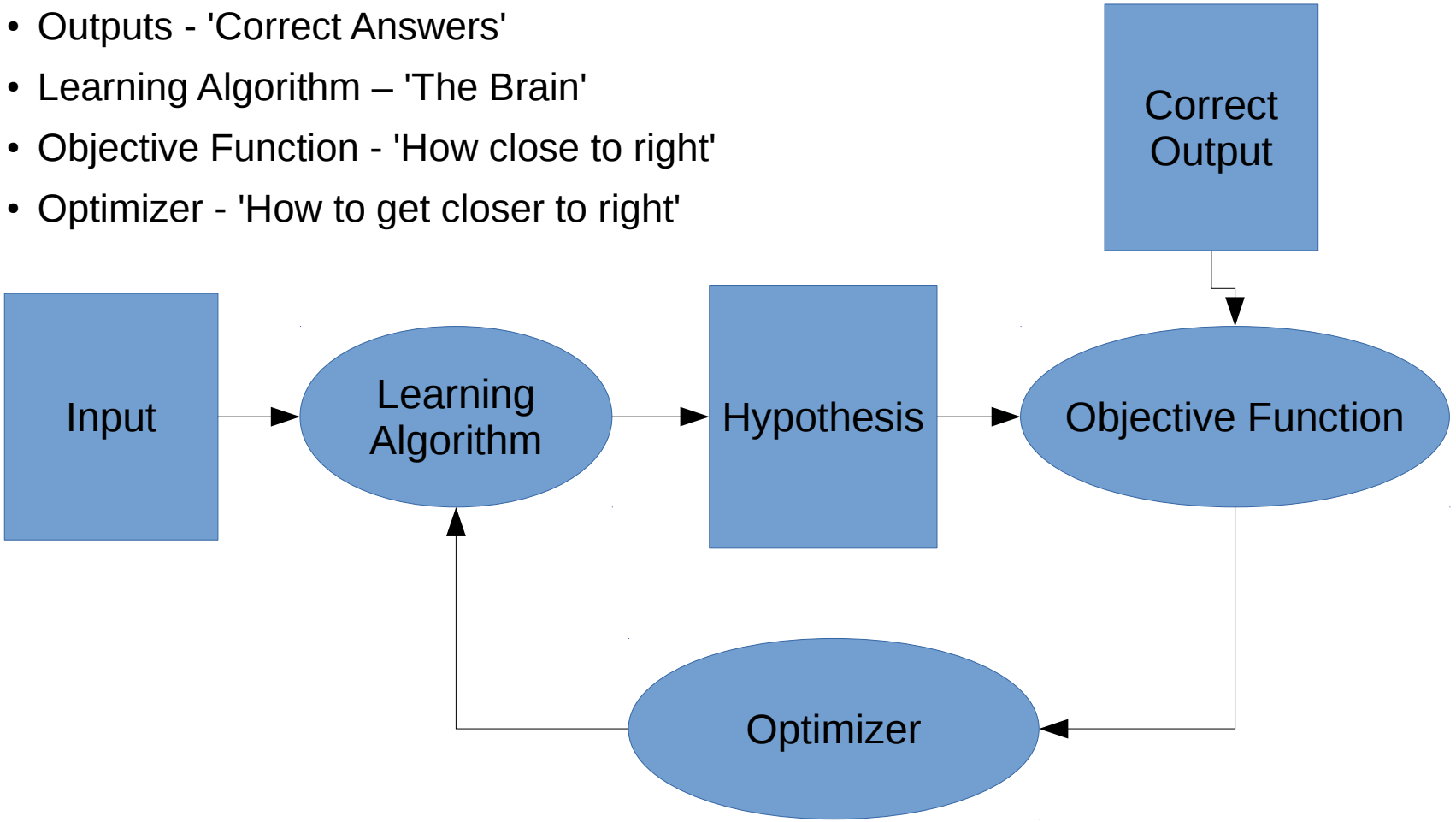


https://gym.openai.com/evaluations/eval_i3x1JpReRukTZrznxypCw, https://github.com/tambetm/simple_dqn

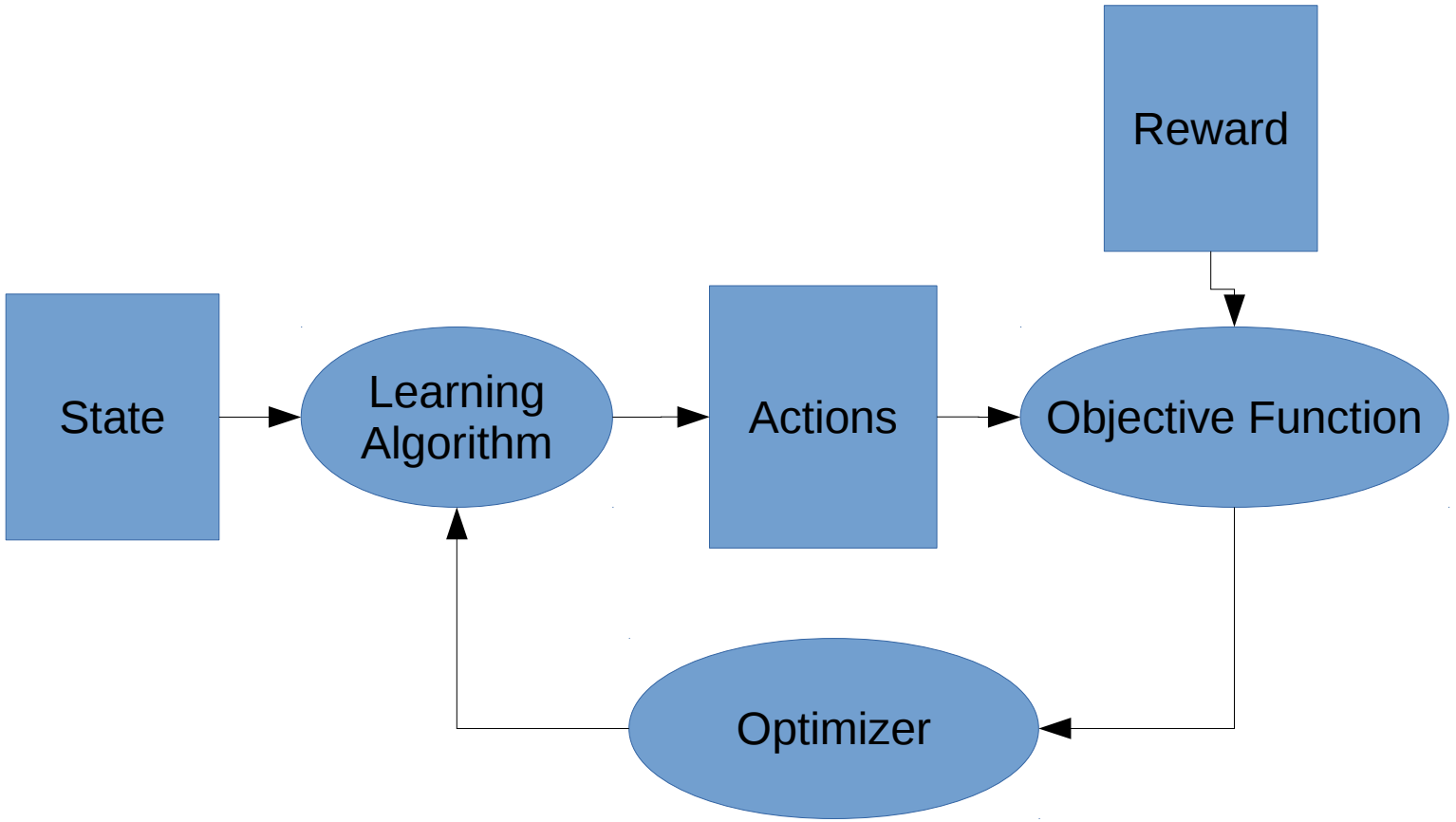
# ML basics

Components of an iterative learning system:

- Inputs - 'Data'
- Outputs - 'Correct Answers'
- Learning Algorithm – 'The Brain'
- Objective Function - 'How close to right'
- Optimizer - 'How to get closer to right'



# Reinforcement Learning



# Reinforcement Learning

- The real world: unsupervised

  Goal:
- Learn a policy that maximizes cumulative future reward

  Difficulties:
- Sparse, time delayed reward
- Credit assignment
- Exploration vs exploitation
- Action space size
- Observation space size

# Reinforcement Learning Approaches

- Policy Iteration / Gradient
  - Temporal Difference
    - TD-Gammon
    - Q-Learning
- Stochastic Derivative Free
  - Cross Entropy Method

# Markov Decision Process

- Agent
- Environment
- State
- Actions
- Reward
- Policy
- Discount
  - Discounted Future Reward
- Learning Rate

- Markov Assumption: $P(S_{i+1})$ is determined solely by $S_i$ and $a_i$

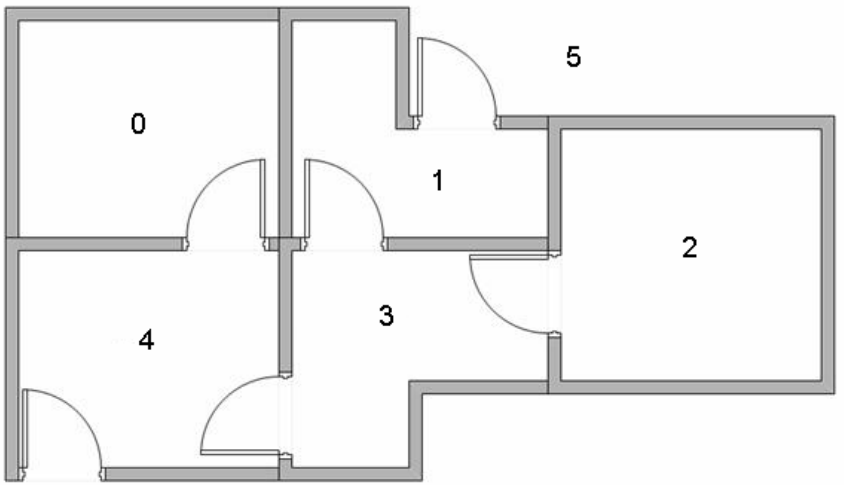
- Partially observable
- Partially Random
- Discrete Time

# Q-Learning

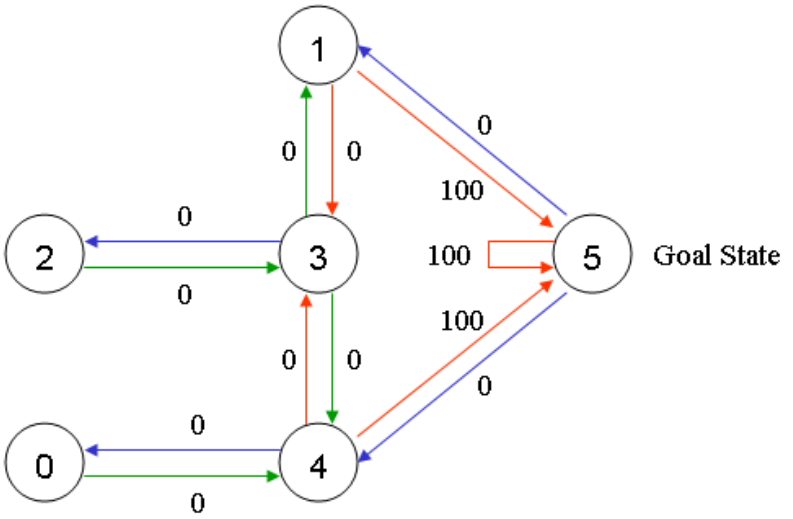
- Basic Q-Learning Algorithm

Initialize *Q(s,a)* arbitrarily
Repeat until terminal
Choose *a* using policy given *Q* (eg epsilon-greedy)
Take action *a*, observe *r*, *s'*
Update *Q(s,a)* towards $r + \max_{a'} Q(s',a')$

Goal: get outside




State, Action, Reward




http://mnemstudio.org/path-finding-q-learning-tutorial.htm

# Q-Learning

- Will converge to optimal solution
- Off Policy, what if exploration is costly?
- What if search space is too large

# Deep Q-Learning

- Replace Q-table with deep network
- Q(s,a) estimated by network

# Deep Q-Learning

Improvements:

- Experience Replay
  - <s,a,r,s'>
  - Minibatch from memory
  - Prevents training on too similar data
- Epsilon Greedy Exploration
  - Explore randomly occasionally
  - Decay over time
- Other methods:
  - Clip Error
  - Clip Reward
  - Target Network

# Q-Learn Loop

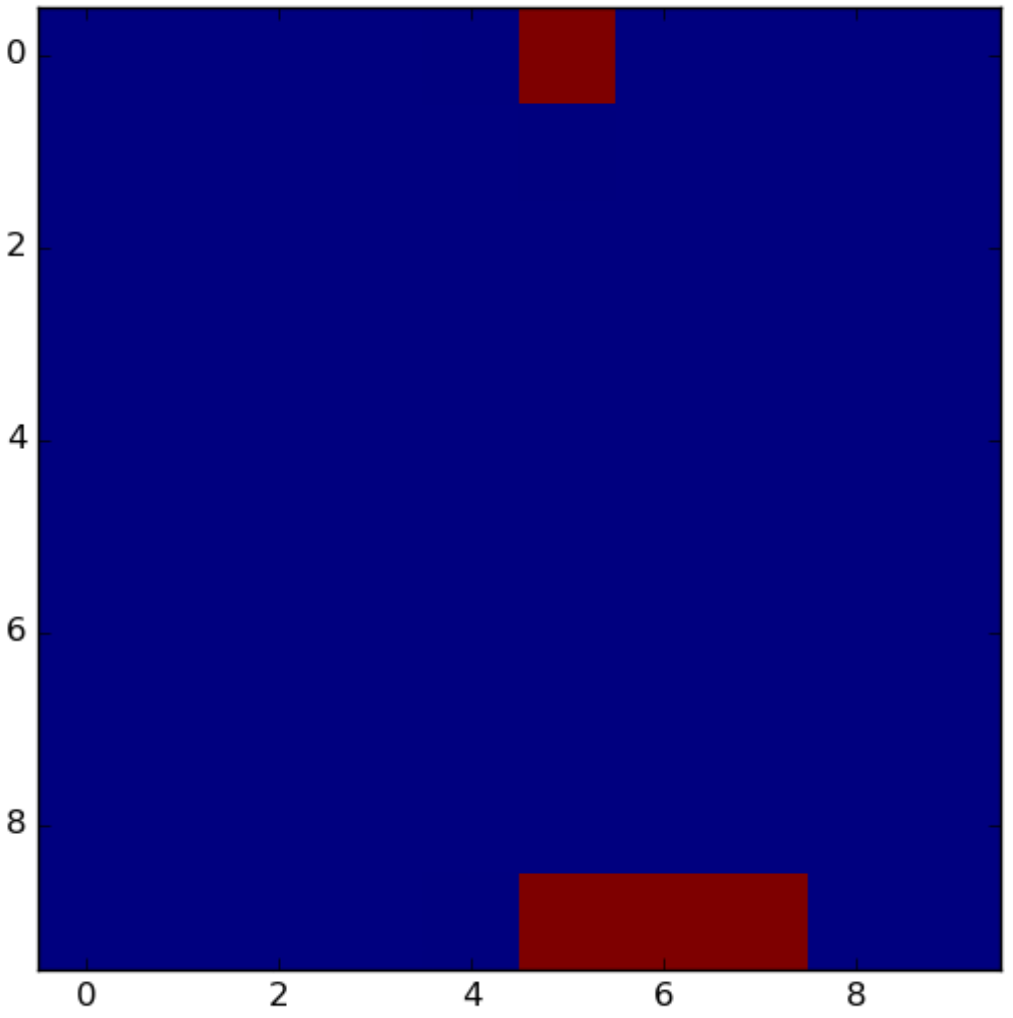
```
while not game_over:
    if np.random.random() < epsilon:
        a = int(np.random.randint(game.nb_actions))
    else:
        q = model.predict(S)
        a = int(np.argmax(q[0]))
    game.play(a)
    r = game.get_score()
    S_prime = self.get_game_data(game)
    game_over = game.is_over()
    transition = [S, a, r, S_prime, game_over]
    self.memory.remember(*transition)
    S = S_prime
    batch = self.memory.get_batch(model=model, batch_size=batch_size, gamma=gamma)
    if batch:
        inputs, targets = batch
        loss += float(model.train_on_batch(inputs, targets))
```

https://github.com/farizrahman4u/qlearning4k

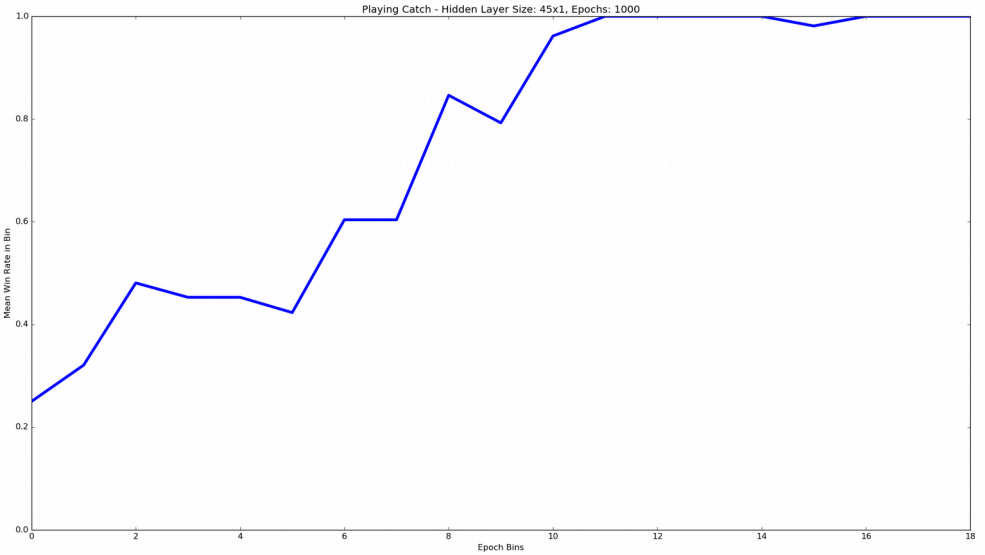
# Model Code

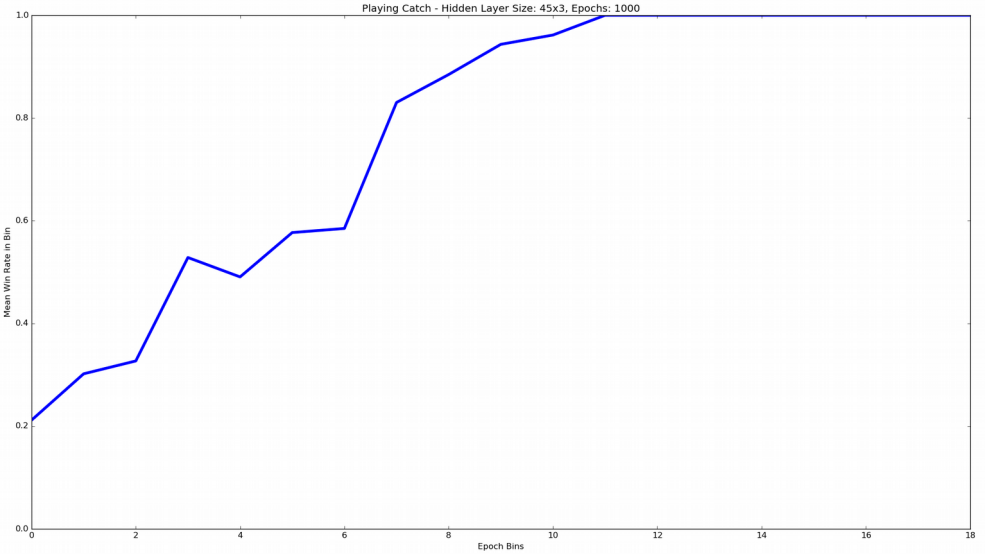
```
model = Sequential()
model.add(BatchNormalization(axis=1, input_shape=(nb_frames, grid_size, grid_size)))
model.add(Convolution2D(16, nb_row=3, nb_col=3, activation='relu'))
model.add(Convolution2D(32, nb_row=3, nb_col=3, activation='relu'))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(256, activation='relu'))
model.add(Dense(nb_actions))
model.compile(RMSprop(), 'MSE')

snake = Snake(grid_size)

agent = Agent(model=model, memory_size=1000, nb_frames=nb_frames)
agent.train(snake, batch_size=200, nb_epoch=5000, gamma=0.8, epsilon=[1,.01])
```

# Playing Catch

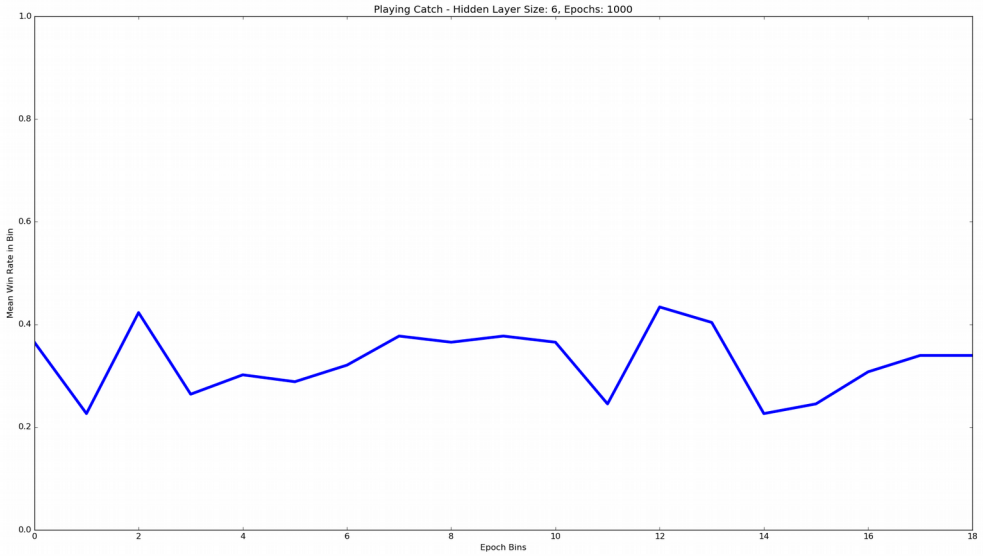


# Training: Hyperparameters

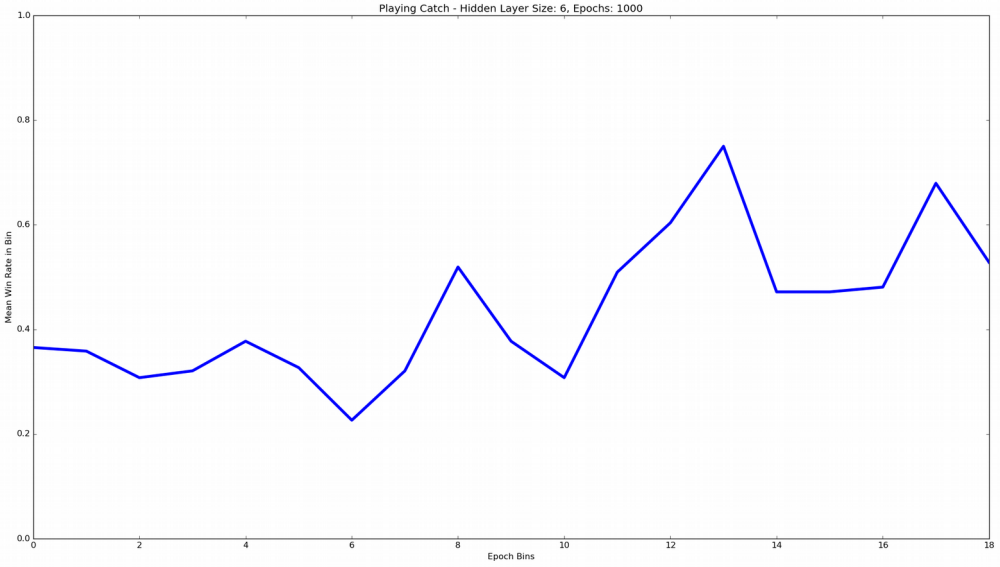


# Training: Hyperparameters

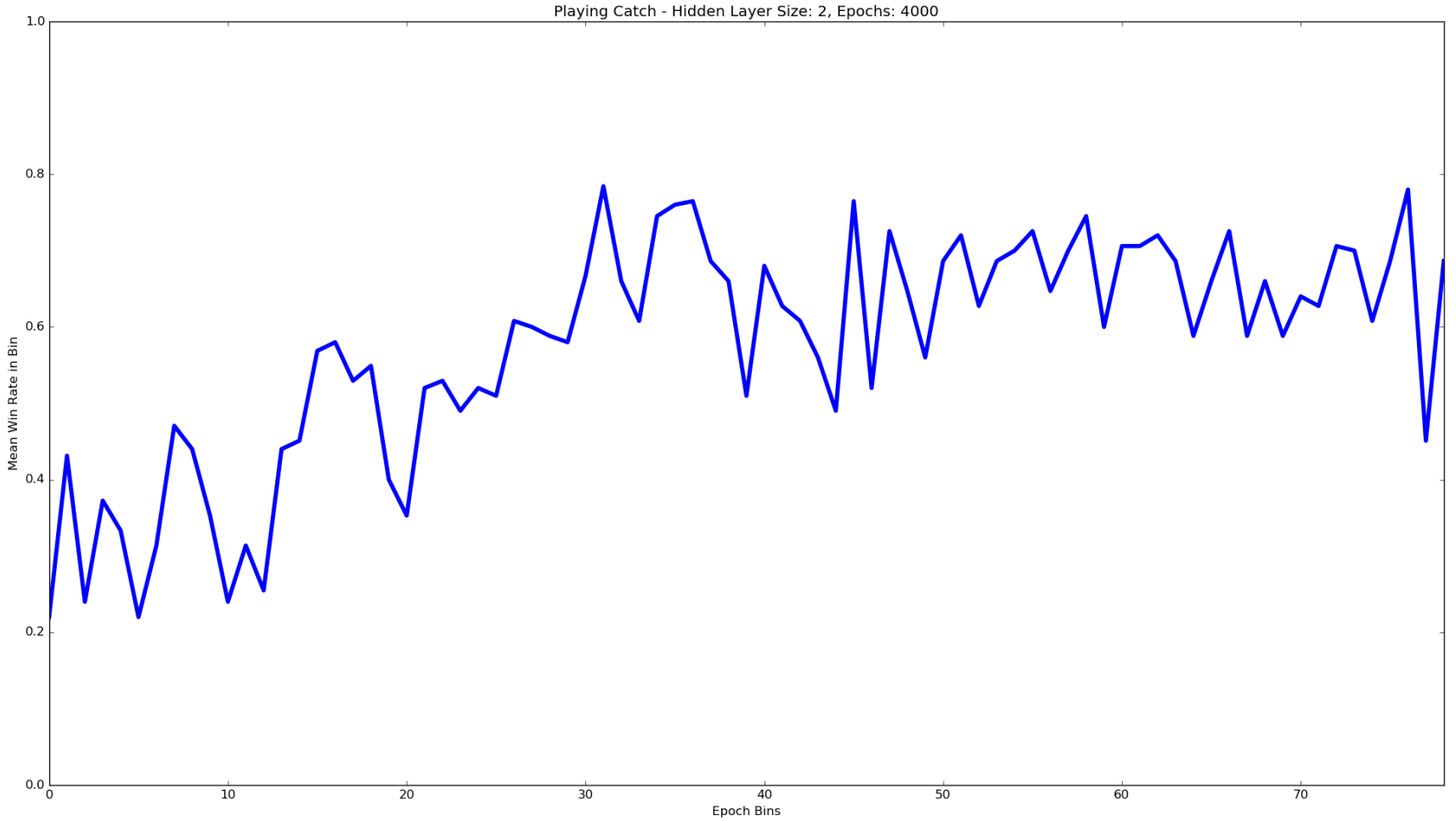



2x3 HL: Mean Loss over last 100: .256

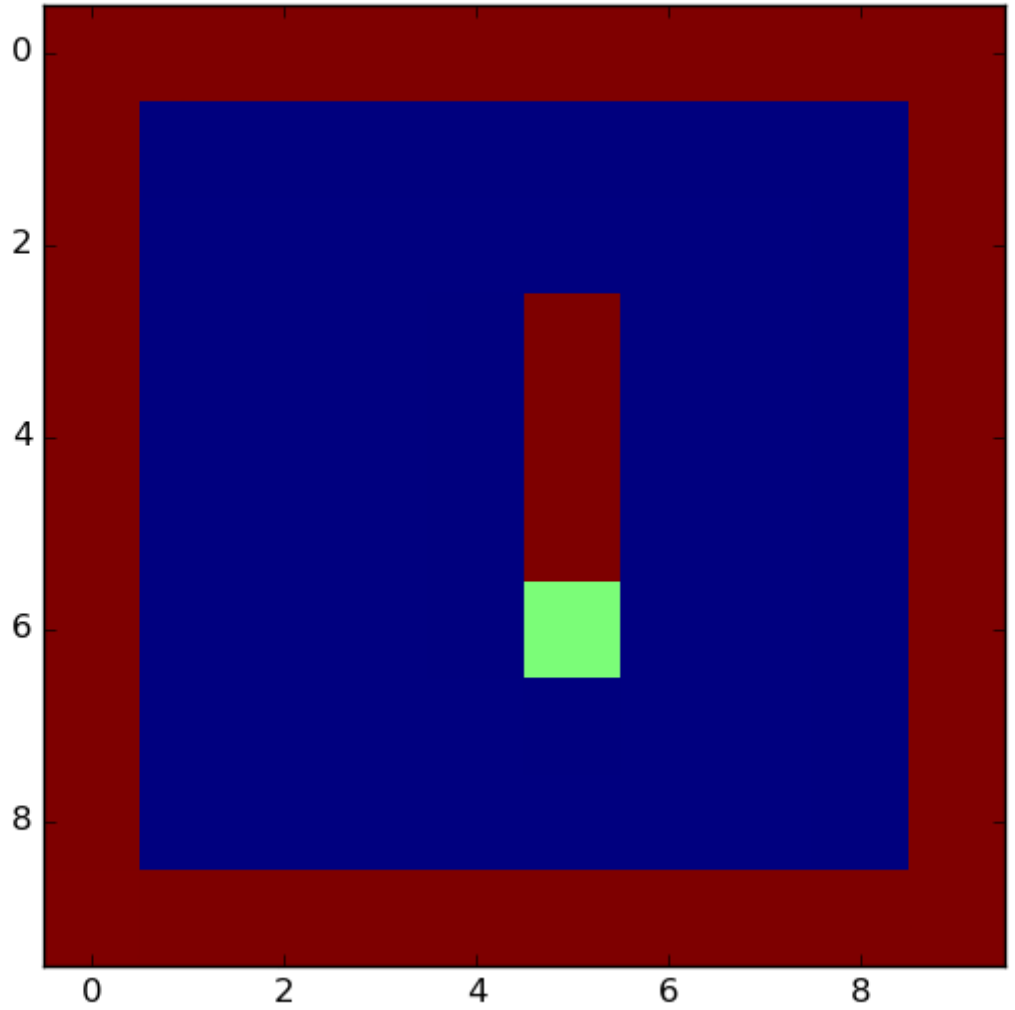



6x1 HL: Mean Loss over last 100: .112

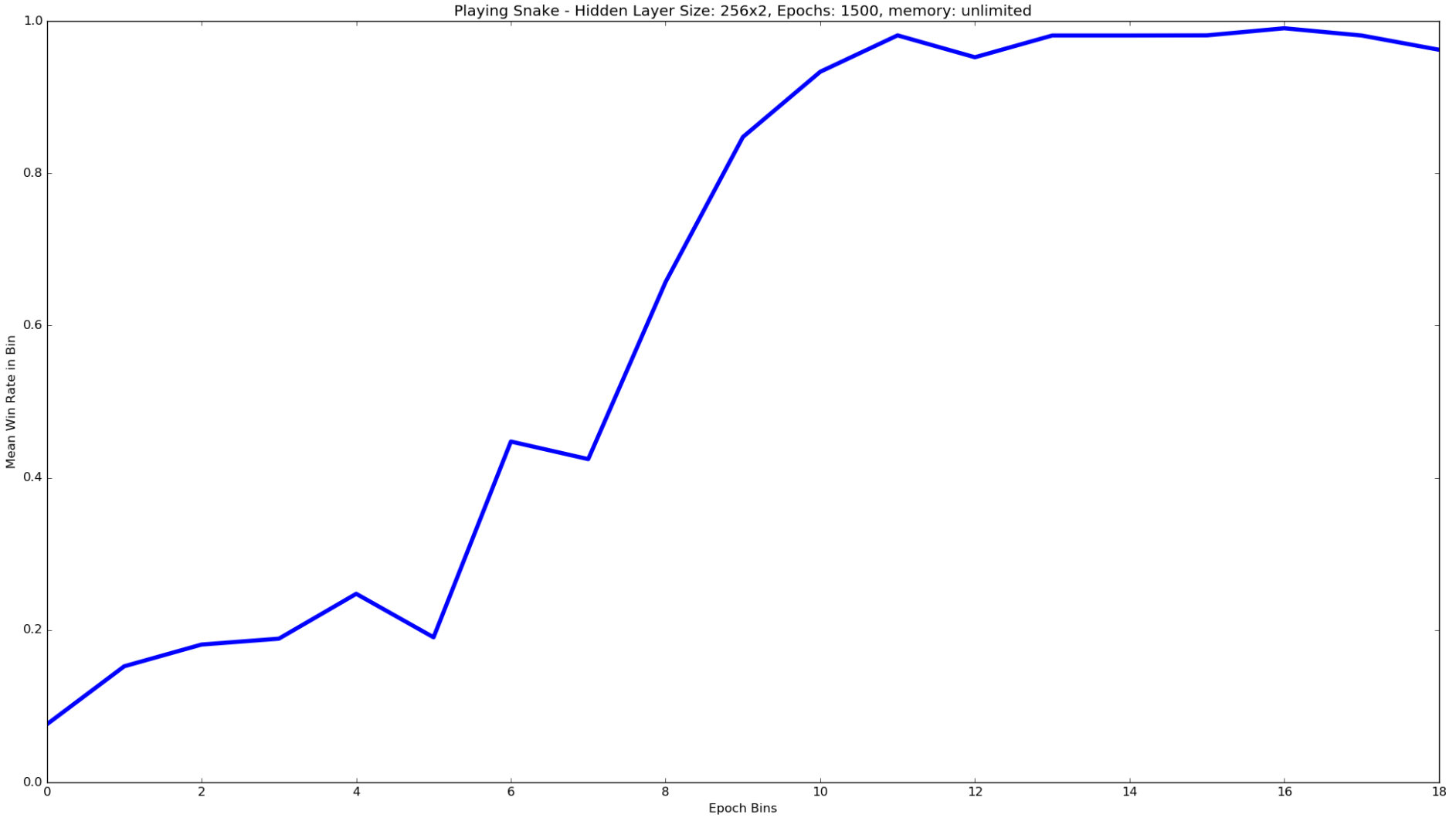
# Training: Hyperparameters



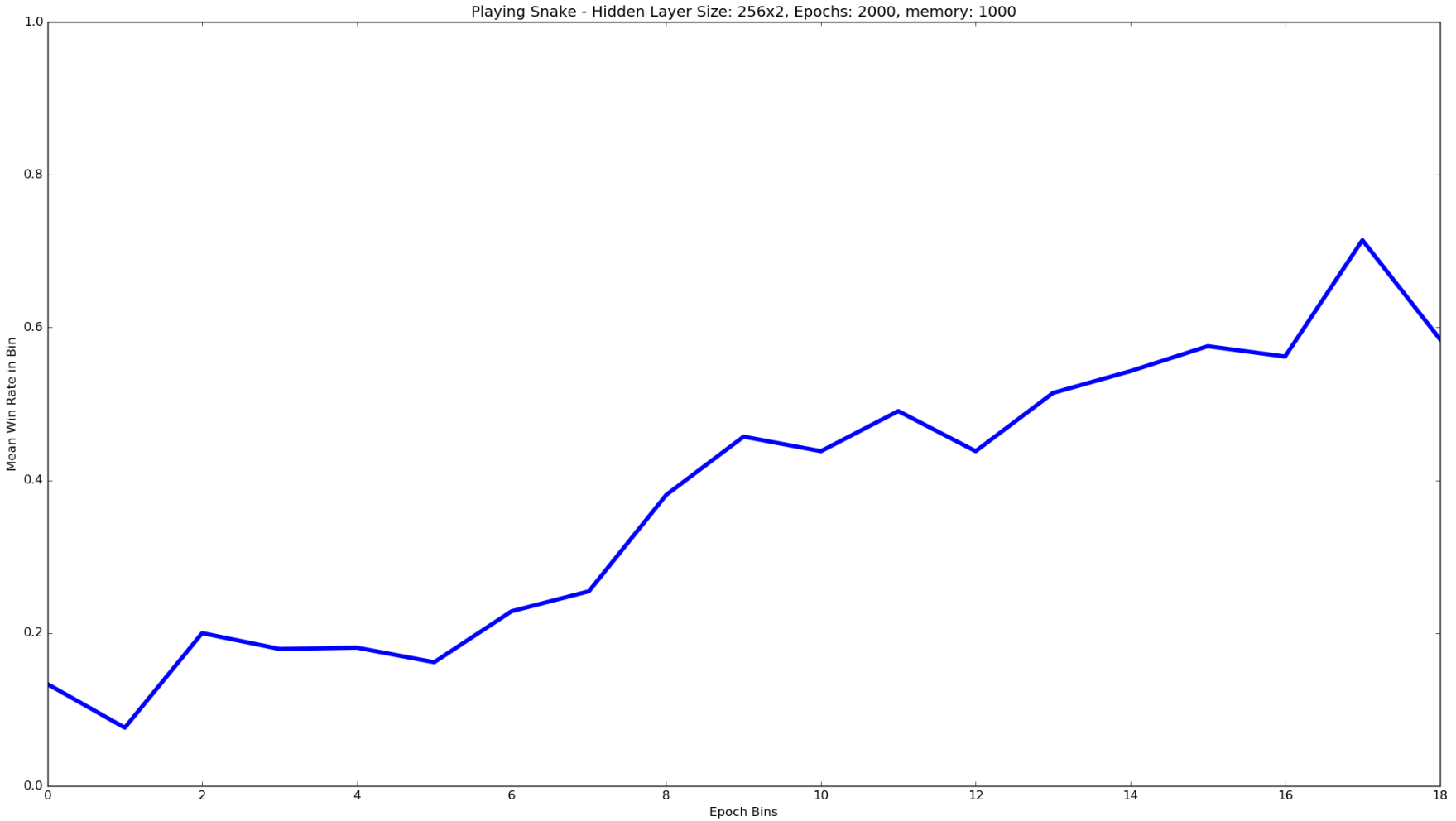
# Playing Snake



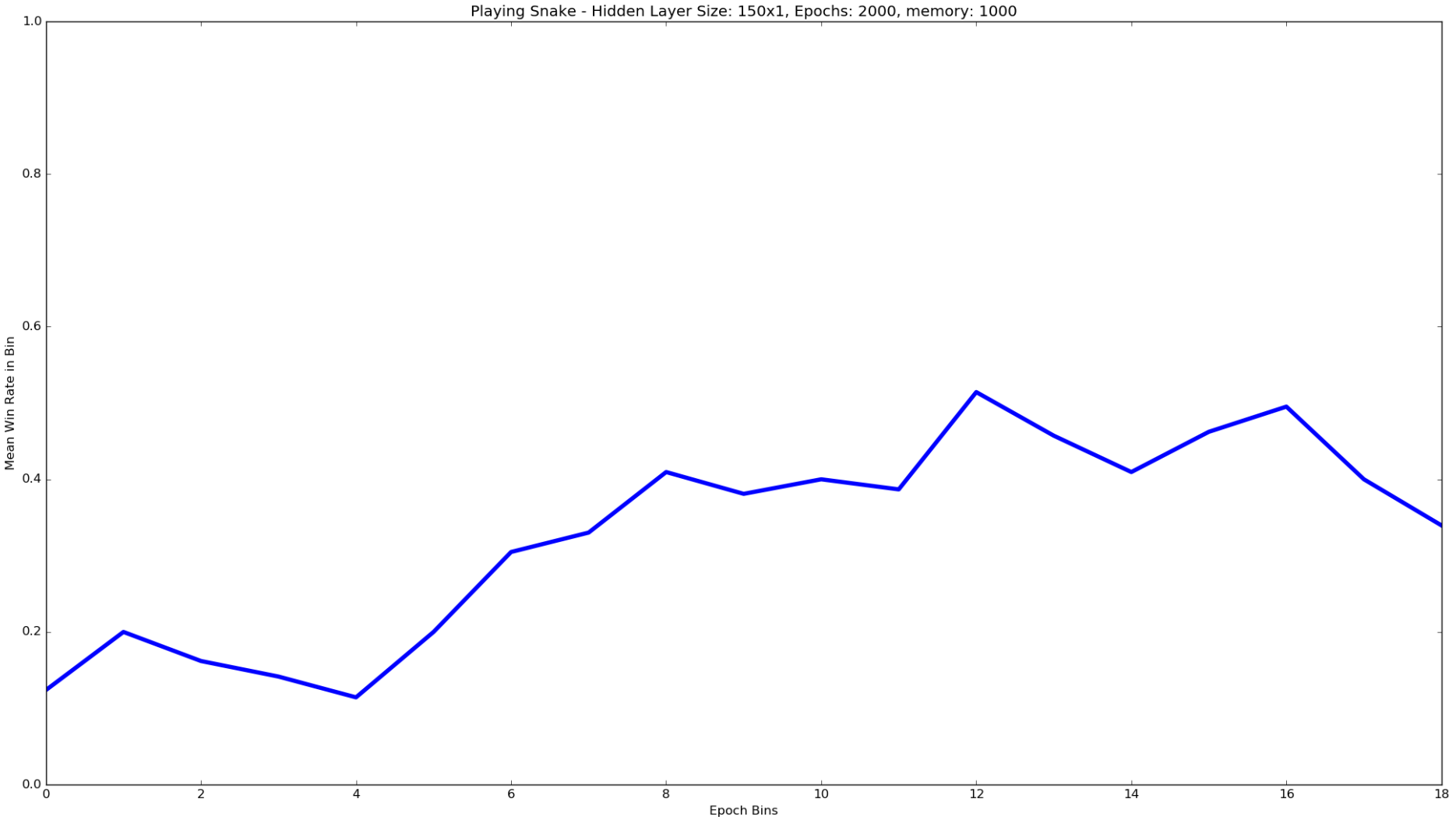
# Snake


Playing Snake - Hidden Layer Size: 256x2, Epochs: 1500, memory: unlimited
Mean Win Rate in Bin
Epoch Bins
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12
14
16
18

# Snake

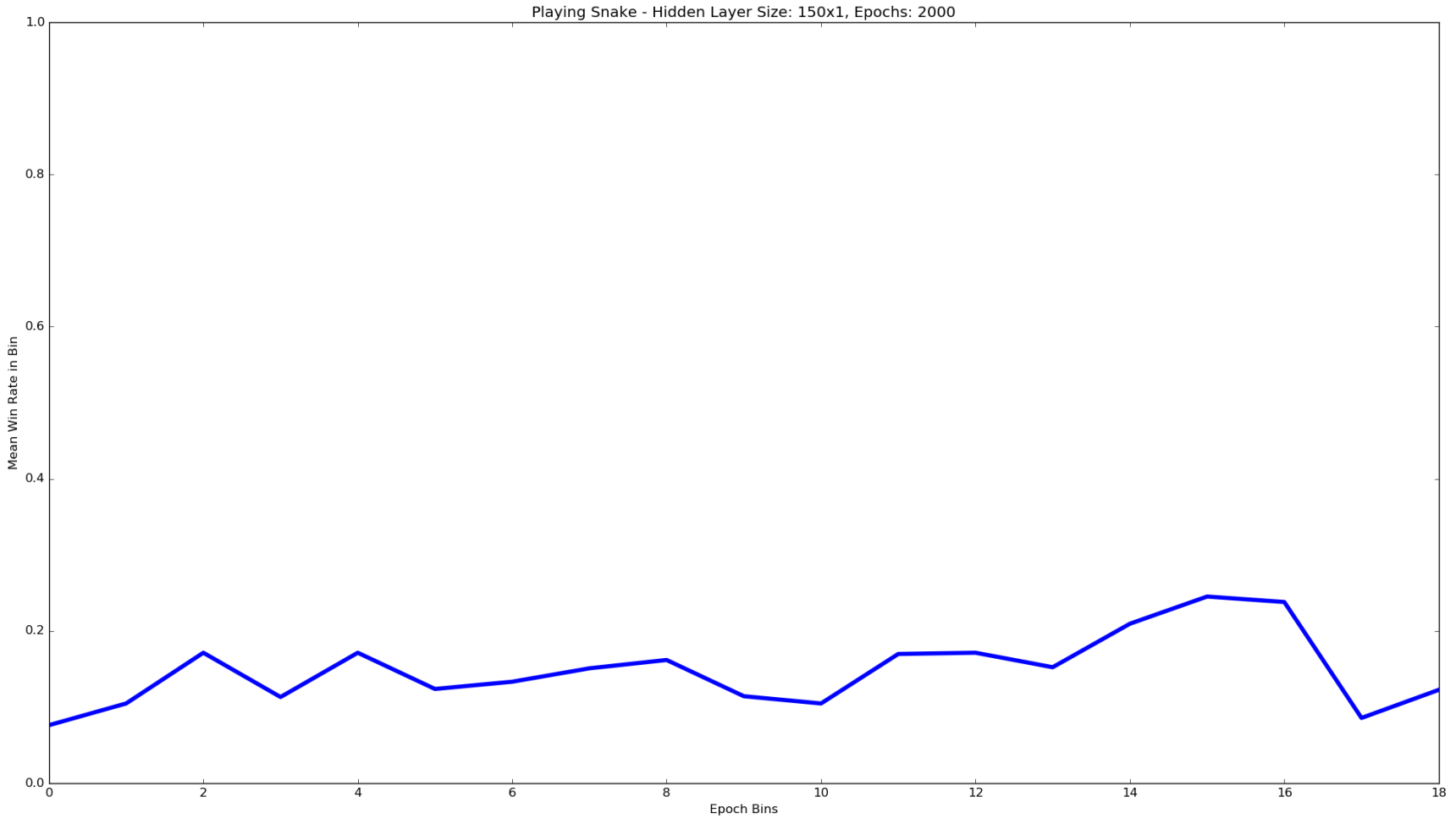

Playing Snake - Hidden Layer Size: 256x2, Epochs: 2000, memory: 1000
Mean Win Rate in Bin
Epoch Bins
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12
14
16
18

# Snake


Playing Snake - Hidden Layer Size: 150x1, Epochs: 2000, memory: 1000
Mean Win Rate in Bin
Epoch Bins
0.0
0.2
0.4
0.6
0.8
1.0
0
2
4
6
8
10
12
14
16
18

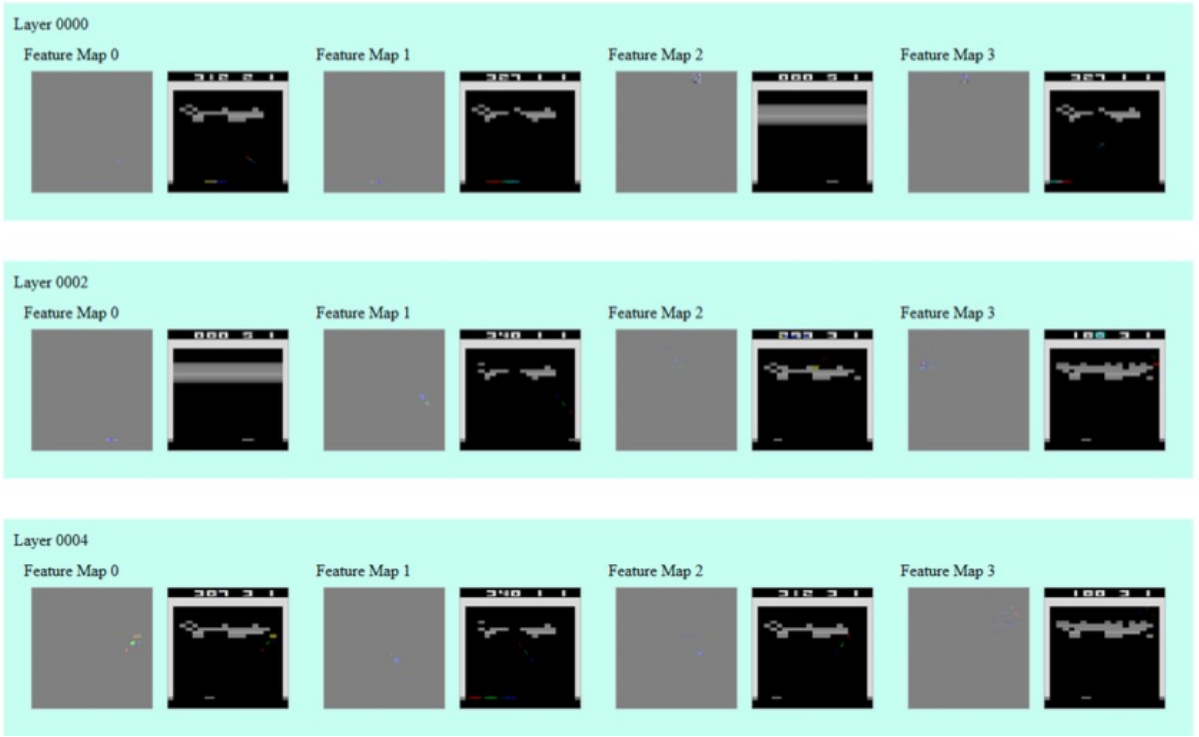
# Snake

Memory: 100



# Atari: Looking at the Layers

Left is the activation. Right is the Screen.




http://www.nervanasys.com/deep-reinforcement-learning-with-neon/

# Open-AI Gym

- Standardized comparisons
- Many Tasks (usually games)
- Reproduction and Review
- Blackbox Challenge

# More Games

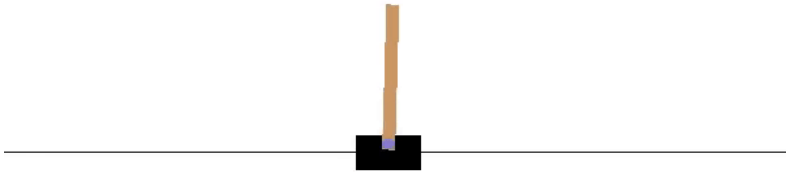
Emulate Turing Machine Functions

Go

Continuous Control in Physics Environments

More Atari: Space Invaders

Classic Control: Cartpole



https://github.com/tambetm/simple_dqn https://gym.openai.com/envs/CartPole-v0

# Advancements

- Double Q-Learn
- Trust Policy Region Optimization
- Actor Critic for continuous action regimes

# Run your own

git clone https://github.com/reidsanders/dl-talk.git

A quick overview of how I run my models:

Small models on my laptop with an nvidia discrete chip
For larger: Amazon gpu instance
Aquired via spot bid (aws cli, set your security group outbound rule to your ip, and recheck regularly if you can't connect it's probably your dynamic ip changing)
Running ubuntu 14.04 (possibly use a ml ami, but many of these are out of date)
Use Cuda or opencl (probably cuda)
Install your libraries (use virtualenv for python please)
Deploy with fabric
When using ssh, use tmux
Train, keeping an eye on validation and training loss
Download results and checkpoints with fabric, save a snapshot in ec2 console or cli

Try: Fomoro (Free trial cloud gpu)

# Resources

Q-Learning Tutorial

Demystifying Deep Reinforcement Learning

Open AI-Gym

Projects:

Deep Q-Learning in Keras

Modular RL - TRPO

DQN Atari in Tensorflow