Deep Q-Networks Playing Atari Games

Chris Barrick, Rajeswari Sivakumar, Layton Hayes

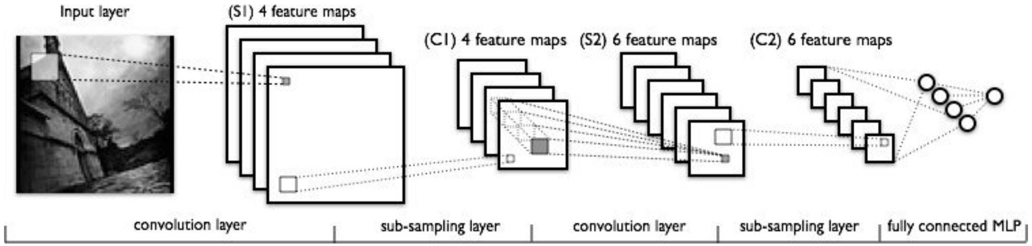
AML Spring 2017

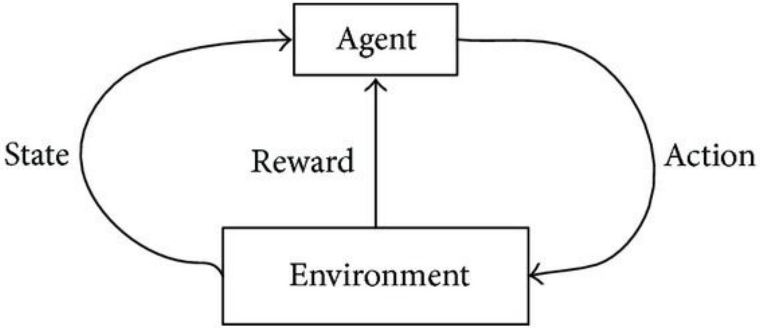
Overview:

- Can modern AI models improve the state of the art in reinforcement learning on Atari 2600 video games?
- Replicate DeepMind’s Deep Q Network (DQN) paper
- Attempt to improve by applying a Differentiable Neural Computer (DNC) to the same problem

Background

- Neural Networks
  - Convolutional neural networks
  - Type of feed forward neural network
- Reinforcement Learning
  - Means of evaluating policy
  - Solving games





Environment

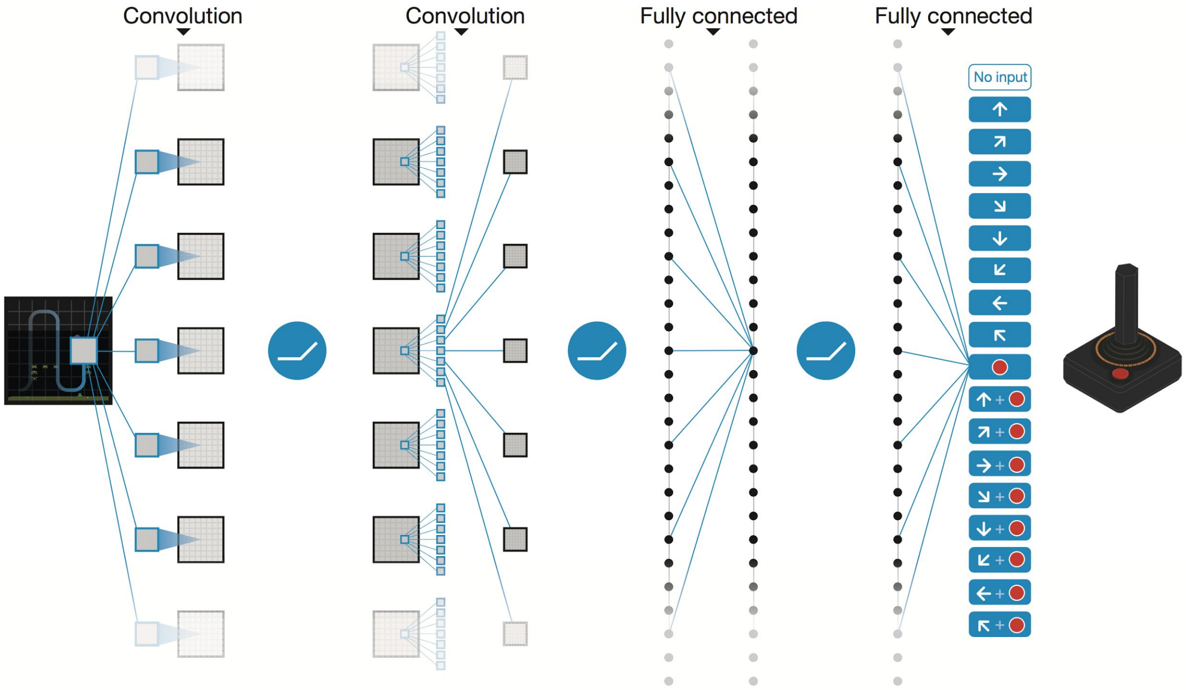
- Simulator: Open AI Atari simulator
  - Download from gym and atari packages
- Models built and trained using Tensorflow
  - Coded in Python 3

Methods: Preprocessing

- Obtain raw frames from Atari simulator
- De-flicker
- Obtain Y channel and use to convert black and white
- Limit to 84 x 84
  - Input layer of convolutional neural net
- Obtain up to next *m* frames (in this case 4)

Methods: Model

- Input layer: 84x 84 convolutional neural net
- 1st hidden layer: 32 8x8 filters with stride of 4, rectifier nonlinearity
- 2nd hidden layer:64 4x4 filters with stride of 2, rectifier nonlinearity
- 3rd hidden layer: fully connected layer with 512 rectifier units
- Output layer: fully connected layer that maps each potential action to an expected output.



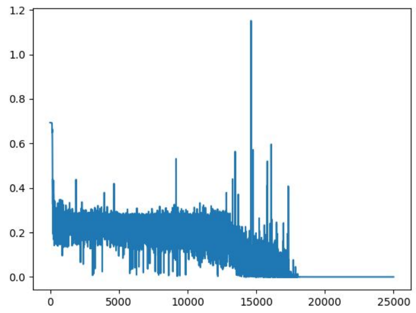
Methods: Training

- Initialize replay memory and action-values
- Initialize state sequence with each episode.
- Select an action randomly or according to Q-function
- Update transition function
- Sample random minibatch transitions from replay memory and use to update expected output.

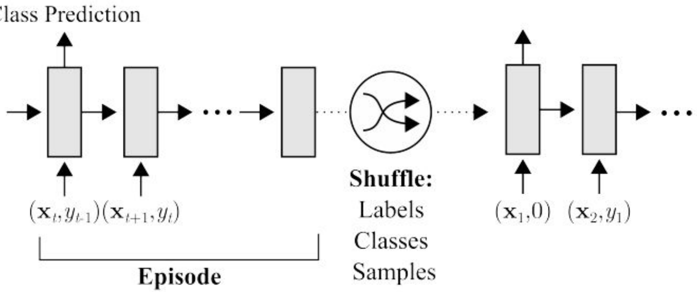
Algorithm 1: deep Q-learning with experience replay.
Initialize replay memory D to capacity N
Initialize action-value function Q with random weights θ
Initialize target action-value function $\hat{Q}$ with weights $\theta^{-} = \theta$
For episode = 1, M **do**
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
  For t = 1,T **do**
    With probability ε select a random action a_t
    otherwise select $a_t = \mathrm{argmax}_a Q(\phi(s_t), a; \theta)$
    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 D
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from D
    Set $y_j = \begin{cases} r_j & \text{if episode terminates at step j+1} \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^{-}) & \text{otherwise} \end{cases}$
    Perform a gradient descent step on $\left(y_j - Q(\phi_j, a_j; \theta)\right)^2$ with respect to the network parameters θ
    Every C steps reset $\hat{Q} = Q$
  End For
End For

Memory Augmented Neural Network

- Enables long term memory preservation
- Enables neural networks to write algorithms for themselves
- Is advancing the state of the art in many complex tasks
- Capable of rapid adaptation -- meta / transfer learning



One-shot Learning



(a) Task setup

MODEL	INSTANCE (% CORRECT) 1ST	2ND	3RD	4TH	5TH	10TH
HUMAN	34.5	57.3	70.1	71.8	81.4	92.4
FEEDFORWARD	24.4	19.6	21.1	19.9	22.8	19.5
LSTM	24.4	49.5	55.3	61.0	63.6	62.5
MANN	**36.4**	**82.8**	**91.0**	**92.6**	**94.9**	**98.1**

Learning to learn videogames

- Each episode consists of a number of trails on a single game
- The game being played is swapped every episode
  - Would need a lot of different games
  - Perhaps alter controls, or change something about the input, to make one game into many
- Reward per episode is total score earned in each trial