ENGR-E 533

Deep Learning Systems

Module 08

Deep Reinforcement Learning

Minje Kim

Department of Intelligent Systems Engineering

Email: minje@indiana.edu

Website: http://minjekim.com

Research Group: http://saige.sice.indiana.edu

Meeting Request: http://doodle.com/minje

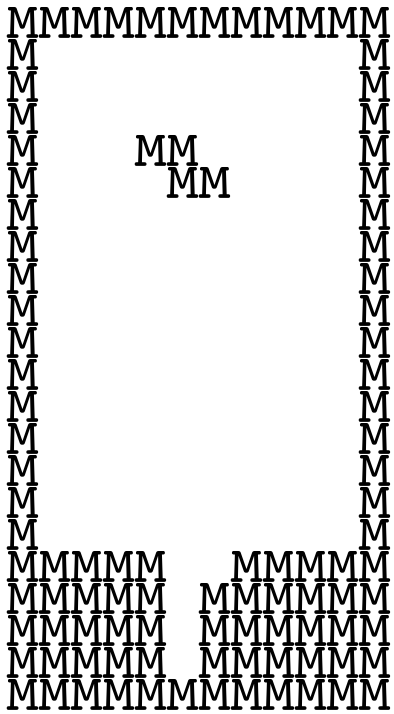
INDIANA UNIVERSITY

SCHOOL OF INFORMATICS, COMPUTING, AND ENGINEERING

AI Plays Games

- Tetris

- Developing the Tetris game isn't that difficult
- But, playing Tetris is a totally different story



AI Plays Games

- DeepMind

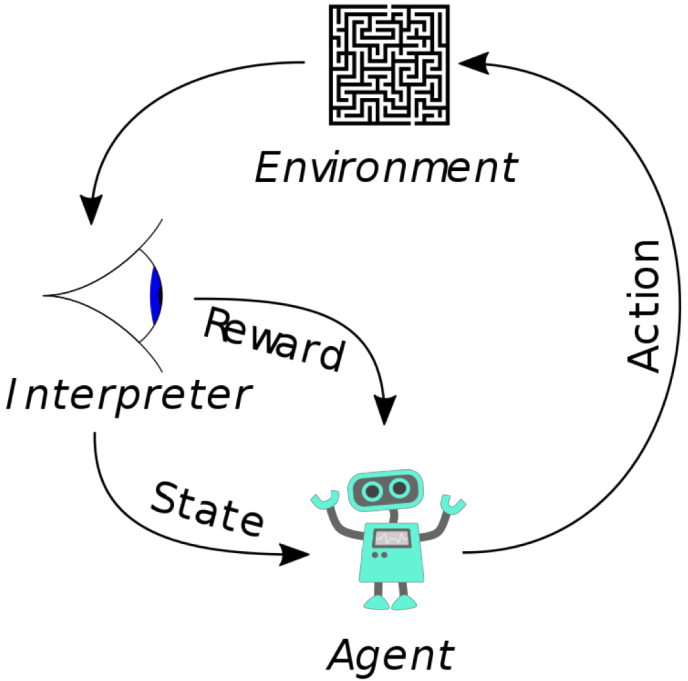
- ATARI: https://www.youtube.com/watch?v=V1eYniJ0Rnk
- Go: https://youtu.be/vFr3K2DORc8?t=3h47m48s
- Starcraft: https://youtu.be/UuhECwm31dM

Baby Reinforcement Learning

- In a supervised learning system we need
  - Input data, target data, parameters to estimate, prediction, cost function
- Learning how to game is a bit different
- Imagine a machine learning system that does this game
  - Prediction
    - Left, right, or stay put
  - Input
    - Position of the ball (and its predicted movement)
    - The entire scene (e.g. all pixels in the screen)
  - Target
    - The action that can eventually lead to the best score
  - Cost function
    - Potential loss of the score by taking that action
- These are difficult to quantify

https://en.wikipedia.org/wiki/Breakout_(video_game)

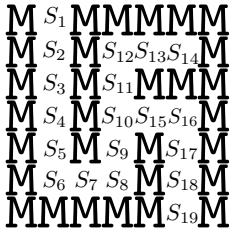
Baby Reinforcement Learning



Q-Learning

- A simpler game

- Which way to go?
  - You cannot come back
  - $S_{10}, U, 0, S_{11}$ versus $S_{10}, R, 0, S_{15}$ (Current State, Action, Reward, Next State)
  - Unless you knew the maze, it's difficult to make the best local decision
  - That's why I give zero rewards
  - Eventually the reward (+1) will be given when we arrive at S_{19}



- Therefore the decision should be made based on the total reward $R = \sum_{i=1}^{N} r^{(i)}$
  - Or the total future reward $R_t = \sum_{i=t}^{N} r^{(i)}$
  - So the game player should be able to calculate the long-term expected reward
  - In this very simple deterministic maze, we can say that
    - $S_{10}, U, 0, S_{11}$ always leads to $R_t = 0$
    - $S_{10}, R, 0, S_{15}$ always leads to $R_t = 1$
  - But sometimes if the game is difficult and decisions are stochastic the future reward is not reliable

Q-Learning

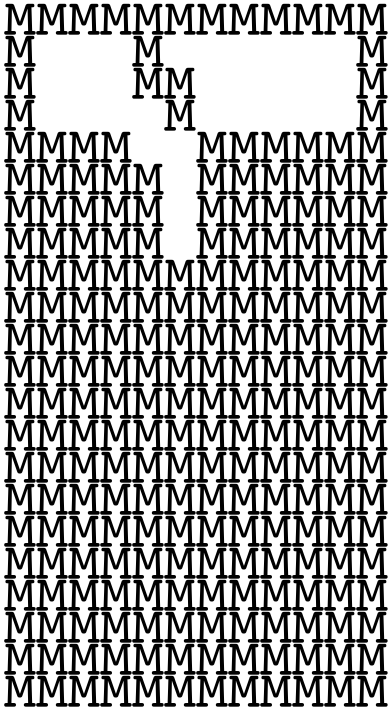
- Discounted future reward

- Be conservative or not

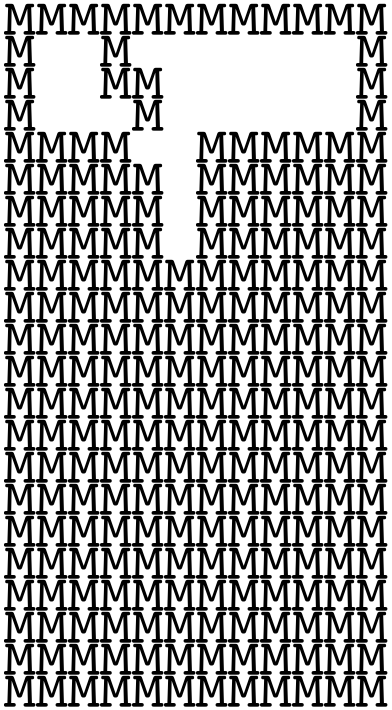
$$R_t = r^{(t)} + \gamma R_{t+1}$$

- $\gamma = 0$ “Carpe Diem”
- $\gamma = 0.99$ weigh more on the future

When γ is small



When γ is large



Q-Learning

- Q-table

- We want to know the Q function $Q(S^{(t)}, A^{(t)}) = \max_{\pi} R_{t+1}$
  - Q function encodes this:
  - "For a given state at a given time t, e.g. $S^{(t)} = S_{10}$, the maximum possible (discounted) future reward by taking the action $A^{(t)} = R$," which is 1, in this particular example
  - This maximum future reward varies depending on the choice of the policy $\pi(S) = \arg\max_{A} Q(S, A)$
  - If know this function, the problem is solved
    - At any given state, we can make the best choice with a holistic view to the problem
- In Q-learning this function could be a table
- How do we learn this magical function?
  - We randomly initialize it
  - Iteration over t
    - Perform action $A^{(t)}$, get the reward $r^{(t)}$, identify new state $S^{(t+1)}$
    - Update the Q-value: $Q(S^{(t)}, A^{(t)}) = (1-\rho) \cdot Q(S^{(t)}, A^{(t)}) + \rho\big(r^{(t)} + \gamma \max_{A^{(t+1)}} Q(S^{(t+1)}, A^{(t+1)})\big)$

Bellman equation

- Originally Q-values didn't know each other but the updates make them interact

The ideal Q-table

	U	D	L	R
S_1	0	1	0	0
S_2	0	1	0	0
…	…	…	…	…
S_{10}	0	0	0	1
…	…	…	…	…

Deep Q Networks

- Let's learn Q-function

- The table version of Q-function is not the best choice
  - In the Breakout game, how many states are there?
  - 84X84 pixels; each can have 256 intensity values➔256^{84X84}
  - The table will be with too many rows
- Trying to keep all possible pairs is the least efficient way to learn the mapping
- Neural network can generalize to do this job

$$\mathcal{L} = \Big(\underbrace{r^{(t)} + \gamma \underbrace{\max_{A^{(t+1)}} Q\big(S^{(t+1)}, A^{(t+1)}\big)}_{\text{Another DQN feedforward, but with the next state as input}}}_{\text{Target}} - \underbrace{Q\big(S^{(t)}, A^{(t)}\big)}_{\text{DQN feedforward predicts this}} \Big)^2$$

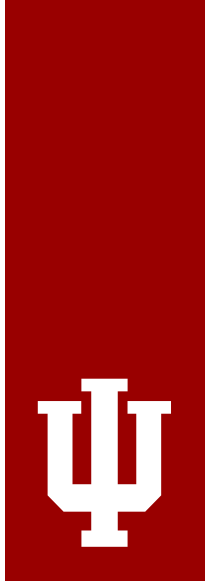
  - For Atari games, CNN just works

Deep Q Networks

- Experience replay
  - Randomly sample from previous plays
- ε-greedy exploration
  - Add some noise in choosing the best action

Reading

- Sutton and Barto, “Reinforcement Learning: An Introduction”
  - http://www.incompleteideas.net/book/the-book-2nd.html
- Mnih, Volodymyr, et al. “Playing Atari with Deep Reinforcement Learning”
  - https://arxiv.org/abs/1312.5602
- Mnih, Volodymyr, et al. "Human-level control through deep reinforcement learning." *Nature* 518.7540 (2015): 529.
  - https://storage.googleapis.com/deepmind-media/dqn/DQNNaturePaper.pdf
- https://medium.com/emergent-future/simple-reinforcement-learning-with-tensorflow-part-0-q-learning-with-tables-and-neural-networks-d195264329d0



Thank You!