DQN Tricks

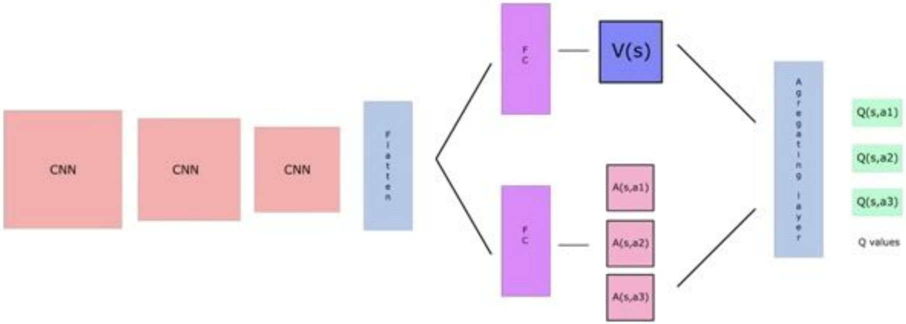
- Experience Replay
  - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
- Fixed Target Network
  - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000 steps increases stability of training process.

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p \boxed{Q(s_{t+1}, p)} - Q(s_t, a) \right]$$

target Q function in the red rectangular is fixed

Replay	○	○	×	×
Target	○	×	○	×
Breakout	**316.8**	240.7	10.2	3.2
River Raid	**7446.6**	4102.8	2867.7	1453.0
Seaquest	**2894.4**	822.6	1003.0	275.8
Space Invaders	**1088.9**	826.3	373.2	302.0

Dueling DQN (DDQN)



- Decompose **Q(s,a)**

$$Q(s,a) = A(s,a) + V(s)$$

  - **V(s)**: the value of being at that state
  - **A(s,a)**: the advantage of taking action ***a*** in state ***s*** versus all other possible actions at that state
- Use two streams:
  - one that estimates the state value V(s)
  - one that estimates the advantage for each action A(s,a)
- Useful for states where action choice does not affect Q(s,a)

Advantage Actor-Critic (A2C)

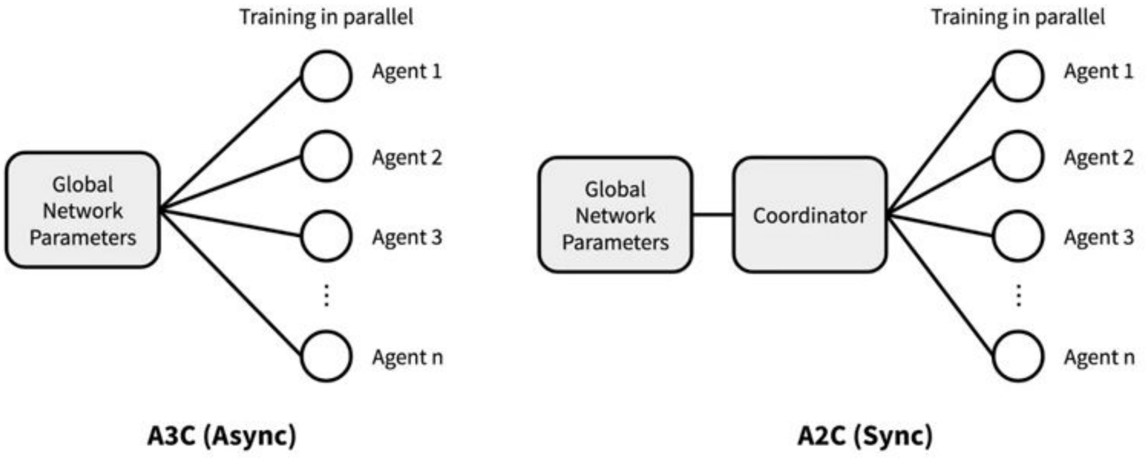
- Combine DQN (value-based) and REINFORCE (policy-based)
- Two neural networks (Actor and Critic):
  - **Actor** is policy-based: Samples the action from a policy
  - **Critic** is value-based: Measures how good the chosen action is

Policy Update: $$\Delta\theta = \alpha * \nabla_\theta * (log\, \pi(S_t, A_t, \theta)) * \cancel{R(t)}$$

New update: $$\Delta\theta = \alpha * \nabla_\theta * (log\, \pi(S_t, A_t, \theta)) * \boxed{Q(S_t, A_t)}$$

- Update at each time step - temporal difference (TD) learning

Asynchronous Advantage Actor-Critic (A3C)



- Both use parallelism in training
- A2C syncs up for global parameter update and then start each iteration with the same policy

Actor-Critic Algorithm

Initialize policy parameters θ, critic parameters ϕ

For iteration=1, 2 … **do**

Sample m trajectories under the current policy

$\Delta\theta \leftarrow 0$

For i=1, …, m **do**

For t=1, ... , T **do**

$$A_t = \sum_{t' \geq t} \gamma^{t'-t} r_t^i - V_\phi(s_t^i)$$

$$\Delta\theta \leftarrow \Delta\theta + A_t \nabla_\theta \log(a_t^i | s_t^i)$$

$$\Delta\phi \leftarrow \sum_i \sum_t \nabla_\phi ||A_t^i||^2$$

$\theta \leftarrow \alpha\Delta\theta$

$\phi \leftarrow \beta\Delta\phi$

End for

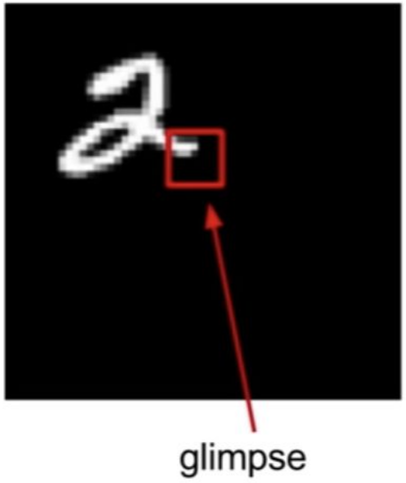
REINFORCE in action: Recurrent Attention Model (RAM)

Objective: Image Classification

Take a sequence of “glimpses” selectively focusing on regions of the image, to predict class

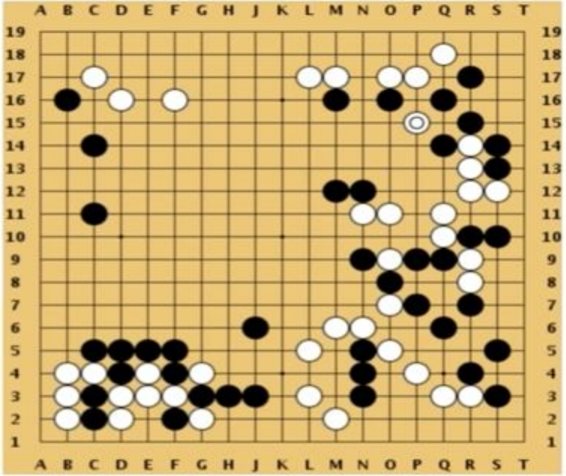
- Inspiration from human perception and eye movements
- Saves computational resources => scalability
- Able to ignore clutter / irrelevant parts of image

State: Glimpses seen so far
Action: (x,y) coordinates (center of glimpse) of where to look next in image
Reward: 1 at the final timestep if image correctly classified, 0 otherwise



Glimpsing is a non-differentiable operation => learn policy for how to take glimpse actions using REINFORCE
Given state of glimpses seen so far, use RNN to model the state and output next action

More policy gradients: AlphaGo



Overview:

- Mix of supervised learning and reinforcement learning
- Mix of old methods (Monte Carlo Tree Search) and recent ones (deep RL)

How to beat the Go world champion:

- Featurize the board (stone color, move legality, bias, …)
- Initialize policy network with supervised training from professional go games, then continue training using policy gradient (play against itself from random previous iterations, +1 / -1 reward for winning / losing)
- Also learn value network (critic)
- Finally, combine combine policy and value networks in a Monte Carlo Tree Search algorithm to select actions by lookahead search

[Silver et al., Nature 2016

Deepmind Atari Agent57

https://deepmind.com/blog/article/Agent57-Outperforming-the-human-Atari-benchmark?fbclid=IwAR3WCnx10wQMo1uu35pcmiQN1MFQrrZnhGVkOkOsidCpEqjqkoTSawcW-Ao