Deep Learning papers

Reinforcement Learning & Deep Learning

Outline

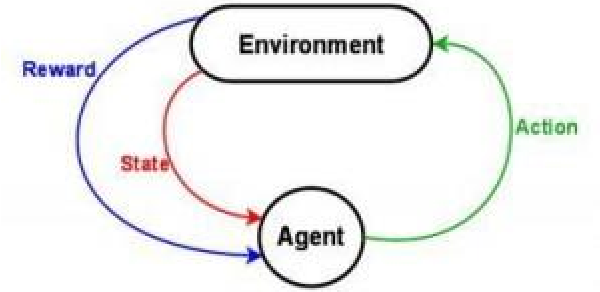
- Reinforcement Learning
- Examples of RL + DL:
    - DeepMind Atari
    - DeepMind AlphaGo
    - ...

Reinforcement learning

Different paradigm, != supervised learning

Learn from experience, by iteratively trying:

- Observe current state
  - Markovian: state should include all that's needed
- Take action
  - Use learned policy, but sometimes explore randomly (exploration-exploitation dilemma)
- Get (delayed) reward / punishment
  - Rewards are delayed, noisy, sparse
  - Optimal policy should optimise future reward: take best action



Makes a lot of sense if there's an environment to interact with:
directed learning, machine can do (simple) experiments, using knowledge so far

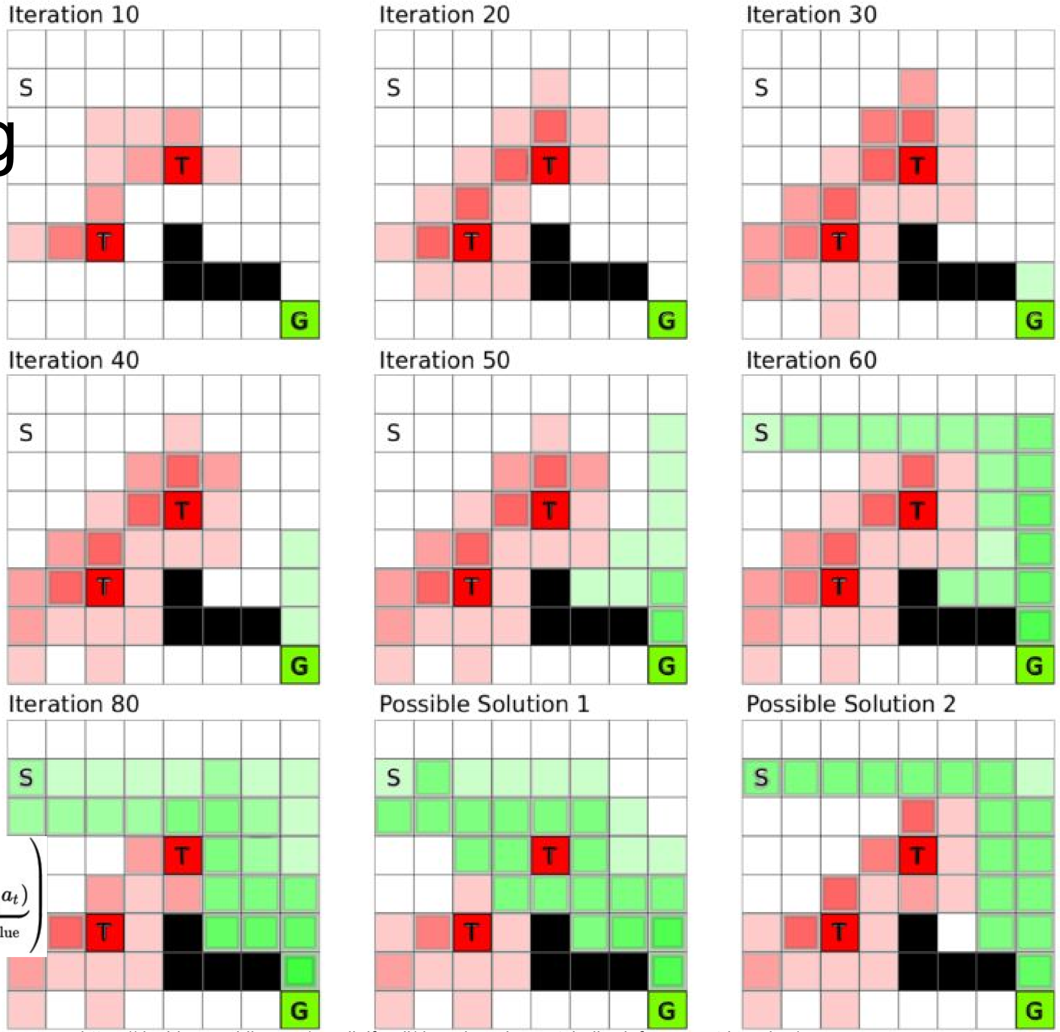
Reinforcement learning

We want to learn the expected future reward (Q) for every state.

The policy function simply returns the best action given a state.

During training, the immediate rewards (r) 'smooth out' over neighbouring states.

$$Q(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha_t}_{\text{learning rate}} \cdot \left(\overbrace{\underbrace{r_{t+1}}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}^{\text{learned value}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \right)$$

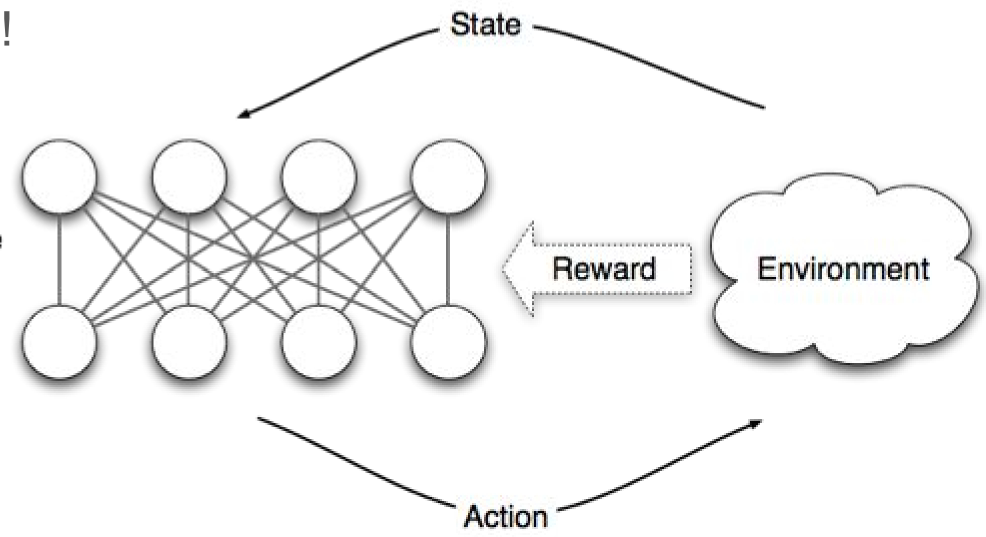


source: https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-reinforcement-learning/

RL + NN: "Deep Reinforcement Learning"

Policies are (usually) function approximations: `max_a[Q(s,a) -> R]`

NNs are good function approximators!
Generalise well over high-D states.



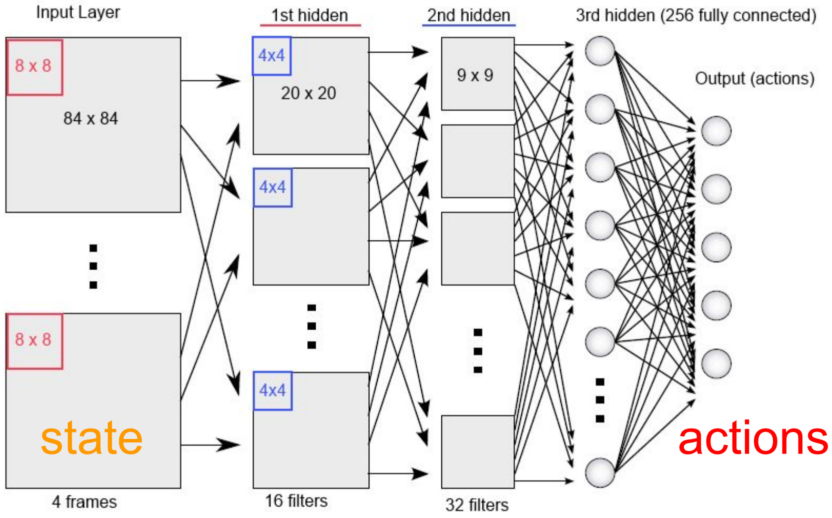
But:

- Reward is delayed, sparse and noisy
- Changing actions change the distribution of states (model influences dataset)
- States come in sequences and are correlated

Example 1: DeepMind Atari games

Play simple games using raw pixels

- State: [x(1), a(1), x(2), a(2), …, a(t-1), x(t)]
  - Historic frame sequence (in practice: 4) is part of state
- Reward: *change* in gameScore
- Policy:

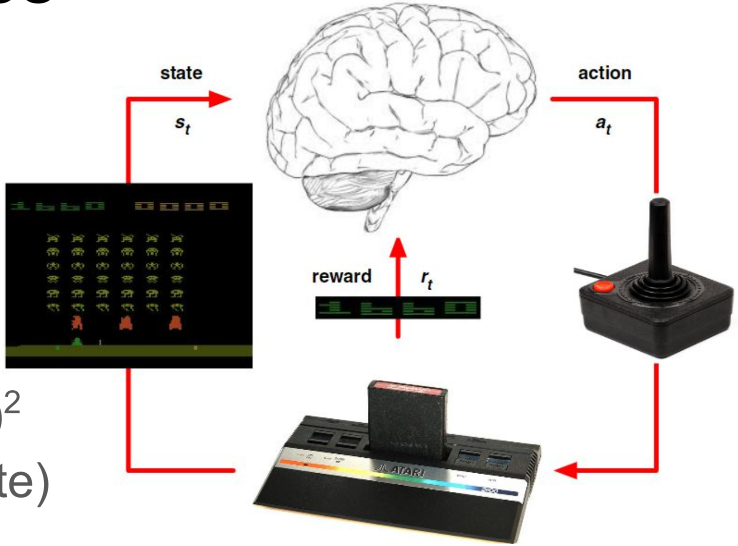


actions → Q(s, a)

Example 1: DeepMind Atari games

Deep Q-Learning

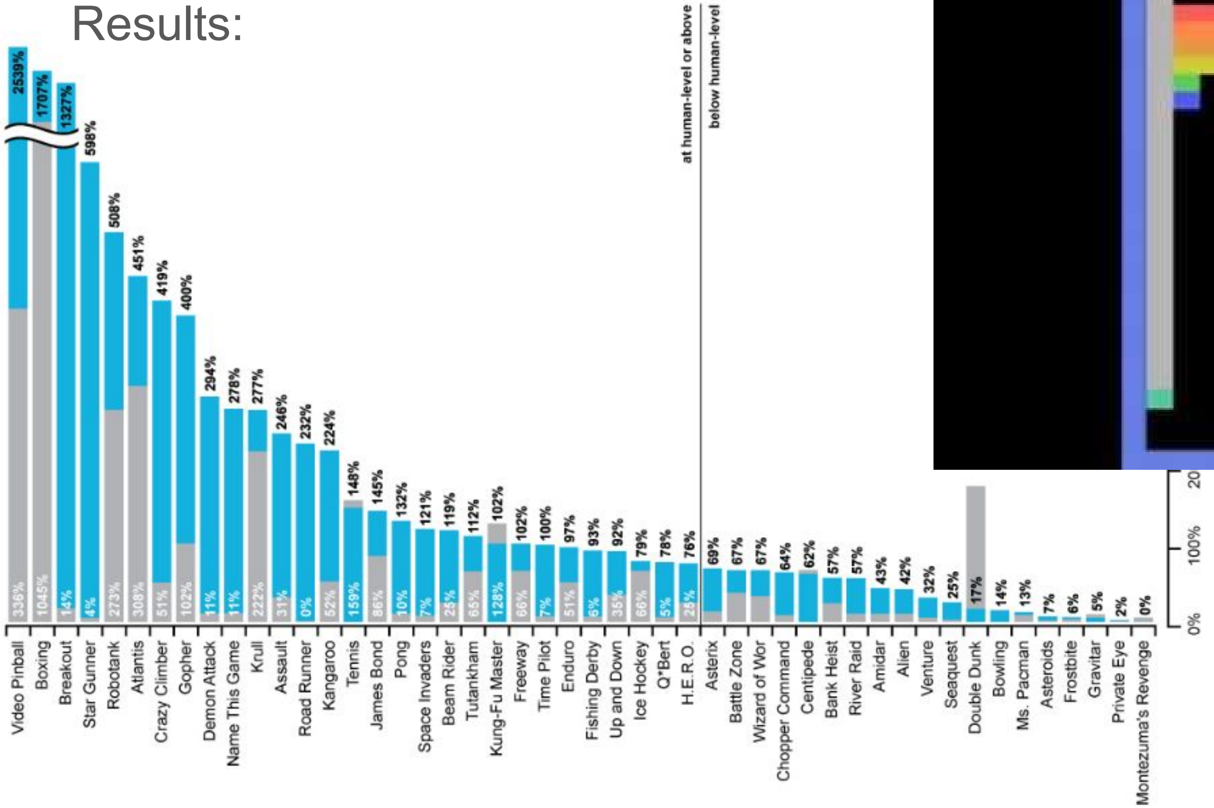
repeat:
  [RL] action + observe s -> save in replay memory
  [SL] sample mini-batch from replay memory;
    train NN with loss: $([r + \gamma \max_{a'} Q(s', a')] - Q(s, a))^2$
    (difference between current and new Q estimate)

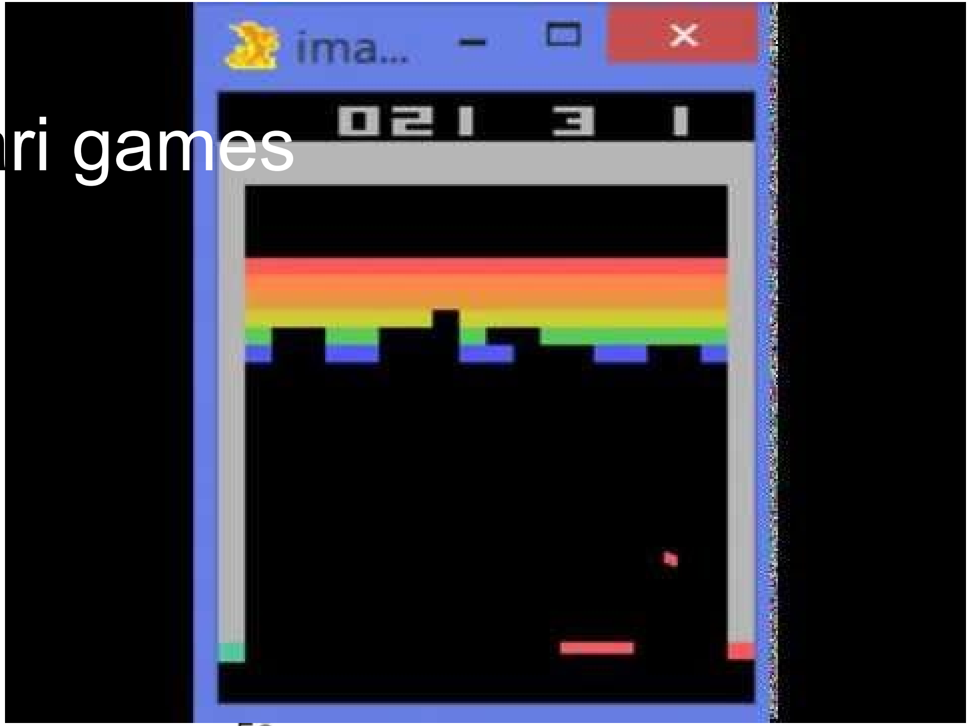


Uses the same network architecture for all games;
no game-specific knowledge used anywhere

Example 1: DeepMind Atari games

Results:

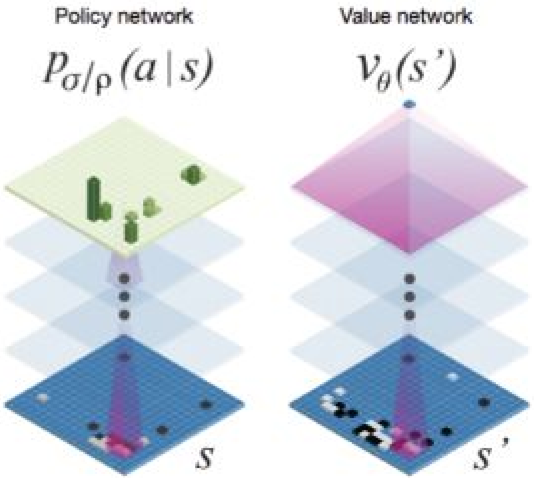


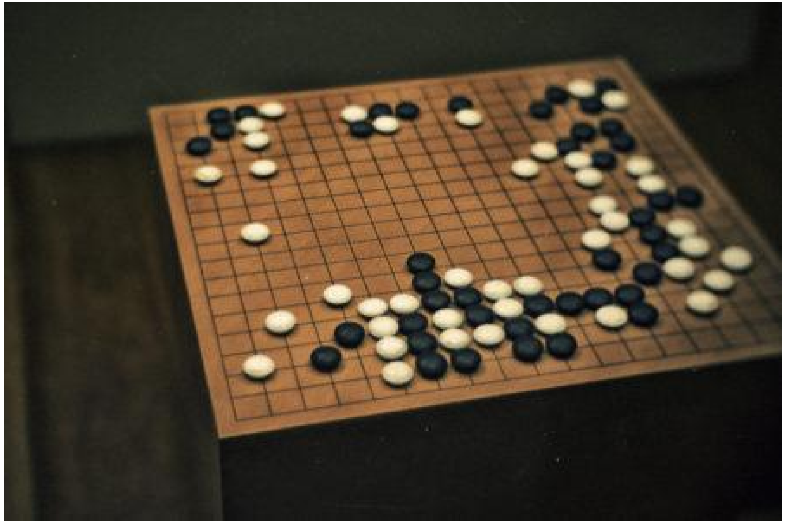


Example 2: DeepMind AlphaGo

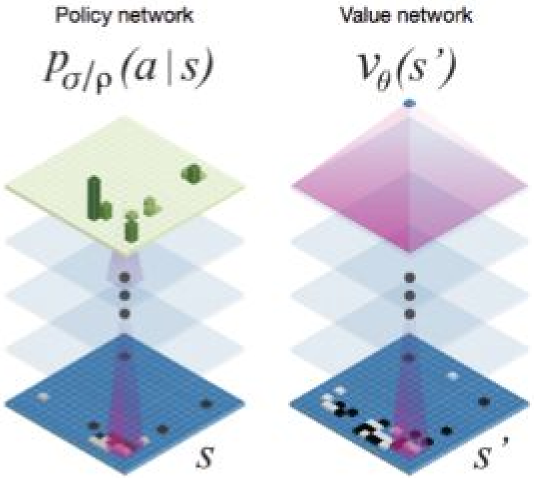
Mastering a very hard board game

Challenge: no intermediate score!
Solution: second NN for value estimation of board state





Example 2: DeepMind AlphaGo



Traditional approach for board games:
tree search (breadth^{depth} positions to evaluate)
much larger than for chess!

Training steps:

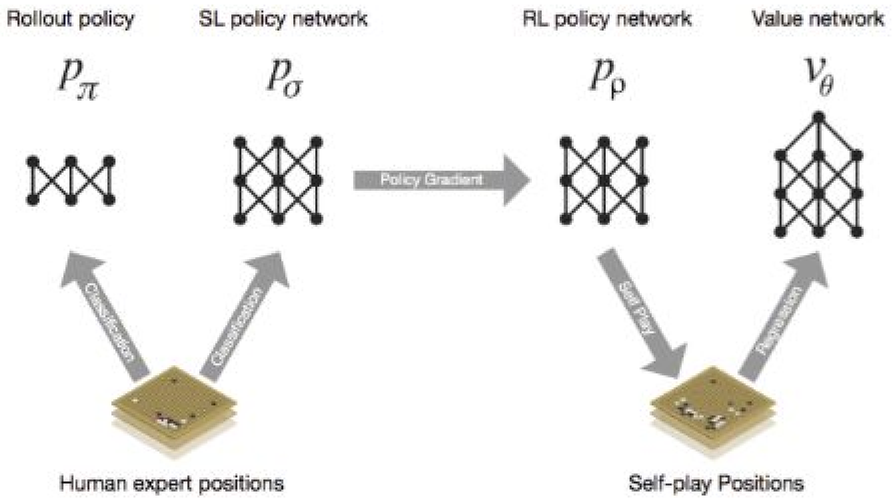
1. [SL] train policy network (CNN)
   on games by expert human players

   actually two versions: large (13L; 57%) NN + fast NN (24%)

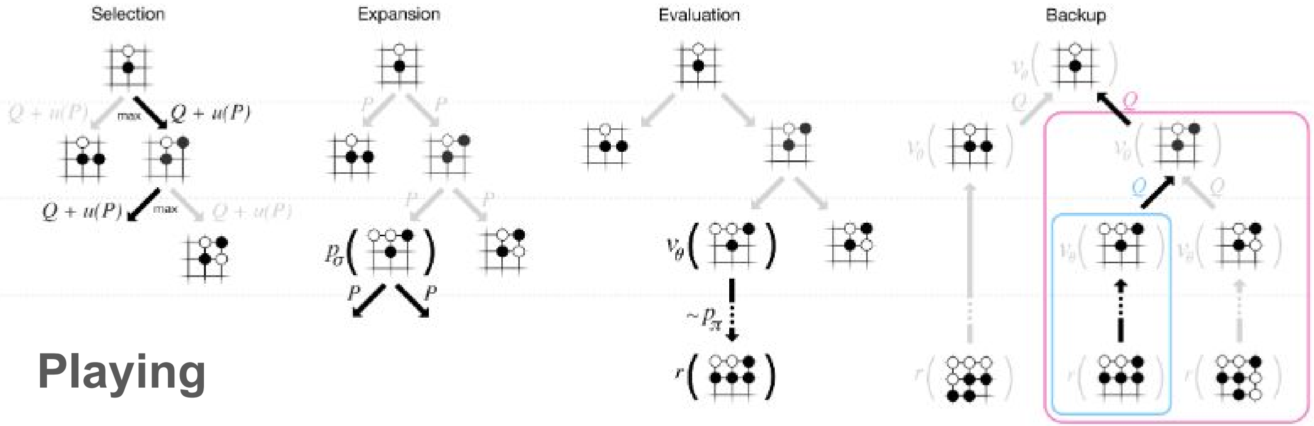
2. [RL] improve policy network using
   self-play against earlier version

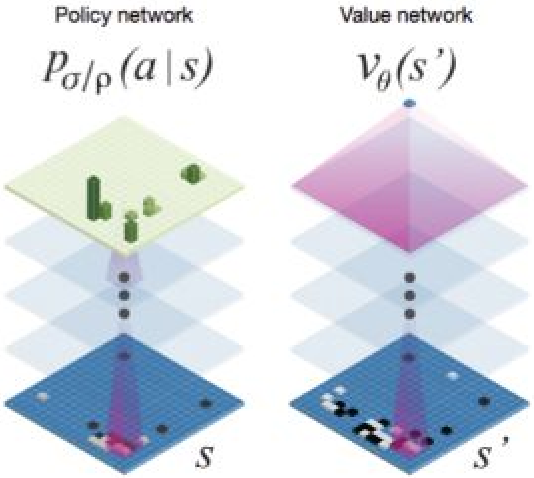
   reward = 0 until won (1) or lost (-1)

3. train value network (chance of winning)
   on state,result pairs (30M, each randomly selected out of new self-play game)



Example 2: DeepMind AlphaGo





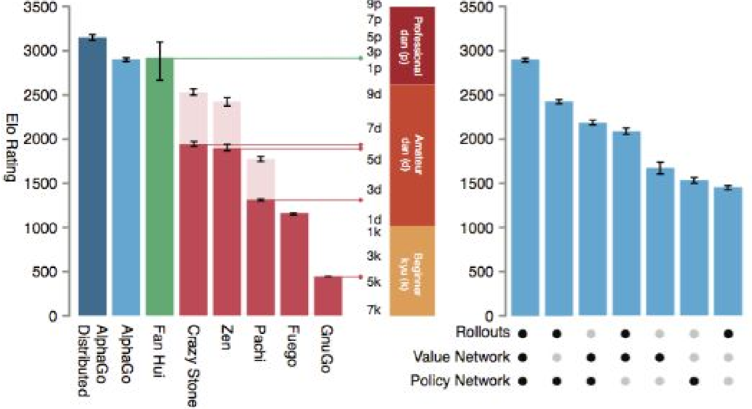
Playing

Monte Carlo Tree Search, combining all the pieces:

1. from current state, expand some parts of the tree using value network and policy network: follow a = argmax(Q(s,a) + p(a|s) / (1+N(s,a)); that is, max[*average value leaf nodes of subtree* + *policy (discounted by exploration #)*]; on leaf node, expand one state and evaluate: the value of leaf nodes is $(1-\lambda)*v(s_L) + \lambda*z_L$, where z_L is the final result from a rollout (full game, using fast policy) starting from L
2. when time is up: take action that was *visited* most

Example 2: DeepMind AlphaGo

Results: 4-1 for AlphaGo



Readings

- Introductory blog posts:
  https://www.nervanasys.com/demystifying-deep-reinforcement-learning/
  https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-reinforcement-learning/
  https://karpathy.github.io/2016/05/31/rl/
- Tutorial by David Silver (video):
  http://techtalks.tv/talks/deep-reinforcement-learning/62360/