Deep
Reinforcement Learning

romeo kienzler

intelligent behaviour in complex dynamic environments

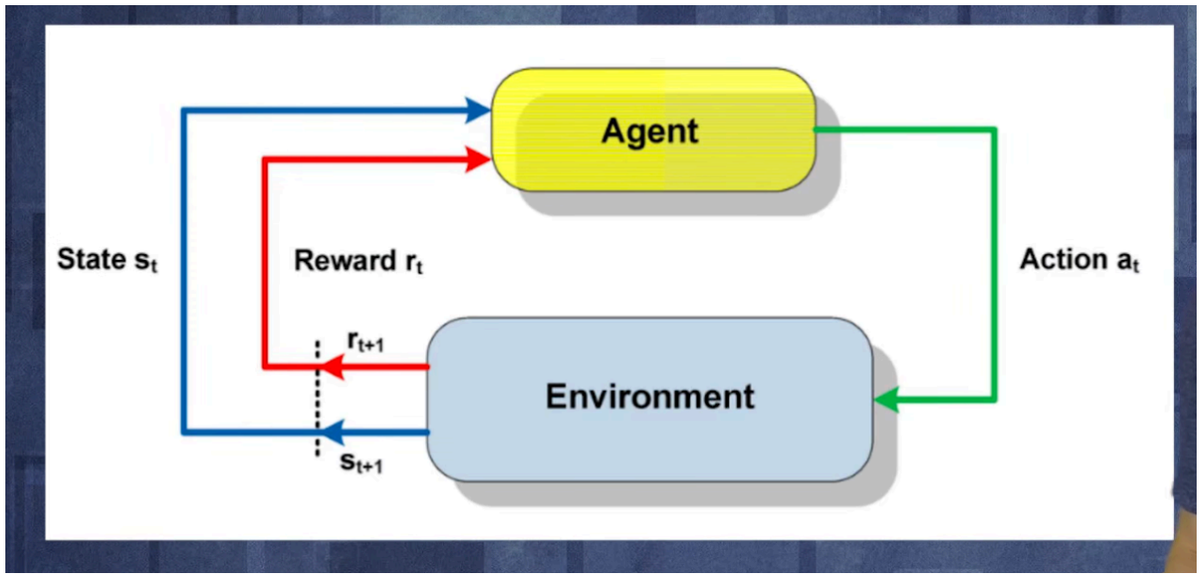
currently - supervised learning is used in most ML tasks - DeepRL can be seen as an unsupervised learning method

Outline

- 8:30-12:00: 2 Theory Modules + 2 Exercises
  - Reinforcement Learning Overview
  - Introduction to AlphaGo Zero
  - Exercises on Policy Gradient Methods using the OpenAI Gym
- 13:00-16:00: 1 short Theory Module + microHackathon
  - Understanding influence on human priors
  - microHackahon: Implement a full Policy Gradient DeepRL agent for OpenAI Gym

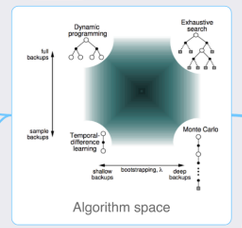
Reinforcement learning involves an agent, a set of *states* , and a set of *actions* per state. By performing an action $a \in A$, the agent transitions from state to state. Executing an action in a specific state provides the agent with a *reward* (a numerical score).

The agent is trained to choose the optimal action in a given state s in order to maximize reward r.


Agent
State s_t
Reward r_t
Action a_t
r_{t+1}
Environment
s_{t+1}

latest achievements

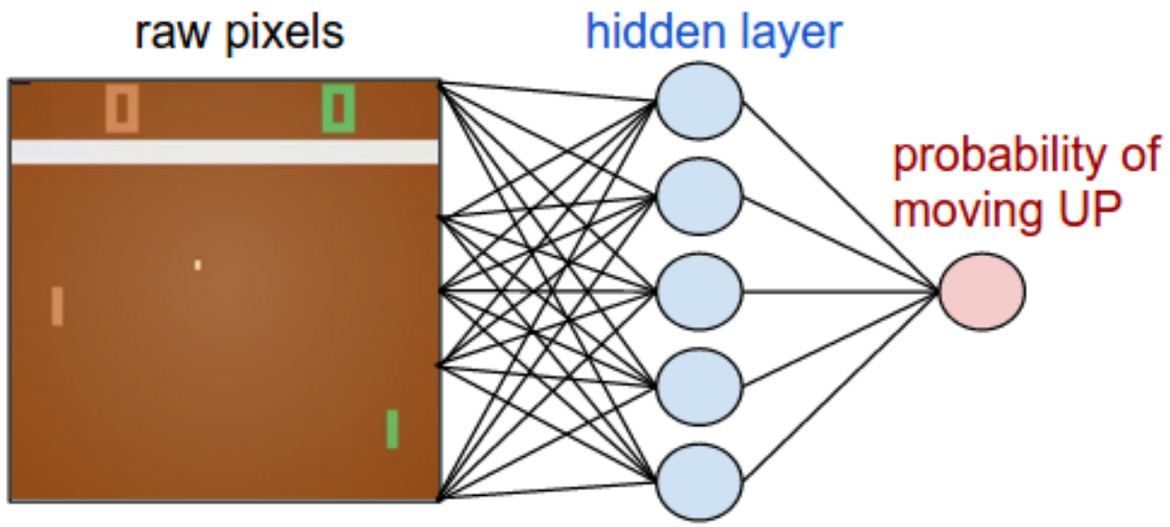
- March 2016: AlphaGo Lee beats 18-time world champion Lee Sedol in the game of GO October 2017: AlphaGo Zero beats AlphaGo Lee 100:0
- July 2018: Dactyl, trained entirely in simulation but applied to real-world physics
- 2018: Show and Tell, a agent capable of following social laws
- 2018: IBM Reinforced Reader-Ranker for Open-Domain Question Answering (R^3)
- December 2018: AlphaFold won the 13th Critical Assessment of Techniques for Protein Structure Prediction (CASP)
- April 2019: OpenAI Five wins back-to-back games versus Dota 2 world champions OG at Finals, becoming the first AI to beat the world champions in an esports game (PPO)
- 2019: DQ Scheduler, a Deep Reinforcement Learning Based Controller Synchronization in Distributed SDN
- 2019: Dialog-Based Interactive Image Retrieval
- October 2019: Solving Rubik's Cube with a Robot Hand

- Methods
  - Value based
    - Implementations
      - Q-Learning
      - Sarsa
      - Dyna
      - DDPG
    - 

    - Full backups vs sample backups (tree width)
    - Shallow backups vs deep backups (tree depth)
    - Advantages
      - More data efficient than policy based RL
    - Disadvantages
      - High dimensional / continuous action spaces
      - Stochastic policies
  - Imitation learning
    - DAGGER
    - GAIL
    - DeepMimic
    - VAIL
    - MetaMimic
  - Policy based
    - Policy gradient
      - REINFORCE (Vanilla)
      - TRPO
      - PPO
      - A2C
      - A3C
      - Generalized Advantage Estimation (GAE)
      - TRPO Kronecker-factored (ACKTR)
      - ACER
      - SAC
    - Deterministic Policy Gradients
      - DPG
      - DDPG
      - TD3
    - Evolutionary
      - CEM
      - CMA-ES
      - PEPG
      - NES
      - ES
    - Policy search
      - PILCO
      - REPS
  - Model-based
    - Pure planning
    - Expert Iteration
    - Data Augmentation
    - Embedding planning loops into policies
    - AIXI

http://louiskirsch.com

pong from pixels

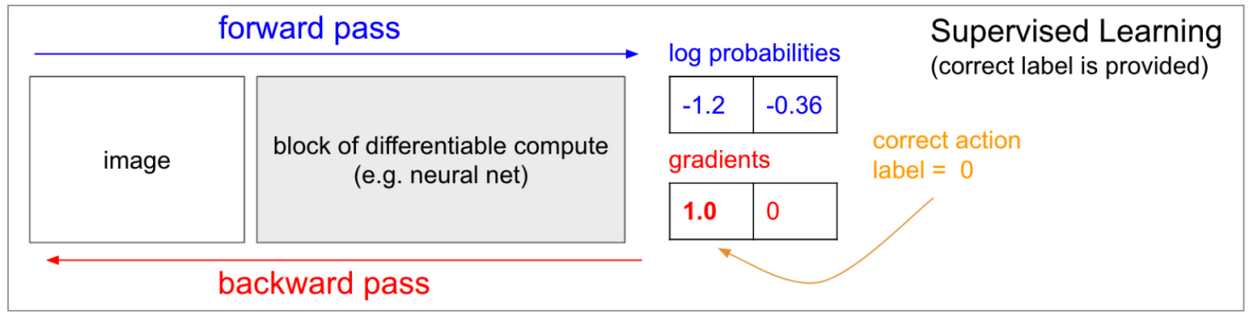


Source: http://karpathy.github.io/2016/05/31/rl/

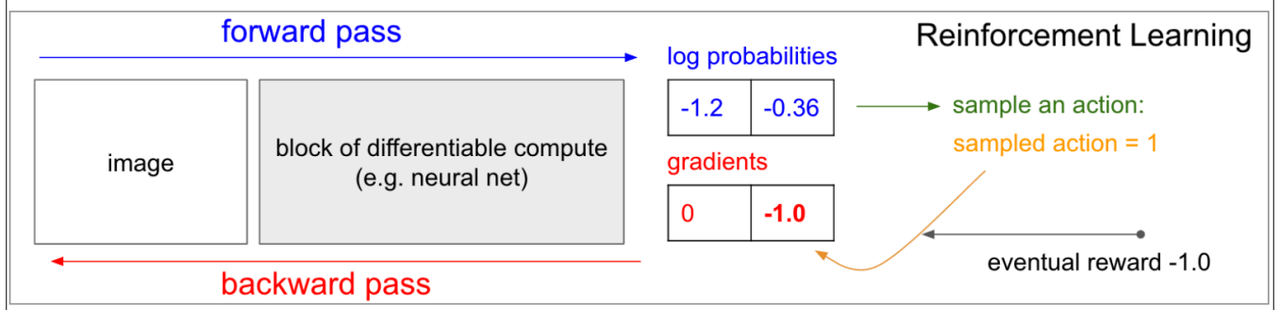
In a supervised setting, training data are image frames plus a binary label for moving up or down

Can’t get better than humans since data was generated by humans

The neural network between input (state) and output (action) is called policy network



Source: http://karpathy.github.io/2016/05/31/rl/

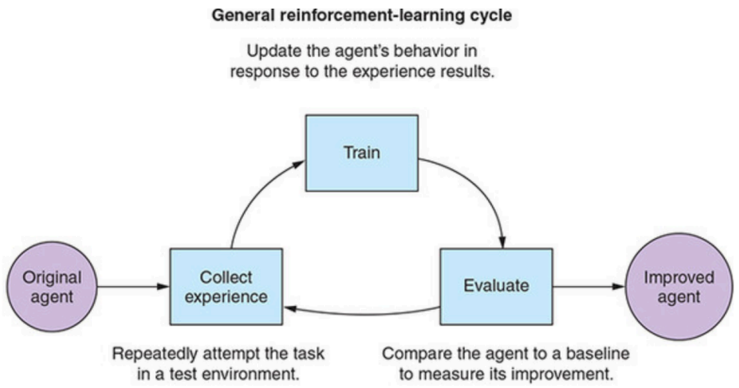


Source: http://karpathy.github.io/2016/05/31/rl/

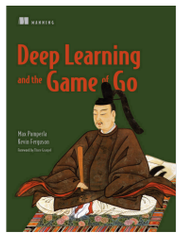
Policy Network assigns probability for each action given a state. Action is chosen. Reward is evaluated. PN is updated accordingly. Positive reward further encourages that action, negative reward further discourages that action taken at a given state.

Note: In pong, you get positive reward if own score increases negative reward if own score decreases

Initially most of the games the agent will loose but some random series of actions will result in a positive reward



source:



In a DeepRL setting, agent collects experience by interacting with the environment through actions and the resulting state and reward. The policy network is trained using the reward information via policy gradients for improvement. The term prediction error in supervised learning is equivalent to the reward. In both cases, a gradient is derived to update weights in a neural network

Exercise

https://github.com/romeokienzler/DeepRL => DeepRL.ipynb

epsilon greedy exploration

Given policy vector π, with probability 1- ϵ

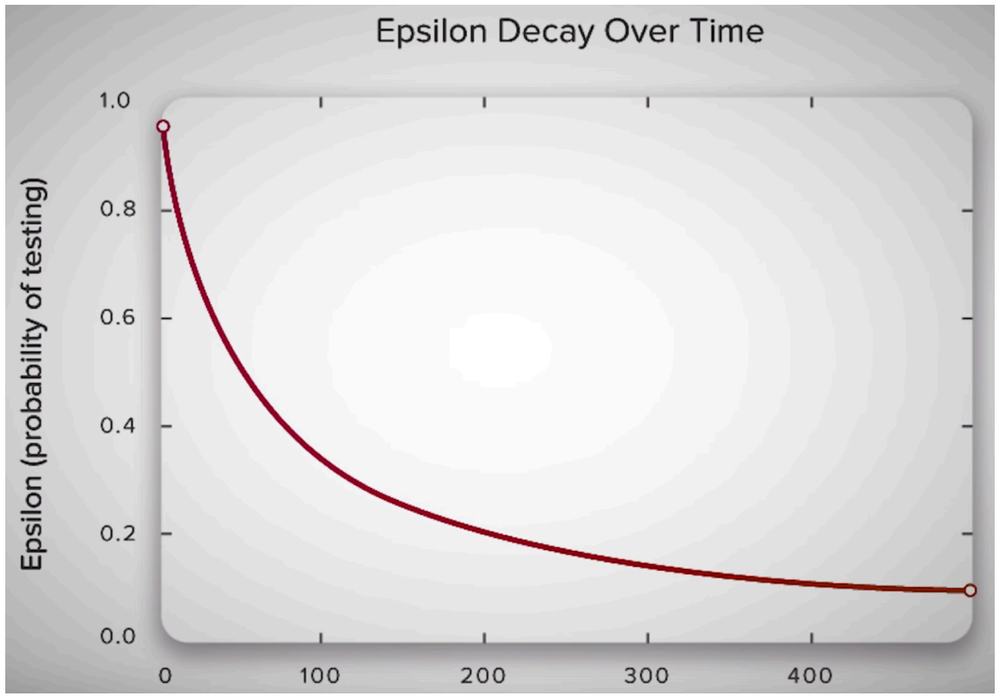
choose best action, with probability ϵ

choose any other random action

$$\pi = (0.2,0.3,0.1,0.2,0.2)$$

adding randomness increases exploration but reduces exploitation

exploration vs. exploitation problem



usually, epsilon decreases over training time

credit assignment problem

Problem: Sparse Extrinsic Rewards

Sparse Rewards: Many actions need to be chosen until feedback in form of a reward is provided

DeepRL algorithms are very sample inefficient -> need millions of training examples, therefore some settings are impossible to learn because random actions never lead to a reward (and gradient / weight update) vs. one label per frame in cnn here only one label after millions of frames if at all (or only negative labels/rewards and never a positive label/reward to learn from)

credit assignment problem

Solution 1: Dense Intrinsic Reward Shaping

reward shaping is the process of manually designing a reward function

downside: need to craft a function for each environment

downside: the alignment problem-> reward shaping leads to agent learning only to make your reward function happy

downside: constraining pn to the behaviour of humans (bias)

credit assignment problem

Solution 2: auxiliary reward signals

add extrinsic rewards like

- maximal change of pixel intensities (visual change score)
- reward prediction of next action based on some historic actions
- predict total future reward from any environment state

credit assignment problem

Solution 3: intrinsic curiosity

usually epsilon greedy exploration is used to choose between exploration and exploitation

- latent representation is learned from input (e.g. frame)
- future latent representation is predicted
- if prediction error is low, agent gets bored and is encouraged to explore unseen regions
- (in other words, instead of slowly decrease epsilon, adjust epsilon based on how unseen a particular state is)

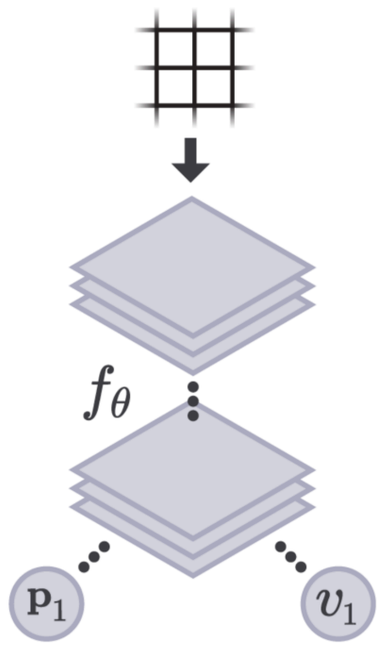
There are many more of those solutions and this is an active area of research

Understanding AlphaGo Zero

- no hand crafted features
- no hand crafted reward
- no human training data
- alpha go zero learns entirely from scratch (self play only)

note: go is a perfect information game and we have a perfect simulator which both we don’t have in the real world

AlphaGo Zero Architecture



board representation (19x19x17)

residual convolutional network

fully connected feature layers

p => policy vector, v => value (winning probability)

two feature maps for white and black stones 19x19 => 19x19x2

7 past plus 1 current =8 => 19x19x16(2*8) works as some kind of attention mechanism when observing what the opponent was playing

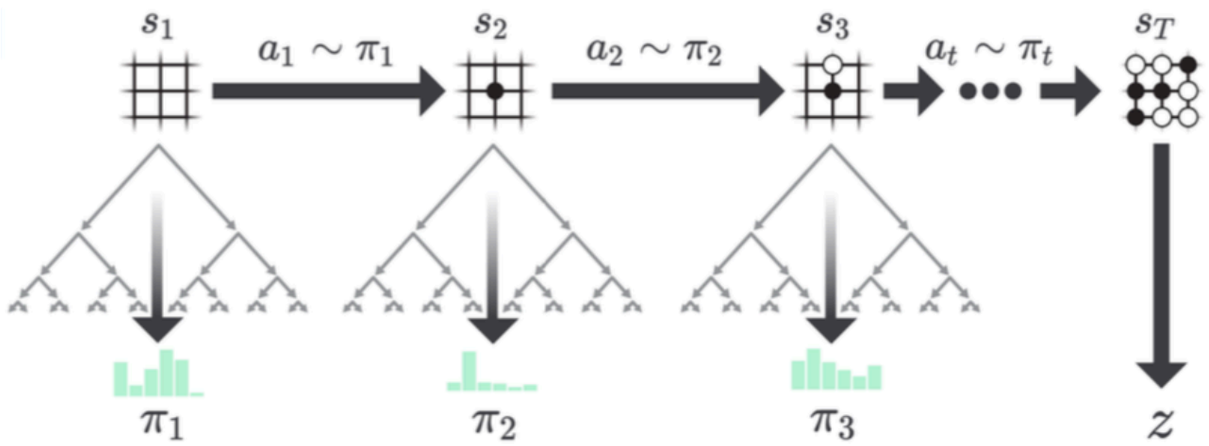
plus one bit who's turn it is replicated on all feature maps => 19x19x17

resnet->feature vector->split->value function on own confidence for winning/policy network or probability distribution on all possible next moves

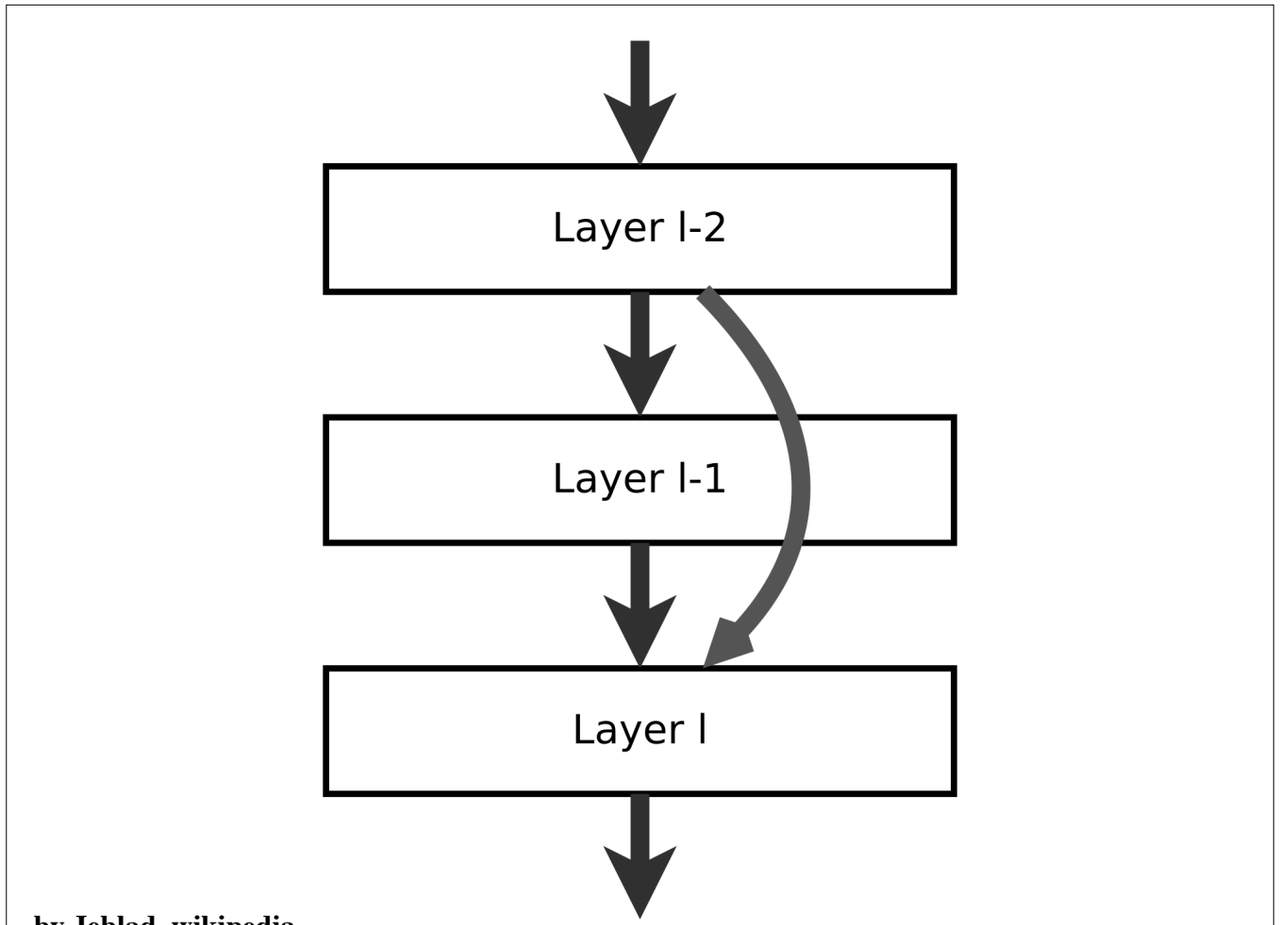
note: the value network produces an auxiliary reward signal

combined policy and value network

AlphaGo Zero training

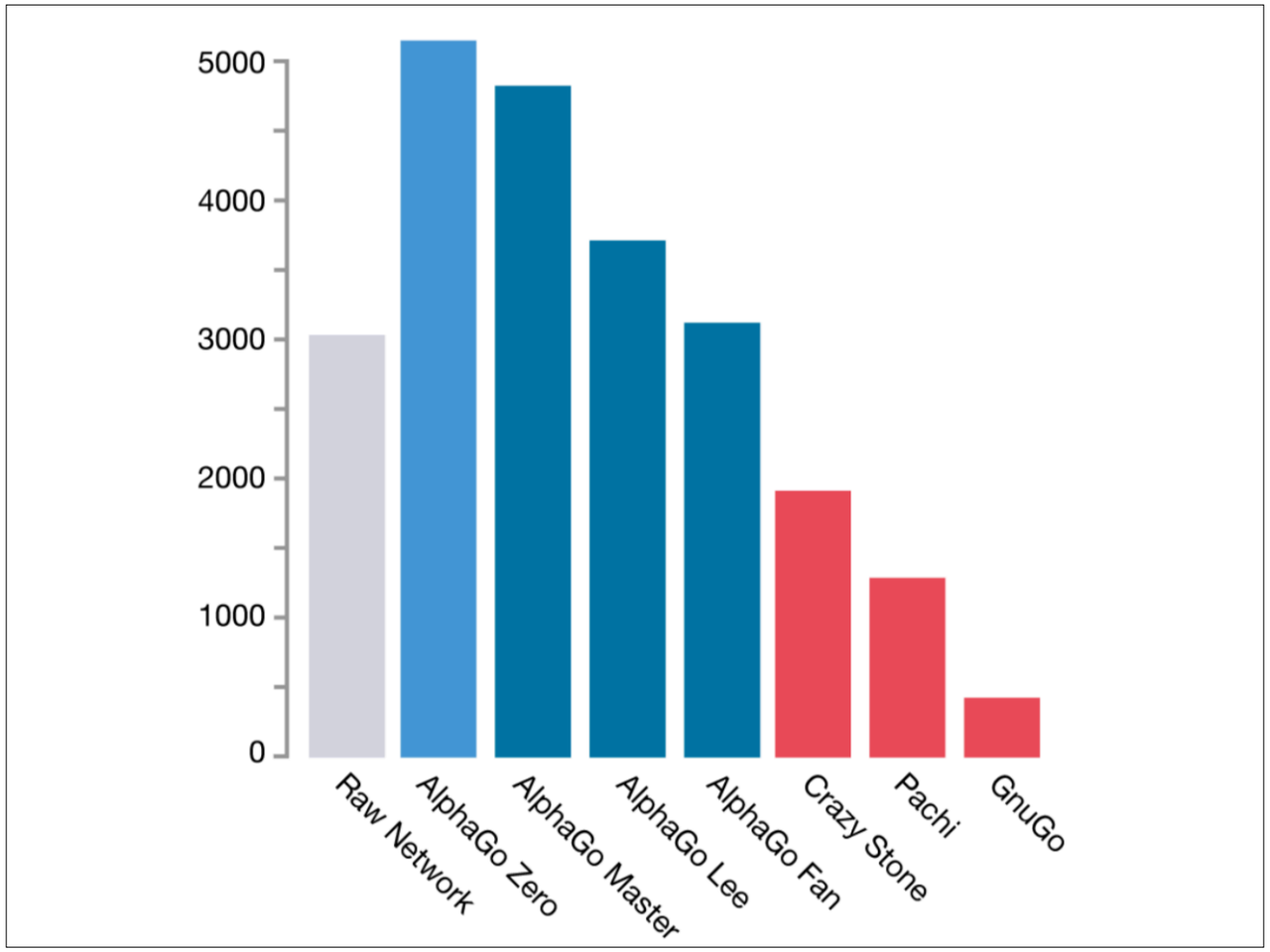


- uses policy network to decide which action to take
- uses monte carlo tree search (MCTS) to randomly explore other branches using epsilon greedy exploration
- each board state starts 1600 parallel evaluations (because of epsilon greedy exploration, the simulation takes other branches later, down the tree)
- value function is used to estimate winning probability, low score branches aren't further evaluated (early stopping)
- after leaf of tree is reached (either through winning or loosing game) policy gradients are calculated and used to update the neural network
- value net gets trained simultaneously to train an estimator for winning probability of a particular board state

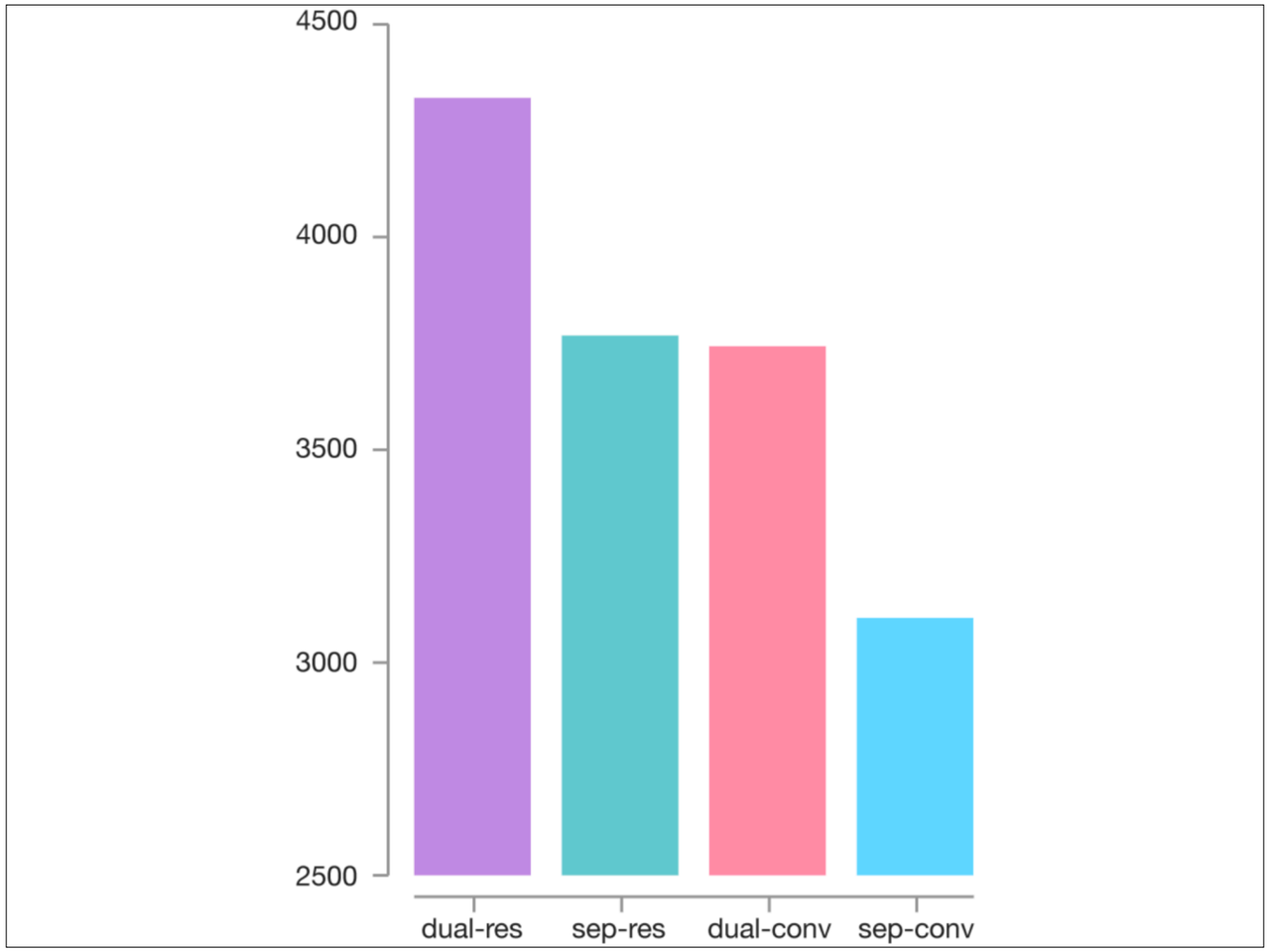


by Jeblad, wikipedia

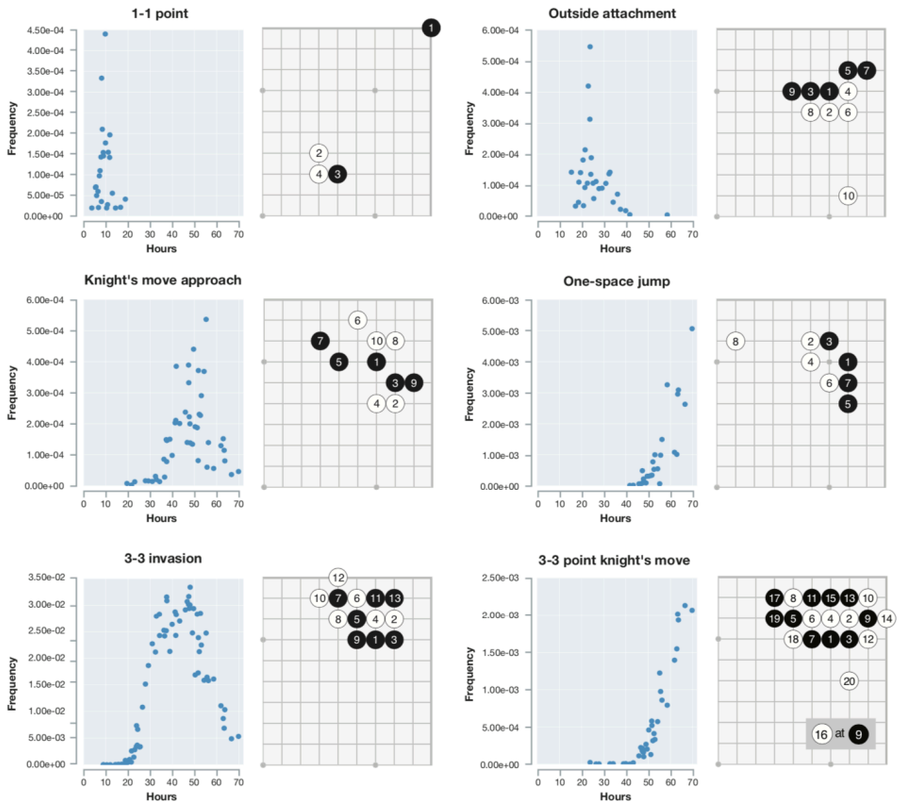
residual network has advantages on training because layers can be skipped to mitigate the vanishing gradient problem



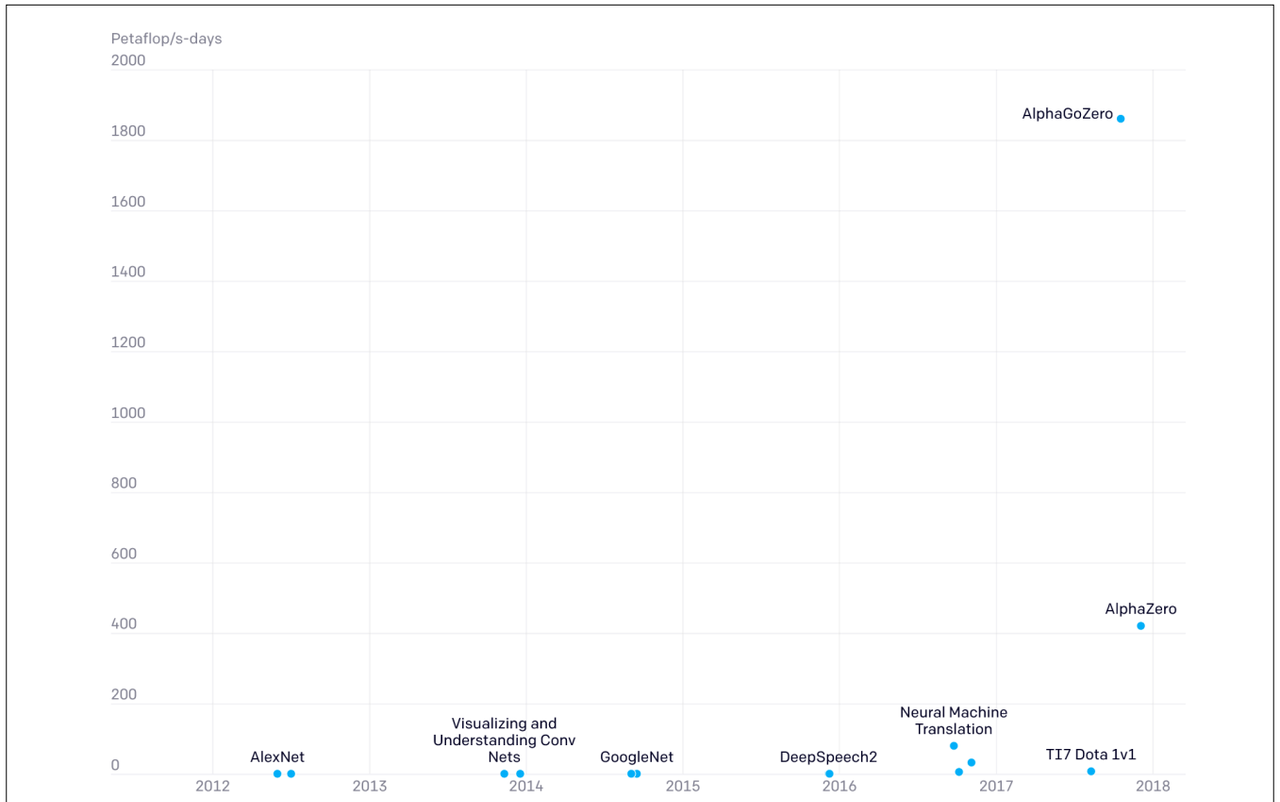
It is interesting to see that training AlphaGo Zero without MCTS by always only evaluating the highest probability action branch (Raw Network) leads to performance worse that AlphaGo Lee



system performance
sep-conv => conv net, policy and value network separated
dual-conv => conv net, joint policy and value network
sep-res => residual net, policy and value network separated
dual-res => residual net, joint policy and value network



discovers known moves/tactics but stops using them and discovers stronger ones



Source: https://openai.com/blog/ai-and-compute/

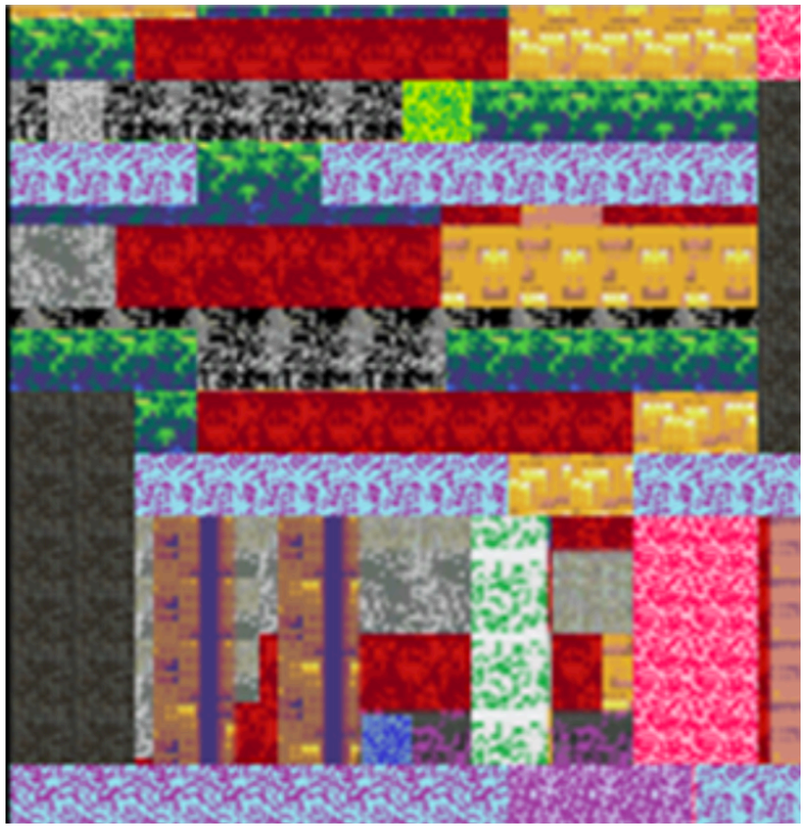
Price of single AlphaGo Zero system estimated US$25 million

64 GPUs and 19 CPUs? Excluding self-play with MTCS

Exercise

- Get familiar with the OpenAI gym (pip install gym)
  - https://github.com/romeokienzler/DeepRL => t-randomplay.py
- Create training data by randomly sample the environment (using random actions)
  - https://github.com/romeokienzler/DeepRL => t-trainingdata.ipynb
- Implement an agent using a policy network
  - https://github.com/romeokienzler/DeepRL => t-dlmodel.py

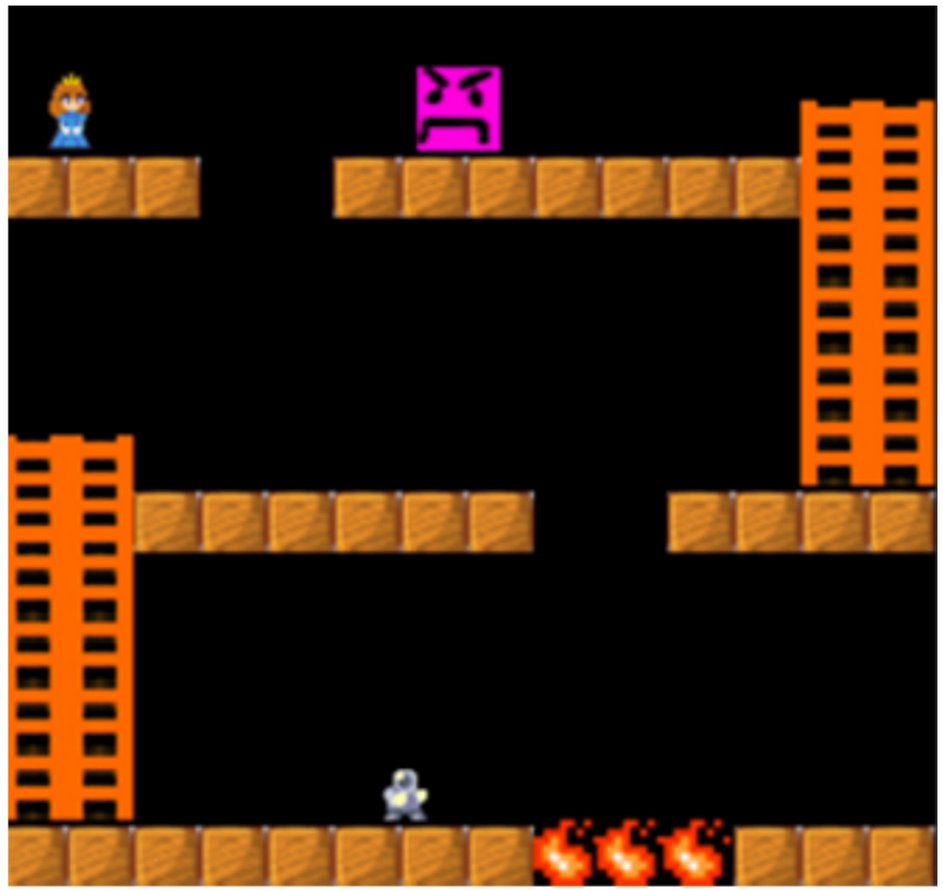
Influence
of
Human Priors



http://bit.ly/playhardcore

What do you think you need to do?

https://rach0012.github.io/humanRL_website/



What do you think you need to do?

best DeepRL algorithms need 4.000.000 frames for successfully solving this level

at 30 fps 37h of nonnstop playing

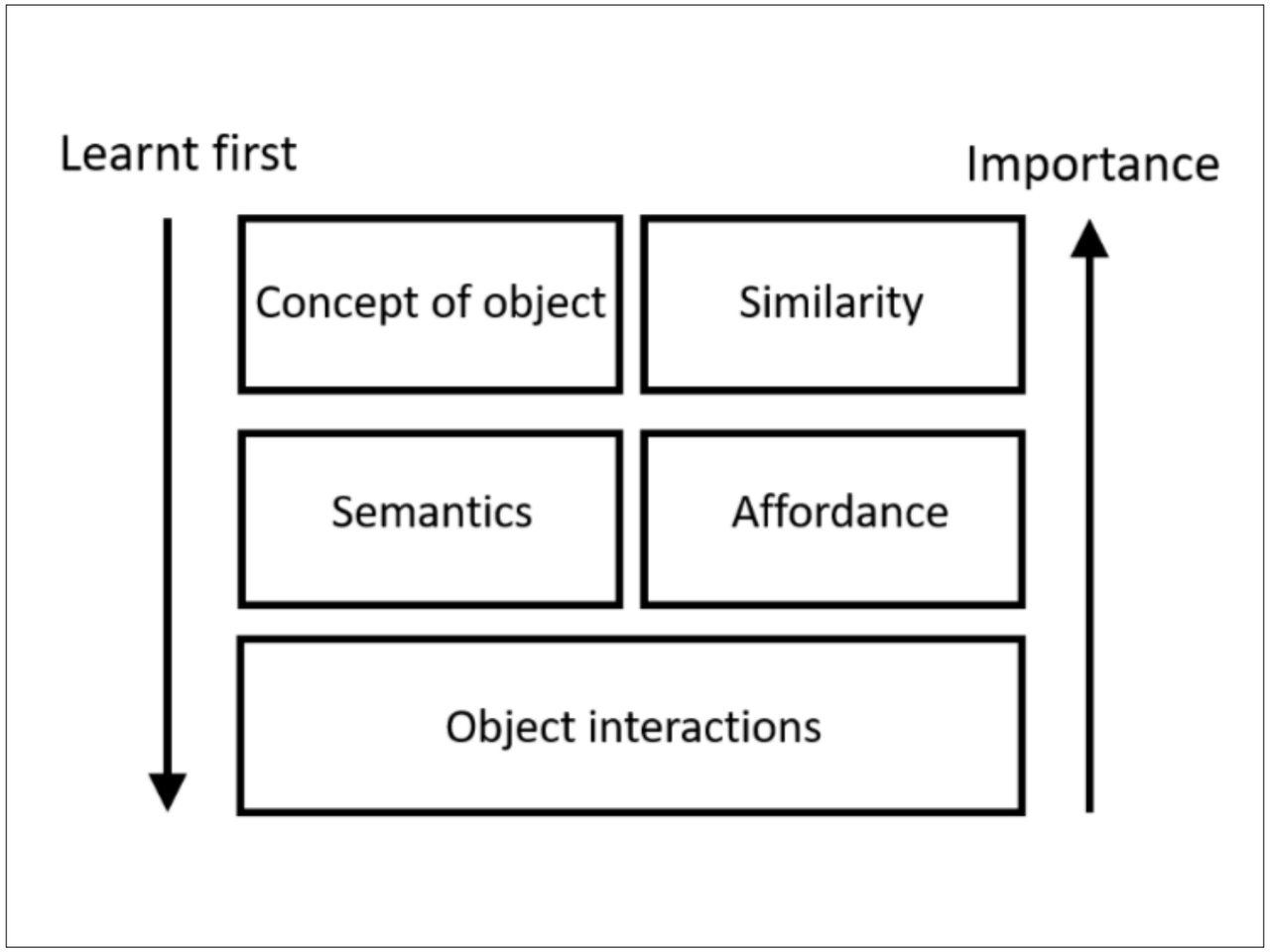
humans have prior information: ladders are to be climbed , non smiling faces and fire is unfriendly

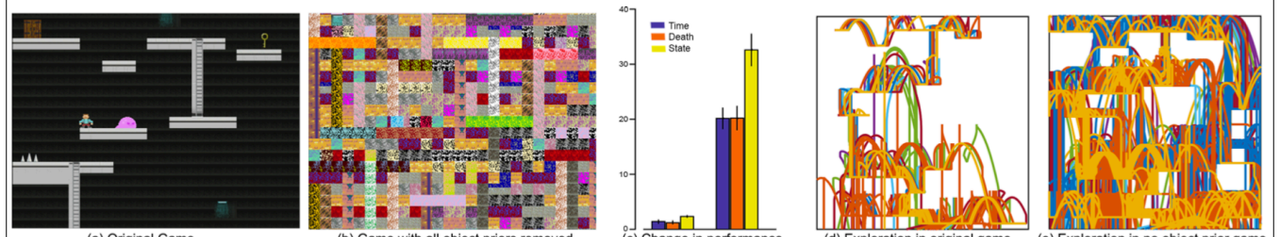
current DeepRL algorithms have what is called ,bad sample efficiency,

https://www.youtube.com/watch?v=cJ3Ec70Mnqw

human prior built into even new born: imitating, attention to faces

object permanence evolves very fast although slower than in monkeys: if something has been seen, it must exist even if it is hidden


Learnt first
Importance
Concept of object
Similarity
Semantics
Affordance
Object interactions



(a) Original Game (b) Game with all object priors removed (c) Change in performance (d) Exploration in original game (e) Exploration in no object prior game

deepRL algo always same performance on all games

prior knowledge can also be hindering (e.g. alpha go zero than alpha go lee)

Current Research Focus

- **Gradient update strategies, e.g.**
  - **TRPO (Trusted Region Policy Optimization)**
  - **PPO (Proximal Policy Optimization)**
- **Reward shaping**

The training data that is generated is itself dependent on the current policy because our agent is generating it's own training data by interacting with the environment rather then relying on a static data set as it is the case in supervised learning. This means the data distributions over observations and rewards are constantly changing which is a major cause of instability in the whole training process

DeepRL highly sensitive to initialization and hyperparameters

e.g. a policy can evolve to push an agent into an area of the environment where it can never recover again

in DQN experience is stored in replay buffer, in DRL experience is discarded once network is updated, therefore DQN is more sample efficient than DRL meaning "wasting" less data

advantage function: how much better was the action that I took vs. the expectation of our normal estimate: real reward - prediction of value function

micro Hackathon

- **In the last exercise you've used a randomly sampled training set of environment interactions**
- **In this Hackathon, we'll change the behaviour and create an intelligent agent**
- **The main difference is, that the intelligent agent is used to determine the next action (and therefore state and reward) used for computing the weights updates of the neural network**
- **Once implemented, please compare (ideally plot) the learning progress of your agent vs. the random sampled training data**
- **btw. using your agent to decide which part of the environment to explore is called "on-policy"**