Introduction to Reinforcement Learning
Session 5

Jean Martinet, based on the course of
Diane Lingrand

Polytech SI4 / EIT Digital

2020 - 2021

Feedback after second submission

- 41 submissions
- Globally well, but only few obtained good agent players
  - No training dataset recorded
    - Review the methodology in the lab description
  - Training yields low accuracy
    - Play well and longer
  - Agent didn't play at all
    - Make sure the Y values include all 3 possible moves
    - Check test accuracy (model generalises well ?)
    - Use imblearn lib to handle imbalanced datasets
  - Agent played well
    - Congratulations

Information about second assignment

- Instructions will be posted today (April 22)
- Work in groups of 2 (groups of 3 exceptionally, grading more strict)
  - Select an Atari game from a list
  - Select a training method and subject
  - Submit py+pdf or ipynb before Friday May 14 18 : 00 (3 weeks)

Mini-quizz

- How can should proceed for continuous action domains ?
  -
- By using gradient descent, are we guaranteed to reach the best solution ?
  -
- How can an ANN generate actions for states it has never seen ?
  -
- ANN are supervised... Are we actually doing RL ?
  -

Mini-quizz

- How can should proceed for continuous action domains ?
  - Tabular methods do not scale, use Value Function Approximation
- By using gradient descent, are we guaranteed to reach the best solution ?
  - No, we may reach a local minimum
- How can an ANN generate actions for states it has never seen ?
  - It can generalise and output a fair guess
- ANN are supervised... Are we actually doing RL ?
  - Good point. We're using SL to solve an RL problem, with a *continuously changing dataset*

Remember last week

$$\widehat{V^\pi(s)} = V_\theta(s) = V(s, \theta)$$

$$\widehat{Q^\pi(s, a)} = V_\theta(s, a) = V(s, a, \theta)$$

- Use a parametrised function to estimate value function and state-action value function
  - Compact representation that generalises across states and actions
- Need to have an oracle that tells us what the true value is
- The loss is defined with a mean squared error :

$$J(\theta) = \mathbb{E}_\pi[(V^\pi(s) - V(s, \theta))^2]$$

- Q-learning with a Deep ANN is *Deep Q-Learning* (DQN)

How to select the best action ?

- Discrete action space ?
  - Use argmax
- Continuous action space ?
  - Still possible to estimate $V_\theta(s, a)$
  - But requires to find the maximum with gradient ascent
  - (optimisation over action space)
  - Still possible, but likely to be very slow and inefficient
  - (+ risk of finding local maximum)
- Value function much more complex than the policy
  - Rather estimate directly $\pi_\theta(s, a)$!

Policy gradient

- Gradient ascent

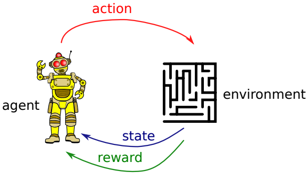
$$\theta_{t+1} = \theta_t + \alpha \widehat{\nabla J(\theta_t)}$$

- With $\widehat{\nabla J(\theta_t)}$ a stochastic estimate of the performance w.r.t θ_t
- Some methods learn both action value and policy : *actor-critic*
- Critic : estimate the value function
  - Estimate relevant value function, and then e.g. use ϵ-greedy
- Actor : learn the policy
  - Stochastic Policy Gradients (SPG) : output is a probability over actions
  - Deterministic Policy Gradients (DPG) : output is the value of an action (e.g. up, down, etc)

Policy gradient

- Estimate directly the policy
- Discounted rewards
- Neural network for modelling the policy :
  - π_θ where θ are the weights of the ANN

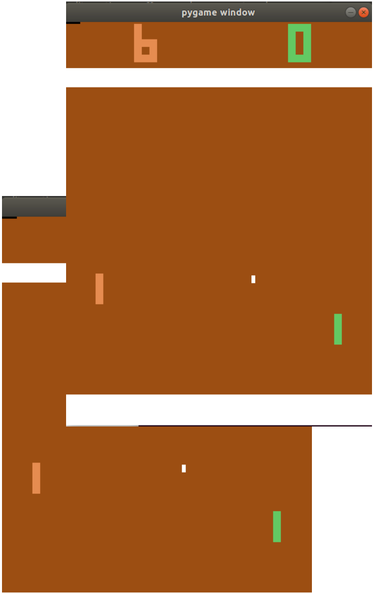
Remember Markov Decision Process (MDP)

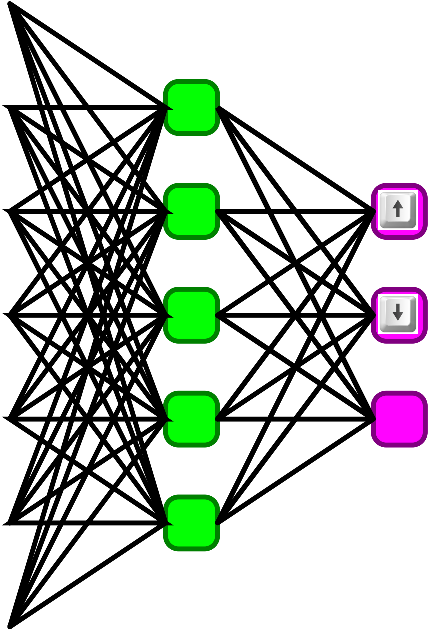


- Formal description of an environment for decision making / RL
- Tuple $\{S, A, P, R, \gamma\}$
  - **States :** s_t
  - **Action :** a_t
  - **Dynamics model (transitions) :** $P(s_t, a_t, s_{t+1}) \sim p(s_{t+1}|s_t, a_t)$
  - **Reward model :** $R(s_t)$ immediate reward
  - **Policy :** $\pi(s) \to a$.

Optimal policy : π^* Maximizes the long term expected reward or cumulative reward

Last week : supervised learning





Last week : learning

- pre-processing
  - image cropping (`obs_t[34:194,:,1]`)
  - one channel
  - downsampling (factor 2 in x and in y)
  - difference of images : need for a direction of displacement
- data :
  - recorded games : set of pairs (difference of images 80x80 ; actions)
  - action = up, down or nope

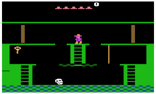
Even if the supervised learning step is performed at the highest quality, it will never outperform human players !

Could we play randomly and ...

... keep the best scores ?

- No, because of complexity
  - $\#$ pixels $= 6400$, $\#$ of connections, etc.
- Even if we start to play randomly, we need to learn from each experience
  - Yes, but we don't know the label – action performed correct or not ?
  - We only have sparse labels – game/round won or lost
- The main idea is to sample actions from a probabilistic model $a_i = y_i \sim p(.|x_i)$
  - this model is a neural network (forward pass)
  - try to maximise $\sum_i A_i \times \log(p(y_i|x_i))$ (backpropagation)
  - $A_i > 0$ will make that action more likely in the future for that x_i state
  - $A_i < 0$ will make that action less likely in the future for that x_i state

Reward function

- Example of pong :
  - Set of actions and reward at the end of the game/round
    - $+1$ if win
    - -1 if loose for the whole sequence even if only the last action was wrong : *Credit Assignment Problem*
  - This method fails if, by taking random action, you never go to a positive reward (e.g. Montezuma's Revenge) 
- Reward shaping :
  - Manually design a reward function to guide the policy
  - Needs to be redone for every new environment / game
  - Sometimes fails : agent focusing on highest rewards instead of final goal
  - For GO game, it is not optimal ($=$ playing like a human)

Back to pong reward : rounds

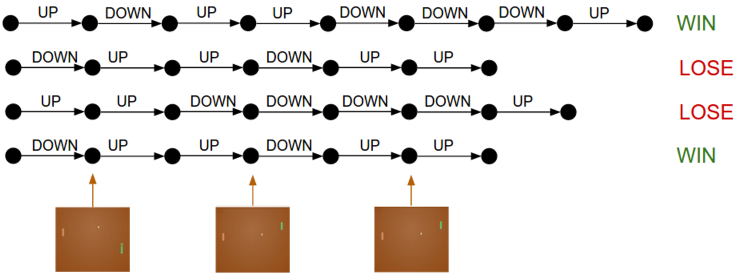


image from A. Karpathy's blog

- If we win the game : we assume that every action we took was correct (correct label)
- If we loose : we assume that every action we took was wrong (wrong label)

Discounted rewards

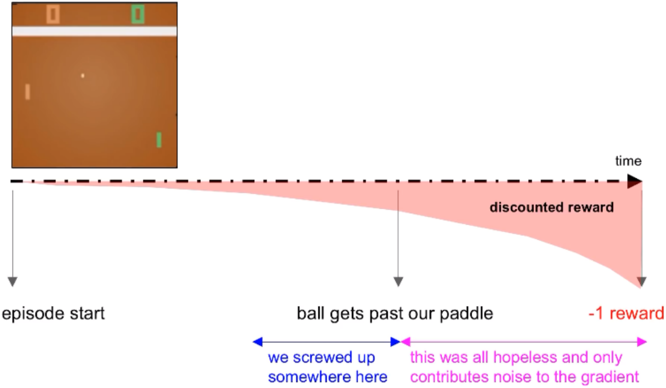


image from A. Karpathy's blog

- at t : -1
- at t-1 : $-\gamma$
- at t-2 : $-\gamma^2$

Discounted rewards

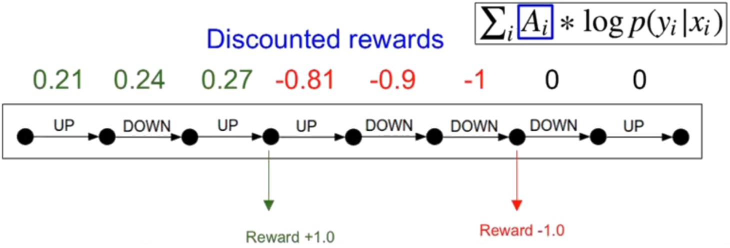


image from A. Karpathy's blog

where $\gamma = 0.9$

A simple code for policy gradient

- from Andrej Karpathy
  - available at `https://gist.github.com/karpathy/a4166c7fe253700972fcbc77e4ea32c5#file-pg-pong-py`
  - also an adaptation using keras `https://raw.githubusercontent.com/mkturkcan/Keras-Pong/master/keras_pong.py`
  - no dependancy except gym and numpy
- data collected during an episode (until one of the players reaches a score of 21)
  - neural network (one hidden layer of 200 neurons)
  - only one ouput neuron : probability of going up
  - random action according to estimated probabilities
  - batch for gradient computation
  - discounted rewards for a round
  - one step gradient iteration (using RMSprop)

Policy gradient

- Neural network with probabilities of actions as output
- Optimisation objective : $\hat{A}_t \log \pi_\theta(a_t|s_t)$
- Default Policy Gradient Loss : $L_P G(\theta) = \hat{E}_t[\log \pi_\theta(a_t|s_t)\hat{A}_t]$
  - The advantage $\hat{A}_t$ measures how good is an action compared to other actions
  - If $\hat{A}_t$ is positive, the gradient is positive, increasing these action probability
  - If $\hat{A}_t$ is negative, the gradient is negative, decreasing these action probability
- $\hat{A}_t = R_t - b(s_t)$
  - R_t : cumulative discounted rewards (we know what happened)
  - $b(s_t)$: baseline estimation (what we expected)

Policy gradient (2)

Vanilla Policy Gradient

Algorithm 1 "Vanilla" policy gradient algorithm

Initialize policy parameter θ, baseline b
for iteration$=1, 2, \ldots$ **do**
  Collect a set of trajectories by executing the current policy
  At each timestep in each trajectory, compute
   the *return* $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and
   the *advantage estimate* $\hat{A}_t = R_t - b(s_t)$.
  Re-fit the baseline, by minimizing $\|b(s_t) - R_t\|^2$,
   summed over all trajectories and timesteps.
  Update the policy, using a policy gradient estimate $\hat{g}$,
   which is a sum of terms $\nabla_\theta \log \pi(a_t \mid s_t, \theta)\hat{A}_t$
end for

Policy gradient algorithms

- TRPO (Trust Region Policy Optimization)
  https://arxiv.org/abs/1502.05477
  - The core idea is to avoid parameter updates that change the policy too much
- PPO (Proximal Policy Optimization)
  https://arxiv.org/abs/1707.06347
  - simpler objective function

Algorithm 1 PPO, Actor-Critic Style

for iteration$=1, 2, \dots$ **do**
    for actor$=1, 2, \dots, N$ **do**
        Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
        Compute advantage estimates $\hat{A}_1, \dots, \hat{A}_T$
    end for
    Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
    $\theta_{\text{old}} \leftarrow \theta$
end for

- AlphaGo uses policy gradients with Monte Carlo Tree Search (MCTS)

Today's lab

- download policy gradient algorithm for Pong by Andrej Karpathy
  - adapt to python3
    - print syntax
    - `model.iteritems` to `model.items`
- increase speed by modifying the learning rate to 10^{-3}
- try different ideas of improvement
  - reward shaping (positive reward if ball hitting the spade)
  - change the topology of the neural network :
    - 1 or 2 layers of convolution
    - 1 dense layer
  - modify the output : allow the possibility of no displacement