1706.10295 - Noisy Networks for Exploration

Noisy Networks for Exploration

Meire Fortunato* **Mohammad Gheshlaghi Azar*** **Bilal Piot ***
Jacob Menick **Ian Osband** **Alex Graves** **Vlad Mnih**
Remi Munos **Demis Hassabis** **Olivier Pietquin** **Charles Blundell**
Shane Legg
DeepMind
`{meirefortunato,mazar,piot,`
`jmenick,iosband,gravesa,vmnih,`
`munos,dhcontact,pietquin,cblundell,`
`legg}@google.com`

- **Yunqiu Xu**
- DeepMind's NoisyNet
    - Add parameytric noise to weights $\rightarrow$ introduce stochasticity to policy $\rightarrow$ efficient exploration
    - Has been integrated into Rainbow
- Further readings, can be seen in my notes
    - 1703.09327 - DART- Noise Injection for Robust Imitation Learning
        - Add noise to off-policy imitation learning
    - 1706.01905 - Parameter Space Noise for Exploration
        - Very similar work by OpenAI
    - 1710.02298 - Rainbow: Combining Improvements in Deep Reinforcement Learning
        - A combination of different DQN including NoisyNet

---

1. Introduction

- Challenges: how to explore (introduce new behabiors) efficiently
    - Dithering pertubations such as ϵ-greedy or entropy regularization

    - Random pertubations of agent's policy, e.g. trade-off exploration and exploitation with epsilon
    - **In every timestep, the noise is decorrelated**
    - Unefficient
  - Current methods' limitation:
    - Small state-action spaces
    - Linear function approximations
    - Not easy to be applied with more complicated system
  - A more structured method : add intrinsic motivation term to reward
    - Explicitly rewards novel discoveries
    - Limitation:
      - Separate the mechanism of generalisation from exploration
      - Need to balance the importance between additional term and reward manually
      - Not data efficient
- Our work: NoisyNet
  - Learn perturbations of weights to drive exploration
  - Key insight : a single change on weight can introduce effective changes in policy over multiple timesteps (**correlated noise**)
  - High level : introduce a randomised network for exploration
  - Requires only one extra parameter per weight
  - Can apply to PG methods such as A3C
- Similar work by OpenAI: **1706.01905 - Parameter Space Noise for Exploration**
  - Add constant Gaussian noise to parameters
  - Our difference:
    - Adapt the noise injection with time
    - Not restricted to Gaussian noise distributions
    - Can be adapted to any DRL such as DQN and A3C

2. NoisyNets for RL

- What is NoisyNets:

- NN whose weights and biases are pertubed by a parametric function of noise
- These parameters are adapted with GD

- Noisy layer $y = f_\theta(x)$
  - Take x as input, then output noised data y
  - Here x, y means weights or biases of general NN
  - Noisy parameters $\theta = \mu + \sum \odot \epsilon$
    - $\zeta = (\mu, \sum)$: set of learnable parameter vectors
    - ϵ : zero-mean noise with fixed statistics (e.g. Gaussian distribution)
- So for a general NN layer $y = wx + b$, the noise version can be:

$$y = (\mu^w + \sigma^w \odot \epsilon^w)x + \mu^b + \sigma^b \odot \epsilon^b$$

  where w, b are processed by noisy layer

- Combine NoisyNets with DRL
  - NoisyNet agent: sample a new set of parameters after each step of optimisation
  - Between optimisation steps, this agent acts according to a fixed set of parameters
  - 每回合选择不同的参数集合, 但回合中保持参数不变
- NoisyNet-DQN:
  - No ϵ-greedy, the policy optimises value function greedily
  - FC layers of value function are parameterised as a noise network $\rightarrow$ processed per replay step
  - Factorised Gaussian noise
  - Action-value function $Q(x, a, \sigma; \zeta)$
  - Noisy-DQN loss:

$$\bar{L}(\zeta) = \mathbb{E}\left[\mathbb{E}_{(x,a,r,y)\sim D}[r + \gamma \max_{b \in A} Q(y, b, \varepsilon'; \zeta^-) - Q(x, a, \varepsilon; \zeta)]^2\right]. \qquad (11)$$

- NoisyNet-A3C: similar to DQN
  - Entropy bonus of the policy loss is removed
  - FC layers of value function are parameterised as a noise network $\rightarrow$ processed per replay step

- Independent Gaussian noise
- As A3C uses n-step returns, optimisatin occurs every n steps, after each optimisation, the parameters of policy network are resampled

$$\hat{Q}_i = \sum_{j=i}^{k} \gamma^{j-i} r_{t+j} + \gamma^{k-i} V(x_{t+k}; \zeta, \varepsilon_i). \tag{12}$$

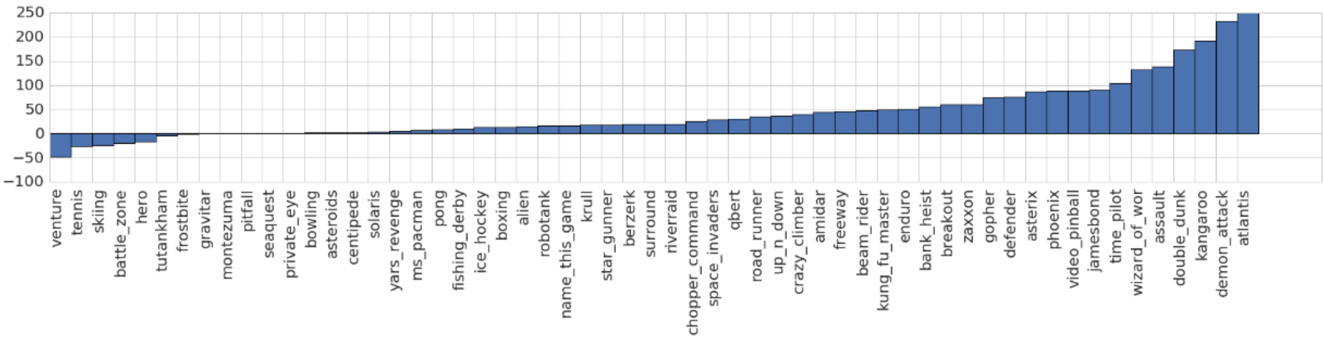
3. Experiment

- Task: 57 Atari games
- Comparison: DRL with originam exploration methods (ϵ-greedy and entropy bonus)
- Evaluation:
  - Absolute performance : human normalised score

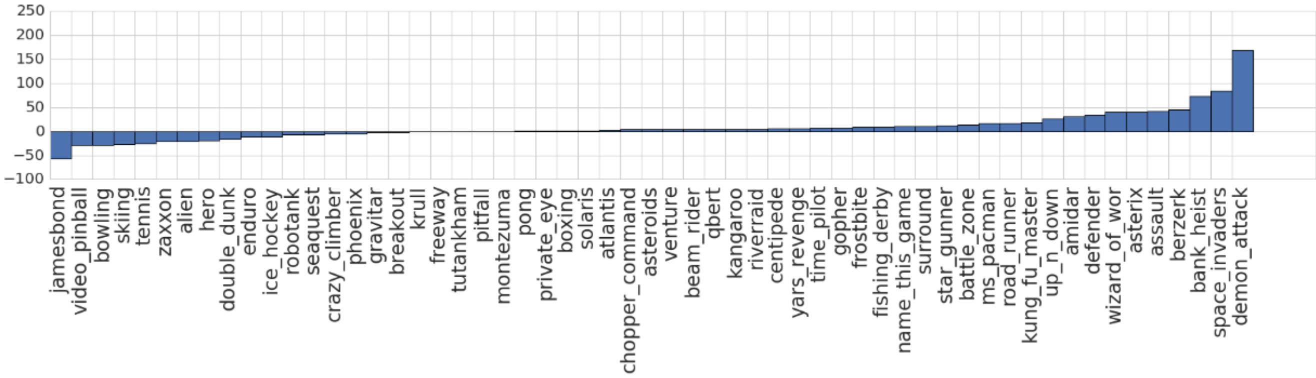
$$100 \times \frac{Score_{Agent} - Score_{Random}}{Score_{Human} - Score_{Random}}$$

  - Relative performance of NoisyNet agents to the respective baseline agent without noisy networks

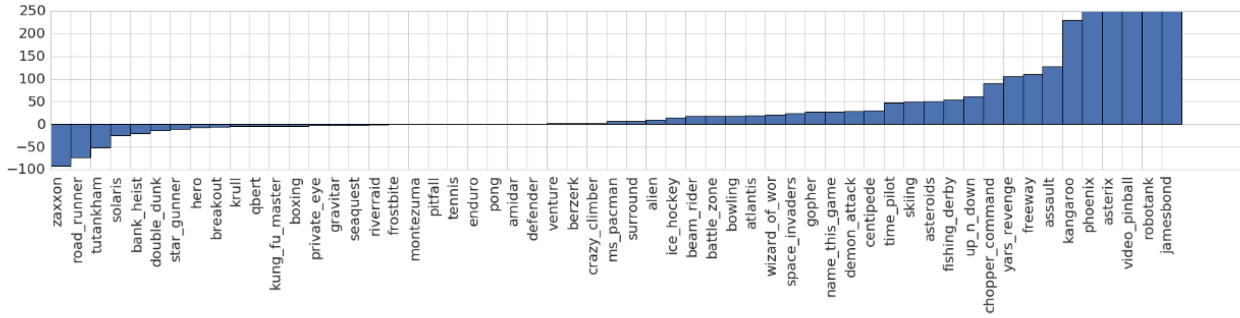
$$100 \times \frac{Score_{NoisyNet} - Score_{Baseline}}{max(Score_{Human}, Score_{Baseline}) - Score_{Random}}$$



(a) Improvement in percentage of NoisyNet-DQN over DQN [21]



(b) Improvement in percentage of NoisyNet-Dueling over Dueling [36]



(c) Improvement in percentage of NoisyNet-A3C over A3C [20]

Figure 1: Comparison of NoisyNet agent versus the baseline according to Eq. (14). The maximum score is truncated at 250%.

	Baseline		NoisyNet	
	Mean	Median	Mean	Median
DQN	213	47	**1210**	**89**
A3C	418	93	**1112**	**121**
Dueling	**2102**	126	1908	**154**

Table 1: Comparison between the baseline DQN [21], A3C [20] and Dueling [36] and their NoisyNet version in terms of median and mean human-normalised scores defined in Eq. (13).

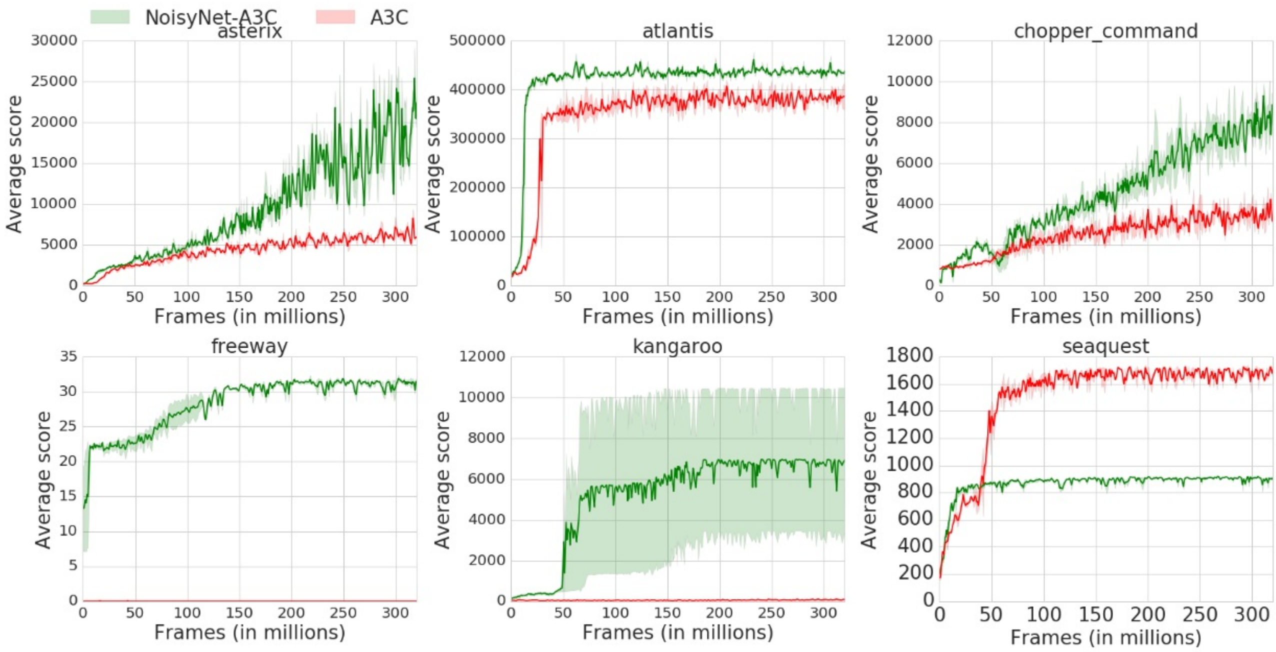


Figure 2: Training curves for selected Atari games comparing A3C and NoisyNet-A3C. Please refer to Fig. 3 in the Appendix for additional games.

4. Summary

- NoisyNet is a general method for exploration, which is easy to understand and implement
- Can be applied to DQN (off-policy) and A3C (on-policy)
- Surpass ϵ-greedy and entropy bonus
- **Have been integrated with other methods $\rightarrow$ Rainbow**
- Further reading: make comparison with OpenAI's similar work