Playing Atari with Deep Reinforcement Learning 2013

Gabriel Buginga

PESC

Sumário

Gabriel Buginga
PESC

1 Introdução

- Aprendizado por reforço é um subconjunto de aprendizado de máquina que modela um agente e um ambiente onde os dois interagem. O agente precisa ser controlado de tal forma que as suas recompensas sejam as maiores possíveis nessa tarefa.
- Até o artigo de 2013, se quiséssemos aplicar RL para inputs sensoriais de alta dimensão (imagens) usaríamos atributos feitos a mão e funções de representação lineares.
- O aprendizado profundo é exatamente esse conjunto de métodos montados para extrair atributos dessas fontes de alta dimensão. Lembrem das redes convolucionais.
- Porém não se pode aplicar o aprendizado profundo diretamente para o caso RL, dado algumas razões:
  - Deve aprender baseado num sinal de recompensa escalar o qual é frequentemente esparso, ruidoso e atrasado.
  - Atraso entre ações e recompensas.
  - Os pontos dos dados não são identicamente distruibuídos e a distribuição muda de acordo com o aprendizado de novos comportamentos.

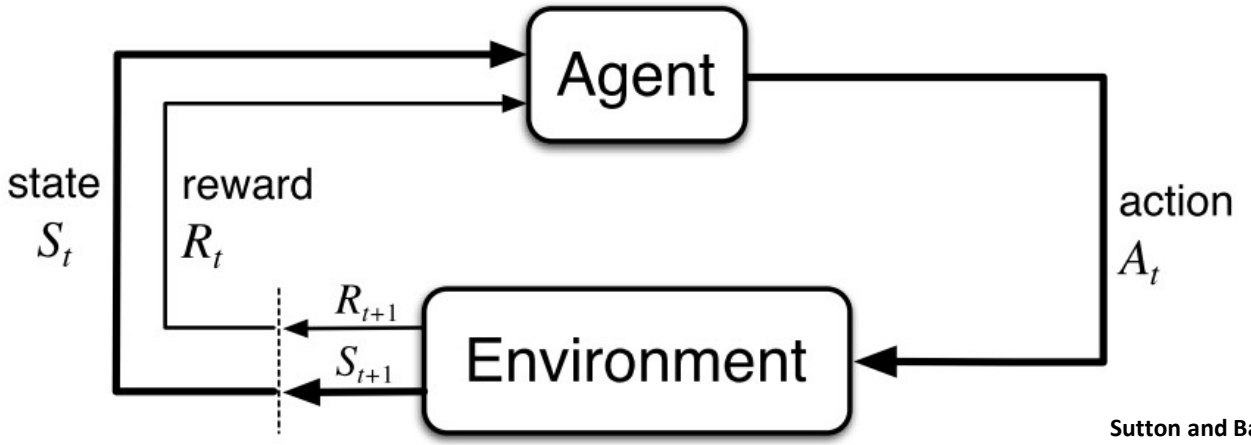
1 Introdução

- O artigo mostra que as CNNs podem aprender a controlar agentes baseado apenas no input de vídeo em ambientes complexos. A rede é treinada utilizando uma variação do *Q-Learning* com a ajuda do gradiente descendente estocástico, e um mecanismo chamado *replay buffer* – maiores descrições avante.
- Os Ambientes são sete jogos Atari 2600: input são vídeos 210x160 RGB com 60Hz.
- O objetivo é obter um agente CNN sem informação específica de cada jogo ou atributos específicos que aprenda a jogar apenas com as **imagens**, **recompensas**, **sinais de término** e **conjunto de possíveis ações**. De fato foi obtido: melhorou o desempenho de todos os algoritmos RL passados além de ultrapassar especialistas humanos em 3 dos jogos.

Figure 1: Screen shots from five Atari 2600 Games: (*Left-to-right*) Pong, Breakout, Space Invaders, Seaquest, Beam Rider

2 Contexto

- O agente interage com um ambiente $\mathcal{E}$.
- Em cada passo no tempo o agente seleciona uma ação a_t de um conjunto $\mathcal{A} = \{1, \ldots, K\}$ e recebe uma imagem $x_t \in \mathbb{R}^d$ e uma recompensa escalar r_t.
- O estado na verdade é $s_t = x_1, a_1, x_2, \ldots, a_{t-1}, x_t$, pois por causa do *perceptual aliasing* é impossível entender a situação observando apenas x_t.
- Portanto tem-se uma Markov Decision Process (MDP) tradicional.



Sutton and Barto (2018).

Figure 3.1: The agent–environment interaction in a Markov decision process.

2 Contexto

- O objetivo é maximizar as soma das recompensas submetidas à um fator de desconto γ. Sendo T o tempo de término, define-se o retorno descontado como:

$$R_t = \sum_{t'=t}^{T} \gamma^{t'-t} r_{t'}$$

- E a *optimal action-value function* como sendo o valor máximo esperado do retorno atingido por seguir a *policy* ótima (modo de escolher ações) depois que foi observado um estado s e uma ação foi tomada:

$$Q^*(s,a) = \max_\pi \mathbb{E}\left[R_t | s_t = s, a_t = a, \pi\right]$$

Q ótima respeita as
equações de Bellman

Policy é uma função: $S \rightarrow A$

$$Q^*(s,a) = \mathbb{E}_{s' \sim \mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') \middle| s,a\right]$$

2 Contexto

- A ideia básica é utilizar a equação de Bellman como uma atualização iterativa (caso tabular, essa família de algoritmos chama-se *value iteration* e converge para a função Q ótima). Ou seja:

$$Q_{i+1}(s,a) \quad = \mathbb{E}\left[r + \gamma \max_{a'} Q_i(s',a') | s,a\right]$$

- Como o espaço de estados é enorme a função Q será modelada como uma rede neural Q.
- Portanto em cada iteração resulta um treinamento utilizando uma *loss function* de regressão:

$$L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right]; \; y_i \; = \; \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) | s,a\right]$$

- Note que os parâmetros para $i-1$ são mantidos fixos na otimização.
- O que se faz quando se quer otimar uma *loss function*?

$$\nabla_{\theta_i} L_i(\theta_i) = \mathbb{E}_{s,a\sim\rho(\cdot);s'\sim\mathcal{E}}\left[\left(r + \gamma \max_{a'} Q(s',a';\theta_{i-1}) - Q(s,a;\theta_i)\right)\nabla_{\theta_i} Q(s,a;\theta_i)\right]$$

- Porém, utiliza-se o gradiente descendente estocástico – não se precisa calcular esse valor esperado. O *Q-learning* tradicional faz as atualizações dos pesos a cada passo no tempo (o que não se fará no artigo, ver-se-á o porquê).
- É *off-policy* e *model-free*.

3 Trabalhos relacionados

- TD-gammon: aprendizado por reforço *model-free* utilizando um algoritmo similar ao *Q-learning*, onde a função de estimativa foi modelo com um MLP com apenas uma camada escondida. Feito em 1995.
- Porém essa mesma abordagem não obteve o mesmo sucesso com xadrez, Go ou damas.
- Além disso, há resultados teóricos que mostram que algoritmos *model-free* e *off-policy* podem causar divergência na rede.
- Redes profundas já foram usadas para estimar o ambiente, Restricted Boltzmann Machines para a *value function* ou a *policy*. E o problema da divergência foi parcialmente tratado via métodos de *gradient temporal-difference.*
- Os trabalhos mais parecidos com o do presente artigo:
  - Utilizam o *neural fitted Q-learning (NFQ)*, otimizando aquela sequência de *loss functions* utilizando o RPROP para atualizar os parâmetros. Porém utiliza uma atualização que escala com o tamanho do dataset, e implementação em tarefas reais precisou de um autoencoder profundo para aprender uma representação com dimensões menores.
- O presente trabalho aplica aprendizado por reforço fim-a-fim.

4 *Deep Reinforcement Learning*

- O objetivo é, inspirados pelo trabalho do TD-gammon, utilizar um rede neural profunda para aprender a jogar nos ambientes do Atari 2600. Produzindo um novo algoritmo chamado: *deep Q-learning*.
- Utiliza-se uma técnica chamada *experience replay* onde se guarda as experiências do agente em cada passo de tempo, $e_t = (s_t, a_t, r_t, s_{t+1})$ dentro de um dataset $\mathcal{D} = e_1, ..., e_N$ agrupados sobre vários episódios dentro de uma *replay memory.*
- As atualizações da *Q-function* são realizadas em amostras da *replay memory:* $e \sim \mathcal{D}$.
- Além disso cada estado é passado por uma função ϕ que realiza as seguintes transformações:
  - As imagens são tranformadas de 210x160 RGB para escala cinza sub-amostradas até 110x84; mais um corte para uma região quadrada de 84x84.
  - Aplica esse processamento nos últimos **4 frames e os empilha** e passa para a *Q*.

4 *Deep Reinforcement Learning*

- **Rede Neural Convolucional (DQN) utilizada:**
  - Input: 84x84x4 tensor produzido por ϕ.
  - Primeira camada: 16 filtros 8x8 com stride 4 e RELU.
  - Segunda camada: 32 filtros 4x4 com stride 2 e RELU.
  - Última camada: 256 unidades RELU totalmente conectadas.
  - Camada de saída: camada linear totalmente conectada com número de saídas sendo o número de ações possíveis.
- **Pontos positivos do algoritmo:**
  - Cada amostra de experiência pode ser usada várias vezes.
  - Randomizar as amostras quebra a correlação delas, diminuindo a variância.
  - Se fosse *on-policy* os parâmetros atuais determinariam as próximas amostras que os parâmetros iriam ser treinados. Todavia o presente algoritmo é *off-polic*y.

Algorithm 1 Deep Q-learning with Experience Replay

Initialize replay memory $\mathcal{D}$ to capacity N
Initialize action-value function Q with random weights
for episode $= 1, M$ **do**
  Initialise sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    With probability ϵ select a random action a_t
    otherwise select $a_t = \max_a Q^*(\phi(s_t), a; \theta)$
    Execute action a_t in emulator and observe reward r_t and image x_{t+1}
    Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
    Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $\mathcal{D}$
    Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $\mathcal{D}$
    Set $y_j = \begin{cases} r_j & \text{for terminal } \phi_{j+1} \\ r_j + \gamma \max_{a'} Q(\phi_{j+1}, a'; \theta) & \text{for non-terminal } \phi_{j+1} \end{cases}$
    Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \theta))^2$ according to equation 3
  end for
end for

5 Experimentos

- Os mesmos arquitetura, algoritmo de aprendizado e hiperparametros foram usados para todos os 7 jogos. Mudou-se a escala das recompensas para -1,0,1 durante o treinamneto.
- Utilizou-se o RMSProp com *minibatches* de tamanho 32.
- A *behaviour policy* (aquela usada para angariar expriência e gerar os dados) foi *epsilon greedy* de 1 e diminuída para 0.1 depois de 1 milhão de frames.
- Foi treinado um total de 10 milhões de frames e utilizado uma *replay memory* de 1 milhão mais recentes frames.
- Técnica simples de pular frames: ações são escolhidas de k em k frames e repetidas nesses frames pulados. (k=4 para todos menos o Space Invaders onde k=3)

5 Experimentos

- Métricas:
  - A*verage total reward*: recompensa total recolhida pelo agente em um episódio, tirando a média de algumas tentativas.
  - *Estimated action-value function Q*: coleta um conjunto fixo de estados conforme uma *policy* randômica antes do treinamento acontecer e mantêm uma média do valor máximo predito por *Q* para esses estados.

5 Experimentos

- Os resultados para os outros jogos foram similares – incluindo a forma das gráficos, mostrando o padrão de funcionamento do DQN.

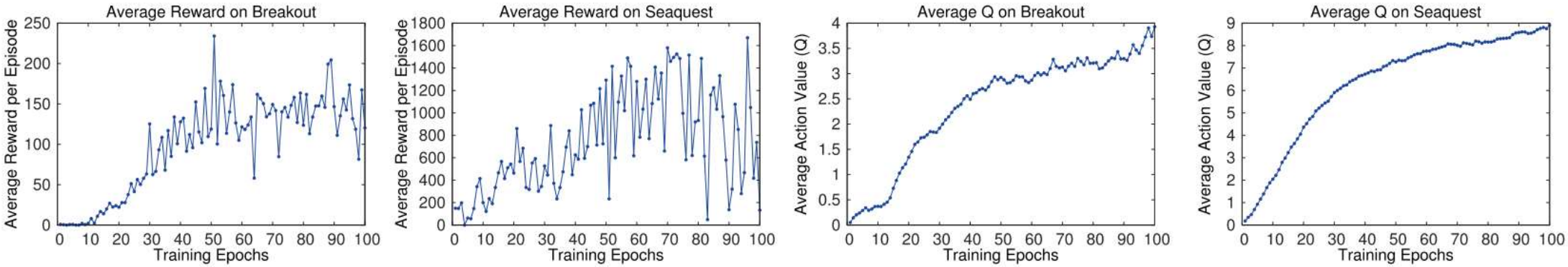


Figure 2: The two plots on the left show average reward per episode on Breakout and Seaquest respectively during training. The statistics were computed by running an ϵ-greedy policy with $\epsilon = 0.05$ for 10000 steps. The two plots on the right show the average maximum predicted action-value of a held out set of states on Breakout and Seaquest respectively. One epoch corresponds to 50000 minibatch weight updates or roughly 30 minutes of training time.

5 Experimentos

- A figura 3 mostra como a função prevê uma sequência de eventos:
  - ponto A um inimigo aparece no canto esquerdo da tela,
  - ponto B o torpedo emitido pelo jogador está prestes a atingir o inimigo,
  - ponto C o valor retorna aproximadamente ao seu valor original.

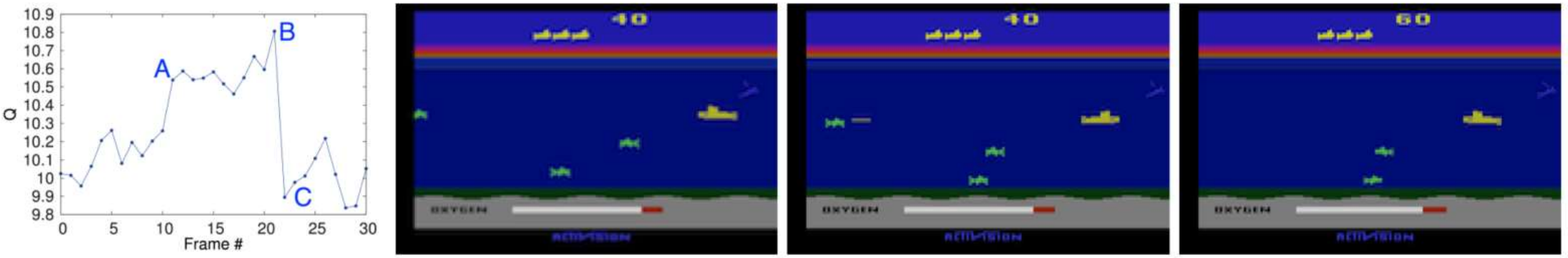


Figure 3: The leftmost plot shows the predicted value function for a 30 frame segment of the game Seaquest. The three screenshots correspond to the frames labeled by A, B, and C respectively.

5 Experimentos

	B. Rider	Breakout	Enduro	Pong	Q*bert	Seaquest	S. Invaders
Random	354	1.2	0	−20.4	157	110	179
Sarsa [3]	996	5.2	129	−19	614	665	271
Contingency [4]	1743	6	159	−17	960	723	268
DQN	**4092**	**168**	**470**	**20**	**1952**	**1705**	**581**
Human	7456	31	368	−3	18900	28010	3690
HNeat Best [8]	3616	52	106	19	1800	920	**1720**
HNeat Pixel [8]	1332	4	91	−16	1325	800	1145
DQN Best	**5184**	**225**	**661**	**21**	**4500**	**1740**	1075

Table 1: The upper table compares average total reward for various learning methods by running an ϵ-greedy policy with $\epsilon = 0.05$ for a fixed number of steps. The lower table reports results of the single best performing episode for HNeat and DQN. HNeat produces deterministic policies that always get the same score while DQN used an ϵ-greedy policy with $\epsilon = 0.05$.

6 Conclusão

- Esse trabalho pioneiro mostrou a possibilidade de utilizar o aprendizado por reforço com a utilização de redes neurais profundas com intuito de aprender uma *policy* de controle obtendo resultados do estado da arte (para 2013). Sendo que o comportamento foi aprendido diretamente sobre o input de dimensão alta, ou seja, a arquitetura foi fim-a-fim.

- Além disso, o mesmo modelo foi utilizada para todos os 7 jogos do Atari 2600, mostrando a essência da abordagem de aprendizado de máquina: aprender com o mínimo possível de suposições.

- Para isso utilizou uma variação de *online Q-learning*, combinando atualizações estocásticas via minibatches com um *experience replay buffer*. Obtendo o algoritmo chamado *Deep Q-Learning*.

Referência:

- *Playing Atari with Deep Reinforcement Learning*, (Mnih, 2013).