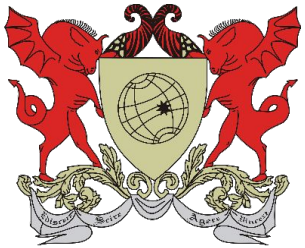


Deep Q-learning

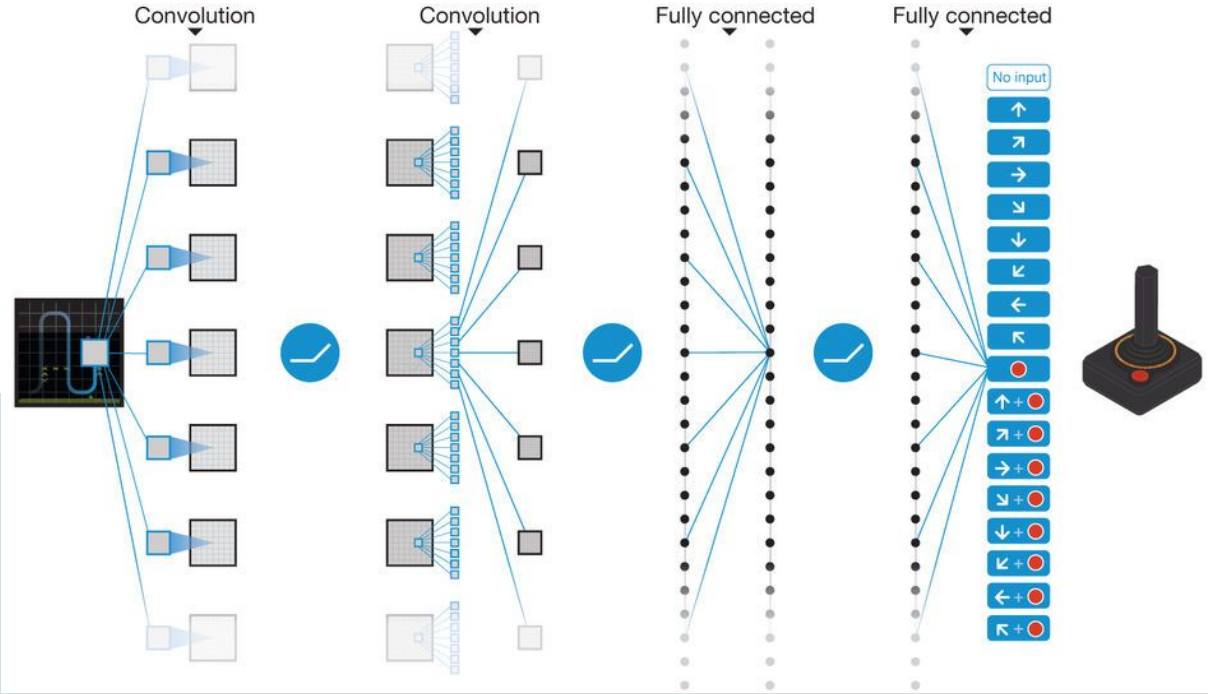
Alcione de Paiva - DPI/UFV

Introdução

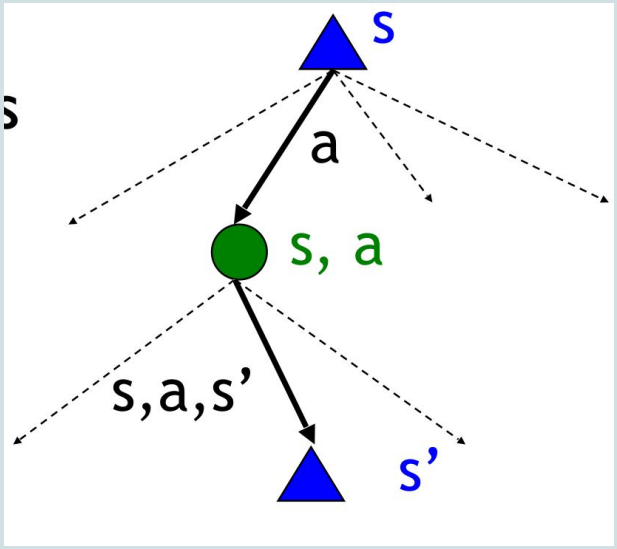
- Publicado na Nature.
- É uma CNN treinada com uma variante de Q-learning.
- Usa jogos Atari como teste.
- Usa os pixels brutos como entrada.
- Não recebe informação específica do jogo ou recursos visuais desenhados à mão.

Introdução

- Nova função objetivo



Relembrando



Relembrando

- **Aprendizagem-Q**: Iteração de Valor Q baseado em amostragens.

$$Q_{k+1}(s,a) \leftarrow \sum_{s'} T(s,a,s')\left[R(s,a,s') + \gamma \max_{a'} Q(s',a')\right]$$

- Aprende a medida que vai agindo:
  1. Recebe amostra (s, a, s', r)
  2. Considera o valor antigo de Q(s, a)
  3. Considera a estimativa da nova amostra:

$$amostra = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  4. Atualiza o valor de Q(s, a) de acordo com a média:

$$Q(s,a) \leftarrow (1-\alpha)Q(s,a) + \alpha[amostra]$$

Introdução

- No algoritmo Q-learning original, o agente aprende a política testando ações aleatoriamente para cada estado.

- Mas como aprender em situações onde o número de estados é muito grande ou muito complexos, como em jogos ou mundo real?

Introdução

- Em um jogo simples como o breakout o número de posições e peças é muito grande. Além disso, não ficaria independente do tipo de jogo.

O Algoritmo

- Uma forma de tornar independente é definir o estado como a configuração dos pixels.

- Eles implicitamente contém todas as informações relevantes sobre a situação do jogo, exceto pela velocidade e direção da bola. Duas telas consecutivas conteria essa informação.

O Algoritmo

- Se aplicarmos o mesmo pré-processamento a telas de jogos como no artigo da DeepMind: obtém as quatro últimas imagens de tela, redimensiona para 84 × 84 e converte para tons de cinza com 256 níveis de cinza - obtém-se $256^{84 \times 84 \times 4} \approx 10^{67970}$ estados de jogo possíveis .

- Isso significa 10^{67970} linhas em nossa tabela-Q imaginária - isso é mais do que o número de átomos no universo conhecido.

O Algoritmo

- Esse é o ponto em que o aprendizado profundo entra em cena.
- As redes neurais são excepcionalmente boas para descobrir features em dados altamente estruturados.
- Poderíamos representar a função Q com uma rede neural, que toma o estado (quatro telas de jogo) e a ação como entrada e produz como saída do valor Q correspondente.

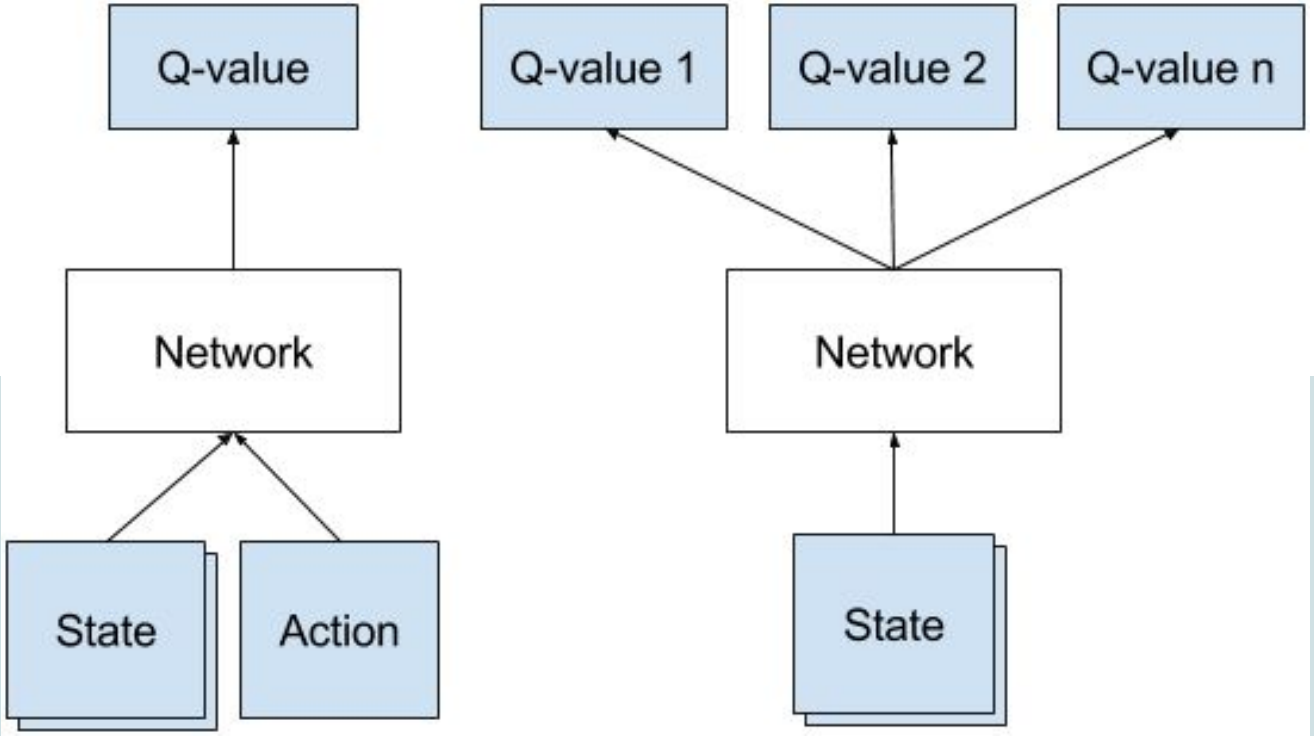
O Algoritmo

- Alternativamente, poderíamos ter apenas telas do jogo como entrada e como saída do valor Q para cada ação possível.

- Essa abordagem tem a vantagem de que, se quisermos executar uma atualização do valor Q ou escolher a ação com o maior valor Q, só temos que fazer uma passagem à frente pela rede e ter todos os valores Q para todas as ações prontamente disponíveis.

O Algoritmo

- As duas abordagens



O Algoritmo

- A arquitetura usada pela DeepMind foi a seguinte

Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8×8	4	32	ReLU	20x20x32
conv2	20x20x32	4×4	2	64	ReLU	9x9x64
conv3	9x9x64	3×3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

O Algoritmo

- Note que não há camadas de pooling.
- As camadas de pooling faz com que a rede se torne insensível à localização de um objeto na imagem.
- Isso faz muito sentido para uma tarefa de classificação, mas para jogos a localização da bola, por exemplo, é crucial para determinar a recompensa em potencial e não se pode descartar essa informação!

O Algoritmo

- Entrada para a rede são quatro telas de jogo em tons de cinza 84 × 84.
- As saídas da rede são valores de Q para cada ação possível (18 no Atari).
- Os valores Q podem ser quaisquer valores reais, o que torna uma tarefa de regressão, onde pode ser otimizada com uma função de perda de erro quadrádico simples.

O Algoritmo

- Nova função objetivo

$$L = \frac{1}{2}[\underbrace{r + max_{a'}Q(s', a')}_{\text{target}} - \underbrace{Q(s, a)}_{\text{prediction}}]^2$$

O Algoritmo

- Dada uma transição <s, a, r, s ’>, a regra de atualização dos valores Q no algoritmo original deve ser substituída pelo seguinte:

O Algoritmo

1. Faça um feedforward para os estados atuais para obter Q-valores estimados para todas as ações.
2. Faça um feedforward para o próximo estado s′ e calcule o máximo para todas as saídas de rede $max_{a'}Q\ (s', a')$.
3. Defina o valor Q para a ação a para $r + \gamma max_{a'}Q\ (s', a')$. Para todas as outras ações, defina o valor Q como a originalmente retornada da etapa 1, fazendo o erro 0 para essas saídas.
4. Atualize os pesos usando backpropagation.

O Algoritmo

- Acontece que a aproximação dos valores Q usando funções não lineares não é muito estável.
- Existem vários truques que você precisa usar ajudar convergir.
- E leva muito tempo, quase uma semana em uma única GPU.
- O truque mais importante é o replay da experiência.

O Algoritmo

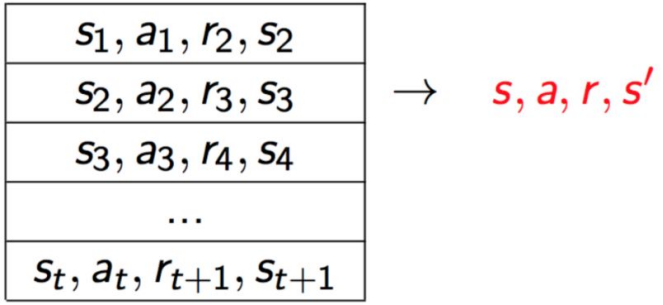
Memória de replay

- Durante o jogo, todas as experiências <s, a, r, s'> são armazenadas em uma memória de replay.

- Ao treinar a rede, amostras aleatórias da memória de replay são usadas em vez da transição mais recente. Isso quebra a similaridade de amostras de treinamento subsequentes, o que de outra forma poderia levar a rede a um mínimo local.

O Algoritmo

Deep Q-learning



- Obtenha uma tupla da memória (s,a,r,s')
- Obtenha um valor alvo para a amostra s: $r + \gamma \ max_{a'} \hat{Q}(s', a'; \mathbf{w})$.
- Use gradient descent para atualizar os pesos

$$\Delta w = \alpha(r + \gamma \ max_{a'} \hat{Q}(s', a'; w) - \hat{Q}(s, a; w)) \nabla w \ \hat{Q}(s, a; w)$$

- Para melhorar a estabilidade, fixe os pesos usados no cálculo do alvo

$$\Delta w = \alpha(r + \gamma \ max_{a'} \hat{Q}(s', a'; w^-) - \hat{Q}(s, a; w)) \nabla w \ \hat{Q}(s, a; w)$$

1: Input C, α, $D = \{\}$, Initialize $\boldsymbol{w}$, $\boldsymbol{w}^- = \boldsymbol{w}$, $t = 0$
2: Get initial state s_0
3: **loop**
4: Sample action a_t given ϵ-greedy policy for current $\hat{Q}(s_t, a; \boldsymbol{w})$
5: Observe reward r_t and next state s_{t+1}
6: Store transition (s_t, a_t, r_t, s_{t+1}) in replay buffer D
7: Sample random minibatch of tuples (s_i, a_i, r_i, s_{i+1}) from D
8: **for** j in minibatch **do**
9: **if** episode terminated at step $i + 1$ **then**
10: $y_i = r_i$
11: **else**
12: $y_i = r_i + \gamma \max_{a'} \hat{Q}(s_{i+1}, a'; \boldsymbol{w}^-)$
13: **end if**
14: Do gradient descent step on $(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))^2$ for parameters $\boldsymbol{w}$: $\Delta \boldsymbol{w} = \alpha(y_i - \hat{Q}(s_i, a_i; \boldsymbol{w}))\nabla_{\boldsymbol{w}} \hat{Q}(s_i, a_i; \boldsymbol{w})$
15: **end for**
16: $t = t + 1$
17: **if** mod(t,C) $==$ 0 **then**
18: $\boldsymbol{w}^- \leftarrow \boldsymbol{w}$
19: **end if**
20: **end loop**

Fonte: Emma Brunskill, CS234 Reinforcement Learning. Stanford.

O Algoritmo

Exploração e Explotação

- Quando uma Q-table ou Q-network é inicializada aleatoriamente, suas previsões são inicialmente aleatórias também. Se escolhermos uma ação com o maior valor Q, a ação será aleatória e o agente executará "exploração" básica.

- À medida que uma função Q converge, ela retorna valores Q mais consistentes e a quantidade de exploração diminui. Então, pode-se dizer que o Q-learning incorpora a exploração como parte do algoritmo. Mas essa exploração é "gulosa", escolhendo a primeira estratégia efetiva que encontra.

O Algoritmo

Exploração e Explotação

- Uma correção simples e eficaz para o problema acima é a exploração ε-gulosa - com probabilidade ε escolher uma ação aleatória, caso contrário, vá com a ação "gulosa" com o maior valor-Q.

- O DeepMind diminui ε ao longo do tempo de 1 para 0.1 - no início, o sistema faz movimentos completamente aleatórios para explorar o espaço de estados ao máximo e, em seguida, estabelece uma taxa de exploração fixa.

Algoritmo Final

```
inicializar a memória de replay D
inicializar a função valor-ação Q com pesos aleatórios
observe o estado inicial s
repetir
     selecione uma ação a
          com probabilidade ε seleciona uma ação aleatória
          caso contrário, selecione a = argmaxa'Q(s, a') // Aqui executa a rede
neural
     realizar uma ação
     observar recompensa r e novo estado s'
     Armazene experiência <s, a, r, s'> na memória de replay D

     Amostre transições aleatórias <ss, aa, rr, ss'> da memória de replay D
     calcular o resultado para cada transição do minibatch
          se ss' é o estado terminal, então tt = rr
          Caso contrário, tt = rr + γmaxa'Q (ss', aa')
     treinar a rede Q usando (tt - Q (ss, aa))^2 como perda

     s = s'
até fim
```

Conclusões

- Muitas melhorias para o Deep Q-learning foram propostas desde sua primeira introdução - Q-learning duplo, Replay priorizado de experiência, Arquitetura de redes Adversárias (GANS) e extensão para o espaço de ação contínua, para citar apenas alguns.

- Deep Q-learning foi patenteado pelo Google.

Links interessantes

- http://neuro.cs.ut.ee/demystifying-deep-reinforcement-learning/
- https://medium.freecodecamp.org/an-introduction-to-deep-q-learning-lets-play-doom-54d02d8017d8
- https://medium.com/@joshpatterson_5192/introduction-to-deep-q-learning-1bded90a6193
- https://yanpanlau.github.io/2016/07/10/FlappyBird-Keras.html
- https://keon.io/deep-q-learning/
- https://medium.com/@gtnjuvin/my-journey-into-deep-q-learning-with-keras-and-gym-3e779cc12762
- https://ibrahimsobh.github.io/Practical-DRL/
- https://www.deeplearningbook.com.br/
- https://gym.openai.com/
- https://unity3d.com/pt/how-to/unity-machine-learning-agents

Livros

Russel S. & Norvig P. *Inteligência Artificial, Campus*; ISBN: 8535211772, 2010. Terceira Ed.

Haykin, Simon. Neural networks and learning machines. 3 ed. Pearson Education Ed. 2009.