



# Métodos de Regressão para Aprendizado por Reforço

**Lucas de Almeida Teixeira**

JUIZ DE FORA

DEZEMBRO, 2016

# Métodos de Regressão para Aprendizado por Reforço

Lucas de Almeida Teixeira

Universidade Federal de Juiz de Fora
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Orientador: Saul de Castro de Leite

JUIZ DE FORA
DEZEMBRO, 2016

# Métodos de Regressão para Aprendizado por Reforço

Lucas de Almeida Teixeira

MONOGRAFIA SUBMETIDA AO CORPO DOCENTE DO INSTITUTO DE CIÊNCIAS EXATAS DA UNIVERSIDADE FEDERAL DE JUIZ DE FORA, COMO PARTE INTEGRANTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE BACHAREL EM CIÊNCIA DA COMPUTAÇÃO.

Aprovada por:

---

Saul de Castro de Leite
Dr. em Modelagem Computacional

---

Luciana Conceição Dias Campos
Dra. em Engenharia Elétrica

---

Carlos Cristiano Hasenclever Borges
Dr. em Engenharia Civil

JUIZ DE FORA

02 DE DEZEMBRO, 2016

*Dedico aos meus pais, sem os quais eu não existiria.*

# Resumo

Problemas de aprendizado por reforço resumem-se a situações onde um agente inteligente deve agir com base na observação do ambiente no qual se encontra. A ação do agente afeta o ambiente resultando em uma recompensa e uma nova observação do ambiente. Porém o efeito de uma ação é estocástico, ou seja, é possível que ao tomar duas vezes a mesma ação no mesmo momento os efeitos sejam diferentes. Para tratar problemas dessa natureza desenvolveu-se uma vasta teoria para modelá-los e resolvê-los. Uma forma bastante popular de resolver esse tipo de problema é utilizar *Q-Learning*, porém essa técnica necessita que seja realizada a aproximação da função $Q$. Este trabalho propõe uma nova maneira de realizar tal aproximação utilizando *Perceptron* de $\varepsilon$-Raio Fixo e a compara com outra abordagem já consolidada que são as Redes Neurais Artificiais. Para realização da comparação foram implementados diferentes problemas de aprendizado por reforço e ambos algoritmos são usados para resolvê-los.

**Palavras-chave:** Aprendizado por Reforço, Máquina de Vetores de Suporte, Redes Neurais Artificiais, Aprendizado de Máquinas, Inteligência Artificial.

# Abstract

Reinforcement learning problems can be described as problems where an intelligent agent must act based on the observation of its environment. Each action taken by the agent affects the environment resulting in a reward and a new state for the environment. However, the effect of an action is stochastic, which means that the same action on the same state may have different outcomes. In order to deal with problems of this nature a vast theory has been developed to model and solve them. A very popular way of solving this type of problem is to use the $Q$-Learning algorithm, but this technique requires the approximation of the function $Q$. This work proposes a new way of performing such an approximation using the Fixed $\varepsilon$-Radius Perceptron algorithm. This new approach is compared to Artificial Neural Networks, which are often used for this task. In order to compare these two approaches, a set of different reinforcement learning problems were implemented and both algorithms are used to solve them.

**Keywords:** Reinforcement Learning, Support Vector Machine, Artificial Neural Network, Machine Learning, Artificial Intelligence.

# Agradecimentos

A minha família, pelo encorajamento, apoio e confiança.

Ao professor Saul pela orientação, amizade e principalmente, pela grande paciência, sem a qual este trabalho não se realizaria.

Aos professores do Departamento de Ciência da Computação pelos seus ensinamentos e aos funcionários do curso, que durante esses anos, contribuíram de algum modo para o nosso enriquecimento pessoal e profissional.

Aos meus amigos e aos colegas do GETComp, pela amizade e por todo o tempo de convivência.

*"Wyrd bið ful ãræd".*

*Bernard Cornwell (As Crônicas Saxônicas)*

# Conteúdo

# Lista de Figuras

# Lista de Tabelas

# 1 Introdução

O objetivo da inteligência artificial (IA) é fazer com que máquinas exibam comportamento inteligente. Dentre as várias subáreas da IA destaca-se o aprendizado de máquina cuja finalidade é possibilitar que os computadores aprendam uma tarefa, a partir de experiências passadas, e sejam capazes de resolver essa tarefa sozinhos. Ou seja, sem a necessidade de que um algoritmo especifico para esta tarefa seja construído. A área de aprendizado de máquinas é divida em algumas subáreas, dentre elas estão o aprendizado supervisionado e o aprendizado por reforço. O aprendizado supervisionado considera problemas cujo objetivo é aprender alguma tarefa a partir de um conjunto de exemplos de valores de entradas e saídas. Geralmente as saídas são fornecidas por algum especialista na tarefa executada. Já o aprendizado por reforço se baseia no aprendizado animal que ocorre através de tentativas e erros, isto é, o aprendizado é realizado através da interação com o ambiente. Enquanto o aprendizado supervisionado depende do especialista, através do conjunto de exemplos de entradas em saídas, o aprendizado por reforço não precisa de um "supervisor"e aprende através de suas próprias experiências.

Pode-se dizer que o objetivo do aprendizado por reforço é o desenvolvimento de um agente capaz de tomar decisões e interagir com seu ambiente com base em suas observações do ambiente. A escolha da decisão a ser tomada a cada instante é feita de forma a garantir a maximização de uma certa função objetivo, que atribui recompensa a estados favoráveis para o agente.

Através dos anos várias aplicações bem sucedidas de aprendizado por reforço tem aparecido em diferentes áreas como por exemplo na robótica (1) e (2), em controle de helicópteros (3), na interação humano-computador (4), em jogos como gamão (5), *videogames* (6) e go (7), dentre outras aplicações.

Os principais métodos de aprendizado por reforço se baseiam na estimativa da função $Q(s, a)$, que consiste da recompensa total que o agente espera receber seguindo uma política de decisões de interesse ao iniciar suas observações no estado $s$ tomando a ação $a$. Esta função pode ser usada para decidir qual a melhor ação que poderá ser

tomada a cada estado $s$. Um dos problemas relacionados ao aprendizado por reforço diz respeito a representação desta função $Q(s, a)$ computacionalmente para ser utilizada durante o processo de aprendizado. Em situações mais simples, em que o número de configurações possíveis para a representação do estado é pequena, é possível representar $Q(s, a)$ através de uma tabela. Contudo, quando este número é grande, a representação de $Q(s, a)$ através de uma tabela se torna inviável. Este problema é geralmente conhecido como a "maldição da dimensionalidade". Este problema é agravado em situações em que o estado é representado por variáveis que tomam valores contínuos. Isso porque a discretização adequada da representação do estado pode levar a problemas cuja tabela de representação de $Q(s, a)$ é muito grande. Desta forma, o preenchimento dos valores adequados na tabela para todos os pares $(s, a)$ levaria muito tempo.

Uma abordagem muito usada para tratar deste problema é a utilização de métodos de regressão para a representação da função $Q(s, a)$. Dentre as motivações para a utilização de métodos de regressão está o fato da representação ser eficiente em relação ao consumo de memória e também em relação ao processo de aprendizagem, já que o agente não precisa preencher cada célula da tabela com valores. Isto é, através da regressão valores de estados próximos compartilham informação sobre $Q(s, a)$, mesmo quando estes estados não foram visitados pelo agente durante o processo de aprendizagem. Redes neurais artificiais (ANN) são usadas como método de regressão devido a sua capacidade de representar funções não-lineares. Em (8) o autor apresenta o *Neural Fitted Q Iteration* (NFQ) que é um algoritmo baseado no *Q-Learning* que utiliza uma ANN para realizar a aproximação da função.

Neste trabalho será analisada a possibilidade de substituição de redes neurais artificiais por métodos de aproximação de funções com base em Regressão por Vetores Suporte (SVR) (9).

## 1.1 Descrição do Problema

Algoritmos de aprendizado por reforço que utilizam métodos de aproximação de função dependem que ocorra uma boa aproximação por parte do aproximador para alcançar o aprendizado correto. No caso das redes neurais artificiais, a qualidade da aproximação

depende de sua configuração, isto é, seu número de nós e a disposição deles na rede. Essa dependência torna difícil a escolha dos melhores parâmetros iniciais para o programa. Além disso, redes neurais artificiais podem encontrar ótimos locais e, portanto, não possuem garantia de convergência para a aproximação ótima.

## 1.2 Motivação

Regressão baseada em vetores suporte são muito utilizadas na literatura e são muitas vezes empregadas por depender de um número menor de parâmetros quando comparados com as Redes Neurais Artificiais. E ainda, os métodos que utilizam vetores suporte são mais bem fundamentados matematicamente e possuem algoritmos com garantia de convergência quando algumas condições são atendidas.

## 1.3 Objetivos

Propõe-se um algoritmo semelhante ao *Neural Fitted Q Iteration* entretanto substituindo a rede neural por um algoritmo baseado em Máquina de Vetores Suporte para Regressão, no caso, o *Perceptron* de $\varepsilon$-Raio Fixo (FRP) (10). Este algoritmo é mais bem fundamentado e a sua escolha dos parâmetros inicias é mais simples que os de uma rede neural artificial. Pretende-se comparar o desempenho deste com o algoritmo original.

# 2 Fundamentação Teórica

Este capítulo tem como objetivo formalizar os conceitos apresentados anteriormente, bem como apresentar novos conceitos necessários para o entendimento desses. O capítulo está dividido em duas seções, a primeira seção contém definições acerca de aprendizado por reforço. A segunda contém um detalhamento sobre os algoritmos de regressão empregados neste trabalho.

## 2.1 Aprendizado por Reforço

Nesta seção é definido formalmente o problema do aprendizado por reforço, apresentando o modelo matemático utilizado para a representação destes problemas e as técnicas usadas para resolvê-los.

O aprendizado por reforço é um campo do aprendizado de máquinas com foco na criação de agentes capazes de tomar decisões acertadas em um ambiente sem que se tenha qualquer conhecimento prévio sobre o tal ambiente. Para que ocorra aprendizado é preciso que o agente após perceber o ambiente tome ações e observe a recompensa imediata proveniente da ação e a mudança no ambiente decorrente da ação tomada. O objetivo do agente é maximizar o total de recompensas imediatas recebidas durante a interação com o ambiente. Desta forma, o agente não está sempre interessado em ações que geram a maior recompensa imediata, mas sim em ações que o leve ao maior acúmulo de recompensa a longo prazo.

Neste trabalho serão considerados problemas de aprendizado por reforço a tempo discreto e espaço de estados discretos – note que problemas a tempo contínuo e com estados contínuos podem ser discretizados e tratados como os problemas discretos considerados neste trabalho. Desta forma, pode-se pensar do estado do ambiente no tempo $n \in N$ como sendo uma variável aleatória $S_t$ tomando valor em um conjunto finito $\mathcal{S}$. A cada instante de tempo, o agente pode tomar uma ação $A_t$, que depende do seu estado atual e toma valor em um conjunto finito $\mathcal{A}$. Após cada ação, o agente observa o novo estado

do ambiente $S_{t+1}$ e recebe a recompensa $r(S_t, A_t, S_{t+1})$. A Figura 2.1 representa o ciclo encontrado em problemas desta natureza. O ciclo começa com o agente em um estado $s \in \mathcal{S}$ do ambiente, ao observar o estado o agente escolhe a ação $a \in \mathcal{A}$ a qual o ambiente responde com um novo estado $s' \in \mathcal{S}$ e uma recompensa $r$ e então o ciclo se repete.

Na seção seguinte será descrito um modelo para este tipo de interação que é comumente usado em problemas de aprendizado por reforço.

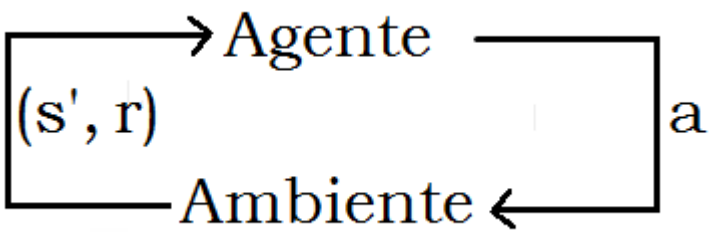



Figura 2.1: Interação entre o agente e o ambiente na aprendizagem por reforço.

### 2.1.1 *Markovian Decision Process*

Um processo de decisão de Markov (MDP) pode ser descrito como um problema de tomada de decisão sequencial que pode sofrer com influencia de fatores aleatórios. Assume-se que o sistema evolui no tempo através de probabilidades independentes da história do processo, dada a informação do estado mais recente do sistema e a última ação empregada. Formalmente, o MDP é representado usando a quádrupla $(\mathcal{S}, \mathcal{A}, p(\cdot,\cdot), r(\cdot,\cdot,\cdot))$ em que $\mathcal{S}$ é o conjunto de estados possíveis, $\mathcal{A}$ é o conjunto de ações, $p(\cdot,\cdot)$ é a probabilidade de transição e $r(\cdot,\cdot,\cdot)$ é a função de recompensa imediata.

Em palavras temos a seguinte descrição do processo: a cada instante de tempo o ambiente se encontra em um estado $s$ que é observado pelo agente para tomar uma ação $a$ dentre as ações possíveis. Quando essa ação é tomada ocorre um passo que chega ao instante de tempo seguinte no qual o ambiente mudou aleatoriamente seguindo uma probabilidade de transição $p(s, a, s')$ que é a probabilidade de sair do estado $s$ e ir para o estado $s'$ ao se tomar a ação $a$. Essa transição resulta em uma recompensa que é uma função denotada $r(s, a, s')$. Um exemplo de MDP é representado no grafo pela Figura 2.2. Esta figura ilustra um MDP com estados $\mathcal{S} = \{S_0, S_1, S_2\}$ e $\mathcal{A} = \{a_0, a_1\}$, em que os números nos arcos representam as probabilidades de transição e as recompensas são

representadas pelas setas amarelas em alguns arcos.

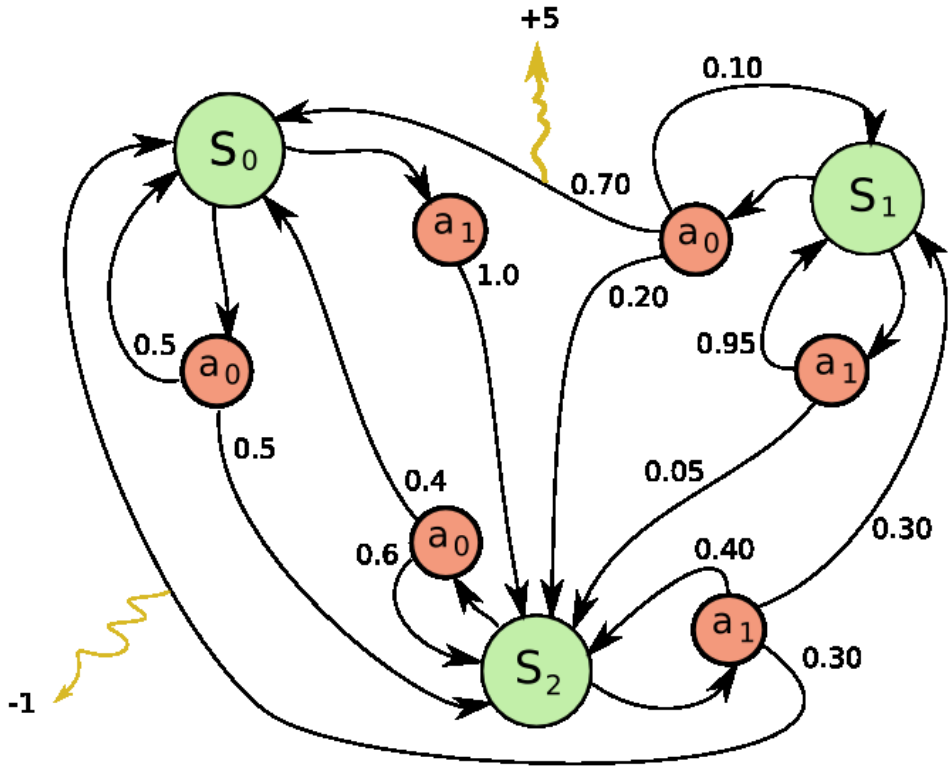



Figura 2.2: Exemplo de um modelo de um processo de decisão de Markov.

O objetivo em se modelar um problema como um MDP é determinar uma política de ações $\pi$ que mapeia estados em ações de forma a maximizar a soma total das recompensas imediatas.

Observe que em um MDP, precisa-se conhecer apenas as probabilidades $p(s, a, s')$ que depende somente de $s$, $a$ e $s'$, ou seja a sequência de estados e ações anteriores aos últimos não precisam ser considerados pelo agente. Desta forma, é possível considerar apenas politicas dadas por $\pi : \mathcal{S} \rightarrow \mathcal{A}$, que independem dos estados e ações passados.

Conhecendo a probabilidade de transição $p(s, a, s')$ é possível através de programação dinâmica determinar a política ótima $\pi^*(s)$ que é uma política para a qual a recompensa esperada $V_\pi(s)$ é máxima. Em um MDP a forma mais comum e abrangente de denotar $V_\pi(s)$ é usando recompensa descontada e considerando o horizonte infinito,

como na Equação (2.1).

$$V_\pi(s) = \mathbb{E}^\pi \left[ \sum_{t=0}^{\infty} \lambda^t r(S_t, A_t, S_{t+1}) \middle| S_0 = s \right], \tag{2.1}$$

onde $s$ é o estado inicial, $\lambda \in (0, 1)$ é um fator de desconto, $r(\cdot, \cdot, \cdot)$ é a função de recompensa, $S_t$ é o estado do ambiente no tempo $n$, e $A_t$ é a ação do agente no estado $S_t$ de acordo com a atual política $\pi$.

Sabe-se que recompensa esperada ótima $V^*(s)$ satisfaz a Equação de Bellman definida na Equação (2.2).

$$V^*(s) = \max_a \left( \sum_{s' \in \mathcal{S}} p(s, a, s')(r(s, a, s') + \lambda V^*(s')) \right), \forall s \in \mathcal{S}. \tag{2.2}$$

Além disso, é possível calcular essa recompensa ótima utilizando o algoritmo de iteração de valor que se encontra descrito no Algoritmo 1. Conhecendo $V^*(s)$ é possível determinar uma política ótima $\pi^*$ de acordo com a Equação (2.3).

$$\pi^*(s) = \operatorname*{argmax}_a \left( \sum_{s' \in \mathcal{S}} p(s, a, s')(r(s, a, s') + \lambda V^*(s')) \right). \tag{2.3}$$

**Algoritmo 1:** Iteração de Valor

**begin**
  Inicialize $V(s)$ aleatoriamente
  **repeat** até convergir
    **repeat** para cada $s \in \mathcal{S}$
      $V(s) = \max_{a \in \mathcal{A}} \left( \sum_{s' \in \mathcal{S}} p(s, a, s')(r(s, a, s') + \lambda V^*(s')) \right)$
**end**

Porém em problemas de aprendizado por reforço não se tem conhecimento prévio do ambiente por isso a probabilidade de transição $p(s, a, s')$ é desconhecida e deve-se utilizar a interação do agente com o ambiente para buscar pela política ótima $\pi^*$. A seguir é apresentada uma dessas técnicas: o *Q-Learning*.

### 2.1.2 *Q-Learning*

Antes de apresentar o *Q-Learning* (11) é necessário introduzir a função $Q^{\pi}(s,a)$ que é conhecida como função ação-valor. $Q^{\pi}(s,a)$ é a recompensa acumulada ao se tomar a ação $a$ no estado $s$ e a partir do estado seguinte escolher as ações como determinado pela política $\pi$ e seu calculo é dado na Equação (2.4).

$$Q^{\pi}(s,a) = \mathbb{E}^{\pi}\left[\sum_{t=0}^{\infty} \lambda^{t} r(S_t, A_t, S_{t+1}) \middle| S_0 = s, A_0 = a\right]. \tag{2.4}$$

A função ação-valor ótima $Q^{*}(s,a)$ é uma função ação-valor que após tomar a ação $a$ no estado $s$ e a partir do estado seguinte segue uma política ótima $\pi^{*}$. Pode-se determinar uma política ótima $\pi^{*}$(a política que maximiza a recompensa total esperada $V_{\pi}(x)$, dentre a classe de políticas $\pi$) de modo similar ao que é feito utilizando $V^{*}(s)$, i.e., escolhendo a ação que leva ao maior $Q^{*}(s,a)$, ou seja, $\pi^{*}(s) = \underset{a}{\text{argmax}}(Q^{*}(s,a))$.

Como esperado, o objetivo do *Q-Learning* é chegar à função $Q^{*}(s,a)$ e em sequência usar esse valor para encontrar a política ótima $\pi^{*}$. Para tanto parte-se do algoritmo de iteração de valor descrito anteriormente para chegar a uma regra de atualização que pode ser aplicada após cada interação com o ambiente.

Primeiramente vamos escrever as iterações realizadas no algoritmo de iteração de valor como a Equação (2.5):

$$V_{n+1}(s) = \underset{a}{max}\left(\sum_{s' \in \mathcal{S}} p(s,a,s')(r(s,a,s') + \lambda V_n(s'))\right). \tag{2.5}$$

Então através de manipulação algébricas pode-se reescrever $Q_n(s,a)$ que é a função ação-valor na iteração $n$ como mostrado na Equação (2.6):

$$\begin{aligned} Q_n(s,a) &= \mathbb{E}\left[\sum_{i=0}^{\infty} \lambda^{i} r(S_i, A_i, S_{i+1}) \middle| S_0 = s, A_0 = a\right] \\ &= \sum_{s' \in \mathcal{S}} p(s,a,s')(r(s,a,s') + \lambda V_n(s')). \end{aligned} \tag{2.6}$$

Perceba que pode-se colocar $V_{n+1}(s)$ em função de $Q_n(s,a)$ como na Equação (2.7):

$$V_{n+1}(s) = \max_{a \in \mathcal{A}} \left( Q_n(s,a) \right). \tag{2.7}$$

E então, é possível colocar $Q_{n+1}(s,a)$ em função de $Q_n(s,a)$ conforme a Equação (2.8):

$$\begin{aligned} Q_{n+1}(s,a) &= \sum_{s' \in \mathcal{S}} p(s,a,s') \left( r(s,a,s') + \lambda \max_{a' \in \mathcal{A}} \left( Q_n(s',a') \right) \right) \\ &= \mathbb{E}\left[ r(s,a,S_1) + \lambda \max_{a' \in \mathcal{A}} Q_n(S_1,a') \middle| S_0 = s, A_0 = a \right], \end{aligned} \tag{2.8}$$

em que $S_1$ representa uma variável aleatória representando o estado do sistema após o par $(s,a)$. Por fim, pode-se utilizar a Equação (2.9) como aproximação para a esperança na Equação (2.8).

$$Q_{n+1}(S_t,A_t) = (1-\alpha) Q_n(S_t,A_t) + \alpha [ r(S_t,A_t,S_{t+1}) + \lambda \max_{a \in \mathcal{A}} Q_n(S_{t+1},a) ], \tag{2.9}$$

onde $\alpha$ é uma taxa de aprendizado que deve estar no intervalo $(0,1)$, e $(S_t, A_t, S_{t+1})$ são variáveis aleatórias representando a sequencia de estado, ação e estado observada durante a interação do agente com o meio.

É possível mostrar que $Q_n$ sempre irá convergir para $Q^*$ após várias atualizações para todos os pares estado-ação sob algumas condições sobre $\alpha$ (12). A convergência é independente da política usada para gerar os pares estado-ação, algoritmos com esta característica são denominados *off-police*. O *Q-Learning* é apresentado como o Algoritmo 2.

Como $Q$ deve ser aproximado para todos os pares estado-ação a forma mais simples de representá-lo é através de uma matriz. Porém para casos com um grande número de estados ou ações a representação de $Q$ como uma matriz torna-se custosa tanto em relação à memória quanto em relação ao processamento requerido para iterar um número suficiente de vezes sobre cada par estado-ação. Além disso, para casos onde o estado é contínuo seria necessária a discretização do espaço contínuo gerando aproximações e, dependendo da discretização, levando ao problema de ter um número muito grande de estados.

**Algoritmo 2:** *Q-Learning*

**begin**
  Inicialize $Q(s, a)$ aleatoriamente, $\forall s \in \mathcal{S}, \forall a \in \mathcal{A}$
  **for** $s' \in$ *Estados finais* **do**
    $Q(s', \cdot) \longleftarrow 0$
  **end for**
  **repeat** para cada episódio
    Inicialize $s$
    **repeat** para cada passo
      $a \longleftarrow \pi(s)$ `/* qualquer` $\pi$ `mas apresenta melhores resultados quando` $\pi$ `explora` $Q$ `de forma` $\varepsilon$`-gulosa */`

      Tome a ação $a$ e observe $s'$ e $r(s, a, s')$
      $Q(s, a) \longleftarrow Q(s, a) + \alpha(r(s, a, s') + \lambda \underset{a'}{max} Q(s', a') - Q(s, a))$
      $s \longleftarrow s'$

**end**

Tais problemas podem ser contornados através da representação da função $Q$ por um aproximador de função. Na Subseção 2.1.3, a seguir, é apresentado um algoritmo que utiliza um algoritmo de regressão para aproximar $Q$.

### 2.1.3 *Fitted Q Iteration*

A partir do trabalho realizado em (13) foi derivado o algoritmo *Fitted Q Iteration* (14) que pode ser usado em conjunto com qualquer algoritmo de regressão para aproximar a função $Q$. Nessa abordagem o agente não está interagindo diretamente com o ambiente mas aproximando $Q$ a partir de um conjunto de experiências previamente coletadas a partir de interação com o ambiente em quadruplas $(s, a, r, s')$. Desta forma é possível utilizar técnicas de aprendizado em lote.

Na primeira iteração o algoritmo de regressão recebe como entrada os pares estado $s$ e ação $a$ e tem como objetivo aproximar as recompensas imediatas, chama-se a função aproximada resultante de $Q_1$. Cada iteração seguinte utiliza o $Q_k$ calculado anteriormente para recalcular as novas recompensas alterando o valor a ser aproximado. Desta forma, é importante que o algoritmo aprendizado supervisionado convirja corretamente de forma a não propagar erros para a próxima iteração.

O Algoritmo 3 é uma versão geral do *Fitted Q Iteration* para um algoritmo de regressão qualquer.

**Algoritmo 3:** *Fitted Q Iteration*

**Data**: Um conjunto $D$ de quadruplas de transição
**Result**: A função $Q$ após a n-ésima iteração sobre o conjunto $D$
**begin**
  $k \longleftarrow 0$
  $Q_0 \longleftarrow new\ Regressor\_Algorithm()$
  **while** $k < n$ **do**
    $generate\_pattern\_set\ P = \{(input^l, target^l), l = 1, ..., \#D\}\ where:$
      $input^l = s^l, a^l,$
      $target^l = r(s^l, a^l, s'^l) + \lambda \max_{a'} Q_k(s'^l, a')$
    $Q_{k+1} \longleftarrow train(P)$
    $k \longleftarrow k + 1$
  **end while**
  **return** $Q_n$
**end**

Alguns dos algoritmos de regressão de interesse para esse trabalho são apresentados no Capítulo 2.2, a seguir.

## 2.2 Algoritmos de Regressão

Nesta seção são apresentadas técnicas de aprendizado supervisionado que podem ser usadas para aproximar a função $Q$ através do algoritmo *Fitted Q Iteration*.

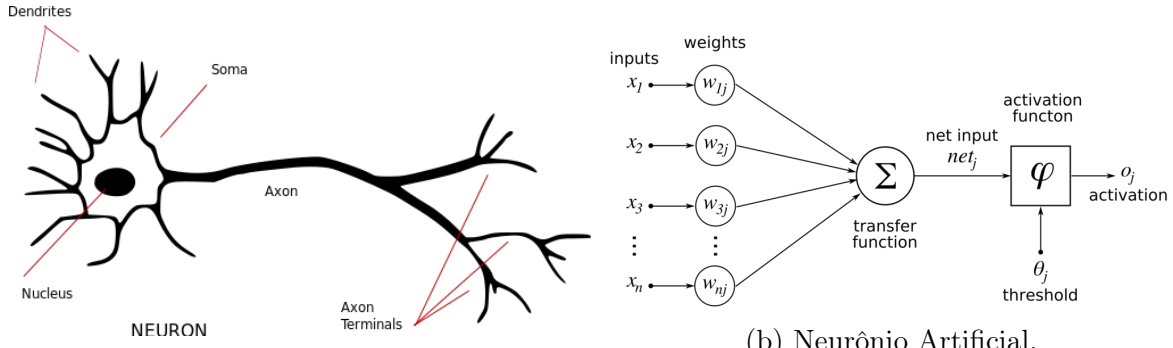
### 2.2.1 Rede Neural Artificial

Rede Neural Artificial (ANN) é um modelo matemático bioinspirado capaz de aproximar funções não-lineares através de aprendizado supervisionado. Assim como as redes neurais biológicas são formadas por neurônios as redes neurais artificiais são formadas por neurônios artificiais.

Para entendermos uma ANN precisamos entender o neurônio artificial que também é conhecido como *Perceptron* (15). A Figura 2.3 é uma comparação lado a lado entre um neurônio biológico e um neurônio artificial. Da mesma forma que os dendritos recebem sinais elétricos, os modificam de acordo com cada dendrito e, por fim, os transmitem ao soma as entradas de um neurônio artificial recebem um vetor $\boldsymbol{x}$, multiplicam-no pelo

peso das entradas denotado pelo vetor $\boldsymbol{w}$ e, enfim, o transmite ao somatório. Uma das entradas do neurônio artificial é sempre 1, seu peso é denominado *bias* e ele é denotado por $b$, a função do *bias* é fazer com que o resultado do somatório não dependa somente do produto linear entre os vetores $\boldsymbol{x}$ e $\boldsymbol{w}$. O somatório emula a função do soma em um neurônio biológico, que é acumular os valores dos sinais modificados. Por último, a saída é o valor acumulado transformado por uma função de ativação $\phi(\cdot)$, assim como o axônio transmite sinais de acordo com o potencial elétrico acumulado no soma. A operação realizada por um neurônio pode ser descrita pela Equação (2.10):

$$o = \phi\left(\boldsymbol{w} \cdot \boldsymbol{x} + b\right). \tag{2.10}$$




(a) Neurônio Biológico.

(b) Neurônio Artificial.

Figura 2.3: Comparação lado a lado entre um neurônio biológico e um neurônio artificial.

Cada ANN possui uma topologia que define o número de neurônios e suas ligações, neste trabalho é usado o *Multilayer Perceptron* que é um tipo de rede neural artificial cujas ligações ocorrem de forma a dividir a rede em camadas nas quais a entrada de cada neurônio de uma camada é a saída de todos os neurônios da camada anterior e a camada de entrada é o vetor de valores de entrada. A Figura 2.4 apresenta um exemplo dessa topologia. Os 3 círculos pretos são neurônios da camada de entrada, a segunda camada tem 3 neurônios e é uma camada oculta (todas as camadas que não são a camada de entrada ou a de saída são conhecidas como camadas ocultas), a terceira e última camada é formada por 2 neurônios e é conhecida como camada de saída.

O processo de treinamento de uma rede neural é feito da seguinte forma: recebe-se um conjunto de treinamento $\{(\boldsymbol{x_j}, \boldsymbol{y_j}) : j = 1, \ldots, m\}$, onde $\boldsymbol{x_j} \in \mathbb{R}^n$ e $\boldsymbol{y_j} \in \mathbb{R}^l$, onde $l$ é o número de nós de saída da rede neural. Em seguida o erro médio quadrático para os

pontos de treinamento é calculado usando Equação (2.11):

$$E(\boldsymbol{w}) = \frac{1}{m}\sum_{j=1}^{m} |o_j(\boldsymbol{w}) - \boldsymbol{y_j}|^2 , \tag{2.11}$$

onde $o_j(\boldsymbol{w})$ representa o resultado de $\boldsymbol{x_j}$ aplicado na rede para os pesos $\boldsymbol{w}$. O processo de aprendizado é feito minimizando esta função em relação aos parâmetros $\boldsymbol{w}$, geralmente usando a técnica de *Backpropagation* (16). Nesta técnica, o erro calculado nas camadas de saída são propagados para os nós internos da rede.

Neste trabalho foi usada a técnica *Rprop* apresentada em (17). Neste algoritmo, usa-se somente informação do sinal do gradiente para atualizar os pesos e o tamanho do passo é adaptado de forma a evitar passar por mínimos locais.

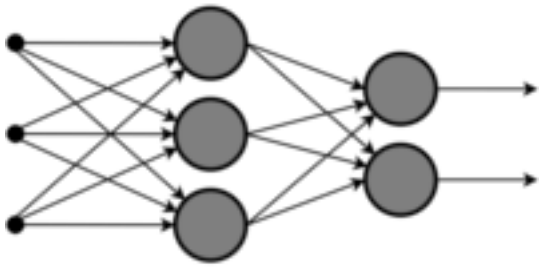


Figura 2.4: *Multilayer Perceptron.*

### 2.2.2 *Support Vector Regressor*

*Support Vector Regressor* é um algoritmo de regressão linear baseado em vetores suporte apresentado em (18). Neste algoritmo, o objetivo é ajustar os valores de treinamento $\{(\boldsymbol{x_i}, y_i) : j = 1, \ldots, m\}$ à uma função linear como na Equação (2.12):

$$f(\boldsymbol{x}) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b, \tag{2.12}$$

em que $\boldsymbol{w} \in \mathbb{R}^n$ são parâmetros. O ajuste é feito de forma que o erro ponto a ponto não seja maior do que um valor pré-estipulado $\varepsilon > 0$. Dessa maneira serão encontrados pesos $\boldsymbol{w} \in \mathbb{R}^n$ que satisfazem a Equação (2.13):

$$|y_i - f(\boldsymbol{x_i})| < \varepsilon \tag{2.13}$$

para todo $i = 1, \ldots, m$. Para ter um efeito de regularização, a norma $\ell_2$ do vetor $\boldsymbol{w}$ é minimizada.

O problema é formulado como o problema de otimização representado na Equação (2.14):

$$\begin{aligned} &\text{minimizar} && \frac{1}{2}||\boldsymbol{w}||^2 \\ &\text{sujeito a} && y_i - \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b \leqslant \epsilon \\ & && \langle \boldsymbol{w}, \boldsymbol{x_i} \rangle + b - y_i \leqslant \epsilon \end{aligned} \tag{2.14}$$

Este problema de regressão é relacionado ao problema de classificação de margem máxima como visto em (9).

A formulação dual deste problema pode ser escrita como a Equação (2.15):

$$\begin{aligned} &\text{maximizar} && -\frac{1}{2}\sum_{i,j=1}^{m}\left(\alpha_i - \alpha_i^*\right)\left(\alpha_j - \alpha_j^*\right)\langle \boldsymbol{x_i}, \boldsymbol{x_j} \rangle \\ & && -\varepsilon\sum_{i=1}^{l}\left(\alpha_i + \alpha_i^*\right) + \sum_{i=1}^{l} y_i\left(\alpha_i - \alpha_i^*\right) \\ &\text{sujeito a} && \sum_{i=1}^{l} y_i\left(\alpha_i - \alpha_i^*\right) = 0 \\ & && \alpha_i, \alpha_j \geq 0 \end{aligned} \tag{2.15}$$

Utilizando a formulação dual apresentada na Equação (2.15), é possível aplicar o chamado "*kernel trick*". O *kernel trick* consiste em tornar um problema não-linear em um problema linear ao mapear a entrada em um espaço de dimensão mais elevada, denominado espaço de características, e nesse espaço realiza a aproximação linear. O processo de mapeamento pode ser visto na Figura 2.5, observe que os dados não são linearmente separáveis no espaço de entradas $\mathbb{R}^2$ entretanto passam a ser lineares após serem mapeados pelo *kernel* para o estado de características $\mathbb{R}^3$.

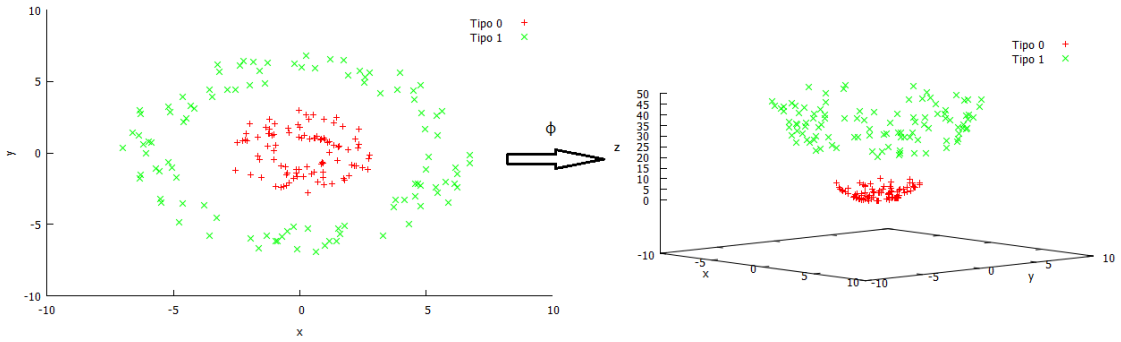



Figura 2.5: Um *kernel* mapeando pontos de $\mathbb{R}^2$ para $\mathbb{R}^3$ em um exemplo de problema de classificação.

Essa dimensão elevada depende do *kernel* usado e é possível usar um *kernel* com dimensão infinita como é o caso do *Kernel* Gaussiano descrito na Equação (2.16).

$$k(\mathbf{x}, \mathbf{x_i}) = exp(-\gamma \left\|\mathbf{x} - \mathbf{x_i}\right\|), \tag{2.16}$$

em que $\lambda > 0$ é um parâmetro.

### 2.2.3 *Perceptron* de $\varepsilon$-Raio Fixo

O *Perceptron* de $\varepsilon$-Raio Fixo (10) é um algoritmo aproximador de funções não-lineares baseado em SVR. Neste algoritmo assim como na SVR, o objetivo é determinar uma função linear como na Equação (2.17):

$$f(x) = \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b, \tag{2.17}$$

de forma que tenha-se: $|y_i - f(\boldsymbol{x_i})| < \varepsilon$ para todo $i = 1, \ldots, m$ para um parâmetro $\varepsilon > 0$ dado. Em contraste com o SVR, este algoritmo define a função de erro apresentada na Equação (2.18):

$$J(\boldsymbol{w}) = \sum_{i \in M_\varepsilon} \left| \langle \boldsymbol{w}, \boldsymbol{x} \rangle + b - y_i \right|^2, \tag{2.18}$$

em que $M_\varepsilon = \{i = 1, \ldots, m | \, |\langle \boldsymbol{w}, \boldsymbol{x} \rangle + b - y_i| < \varepsilon\}$.

A partir desta função de erro, um método de gradiente estocástico – similar ao usado no algoritmo *Perceptron* – é utilizado para minimizar esta função. Isto é, a cada

iteração do algoritmo escolhe-se um ponto de treinamento e atualizam-se os pesos:

$$\begin{aligned} w_j &\leftarrow w_j + \eta \frac{\partial}{\partial w_j} J(\boldsymbol{w}) \qquad j = 1, \ldots, m \\ b &\leftarrow b + \eta \frac{\partial}{\partial b} J(\boldsymbol{w}). \end{aligned} \tag{2.19}$$

O algoritmo é repetido até que $M_\varepsilon = \emptyset$.

O *kernel trick* também é aplicável na forma dual deste algoritmo, sendo portanto capaz de representar funções não-lineares. É importante mencionar também que o *Incremental Strategy Algorithm* (ISA) é empregado para identificar os pontos suportes mais facilmente, como sugerido em (10). Nesta abordagem o valor de $\varepsilon$ é decrescido sucessivamente até o valor desejado.

É importante notar que em (10) o autor encontrou resultados melhores ao começar a aproximação com um $\varepsilon$ maior e diminuí-lo progressivamente ao longo do tempo. Uma abordagem similar foi realizada neste trabalho.

# 3 Implementação

Nesta seção é apresentado o *framework* desenvolvido em C++ para facilitar a aplicação das técnicas de aprendizado por reforço em diferentes ambientes. Neste *framework* destacam-se as classes: *regressor*, *game*, *sample* e *dataset*.

A classe *regressor* é uma classe abstrata usada para representar o aproximador de função e, para tanto, possui métodos de treinamento e cálculo do valor da função em um ponto. Outros métodos desta classe servem para escrita e leitura em arquivo e para leitura do arquivo de configuração. O treinamento realizado pela classe *regressor* depende de um conjunto de instâncias da classe *sample* que representam transições obtidas durante a simulação do ambiente na forma $(s, a, r, s')$ em que $s$ é o estado atual, $a$ é a ação tomada, $r$ é a recompensa imediata e $s'$ é o estado seguinte. Esse conjunto é representado pela classe *dataset*.

Por sua vez, o simulador do ambiente é representado pela classe abstrata *game* que, assim como a classe *regressor*, também é configurável através de um arquivo. As classes derivadas de *game* devem implementar três métodos os quais cada um representa um modo de execução, a saber:

- O primeiro é o modo de apresentação no qual, dado um *regressor*, são realizados vários episódios no ambiente seguindo a política gulosa. Ou seja, a política que seleciona sempre a ação com maior recompensa acumulada de acordo com o valor de saída do *regressor* para o estado atual. Este modo é usado para calcular a qualidade do aproximador de função.

- O segundo é o modo de exploração que depende de um parâmetro $\varepsilon$, que é um valor no intervalo $[0, 1]$ e de um *regressor*. Sua execução é semelhante ao modo de apresentação porém usando uma política $\varepsilon$-gulosa que com probabilidade $1 - \varepsilon$ é idêntica a política gulosa e com probabilidade $\varepsilon$ é uma política aleatória uniforme. Isto é, uma politica que tem a mesma probabilidade de escolher qualquer ação. Ao final dos episódios é gerado um *dataset* que é utilizado para treinar o *regressor*.

- O terceiro é o modo especialista que deve permitir à pessoa controlar as ações do agente durante vários episódios no ambiente. Pode-se utilizar este modo para testar o ambiente ou para gerar um *dataset* que pode ser usado futuramente para treinar um *regressor*.

Ainda foi criada uma função que com um *sample* e um *regressor* determina a melhor ação de acordo com o *regressor* e o estado seguinte do *sample*. Essa função é necessária pois em alguns ambientes deve-se minimizar o custo e em outros maximizar a recompensa. Além disso, se encontram no *framework* implementações genéricas da Rede Neural Artificial e do *Perceptron* de $\varepsilon$-Raio Fixo que são usadas pelas classes derivadas da classe *regressor*.

A Figura 3.1 contém um diagrama de classe do *framework* construído nesse trabalho.

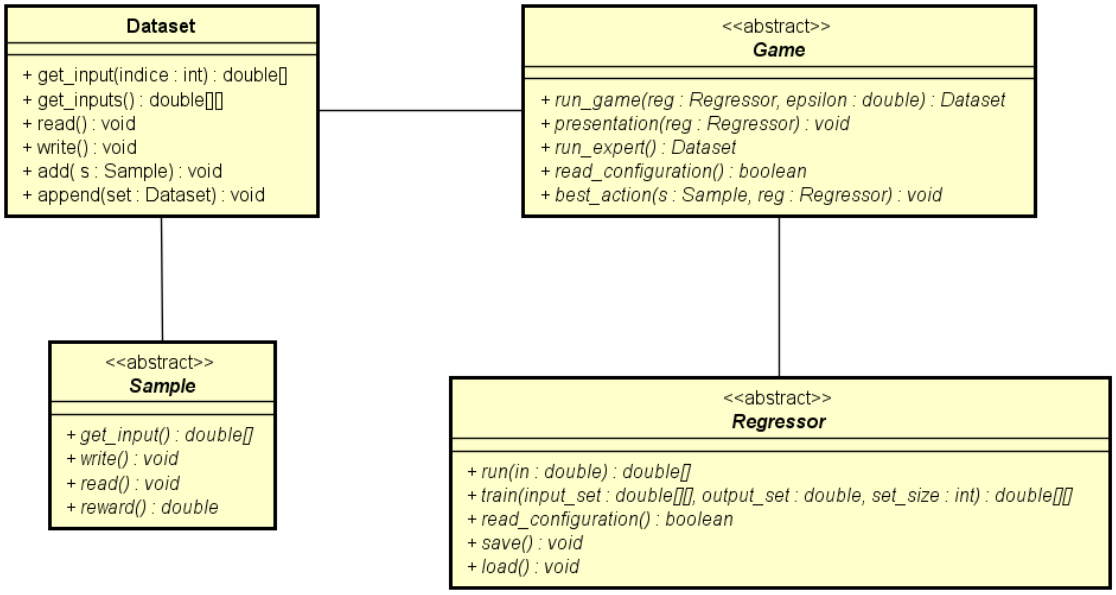



Figura 3.1: Diagrama de classes do *framework* usado no projeto.

# 4 Resultados

Este capítulo apresenta os resultados de testes comparativos entre algoritmos *Fitted Q Iteration* utilizando diferentes algoritmos de regressão (no caso a Rede Neural Artificial e o *Perceptron* de $\varepsilon$-Raio Fixo), a análise desses resultados e a metodologia usada para a obtenção dos mesmos.

## 4.1 Problemas de *Benchmark*

A fim de comparar o impacto dos algoritmos de regressão no aprendizado e analisar seus pontos fortes e fracos foi necessário implementar diferentes cenários para o aprendizado. Nesta seção são apresentados os 3 cenários utilizados nos resultados realizados por este trabalho, a saber: *Gridworld*, *Mountain Car* e *Pole Balancing*.

### 4.1.1 *Gridworld*

O problema denominado *Gridworld* neste trabalho foi apresentado por (19) como um problema simples porém que apresenta um desafio de aprendizado para métodos de aprendizado por reforço que utilizam um aproximador de função. Isso ocorre porque pequenos erros de aproximação se acumulam de uma iteração para a próxima e que no fim resultam em uma grande diferença ou na divergência do aproximador de função.

O problema consiste em navegar por um espaço bidimensional discretizado por uma malha 20x20 com o objetivo de alcançar o canto superior direito, quando isto ocorre o episódio acaba. A representação do estado é um ponto $(x, y) \in [0, 1]^2$, as ações possíveis são apenas 4, que correspondem a um passo de 0,05 em cada uma das direções cardeais. Cada ação resulta em um custo imediato igual a 0,5. O estado inicial deste problema é escolhido aleatoriamente entre todos os estados possíveis. A Figura 4.1 é o custo esperado total deste problema.

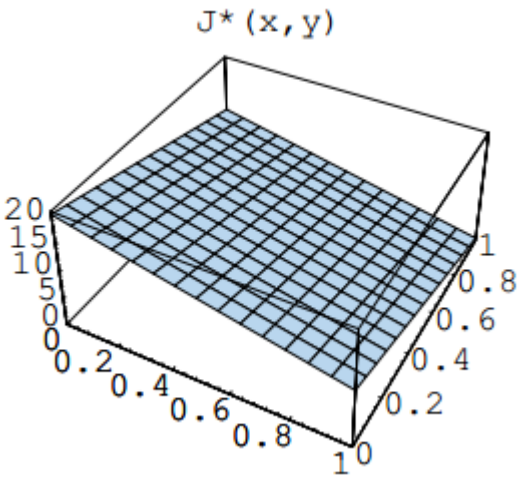



Figura 4.1: Custo acumulado do problema *Gridworld*

### 4.1.2 *Mountain Car*

O *Mountain Car* (20) é um problema com o estado composto de variáveis contínuas. Neste problema o agente deve controlar um carro para que ele chegue ao topo de uma montanha. No entanto há uma dificuldade o motor do carro não é forte o suficiente para acelerar diretamente até o objetivo. Porém um agente inteligente pode usar a gravidade a seu favor usando a ré para subir a colina anterior e então acelerar o máximo possível para frente usando a inércia para alcançar o objetivo. A Figura 4.2 pode ser útil para o entendimento do problema.

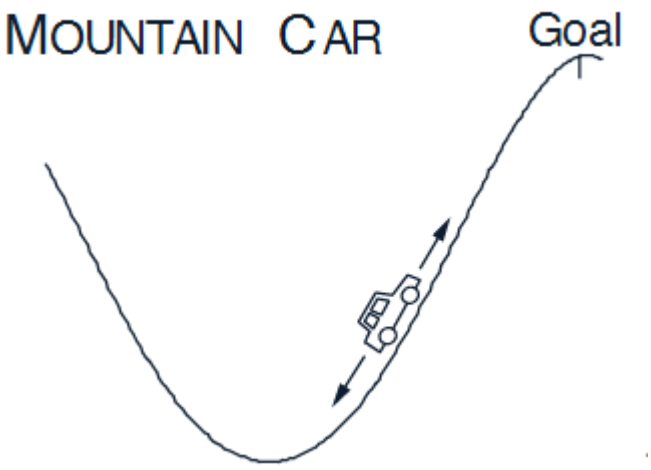



Figura 4.2: Imagem explicativa do Mountain Car

Para simplificar o problema, apenas 3 ações são possíveis: aceleração para trás ($a = -1$), não acelerar ($a = 0$) e aceleração para frente ($a = 1$). Por sua vez a representa-

ção do estado possui duas variáveis contínuas $x_t$ e $x'_t$ que são, respectivamente, a posição do carro e sua velocidade no tempo $t$. A transição se dá pelas seguintes equações:

$$x'_{t+1} = \min(0{,}07,\ \max(-0{,}07,\ x'_t + 0.001a_t - 0{,}0025\cos(3x))) \tag{4.1}$$

$$x_{t+1} = x_t + x'_{t+1} \tag{4.2}$$

Na primeira as funções min e max são usados para garantir que o carro não ultrapasse a velocidade máxima em nenhum dos dois sentidos, ou seja, $-0{,}07 \leq x'_{t+1} \leq 0{,}07$. Uma outra restrição ocorre em relação à $x_{t+1}$, pois caso $x_{t+1} \leq -1{,}2$ faz-se $x_{t+1} = -1{,}2$ e $x'_{t+1} = 0$. Após uma transição se o carro chegou ao objetivo, i.e., $x_{t+1} \geq 0{,}5$ ele recebe um custo imediato igual a zero e o episódio termina caso contrário o custo imediato é 0,01.

### 4.1.3 *Pole Balancing*

O *Pole Balancing* (20) é um outro problema com o estado composto de variáveis contínuas. Neste problema temos um carrinho com uma barra vertical afixada em sua parte superior. Desta forma, quando é aplicada uma força fazendo o carrinho se mover na horizontal o momento faz com que a barra também se movimente, como pode ser visto na Figura 4.3. O objetivo do agente é fazer com que a barra não caia durante a maior quantidade de tempo possível. Isto é, o ângulo da barra com o carrinho não pode ser maior do que um valor pré-estipulado.

Diferente dos dois problemas apresentados anteriormente, cujo objetivo era alcançar uma região do espaço de estados o objetivo deste problema é manter-se dentro de um espaço de estados a maior quantidade de tempo possível. Da mesma forma pode-se notar que nos dois primeiros problemas ao atingir o objetivo o episódio acabava, mas que no caso deste problema o episódio acaba ao deixarmos o espaço de estados objetivos ou ao chegar a um número limite de transições.

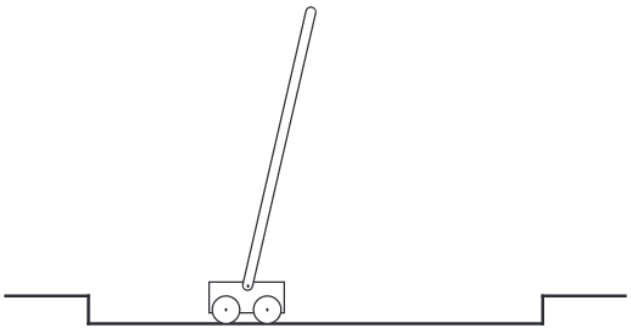


Figura 4.3: Imagem explicativa do Pole Balancing

A representação do estado deste problema contém 4 variáveis: a posição do carrinho $x$, a velocidade do carrinho $x'$, o ângulo de inclinação da barra em relação ao eixo vertical $\theta$ e a velocidade angular $\theta'$. Para este problema apenas duas ações são possíveis: aplicar uma força positiva ou negativa de mesma magnitude ($a \in [-10, 10]$). A transição é dada da mesma forma como descrito na referência utilizando as mesmas constantes. A recompensa imediata é 1 enquanto $|\theta|$ é menor que o ângulo máximo e $|x| \leq 2{,}4$, caso contrário a recompensa imediata é zero e o episódio termina.

## 4.2 Parâmetros

Nesta seção são apresentados os parâmetros da ANN, *Perceptron* de $\varepsilon$-Raio Fixo e os parâmetros gerais usados para realização dos testes. A menos que mencionado o contrário todos os parâmetros usados foram selecionados de modo empírico escolhendo os valores que alcançavam as melhores aproximações.

Os parâmetros da rede neural artificial são:

- o **formato** que é a quantidade de camadas e o número de neurônios em cada camada, é o mesmo usado em (8) portanto em todos os cenários a rede possui 4 camadas sendo a primeira camada composta de um número de neurônios igual ao número de variáveis usadas na representação do estado e das ações do cenário (a saber 6 para o *Gridworld*, 3 para o *Mountain Car* e 5 para o *Pole Balancing*), as duas camadas ocultas possuem 5 neurônios cada e a camada de saída contém um neurônio;

- a **função de ativação** de cada neurônio, nesse caso fez-se distinção entre o neurônio

de saída e os outros, todos os neurônios utilizaram função de ativação sigmoidal exceto o de saída para o qual a função de ativação usada foi sigmoidal para o *Mountain Car* e para o *Pole Balancing* e linear para *Gridworld*.

- o **algoritmo de aprendizado** e seus parâmetros de aprendizado, em todos os casos foi usado o *Rprop* com os parâmetros sugeridos em (17);

- o **erro máximo tolerado** que foi igual a 0,005, em equivalência a margem do *Perceptron* de $\varepsilon$-Raio Fixo.

Os parâmetros do *Perceptron* de $\varepsilon$-Raio Fixo são:

- o ***kernel***, sendo usado em todos os casos o *kernel* gaussiano;

- o **parâmetro** $\gamma$ usado no *kernel* gaussiano, para o *Pole Balancing* foi usado $\gamma = 10$, para o *Mountain Car* usou-se $\gamma = 1$ e para o *Gridworld* fez-se $\gamma = 0{,}5$;

- a **taxa de aprendizado**, em todos os casos foi usado o valor 0,1.

- a **margem** mínima $\varepsilon$ para o qual usou-se o mesmo valor que o erro máximo tolerado da rede neural, isto é, foi usado o valor 0,005 em todos os cenários.

Para ambos os algoritmos de regressão foram executados 250 iterações do laço principal que consiste na interação com o ambiente, aprendizado e teste. Durante a interação com o ambiente apenas 1 episódio é realizado (utilizando a politica $\varepsilon$-gulosa com taxa 1 para o *Gridworld* e 0 para os demais) e então o *dataset* gerado é acumulado aos anteriores. O aprendizado consiste de uma execução do laço interno do Algoritmo 3 no qual o fator de desconto $\lambda$ recebeu 1 para o *Gridworld* e 0,95 para os outros. Após cada iteração de aprendizado foram rodados 1000 episódios de teste para avaliar a performance dos algoritmos durante o processo. Este formato para os experimentos é o mesmo apresentado em (8).

Durante a realização dos experimentos, percebeu-se que o aprendizado ocorre mais rapidamente quando é realizado do início do que quando reaproveitando da regressão previamente realizada na iteração anterior. Para entendermos melhor este problema realizamos alguns testes paralelos ao estudo deste trabalho. O intuito do teste é comparar se é melhor reiniciar a regressão ou não. O teste consiste na aproximação da função

$sinc(x)$ dada abaixo, e posteriormente a aproximação da mesma função deslocada em 0,1 unidades no eixo $y$.

$$sinc(x) = \frac{sin(x)}{x} \tag{4.3}$$

Os resultados podem ser observados na Tabela 4.1. Foi observado que o número de vetores suportes é maior quando não se reinicia a regressão, o que aumenta o tempo de aprendizado, porém esse pode não ser o único motivo pois outros testes indicaram que o mesmo ocorre com a rede neural artificial. Após a analise destes resultados decidiu-se que para ambos os algoritmos a regressão deve ser descartada após ser usada para derivar a política durante a interação com o ambiente. No próximo capítulo essa escolha será discutida em meio aos outros resultados.

Tabela 4.1: Média de 9 regressões da *sinc* deslocada partindo da regressão da *sinc* e partindo do zero.

| | Tempo | Erro quadrático máximo | Vetores suportes |
|---|---|---|---|
| Não reiniciando | 70070 $ms$ | 0,553 | 908,0 |
| Reiniciando | **319 ms** | **0,010** | **384,1** |

## 4.3 Experimentos e Resultados

Utilizando os parâmetros descritos na seção anterior foram executados 6 aprendizados em cada um dos três cenários descritos na Seção 4.1 para cada um dos algoritmos de regressão, com exceção cenário do *Gridworld* utilizando a rede neural artificial que por motivos de tempo fez-se apenas 5 execuções. Em cada execução o algoritmo de regressão inicia com seus pesos possuindo valores aleatórios. Estes testes foram feitos em um computador Linux com um processador Intel® Core™ i7-2630QM CPU @ 2.00GHz e 6 GB de memória RAM.

A tabela 4.2 mostra a média dos melhores resultados obtidos durante o aprendizado e a média do número de iterações até encontrar o melhor resultado. Através dessa tabela podemos notar que existe um certo balanço entre o uso da Rede Neural Artificial e do *Perceptron* de $\varepsilon$-Raio Fixo no *Fitted Q Iteration* pois no cenário *Mountain Car* o primeiro se saiu melhor e nos cenários *Gridworld* e *Pole Balancing* o segundo foi o melhor.

Ambos os algoritmo alcançaram bons resultados com exceção da Rede Neural no cenário *Pole Balancing*.

Tabela 4.2: Média dos melhores resultados encontrados e do número de iterações médio.

| | *Gridworld* | | *Mountain Car* | | *Pole Balancing* | |
|---|---|---|---|---|---|---|
| | Vitórias | Iter. | Vitórias | Iter. | Vitórias | Iter. |
| Rede Neural Artificial | $85,46\%$ | $\mathbf{30,40}$ | $\mathbf{99,95}\%$ | $126,83$ | $33,42\%$ | $\mathbf{38,67}$ |
| *Perceptron* de $\varepsilon$-Raio Fixo | $\mathbf{100}\%$ | $30,50$ | $83,07\%$ | $\mathbf{52,17}$ | $\mathbf{89,63}\%$ | $187,67$ |

Nas Figuras 4.4, 4.5 e 4.6 são exibidas comparações ao longo de um aprendizado entre a Rede Neural Artificial e o *Perceptron* de $\varepsilon$-Raio Fixo. Esses resultados são da primeira execução de cada instância em cada cenário. Observam-se grandes quedas do número de vitórias após algumas iterações isso é atribuído ao reinicio discutido na seção anterior.

Pode-se conjecturar pela Figura 4.4 que não é devido ao reinicio que os algoritmos falham em alguns cenários pois o número de vitórias volta ao valor obtido anteriormente em iterações posteriores a uma queda.

Também há de se notar que para cenários em que o aprendizado ocorre corretamente – como é o caso do *Perceptron* de $\varepsilon$-Raio Fixo no *Gridworld* mostrado na Figura 4.4 – a quantidade de quedas é menor que em cenários nos quais o aprendizado não se completa – como ocorre com o *Perceptron* de $\varepsilon$-Raio Fixo no *Pole Balancing* como mostrado na Figura 4.4.

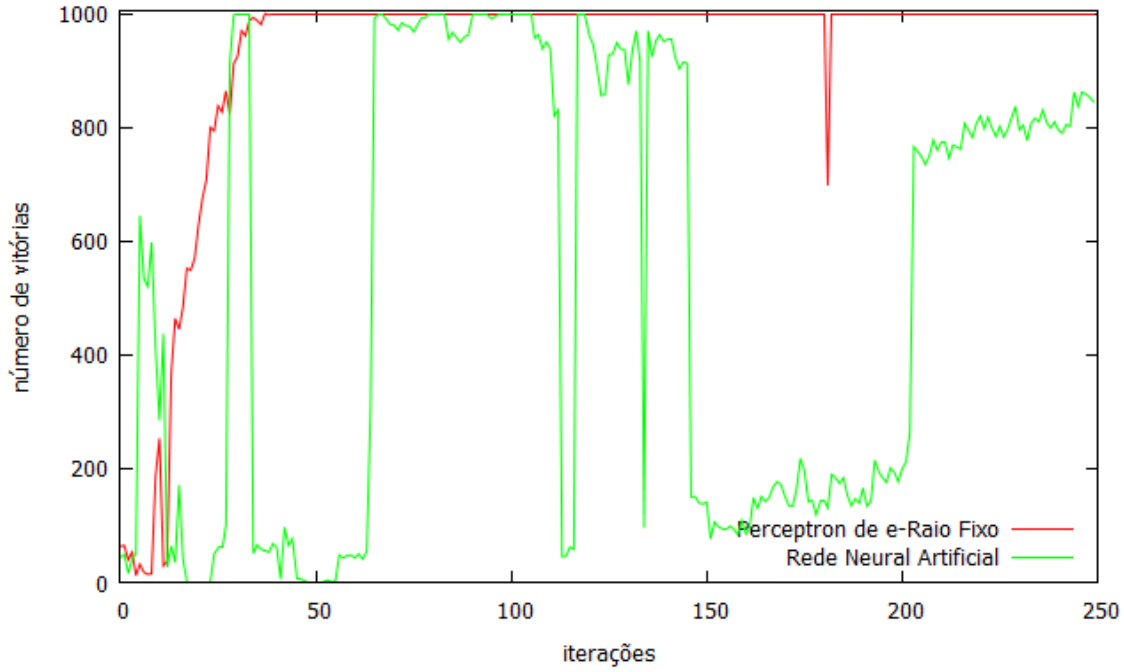



Figura 4.4: Gráfico do número de acertos sobre o tempo para o cenário *Gridworld*.

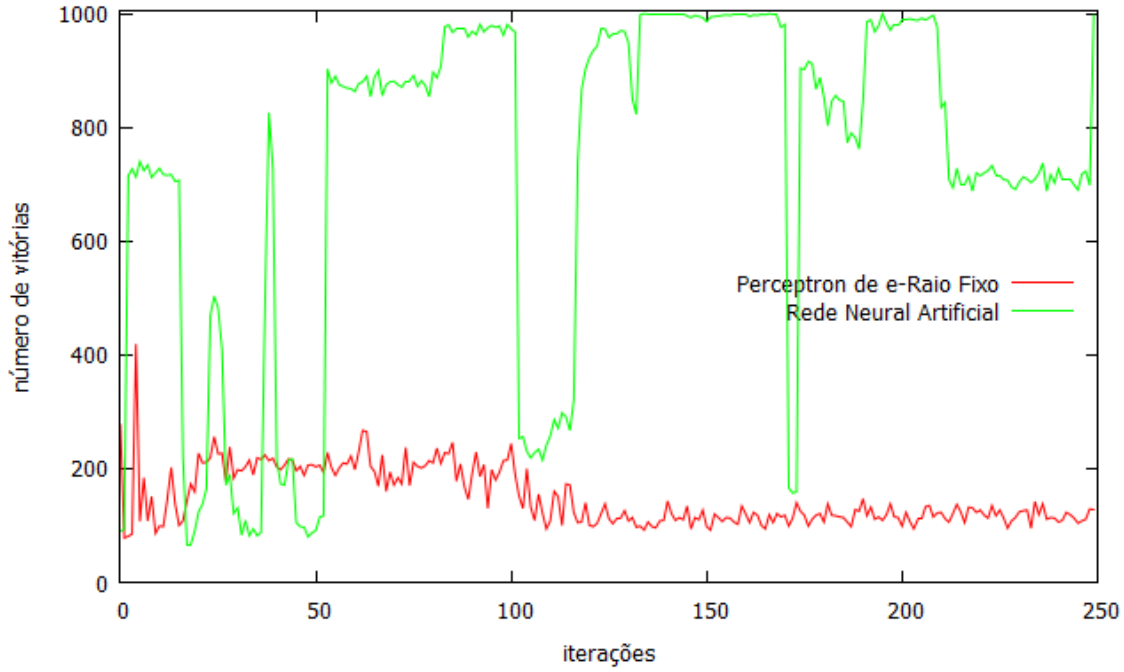



Figura 4.5: Gráfico do número de acertos sobre o tempo para o cenário *Mountain Car*.

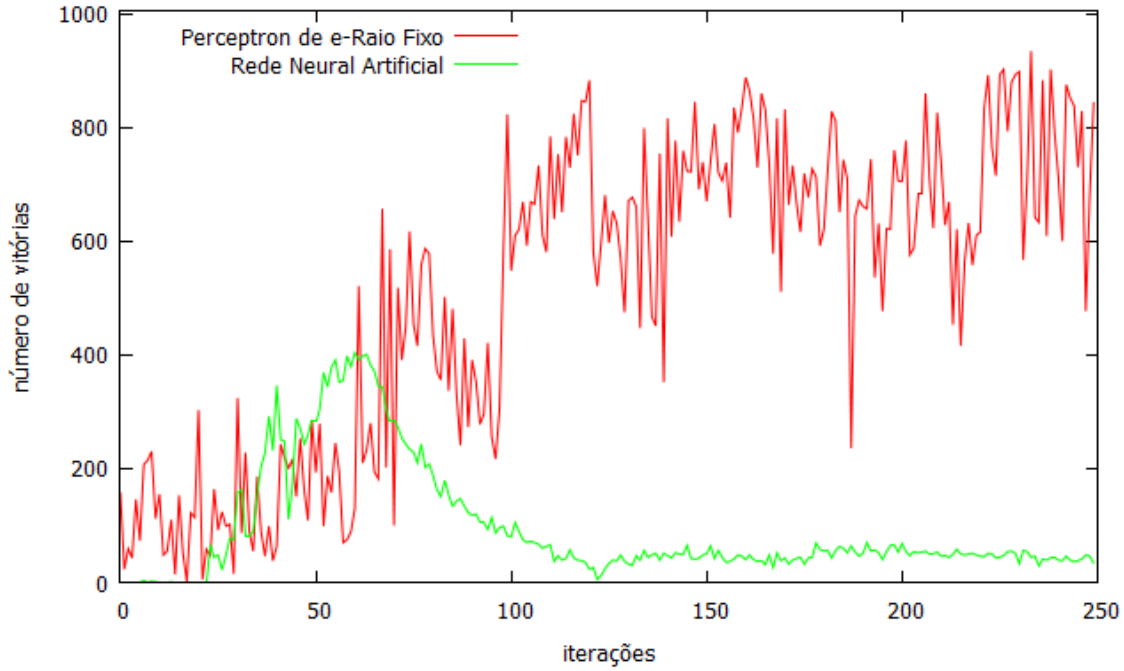



Figura 4.6: Gráfico do número de acertos sobre o tempo para o cenário *Pole Balancing*.

Na Tabela 4.3 podemos observar a média dos tempos de aprendizado. É perceptível como as execuções mais demoradas são as quais o aprendizado não ocorreu corretamente com exceção no cenário *Pole Balancing* no qual ambos algoritmos de regressão demoraram aproximadamente o mesmo tempo.

Tabela 4.3: Tempo médio consumido durante o aprendizado.

| | *Gridworld* | *Mountain Car* | *Pole Balancing* |
|---|---|---|---|
| Rede Neural Artificial | $127h03m25s$ | **1h21m08s** | **5h11m01s** |
| *Perceptron* de $\varepsilon$-Raio Fixo | **1h31m38s** | $99h33m53s$ | $6h19m33s$ |

A Tabela 4.4 contém as médias do número de vitórias testado após o aprendizado. No geral o número de vitórias condiz com os valores da Tabela 4.2.

Tabela 4.4: Número de vitórias médio ao final da última interação.

| | *Gridworld* | *Mountain Car* | *Pole Balancing* |
|---|---|---|---|
| Rede Neural Artificial | $59,74\%$ | $\mathbf{80,57}\%$ | $17,45\%$ |
| *Perceptron* de $\varepsilon$-Raio Fixo | $\mathbf{100}\%$ | $19,83\%$ | $\mathbf{49,57}\%$ |

# 5 Conclusão

Neste trabalho foi proposto um novo algoritmo de aprendizado por reforço baseado no algoritmo *Fitted Q Iteration*. Este algoritmo utiliza o *Perceptron* de $\varepsilon$-Raio Fixo para fazer a regressão da função-valor $Q$. Os testes comparativos realizados contra o algoritmo *Neural Fitted Q Iteration* demonstram que o algoritmo proposto tem um desempenho comparável, com o benefício de ter um número de parâmetros menor para sua configuração. Além disso, o algoritmo proposto parece ser mais robusto no sentido que há uma variação menor do aprendizado a medida que o algoritmo itera.

Como trabalho futuro pretende-se investigar características de convergência deste do método proposto. Outros ramos de investigação que apareceram durante o decorrer do trabalho foram: a realização de uma análise sobre as vantagens, desvantagens e impacto no aprendizado da utilização do reinicio e a possibilidade de realizar uma regressão para cada ação possível de forma que o aprendizado realizado para uma ação não interfira no aprendizado das outras.

# Bibliografia

[1] KOHL, N.; STONE, P. Policy gradient reinforcement learning for fast quadrupedal locomotion. In: *2004 IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04*. [S.l.: s.n.], 2004. v. 3.

[2] CUTLER, M.; HOW, J. P. Efficient reinforcement learning for robots using informative simulated priors. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.]: IEEE, 2015. p. 2605–2612.

[3] ABBEEL, P. et al. An application of reinforcement learning to aerobatic helicopter flight. In: *Proceedings of the 19th International Conference on Neural Information Processing Systems*. Cambridge, MA, USA: MIT Press, 2006. p. 1–8.

[4] SINGH, S. et al. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. *J. Artif. Int. Res.*, AI Access Foundation, USA, v. 16, n. 1, p. 105–133, 2002.

[5] TESAURO, G. Practical issues in temporal difference learning. *Machine Learning*, v. 8, n. 3-4, p. 257–277, 1992.

[6] MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, v. 518, n. 7540, p. 529–533, 2015.

[7] SILVER, D.; HASSABIS, D. *AlphaGo: Mastering the ancient game of Go with Machine Learning*. 2016. Disponível em: <https://research.googleblog.com/2016/01/alphago-mastering-ancient-game-of-go.html>.

[8] RIEDMILLER, M. Neural Fitted Q Iteration – First Experiences with a Data Efficient Neural Reinforcement Learning Method. In: *Machine Learning: ECML 2005*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005. v. 3720, p. 317–328.

[9] SCHOLKOPF, B.; SMOLA, A. J. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001.

[10] SOUZA, R. C. S. N. P. et al. Online algorithm based on support vectors for orthogonal regression. *Pattern Recognition Letters*, Elsevier Science Inc., New York, NY, USA, v. 34, n. 12, p. 1394–1404, 2013.

[11] WATKINS, C. J. C. H. *Learning from delayed rewards*. Tese (Doutorado) — King's College, Cambridge, 1989.

[12] JAAKKOLA, T.; JORDAN, M. I.; SINGH, S. P. On the convergence of stochastic iterative dynamic programming algorithms. *Neural Computation*, v. 6, n. 6, p. 1185–1201, 1994.

[13] ORMONEIT, D.; SEN, S. Kernel-Based Reinforcement Learning. *Machine Learning*, v. 49, n. 2, p. 161–178, 2002.

[14] ERNST, D.; GEURTS, P.; WEHENKEL, L. Tree-based batch mode reinforcement learning. *J. Mach. Learn. Res.*, v. 6, p. 503–556, 2005.

[15] ROSENBLATT, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, p. 65–386, 1958.

[16] DREYFUS, S. The numerical solution of variational problems. *Journal of Mathematical Analysis and Applications*, v. 5, n. 1, p. 30–45, 1962.

[17] RIEDMILLER, M.; BRAUN, H. *RPROP - A Fast Adaptive Learning Algorithm*. [S.l.], 1992.

[18] DRUCKER, H. et al. Support vector regression machines. In: *ADVANCES IN NEURAL INFORMATION PROCESSING SYSTEMS 9*. [S.l.]: MIT Press, 1997. p. 155–161.

[19] BOYAN, J. A.; MOORE, A. W. Generalization in reinforcement learning: Safely approximating the value function. In: *Advances in Neural Information Processing Systems 7*. [S.l.]: MIT Press, 1995. p. 369–376.

[20] SUTTON, R. S.; BARTO, A. G. *Introduction to Reinforcement Learning*. 1st. ed. Cambridge, MA, USA: MIT Press, 1998.